O_2scl - An Object-Oriented Scientific Computing Library

Version 0.910

Contents

1 O2scl User's Guide

1.1 Feature Overview

O_2scl is a C++ class library for object-oriented numerical programming. It includes

- Classes based on numerical routines from GSL and CERNLIB
- Vector and matrix classes which are fully compatible with `gsl_vector` and `gsl_matrix` , yet offer indexing with `operator[]` and other object-oriented features
- CERNLIB-based classes which are completely rewritten in C++
- Classes which require function inputs are designed to accept (public or private) member functions, even if they are virtual.
- Classes use templated vector types, which allow the use of user-specified object-oriented vectors or C-style arrays.
- Highly compatible - Recent versions have been tested on Linux (32- and 64-bit systems, with Intel and AMD chips), Windows XP with Cygwin, and MacOSX.
- Free! O_2scl is provided under Version 3 of the GNU Public License (see License Information for more).
- Two sub-libraries
  - Thermodynamics of ideal and nearly-ideal particles with quantum statistics
  - Equations of state for finite density relevant for neutron stars

This is a beta version. The library should install and test successfully, and most of the classes are ready for production use. Some of the interfaces may change slightly in future versions. There are a few classes which are more experimental, and this is clearly stated at the top of the documentation for these classes.

1.2 Quick Reference to User's Guide

- Installation
- General Usage
- Compiling Examples
- Related Projects
- Complex Numbers

2 Installation

The rules for installation are generally the same as that for other GNU libraries. The file `INSTALL` has some details on this procedure. Generally, you should be able to run `./configure` and then type `make` and `make install`. More information on the `configure` command can also be obtained from `./configure --help`. O_2scl requires the GSL library. If the `configure` script cannot find them, you may have to specify their location in the `CPPFLAGS` and `LDFLAGS` environment variables (`./configure --help` shows some information on this). The documentation is included in the distribution and automatically installed by `make install`.

After `make install`, you may test the library with `make o2scl-test`. At the end, the phrase `All O2scl tests passed` indicates that the testing was successful. You may also run `make o2scl-test` in the individual subdirectories of the src directory to individually test the classes and functions in that part of O_2scl . Note that `make check` will not work.

This library requires GSL and is designed to work with GSL versions 1.15 or greater, which can be obtained from http://www.gnu.org/software/gsl . Some classes may work with older versions of GSL, but this cannot be guaranteed. A CBLAS library is also required, and `./configure` will look for `libcblas` first, and if not found then it will look for `libgslcblas`. If neither is present, then you may have to manually specify a CBLAS library using the `LIBS` and `LDFLAGS` environment variables.

This library also performs file I/O using the HDF5 (not HDF4) library, which can be obtained from http://www.hdfgroup.org . To compile O_2scl without HDF support, use the argument `--disable-hdf` to `configure` during installation. If HDF5 is not located in a typical location for your system, in order to compile O_2scl with HDF support you may have to specify the location of the HDF header files and libraries with the `-I` and `-L` flags. Some O_2scl functions use the high-level HDF routines found in the `hdf5_hl` library, but this library is usually in the same location as the base hdf5 library.

Range-checking for vectors and matrices is turned on by default. You can disable range-checking by defining -DO2SCL_NO_RANGE_CHECK, e.g.

```
CPPFLAGS="-DO2SCL_NO_RANGE_CHECK" ./configure
```

The separate libraries O_2scl_eos and O_2scl_part are installed by default. To disable the installation of these libraries and their associated documentation, run `./configure` with the flags `--disable-eoslib` or `--disable-partlib`. Note that O_2scl_eos depends on O_2scl_part so using `--disable-partlib` without `--disable-eoslib` will not work. Note also that both O_2scl_part and O_2scl_eos require HDF support.

There are several warning flags that are useful when configuring and compiling with O_2scl . See the GSL documentation for an excellent discussion, and also see the generic installation documentation in the file `INSTALL` in the O_2scl top-level directory. For running `configure`, for example, if you do not have privileges to write to `/usr/local`,

```
CPPFLAGS="-O3 -I/home/asteiner/install/include" \
LDFLAGS="-L/home/asteiner/install/lib" ./configure -C \
--prefix=/home/asteiner/install
```

In this example, specifying `-I/home/asteiner/install/include` and `-L/home/asteiner/install/lib` above ensures that the GSL libraries can be found (this is where they are installed on my machine). The `--prefix=/home/asteiner/install` argument to `./configure` ensures that O_2scl is installed there as well.

For those users wanting to compile O_2scl under Xcode on a Mac, some advice for doing this has been posted to the `o2scl-help` mailing list.

The O_2scl documentation is generated with Doxygen . In principle, the documentation can be regenerated by the end-user, but this is not supported and requires several external applications not included in the distribution.

Un-installation: While there is no explicit "uninstall" makefile target, there are only a couple places to check. - Installation creates directories named `o2scl` in the include, doc and shared files directory (which default to `/usr/local/include`, `/usr/local/doc`, and `/usr/local/share`) which can be removed. Finally, all of the libraries are named with the prefix `libo2scl` and are created by default in /usr/local/lib. As configured with the settings above, the files are in `/home/asteiner/install/include/o2scl`, `/home/asteiner/install/lib`, `/home/asteiner/install/share/o2scl`, and `/home/asteiner/install/doc/o2scl`.

3 General Usage

3.1 Namespaces

Most of the classes reside in the namespace o2scl (this namespace has been removed from the documentation for clarity). - Numerical constants (many of them based on the GSL constants) are placed in separate namespaces (gsl_cgs, gsl_cgsm, gsl_mks, gsl_mksa, gsl_num, o2scl_const, and o2scl_fm). There are also two namespaces which hold integration coefficients, o2scl_inte_gk_coeffs and o2scl_inte_qng_coeffs. There are also some namespaces for the linear algebra functions, see Linear Algebra for more information.

3.2 Documentation conventions

In the following documentation, function parameters are denoted by parameter, except when used in mathematical formulas as in variable .

3.3 Nomenclature

Classes directly derived from the GNU Scientific Library are preceeded by the prefix gsl_ and classes derived from CERNLIB are preceeded by the prefix cern_. Some of those classes derived from GSL and CERNLIB operate slightly differently from the original versions. The differences are detailed in the corresponding class documentation.

3.4 Basic error handling

Error handling is a hybrid approach combining GSL with C++ exceptions. An abstract type has been defined which operates as a GSL-like error hander. The default error handler is a implementation of this abstract type which throws a C++ exception when an error is encountered. The various exceptions, and their correspondence with the GSL error codes, are given in the GSL error codes and C++ exception types section. By default in O_2scl , the default GSL error handler is replaced with the O_2scl default error handler, i.e. GSL functions will throw C++ exceptions.

Errors can be set by the user through the macros O2SCL_ERR, which sets an error, and O2SCL_ERR_RET, which sets an error and returns the error number.

The error handler, err_hnd is a global pointer to an object of type err_hnd_type. There is a global default error handler, def_err_hnd, of type err_hnd_cpp, which throws C++ exceptions, and an alternate default error handler, alt_err_hnd, of type err_hnd_gsl, which outputs an error message and aborts execution.

Generally, functions also follow the GSL-like behavior of calling a GSL-like error handler and (when appropriate) return a non-zero value. When functions succeed they return gsl_success (zero). O_2scl functions never reset the error handler. Also, object destructors do not generally call the error handler. Internally, O_2scl does not use try blocks, but these can easily be effectively employed by an O_2scl user.

The list of GSL error codes (including a few extra ones for O_2scl) is given in the documentation for the file err_hnd.h. The default error handler can be replaced by simply assigning the address of a descendant of err_hnd_type to err_hnd.

Functionality similar to assert() is provided with the macro O2SCL_ASSERT, which exits if its argument is non-zero, and O2SCL_BOOL_ASSERT which exits if its argument is false.

One can instruct the library to use the GSL-like O_2scl error handler alt_err_hnd by default, by defining the constant O2SCL_USE_GSL_HANDLER. This is also useful if one wants to compile without C++ exceptions (which does have a small overhead).

3.5 What is an error?

O_2scl assumes that errors are events which should happen infrequently. Error handling strategies are often time-consuming and they are not a replacement for normal code flow. However, even with this in mind, one can still distinguish a large spectrum of posibillities from "fatal" errors, those likely to corrupt the stack and/or cause a dreaded "segmentation fault" and "non-fatal" errors, those errors

which might cause incorrect results, but might be somehow recoverable. One of the purposes of error handling is to decide if and how these different types of errors should be handled differently.

Sometimes, it is undesirable to abort execution upon a failure to reach numerical convergence. While these failures are treated as errors (and by default an exception is thrown), some of the classes which attempt to reach numerical convergence have an option (e.g. mroot::err_nonconv) to turn this default behavior off for these convergence errors. To set these "convergence" errors in code provided by the user, the macros O2SCL_CONV and O2SCL_CONV_RET can be used.

Of course, the standard `try`, `catch` mechanism of error handling may also be used for finer-grained control.

Another related issue is that O_2scl often calls functions which are supplied by the user, these user-designed functions may create errors, and the library needs to decide how to deal with them, even though it knows little about what is actually happening inside these user-defined functions. Most of the time, O_2scl assumes that if a function returns a nonzero value, then an error has occured and the present calculation should abort.

3.6 GSL error codes and C++ exception types

See also the description of the error codes in err_hnd.h .

gsl_success=0, (no error thrown)

gsl_failure=-1, exc_exception

gsl_continue=-2, (no error thrown)

gsl_edom=1, exc_range_error

gsl_erange=2, exc_range_error

gsl_efault=3, exc_runtime_error

gsl_einval=4, exc_invalid_argument

gsl_efailed=5, exc_exception

gsl_efactor=6, exc_runtime_error

gsl_esanity=7, exc_exception

gsl_enomem=8, exc_runtime_error

gsl_ebadfunc=9, exc_runtime_error

gsl_erunaway=10, exc_runtime_error

gsl_emaxiter=11, exc_runtime_error

gsl_ezerodiv=12, exc_overflow_error

gsl_ebadtol=13, exc_invalid_argument

gsl_etol=14, exc_runtime_error

gsl_eundrflw=15, exc_range_error

gsl_eovrflw=16, exc_overflow_error

gsl_eloss=17, exc_runtime_error

gsl_eround=18, exc_runtime_error

gsl_ebadlen=19, exc_invalid_argument

gsl_enotsqr=20, exc_invalid_argument

gsl_esing=21, exc_runtime_error

gsl_ediverge=22, exc_runtime_error

gsl_eunsup=23, exc_exception

gsl_eunimpl=24, exc_exception

gsl_ecache=25, exc_runtime_error

gsl_etable=26, exc_runtime_error

gsl_enoprog=27, exc_runtime_error

gsl_enoprogj=28, exc_runtime_error

gsl_etolf=29, exc_runtime_error

gsl_etolx=30, exc_runtime_error

gsl_etolg=31, exc_runtime_error

gsl_eof=32, exc_ios_failure

gsl_enotfound=33, exc_runtime_error

gsl_ememtype=34, exc_logic_error

gsl_efilenotfound=35, exc_ios_failure

gsl_eindex=36, exc_invalid_argument

gsl_outsidecons=37, exc_runtime_error

3.7 Objects and scope

O_2scl objects frequently take inputs which are of the form of a reference to a smaller object. This is particularly convenient because it allows a lot of flexibility, while providing a certain degree of safety. In many cases, the user retains the responsibility of ensuring that input objects do not go out of scope before they are utilized by objects which require them. This is actually no different than the requirements on the user imposed by GSL, for example.

A simple example of this is provided by the multi-dimensional integrator composite_inte. It works by combining several one-dimensional integrators to integrate over a multi-dimensional hybercube. If the specified one-dimensional integrator goes out of scope an integration is attempted, the integration will fail, for example,

```
// How not to provide subobjects to O2scl classes

void set_integrator(multi_inte<int> &mi) {
  gsl_inte_qag<size_t> one_dim_it[2];
  mi.set_oned_inte(one_dim_it[0],0);
  mi.set_oned_inte(one_dim_it[1],1);
}

void function() {
  composite_inte<int> cit;
  set_integrator(cit);
  cit.minteg_err(func,2,a,b,pa,res,err);
}
```

This will fail because the one-dimensional integration objects go out of scope (it's a local variable in the function and its destructor is called before the `set_integrator()` function exits) before the `minteg_err()` function is called.

In C++, this difficulty can be solved by using shared pointers, which ensure that an object is deleted only when it has no more references which are pointing to it. While shared pointer objects are part of the new TR1 standard, not all compilers implement this standard yet. O_2scl defines a template struct o2_shared_ptr, which provides the user a consistent interface to a working shared pointer type, `o2_shared_ptr<T>::type` (located in the `o2scl` namespace). If TR1 is not available, then one can compile O_2scl with O2SCL_NO_TR1_MEMORY. In this case, if O2SCL_HAVE_BOOST is defined, then the boost shared pointer type is used, and otherwise an internal O_2scl shared pointer template named o2_int_shared_ptr is used . See the documentation for o2_shared_ptr for more details.

3.8 Reference parameters

When a O_2scl function contains two reference parameters for objects, it is not typically possible to provide the same object to both parameters or to provide two objects which share the same memory. This is particularly an issue when the associated types are template types, since then the O_2scl library has no way of knowing how memory is organized in these unspecified types.

4 Compiling Examples

The example programs are in the `examples` directory. After installation, they can be compiled and executed by running `make o2scl-examples` in that directory. This will also test the output of the examples to make sure it is correct. If all the examples succeed, the message `All O2scl tests passed.` will appear at the end. The output for each example is placed in the corresponding file with a `.scr` extension.

Alternatively, you can make the executable for each example in the `examples` directory individually using, e.g. `make ex_mroot`.

Also, the testing code for each class is occasionally useful for providing examples of their usage. The testing source code for each source file is named with an `_ts.cpp` prefix in the same directory as the class source.

Note

All of these examples are provided under the GPLv3 license, although to save space the licenses are not reproduced in the documentation for each example. See License Information for more information.

5 Related Projects

Several noteworthy related projects:

- GSL - The GNU Scientific Library

  The first truly free, ANSI-compliant, fully tested, and well-documented scientific computing library. The GSL is located at `http://www.gnu.org/software/gsl` , and the manual is available at

  `http://www.gnu.org/software/gsl/manual/html_node/` . Many GSL routines are included and reworked in O_2scl (the corresponding classes begin with the `gsl_` prefix) and O_2scl was specifically designed to be used with GSL. GSL is a must-have for most serious numerical work in C or C++ and is required for installation of O_2scl .

- HDF - Hierarchical Data Format, (`http://www.hdfgroup.org`)

  A library developed for file I/O of scientific data. Contains C, C++, FORTRAN and Java APIs, several command-line tools, and a longstanding active community of users.

- CERNLIB - (`http://cernlib.web.cern.ch/cernlib/mathlib.html`)

  The gold standard in FORTRAN computing libraries Several CERNLIB routines are rewritten completely in C++ and included in O_2scl (they begin with the `cern_` prefix). CERNLIB is located at

- LAPACK and ATLAS - The gold standard for linear algebra

  Available at `http://www.netlib.org/lapack` and `http://www.netlib.org/atlas` . See also `http://www.-netlib.org/clapack` .

- QUADPACK - FORTRAN adaptive integration library, `http://www.netlib.org/quadpack`

  This is the library on which the GSL integration routines are based (prefix `gsl_inte_`).

- Blitz++ - (`http://www.oonumerics.org/blitz`)

  Another linear algebra library designed in C++ which includes vector and matrix types and basic arithmetic. As the name implies, Blitz++ has the capability to be particularly fast. Distributed with a Perl-like artistic license "or the GPL".

- Eigen - (http://eigen.tuxfamily.org)

  A linear algebra library in C++. Eigen is licensed under GPLv2 or LGPLv3.

- Boost - (http://www.boost.org)

  Free (license is compatible with GPL) peer-reviewed portable C++ source libraries that work well with the C++ Standard Library. Boost also contains uBlas, for linear algebra computing.

- FLENS - (http://flens.sourceforge.net)

  A C++ library with object-oriented linear algebra types and an interface to BLAS and LAPACK.

- MESA - Modules for Experiments in Stellar Astrophysics, (http://mesa.sourceforge.net)

  An excellent FORTRAN library with accurate low-density equations of state, interpolation, opacities and other routines useful in stellar physics. Work is currently under way to rewrite some of the MESA routines in C/C++ for O_2scl . Licensed with LGPL (not GPL).

- TNT - Template numerical toolkit (http://math.nist.gov)

  TNT provides vector and matrix types and basic arithmetic operations. Most of the classes in O_2scl which use vector and matrix types are templated to allow compatbility with TNT. (Though there are a few small differences.) This software is in the public domain. TNT appears not to be under current development.

- OOL - Open Optimization Library (http://ool.sourceforge.net)

  Constrained minimization library designed with GSL in mind. The O_2scl constrained minimization classes are derived from this library, though OOL appears not to be under current development. Constrained Minimization .

- Root - CERN's new C++ analysis package (http://root.cern.ch)

  A gargantuan library for data analysis, focused mostly on high-energy physics. Their histograms, graphics, file I/O and support for large data sets is particuarly good. See graph.h for some functions which provide some plotting functions for use with Root.

- Aegis - a transaction-based software configuration management system (http://aegis.sourceforge.net)

  Related because the o2_int_shared_ptr class was adapted from Aegis.

6 Complex Numbers

Several arithmetic operations for `gsl_complex` are defined in cx_arith.h, but no constructor has been written. The object `gsl_complex` is still a `struct`, not a `class`. For example,

```
#include <o2scl/cx_arith.h>
gsl_complex a={{1,2}}, b={{3,4}};
gsl_complex c=a+b;
cout << GSL_REAL(c) << " " << GSL_IMAG(C) << endl;
```

In case the user needs to convert between `gsl_complex` and `std::complex<double>`, two conversion functions gsl_to_complex() and complex_to_gsl() are also provided in cx_arith.h.

A short example using complex number arithmetic is given below.

6.1 Mandelbrot set

```
/* Example: ex_mandel.cpp
   -------------------------------------------------------------------
   Mandelbrot example demonstrating table3d and complex arithmetic
```

```
*/

#include <iostream>
#include <o2scl/cx_arith.h>
#include <o2scl/test_mgr.h>
#include <o2scl/table3d.h>
#ifdef O2SCL_HDF_IN_EXAMPLES
#include <o2scl/hdf_file.h>
#include <o2scl/hdf_io.h>
#endif

using namespace std;
using namespace o2scl;
#ifdef O2SCL_HDF_IN_EXAMPLES
using namespace o2scl_hdf;
#endif

int main(void) {

  test_mgr tm;
  tm.set_output_level(2);

  // Create a table3d object
  table3d t;

  // Add parameters
  double delta=0.001, minx=-1.5, maxx=0.8, miny=-1.0, maxy=1.0;
  size_t maxtime=0, limit=100;
  t.add_constant("delta",delta);
  t.add_constant("minx",minx);
  t.add_constant("maxx",maxx);
  t.add_constant("miny",miny);
  t.add_constant("maxy",maxy);

  // Set grid
  ovector ox, oy;
  for(double x=minx;x<=maxx;x+=delta) ox.push_back(x);
  for(double y=miny;y<=maxy;y+=delta) oy.push_back(y);
  t.set_xy("x",ox.size(),ox,"y",oy.size(),oy);

  // Create slice
  t.new_slice("time");

  // Compute escape times
  for(size_t i=0;i<ox.size();i++) {
    for(size_t j=0;j<oy.size();j++) {
      gsl_complex c={{ox[i],oy[j]}};
      gsl_complex z={{0,0}};
      size_t time=0;
      for(size_t k=0;k<limit;k++) {
        // Arithmetic with gsl_complex objects
        z=z*z+c;
        if (abs(z)>10.0) {
          time=k;
          k=limit;
        }
      }
      t.set(i,j,"time",time);
      if (time>maxtime) maxtime=time;
    }
  }

  // Maximum escape time for color normalization
  t.add_constant("maxtime",maxtime);

  // Output to file if O2scl is compiled with HDF support
#ifdef O2SCL_HDF_IN_EXAMPLES
  hdf_file hf;
  hf.open("ex_mandel.o2");
  hdf_output(hf,t,"mandel");
```

```
  hf.close();
#endif

  tm.test_gen(maxtime==99,"maxtime test");
  tm.report();

  return 0;
}
// End of example
```

The information stored in the table3d object in ex_mandel.out can be plotted:

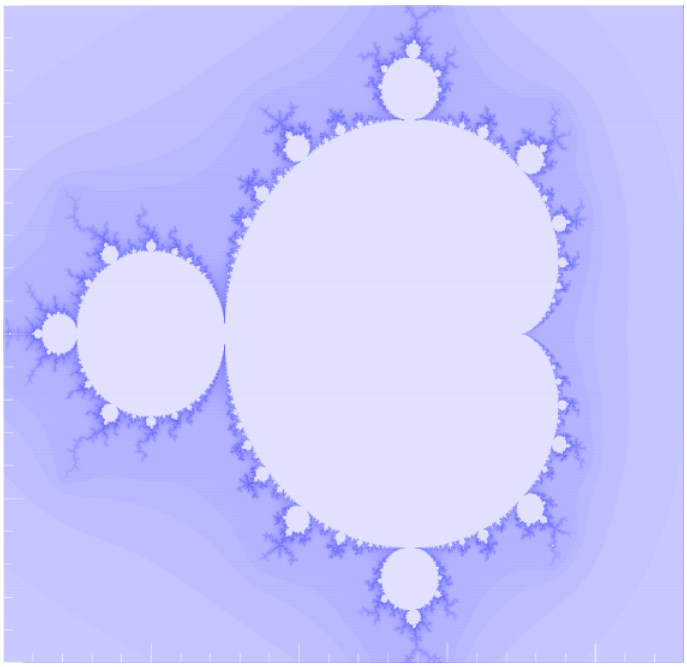


Figure 1: Mandelbrot example plot

7 Arrays, Vectors, Matrices and Tensors

7.1 Introduction

The O_2scl library uses a standard nomenclature to distinguish a couple different concepts. The word "array" is always used to refer to C-style arrays, i.e. `double[]`. If there are two dimensions in the array, it is a "two-dimensional array", i.e. `double[][]` . The word "vector" is reserved generic objects with array-like semantics, i.e. any type of object or class which can be treated similar to an array in that it has an function `operator[]` which gives array-like indexing. Thus arrays are vectors, but not all vectors are arrays. There are a couple specific vector types defined in O_2scl like ovector and uvector . The STL vector `std::vector<double>` is also a vector type in this language. The word "matrix" is reserved for the a generic object which has matrix-like semantics and can be accessed using either `operator()` or two successive applications of `operator[]` (or sometimes both). The O_2scl objects omatrix and umatrix are matrix objects, as is a C-style two-dimensional array, `double[][]` . The header files are named in this spirit also: functions and classes which operate on generic vector types are in vector.h and functions and classes which work only with C-style arrays are in array.h . The word "tensor" is used for a generic object which has rank `n` and then has `n` associated indices. A vector is just a tensor of rank 1 and a matrix is just a tensor of rank 2.

Most of the classes in O_2scl which use vectors and/or matrices are designed so that they can be used with any appropriately-defined vector or matrix types. This is a major part of the design goals for O_2scl and most of the classes are compatible with matrix and vector objects from GSL, TNT, MV++, uBlas, and Blitz++.

The first index of a matrix type is defined always to be the index associated with "rows" and the second is associated with "columns". The O_2scl matrix output functions respect this notation as well, so that all of the elements of the first row is sent to the screen, then all of the elements in the second row, and so on. With this in mind, one can make the distinction between "row-major" and "column-major" matrix storage. C-style two-dimensional arrays are "row-major" in that the elements of the first row occupy the first locations in memory as opposed "column-major" where the first column occupies the first locations in memory. It is important to note that the majority of the classes in O_2scl do not care about the details of the underlying memory structure, so long as two successive applications of `operator[]` (or in some cases `operator(,)`) works properly. The storage format used by omatrix and umatrix is row-major, and there are no column-major matrix classes in O_2scl (yet).

7.2 Matrix indexing with [][] vs. (,)

While vector indexing with `operator[]` is very compatible with almost any vector type, matrix indexing is a bit more difficult. There are two options: assume matrix objects provide an `operator[]` method which can be applied twice, i.e. `m[i][j]`, or assume that matrix elements should be referred to with `m(i,j)`. Most of the O_2scl classes use the former approach so that they are also compatible with two-dimensional arrays. However, there are sometimes good reasons to want to use `operator()` for matrix-intensive operations from linear algebra. For this reason, some of the functions given in the `linalg` directory are specified in two forms: the first default form which assumes `[][]`, and the second form with the same name, but in a namespace which has a suffix `_paren` and assumes matrix types use `(,)`.

7.3 Vector types for template classes

Many of the classes and functions are designed to be generic with regard to vector types, permitting the user to use any type which has a suitably defined `operator[]` method (or sometimes `operator()` method). In order to do this, some classes also require the ability to allocate new vectors, and for this reason they have three different templated vector types:

- `vec_t` - the generic vector type
- `alloc_t` - a vector allocation type
- `alloc_vec_t` - a vector type which can be allocated with an object of type `alloc_t` and can be passed as an argument of type `vec_t`.

The typical requirement is that one must be able to write something like

```
alloc_t ao;
alloc_vec_t vec;
ao.allocate(vec,5);
```

to allocate a new vector `vec` of length 5. In addition, to provide exception safety, either the objects of type `alloc_vec_t` must either free its memory in its destructor, or it must be taken care of in the destructor of the `alloc_t` object. The native O_2scl vector classes provide this kind of functionality by default, so some typical choices for these template types are

- `vec_t` = ovector_base, `alloc_t` = ovector_alloc, and `alloc_vec_t` = ovector
- `vec_t` = uvector_base, `alloc_t` = uvector_alloc, and `alloc_vec_t` = uvector
- `vec_t` = `double[n]`, `alloc_t` = `array_alloc<double[n]>` , and `alloc_vec_t` = `double[n]`
- `vec_t` = `double *`, `alloc_t` = `pointer_alloc<double>` , and `alloc_vec_t` = `double *`

An example demonstrating how this works is given in the Multi-dimensional solver example.

7.4 Rows and columns vs. x and y

Sometimes its useful to think about the rows and columns in a matrix as referring to a elements of a grid, and the matrix indices refer to points in a grid in (x,y). It might seem intuitive to think of a matrix as `A`[ix][iy] where `ix` and `iy` are the x and y indices because the ordering of the indices is alphabetical. However, it is useful to note that because functions like matrix_out print the first "row" first rather than the first column, a matrix constructed as `A`[ix][iy] will be printed out with x on the "vertical axis" and y on the "horizontal axis", so it is sometimes useful to store data in the form `A`[iy][ix] (for example, in the two dimensional interpolation class, twod_intp). In any case, all classes which take matrix data as input will document how the matrix ought to be arranged.

7.5 Generic vector functions

There are a couple functions which operate on generic vectors of any type in vector.h . They perform sorting, summing, rotating, copying, and computations of minima and maxima. For more statistically-oriented operations, see also vec_stats.h .

7.6 Native matrix and vector types

The rest of this section in the User's guide is dedicated to describing the O_2scl implementations of vector, matrix, and tensor types.

Vectors and matrices are designed using the templates ovector_tlate and omatrix_tlate, which are compatible with `gsl_vector` and `gsl_matrix`. Vectors and matrices with unit stride are provided in uvector_tlate and umatrix_tlate. The most commonly used double-precision versions of these template classes are ovector, omatrix, uvector and umatrix, and their associated "views" (analogous to GSL vector and matrix views) which are named with a `_view` suffix. The classes ovector_tlate and omatrix_tlate offer the syntactic simplicity of array-like indexing, which is easy to apply to vectors and matrices created with GSL.

The following sections primarily discuss the operation objects of type ovector. The generalizations to objects of type uvector, omatrix, and the complex vector and matrix objects ovector_cx, omatrix_cx, uvector_cx, and umatrix_cx are straightforward.

7.7 Vector and matrix views

Vector and matrix views are provided as parents of the vector and matrix classes which do not have methods for memory allocation. As in GSL, it is simple to "view" normal C-style arrays or pointer arrays (see ovector_array), parts of vectors (ovector_subvector), and rows and columns of matrices (omatrix_row and omatrix_col). Several operations are defined, including addition, subtraction, and dot products.

7.8 Vector and matrix typedefs

Several typedefs are used to give smaller names to often used templates. Vectors and matrices of double-precision numbers all begin with the prefixes ovector and omatrix. Integer versions begin with the prefixes ovector_int and omatrix_int. Complex versions have an additional `"_cx"`, e.g. ovector_cx. See ovector_tlate.h, ovector_cx_tlate.h, omatrix_tlate.h, and omatrix_cx_tlate.h.

7.9 Unit-stride vectors

The uvector_tlate objects are naturally somewhat faster albeit less flexible than their finite-stride counterparts. Conversion to GSL vectors and matrices is not trivial for uvector_tlate objects, but demands copying the entire vector. Vector views, operators, and the nomenclature is similar to that of ovector.

7.10 Memory allocation

Memory for vectors can be allocated using ovector::allocate() and ovector::free(). Allocation can also be performed by the constructor, and the destructor automatically calls `free()` if necessary. In contrast to `gsl_vector_alloc()`, ovector::allocate() will

call ovector::free(), if necessary, to free previously allocated space. Allocating memory does not clear the recently allocated memory to zero. You can use ovector::set_all() with a zero argument to clear a vector (and similarly for a matrix).

If the memory allocation fails, either in the constructor or in allocate(), then the error handler will be called, partially allocated memory will be freed, and the size will be reset to zero.

Although memory allocation in O_2scl vectors is very similar to that in GSL, the user must not mix allocation and deallocation between GSL and O_2scl .

7.11 Get and set

Vectors and matrices can be modified using ovector::get() and ovector::set() methods analogous to `gsl_vector_get()` and `gsl_vector_set()`, or they can be modified through ovector::operator[ ] (or ovector::operator()), e.g.

```
ovector a(4);
a.set(0,2.0);
a.set(1,3.0);
a[2]=4.0;
a[3]=2.0*a[1];
```

If you want to set all of the values in an ovector or an omatrix at the same time, then use ovector::set_all() or omatrix::set_all().

7.12 Range checking

Range checking is performed depending on whether or not `O2SCL_NO_RANGE_CHECK` is defined. It can be defined in the arguments to ./configure upon installation to turn off range checking. Note that this is completely separate from the GSL range checking mechanism, so range checking may be on in O_2scl even if it has been turned off in GSL. Range checking is used primarily in the vector, matrix, and tensor `get()` and `set()` methods.

To see if range checking was turned on during installation (without calling the error handler), use lib_settings_class::range_check().

Note that range checking in O_2scl code is most often present in header files, rather than in source code. This means that range checking can be turned on or off in user-defined functions separately from whether or not it was used in the library classes and functions.

7.13 Shallow and deep copy

Copying O_2scl vectors using constructors or the = operator is performed according to what kind of object is on the left-hand side (LHS) of the equals sign. If the LHS is a view, then a shallow copy is performed, and if the LHS is a ovector, then a deep copy is performed. If an attempt is made to perform a deep copy onto a vector that has already been allocated, then that previously allocated memory is automatically freed. The user must be careful to ensure that information is not lost this way, even though no memory leak will occur.

For generic deep vector and matrix copying, you can use the template functions vector_copy(), matrix_copy(). These would allow you, for example, to copy an ovector to a `std::vector<double>` object. These functions do not do any memory allocation so that must be handled beforehand by the user.

7.14 Vector and matrix arithmetic

Several operators are available as member functions of the corresponding template:

Vector_view unary operators:

- vector_view += vector_view
- vector_view -= vector_view

- vector_view += scalar
- vector_view -= scalar
- vector_view *= scalar
- scalar = norm(vector_view)

Matrix_view unary operators:

- matrix_view += matrix_view
- matrix_view -= matrix_view
- matrix_view += scalar
- matrix_view -= scalar
- matrix_view *= scalar

Binary operators like addition, subtraction, and matrix multiplication are also defined for ovector, uvector, and related objects. The generic template for a binary operator, e.g.

```
template<class vec_t> vec_t &operator+(vec_t &v1, vec_t &v2);
```

is difficult because the compiler has no way of distinguishing vector and non-vector classes. At the moment, this is solved by creating a define macro for the binary operators. In addition to the predefined operators for native classes, the user may also define binary operators for other classes using the same macros. For example,

```
O2SCL_OP_VEC_VEC_ADD(o2scl::ovector,std::vector<double>,
std::vector<double>)
```

would provide an addition operator for ovector and vectors from the Standard Template Library. A full list of the operators which are defined by default and the associated macros are detailed in the documentation for vec_arith.h.

The GSL BLAS routines can also be used directly with ovector and omatrix objects.

Note that some of these arithmetic operations succeed even with non-matching vector and matrix sizes. For example, adding a 3x3 matrix to a 4x4 matrix will result in a 3x3 matrix and the 7 outer elements of the 4x4 matrix are ignored.

7.15 Converting to and from GSL forms

Because of the way ovector is constructed, you may use type conversion to convert to and from objects of type `gsl_vector`.

```
ovector a(2);
a[0]=1.0;
a[1]=2.0;
gsl_vector *g=(gsl_vector *)(&a);
cout << gsl_vector_get(g,0) << " " << gsl_vector_get(g,1) << endl;
```

Or,

```
gsl_vector *g=gsl_vector_alloc(2);
gsl_vector_set(0,1.0);
gsl_vector_set(1,2.0);
ovector &a=(ovector &)(*g);
cout << a[0] << " " << a[1] << endl;
```

This sort of type-casting is discouraged among unrelated classes, but is permissible here because ovector_tlate is a descendant of gsl_vector. In particular, this will not generate "type-punning" warnings in later gcc versions. If this bothers your sensibilities, however, then you can use the following approach:

```
ovector a(2);
gsl_vector *g=a.get_gsl_vector();
```

The ease of converting between these two kind of objects makes it easy to use gsl functions on objects of type ovector, i.e.

```
ovector a(2);
a[0]=2.0;
a[1]=1.0;
gsl_vector_sort((gsl_vector *)(&a));
cout << a[0] << " " << a[1] << endl;
```

7.16 Converting from STL form

To "view" a `std::vector<double>`, you can use ovector_array

```
std::vector<double> d;
d.push_back(1.0);
d.push_back(3.0);
ovector_array aa(d.size,&(d[0]));
cout << aa[0] << " " << aa[1] << endl;
```

However, you should note that if the memory for the `std::vector` is reallocated (for example because of a call to `push_back()`), then a previously created ovector_view will be incorrect.

7.17 push_back() and pop() methods

These two functions give a behavior similar to the corresponding methods for `std::vector<>`. This will work in O_2scl classes, but may not be compatible with all of the GSL functions. This will break if the address of a ovector_tlate is given to a GSL function which accesses the `block->size` parameter instead of the `size` parameter of a `gsl_vector`. Please contact the author of O_2scl if you find a GSL function with this behavior.

7.18 Vector and matrix views

Views are slightly different than in GSL in that they are now implemented as parent classes. Effectively, this means that vector views are just the same as normal vectors, except that they have no memory allocation functions (because the memory is presumably owned by a different object or scope). The code

```
double x[2]={1.0,2.0};
gsl_vector_view_array v(2,x);
gsl_vector *g=&(v.vector);
gsl_vector_set(g,0,3.0);
cout << gsl_vector_get(g,0) << " " << gsl_vector_get(g,1) << endl;
```

can be replaced by

```
double x[2]={1.0,2.0};
ovector_array a(2,x);
a[0]=3.0;
cout << a << endl;
```

7.19 Passing ovector parameters

It is often best to pass an ovector as a const reference to an ovector_base, i.e.

```
void function(const ovector_base &a);
```

If the function may change the values in the ovector, then just leave out `const`

```
    void function(ovector_base &a);
```

This way, you ensure that the function is not allowed to modify the memory for the vector argument.

If you intend for a function (rather than the user) to handle the memory allocation, then some care is necessary. The following code

```
class my_class {
  int afunction(ovector &a) {
    a.allocate(1);
    // do something with a
    return 0;
  }
};
```

is confusing because the user may have already allocated memory for `a`. To avoid this, you may want to ensure that the user sends an empty vector. For example,

```
class my_class {
  int afunction(ovector &a) {
    if (a.get_size()>0 && a.is_owner()==true) {
      O2SCL_ERR("Unallocated vector not sent to afunction().",1);
      return 1;
    } else {
      a.allocate(1);
      // do something with a
      return 0;
    }
  }
};
```

In lieu of this, it is often preferable to use a local variable for the storage and offer a `get()` function,

```
class my_class {
protected:
  ovector a;
public:
  int afunction() {
    a.allocate(1);
    // do something with a
    return 0;
  }
  int get_result(const ovector_view &av) { av=a; return 0; }
};
```

The O_2scl classes run into this situation quite frequently, but the vector type is specified through a template

```
template<class vec_t> class my_class {
protected:
  vec_t a;
public:
  int afunction(vec_t &a) {
    // do something with a
    return 0;
  }
};
```

where the drawback is that the user must perform the allocation.

7.20 Vectors and operator=()

An "operator=(value)" method for setting all vector elements to the same value is not included because it leads to confusion between, `ovector_tlate::operator=(const data_t &val)` and ovector_tlate::ovector_tlate(size_t val) For example, after implementing `operator=()` and executing the following

```
ovector_int o1=2;
ovector_int o2;
o2=2;
```

`o1` will be a vector of size two, and `o2` will be an empty vector!

To set all of the vector elements to the same value, use ovector_tlate::set_all(). Because of the existence of constructors like ovector_tlate::ovector_tlate(size_t val), the following code

```
ovector_int o1=2;
```

still compiles, and is equivalent to

```
ovector_int o1(2);
```

while the code

```
ovector_int o1;
o1=2;
```

will not compile. As a matter of style, `ovector_int o1(2);` is preferable to `ovector_int o1=2;` to avoid confusion.

7.21 Matrix structure

The matrices from omatrix_tlate are structured in exactly the same way as in GSL. For a matrix with 2 rows, 4 colums, and a "tda" or "trailing dimension" of 7, the memory for the matrix is structured in the following way:

```
00 01 02 03 XX XX XX
10 11 12 13 XX XX XX
```

where `XX` indicates portions of memory that are unreferenced. The tda can be accessed through, for example, the method omatrix_view_tlate::tda(). The `get(size_t,size_t)` methods always take the row index as the first argument and the column index as the second argument. The matrices from umatrix_tlate have a trailing dimension which is always equal to the number of columns.

7.22 Reversing the order of vectors

You can get a reversed vector view from ovector_reverse_tlate, or uvector_reverse_tlate. For these classes, `operator[]` and related methods are redefined to perform the reversal. If you want to make many calls to these indexing methods for a reversed vector, then simply copying the vector to a reversed version may be faster.

7.23 Const-correctness with vectors

Vector objects, like ovector, can be const in the usual way. However because vector views, like ovector_view are like pointers to vectors, there are two ways in which they can be const. The view can point to a const vector, or the view can be a 'const view' which is fixed to point only to one vector (whether or not the vector pointed to happens to be const). This is exactly analogous to the distinction between const double *p, double * const p, and const double * const p. The first is a non-const pointer to const data, the second is a const pointer to non-const data, and the third is a const pointer to const data. The equivalent expressions in O_2scl are

```
ovector_const_view v1;
const ovector_view v2;
const ovector_const_view v3;
```

Explicitly,

- `ovector_const_view` is a view of a const vector, the view may change, but the vector may not.

- `const ovector_const_view` is a const view of a const vector, the view may not point to a different vector and the vector may not change.
- `const ovector_view` is a const view of a normal vector, the view may not change, but the vector can.

  This same distinction is also present, for example, in ublas vectors views within Boost.

A reference of type ovector_base is often used as a function argument, and can hold either a ovector or a ovector_view. The important rule to remember with ovector_base is that, a const reference, i.e.

```
const ovector_base &v;
```

is always a const reference to data which cannot be changed. A normal ovector_base reference does not refer to const data.

7.24 Vector and matrix output

Both the ovector_view_tlate and omatrix_view_tlate classes have an << operator defined for each class. For writing generic vectors to a stream, you can use vector_out() which is defined in vector.h . Pretty matrix output is performed by global template functions matrix_out(), matrix_cx_out_paren(), and matrix_out_paren() which are defined in columnify.h since they internally use a columnify object to format the output.

7.25 Tensors

Some preliminary support is provided for tensors of arbitrary rank and size in the class tensor. Classes tensor1, tensor2, tensor3, and tensor4 are rank-specific versions for 1-, 2-, 3- and 4-rank tensors. For n-dimsional data defined on a grid, tensor_grid provides a space to define a hyper-cubic grid in addition to the the tensor data. This class tensor_grid also provides simple n-dimensional interpolation of the data defined on the specified grid.

8 Permutations

Permutations are implemented through the permutation class. This class is fully compatible with gsl_permutation objects since it is inherited from gsl_permutation_struct. The class also contains no new data members, so upcasting and downcasting can always be performed. It is perfectly permissible to call GSL permutation functions from permutation objects by simply passing the address of the permutation, i.e.

```
permutation p(4);
p.init();
gsl_permutation_swap(&p,2,3);
```

The functions which apply a permutation to a user-specified vector are member template functions in the permutation class (see permutation::apply()).

Memory allocation/deallocation between the class and the gsl_struct is compatible in many cases, but mixing these forms is strongly discouraged, i.e. avoid using `gsl_permutation_alloc()` on a permutation object, but rather use permutation::allocate() instead. The use of permutation::free() is encouraged, but any remaining memory will be deallocated in the object destructor.

9 Linear Algebra

There is a small set of linear algebra routines. These are not intended to be a replacement for higher performance linear algebra libraries, but offer a very generic and flexible interface while providing performance sufficient for all but the most intensive applications. They work for almost all vector and matrix types. For vector and matrix types using `operator[]`, the BLAS and linear algebra routines are inside the o2scl_cblas and o2scl_linalg namespaces. For vector and matrix types using `operator()`, the BLAS and linear algebra routines routines are inside the o2scl_cblas_paren and o2scl_linalg_paren namespaces.

The linear algebra classes and functions include:

- Householder transformations (householder.h)
- Householder solver (hh.h)
- LU decomposition and solver (lu.h)
- Cholesky decomposition (cholesky.h)
- QR decomposition and solver (qr.h)
- Solve tridiagonal systems (tridiag.h)
- Lanczos diagonalization is inside class lanczos, which also can compute the eigenvalues of a tridiagonal matrix.
- Givens rotations (givens.h)

There is also a set of linear solvers for generic matrix and vector types which descend from o2scl_linalg::linear_solver. These classes provide GSL-like solvers, but are generalized so that they are compatible with vector and matrix types which allow access through `operator[]`.

For users who require high-performance linear algebra, the ovector and omatrix objects can be used to call LAPACK routines directly, just as can be done with GSL. For an example of how to do this, see

`http://sourceware.org/ml/gsl-discuss/2001/msg00326.html` . Finally, there are also a couple of examples, `gesvd.cpp` and `zheev.cpp` in the `src/internal` directory which show how to call LAPACK with O_2scl objects which may be adaptable for your platform and configuration.

10 Interpolation

The lower-level interpolation types based on GSL all inherit from base_interp. These lower-level interpolation classes automatically use the GSL accelerated lookup system and have also been modified to handle monotonically decreasing vectors. They are template classes compatible with any vector type whose elements can be accessed with `operator[]`.

Integrals are always computed assuming that if the limits are ordered so that if the upper limit appears earlier in the array `x` in comparison to the lower limit, that the value of the integral has the opposite sign than if the upper limit appears later in the array `x` .

The classes o2scl_interp and o2scl_interp_vec provide simpler interfaces to the lower-level interpolation types taking care of memory allocation and initialization automatically. The difference between the two classes, o2scl_interp and o2scl_interp_vec, analogous to the difference between using `gsl_interp_eval()` and `gsl_spline_eval()` in GSL. You can create a o2scl_interp object and use it to interpolate among any pair of chosen vectors, i.e.

```
ovector x(20), y(20);
// fill x and y with data
o2scl_interp oi;
double y_half=oi.interp(0.5,20,x,y);
```

Alternatively, you can create a o2scl_interp_vec object which can be optimized for a pair of vectors that you specify in advance

```
ovector x(20), y(20);
// fill x and y with data
o2scl_interp_vec oi(20,x,y);
double y_half=oi.interp(0.5);
```

Two specializations for C-style arrays of double-precision numbers are provided in array_interp and array_interp_vec.

10.1 Lookup and binary search

The classes search_vec and search_vec_ext contain searching functions for generic vector types which contain monotonic (either increasing or decreasing) data. It is search_vec which is used internally by the interpolation classes to perform cached binary searching. These classes also allow one to to exahaustively search for the index of an element in a vector without regard to any kind of ordering, e.g. search_vec::ordered_lookup() (see also ovector::lookup()).

10.2 "Smart" interpolation

The classes smart_interp and smart_interp_vec allow interpolation, lookup, differentiation, and integration of data which is non-monotonic or multiply-valued outside the region of interest. As with o2scl_interp above, the corresponding array versions are given in sma_interp and sma_interp_vec.

10.3 Interpolation manager objects

Many O_2scl classes require the ability to create and destroy several interpolation objects to interpolate many different vectors at the same time. For this purpose, interpolation managers have been created, which allow the user to provide a class with an interpolation manager associated with a low-level interpolation type (and thus free the user from having to worry about the associated memory management). The abstract base interpolation manager type is base_interp_mgr, and the default interpolation manager object is def_interp_mgr . The def_interp_mgr class has two template arguments: the first is the vector type to be interpolated, and the second is the lower-lever interpolation type. The user is free to use the same interpolation manager instance for many different classes which require interpolation because the interpolation manager allocates separate interpolation instances for each class as necessary.

For an example usage of the default interpolation manager, see the Contour lines , which specifies an interpolation manager for the contour class. The table class also works with interpolation manager objects (see table::set_interp()).

10.4 Two and higher dimensional interpolation

Support for two-dimensional interpolation (by successive one-dimensional interpolations) is given in twod_intp (see Two-dimensional Interpolation), and a simple version of n-dimensional interpolation in tensor_grid.

10.5 Inverse interpolation and other functions

The equivalent to "inverse" linear interpolation, which computes all the abcissae which have a fixed value of the ordinate is implemented in the template function vector_find_level() which is documented in smart_interp.h . This function together with vector_invert_enclosed_sum() can be used to determine confidence limits surrounding the peak of a 1-dimensional data set using linear interpolation.

11 Physical Constants

The constants from GSL are reworked with the type `const double` and placed in namespaces called gsl_cgs, gsl_cgsm, gsl_mks, gsl_mksa, and gsl_num. Some additional constants are given in the namespace o2scl_const. Some of the numerical values have been updated from recently released data from NIST.

12 Function Objects

Functions are passed to numerical routines using template-based function classes, sometimes called "functors". There are several basic kinds of function objects:

- funct : One function of one variable
- multi_funct : One function of several variables
- mm_funct : `n` functions of `n` variables
- fit_funct : One function of one variable with `n` fitting parameters
- ode_funct : `n` derivatives as a function of `n` function values and the value of the independent variable

- jac_funct : Jacobian function for solver
- ool_hfunct : Hessian product for constrained minimization

The class name suffixes denote children of a generic function type which are created using different kinds of inputs:

- _fptr: function pointer for a static or global function
- _gsl: GSL-like function pointer
- _mfptr: function pointer template for a class member function
- _cmfptr: function pointer template for a class member function which is const
- _strings: functions specified using strings, e.g. "x^2-2", which are in the separate O$_2$scl_ext package.
- _noerr: (for funct and multi_funct) a function which directly returns the function value rather than returning an integer error value
- _nopar: (for funct) a function which has no parameters and directly returns the function value.

See the Function object example and the Multi-dimensional solver which provide detailed examples of how functions can be specified to classes through these function objects.

12.1 Connection to STL and Boost

The O$_2$scl function types are very closely related to similar types in the Standard Template Library (STL) and in Boost. In the language of the STL, funct implements the concept of a `Unary Function` where the argument and return types are both `double`. In turn, most O$_2$scl classes which have a template parameter `func_t` will accept all classes which implement this same concept. Note that in the STL, Unary Functions also implement the Assignable concept, and O$_2$scl classes often do not require this for objects of type `func_t`. The Boost library handles things in a very similar way, except that STL concepts are referred to as "traits", and there is a much more complete list of "type traits".

12.2 Function object details

There is a small overhead associated with the indirection: a "user class" accesses the function class which then calls function which was specified in the constructor of the function class. In many problems, the overhead associated with the indirection is small. Some of this overhead can always be avoided by inheriting directly from the function class and thus the user class will make a direct virtual function call. To eliminate the overhead entirely, one can specify a new type for the template parameter in the user class.

Virtual functions can be specified through this mechanism as well. For example, if cern_mroot is used to solve a set of equations specified as

```
class my_type_t {
  virtual member_func();
};
my_type_t my_instance;
class my_derived_type_t : public my_type_t {
  virtual member_func();
};
my_derived_type_t my_inst2;
mm_funct_mfptr<my_type_t> func(&my_inst2,&my_instance::member_func);
```

Then the solver will solve the member function in the derived type, not the parent type.

Note that providing a user access to a function object instantiatied with a protected or private member function is (basically) the same as providing them access to that function.

12.3 Function object example

This example shows how to provide functions to O$_2$scl classes by solving the equation

$$\left\{1+\frac{1}{p_2}\sin\left[50\left(x-p_1\right)\right]\right\}\tan^{-1}\left[4\left(x-p_1\right)\right]=0$$

Where $p_1=0.01$ and $p_2=1.1$. The parameter p_1 is stored as member data for the class, and the parameter p_2 is an argument to the member function.

The image below shows how the solver progresses to the solution of the example function.

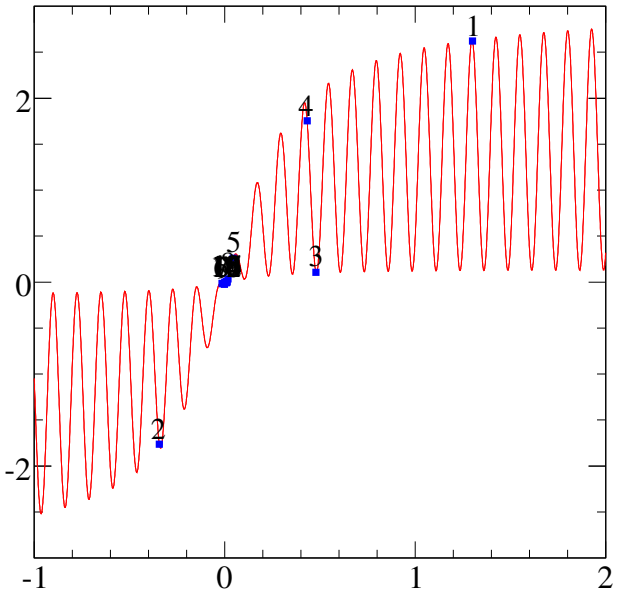


Figure 2: Function object example plot

```
/* Example: ex_fptr.cpp
   -------------------------------------------------------------------
   This gives an example of the how member functions and external
   parameters are supplied to numerical routines. In this case, a
   member function with two parameters is passed to the gsl_root_brent
   class, which solves the equation. One of the parameters is member
   data, and the other is specified using the extra parameter argument
   to the function.
*/

#include <o2scl/funct.h>
#include <o2scl/gsl_root_brent.h>
#include <o2scl/test_mgr.h>

using namespace std;
using namespace o2scl;

class my_class {

private:

  double parameter;

public:
```

```
  void set_parameter() { parameter=0.01; }

  // A function demonstrating the different ways of implementing
  // function parameters
  double function_to_solve(double x, double &p) {
    return atan((x-parameter)*4)*(1.0+sin((x-parameter)*50.0)/p);
  }

};

// Simple code to write the function to a file
int write_file(double sol);

int main(void) {

  cout.setf(ios::scientific);

  test_mgr t;
  // Only print something out if one of the tests fails
  t.set_output_level(1);

  // The solver, specifying the type of the parameter (double)
  // and the function type (funct<double>)
  gsl_root_brent<> solver;

  my_class c;
  c.set_parameter();

  double p=1.1;

  // This is the code that allows specification of class member
  // functions as functions to solve. This approach avoids the use of
  // static variables and functions and multiple inheritance at the
  // expense of a little overhead. We need to provide the address of
  // an instantiated object and the address of the member function.
  funct_mfptr_param<my_class,double>
    function(&c,&my_class::function_to_solve,p);

  double x1=-1;
  double x2=2;

  // The value verbose=1 prints out iteration information
  // and verbose=2 requires a keypress between iterations.
  solver.verbose=1;
  solver.solve_bkt(x1,x2,function);

  // This is actually a somewhat difficult function to solve because
  // of the sinusoidal behavior.
  cout << "Solution: " << x1
       << " Function value: " << c.function_to_solve(x1,p) << endl;

  // Write the function being solved to a file (see source code
  // in examples directory for details)
  write_file(x1);

  // Obtain and summarize test results
  t.test_abs(c.function_to_solve(x1,p),0.0,1.0e-10,"ex_fptr");
  t.report();

  return 0;
}
// End of example
```

13 Data Tables

The class table is a container to hold and perform operations on related columns of data. It supports column operations, interpolation, column reference by either name or index, binary searching (in the case of ordered columns), sorting, and fitting two columns to a user-specified function.

14 String Manipulation

There are a couple classes and functions to help manipulate strings of text. Conversion routines for `std::string` objects are given in string_conv.h and include

- ptos() - pointer to string
- itos() - integer to string
- dtos() - double to string
- stoi() - string to integer
- stod() - string to double

See also size_of_exponent() and double_to_ieee_string().

The columnify class converts a set of strings into nicely formatted columns by padding with the necessary amount of spaces. This class operates on string objects of type `std::string`, and also works will for formatting columns of floating-point numbers. This class is used to provide output for matrices in the functions matrix_out(), matrix_out_paren(), and matrix_cx_out_paren(). For output of vectors, see vector_out() in array.h.

The format_float class will reformat double precision numbers into a form appropriate for HTML or LaTeX documents.

A related function, screenify(), reformats a column of strings into many columns stored row-by-row in a new string array. It operates very similar to the way the classic Unix command `ls` organizes files and directories in multiple columns in order to save screen space.

The function count_words() counts the number of "words" in a string, which are delimited by whitespace.

15 Differentiation

Differentiation is performed by descendants of deriv and the classes are provided. These allow one to calculate either first, second, and third derivatives. The GSL approach is used in gsl_deriv, and the CERNLIB routine is used in cern_deriv. Both of these compute derivatives for a function specified using a descendant of funct. For functions which are tabulated over equally-spaced abscissas, the class eqi_deriv is provided which applies the formulas from Abramowitz and Stegun at a specified order. The class cern_deriv is slower and sometimes more accurate, but also fails more often than gsl_deriv, which never calls the error handler.

Warning: For gsl_deriv and cern_deriv, the second and third derivatives are calculated by naive repeated application of the code for the first derivative and can be particularly troublesome if the function is not sufficiently smooth. Error estimation is not provided for second and third derivatives.

15.1 Numerical differentiation

This example computes first and second derivatives of

$$y(x) = \sin(2x) + \frac{1}{2}$$

with both gsl_deriv and cern_deriv .

```
/* Example: ex_deriv.cpp
   -------------------------------------------------------------------
   An example to demonstrate numerical differentiation
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/funct.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/gsl_deriv.h>
#include <o2scl/cern_deriv.h>

using namespace std;
using namespace o2scl;

class cl {

public:

  // This is the function we'll take the derivative of
  double function(double x) {
    return sin(2.0*x)+0.5;
  }
};

int main(void) {

  test_mgr t;
  t.set_output_level(2);

  // The class and associated function
  cl acl;
  funct_mfptr<cl> f1(&acl,&cl::function);

  gsl_deriv<> gd;
  // Note that the GSL derivative routine requires an initial stepsize
  gd.h=1.0e-3;
  cern_deriv<> cd;

  // Compute the first derivative using the gsl_deriv class and
  // verify that the answer is correct
  double d1=gd.calc(1.0,f1);
  t.test_rel(d1,2.0*cos(2.0),1.0e-10,"gsl_deriv");

  // Compute the first derivative using the cern_deriv class and
  // verify that the answer is correct
  double d2=cd.calc(1.0,f1);
  t.test_rel(d2,2.0*cos(2.0),1.0e-10,"cern_deriv");

  // Compute the second derivative also
  double d3=gd.calc2(1.0,f1);
  t.test_rel(d3,-4.0*sin(2.0),5.0e-7,"gsl_deriv");

  double d4=cd.calc2(1.0,f1);
  t.test_rel(d4,-4.0*sin(2.0),1.0e-8,"cern_deriv");

  t.report();
  return 0;
}
// End of example
```

16 Integration

16.1 One-dimensional integration

Several classes integrate arbitrary one-dimensional functions

- Integration over a finite interval: cern_adapt, cern_gauss, cern_gauss56, gsl_inte_qag, and gsl_inte_qng.
- Integration from 0 to ∞ : gsl_inte_qagiu
- Integration from $-\infty$ to 0: gsl_inte_qagil
- Integration from $-\infty$ to ∞ : gsl_inte_qagi
- Integration over a finite interval for a function with singularities: gsl_inte_qags (see also gsl_inte_qaws).
- Cauchy principal value integration over a finite interval: cern_cauchy and gsl_inte_qawc
- Integration over a function weighted by `cos(x)` or `sin(x)` : gsl_inte_qawo_cos and gsl_inte_qawo_sin
- Fourier integrals: gsl_inte_qawf_cos and gsl_inte_qawf_sin
- Integration over a weight function

$$W(x) = (x-a)^{\alpha}(b-x)^{\beta}\log^{\mu}(x-a)\log^{\nu}(b-x)$$

  is performed by gsl_inte_qaws.

All of these classes are children of inte, and the relative and absolute tolerances are stored in inte::tol_rel and inte::tol_abs, respectively.

16.2 GSL-based integration routines

For the GSL-based integration routines, the variables inte::tol_abs and inte::tol_rel have the same role as the quantities usually denoted in the GSL integration routines by `epsabs` and `epsrel`. In particular, the integration classes attempt to ensure that

$$|\text{result} - I| \leq \text{Max}(\text{tolx}, \text{tolf}|I|)$$

and returns an error to attempt to ensure that

$$|\text{result} - I| \leq \text{abserr} \leq \text{Max}(\text{tolx}, \text{tolf}|I|)$$

where `I` is the integral to be evaluated. Even when the corresponding descendant of inte::integ() returns success, these inequalities may fail for sufficiently difficult functions. All of the GSL integration routines except for gsl_inte_qng use a workspace given in gsl_inte_table which holds the results of the various subdivisions of the original interval. The size of this workspace (and thus then number of subdivisions) can be controlled with gsl_inte_table::set_wkspace().

The GSL routines were originally based on QUADPACK, which is available at `http://www.netlib.org/quadpack` .

For adaptive GSL integration classes, the type of Gauss-Kronrod quadrature rule that is used to approximate the integral and estimate the error of a subinterval is set by gsl_inte_kronrod::set_rule().

There are two competing factors that can slow down an adaptive integration algorithm: (1) each evaluation of the integrand can be numerically expensive, depending on how the function is defined, and (2) the process of subdividing regions and recalculating values is almost always numerically expensive in its own right. For integrands that are very smooth (e.g., analytic functions), a high-order Gauss-Kronrod rule (e.g., 61-point) will achieve the desired error tolerance with relatively few subdivisions. For integrands with discontinuities or singular derivatives, a low-order rule (e.g., 15-point) is often more efficient.

The number of subdivisions of the integration region is limited by the size of the workspace, set in gsl_inte_kronrod::set_limit(). The number of subdivisions required for the most recent call to inte::integ() or inte::integ_err() is given in inte::last_iter. This number will always be less than or equal to the workspace size.

Note

The GSL integration routines can sometimes lose precision if the integrand is everywhere much smaller than unity. Some rescaling is required in these cases.

16.3 GSL-based integration error messages

The error messages given by the adaptive GSL integration routines tend to follow a standard form and are documented here. - There are several error messages which indicate improper usage and cause the error handler to be called regardless of the value of inte::err_nonconv :

- `"Iteration limit exceeds workspace in class::function()."` [gsl_einval]
- `"Could not integrate function in class::function() (it may have returned a non-finite result)."` [gsl_efailed] This often occurs when the user-specified function returns `inf` or `nan`.
- `"Tolerance cannot be achieved with given value of tol_abs and tol_rel in class::function()."` [gsl_ebadtol]

  This occurs if the user supplies unreasonable values for inte::tol_abs and inte::tol_rel. All positive values for inte::tol_abs are allowed. If zero-tolerance for inte::tol_abs is desired, then inte::tol_rel must be at least $50 \cdot \varepsilon_{\text{mach}}$ ($\approx 1.11 \times 10^{-14}$).
- `"Cannot integrate with singularity on endpoint in gsl_inte_qawc::qawc()."` [gsl_einval] The class gsl_inte_qawc cannot handle the case when a singularity is one of the endpoints of the integration.

There are also convergence errors which will call the error handler unless inte::err_nonconv is false:

- `"Cannot reach tolerance because of roundoff error on first attempt in class::function()."` [gsl_eround]

  Each integration attempt tests for round-off errors by comparing the computed integral with that of the integrand's absolute value (i.e., L^1-norm). A highly oscillatory integrand may cause this error.
- `"A maximum of 1 iteration was insufficient in class::function()."` [gsl_emaxiter]

  This occurs if the workspace is allocated for one interval and a single Gauss-Kronrod integration does not yield the accuracy demanded by inte::tol_abs and inte::tol_rel.
- `"Bad integrand behavior in class::function()."` [gsl_esing]

  This occurs if the integrand is (effectively) singular in a region, causing the subdivided intervals to become too small for floating-point precision.
- `"Maximum number of subdivisions 'value' reached in class::function()."` [gsl_emaxiter]

  This occurs if the refinement algorithm runs out of allocated workspace. The number of iterations required for the most recent call to inte::integ() or inte::integ_err() is given in inte::last_iter. This number will always be less than or equal to the workspace size.
- `"Roundoff error prevents tolerance from being achieved in class::function()."` [gsl_eround]

  The refinement procedure counts round-off errors as they occur and terminates if too many such errors accumulate.
- `"Roundoff error detected in extrapolation table in gsl_inte_singular::qags()."` [gsl_eround]

  This occurs when error-terms from the ε-algorithm are are monitored and compared with the error-terms from the refinement procedure. The algorithm terminates if these sequences differ by too many orders of magnitude. See gsl_inte_singular::qelg().
- `"Integral is divergent or slowly convergent in gsl_inte_singular::qags()."` [gsl_ediverge]

  This occurs if the approximations produced by the refinement algorithm and the extrapolation algorithm differ by too many orders of magnitude.
- `"Exceeded limit of trigonometric table in gsl_inte_qawo_sin()::qawo()."` [gsl_etable]

  This occurs if the maximum **level** of the table of Chebyshev moments is reached.

16.4 Multi-dimensional integration routines

Multi-dimensional hypercubic integration is performed by composite_inte, the sole descendant of multi_inte. composite_inte allows you to specify a set of one-dimensional integration routines (objects of type inte) and apply them to a multi-dimensional problem.

General multi-dimensional integration is performed by comp_gen_inte, the sole descendant of gen_inte. The user is allowed to specify a upper and lower limits which are functions of the variables for integrations which have not yet been performed, i.e. the n-dimensional integral

$$\int_{x_0=a_0}^{x_0=b_0} f(x_0) \int_{x_1=a_1(x_0)}^{x_1=b_1(x_0)} f(x_0,x_1) \ldots \int_{x_{n-1}=a_{n-1}(x_0,x_1,\ldots,x_{n-2})}^{x_{n-1}=b_{n-1}(x_0,x_1,\ldots,x_{n-2})} f(x_0,x_1,\ldots,x_{n-1})\, dx_{n-1} \ldots dx_1\, dx_0$$

Again, one specifies a set of inte objects to apply to each variable to be integrated over.

Monte Carlo integration is also provided (see Monte Carlo Integration).

16.5 Integration Example

This example computes the integral $\int_{-\infty}^{\infty} e^{-x^2}\, dx$ with gsl_inte_qagi, the integral $\int_{0}^{\infty} e^{-x^2}\, dx$ with gsl_inte_qagiu, the integral $\int_{-\infty}^{0} e^{-x^2}\, dx$ with gsl_inte_qagil, and the integral $\int_0^1 \left[\sin(2x)+\frac{1}{2}\right]\, dx$ with both gsl_inte_qag and cern_adapt, and compares the computed results with the exact results.

```
/* Example: ex_inte.cpp
   -------------------------------------------------------------------
   An example to demonstrate numerical integration.
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/constants.h>
#include <o2scl/funct.h>
#include <o2scl/gsl_inte_qag.h>
#include <o2scl/gsl_inte_qagi.h>
#include <o2scl/gsl_inte_qagiu.h>
#include <o2scl/gsl_inte_qagil.h>
#include <o2scl/cern_adapt.h>

using namespace std;
using namespace o2scl;
using namespace o2scl_const;

class cl {

public:

  // We'll use this to count the number of function
  // evaulations required by the integration routines
  int nf;

  // A function to be integrated
  double integrand(double x) {
    nf++;
    return exp(-x*x);
  }

  // Another function to be integrated
  double integrand2(double x) {
    nf++;
    return sin(2.0*x)+0.5;
  }
};

int main(void) {
  cl acl;
  test_mgr t;
```

```
t.set_output_level(1);

funct_mfptr<cl> f1(&acl,&cl::integrand);
funct_mfptr<cl> f2(&acl,&cl::integrand2);

// We don't need to specify the function type in the integration
// objects, because we're using the default function type (type
// funct).
gsl_inte_qag<> g;
gsl_inte_qagi<> gi;
gsl_inte_qagiu<> gu;
gsl_inte_qagil<> gl;
cern_adapt<> ca;

// The result and the uncertainty
double res, err;

// An integral from -infinity to +infinity (the limits are ignored)
acl.nf=0;
int ret1=gi.integ_err(f1,0.0,0.0,res,err);
cout << "gsl_inte_qagi: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
cout << "Number of iterations: " << gi.last_iter << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(res,sqrt(pi),1.0e-8,"inte 1");

// An integral from 0 to +infinity (the second limit argument is
// ignored in the line below)
acl.nf=0;
gu.integ_err(f1,0.0,0.0,res,err);
cout << "gsl_inte_qagiu: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
cout << "Number of iterations: " << gu.last_iter << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(res,sqrt(pi)/2.0,1.0e-8,"inte 2");

// An integral from -infinity to zero (the first limit argument is
// ignored in the line below)
acl.nf=0;
gl.integ_err(f1,0.0,0.0,res,err);
cout << "gsl_inte_qagil: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
cout << "Number of iterations: " << gl.last_iter << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(res,sqrt(pi)/2.0,1.0e-8,"inte 3");

// An integral from 0 to 1 with the GSL integrator
acl.nf=0;
g.integ_err(f2,0.0,1.0,res,err);
cout << "gsl_inte_qag: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
cout << "Number of iterations: " << g.last_iter << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << endl;
t.test_rel(res,0.5+sin(1.0)*sin(1.0),1.0e-8,"inte 4");

// The same integral with the CERNLIB integrator
acl.nf=0;
ca.integ_err(f2,0.0,1.0,res,err);
cout << "cern_adapt: " << endl;
cout << "Return value: " << ret1 << endl;
cout << "Result: " << res << " Uncertainty: " << err << endl;
```

```
  cout << "Number of iterations: " << ca.last_iter << endl;
  cout << "Number of function evaluations: " << acl.nf << endl;
  cout << endl;
  t.test_rel(res,0.5+sin(1.0)*sin(1.0),1.0e-8,"inte 5");

  t.report();
  return 0;
}
// End of example
```

17 Roots of Polynomials

Classes are provided for solving quadratic, cubic, and quartic equations as well as general polynomials. There is a standard nomenclature: classes which handle polynomials with real coefficients and real roots end with the suffix `_real` (quadratic_real, cubic_real and quartic_real), classes which handle real coefficients and complex roots end with the suffix `_real_coeff` (quadratic_real_coeff, cubic_real_coeff, quartic_real_coeff, and poly_real_coeff), and classes which handle complex polynomials with complex coefficients end with the suffix `_complex` (quadratic_complex, cubic_complex, quartic_complex, and poly_complex). As a reminder, complex roots may not occur in conjugate pairs if the coefficients are not real. Most of these routines will return an error if the leading coefficient is zero.

In the public interfaces to the polynomial solvers, the complex type `std::complex<double>` is used. These can be converted to and from the GSL complex type using the complex_to_gsl() and gsl_to_complex() functions.

For quadratics, gsl_quadratic_real_coeff is the best if the coefficients are real, while if the coefficients are complex, use quadratic_std_complex. For cubics with real coefficients, cern_cubic_real_coeff is the best, while if the coefficients are complex, use cubic_std_complex.

For a quartic polynomial with real coefficients, cern_quartic_real_coeff is the best, unless the coefficients of odd powers happen to be small, in which case, gsl_quartic_real2 tends to work better. For quartics, generic polynomial solvers such as gsl_poly_real_coeff can provide more accurate (but slower) results. If the coefficients are complex, then you can use simple_quartic_complex.

17.1 Polynomial solver example

This example shows how to find the roots of the second-, third-, fourth-, and fifth-order Chebyshev polynomials

$$2x^2 - 1$$
$$4x^3 - 3x$$
$$8x^4 - 8x^2 + 1$$
$$16x^5 - 20x^3 + 5x$$

For the Chebyshev polynomial of order n, the roots are given by

$$\cos\left[\frac{\pi(k-1/2)}{n}\right]$$

for $k = 1, \ldots, n$ These roots are used in gsl_chebapp to approximate functions using Chebyshev polynomials .

```
/* Example: ex_poly.cpp
   -------------------------------------------------------------------
   Demonstrate the solution of the Chebyshev polynomials
*/

#include <o2scl/poly.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/vector.h>
// For pi
#include <o2scl/constants.h>
#include <o2scl/test_mgr.h>
```

```
using namespace std;
using namespace o2scl;
using namespace o2scl_const;

int main(void) {

  cout.setf(ios::scientific);
  cout.setf(ios::showpos);

  test_mgr t;
  t.set_output_level(1);

  // Quadratic solver
  gsl_quadratic_real_coeff quad;
  // Cubic solver
  cern_cubic_real_coeff cubic;
  // Quartic solver
  cern_quartic_real_coeff quart;
  // Generic polynomial solver
  gsl_poly_real_coeff gen;

  // Storage for the roots
  ovector v(5);
  double d;
  std::complex<double> ca[5];

  // The second order polynomial
  cout << "Second order roots: " << endl;
  quad.solve_r(2.0,0.0,-1.0,v[0],v[1]);

  // Sort the roots and compare with the exact results
  vector_sort<ovector,double>(2,v);
  for(size_t i=0;i<2;i++) {
    double exact=cos(pi*(((double)(2-i))-0.5)/2.0);
    cout << v[i] << " " << exact << endl;
    t.test_abs(v[i],exact,1.0e-14,"2nd order");
  }
  cout << endl;

  // The third order polynomial
  cout << "Third order roots: " << endl;
  cubic.solve_rc(4.0,0.0,-3.0,0.0,v[0],ca[0],ca[1]);

  // Sort the roots and compare with the exact results
  v[1]=ca[0].real();
  v[2]=ca[1].real();
  vector_sort<ovector,double>(3,v);
  for(size_t i=0;i<3;i++) {
    double exact=cos(pi*(((double)(3-i))-0.5)/3.0);
    cout << v[i] << " " << exact << endl;
    if (i==1) {
      t.test_abs(v[i],exact,1.0e-14,"3rd order");
    } else {
      t.test_abs(v[i],exact,1.0e-14,"3rd order");
    }
  }
  cout << endl;

  // The fourth order polynomial
  cout << "Fourth order roots: " << endl;
  quart.solve_rc(8.0,0.0,-8.0,0.0,1.0,ca[0],ca[1],ca[2],ca[3]);

  // Sort the roots and compare with the exact results
  for(size_t i=0;i<4;i++) v[i]=ca[i].real();
  vector_sort<ovector,double>(4,v);
  for(size_t i=0;i<4;i++) {
    double exact=cos(pi*(((double)(4-i))-0.5)/4.0);
    cout << v[i] << " " << exact << endl;
    t.test_abs(v[i],exact,1.0e-14,"4th order");
```

```
  }
  cout << endl;

  // The fifth order polynomial
  cout << "Fifth order roots: " << endl;
  double co[6]={16.0,0.0,-20.0,0.0,5.0,0.0};
  gen.solve_rc(5,co,ca);

  // Sort the roots and compare with the exact results
  for(size_t i=0;i<5;i++) v[i]=ca[i].real();
  vector_sort<ovector,double>(5,v);
  for(size_t i=0;i<5;i++) {
    double exact=cos(pi*(((double)(5-i))-0.5)/5.0);
    cout << v[i] << " " << exact << endl;
    t.test_abs(v[i],exact,1.0e-14,"5th order");
  }
  cout << endl;

  t.report();
  return 0;
}
// End of example
```

18 Equation Solving

18.1 One-dimensional solvers

Solution of one equation in one variable is accomplished by children of the class root.

For one-dimensional solving, use cern_root or gsl_root_brent if you have the root bracketed, or gsl_root_stef if you have the derivative available. If you have neither a bracket or a derivative, you can use cern_mroot_root.

The root base class provides the structure for three different solving methods:

- root::solve() which solves a function given an initial guess `x`
- root::solve_bkt() which solves a function given a solution bracketed between `x1` and `x2`. The values of the function at `x1` and `x2` should have different signs.
- root::solve_de() which solves a function given an initial guess `x` and the derivative of the function `df`.

If not all of these three functions are overloaded, then the source code in the root base class is designed to try to automatically provide the solution using the remaining functions. Most of the one-dimensional solving routines, in their original form, are written in the second or third form above. For example, gsl_root_brent is originally a bracketing routine of the form root::solve_bkt(), but calls to either root::solve() or root::solve_de() will attempt to automatically bracket the function given the initial guess that is provided. Of course, it is frequently most efficient to use the solver in the way it was intended.

18.2 Multi-dimensional solvers

Solution of more than one equation is accomplished by descendants of the class mroot . The higher-level interface is provided by the function mroot::msolve() .

For multi-dimensional solving, you can use either cern_mroot or gsl_mroot_hybrids. While cern_mroot cannot utilize user-supplied derivatives, gsl_mroot_hybrids can use user-supplied derivative information (as in the GSL hybridsj method) using the function gsl_mroot_hybrids::msolve_de() .

18.3 Multi-dimensional solver

This demonstrates several ways of using the multi-dimensional solvers to solve the equations

$$\sin\left(x_0-\frac{1}{4}\right) = 0$$
$$\sin\left(x_1-\frac{1}{5}\right) = 0$$

```
/* Example: ex_mroot.cpp
   -------------------------------------------------------------------
   Several ways to use an O2scl solver to solve a simple function
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/mm_funct.h>
#include <o2scl/gsl_mroot_hybrids.h>
#include <o2scl/cern_mroot.h>

using namespace std;
using namespace o2scl;

int gfn(size_t nv, const ovector_base &x, ovector_base &y) {
  y[0]=sin(x[1]-0.2);
  y[1]=sin(x[0]-0.25);
  return 0;
}

typedef double arr_t[2];
typedef double mat_t[2][2];

class cl {

public:

  // Store the number of function and derivative evaluations
  int nf, nd;

  int mfn(size_t nv, const ovector_base &x, ovector_base &y) {
    y[0]=sin(x[1]-0.2);
    y[1]=sin(x[0]-0.25);
    nf++;
    return 0;
  }

  int operator()(size_t nv, const ovector_base &x, ovector_base &y) {
    y[0]=sin(x[1]-0.2);
    y[1]=sin(x[0]-0.25);
    nf++;
    return 0;
  }

  int mfnd(size_t nv, ovector_base &x, ovector_base &y, omatrix_base &j) {
    j[0][0]=0.0;
    j[0][1]=cos(x[1]-0.2);
    j[1][0]=cos(x[0]-0.25);
    j[1][1]=0.0;
    nd++;
    return 0;
  }

  int mfna(size_t nv, const arr_t &x, arr_t &y) {
    y[0]=sin(x[1]-0.2);
    y[1]=sin(x[0]-0.25);
    return 0;
  }
```

```
  int mfnad(size_t nv, arr_t &x, arr_t &y, mat_t &j) {
    j[0][0]=0.0;
    j[0][1]=cos(x[1]-0.2);
    j[1][0]=cos(x[0]-0.25);
    j[1][1]=0.0;
    return 0;
  }

};

int main(void) {
  cl acl;
  ovector x(2);
  double xa[2];
  int i;
  size_t tmp;
  int r1, r2, r3;
  bool done;
  test_mgr t;

  t.set_output_level(1);

  /*
    Using a member function with \ref ovector objects
  */
  mm_funct_mfptr<cl> f1(&acl,&cl::mfn);
  gsl_mroot_hybrids<> cr1;

  x[0]=0.5;
  x[1]=0.5;
  acl.nf=0;
  int ret1=cr1.msolve(2,x,f1);
  cout << "GSL solver (numerical Jacobian): " << endl;
  cout << "Return value: " << ret1 << endl;
  cout << "Number of iterations: " << cr1.last_ntrial << endl;
  cout << "Number of function evaluations: " << acl.nf << endl;
  cout << endl;
  t.test_rel(x[0],0.25,1.0e-6,"1a");
  t.test_rel(x[1],0.2,1.0e-6,"1b");

  /*
    Using the CERNLIB solver
  */
  cern_mroot<> cr2;

  x[0]=0.5;
  x[1]=0.5;
  acl.nf=0;
  int ret2=cr2.msolve(2,x,f1);
  cout << "CERNLIB solver (numerical Jacobian): " << endl;
  cout << "Return value: " << ret2 << endl;
  cout << "INFO parameter: " << cr2.get_info() << endl;
  cout << "Number of function evaluations: " << acl.nf << endl;
  cout << endl;
  t.test_rel(x[0],0.25,1.0e-6,"2a");
  t.test_rel(x[1],0.2,1.0e-6,"2b");

  /*
    Using a member function with \ref ovector objects, but
    using the GSL-like interface with set() and iterate().
  */
  gsl_mroot_hybrids<> cr3;

  x[0]=0.5;
  x[1]=0.5;
  cr3.allocate(2);
  cr3.set(2,x,f1);
  done=false;
  do {
    r3=cr3.iterate();
```

```
  double resid=fabs(cr3.f[0])+fabs(cr3.f[1]);
  if (resid<cr3.tol_rel || r3>0) done=true;
} while (done==false);
t.test_rel(cr3.x[0],0.25,1.0e-6,"3a");
t.test_rel(cr3.x[1],0.2,1.0e-6,"3b");
cr3.free();

/*
  Now instead of using the automatic Jacobian, using
  a user-specified Jacobian.
*/
jac_funct_mfptr<cl,ovector_base,omatrix_base> j4(&acl,&cl::mfnd);

x[0]=0.5;
x[1]=0.5;
acl.nf=0;
acl.nd=0;
int ret4=cr1.msolve_de(2,x,f1,j4);
cout << "GSL solver (analytic Jacobian): " << endl;
cout << "Return value: " << ret4 << endl;
cout << "Number of iterations: " << cr1.last_ntrial << endl;
cout << "Number of function evaluations: " << acl.nf << endl;
cout << "Number of Jacobian evaluations: " << acl.nd << endl;
cout << endl;
t.test_rel(x[0],0.25,1.0e-6,"4a");
t.test_rel(x[1],0.2,1.0e-6,"4b");

/*
  Using a user-specified Jacobian and the GSL-like interface
*/
gsl_mroot_hybrids<> cr5;

x[0]=0.5;
x[1]=0.5;
cr5.allocate(2);
cr5.set_de(2,x,f1,j4);
done=false;
do {
  r3=cr5.iterate();
  double resid=fabs(cr5.f[0])+fabs(cr5.f[1]);
  if (resid<cr5.tol_rel || r3>0) done=true;
} while (done==false);
t.test_rel(cr5.x[0],0.25,1.0e-6,"5a");
t.test_rel(cr5.x[1],0.2,1.0e-6,"5b");
cr5.free();

/*
  Using C-style arrays instead of ovector objects
*/
mm_funct_mfptr<cl,arr_t> f6(&acl,&cl::mfna);
gsl_mroot_hybrids<mm_funct_mfptr<cl,arr_t>,arr_t,
  arr_t,array_alloc<arr_t> > cr6;

xa[0]=0.5;
xa[1]=0.5;
cr6.msolve(2,xa,f6);
t.test_rel(xa[0],0.25,1.0e-6,"6a");
t.test_rel(xa[1],0.2,1.0e-6,"6b");

/*
  Using the CERNLIB solver with C-style arrays instead of ovector objects
*/
cern_mroot<mm_funct_mfptr<cl,arr_t>,arr_t,
  arr_t,array_alloc<arr_t> > cr7;

xa[0]=0.5;
xa[1]=0.5;
cr7.msolve(2,xa,f6);
t.test_rel(xa[0],0.25,1.0e-6,"7a");
t.test_rel(xa[1],0.2,1.0e-6,"7b");
```

```
  /*
    Using C-style arrays with a user-specified Jacobian
  */
  jac_funct_mfptr<cl,arr_t,mat_t> j8(&acl,&cl::mfnad);
  gsl_mroot_hybrids<mm_funct_mfptr<cl,arr_t>,arr_t,
    arr_t,array_alloc<arr_t>,mat_t,mat_t,
    array_2d_alloc<mat_t>,jac_funct<arr_t,mat_t> > cr8;

  xa[0]=0.5;
  xa[1]=0.5;
  cr8.msolve_de(2,xa,f6,j8);
  t.test_rel(xa[0],0.25,1.0e-6,"8a");
  t.test_rel(xa[1],0.2,1.0e-6,"8b");

  /*
    Using a class with an operator(). Note that there can be only one
    operator() function in each class.
  */
  gsl_mroot_hybrids<cl> cr9;

  x[0]=0.5;
  x[1]=0.5;
  cr9.msolve(2,x,acl);
  t.test_rel(x[0],0.25,1.0e-6,"9a");
  t.test_rel(x[1],0.2,1.0e-6,"9b");

  /*
    Using a function pointer to a global function.
  */
  typedef int (*gfnt)(size_t, const ovector_base &, ovector_base &);
  gsl_mroot_hybrids<gfnt> cr10;
  gfnt f10=&gfn;

  x[0]=0.5;
  x[1]=0.5;
  cr10.msolve(2,x,f10);
  t.test_rel(x[0],0.25,1.0e-6,"10a");
  t.test_rel(x[1],0.2,1.0e-6,"10b");

  t.report();
  return 0;
}
// End of example
```

19 Minimization

One-dimensional minimization is performed by descendants of minimize . There are two one-dimensional minimization algorithms, cern_minimize and gsl_min_brent, and they are both bracketing algorithms type where an interval and an initial guess must be provided. If only an initial guess and no bracket is given, these two classes will attempt to find a suitable bracket from the initial guess. While the minimize base class is designed to allow future descendants to optionally use derivative information, this is not yet supported for any one-dimensional minimizers.

Multi-dimensional minimization is performed by descendants of multi_min: gsl_mmin_simp2, gsl_mmin_conp, gsl_mmin_conf, and gsl_mmin_bfgs2. (The class gsl_mmin_simp2 has been updated with the new "simplex2" method from GSL-1.12. The older "simplex" method is also available in gsl_mmin_simp .) The classes gsl_mmin_simp and gsl_mmin_simp2 do not require any derivative information. The remaining minimization classes are intended for use when the gradient of the function is available, but they can also automaticallly compute the gradient numerically. The standard way to provide the gradient is to use a child of mm_funct Finally, the user may specify the automatic gradient object of type gradient which is used by the minimizer to compute the gradient numerically when a function is not specified.

Generally, when a closed expression is available for the gradient, the gsl_mmin_conp, gsl_mmin_conf, and gsl_mmin_bfgs2 are probably better. The simplex methods are somewhat robust for sufficiently difficult functions, though restarts are sometimes required

to find the correct minimum.

See an example for the usage of the multi-dimensional minimizers in the Multidimensional minimizer example.

Simulated annealing methods are also provided for multi-dimensional minimization (see Simulated Annealing).

It is important to note that not all of the minimization routines test the second derivative to ensure that it doesn't vanish to ensure that we have indeed found a true minimum.

The class multi_min_fix provides a convenient way of fixing some of the parameters and minimizing over others, without requiring a the function interface to be rewritten. An example is given in the Minimizer fixing variables example.

19.1 Multidimensional minimizer

This example uses the O_2scl minimizers based on GSL to minimize a rather complicated three-dimensional function which has constant level surfaces which look like springs oriented along the z-axis. This example function, originally created here for O_2scl , was added later to the GSL library minimization test functions.

```
/* Example: ex_mmin.cpp
   -------------------------------------------------------------------
   Example usage of the multidimensional minimizers with and without
   gradients.
*/
#include <fstream>
#include <string>
#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/multi_funct.h>
#include <o2scl/constants.h>
#include <o2scl/gsl_mmin_simp2.h>
#include <o2scl/gsl_mmin_conf.h>
#include <o2scl/gsl_mmin_conp.h>
#include <o2scl/gsl_mmin_bfgs2.h>

using namespace std;
using namespace o2scl;

class cl {

public:

  cl() {
    param=30.0;
  }

  // To output function evaluations to a file
  ofstream fout;

  // Parameter of the quadratic
  double param;

  // Updated spring function
  double spring_two(size_t nv, const ovector_base &x) {
    double theta=atan2(x[1],x[0]);
    double r=sqrt(x[0]*x[0]+x[1]*x[1]);
    double z=x[2];
    double tmz=theta-z;
    double fact=8.0-pow(sin(tmz+o2scl_const::pi/2.0)+1.0,3.0);
    double rm1=r-1.0;
    double ret=fact+exp(rm1*rm1)+z*z/param;
    fout << x << " " << ret << endl;
    return ret;
  }

  // Gradient of the spring function
  int sgrad(size_t nv, ovector_base &x, ovector_base &g) {
```

```
    double theta=atan2(x[1],x[0]);
    double r=sqrt(x[0]*x[0]+x[1]*x[1]);
    double z=x[2];
    double tmz=theta-z;
    double rm1=r-1.0;
    double fact=8.0-pow(sin(tmz+o2scl_const::pi/2.0)+1.0,3.0);

    double dtdx=-x[1]/r/r;
    double dtdy=x[0]/r/r;
    double drdx=x[0]/r;
    double drdy=x[1]/r;
    double dfdt=-3.0*pow(sin(tmz+o2scl_const::pi/2.0)+1.0,2.0)*
      cos(tmz+o2scl_const::pi/2.0);
    double dfdz=2.0*z/param+3.0*pow(sin(tmz+o2scl_const::pi/2.0)+1.0,2.0)*
      cos(tmz+o2scl_const::pi/2.0);
    double dfdr=2.0*rm1*exp(rm1*rm1);

    g[0]=dfdr*drdx+dfdt*dtdx;
    g[1]=dfdr*drdy+dfdt*dtdy;
    g[2]=dfdz;

    return 0;
  }

};

int main(void) {
  cl acl;
  ovector x(3);
  double fmin;
  test_mgr t;

  t.set_output_level(1);
  cout.setf(ios::scientific);

  // Using a member function with \ref ovector objects
  multi_funct_mfptr<cl> f1(&acl,&cl::spring_two);
  grad_funct_mfptr<cl> f1g(&acl,&cl::sgrad);

  gsl_mmin_simp2<> gm1;
  gsl_mmin_conf<> gm2;
  gsl_mmin_conp<> gm3;
  gsl_mmin_bfgs2<> gm4;

  // This function is difficult to minimize, so more trials
  // are required.
  gm1.ntrial*=10;
  gm2.ntrial*=10;
  gm3.ntrial*=10;
  gm4.ntrial*=10;

  // Simplex minimization
  acl.fout.open("ex_mmin1.dat");
  x[0]=1.0;
  x[1]=1.0;
  x[2]=7.0*o2scl_const::pi;
  gm1.mmin(3,x,fmin,f1);
  acl.fout.close();
  cout << gm1.last_ntrial << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],1.0,1.0e-4,"1a");
  t.test_rel(x[1],0.0,1.0e-4,"1b");
  t.test_rel(x[2],0.0,1.0e-4,"1c");

  // Fletcher-Reeves conjugate
  acl.fout.open("ex_mmin2.dat");
  x[0]=1.0;
  x[1]=0.0;
  x[2]=7.0*o2scl_const::pi;
  gm2.mmin(3,x,fmin,f1);
```

```
  acl.fout.close();
  cout << gm2.last_ntrial << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],1.0,4.0e-3,"2a");
  t.test_rel(x[1],0.0,4.0e-3,"2b");
  t.test_rel(x[2],0.0,4.0e-3,"2c");

  // Fletcher-Reeves conjugate with gradients
  acl.fout.open("ex_mmin2g.dat");
  x[0]=1.0;
  x[1]=0.0;
  x[2]=7.0*o2scl_const::pi;
  gm2.mmin_de(3,x,fmin,f1,f1g);
  acl.fout.close();
  cout << gm2.last_ntrial << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],1.0,4.0e-3,"2a");
  t.test_rel(x[1],0.0,4.0e-3,"2b");
  t.test_rel(x[2],0.0,4.0e-3,"2c");

  // Polak-Ribere conjugate
  acl.fout.open("ex_mmin3.dat");
  x[0]=1.0;
  x[1]=0.0;
  x[2]=7.0*o2scl_const::pi;
  gm3.mmin(3,x,fmin,f1);
  acl.fout.close();
  cout << gm3.last_ntrial << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],1.0,4.0e-3,"3a");
  t.test_rel(x[1],0.0,4.0e-3,"3b");
  t.test_rel(x[2],0.0,4.0e-3,"3c");

  // Polak-Ribere conjugate with gradients
  acl.fout.open("ex_mmin3g.dat");
  x[0]=1.0;
  x[1]=0.0;
  x[2]=7.0*o2scl_const::pi;
  gm3.mmin_de(3,x,fmin,f1,f1g);
  acl.fout.close();
  cout << gm3.last_ntrial << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],1.0,4.0e-3,"3a");
  t.test_rel(x[1],0.0,4.0e-3,"3b");
  t.test_rel(x[2],0.0,4.0e-3,"3c");

  // BFGS method

  // BFGS has trouble converging (especially to zero, since the
  // minimimum of x[0] is exactly at zero) if the derivative is not
  // very accurate.
  gm4.def_grad.epsrel=1.0e-8;

  gm4.err_nonconv=false;
  acl.fout.open("ex_mmin4.dat");
  x[0]=1.0;
  x[1]=0.0;
  x[2]=7.0*o2scl_const::pi;
  gm4.mmin(3,x,fmin,f1);
  acl.fout.close();
  cout << gm4.last_ntrial << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],1.0,1.0e-4,"4a");
  t.test_rel(x[1],0.0,1.0e-4,"4b");
  t.test_rel(x[2],0.0,1.0e-4,"4c");

  t.report();
  return 0;
}
// End of example
```

Idea for Future Plot the spring function

19.2 Minimizer fixing variables

This example uses the multi_min_fix class to minimize the function

$$y = (x_0 - 2)^2 + (x_1 - 1)^2 + x_2^2$$

while fixing some of the parameters.

```
/* Example: ex_mmin_fix.cpp
   -------------------------------------------------------------------
   Example usage of the mmin_fix class, which fixes some of the
   paramters for a multidimensional minimization.
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/multi_funct.h>
#include <o2scl/gsl_mmin_simp2.h>
#include <o2scl/multi_min_fix.h>

using namespace std;
using namespace o2scl;

class cl {

public:

  double mfn(size_t nv, const ovector_base &x) {
    return (x[0]-2.0)*(x[0]-2.0)+(x[1]-1.0)*(x[1]-1.0)+x[2]*x[2];
  }

};

int main(void) {
  cl acl;
  ovector x(3);
  double fmin;
  test_mgr t;

  t.set_output_level(1);
  cout.setf(ios::scientific);

  /*
    Perform the minimization the standard way, with the
    simplex2 minimizer
  */
  multi_funct_mfptr<cl> f1(&acl,&cl::mfn);
  gsl_mmin_simp2<> gm1;

  x[0]=0.5;
  x[1]=0.5;
  x[2]=0.5;
  gm1.mmin(3,x,fmin,f1);
  cout << gm1.last_ntrial << " iterations." << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],2.0,1.0e-4,"1a");
  t.test_rel(x[1],1.0,1.0e-4,"1b");
  t.test_rel(x[2],0.0,1.0e-4,"1c");

  // Create a new multi_min_fix object
  multi_min_fix<bool[3]> gmf;
```

```
  // Create a base minimizer which can be used by the multi_min_fix
  // object. Note that we can't use 'gm1' here, because it has a
  // different type than 'gm2', even though its functionality is
  // effectively the same.
  gsl_mmin_simp2<multi_funct_mfptr<multi_min_fix<bool[3]> > > gm2;

  // Set the base minimizer
  gmf.set_mmin(gm2);

  /*
    First perform the minimization as above.
  */
  x[0]=0.5;
  x[1]=0.5;
  x[2]=0.5;
  gmf.mmin(3,x,fmin,f1);
  cout << gmf.last_ntrial << " iterations." << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],2.0,1.0e-4,"2a");
  t.test_rel(x[1],1.0,1.0e-4,"2b");
  t.test_rel(x[2],0.0,1.0e-4,"2c");

  /*
    Now fix the 2nd variable, and re-minimize.
  */
  bool fix[3]={false,true,false};
  x[0]=0.5;
  x[1]=0.5;
  x[2]=0.5;
  gmf.mmin_fix(3,x,fmin,fix,f1);
  cout << gmf.last_ntrial << " iterations." << endl;
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],2.0,1.0e-4,"3a");
  t.test_rel(x[1],0.5,1.0e-4,"3b");
  t.test_rel(x[2],0.0,1.0e-4,"3c");

  t.report();
  return 0;
}
// End of example
```

20 Constrained Minimization

O_2scl reimplements the Open Optimization Library (OOL) available at http://ool.sourceforge.net. The associated classes allow constrained minimization when the constraint can be expressed as a hyper-cubic constraint on all of the independent variables. The routines have been rewritten and reformatted for C++ in order to facilitate the use of member functions and user-defined vector types as arguments. The base class is ool_constr_mmin and there are two different constrained minimzation algorithms implemented in ool_mmin_pgrad, ool_mmin_spg. (The ool_mmin_gencan minimizer is not yet finished). The O_2scl implementation should be essentially identical to the most recently released version of OOL.

The constrained minimization classes operate in a similar way to the other multi-dimensional minimization classes (which are derived from multi_min). The constraints are specified with the function

```
ool_constr_mmin::set_constraints(size_t nc, vec_t &lower,
vec_t &upper);
```

and the minimization can be performed by calling either multi_min::mmin() or multi_min::mmin_de() (if the gradient is provided by the user). The method in ool_mmin_gencan requires a Hessian vector product and the user can specify this product for the minimization by using ool_constr_mmin::mmin_hess(). The Hessian product function can be specified as an object of type ool_hfunct in a similar way to the other function objects in O_2scl .

There are five error codes defined in ool_constr_mmin which are specific to the classes derived from OOL.

The class gsl_anneal can handle some kinds of constraints by ignoring proposed steps which cause the user-specified function to

return a non-zero value.

Also, a simple way of implementing constraints is to add a function to the original which increases the value outside of the allowed region. This can be done with the functions constraint() and lower_bound(). There are two analogous functions, cont_constraint() and cont_lower_bound(), which continuous and differentiable versions. Where possible, it is better to use the constrained minimization routines described above.

20.1 Constrained minimization example

This example minimizes the function

$$f(x,y) = \left[x^2 \log(x) + 1\right] \left[\sqrt{y}(y-1) + 1)\right]$$

which is undefined for $x < 0$ and $y < 0$. The function is also minimized by gsl_mmin_simp2, which goes outside the allowed region where the function is undefined.

```
/* Example: ex_conmin.cpp
   ----------------------------------------------------------------
   This gives an example of the use of a constrained minimizer. This
   code finds the global minimum of a two-dimensional function which
   is not well-defined outside the region of interest.
*/
#include <gsl/gsl_math.h>
#include <gsl/gsl_blas.h>
#include <o2scl/test_mgr.h>
#include <o2scl/ool_mmin_spg.h>
#include <o2scl/gsl_mmin_simp2.h>

using namespace std;
using namespace o2scl;

double func(size_t nv, const ovector_base &x) {
  if (x[0]<0.0 || x[1]<0.0 || x[0]>100.0 || x[1]>100.0) {
    cout << "Outside constraint region." << endl;
  }
  double ret=(log(x[0])*x[0]*x[0]+1.0)*(sqrt(x[1])*(x[1]-1.0)+1.0);
  return ret;
}

int dfunc(size_t nv, ovector_base &x, ovector_base &g) {
  g[0]=(x[0]+2.0*x[0]*log(x[0]))*(sqrt(x[1])*(x[1]-1.0)+1.0);
  g[1]=(log(x[0])*x[0]*x[0]+1.0)*(sqrt(x[1])+(x[1]-1.0)/2.0/sqrt(x[1]));
  return 0;
}

int main(void) {
  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);

  static const size_t nv=2;

  // Specify the function to minimize and its gradient
  multi_funct_fptr<> mff(func);
  grad_funct_fptr<> gff(dfunc);

  // The unconstrained minimizer
  gsl_mmin_simp2<> gm1;
  // The constrained minimizer
  ool_mmin_spg<> omp;

  // The constraints and the location of the minimum
  ovector c1(nv), c2(nv), x(nv);
  double fmin;

  cout << "Simple minimizer: " << endl;
```

```
  // Initial guess
  for(size_t i=0;i<nv;i++) {
    x[i]=10.0;
  }

  // Minimize
  gm1.mmin(nv,x,fmin,mff);
  cout << endl;

  cout << "Constrained minimizer: " << endl;

  // Initial guess
  for(size_t i=0;i<nv;i++) {
    x[i]=10.0;
  }

  // Set constraints
  c1.set_all(1.0e-9);
  c2.set_all(100.0);
  omp.set_constraints(nv,c1,c2);

  // Minimize
  omp.mmin_de(nv,x,fmin,mff,gff);

  // Output results
  cout << x[0] << " " << x[1] << " " << fmin << endl;

  // Test the constrained minimizer results
  t.test_rel(x[0],0.60655,1.0e-4,"x0");
  t.test_rel(x[1],1.0/3.0,1.0e-4,"x1");

  t.report();
  return 0;
}
// End of example
```

21 Monte Carlo Integration

Monte Carlo integration is performed by descendants of mcarlo_inte (gsl_monte, gsl_miser, and gsl_vegas). These routines are generally superior to the direct methods for integrals over regions with large numbers of spatial dimensions.

21.1 Monte Carlo Integration Example

This example computes the integral

$$\int_0^{\pi}\int_0^{\pi}\int_0^{\pi}\frac{1}{\pi^3}\left(1-\cos x\cos y\cos z\right)^{-1}$$

and compares the result with the exact value, 1.3932039296.

```
/* Example: ex_minte.cpp
   -------------------------------------------------------------------
   An example to demonstrate multidimensional integration
*/

#include <o2scl/test_mgr.h>
#include <o2scl/multi_funct.h>
#include <o2scl/composite_inte.h>
#include <o2scl/gsl_inte_qng.h>
#include <o2scl/gsl_vegas.h>

/// For M_PI
#include <gsl/gsl_math.h>
```

```
using namespace std;
using namespace o2scl;

double test_fun(size_t nv, const ovector_base &x) {
  double y=1.0/(1.0-cos(x[0])*cos(x[1])*cos(x[2]))/M_PI/M_PI/M_PI;
  return y;
}

int main(void) {
  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);

  double exact=1.3932039296;
  double res;

  double err;

  gsl_vegas<multi_funct<> > gm;
  ovector a(3), b(3);
  a.set_all(0.0);
  b.set_all(M_PI);

  multi_funct_fptr<> tf(test_fun);

  gm.n_points=100000;
  gm.minteg_err(tf,3,a,b,res,err);

  cout << res << " " << exact << " " << (res-exact)/err << endl;
  t.test_rel(res,exact,err*10.0,"O2scl");

  t.report();

  return 0;
}
// End of example
```

22 Simulated Annealing

Minimization by simulated annealing is performed by descendants of sim_anneal (see gsl_anneal). Because simulated annealing is particularly well-suited to parallelization, a multi-threaded minimizer analogous to gsl_anneal is given in anneal_mt. This header-only class uses the Boost libraries and requires it for use.

22.1 Simulated annealing

This example minimizes the function

$$f(x,y) = J_0(x-2)J_0(y+3)$$

over (x,y) where $J_0(x)$ is the Bessel function given in `gsl_sf_bessel_J0`. The initial guess at $(9,9)$ is several local minima away from the global minimum.

The plot below plots the function above, the initial guess, and the minimum obtained by the example program.

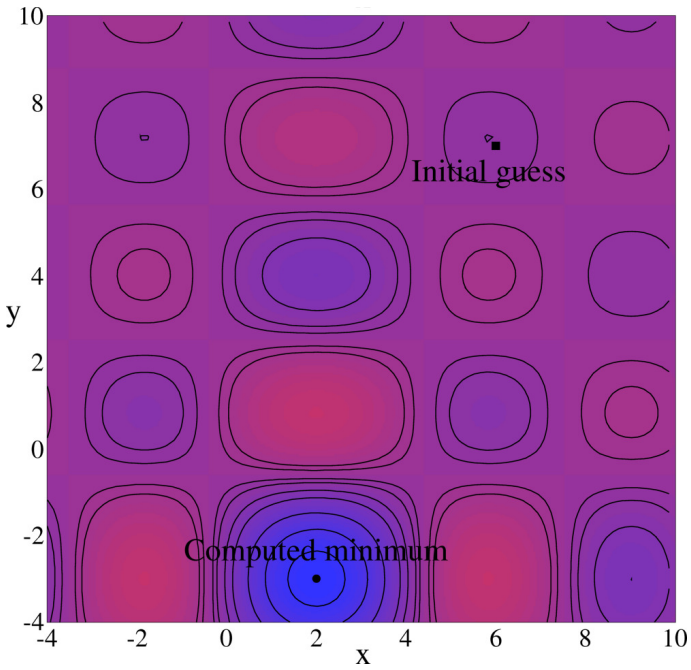


Figure 3: Simulated annealing example plot

```
/* Example: ex_anneal.cpp
   -------------------------------------------------------------------
   An example to demonstrate minimization by simulated annealing
*/

#include <iostream>
#include <cmath>
#include <gsl/gsl_sf_bessel.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/multi_funct.h>
#include <o2scl/funct.h>
#include <o2scl/gsl_anneal.h>
#include <o2scl/test_mgr.h>

using namespace std;
using namespace o2scl;

// A simple function with many local minima. A "greedy" minimizer
// would likely fail to find the correct minimum.
double function(size_t nvar, const ovector_base &x) {

  double a, b;
  a=(x[0]-2.0);
  b=(x[1]+3.0);

  // This is important to prevent the annealing algorithm to
  // random walk to infinity since the product of Bessel
  // functions is flat far from the origin
  if (fabs(x[0])>10.0 || fabs(x[1])>10.0) return 10.0;

  return -gsl_sf_bessel_J0(a)*gsl_sf_bessel_J0(b);
}

int main(int argc, char *argv[]) {
  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);
```

```
  gsl_anneal<> ga;
  double result;
  ovector init(2);

  multi_funct_fptr<> fx(function);

  ga.ntrial=4000;
  ga.verbose=1;
  ga.tol_abs=1.0e-7;
  ga.T_dec=1.1;

  // Set a large initial step size
  double step[1]={10.0};
  ga.set_step(1,step);

  // Choose an initial point at a local minimum away from
  // the global minimum
  init[0]=6.0;
  init[1]=7.0;

  // Perform the minimization
  ga.mmin(2,init,result,fx);
  cout << "x: " << init[0] << " " << init[1]
       << ", minimum function value: " << result << endl;
  cout << endl;

  // Test that it found the global minimum
  t.test_rel(init[0],2.0,1.0e-3,"another test - value");
  t.test_rel(init[1],-3.0,1.0e-3,"another test - value 2");
  t.test_rel(result,-1.0,1.0e-3,"another test - min");

  t.report();

  return 0;
}
// End of example
```

23 Non-linear Least-Squares Fitting

Fitting is performed by descendants of fit_base and fitting functions can be specifed using fit_funct. The GSL fitting routines (scaled and unscaled) are implemented in gsl_fit. A generic fitting routine using a minimizer object specified as a child of multi_min is implemented in min_fit. When the multi_min object is (for example) a sim_anneal object, min_fit can avoid local minima which can occur when fitting noisy data.

24 Ordinary Differential Equations

Classes for non-adaptive integration are provided as descendents of ode_step and classes for adaptive integration are descendants of adapt_step. To specify a set of functions to these classes, use a child of ode_funct for a generic vector type. The classes gsl_rkf45 and gsl_rkck are reasonable general-purpose non-adaptive integrators and gsl_astep is a good general-purpose adaptive method for non-stiff problems. For stiff ODE's, use gsl_bsimp (see the Stiff differential equations example).

Solution of simple initial value problems is performed by ode_iv_solve. This class uses an adaptive integrator (default is gsl_astep) and does some bookkeeping to simplify the solution of initial value problems. The ode_iv_solve class will give you only the final value of the functions at the endpoint, will provide the functions on a user-specified grid, or will tabulate the ODE solution for you using the grid chosen with the adaptive stepper. A example demonstrating the solution of initial value problems is given in the Ordinary differential equations example.

The solution of boundary-value problems is based on the abstract base class ode_bv_solve. At the moment, a simple shooting method is the only implementation of this base class and is given in ode_bv_shoot . An experimental multishooting class is given in ode_bv_multishoot .

An application of linear solvers to solve finite-difference equations approximating a boundary value problem is given in ode_it_solve . A example demonstrating the iterative solution of a boundary value problem is given in the Iterative solution of ODEs example.

24.1 Ordinary differential equations

This example solves the differential equations defining the the Bessel and Airy functions with both the Cash-Karp and Prince--Dormand steppers. It demonstrates the use of gsl_rkck, gsl_rk8pd, gsl_astep, and ode_iv_solve.

The Bessel functions are defined by

$$y'' = \frac{1}{x^2}\left[y\left(\alpha^2 - x^2\right) - xy'\right]$$

The Bessel functions of the first kind, $J_\alpha(x)$ are finite at the origin, and the example solves the $\alpha = 1$ case, where $J_1(0) = 0$ and $J_1'(0) = 1/2$.

```
/* Example: ex_ode.cpp
   -------------------------------------------------------------------
   An example to demonstrate solving differential equations
*/

#include <gsl/gsl_sf_bessel.h>
#include <gsl/gsl_sf_airy.h>
#include <gsl/gsl_sf_gamma.h>
#include <o2scl/test_mgr.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/ode_funct.h>
#include <o2scl/gsl_rkck.h>
#include <o2scl/gsl_rk8pd.h>
#include <o2scl/gsl_astep.h>
#include <o2scl/ode_iv_solve.h>
#include <o2scl/table.h>
#ifdef O2SCL_HDF_IN_EXAMPLES
#include <o2scl/hdf_file.h>
#include <o2scl/hdf_io.h>
#endif

using namespace std;
using namespace o2scl;
#ifdef O2SCL_HDF_IN_EXAMPLES
using namespace o2scl_hdf;
#endif

// Differential equation defining the Bessel function. This assumes
// the second derivative at x=0 is 0 and thus only works for odd alpha.
int derivs(double x, size_t nv, const ovector_base &y,
           ovector_base &dydx, double &alpha) {
  dydx[0]=y[1];
  if (x==0.0) dydx[1]=0.0;
  else dydx[1]=(-x*y[1]+(-x*x+alpha*alpha)*y[0])/x/x;
  return 0;
}

// Differential equation defining the Airy function, Ai(x)
int derivs2(double x, size_t nv, const ovector_base &y,
            ovector_base &dydx, double &alpha) {
  dydx[0]=y[1];
  dydx[1]=y[0]*x;
  return 0;
}

int main(void) {

  cout.setf(ios::scientific);
  cout.setf(ios::showpos);

  // The independent variable and stepsize
  double x, dx=1.0e-1;
```

```
// The function and derivative values and the estimated errors
ovector y(2), dydx(2), yout(2), yerr(2), dydx_out(2);

test_mgr t;
t.set_output_level(1);

// The parameter for the Bessel function
double alpha=1.0;

// Specify the differential equations to solve
ode_funct_fptr_param<double,ovector_base> od(derivs,alpha);
ode_funct_fptr_param<double,ovector_base> od2(derivs2,alpha);

// The basic ODE steppers
gsl_rkck<> ode;
gsl_rk8pd<> ode2;

// ------------------------------------------------------------
// Store the results in tables

table tab[8];
tab[0].line_of_names("x calc exact diff err");
tab[1].line_of_names("x calc exact diff err");
tab[2].line_of_names("x calc exact diff err");
tab[3].line_of_names("x calc exact diff err");
tab[4].line_of_names("x calc exact diff err0 err1");
tab[5].line_of_names("x calc exact diff err0 err1");
tab[6].line_of_names("x calc exact diff err0 err1");
tab[7].line_of_names("x calc exact diff err0");

// ------------------------------------------------------------
// Solution 1: Solve using the non-adaptive Cash-Karp stepper.

cout << "Bessel function, Cash-Karp: " << endl;

// Initial values at x=0
x=0.0;
y[0]=0.0;
y[1]=0.5;

// The non-adaptive ODE steppers require the derivatives as
// input
derivs(x,2,y,dydx,alpha);

cout << " x             J1(calc)      J1(exact)     rel. diff.    "
     << "err" << endl;

while (x<1.0) {

  // Perform a step. Since the fourth and sixth arguments are
  // the same, the values in 'y' are updated with the new values
  // at x+dx.
  ode.step(x,dx,2,y,dydx,y,yerr,dydx,od);

  // Update the x value
  x+=dx;

  // Print and test
  cout << x << " " << y[0] << " "
       << gsl_sf_bessel_J1(x) << " ";
  cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
  cout << yerr[0] << endl;
  t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-5,"rkck");

  // Also output the results to a table
  double line[5]={x,y[0],gsl_sf_bessel_J1(x),
                  fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)),
                  yerr[0]};
  tab[0].line_of_data(5,line);
```

```
}

// Compare with the exact result at the last point
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// End of Solution 1
// ------------------------------------------------------------
// Solution 2: Solve using the non-adaptive Prince-Dormand stepper.
// Note that for the Bessel function, the 8th order stepper performs
// worse than the 4th order. The error returned by the stepper is
// larger near x=0, as expected.

cout << "Bessel function, Prince-Dormand: " << endl;
x=0.0;
y[0]=0.0;
y[1]=0.5;
derivs(x,2,y,dydx,alpha);
cout << " x             J1(calc)      J1(exact)     rel. diff.    "
     << "err" << endl;
while (x<1.0) {
  ode2.step(x,dx,2,y,dydx,y,yerr,dydx,od);
  x+=dx;
  cout << x << " " << y[0] << " "
       << gsl_sf_bessel_J1(x) << " ";
  cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
  cout << yerr[0] << endl;
  t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-4,"rk8pd");

  // Also output the results to a table
  double line[5]={x,y[0],gsl_sf_bessel_J1(x),
                  fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)),
                  yerr[0]};
  tab[1].line_of_data(5,line);
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// End of Solution 2
```

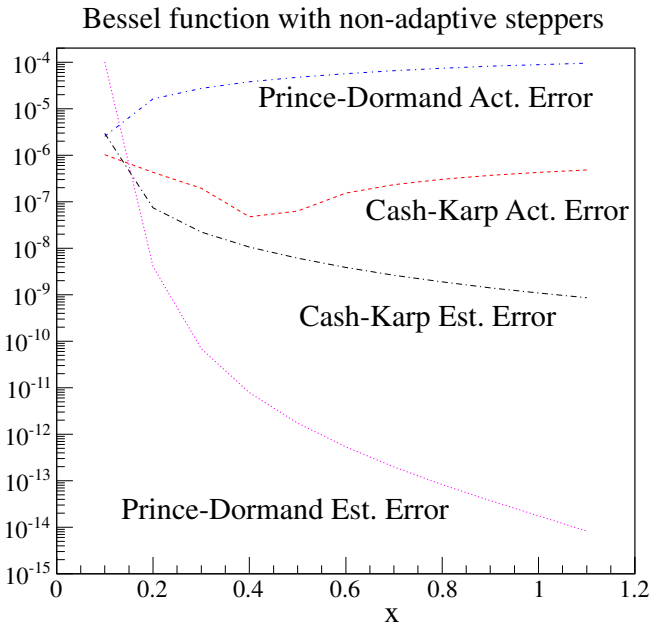


Figure 4: Bessel example plot

Note that with a Bessel function and a fixed step size, the Prince-Dormand stepper (even though of higher order than the Cash-Karp stepper) is actually less accurate, and seriously underestimates the error.

The Airy functions are defined by

$$y'' = yx$$

This example solves for the Airy function of the first kind, $Ai(x)$.

```
// Solution 3: Solve using the non-adaptive Cash-Karp stepper.

cout << "Airy function, Cash-Karp: " << endl;
x=0.0;
y[0]=1.0/pow(3.0,2.0/3.0)/gsl_sf_gamma(2.0/3.0);
y[1]=-1.0/pow(3.0,1.0/3.0)/gsl_sf_gamma(1.0/3.0);
derivs2(x,2,y,dydx,alpha);
cout << " x             Ai(calc)      Ai(exact)     rel. diff.    "
     << "err" << endl;
while (x<1.0) {
  ode.step(x,dx,2,y,dydx,y,yerr,dydx,od2);
  x+=dx;
  cout << x << " " << y[0] << " "
       << gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << " ";
  cout << fabs((y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
               gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE)) << " ";
  cout << yerr[0] << endl;
  t.test_rel(y[0],gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE),1.0e-8,"rkck");

  // Also output the results to a table
  double line[5]={x,y[0],gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE),
                  fabs((y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
                       gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE)),
                  yerr[0]};
  tab[2].line_of_data(5,line);
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
  gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << endl;
cout << endl;
```

```
// End of Solution 3
// ------------------------------------------------------------
// Solution 4: Solve using the non-adaptive Prince-Dormand stepper.
// On this function, the higher-order routine performs significantly
// better.

cout << "Airy function, Prince-Dormand: " << endl;
x=0.0;
y[0]=1.0/pow(3.0,2.0/3.0)/gsl_sf_gamma(2.0/3.0);
y[1]=-1.0/pow(3.0,1.0/3.0)/gsl_sf_gamma(1.0/3.0);
derivs2(x,2,y,dydx,alpha);
cout << " x              Ai(calc)      Ai(exact)     rel. diff.    "
     << "err" << endl;
while (x<1.0) {
  ode2.step(x,dx,2,y,dydx,y,yerr,dydx,od2);
  x+=dx;
  cout << x << " " << y[0] << " "
       << gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << " ";
  cout << fabs((y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
               gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE)) << " ";
  cout << yerr[0] << endl;
  t.test_rel(y[0],gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE),1.0e-14,"rk8pd");

  // Also output the results to a table
  double line[5]={x,y[0],gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE),
                  fabs((y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
                       gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE)),
                  yerr[0]};
  tab[3].line_of_data(5,line);
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE))/
  gsl_sf_airy_Ai(x,GSL_PREC_DOUBLE) << endl;
cout << endl;

// End of Solution 4
```

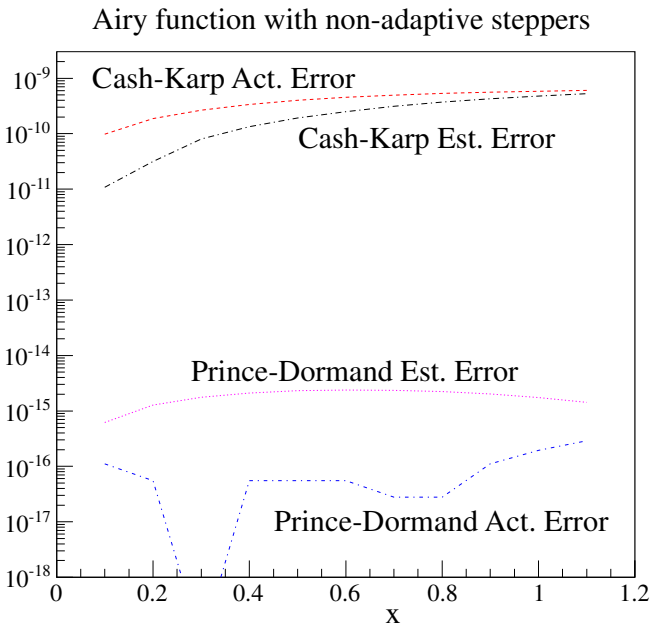


Figure 5: Airy example plot

Here the higher order stepper is more accurate.

```
// Solution 5: Solve using the GSL adaptive stepper

// Lower the output precision to fit in 80 columns
cout.precision(5);

cout << "Adaptive stepper: " << endl;
gsl_astep<> ode3;
x=0.0;
y[0]=0.0;
y[1]=0.5;
cout << "   x            J1(calc)     J1(exact)    rel. diff.";
cout << "   err_0        err_1" << endl;
int k=0;
while (x<10.0) {
  int retx=ode3.astep(x,10.0,dx,2,y,dydx,yerr,od);
  if (k%3==0) {
    cout << retx << " " << x << " " << y[0] << " "
         << gsl_sf_bessel_J1(x) << " ";
    cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
    cout << yerr[0] << " " << yerr[1] << endl;
  }
  t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-3,"astep");
  t.test_rel(y[1],0.5*(gsl_sf_bessel_J0(x)-gsl_sf_bessel_Jn(2,x)),
             5.0e-3,"astep 2");
  t.test_rel(yerr[0],0.0,4.0e-6,"astep 3");
  t.test_rel(yerr[1],0.0,4.0e-6,"astep 4");
  t.test_gen(retx==0,"astep 5");

  // Also output the results to a table
  double line[6]={x,y[0],gsl_sf_bessel_J1(x),
                  fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)),
                  yerr[0],yerr[1]};
  tab[4].line_of_data(6,line);

  k++;
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// End of Solution 5
// ------------------------------------------------------------
// Solution 6: Solve using the GSL adaptive stepper.
// Decrease the tolerances, and the adaptive stepper takes
// smaller step sizes.

cout << "Adaptive stepper with smaller tolerances: " << endl;
ode3.con.eps_abs=1.0e-8;
ode3.con.a_dydt=1.0;
x=0.0;
y[0]=0.0;
y[1]=0.5;
cout << "   x            J1(calc)     J1(exact)    rel. diff.";
cout << "   err_0        err_1" << endl;
k=0;
while (x<10.0) {
  int retx=ode3.astep(x,10.0,dx,2,y,dydx,yerr,od);
  if (k%3==0) {
    cout << retx << " " << x << " " << y[0] << " "
         << gsl_sf_bessel_J1(x) << " ";
    cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
    cout << yerr[0] << " " << yerr[1] << endl;
  }
  t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-3,"astep");
  t.test_rel(y[1],0.5*(gsl_sf_bessel_J0(x)-gsl_sf_bessel_Jn(2,x)),
             5.0e-3,"astep 2");
  t.test_rel(yerr[0],0.0,4.0e-8,"astep 3");
  t.test_rel(yerr[1],0.0,4.0e-8,"astep 4");
```

```
  t.test_gen(retx==0,"astep 5");

  // Also output the results to a table
  double line[6]={x,y[0],gsl_sf_bessel_J1(x),
                  fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)),
                  yerr[0],yerr[1]};
  tab[5].line_of_data(6,line);

  k++;
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// End of Solution 6
// ------------------------------------------------------------
// Solution 7: Solve using the GSL adaptive stepper.
// Use the higher-order stepper, and less steps are required. The
// stepper automatically takes more steps near x=0 in order since
// the higher-order routine has more trouble there.

cout << "Adaptive stepper, Prince-Dormand: " << endl;
ode3.set_step(ode2);
x=0.0;
y[0]=0.0;
y[1]=0.5;
cout << "   x            J1(calc)     J1(exact)    rel. diff.";
cout << "   err_0        err_1" << endl;
k=0;
while (x<10.0) {
  int retx=ode3.astep(x,10.0,dx,2,y,dydx,yerr,od);
  if (k%3==0) {
    cout << retx << " " << x << " " << y[0] << " "
         << gsl_sf_bessel_J1(x) << " ";
    cout << fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)) << " ";
    cout << yerr[0] << " " << yerr[1] << endl;
  }
  t.test_rel(y[0],gsl_sf_bessel_J1(x),5.0e-3,"astep");
  t.test_rel(y[1],0.5*(gsl_sf_bessel_J0(x)-gsl_sf_bessel_Jn(2,x)),
             5.0e-3,"astep");
  t.test_rel(yerr[0],0.0,4.0e-8,"astep 3");
  t.test_rel(yerr[1],0.0,4.0e-8,"astep 4");
  t.test_gen(retx==0,"astep 5");

  // Also output the results to a table
  double line[6]={x,y[0],gsl_sf_bessel_J1(x),
                  fabs((y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x)),
                  yerr[0],yerr[1]};
  tab[6].line_of_data(6,line);

  k++;
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// End of Solution 7
```

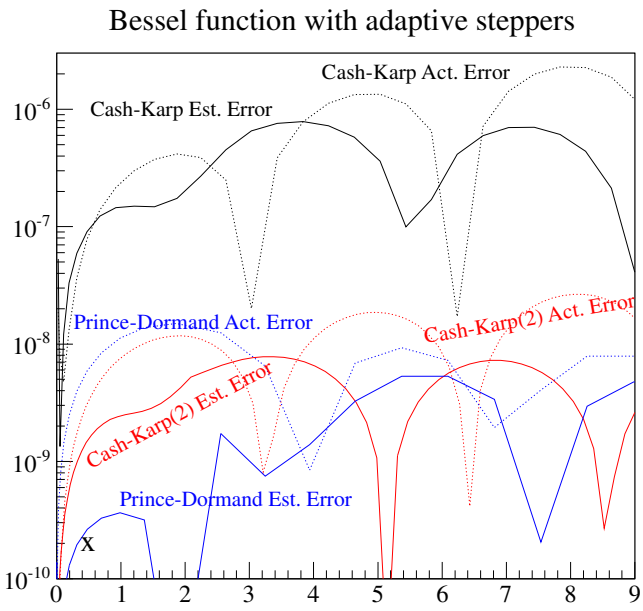


Figure 6: Bessel example plot

```
// Solution 8: Solve using the O2scl initial value solver

// Return the output precision to the default
cout.precision(6);

cout << "Initial value solver: " << endl;
ode_iv_solve<> ode4;

// Define the grid and the storage for the solution
const size_t ngrid=101;
ovector xg(ngrid), yinit(2);
omatrix yg(ngrid,2), ypg(ngrid,2), yerrg(ngrid,2);
for(size_t i=0;i<ngrid;i++) xg[i]=((double)i)/10.0;

// Set the initial value
xg[0]=0.0;
xg[ngrid-1]=10.0;
yg[0][0]=0.0;
yg[0][1]=0.5;

// Perform the full solution
ode4.solve_grid<omatrix,omatrix_row>(0.1,2,ngrid,xg,yg,ypg,yerrg,od);

// Output and test the results
cout << " x             J1(calc)      J1(exact)     rel. diff." << endl;
for(size_t i=1;i<ngrid;i++) {

  if (i%10==0) {
    cout << xg[i] << " " << yg[i][0] << " "
         << gsl_sf_bessel_J1(xg[i]) << " ";
    cout << fabs(yg[i][0]-gsl_sf_bessel_J1(xg[i])) << " "
         << fabs(yerrg[i][0]) << endl;
  }
  t.test_rel(yg[i][0],gsl_sf_bessel_J1(xg[i]),5.0e-7,"astep");
  t.test_rel(yg[i][1],0.5*(gsl_sf_bessel_J0(xg[i])-
                           gsl_sf_bessel_Jn(2,xg[i])),5.0e-7,"astep 2");

  // Also output the results to a table
  double line[5]={xg[i],yg[i][0],gsl_sf_bessel_J1(xg[i]),
```

```
                    fabs(yg[i][0]-gsl_sf_bessel_J1(xg[i])),
                    fabs(yerrg[i][0])};
    tab[7].line_of_data(5,line);
  }

  cout << "Accuracy at end: "
       << fabs(yg[ngrid-1][0]-gsl_sf_bessel_J1(xg[ngrid-1]))/
    gsl_sf_bessel_J1(xg[ngrid-1]) << endl;
  cout << endl;

  // End of Solution 8
  // -------------------------------------------------------------
  // Output results to a file

#ifdef O2SCL_HDF_IN_EXAMPLES
  hdf_file hf;
  hf.open("ex_ode.o2");
  for(size_t i=0;i<8;i++) {
    hdf_output(hf,tab[i],((string)"table_")+itos(i));
  }
  hf.close();
#endif

  // -------------------------------------------------------------

  cout.unsetf(ios::showpos);
  t.report();

  return 0;
}
// End of example
```

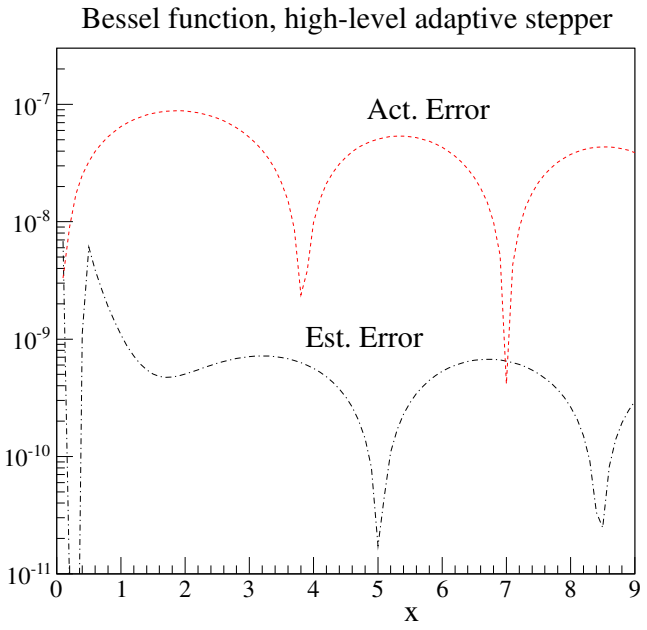


Figure 7: Bessel example plot

24.2 Stiff differential equations

This example solves the differential equations

$$\begin{aligned} y_0' &= 480 y_0 + 980 y_1 \\ y_1' &= -490 y_0 - 990 y_1 \end{aligned}$$

which have the exact solution

$$\begin{aligned} y_0 &= -e^{-500x} + 2e^{-10x} \\ y_1 &= e^{-500x} - e^{-10x} \end{aligned}$$

using both the stiff stepper gsl_bsimp and the standard adaptive stepper gsl_astep . The relative error on the adaptive stepper is orders of magnitude larger.

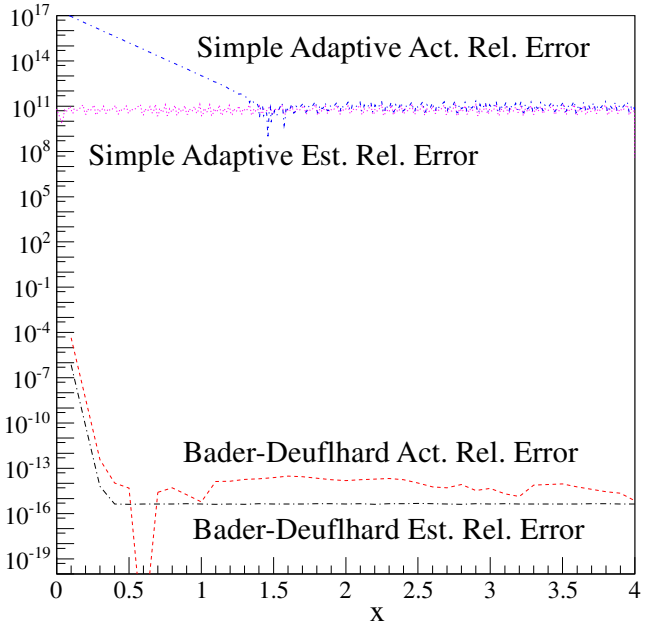


Figure 8: Bader-Deuflhard example plot

```
/* Example: ex_stiff.cpp
   -------------------------------------------------------------------
   An example to demonstrate solving stiff differential equations
*/

#include <o2scl/test_mgr.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/funct.h>
#include <o2scl/ode_funct.h>
#include <o2scl/gsl_astep.h>
#include <o2scl/gsl_bsimp.h>
#include <o2scl/table.h>

#ifdef O2SCL_HDF_IN_EXAMPLES
#include <o2scl/hdf_file.h>
#include <o2scl/hdf_io.h>
#endif

using namespace std;
using namespace o2scl;
```

```
#ifdef O2SCL_HDF_IN_EXAMPLES
using namespace o2scl_hdf;
#endif

int derivs(double x, size_t nv, const ovector_base &y,
           ovector_base &dydx) {
  dydx[0]=480.0*y[0]+980.0*y[1];
  dydx[1]=-490.0*y[0]-990.0*y[1];
  return 0;
}

int jac(double x, size_t nv, const ovector_base &y,
        omatrix_base &dfdy, ovector_base &dfdx) {
  dfdy[0][0]=480.0;
  dfdy[0][1]=980.0;
  dfdy[1][0]=-490.0;
  dfdy[1][1]=-990.0;
  dfdx[0]=0.0;
  dfdx[1]=0.0;
  return 0;
}

int main(void) {
  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);
  cout.precision(3);

  // Specification of the differential equations and the Jacobian
  ode_funct_fptr<ovector_base> od(derivs);
  ode_jac_funct_fptr<ovector_base> oj(jac);

  table tab[2];
  tab[0].line_of_names("x calc exact rel_err rel_diff");
  tab[1].line_of_names("x calc exact rel_err rel_diff");

  // ------------------------------------------------------------
  // First solve with gsl_bsimp, designed to handle stiff ODEs

  gsl_bsimp<ode_funct_fptr<>,ode_jac_funct_fptr<> > gb;

  double x1, dx=1.0e-1;
  ovector y1(2), dydx1(2), yout1(2), yerr1(2), dydx_out1(2);

  x1=0.0;
  y1[0]=1.0;
  y1[1]=0.0;

  derivs(x1,2,y1,dydx1);

  for(size_t i=1;i<=40;i++) {

    gb.step(x1,dx,2,y1,dydx1,y1,yerr1,dydx1,od,oj);
    x1+=dx;

    double exact[2]={-exp(-500.0*x1)+2.0*exp(-10.0*x1),
                     exp(-500.0*x1)-exp(-10.0*x1)};

    cout.setf(ios::showpos);
    cout << x1 << " " << y1 << " " << yerr1 << " " << exact[0] << " "
         << exact[1] << endl;
    cout.unsetf(ios::showpos);

    double line[5]={x1,y1[0],exact[0],yerr1[0]/exact[0],
                    fabs((y1[0]-exact[0])/exact[0])};
    tab[0].line_of_data(5,line);

    t.test_rel(y1[0],exact[0],3.0e-4,"y0");
    t.test_rel(y1[1],exact[1],3.0e-4,"y1");
```

```
  }

  cout << endl;

  // ----------------------------------------------------------
  // Now compare to the traditional adaptive stepper

  gsl_astep<ode_funct_fptr<> > ga;

  double x2;
  ovector y2(2), dydx2(2), yout2(2), yerr2(2), dydx_out2(2);

  x2=0.0;
  y2[0]=1.0;
  y2[1]=0.0;

  derivs(x2,2,y2,dydx2);

  size_t j=0;
  while (x2<4.0) {

    ga.astep(x2,4.0,dx,2,y2,dydx2,yerr2,od);

    double exact[2]={-exp(-500.0*x1)+2.0*exp(-10.0*x1),
                     exp(-500.0*x1)-exp(-10.0*x1)};

    if (j%25==0) {
      cout.setf(ios::showpos);
      cout << x2 << " " << y2 << " " << yerr2 << " "
           << exact[0] << " " << exact[1] << endl;
      cout.unsetf(ios::showpos);
    }
    j++;

    double line[5]={x2,y2[0],exact[0],yerr2[0]/exact[0],
                    fabs((y2[0]-exact[0])/exact[0])};
    tab[1].line_of_data(5,line);

  }

  cout << endl;

 // ----------------------------------------------------------
  // Output results to a file

#ifdef O2SCL_HDF_IN_EXAMPLES
  hdf_file hf;
  hf.open("ex_stiff.o2");
  for(size_t i=0;i<2;i++) {
    hdf_output(hf,tab[i],((string)"table_")+itos(i));
  }
  hf.close();
#endif

  // ----------------------------------------------------------

  t.report();

  return 0;

}
// End of example
```

24.3 Iterative solution of ODEs

This example solves the ODEs

$$\begin{aligned} y_0'' &= y_1 \\ y_1'' &= y_0 + y_1 \\ y_2'' &= y_0 + y_2 \end{aligned}$$

given the boundary conditions

$$\begin{aligned} y_0(x=0) &= 1 \\ y_1(x=0)^2 + y_2(x=0)^2 &= 2 \\ y_1(x=1) &= 3 \end{aligned}$$

by linearizing the ODEs on a mesh and using an iterative method (sometimes called relaxation). The ode_it_solve class demonstrates how this works, but is slow for large grid sizes because the matrix is very sparse. See O$_2$scl_iml which shows how a similar problem can be done more efficiently using the native iterative method and sparse matrix format.

```
/* Example: ex_ode_it.cpp
   -------------------------------------------------------------------
   Demonstrate the iterative method for solving ODEs
*/

#include <o2scl/ode_it_solve.h>
#include <o2scl/ode_iv_solve.h>
#include <o2scl/linear_solver.h>

using namespace std;
using namespace o2scl;
using namespace o2scl_linalg;

// The three-dimensional ODE system
class ode_system {

public:

  // The LHS boundary conditions
  double left(size_t ieq, double x, ovector_base &yleft) {
    if (ieq==0) return yleft[0]-1.0;
    return yleft[1]*yleft[1]+yleft[2]*yleft[2]-2.0;
  }

  // The RHS boundary conditions
  double right(size_t ieq, double x, ovector_base &yright) {
    return yright[1]-3.0;
  }

  // The differential equations
  double derivs(size_t ieq, double x, ovector_base &y) {
    if (ieq==1) return y[0]+y[1];
    else if (ieq==2) return y[0]+y[2];
    return y[1];
  }

  // This is the alternative specification for ode_iv_solve for
  // comparison
  int shoot_derivs(double x, size_t nv, const ovector_base &y,
                   ovector_base &dydx) {
    dydx[0]=y[1];
    dydx[1]=y[0]+y[1];
    dydx[2]=y[0]+y[2];
    return 0;
  }

};
```

```
int main(void) {

  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);

  // The ODE solver
  ode_it_solve<ode_it_funct<ovector_base>,ovector_base,
    omatrix_base,omatrix_row,ovector_base,omatrix_base> oit;

  // The class which contains the functions to solve
  ode_system os;

  // Make function objects for the derivatives and boundary conditions
  ode_it_funct_mfptr<ode_system> f_d(&os,&ode_system::derivs);
  ode_it_funct_mfptr<ode_system> f_l(&os,&ode_system::left);
  ode_it_funct_mfptr<ode_system> f_r(&os,&ode_system::right);

  // Grid size
  size_t ngrid=40;

  // Number of ODEs
  size_t neq=3;

  // Number of LHS boundary conditions
  size_t nbleft=2;

  // Create space for the solution and make an initial guess
  ovector x(ngrid);
  omatrix y(ngrid,neq);
  for(size_t i=0;i<ngrid;i++) {
    x[i]=((double)i)/((double)(ngrid-1));
    y[i][0]=1.0+x[i]+1.0;
    y[i][1]=3.0*x[i];
    y[i][2]=-0.1*x[i]-1.4;
  }

  int pa=0;

  // Workspace objects
  omatrix A(ngrid*neq,ngrid*neq);
  ovector rhs(ngrid*neq), dy(ngrid*neq);

  // Perform the solution
  oit.verbose=1;
  oit.solve(ngrid,neq,nbleft,x,y,f_d,f_l,f_r,A,rhs,dy);

  // Compare with the initial value solver ode_iv_solve
  ode_iv_solve<> ois;
  ode_funct_mfptr<ode_system> f_sd(&os,&ode_system::shoot_derivs);
  ovector ystart(neq), yend(neq);
  for(size_t i=0;i<neq;i++) ystart[i]=y[0][i];
  ois.solve_final_value(0.0,1.0,0.01,neq,ystart,yend,f_sd);

  // Test the result
  t.test_rel(y[0][0],1.0,1.0e-3,"ya");
  t.test_rel(y[ngrid-1][0],yend[0],1.0e-3,"yb");
  t.test_rel(y[0][1],0.25951,1.0e-3,"yc");
  t.test_rel(y[ngrid-1][1],yend[1],1.0e-3,"yd");
  t.test_rel(y[0][2],-1.3902,1.0e-3,"ye");
  t.test_rel(y[ngrid-1][2],yend[2],1.0e-3,"yf");

  t.report();

  return 0;
}
// End of example
```

25 Random Number Generation

Random number generators are descendants of rnga . While the base object rnga is created to allow user-defined random number generators, the only random number generator presently included are from GSL. Because speed is frequently an issue, some design choices were made to ensure fast execution

- The GSL random number generator code is reimplemented, rather than simply wrapped around the GSL code.
- The interface does not use virtual functions.
- Random number generators are implemented as template parameters in sim_anneal and mcarlo_inte, rather than as pointers.

25.1 Generate an arbitrary distribution

This examples creates a Markov chain for an aritrary distribution using the Metropolis-Hastings algorithm. It generates the one-dimensional distribution

$$f(x) = e^{-x^2} \left[\sin(x-1.4)+1\right]$$

for $x \in [-5,5]$.

```
/* Example: ex_markov.cpp
   -------------------------------------------------------------------

   An example to demonstrate generation of an arbitrary distribution
   through the creation of a Markov chain using the Metropolis
   algorithm.
*/

#include <iostream>
#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/hist.h>
#include <o2scl/gsl_rnga.h>
#include <o2scl/expect_val.h>
#include <o2scl/gsl_inte_qag.h>

using namespace std;
using namespace o2scl;

/*
  An unusual two-peaked probability distribution. This distribution is
  is strongly bimodal, and thus the algorithm has to step over a
  region where the probability distribution is small.
*/
double function(double x) {
  return exp(-x*x)*(sin(x-1.4)+1.0);
}

// Same function but now weighted with x
double function2(double x) {
  return x*exp(-x*x)*(sin(x-1.4)+1.0);
}

int main(int argc, char *argv[]) {
  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);

  // The square root of the number of iterations
  static const size_t N=1000;

  // The stepsize for the Metropolis algorithm. If this stepsize was
  // too small, the algorithm would fail to step between the two
  // peaks and give incorrect results.
```

```
double step=1.0;

// The random number generator
gsl_rnga gr;
if (argc>=2) {
  gr.set_seed(stoi(argv[1]));
}

// Construct a histogrmam of the distribution. Initialize the grid
// spacing with a unform grid between -5 and 5.
hist h;
h.set_bins(uniform_grid_end_width<>(-5.0,5.0,0.1));

// Compute the average value of the distribution, and use
// a scalar_ev object to estimate the uncertainty
static const size_t M=20;
scalar_ev sev(M,N*N/M);

// An initial point and the initial weight
double x_old=0.01;
double w_old=function(x_old);

for(size_t k=0;k<N;k++) {
  for(size_t i=0;i<N;i++) {

    // Perform a step and compute the new weight
    double r=gr.random();
    double x_new=x_old+r*step*2.0-step;
    double w_new=function(x_new);

    // The Metrpolis algorithm
    double r2=gr.random();
    if (r2<w_new/w_old && x_new>-5.0 && x_new<5.0) {
      x_old=x_new;
      w_old=w_new;
    }

    // Update the histogram
    h.update(x_old);

    // Update the average value
    sev.add(x_old);

  }
}

// Normalize by the value of the distribution at x=1.05
double norm=h.get_wgt(1.05)/function(1.05);

// Output the generated distribution and compare to the real
// distribution.

for(size_t i=0;i<h.size();i++) {

  double xx=h.get_rep_i(i);

  // Limit output to every third point
  if (i%3==0) {
    cout.width(2);
    cout << i << " ";
    cout.setf(ios::showpos);
    cout << xx << " ";
    cout.unsetf(ios::showpos);
    cout << h.get_wgt_i(i)/norm << " "
         << function(xx) << endl;
  }

  // Only test regions where we'll have enough statistics
  if (function(xx)>5.0e-3) {
    t.test_rel(h.get_wgt_i(i)/norm,function(xx),1.0e-1,"dist");
```

```
    }

  }
  cout << endl;

  // Output the average value and its uncertainty
  double avg, std_dev, avg_err;
  sev.current_avg(avg,std_dev,avg_err);
  cout << avg << " " << avg_err << endl;

  // Compare to result from direct integration
  funct_fptr ff(function);
  funct_fptr ff2(function2);
  gsl_inte_qag<> qag;
  double exact=qag.integ(ff2,-5.0,5.0)/qag.integ(ff,-5.0,5.0);
  cout << exact << endl;

  t.test_abs(avg,exact,3.0*avg_err,"Average value");

  t.report();

  return 0;
}
// End of example
```

Typical output is

```
 0 -4.950000e+00 0.000000e+00 2.131525e-11
 3 -4.650000e+00 0.000000e+00 5.009023e-10
 6 -4.350000e+00 0.000000e+00 9.131547e-09
 9 -4.050000e+00 0.000000e+00 1.309343e-07
12 -3.750000e+00 4.095279e-06 1.488688e-06
15 -3.450000e+00 1.228584e-05 1.348287e-05
18 -3.150000e+00 5.323863e-05 9.747130e-05
21 -2.850000e+00 5.487674e-04 5.624059e-04
24 -2.550000e+00 2.706979e-03 2.584240e-03
27 -2.250000e+00 9.587048e-03 9.410936e-03
30 -1.950000e+00 2.736465e-02 2.693191e-02
33 -1.650000e+00 5.785401e-02 5.970011e-02
36 -1.350000e+00 1.004859e-01 9.993669e-02
39 -1.050000e+00 1.204053e-01 1.202766e-01
42 -7.500000e-01 9.158682e-02 9.293227e-02
45 -4.500000e-01 3.099307e-02 3.162602e-02
48 -1.500000e-01 6.757210e-04 2.114248e-04
51 +1.500000e-01 4.999517e-02 4.988035e-02
54 +4.500000e-01 1.519635e-01 1.523810e-01
57 +7.500000e-01 2.238029e-01 2.249580e-01
60 +1.050000e+00 2.181842e-01 2.181842e-01
63 +1.350000e+00 1.529259e-01 1.535435e-01
66 +1.650000e+00 8.190148e-02 8.196725e-02
69 +1.950000e+00 3.340519e-02 3.397864e-02
72 +2.250000e+00 1.076239e-02 1.108511e-02
75 +2.550000e+00 3.153365e-03 2.868544e-03
78 +2.850000e+00 7.576266e-04 5.914089e-04
81 +3.150000e+00 5.733391e-05 9.733109e-05
84 +3.450000e+00 4.095279e-06 1.278395e-05
87 +3.750000e+00 0.000000e+00 1.336916e-06
90 +4.050000e+00 0.000000e+00 1.107648e-07
93 +4.350000e+00 0.000000e+00 7.207155e-09
96 +4.650000e+00 0.000000e+00 3.628586e-10
99 +4.950000e+00 0.000000e+00 1.376924e-11
46 tests performed.
All tests passed.
```

26 Two-dimensional Interpolation

Successive use of smart_interp is implemented in twod_intp, which is useful for interpolating data presented on a two-dimensional grid (though the spacings between grid points need not be equal). An example demonstrating the use of twod_intp is given in ex_twod_intp_sect. Also, one can compute contour lines from data represented on a grid using the contour class.

If data is arranged without a grid, then planar_intp can be used. At present, the only way to compute contour lines on data which is not defined on a grid is to use planar_intp to recast the data on a grid and then use contour afterwards.

Higher-dimensional interpolation is possible with tensor_grid.

26.1 Two-dimensional interpolation

```
/* Example: ex_twod_intp.cpp
   -------------------------------------------------------------------
   A simple example for two-dimensional interpolation using
   the twod_intp class.
*/

#include <o2scl/twod_intp.h>
#include <o2scl/test_mgr.h>

using namespace std;
using namespace o2scl;

// A function for filling the data and comparing results
double f(double x, double y) {
  return pow(sin(0.1*x+0.3*y),2.0);
}

int main(void) {
  int i,j;

  test_mgr t;
  t.set_output_level(1);

  // Create the sample data

  ovector x(3), y(3);
  omatrix data(3,3);

  cout.setf(ios::scientific);

  // Set the grid
  x[0]=0.0;
  x[1]=1.0;
  x[2]=2.0;
  y[0]=3.0;
  y[1]=2.0;
  y[2]=1.0;

  // Set and print out the data
  cout << endl;
  cout << "Data: " << endl;
  cout << "          x  | ";
  for(i=0;i<3;i++) cout << x[i] << " ";
  cout << endl;
  cout << " y           |" << endl;
  cout << "-------------|-";
  for(i=0;i<3;i++) cout << "-------------";
  cout << endl;
  for(i=0;i<3;i++) {
    cout << y[i] << " | ";
    for(j=0;j<3;j++) {
      data[i][j]=f(x[j],y[i]);
      cout << data[i][j] << " ";
    }
```

```
    cout << endl;
  }
  cout << endl;

  // Perform the interpolation

  cout << "x            y            Calc.        Exact" << endl;

  twod_intp ti;

  // Interpolation, x-first
  double tol=0.05;
  double tol2=0.4;

  ti.set_data(3,3,x,y,data,true);

  double x0, y0, x1, y1;

  x0=0.5; y0=1.5;
  cout << x0 << " " << y0 << " "
       << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

  x0=0.99; y0=1.99;
  cout << x0 << " " << y0 << " "
       << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

  x0=1.0; y0=2.0;
  cout << x0 << " " << y0 << " "
       << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

  cout << endl;

  // Interpolation, y-first

  ti.set_data(3,3,x,y,data,false);

  x0=0.5; y0=1.5;
  cout << x0 << " " << y0 << " "
       << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

  x0=0.99; y0=1.99;
  cout << x0 << " " << y0 << " "
       << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

  x0=1.0; y0=2.0;
  cout << x0 << " " << y0 << " "
       << ti.interp(x0,y0) << " " << f(x0,y0) << endl;

  cout << endl;

  t.report();
  return 0;
}
// End of example
```

This example creates a sample 3 by 3 grid of data with the function $[\sin(x/10+3y/10)]^2$ and performs some interpolations and compares them with the exact result.

```
Data:
          x  | 0.000000e+00 1.000000e+00 2.000000e+00
 y           |
-------------|---------------------------------------
3.000000e+00 | 6.136010e-01 7.080734e-01 7.942506e-01
2.000000e+00 | 3.188211e-01 4.150164e-01 5.145998e-01
1.000000e+00 | 8.733219e-02 1.516466e-01 2.298488e-01

x            y            Calc.        Exact
5.000000e-01 1.500000e+00 2.380255e-01 2.298488e-01
```

```
9.900000e-01 1.990000e+00 4.112589e-01 4.110774e-01
1.000000e+00 2.000000e+00 4.150164e-01 4.150164e-01

5.000000e-01 1.500000e+00 2.380255e-01 2.298488e-01
9.900000e-01 1.990000e+00 4.112589e-01 4.110774e-01
1.000000e+00 2.000000e+00 4.150164e-01 4.150164e-01
```

26.2 Contour lines

This example generates contour lines of the function

$$z = f(x,y) = 15\exp\left[-\frac{1}{20^2}(x-20)^2 - \frac{1}{5^2}(y-5)^2\right] + 40\exp\left[-\frac{1}{500}(x-70)^2 - \frac{1}{2^2}(y-2)^2\right]$$

```
/* Example: ex_contour.cpp
   -------------------------------------------------------------------
   Example for generating contour lines
*/

#include <iostream>
#include <o2scl/test_mgr.h>
#include <o2scl/contour.h>
#include <o2scl/ovector_tlate.h>

using namespace std;
using namespace o2scl;

// A function defining the three-dimensional surface
// for which we want to compute contour levels
double fun(double x, double y) {
  return 15.0*exp(-pow(x-20.0,2.0)/400.0-pow(y-5.0,2.0)/25.0)
    +40.0*exp(-pow(x-70.0,2.0)/500.0-pow(y-2.0,2.0)/4.0);
}

// A function for outputting the data to cout
int print_data(int nx, int ny, ovector_base &x, ovector_base &y,
               omatrix_base &data);

// A function for printing the contour information to a file
int file_out(vector<contour_line> &conts, vector<contour_line> &conts2);

int main(void) {
  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);

  contour co;

  // Initialize the data

  ovector x(12), y(10);
  omatrix data(10,12);
  for(size_t i=0;i<10;i++) {
    y[i]=((double)i);
  }
  for(size_t i=0;i<12;i++) {
    x[i]=((double)i)*((double)i);
  }

  for(size_t j=0;j<12;j++) {
    for(size_t k=0;k<10;k++) {
      data[k][j]=fun(x[j],y[k]);
    }
  }
  co.set_data(12,10,x,y,data);
```

```
  // Print out the data

  print_data(12,10,x,y,data);

  // Set the contour levels

  ovector levels(7);
  levels[0]=5.0;
  levels[1]=10.0;
  levels[2]=0.002;
  levels[3]=20.0;
  levels[4]=0.2;
  levels[5]=30.0;
  levels[6]=2.0;

  co.set_levels(7,levels);

  // Compute the contours

  vector<contour_line> conts;
  co.calc_contours(conts);

  // Print the contours to the screen and test to make sure
  // that they match the requested level

  size_t nc=conts.size();
  for(size_t i=0;i<nc;i++) {
    cout << "Contour " << i << " at level " << conts[i].level << ":" << endl;
    size_t cs=conts[i].x.size();
    for(size_t j=0;j<cs;j++) {
      cout << "(" << conts[i].x[j] << ", " << conts[i].y[j] << ") "
           << fun(conts[i].x[j],conts[i].y[j]) << endl;
      t.test_rel(fun(conts[i].x[j],conts[i].y[j]),conts[i].level,
                 1.0,"curve");
    }
    cout << endl;
  }

  // Refine the data using cubic spline interpolation

  def_interp_mgr<ovector_const_view,cspline_interp> dim1;
  def_interp_mgr<ovector_const_subvector,cspline_interp> dim2;
  co.regrid_data(5,5,dim1,dim2);

  // Recompute the contours

  vector<contour_line> conts2;
  co.calc_contours(conts2);

  // Output the contour information to a file for the documentation
  file_out(conts,conts2);

  t.report();

  return 0;
}
// End of example
```

The figure below shows contour lines in the region $x \in (0,120), y \in (0,9)$. The data grid is represented by plus signs, and the associated generated contours. The figure clearly shows the obvious peaks at $(20,5)$ and $(70,2)$.

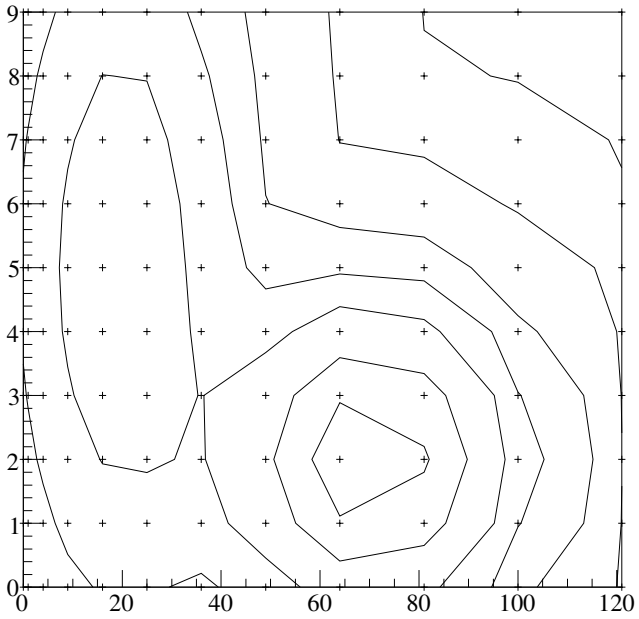


Figure 9: Contour example plot

The contour class can also use interpolation to attempt to refine the data grid. The new contours after a refinement of a factor of 5 is given in the figure below.

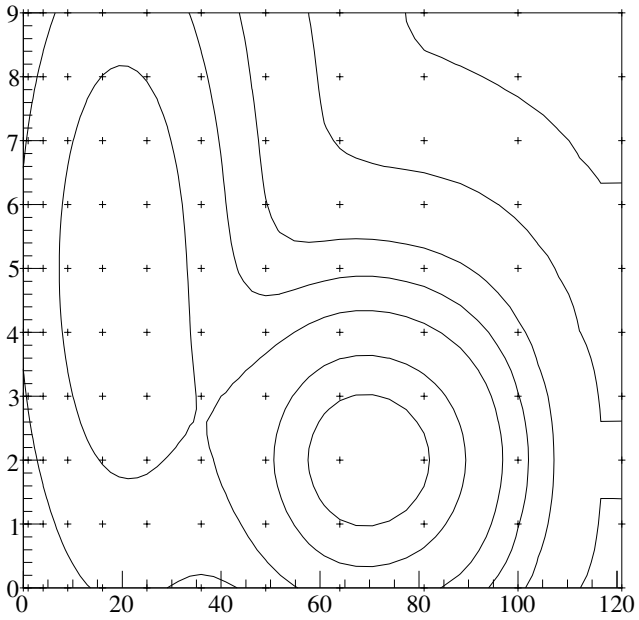


Figure 10: Contour example plot

27 Chebyshev Approximation

A class implementing the Chebyshev approximations based on GSL is given in gsl_chebapp. This class has its own copy constructor, so that Chebyshev approximations can be copied and passed as arguments to functions. Derivatives and integrals of gsl_chebapp objects are created as new gsl_chebapp objects which can be easily manipulated.

27.1 Chebyshev approximation example

This example performs an approximation of the function $y = \sin[1/(x+0.08)]$ over $[0, 2\pi]$. This function oscillates strongly over this interval and requires a high order approximation to be accurate.

The image below shows the approximation for $n = 50$ and $n = 25$. The $n = 100$ would be nearly indistinguishable from the exact result on this scale.

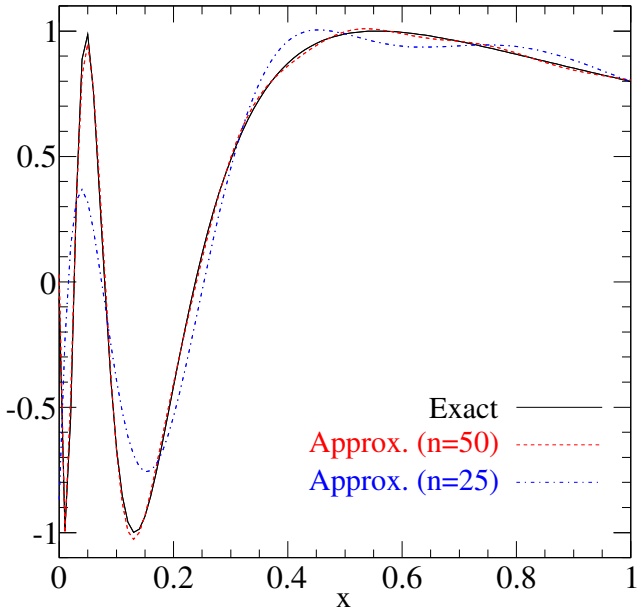


Figure 11: Chebyshev approximation plot

```
/* Example: ex_chebapp.cpp
   -------------------------------------------------------------------
*/

#include <iostream>
#include <o2scl/constants.h>
#include <o2scl/test_mgr.h>
#include <o2scl/gsl_chebapp.h>
#include <o2scl/cern_deriv.h>
#include <o2scl/gsl_inte_qag.h>

using namespace std;
using namespace o2scl;

double func(double x) {
  return sin(1.0/(x+0.08));
}

double dfunc(double x) {
  return -cos(1.0/(x+0.08))/pow(x+0.08,2.0);
}
```

```
// Simple function to output information to file for plotting
void write_file(gsl_chebapp &gc);

int main(void) {
  test_mgr t;
  t.set_output_level(1);

  cout.setf(ios::scientific);

  funct_fptr tf(func);

  gsl_chebapp gc;
  cern_deriv<> cd;
  gsl_inte_qag<> gi;

  double res, err;
  double x0=0.55;

  // Initialize the Chebyshev approximation
  gc.init(func,100,0.0,2.0*o2scl_const::pi);

  // Evaluate the approximation and compare with the exact result
  cout << "f(0.55)" << endl;
  cout << "Exact         : " << func(x0) << endl;
  gc.eval_err(x0,res,err);
  cout << "Approx (n=100): " << res << endl;
  cout << " Est. Error   : " << err << endl;
  cout << " Act. Error   : " << fabs(res-func(x0)) << endl;

  // Evaluate the approximation at lower order
  gc.eval_n_err(50,x0,res,err);
  cout << "Approx (n=50) : " << res << endl;
  cout << " Est. Error   : " << err << endl;
  cout << " Act. Error   : " << fabs(res-func(x0)) << endl;
  gc.eval_n_err(25,x0,res,err);
  cout << "Approx (n=25) : " << res << endl;
  cout << " Est. Error   : " << err << endl;
  cout << " Act. Error   : " << fabs(res-func(x0)) << endl;
  cout << endl;

  t.test_rel(gc.eval(x0),func(x0),1.0e-4,"eval");

  // Show how to use operator=() to create a new approximation
  gsl_chebapp gc2=gc;
  cout << "Using operator=(): " << gc2.eval(x0) << " " << func(x0) << endl;
  cout << endl;

  t.test_rel(gc2.eval(x0),gc.eval(x0),1.0e-10,"op=");

  // Show how to compute the derivative
  gsl_chebapp gc_deriv;
  gc.deriv(gc_deriv);

  cout << "f'(0.55)" << endl;
  cout << "Exact         : " << dfunc(x0) << endl;
  gc_deriv.eval_err(x0,res,err);
  cout << "Approx (n=100): " << res << endl;
  cout << " Est. Error   : " << err << endl;
  cout << " Act. Error   : " << fabs(res-dfunc(x0)) << endl;
  cd.calc_err(x0,tf,res,err);
  cout << "Direct deriv  : " << res << endl;
  cout << " Est. Error   : " << err << endl;
  cout << " Act. Error   : " << fabs(res-dfunc(x0)) << endl;
  cout << endl;

  t.test_abs(res,dfunc(x0),1.0e-12,"deriv with cern_deriv");
  t.test_abs(gc_deriv.eval(x0),dfunc(x0),5.0e-3,"deriv with cheb");

  // Show how to compute the integral
```

```
  gsl_chebapp gc_integ;
  gc.integ(gc_integ);

  cout << "int(f,0,0.55)" << endl;
  gc_integ.eval_err(x0,res,err);
  cout << "Approx (n=100): " << res << endl;
  cout << " Est. Error   : " << err << endl;
  gi.integ_err(tf,0.0,x0,res,err);
  cout << "Direct integ  : " << res << endl;
  cout << " Est. Error   : " << err << endl;
  cout << "Rel. Error    : " << fabs(res-gc_integ.eval(x0)) << endl;
  cout << endl;

  t.test_abs(gc_integ.eval(x0),gi.integ(tf,0.0,x0),1.0e-6,"integral");

  write_file(gc);

  t.report();
  return 0;
}
// End of example
```

28 Unit Conversions

There is a class which performs conversion between units specified with strings in convert_units. The convert_units_gnu class also uses a `system()` call to the GNU units command to obtain the proper conversion factors. Unit conversions are cached so that future requests of the same conversion factor don't require additional `system()` calls.

29 File I/O with HDF5

The class hdf_file facilitates I/O of data to hdf files. This class implements a simple way to I/O basic data types and O_2scl data types. All files created by the hdf_file class are normal HDF5 files, and can be manipulated in the usual way, for example with `h5dump` command-line tool. Users can easily mix code which performs I/O with hdf_file and other O_2scl functions with their own HDF code. The sole caveat is that O_2scl cannot parse generic HDF5 files, so that HDF files which contain data not output through O_2scl cannot always be read by O_2scl .

Objects are stored by refering to their dataset name. I/O for basic objects is provided directly in the hdf_file class, I/O for table, table3d, and other related objects is documented in hdf_io.h.

O_2scl formats complicated data types for HDF I/O by combining basic data into groups. For that reason, one cannot use O_2scl to read or write HDF files where groups have the same name as a dataset in the current HDF id. All O_2scl groups in HDF files come with a string named `o2scl_type`, which refers to the type of object which has been written to the HDF file as a group.

There are some current limitations regarding the matching of error handling policies between O_2scl and the HDF library. HDF functions do not always call the O_2scl error handler and thus do not always throw O_2scl exceptions.

Idea for Future Create an HDF file I/O example.

30 Other Classes and Functions

The O_2scl library contains several classes which are still under development and are to be considered experimental. While it is expected that the testing code associated with these classes will succeed on most any system, the interface, structure, and basic approaches used by these classes may change in future versions of O_2scl .

Three-dimensional data tables - table3d

Series acceleration - gsl_series

Command-line interface - cli

Automatic bin sizing - bin_size

Fourier transforms - gsl_fft

Polylogarithms - polylog

Histograms - hist

Two-dimensional histograms - hist_2d

31 Library Settings

There are a couple library settings which are handled by a global object lib_settings of type lib_settings_class.

There are several data files that are used by various classes in the library. The installation procedure should ensure that these files are automatically found. However, if these data files are moved after installation, then a call to lib_settings_class::set_data_dir() can adjust the library to use the new directory. It is assumed that the directory structure within the data directory has not changed.

32 Development Team

The current O2scl development team is:

- Andrew W. Steiner (Lead dev.)
- Jerry Gagelman

33 Design Considerations

The design goal is to create an object-oriented computing library with classes that perform common numerical tasks. The most important principle is that the library should add functionality to the user while at the same time retaining as much freedom for the user as possible and allowing for ease of use and extensibility. To that end,

- The classes which utilize user-specified functions should be able to operate on member functions without requiring a particular inheritance structure,
- The interfaces ought to be generic so that the user can create new classes which perform related numerical tasks through inheritance.
- The classes should not use static variables or status functions.
- Const-correctness and type-safety should be respected wherever possible.
- The design should be somewhat compatible with GSL.

Header file dependencies

For reference, it's useful to know how the top-level header files depend on each other, since it can be difficult to trace everything down. In the `base` directory, the following are the most "top-level" header files and their associated dependencies within O_2scl (there are other dependencies on GSL and the C standard library not listed here).

```
err_hnd.h : (none)
funct.h : (none)
lib_settings.h : (none)
array.h: err_hnd.h
exception.h: err_hnd.h
vector.h: err_hnd.h
```

```
sring_conv.h : lib_settings.h
misc.h : err_hnd.h lib_settings.h
test_mgr.h : string_conv.h
uvector_tlate.h: err_hnd.h string_conv.h array.h vector.h
ovector_tlate.h: err_hnd.h string_conv.h uvector_tlate.h array.h vector.h
search_vec.h: err_hnd.h ovector_tlate.h vector.h
```

Define constants

An attempt at a complete list:

- O2SCL_NEVER_DEFINED
- O2SCL_NO_EXCEPTIONS
- O2SCL_READLINE
- O2SCL_NO_SYSTEM_FUNC
- O2SCL_NO_RANGE_CHECK
- O2SCL_USE_GSL_HANDLER

The use of templates

Templates are used extensively, and this makes for longer compilation times so any code that can be removed conveniently from the header files should be put into source code files instead.

33.1 Error handling

Thread safety

Two approaches to thread-safe error handling which are worth comparing: the first is GSL which uses return codes and global function for an error handler, and the second is the Math/Special Functions section of Boost, which uses a separate policy type for each function. One issue is thread safety: the GSL approach is thread safe only in the sense that one can in principle use the return codes in different threads to track errors. What one cannot do in GSL is use different user-defined error handlers for different threads. The Special Functions library allows one to choose a different Policy for every special function call, and thus allows quite a bit more flexibility in designing multi-threaded error handling.

33.2 Memory allocation

Several classes have allocate() and free() functions to allocate and deallocate memory. If an error occurs in an allocate() function, the function should free() the partial memory that was allocated and then call the error handler. Functions which deallocate memory should never fail and should never be required to call the error handler. Similarly, class destructors should never be required to call the error handler.

33.3 Vector design

O_2scl vector and matrix types are a hybrid approach: creating objects compatibile with GSL, while providing syntactic simplicity and object-oriented features common to C++ vector classes. In terms of their object-oriented nature, they are not as elegant as the ublas vector types from ublas, but for many applications they are also faster (and they are always at least as fast).

33.4 Type-casting in vector and matrix design

O_2scl uses a GSL-like approach where viewing const double $*$ arrays is performed by explicitly casting aways const'ness internally and then preventing the user from changing the data.

In GSL, the preprocessor output for `vector/view_source.c` is:

```
gsl_vector_const_view_array (const double * base, size_t n)
{
  _gsl_vector_const_view view = {{0, 0, 0, 0, 0}};

  if (n == 0)
    {
      do { gsl_error ("vector length n must be positive integer", "
      view_source.c", 28, GSL_EINVAL) ; return view ; } while (0);

    }
  {
    gsl_vector v = {0, 0, 0, 0, 0};

    v.data = (double *)base ;
    v.size = n;
    v.stride = 1;
    v.block = 0;
    v.owner = 0;
    ((_gsl_vector_view *)&view)->vector = v;

    return view;
  }
}
```

Note the explicit cast from const double * to double *. This is similar to what is done in src/base/ovector_tlate.h.

33.5 Define constants and macros

There are a couple define constants and macros that O$_2$scl understands, they are all in upper case and begin with the prefix O2SCL_.

Range-checking for arrays and matrices is turned on by default, but can be turned off by defining O2SCL_NO_RANGE_CHECK during the initial configuration of the library. To see how the library was configured at runtime, use the lib_settings class.

There are several macros for error handling defined in err_hnd.h, and several for vector/matrix arithmetic in vec_arith.h .

There is a define constant O2SCL_NO_SYSTEM_FUNC which permanently disables the shell command '!' in cli (when the constant is defined, the shell command doesn't work even if cli::shell_cmd_allowed is true).

The constant O2SCL_DATA_DIR is defined internally to provide the directory which contains the O$_2$scl data files. After installation, this can be accessed in lib_settings.

All of the header files have their own define constant of the form O2SCL_HEADER_FILE_NAME which ensures that the header file is only included once.

Finally, I sometimes comment out sections of code with

```
#ifdef O2SCL_NEVER_DEFINED
...
#endif
```

This constant should not be defined by the user as it will cause compilation to fail.

33.6 Parameter ordering

In functions where this makes sense, generally input parameters will appear first, while output parameters or parameters which handle both input and output will appear later.

33.7 Global objects

There are four global objects that are created in libo2scl:

- def_err_hnd is the default error handler
- alt_err_hnd is the GSL-like error handler
- err_hnd is the pointer to the error handler (points to def_err_hnd by default)
- lib_settings to control a few library settings

All other global objects are to be avoided.

33.8 Thread safety

Most of the classes are thread-safe, meaning that two instances of the same class will not clash if their methods are called concurrently since static variables are only used for compile-time constants. However, two threads cannot, in general, safely manipulate the same instance of a class. In this respect, O_2scl is no different from GSL.

33.9 Documentation design

The commands \comment and \endcomment delineate comments about the documentation that are present in the header files but don't ever show up in the HTML or LaTeX documentation.

33.10 Copyright notices

For files where it is appropriate to do so, I have followed the prescription suggested in http://lists.gnu.org/archive/html/help-g
html retaining the GSL copyright notices and putting the O_2scl notices at the top. CERNLIB has no such standard, but their licensing information is outlined at http://cernlib.web.cern.ch/cernlib/conditions.html .

33.11 Design plans

Boost and linear algebra:

I would like to ensure this class is compatible with boost, and start integrating things accordingly. IMHO object-oriented linear algebra is in a rather sad state at the moment. uBlas and MTL are both promising, however, and I'd like to start implementing some sort of compatibility with uBlas vectors and matrices soon. The uBlas documentation is pretty sparse, but that's the pot calling the kettle a cheap piece of metal.

Other Improvements:

I'm particularly interested in improving the ODE and fitting classes, as well as updating the BFGS2 minimizer. Of course, more examples and better documentation are also a must.

Algorithms to include

- Method of lines for PDEs
- Some of the MESA interpolation routines.
- C++ translation of MINUIT (done already by ROOT, but quite difficult).
- Creating closed regions from contour lines (I have no idea how to do this at the moment, though I'm sure someone has solved this problem already somewhere.)

Complex numbers

I'm not sure where to go with complex numbers. My guess is that `std::complex` is not significantly slower (or is faster) than `gsl_complex`, but it would be good to check this. Then there's the C99 standard, which is altogether different. Unfortunately the interfaces may be sufficiently different that it's not easy to make templated classes which operate on generic complex number types.

34 License Information

O_2scl (as well as CERNLIB and the Gnu Scientific Library (GSL)) is licensed under version 3 of the GPL as provided in the files `COPYING` and in `doc/o2scl/extras/gpl_license.txt`. After installation, it is included in the documentation in `PREFIX/doc/extras/gpl_license.txt` where the default `PREFIX` is `/usr/local`. All of the code in the O_2scl documentation is also under version 3 of the GPL. (The license for this O_2scl documentation is provided below.)

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
```

States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary. To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License. Each licensee is addressed as "you". "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy. The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy. Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies. Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License. If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it. "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form. A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to

produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code;

keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7. This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy. This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged. This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit. Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source. This
    alternative is allowed only occasionally and noncommercially, and

only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

```
  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
```

this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a

consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

```
            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

The O_2scl library includes classes which manipulate HDF files. The NCSA and LLNL licenses for HDF are compatible with -GPLv3, as described at http://www.fsf.org/licensing/licenses . The HDF license is given below, and provided in `doc/o2scl/extras/hdf_license.txt`. After installation, it is included in the documentation in `PREFIX/doc/extras/hdf_license.txt` where the default `PREFIX` is `/usr/local`.

```
Copyright Notice and License Terms for
HDF5 (Hierarchical Data Format 5) Software Library and Utilities
-----------------------------------------------------------------------------

HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 2006-2009 by The HDF Group.

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
Copyright 1998-2006 by the Board of Trustees of the University of Illinois.
```

```
   opinions of authors expressed herein do not necessarily state or reflect
   those of the United States Government or the University of California,
   and shall not be used for advertising or product endorsement purposes.
-----------------------------------------------------------------------------
```

This documentation is provided under the GNU Free Documentation License, as given below and provided in `doc/o2scl/extras/fdl_license.txt`. After installation, it is included in the documentation in `PREFIX/doc/extras/fdl_license.txt` where the default `PREFIX` is `/usr/local`.

```
GNU Free Documentation License
  Version 1.2, November 2002


 Copyright (C) 2000,2001,2002  Free Software Foundation, Inc.
     51 Franklin Street, Fifth Floor, Boston, MA  02110-1301, USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other
functional and useful document "free" in the sense of freedom: to
assure everyone the effective freedom to copy and redistribute it,
with or without modifying it, either commercially or noncommercially.
Secondarily, this License preserves for the author and publisher a way
to get credit for their work, while not being considered responsible
for modifications made by others.

This License is a kind of "copyleft", which means that derivative
works of the document must themselves be free in the same sense.  It
complements the GNU General Public License, which is a copyleft
license designed for free software.

We have designed this License in order to use it for manuals for free
software, because free software needs free documentation: a free
program should come with manuals providing the same freedoms that the
software does.  But this License is not limited to software manuals;
it can be used for any textual work, regardless of subject matter or
whether it is published as a printed book.  We recommend this License
principally for works whose purpose is instruction or reference.


1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that
contains a notice placed by the copyright holder saying it can be
distributed under the terms of this License.  Such a notice grants a
world-wide, royalty-free license, unlimited in duration, to use that
work under the conditions stated herein.  The "Document", below,
refers to any such manual or work.  Any member of the public is a
licensee, and is addressed as "you".  You accept the license if you
copy, modify or distribute the work in a way requiring permission
under copyright law.

A "Modified Version" of the Document means any work containing the
Document or a portion of it, either copied verbatim, or with
modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of
the Document that deals exclusively with the relationship of the
publishers or authors of the Document to the Document's overall subject
(or to related matters) and contains nothing that could fall directly
within that overall subject.  (Thus, if the Document is in part a
textbook of mathematics, a Secondary Section may not explain any
mathematics.)  The relationship could be a matter of historical
```

connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other

conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications
   adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice
   giving the public permission to use the Modified Version under the
   terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections
   and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add
   to it an item stating at least the title, year, new authors, and
   publisher of the Modified Version as given on the Title Page. If
   there is no section Entitled "History" in the Document, create one
   stating the title, year, authors, and publisher of the Document as
   given on its Title Page, then add an item describing the Modified
   Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for
   public access to a Transparent copy of the Document, and likewise
   the network locations given in the Document for previous versions
   it was based on. These may be placed in the "History" section.
   You may omit a network location for a work that was published at
   least four years before the Document itself, or if the original
   publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications",
   Preserve the Title of the section, and preserve in the section all
   the substance and tone of each of the contributor acknowledgements
   and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document,
   unaltered in their text and in their titles. Section numbers
   or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section
   may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements"
   or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or
appendices that qualify as Secondary Sections and contain no material
copied from the Document, you may at your option designate some or all
of these sections as invariant. To do this, add their titles to the
list of Invariant Sections in the Modified Version's license notice.
These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains
nothing but endorsements of your Modified Version by various
parties--for example, statements of peer review or that the text has
been approved by an organization as the authoritative definition of a
standard.

You may add a passage of up to five words as a Front-Cover Text, and a
passage of up to 25 words as a Back-Cover Text, to the end of the list
of Cover Texts in the Modified Version. Only one passage of
Front-Cover Text and one of Back-Cover Text may be added by (or
through arrangements made by) any one entity. If the Document already
includes a cover text for the same cover, previously added by you or
by arrangement made by the same entity you are acting on behalf of,
you may not add another; but you may replace the old one, on explicit
permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License
give permission to use their names for publicity for or to assert or
imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this
License, under the terms defined in section 4 above for modified
versions, provided that you include in the combination all of the
Invariant Sections of all of the original documents, unmodified, and
list them all as Invariant Sections of your combined work in its

license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual

```
title.


9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except
as expressly provided for under this License.  Any other attempt to
copy, modify, sublicense or distribute the Document is void, and will
automatically terminate your rights under this License.  However,
parties who have received copies, or rights, from you under this
License will not have their licenses terminated so long as such
parties remain in full compliance.


10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions
of the GNU Free Documentation License from time to time.  Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.  See
http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number.
If the Document specifies that a particular numbered version of this
License "or any later version" applies to it, you have the option of
following the terms and conditions either of that specified version or
of any later version that has been published (not as a draft) by the
Free Software Foundation.  If the Document does not specify a version
number of this License, you may choose any version ever published (not
as a draft) by the Free Software Foundation.


ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of
the License in the document and put the following copyright and
license notices just after the title page:

    Copyright (c)  YEAR  YOUR NAME.
    Permission is granted to copy, distribute and/or modify this document
    under the terms of the GNU Free Documentation License, Version 1.2
    or any later version published by the Free Software Foundation;
    with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
    A copy of the license is included in the section entitled "GNU
    Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts,
replace the "with...Texts." line with this:

    with the Invariant Sections being LIST THEIR TITLES, with the
    Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other
combination of the three, merge those two alternatives to suit the
situation.

If your document contains nontrivial examples of program code, we
recommend releasing these examples in parallel under your choice of
free software license, such as the GNU General Public License,
to permit their use in free software.
```

35 Acknowledgements

The developers would like to thank the creators of GSL and Doxygen for their excellent work! Thanks also to Julien Garaud for contributing the ODE multishooting class and for the `ex_hydrogen` example, and to Marco Cammarata for contributing towards the gsl_smooth class.

36 Bibliography

Some of the references which contain links should direct you to the work referred to using its DOI identifer.

Bader83: G. Bader and P. Deuflhard, Numer. Math. **41** (1983) 373.

Birgin00: E.G. Birgin, J.M. Martinez, and M. Raydan, Nonmonotone Spectral Projected Gradient Methods on Convex Sets, SIAM Journal on Optimization 10:1196-1211, 2000.

Broyden65: C.G. Broyden, "A Class of Methods for Solving Nonlinear Simultaneous Equations", Mathematics of Computation, vol 19 (1965), p 577-593

Bus75: J. C. P. Bus and T. J. Dekker, ACM Trans. Math. Soft. **1** (1975) 330.

Cash90: J. R. Cash and A. H. Karp, ACM Trans. Math. Soft. **16** (1990) 201.

EngelnMullges96: G. Engeln-Müllges and F. Uhlig, Numerical Algorithms in C (1996) Springer, p. 96.

Fletcher87: R. Fletcher, Practical methods of optimization (John Wiley & Sons, Chichester 1987) p. 39.

Garbow80: B. S. Garbow, K. E. Hillstrom, and J. J. More, MINPACK project, Argonne Nat'l Lab, March 1980.

Hairer00: Hairer, E., Norsett S.P., Wanner, G. Solving ordinary differential equations I, Nonstiff Problems, 2nd revised edition, Springer, 2000.

Kelley99: C.T. Kelley, Iterative Methods of Optimization, SIAM, Philadelphia, 1999.

Krabs83: W. Krabs, Einfuhrung in die lieare und nictlineare Optimierung fur Ingenieure (BSB B.G. Teubner, Leipzig 1983) p. 84.

Lepage78: G. P. Lepage, J. Comp. Phys. **27** (1978) 192.

Lewin83: L. Lewin, Polylogarithms and Associated Functions (North-Holland, New York, 1983).

Longman58: I.M. Longman, MTAC (later renamed Math. Comp.) **12** (1958) 205.

More79: J.J. More' and M.Y. Cosnard, ACM Trans. Math. Software, **5** (1979) 64-85.

More80: J.J. More' and M.Y. Cosnard, Algoright 554 BRENTM, Collected Algorithms from CACM (1980).

Muller96: H. Muller and B.D. Serot, Nucl. Phys. A, 606 (1996) 508.

Nelder65: J.A. Nelder and R. Mead, Computer Journal 7 (1965) pp. 308-313.

Piessens83: R. Piessens, E. de Doncker-Kapenga, C. Uberhuber, and D. Kahaner QUADPACK, A Subroutine Package for Automatic Integration (1983) Springer-Verlag.

Press90: W.H. Press and G.R. Farrar, "Recursive Stratified Sampling for Multidimensional Monte Carlo Integration", Computers in Physics, v4 (1990), pp. 190-195.

Prince81: P.J. Prince and J.R. Dormand J. Comp. Appl. Math., 7 (1981) 67.

Rutishauser63: H. Rutishauser, Ausdehnung des Rombergschen Prinzips (Extension of Romberg's Principle), Numer. Math. **5** (1963) 48-54.

Tolstov62: G.P. Tolstov, "Fourier Series" (1962) Englewood Cliffs: Prentice Hall.

Wynn56: P. Wynn On a device for computing the e_m (S_n) transformation, Math. Tables Aids Comput. 10 (1956) 91.

37 Developer Guidelines

This set of guidelines is intended for O_2scl developers.

General coding guidelines:

- Comment code liberally.
- Avoid goto statements.

- Avoid hard-coded numerical values which are not either: (i) well-commented, or (ii) parameters modifiable by a class-user.
- Match the existing coding style when possible.
- Make sure all lines fit in 78 columns.
- Because this is a library (and not a end-user executable), it is generally better to call the error handler when an input is incorrectly specified instead of assuming some alternative correct value.
- All new classes deserve new documentation and new testing code.
- Documentation and code should be in standard American English.
- All code must be released in GPLv3.
- Global variables and static member data should be avoided.
- When reasonable, put input parameters first and output parameters last.
- When possible, templated vector parameters with size_t arguments should appear similar to `size_t &n, vec_t &v`, and in that order.
- All code should be ANSI-compatible, and, inasmuch as is possible, operating system and platform independent.
- Exception messages should be as informative as possible.
- Exception messages should end with the full class and function name from where they are thrown.
- Functions which deallocate memory should never fail and should never be required to call the error handler. Also, class destructors should never be required to call the error handler.
- Avoid passing pointers. Pass either bare objects (which then require copy constructors), shared pointers, or const references.
- For numeric parameters to functions, if not all values are permissible, then the error handler should be called when a non-permissible value is given by the user and the function documentation should clearly explain why.
- Wherever possible, objects should be usable by default. Avoid zombie objects which are instantiated but unusable.
- Objects should thread-safe in the weak sense, that is, different processes should be able to safely access and modify different instances of the same class at any time. Functions which read (but not modify) class data should be thread-safe in the strong sense, that is, different processes should be able to read the same instance of a class at any time.
- Whereever possible, ensure your code compiles without warnings using flags analogous to the gcc string

```
-ansi -pedantic -Wno-long-long -Wall -Wno-unused -Wextra
-Wconversion -Wshadow -Wpointer-arith -Wcast-align -Wwrite-strings
```

- Avoid 'try' blocks, as a goal is that O_2scl should compile with -fno-exceptions.
- Functions which return `void` should end with `return;`.

Documentation guidelines:

- Refer to other classes with \ref if necessary. Refer to function parameters with \c or embed them in html TT (text-type) commands.
- Bibliographic references should be used. When possible, include the DOI link which begins with the prefix `http://dx.doi.org` (not the vendor-specific DOI link).
- Comment Doxygen documentation with \comment and \endcomment. (Yes, sometimes comments in comments are useful.)

Patches:

- For those who are uncomfortable with Subversion, patches are always accepted.
- Prepare patches based on the most recent version of the trunk or the appropriate branch.

SVN:

- Editing the trunk directly should be avoided. Most development should occur on branches, to be merged with the trunk at the appropriate time.
- Ideally the trunk should always compile and all the tests should pass.
- Communicate with the lead developer before, during, and after any non-trivial development. Communicate your ideas before development, so that you don't write many lines of code only to find that your new code will be rejected. Communicate your ideas during development to avoid conflicting changes. Communicate your ideas after development to ensure they have a chance of being implmented. Subversion is not a replacement for real communication.
- Commit often.
- Merge carefully. If you fail to merge correctly, code added by other developers will be (at least temporarily) lost.
- Branches will be merged into the trunk by the lead developer at whatever time they deem appropriate.
- Developer-specific files which are not platform-independent should not be added to the repository. Sometimes svn:ignore can be used to ignore these files, but this should be done sparingly.

38 Download O2scl

The current version is 0.910. The source distribution can be obtained from

- http://sourceforge.net/projects/o2scl/files/

You may also download O_2scl from version from the Subversion repository. To obtain the bleeding-edge developer version, use something like

```
svn co https://o2scl.svn.sourceforge.net/svnroot/o2scl/trunk o2scl
```

The checkout process is a bit more time consuming than directly downloading the source distribution.

Version 0.910 corresponds with version 376 in the svn repository.

Version 0.909 corresponds with version 317 in the svn repository.

Version 0.908 corresponds with version 200 in the svn repository.

Version 0.907 corresponds with version 107 in the svn repository.

Version 0.906 corresponds with version 54 in the svn repository.

Version 0.905 corresponds with version 48 in the svn repository.

Version 0.904 corresponds with version 40 in the svn repository.

Version 0.903 does not exist (this number was skipped to help coordinate numbering with O_2scl_ext).

Version 0.902 corresponds with version 37 in the svn respository.

Version 0.901 corresponds with version 31 in the svn respository.

Version 0.9 corresponds with version 26 in the svn respository.

Version 0.806 was a quick release to fix some issues with compilations in Mac OS X and does not correspond exactly with a version in the repository.

Version 0.805 corresponds with version 16 in the svn respository.

39 Ideas for Future Development

Class anneal_mt< param_t, param_vec_t, func_t, vec_t, alloc_vec_t, alloc_t, rng_t >

There may be a good way to remove the function indirection here to make this class a bit faster.

File array.h

Create a class which views a C-style array as a matrix and offers an `operator(,)`

Class array_alloc< vec_t >

Might it be possible to rework this so that it does range checking and ensures that the user doesn't try to allocate more or less space? I.e. array_alloc<double[2]> complains if you try an allocate(x,3)?

Class array_const_subvector

Make the member data truly const and remove the extra typecast.

Class bin_size

Finish this.

File cblas_base.h

Add float and complex versions.

There are some Level-1 BLAS functions which are already present in vector.h. Should we move all of them to one place or the other? The ones in vector.h are generic in the sense that they can use doubles or floats, but the ones here can use either () or [].

Class cern_adapt< func_t, nsub >

Allow user to set the initial subdivisions?

- It might be interesting to directly compare the performance of this class to gsl_inte_qag .
- There is a fixme entry in the code which could be resolved.
- Output the point where most subdividing was required?

Class cern_deriv< func_t >

All of the coefficients appear to be fractions which could be replaced with exact representation?

Record the number of function calls?

Class cern_gauss< func_t >

Allow user to change `cst`?

Class cern_mroot< func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >

Modify this so it handles functions which return non-zero values.

Move some of the memory allocation out of msolve()

Give the user access to the number of function calls

Rename nier6, nier7, and nier8 to something sensible.

It may be that the O_2scl native Householder transformations should be used here instead of the inline version given here.

Class cern_mroot_root< func_t >

Double-check this class to make sure it cannot fail while returning 0 for success.

Global cern_mroot_root< func_t >::eps

This number should probably default to one of the GSL tolerances.

Class cli

Warn in run_interactive() when extra parameters are given

Include a "remove command" function

A replace command function, there's already some code in cli.cpp for this.

There's some code duplication between comm_option_run() and run_interactive()

Allow the user to set the tilde string

Disallow direct access to cli::par_list in order to ensure parameter names do not contain whitespace

Global cli::cli_gets (const char *c)

Think about whether or not this should be protected? (Possibly not, as it's extensively used by acolm.cpp)

Class columnify

Move the screenify() functionality from misc.h into this class?

It might be better to allow the string table to be specified with iterators.

Class comp_gen_inte< func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t >

Provide an example of usage for this class.

Class composite_inte< func_t, vec_t, alloc_vec_t, alloc_t >

Create a function to set an entire array of one-dimensional integration objects at once

Class contour

Rewrite the code which adjusts the contour levels to ensure contours don't go through the data to adjust the internal copy of the data instead.

It would be nice to have a function which creates a set of closed regions to fill which represent the data. However, this likely requires a completely new algorithm, because it's not easy to simply close the contours already generated by the calc_contours() function. There are, for example, several cases which are difficult to handle, such as filling a region in between several closed contours.

Global contour::get_data (size_t &sizex, size_t &sizey, ovector *&x_fun, ovector *&y_fun, omatrix *&udata)

There is probably a better way than returning pointers to the internal data.

Class contour_line

Write HDF I/O for contour lines?

Make this a subclass of contour .

Class convert_units

Ideally, a real C++ API for the GNU units command would be better.

File cx_arith.h

Define operators with assignment for complex + double?

Class deriv< func_t >

Improve the methods for second and third derivatives

Class edge_crossings

Write HDF I/O object for edge crossings

Make this a subclass of contour .

Class eqi_deriv< func_t, vec_t >

Finish the second and third derivative formulas.

Class err_hnd_type

There may be an issue associated with the string manipulations causing errors in the error handler.

Class expect_val

If `n_per_block` is nonzero and the user adds more than `n_blocks` times `n_per_block` data, set `n_per_block` to zero, allocate the `last` structure and fall back to the variable block size method.

Page File I/O with HDF5

Create an HDF file I/O example.

Class format_float

Handle inf's and nan's correctly.

Allow change of string for the "+" sign for the exponent

Class gen_inte< func_t, lfunc_t, ufunc_t, vec_t >

It might be interesting to construct a child class of gen_inte which automatically transforms variables to a hypercube and then applies a child of multi_inte to do the integration.

Class gen_test_number< tot >

Document what happens if `tot` is pathologically small.

Class gradient< func_t, vec_t >

Consider making an exact_grad class for computing exact gradients.

Class gsl_anneal< func_t, vec_t, alloc_vec_t, alloc_t, rng_t >

There's x0, old_x, new_x, best_x, and x? There's probably some duplication here which could be avoided.

- Implement a more general simulated annealing routine which would allow the solution of discrete problems like the - Traveling Salesman problem.

Class gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >

Compare more directly to GSL

Class gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >

Create an example with a stiff diff eq. which requires this kind of stepper

More detailed documentation about the algorithm

Figure out if this should be a child of ode_step or adapt_step. The function step_local() is actually its own ODE stepper and could be reimplemented as an object of type ode_step.

I don't like setting yerr to GSL_POSINF, there should be a better way to force an adaptive stepper which is calling this stepper to readjust the stepsize.

The functions deuf_kchoice() and compute_weights() can be moved out of this header file

Rework internal arrays as uvectors?

Rework linear solver to be amenable to using a sparse matrix solver

Global gsl_cubic_real_coeff::gsl_poly_complex_solve_cubic2 (double a, double b, double c, gsl_complex ∗z0, gsl_complex ∗z1, gsl_complex ∗z2)

I think the GSL function is now fixed, so we can fall back to the original GSL function here.

Class gsl_deriv< func_t >

Include the forward and backward GSL derivatives?

Class gsl_inte_kronrod< func_t >

Convert 'double ∗' objects to uvector

Global gsl_inte_kronrod< func_t >::gauss_kronrod_base (func2_t &func, double a, double b, double ∗result, double ∗abserr, double ∗resabs, double ∗resasc)

This function, in principle, is an integration object in and of itself, and could be implemented separately as an object of type inte.

Class gsl_inte_qag< func_t >

There are a few fine-tuned parameters which should be re-expressed as data members in the convergence tests.

Should QUADPACK parameters in round-off tests be subject to adjustments by the end user?

Add functions to examine the contents of the workspace to detect regions where the integrand may be problematic; possibly call these functions automatically depending on verbosity settings.

Global gsl_inte_qag< func_t >::qag (func_t &func, const double a, const double b, const double l_epsabs, const double l_epsrel, double ∗result, double ∗abserr)

Just move this function to integ_err().

Class gsl_inte_qawc< func_t >

Make cern_cauchy and this class consistent in the way which they require the user to provide the denominator in the integrand

Global gsl_inte_qawo_sin< func_t >::qawo (func_t &func, const double a, const double epsabs, const double epsrel, gsl_inte_workspace *loc_w, gsl_integration_qawo_table *wf, double *result, double *abserr)

Remove goto statements.

Class gsl_inte_singular< func_t >

Some of the functions inside this class could be moved out of header files?

Global gsl_inte_singular< func_t >::extrap_table

Move this to a new class, with qelg() as a method

Global gsl_inte_singular< func_t >::extrap_table

Move this to a new class, with qelg() as a method

Global gsl_inte_singular< func_t >::qags (func_t &func, const double a, const double b, const double l_epsabs, const double l_epsrel, double *result, double *abserr)

Remove goto statements. Before this is done, it might be best to add some tests which fail in the various ways.

Global gsl_min_brent< func_t >::iterate ()

It looks like x_left and x_right can be removed. Also, it would be great to replace the one-letter variable names with something more meaningful.

Class gsl_min_quad_golden< func_t >

Take common elements of this and gsl_min_brent and move to a generic GSL minimizer type?

Global gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::estimate_corrmc (func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, size_t calls, double &res, double &err, const uvector &lxmid, uvector &lsigma_l, uvector &lsigma_r)

Remove the reference to GSL_POSINF and replace with a function parameter.

Class gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >

Double check that the updates in gsl-1.13 are included here, and also add support for the nmsimplex2rand algorithm in GSL.

Class gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >

It's kind of strange that set() sets jac_given to false and set_de() has to reset it to true. Can this be simplified?

It might be good to make sure that we're directly testing to make sure that GSL and O2SCL are more or less identical.

Many of these minpack functions could be put in their own "minpack_tools" class, or possibly moved to be linear algebra routines instead.

There are still some numbers in here which the user could have control over, for example, the `nslow2` threshold which indicates failure.

Class gsl_quartic_real

Optimize value of `cube_root_tol` and compare more clearly to gsl_quartic_real2

Class gsl_quartic_real2

Optimize value of `cube_root_tol` and compare more clearly to gsl_quartic_real

Class gsl_root_brent< func_t >

There is some duplication in the variables `x_lower`, `x_upper`, `a`, and `b`, which could be removed. Some better variable names would also be helpful.

Create a meaningful enum list for gsl_root_brent::test_form.

Class gsl_root_stef< func_t, dfunc_t >

There's some extra copying here which can probably be removed.

Compare directly to GSL.

This can probably be modified to shorten the step if the function goes out of bounds as in gsl_mroot_hybrids.

Class gsl_series

Move the worspaces to classes?

Class gsl_smooth

Generalize to generic vector types. (Does this require reworking the GSL linear fitting routines?) In the meantime, make a ovector interface.

Possibly create a new gsl_bspline class which replaces the GSL bspline workspace

Allow user to probe chi squared and the covariance?

Class gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

Prettify the verbose output

Allow the user to get information about the how the sampling was done, possibly by converting the bins and boxes into a structure or class.

Allow the user to change the maximum number of bins.

Class hdf_file

Automatically close groups, e.g. by storing hid_t's in a stack?

Class hist

Would be nice not to have to create a new search_vec object in get_bin_index().

Consider adding the analogs of the GSL histogram sampling functions

Add a function which computes the bin sizes?

Class hist_2d

Write a function to create a 1-d histogram from a 2-d histogram either by selecting one bin in one axis or by marginalizing over one direction.

Global hybrids_base::compute_df (size_t n, const vec2_t &ff_trial, const vec2_t &fl, ovector_base &dfl)

Replace with daxpy?

Global hybrids_base::compute_qtf (size_t N, const vec2_t &q2, const vec3_t &f2, vec4_t &qtf2)

This is just right-multiplication, so we could use the O_2scl cblas routines instead.

Global hybrids_base::compute_trial_step (size_t N, vec2_t &xl, vec2_t &dxl, vec2_t &xx_trial)

Replace with daxpy.

Global hybrids_base::enorm (size_t N, const vec2_t &ff)

Replace this with `dnrm2` from cblas_base.h

Class inte< func_t >

It might be useful to have all of the integration classes report the number of function evaluations used in addition to the number of iterations which were taken

Class lanczos< vec_t, mat_t, alloc_vec_t, alloc_t >

The function eigen_tdiag() automatically sorts the eigenvalues, which may not be necessary.

Do something better than the naive matrix-vector product. For example, use dgemm() and allow user to specify column or row-major.

Rework memory allocation to perform as needed.

Global matrix_cx_copy_gsl (size_t M, size_t N, mat_t &src, mat2_t &dest)

At present this works only with complex types based directly on the GSL complex format. This could be improved.

Global matrix_max (size_t n, const size_t m, const mat_t &data)

Write matrix_max_index() and related functions similar to the way vector_max_index() works.

Global matrix_out (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)

If all of the matrix elements are positive integers and scientific mode is not set, then we can avoid printing the extra spaces.

Class matrix_row< mat_t >

Generalize to operator(,)

Page Minimization

Plot the spring function

Class minimize< func_t, dfunc_t >

Add a class for the quad_golden algorithm from GSL.

Global minimize< func_t, dfunc_t >::bracket (double &ax, double &bx, double &cx, double &fa, double &fb, double &fc, func_t &func)

Improve this algorithm with the golden ratio method in gsl/min/bracketing.c?

Class minimize_de< func_t, dfunc_t >

Create a version of gsl_mmin_conf which implements a minimizer with this interface.

Class mroot< func_t, vec_t, jfunc_t >

Change ntrial to size_t?

Global O2SCL_ASSERT (ev)

Make this consistent with assert() using NDEBUG?

Global o2scl_graph::new_graph_ticks (TCanvas *&c1, TPad *&p1, TH1 *&th1, std::string CanvasName, std::string WindowName, std::string PadName, double lleft, double lbottom, double lright, double ltop, TGaxis *&a1, TGaxis *&a2, int left=0, int top=0, int right=700, int bottom=700, bool logx=false, bool logy=false)

Modify this to just directly call new_graph() and add the new axes on top.

Namespace o2scl_linalg

Move this documentation to the linalg directory.

Global o2scl_linalg::HH_svx (size_t N, size_t M, mat_t &A, vec_t &x)

Handle memory allocation like the tridiagonal functions.

Class o2scl_linalg::linear_solver< vec_t, mat_t >

The test code uses a Hilbert matrix, which is known to be ill-conditioned, especially for the larger sizes. This should probably be changed.

Global o2scl_linalg::LU_decomp (const size_t N, mat_t &A, o2scl::permutation &p, int &signum)

The "swap rows j and i_pivot" section could probably be made more efficient using a "matrix_row"-like object as done in GSL. (7/16/09 - I've tried this, and it doesn't seem to improve the speed significantly.)

Global o2scl_linalg::LU_invert (const size_t N, const mat_t &LU, const o2scl::permutation &p, mat2_t &inverse)

Could rewrite to avoid mat_col_t, (9/16/09 - However, the function may be faster if mat_col_t is left in, so it's unclear what's best.)

Global o2scl_linalg::solve_cyc_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N, mem_t &m)

Offer an option to avoid throwing on divide by zero?

Global o2scl_linalg::solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N, mem_t &m)

Offer an option to avoid throwing on divide by zero?

Class ode_bv_mshoot< func_t, mat_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >

Make a class which performs an iterative linear solver which uses sparse matrices like ode_it_solve?

Class ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >

Create a GSL-like set() and iterate() interface

Implement as a child of ode_bv_solve ?

Max and average tolerance?

Class ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >

The form of solve_final_value() is very similar to that of adapt_step::astep_full(), but not quite the same. Maybe should probably be made to be consistent with each other?

Global ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_grid (double h, size_t n, size_t nsol, vec_t &xsol, mat_t &ysol, mat_t &err_sol, mat_t &dydx_sol, func_t &derivs)

Consider making a version of grid which gives the same answers as solve_final_value(). After each proposed step, it would go back and fill in the grid points if the proposed step was past the next grid point.

Global ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_store (double x0, double x1, double h, size_t n, size_t &n_sol, vec_t &x_sol, mat_t &y_sol, mat_t &yerr_sol, mat_t &dydx_sol, func_t &derivs, size_t istart=0)

It might be possible to remove some extra copying by removing the yerrl and dydx vectors

Class ofvector< N >

Consider making allocate() and free() functions private for this class?

Class omatrix_col_tlate< data_t, mparent_t, vparent_t, block_t >

Also do a umatrix constructor since we can't do that with uvectors

File omatrix_tlate.h

The `xmatrix` class demonstrates how operator[] could return an ovector_array object and thus provide more bounds-checking. This would demand including a new parameter in omatrix_view_tlate which contains the vector type.

Class ool_constr_mmin< func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >

Implement automatic computations of gradient and Hessian

Construct a more difficult example for the "examples" directory

Finish mmin() interface

Implement a direct computation of the hessian as the jacobian of the gradient

Class ool_mmin_pgrad< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >

Replace the explicit norm computation below with the more accurate dnrm2 from linalg

Replace the generic variable 'tol' with 'tolf' or 'tolx' from multi_min.

Class ool_mmin_spg< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >

There is some memory allocation which isn't deallocated until the destructor, and should be handled a bit more elegantly.

Global operator<< (std::ostream &os, const umatrix_cx_view_tlate< data_t, complex_t > &v)

Make this function work even when scientific mode is not on, either by converting to scientific mode and converting back, or by leaving scientific mode off and padding with spaces

Global operator<< (std::ostream &os, const umatrix_const_view_tlate< data_t > &v)

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Global operator<< (std::ostream &os, const omatrix_const_view_tlate< data_t, parent_t, block_t > &v)

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Class other_todos_and_bugs

Make sure we have a uvector_cx_alloc, ovector_cx_const_reverse, ovector_cx_const_subvector_reverse, uvector_reverse, uvector_const_reverse, uvector_subvector_reverse, uvector_const_subvector_reverse, omatrix_cx_diag, blah_const_diag, umatrix_diag, and umatrix_cx_diag

- ovector_cx_view::operator=(uvector_cx_view &) is missing
- ovector_cx::operator=(uvector_cx_view &) is missing
- uvector_c_view::operator+=(complex) is missing
- uvector_c_view::operator-=(complex) is missing
- uvector_c_view::operator∗=(complex) is missing

Currently, I believe the testing code requires the shared libraries and may not work if the static libraries only are installed. It would be nice to ensure the tests could be complied in either case.

Class ovector_alloc

Could (or should?) the allocate() functions be adapted to return an integer error value?

Class ovector_const_reverse_tlate< data_t, vparent_t, block_t >

I think that maybe in order to ensure that this isn't created from an already reversed vector, this class has to be separated from the hierarchy altogether. However, I think this might break the smart interpolation stuff.

File ovector_tlate.h

Clean up maybe by moving, for example, ovector reverse classes to a different header file

Define ovector_uint classes?

Class pinside

The inside() functions actually copy the points twice. This can be made more efficient.

Class planar_intp< vec_t, mat_t >

Rewrite so that it never fails unless all the points in the data set lie on a line. This would probably demand sorting all of the points by distance from desired location. (11/1/11 - Maybe not. We might be able to get away with just storing an vector which records the index j of the point in the data set which is the nearest neighbor for each point i in the data set.)

Instead of comparing `den` with zero, add the option of comparing it with a small number.

Allow the user to specify the scales used for x and y, instead of computing max-min every time.

The generalization to more dimensions might be straight forward.

Class pointer_2d_alloc< base_t >

There may be a slight issue here if the pointer allocation succeeds and the list insertion fails, and the user catches the exception thrown by the list insertion failure, then a memory leak will ensue. This might be partially ameliorated by counting allocations and insertions. This failure mechanism is rarely encountered in practice.

Class pointer_alloc< base_t >

There may be a slight issue here if the pointer allocation succeeds and the list insertion fails, and the user catches the exception thrown by the list insertion failure, then a memory leak will ensue. This might be partially ameliorated by counting allocations and insertions. This failure mechanism is rarely encountered in practice.

File poly.h

The quartics are tested only for a4=1, which should probably be generalized.

Class polylog

Give error estimate?

- Improve accuracy?
- Use more sophisticated interpolation?
- Add the series $Li(n,x) = x + 2^{-n}x^2 + 3^{-n}x^3 + \ldots$ for $x \to 0$?
- Implement for positive arguments < 1.0
- Make another polylog class which implements series acceleration?

Global quadratic_extremum_x (double x1, double x2, double x3, double y1, double y2, double y3)

Make a function quadratic_points(double x1, double x2, double x3, double y1, double y2, double y3, double &a, double &b, double &c)?

Make a quadratic_extremum_xy()?

Class rnga

Consider some analog of the GSL function `gsl_rng_uniform_pos()`, i.e. as used in the GSL Monte Carlo classes.

Global rnga::clock_seed ()

Figure out a better way of computing a random seed in a platform-independent way.

Class root< func_t, dfunc_t >

Maybe consider allowing the user to specify the stream to which 'verbose' information is sent.

Class root_de< func_t, dfunc_t >

At the moment, the functions solve() and solve_bkt() are not implemented for derivative solvers.

Global scalar_ev::reblock_avg_stats (size_t new_blocks, double &avg, double &std_dev, double &avg_err, size_t &m_per_block) const

Use recurrence relation for averages here rather than dividing at the end.

Class search_vec< vec_t >

Consider a sanity check to ensure that the cache is never larger than the vector length.

Global search_vec< vec_t >::ordered_lookup (const double x0) const

This function just uses the `find` functions and then adjusts the answer at the end if necessary. It might be possible to improve the speed by rewriting this from scratch.

Global search_vec_ext< vec_t >::search_vec_ext (size_t nn, vec_t &x)

This could be rewritten to allow vectors with only one element (5/2/11: It looks like this might have been done already?)

Class smart_interp_vec< vec_t, svec_t, alloc_vec_t, alloc_t >

Properly implement handling of non-monotonic regions in the derivative functions as well as the interpolation function.

Class table

Rewrite the table::create_array() and table::insert_data() functions for generic vector type

- Return the empty column in the operator[] functions as is done for the get_column() functions.
- A "delete rows" method to delete a range of several rows
- The present structure, `std::map<std::string,col,string_comp>` atree and `std::vector<aiter>` alist; could be replaced with `std::vector<col>` list and `std::map<std::string,int>` tree where the map just stores the index of the the column in the list.

Class table3d

Improve interpolation and derivative caching

Make a 'const' version of the interpolation functions

Should there be a clear_grid() function separate from clear_data() and clear_table()?

Class table_units

Make table methods virtual? (not necessary yet since delete_column() isn't referred to internally)

Class tensor

Could implement arithmetic operators + and - and some different products.

Implement copies to and from vector and matrices

Implement tensor contractions, i.e. tensor = tensor $*$ tensor

Consider making a template type to replace double, e.g. `value_t`.

Could be interesting to write an iterator for this class.

Class tensor_grid

Only allocate space for grid if it is set

Could implement arithmetic operators + and - and some different products.

Consider creating a set_grid() function which takes grids from an object like hist_grid. Maybe make a constructor for a tensor_grid object which just takes as input a set of grids?

Global tensor_grid::interpolate (double *vals)

Maybe make this a template as well?

It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor. However, this should probably be superceded by a more generic alternative which avoids explicit use of the 1-d interpolation types.

It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor. However, this should probably be superceded by a more generic alternative which avoids explicit use of the 1-d interpolation types.

Global tensor_grid::set_grid (const vec_t &grid_vec)

Define a more generic interface for matrix types

Class tensor_old

Could implement arithmetic operators + and - and some different products.

Implement copies to and from vector and matrices

Implement tensor_old contractions, i.e. tensor_old = tensor_old * tensor_old

Consider making a template type to replace double, e.g. `value_t`.

Could be interesting to write an iterator for this class.

Class tensor_old_grid

Only allocate space for grid if it is set

Could implement arithmetic operators + and - and some different products.

Consider creating a set_grid() function which takes grids from an object like hist_grid. Maybe make a constructor for a tensor_old_grid object which just takes as input a set of grids?

Global tensor_old_grid::interpolate (double *vals)

Maybe make this a template as well?

It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor_old. However, this should probably be superceded by a more generic alternative which avoids explicit use of the 1-d interpolation types.

It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor_old. However, this should probably be superceded by a more generic alternative which avoids explicit use of the 1-d interpolation types.

Global tensor_old_grid::set_grid (const vec_t &grid)

Define a more generic interface for matrix types

Class test_mgr

test_mgr::success and test_mgr::last_fail should be protected, but that breaks the operator+() function. Can this be fixed?

Class twod_intp

Could also include mixed second/first derivatives: deriv_xxy and deriv_xyy.

Implement an improved caching system in case, for example `xfirst` is true and the last interpolation used the same value of `x`.

Class uvector_const_view_tlate< data_t >

Could allow user-defined specification of restrict keyword

Class uvector_cx_view_tlate< data_t, complex_t >

Write lookup() method, and possibly an erase() method.

File vec_arith.h

Define operators for complex vector * real matrix

Define == and != for complex vectors

These should be replaced by the BLAS routines where possible?

File vec_stats.h

Consider generalizing to other data types.

File vector.h

Create a matrix transpose copy function?

Create matrix swap row and column functions

Global vector_cx_copy_gsl (size_t N, vec_t &src, vec2_t &dest)

At present this works only with complex types based directly on the GSL complex format. This could be improved.

Global vector_lookup (size_t n, const vec_t &x, double x0)

Write matrix_lookup().

40 Todo List

Class cli

Long options cannot be one letter long, or else process_args() will fail, thus the class should throw if a long option with only one letter is given.

Global cli::process_args (std::string s, std::vector< cmd_line_arg > &ca, int debug=0)

There's a typecast in this function to (char *) from (const char *) which needs reworking.

Global eqi_deriv< func_t, vec_t >::deriv_vector (size_t nv, double dx, const vec_t &y, vec_t &dydx)

generalize to other values of npoints.

File givens_base.h

Make sure create_givens() in givens.h is documented.

Class gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >

Document what happens when the stepper function returns a non-zero value, as it's different now with the ode-initval2 function.

Document count, failed_steps, etc.

Class gsl_fit< func_t, vec_t, mat_t, bool_vec_t >

Properly generalize other vector types than ovector_base

Allow the user to specify the derivatives

Fix so that the user can specify automatic scaling of the fitting parameters, where the initial guess are used for scaling so that the fitting parameters are near unity.

Class gsl_mmin_bfgs2< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

While BFGS does well in the `ex_mmin` example with the initial guess of $(1,0,7\pi)$ it seems to converge more poorly for the spring function than the other minimizers with other initial guesses, and I think this will happen in the GSL versions too. I need to examine this more closely with some code designed to clearly show this.

Class gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >

Are all the set_all() statements really necessary? Do they need to be executed even if memory hasn't been recently allocated?

Class gsl_ode_control< vec_t >

Double check that the improvements in the ode-initval2 routines are available here

Class gsl_rkf45< func_t, vec_t, alloc_vec_t, alloc_t >

Check this because it may not give exact dydt_out.

Class gsl_smooth

Needs a bit more error checking and more documentation.

Class gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

Mode = importance only doesn't give the same answer as GSL yet.

Global gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::vegas_minteg_err (int stage, func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, double &res, double &err)

Should stage be passed by reference?

There was an update between gsl-1.12 and 1.15 which has not been implemented here yet.

Class hdf_file

Error handling with HDF functions is a particular concern. Modify to throw O_2scl exceptions when appropriate.

Class hist

Check that the actions of set_bins(), clear() and other functions perform the correct actions on the user_rep vector. I'm a bit concerned that set_bins_auto() shouldn't call user_rep.allocate().

More documentation and testing.

Class hist_ev

Test set_grid_blocks() function.

Create copy constructors as in the scalar_ev class.

File householder_base.h

Better documentation for the Householder functions.

Global matrix_cx_out_paren (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)

Doesn't this only work for GSL matrices? Compare this to the corresponding vector functions

Class multi_min_fix< bool_vec_t >

Generalize to all vector types

Generalize to minimizers which require derivatives

At the moment, the user has to change def_mmin::ntrial instead of multi_min_fix::ntrial, which is a bit confusing. Fix this.

Class ode_bv_multishoot< func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t, mat_t >

Improve documentation a little and create testing code

Global ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_final_value (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, func_t &derivs)

Document if yend can be the same as ystart.

Class other_todos_and_bugs

Make sure an error message is printed to ensure users do 'make o2scl-test' and not 'make check'

- Make sure cstdlib is included wherever exit() is used [5/22/11 - Most of these are taken care of now.]

Class ovector_const_view_tlate< data_t, vparent_t, block_t >::const_iterator

Default constructor and iterator typedefs

Class ovector_cx_tlate< data_t, vparent_t, block_t, complex_t >

Add subvector_stride, const_subvector_stride

Global smart_interp< vec_t, svec_t >::find_subset (const double a, const double b, size_t sz, const vec_t &x, const vec_t &y, size_t &nsz)

The error handling is a bit off here, as it can return a non-zero value even with there is no real "error". We should just make a new bool reference paramter.

Global smart_interp< vec_t, svec_t >::interp (const double x0, size_t n, const vec_t &x, const vec_t &y)

After calling find_subset, I think we might need to double check that `nn` is larger than the minimum interpolation size.

Global smart_interp_vec< vec_t, svec_t, alloc_vec_t, alloc_t >::find_inc_subset (const double x0, size_t sz, const vec_t &x, const vec_t &y, size_t &nsz)

Variables `row` and `row` appear to be unused? (2/17/11)

Class table

Document or fix the fact that the table copy constructors do not copy the interpolation objects. Maybe def_interp_mgr objects can have their own copy constructors?

Global table::sort_column (std::string scol)

Use vector_sort() rather than qsort().

Global table::sort_table (std::string scol)

Use vector_sort() rather than qsort().

Global uvector_const_view_tlate< data_t >::norm () const

Fix this so that norm() is computed as in ovector and so that integer norms are performed separately.

Global vector_pvariance (size_t n1, const vec_t &data1, size_t n2, const vec2_t &data2)

Document this

41 Namespace Index

41.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

42 Data Structure Index

42.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

43 Data Structure Index

43.1 Data Structures

Here are the data structures with brief descriptions:

44 File Index

44.1 File List

Here is a list of all documented files with brief descriptions:

45 Namespace Documentation

45.1 gsl_cgs Namespace Reference

GSL constants in CGS units.

45.1.1 Detailed Description

Note

electron, neutron, proton, and atomic mass have been updated with CODATA 2010 values. Also electron charge, gravitational constant, plancks_constant_hbar, are updated.

Variables

- const double schwarzchild_radius = 2.95325008e5
  cm
- const double speed_of_light = 2.99792458e10
  cm / s
- const double gravitational_constant = 6.67384e-8
  cm^3 / g s^2
- const double plancks_constant_h = 6.62606876e-27

g cm^2 / s

- const double plancks_constant_hbar = 1.054571726e-27
  g cm^2 / s
- const double astronomical_unit = 1.49597870691e13
  cm
- const double light_year = 9.46053620707e17
  cm
- const double parsec = 3.08567758135e18
  cm
- const double grav_accel = 9.80665e2
  cm / s^2
- const double electron_volt = 1.602176487e-12
  g cm^2 / s^2
- const double mass_electron = 9.10938291e-28
  g
- const double mass_muon = 1.88353109e-25
  g
- const double mass_proton = 1.672621777e-24
  g
- const double mass_neutron = 1.674927351e-24
  g
- const double rydberg = 2.17987190389e-11
  g cm^2 / s^2
- const double boltzmann = 1.3806503e-16
  g cm^2 / K s^2
- const double molar_gas = 8.314472e7
  g cm^2 / K mol s^2
- const double standard_gas_volume = 2.2710981e4
  cm^3 / mol
- const double minute = 6e1
  s
- const double hour = 3.6e3
  s
- const double day = 8.64e4
  s
- const double week = 6.048e5
  s
- const double inch = 2.54e0
  cm
- const double foot = 3.048e1
  cm
- const double yard = 9.144e1
  cm
- const double mile = 1.609344e5
  cm
- const double nautical_mile = 1.852e5
  cm
- const double fathom = 1.8288e2
  cm
- const double mil = 2.54e-3
  cm
- const double point = 3.52777777778e-2
  cm
- const double texpoint = 3.51459803515e-2
  cm
- const double micron = 1e-4
  cm
- const double angstrom = 1e-8
  cm
- const double hectare = 1e8
  cm^2

- const double acre = 4.04685642241e7
  cm^2
- const double barn = 1e-24
  cm^2
- const double liter = 1e3
  cm^3
- const double us_gallon = 3.78541178402e3
  cm^3
- const double quart = 9.46352946004e2
  cm^3
- const double pint = 4.73176473002e2
  cm^3
- const double cup = 2.36588236501e2
  cm^3
- const double fluid_ounce = 2.95735295626e1
  cm^3
- const double tablespoon = 1.47867647813e1
  cm^3
- const double teaspoon = 4.92892159375e0
  cm^3
- const double canadian_gallon = 4.54609e3
  cm^3
- const double uk_gallon = 4.546092e3
  cm^3
- const double miles_per_hour = 4.4704e1
  cm / s
- const double kilometers_per_hour = 2.77777777778e1
  cm / s
- const double knot = 5.14444444444e1
  cm / s
- const double pound_mass = 4.5359237e2
  g
- const double ounce_mass = 2.8349523125e1
  g
- const double ton = 9.0718474e5
  g
- const double metric_ton = 1e6
  g
- const double uk_ton = 1.0160469088e6
  g
- const double troy_ounce = 3.1103475e1
  g
- const double carat = 2e-1
  g
- const double unified_atomic_mass = 1.660538921e-24
  g
- const double gram_force = 9.80665e2
  cm g / s^2
- const double pound_force = 4.44822161526e5
  cm g / s^2
- const double kilopound_force = 4.44822161526e8
  cm g / s^2
- const double poundal = 1.38255e4
  cm g / s^2
- const double calorie = 4.1868e7
  g cm^2 / s^2
- const double btu = 1.05505585262e10
  g cm^2 / s^2
- const double therm = 1.05506e15
  g cm^2 / s^2
- const double horsepower = 7.457e9

g cm^2 / s^3

- const double bar = 1e6
  g / cm s^2
- const double std_atmosphere = 1.01325e6
  g / cm s^2
- const double torr = 1.33322368421e3
  g / cm s^2
- const double meter_of_mercury = 1.33322368421e6
  g / cm s^2
- const double inch_of_mercury = 3.38638815789e4
  g / cm s^2
- const double inch_of_water = 2.490889e3
  g / cm s^2
- const double psi = 6.89475729317e4
  g / cm s^2
- const double poise = 1e0
  g / cm s
- const double stokes = 1e0
  cm^2 / s
- const double stilb = 1e0
  cd / cm^2
- const double lumen = 1e0
  cd sr
- const double lux = 1e-4
  cd sr / cm^2
- const double phot = 1e0
  cd sr / cm^2
- const double footcandle = 1.076e-3
  cd sr / cm^2
- const double lambert = 1e0
  cd sr / cm^2
- const double footlambert = 1.07639104e-3
  cd sr / cm^2
- const double curie = 3.7e10
  1 / s
- const double roentgen = 2.58e-7
  A s / g.
- const double rad = 1e2
  cm^2 / s^2
- const double solar_mass = 1.98892e33
  g
- const double bohr_radius = 5.291772083e-9
  cm
- const double newton = 1e5
  cm g / s^2
- const double dyne = 1e0
  cm g / s^2
- const double joule = 1e7
  g cm^2 / s^2
- const double erg = 1e0
  g cm^2 / s^2
- const double stefan_boltzmann_constant = 5.67039934436e-5
  g / K^4 s^3
- const double thomson_cross_section = 6.65245853542e-25
  cm^2

45.2 gsl_cgsm Namespace Reference

GSL constants in CGSM units.

Variables

- const double schwarzchild_radius = 2.95325008e5
  cm
- const double speed_of_light = 2.99792458e10
  cm / s
- const double gravitational_constant = 6.67384e-8
  cm^3 / g s^2
- const double plancks_constant_h = 6.62606876e-27
  g cm^2 / s
- const double plancks_constant_hbar = 1.054571726e-27
  g cm^2 / s
- const double astronomical_unit = 1.49597870691e13
  cm
- const double light_year = 9.46053620707e17
  cm
- const double parsec = 3.08567758135e18
  cm
- const double grav_accel = 9.80665e2
  cm / s^2
- const double electron_volt = 1.602176487e-12
  g cm^2 / s^2
- const double mass_electron = 9.10938291e-28
  g
- const double mass_muon = 1.88353109e-25
  g
- const double mass_proton = 1.672621777e-24
  g
- const double mass_neutron = 1.674927351e-24
  g
- const double rydberg = 2.17987190389e-11
  g cm^2 / s^2
- const double boltzmann = 1.3806503e-16
  g cm^2 / K s^2
- const double electron_magnetic_moment = 9.28476362e-21
  abamp cm^2
- const double proton_magnetic_moment = 1.410606633e-23
  abamp cm^2
- const double molar_gas = 8.314472e7
  g cm^2 / K mol s^2
- const double standard_gas_volume = 2.2710981e4
  cm^3 / mol
- const double minute = 6e1
  s
- const double hour = 3.6e3
  s
- const double day = 8.64e4
  s
- const double week = 6.048e5
  s
- const double inch = 2.54e0
  cm
- const double foot = 3.048e1
  cm
- const double yard = 9.144e1
  cm
- const double mile = 1.609344e5
  cm
- const double nautical_mile = 1.852e5
  cm
- const double fathom = 1.8288e2

cm

- const double mil = 2.54e-3
  cm
- const double point = 3.52777777778e-2
  cm
- const double texpoint = 3.51459803515e-2
  cm
- const double micron = 1e-4
  cm
- const double angstrom = 1e-8
  cm
- const double hectare = 1e8
  cm^2
- const double acre = 4.04685642241e7
  cm^2
- const double barn = 1e-24
  cm^2
- const double liter = 1e3
  cm^3
- const double us_gallon = 3.78541178402e3
  cm^3
- const double quart = 9.46352946004e2
  cm^3
- const double pint = 4.73176473002e2
  cm^3
- const double cup = 2.36588236501e2
  cm^3
- const double fluid_ounce = 2.95735295626e1
  cm^3
- const double tablespoon = 1.47867647813e1
  cm^3
- const double teaspoon = 4.92892159375e0
  cm^3
- const double canadian_gallon = 4.54609e3
  cm^3
- const double uk_gallon = 4.546092e3
  cm^3
- const double miles_per_hour = 4.4704e1
  cm / s
- const double kilometers_per_hour = 2.77777777778e1
  cm / s
- const double knot = 5.14444444444e1
  cm / s
- const double pound_mass = 4.5359237e2
  g
- const double ounce_mass = 2.8349523125e1
  g
- const double ton = 9.0718474e5
  g
- const double metric_ton = 1e6
  g
- const double uk_ton = 1.0160469088e6
  g
- const double troy_ounce = 3.1103475e1
  g
- const double carat = 2e-1
  g
- const double unified_atomic_mass = 1.660538921e-24
  g
- const double gram_force = 9.80665e2
  cm g / s^2

- const double pound_force = 4.44822161526e5
  cm g / s^2
- const double kilopound_force = 4.44822161526e8
  cm g / s^2
- const double poundal = 1.38255e4
  cm g / s^2
- const double calorie = 4.1868e7
  g cm^2 / s^2
- const double btu = 1.05505585262e10
  g cm^2 / s^2
- const double therm = 1.05506e15
  g cm^2 / s^2
- const double horsepower = 7.457e9
  g cm^2 / s^3
- const double bar = 1e6
  g / cm s^2
- const double std_atmosphere = 1.01325e6
  g / cm s^2
- const double torr = 1.33322368421e3
  g / cm s^2
- const double meter_of_mercury = 1.33322368421e6
  g / cm s^2
- const double inch_of_mercury = 3.38638815789e4
  g / cm s^2
- const double inch_of_water = 2.490889e3
  g / cm s^2
- const double psi = 6.89475729317e4
  g / cm s^2
- const double poise = 1e0
  g / cm s
- const double stokes = 1e0
  cm^2 / s
- const double faraday = 9.64853429775e3
  abamp s / mol
- const double electron_charge = 1.602176565e-20
  abamp s
- const double stilb = 1e0
  cd / cm^2
- const double lumen = 1e0
  cd sr
- const double lux = 1e-4
  cd sr / cm^2
- const double phot = 1e0
  cd sr / cm^2
- const double footcandle = 1.076e-3
  cd sr / cm^2
- const double lambert = 1e0
  cd sr / cm^2
- const double footlambert = 1.07639104e-3
  cd sr / cm^2
- const double curie = 3.7e10
  1 / s
- const double roentgen = 2.58e-8
  abamp s / g
- const double rad = 1e2
  cm^2 / s^2
- const double solar_mass = 1.98892e33
  g
- const double bohr_radius = 5.291772083e-9
  cm
- const double newton = 1e5

cm g / s^2
- const double dyne = 1e0
  cm g / s^2
- const double joule = 1e7
  g cm^2 / s^2
- const double erg = 1e0
  g cm^2 / s^2
- const double stefan_boltzmann_constant = 5.67039934436e-5
  g / K^4 s^3
- const double thomson_cross_section = 6.65245853542e-25
  cm^2
- const double bohr_magneton = 9.27400899e-21
  abamp cm^2
- const double nuclear_magneton = 5.05078317e-24
  abamp cm^2

45.3 gsl_mks Namespace Reference

GSL constants in MKS units.

Variables

- const double schwarzchild_radius = 2.95325008e3
  m
- const double speed_of_light = 2.99792458e8
  m / s
- const double gravitational_constant = 6.67384e-11
  m^3 / kg s^2
- const double plancks_constant_h = 6.62606876e-34
  kg m^2 / s
- const double plancks_constant_hbar = 1.054571726e-34
  kg m^2 / s
- const double astronomical_unit = 1.49597870691e11
  m
- const double light_year = 9.46053620707e15
  m
- const double parsec = 3.08567758135e16
  m
- const double grav_accel = 9.80665e0
  m / s^2
- const double electron_volt = 1.602176487e-19
  kg m^2 / s^2
- const double mass_electron = 9.10938291e-31
  kg
- const double mass_muon = 1.88353109e-28
  kg
- const double mass_proton = 1.672621777e-27
  kg
- const double mass_neutron = 1.674927351e-27
  kg
- const double rydberg = 2.17987190389e-18
  kg m^2 / s^2
- const double boltzmann = 1.3806503e-23
  kg m^2 / K s^2
- const double bohr_magneton = 9.27400899e-24
  A m^2.
- const double nuclear_magneton = 5.05078317e-27
  A m^2.

- const double electron_magnetic_moment = 9.28476362e-24
  A m^2.
- const double proton_magnetic_moment = 1.410606633e-26
  A m^2.
- const double molar_gas = 8.314472e0
  kg m^2 / K mol s^2
- const double standard_gas_volume = 2.2710981e-2
  m^3 / mol
- const double minute = 6e1
  s
- const double hour = 3.6e3
  s
- const double day = 8.64e4
  s
- const double week = 6.048e5
  s
- const double inch = 2.54e-2
  m
- const double foot = 3.048e-1
  m
- const double yard = 9.144e-1
  m
- const double mile = 1.609344e3
  m
- const double nautical_mile = 1.852e3
  m
- const double fathom = 1.8288e0
  m
- const double mil = 2.54e-5
  m
- const double point = 3.52777777778e-4
  m
- const double texpoint = 3.51459803515e-4
  m
- const double micron = 1e-6
  m
- const double angstrom = 1e-10
  m
- const double hectare = 1e4
  m^2
- const double acre = 4.04685642241e3
  m^2
- const double barn = 1e-28
  m^2
- const double liter = 1e-3
  m^3
- const double us_gallon = 3.78541178402e-3
  m^3
- const double quart = 9.46352946004e-4
  m^3
- const double pint = 4.73176473002e-4
  m^3
- const double cup = 2.36588236501e-4
  m^3
- const double fluid_ounce = 2.95735295626e-5
  m^3
- const double tablespoon = 1.47867647813e-5
  m^3
- const double teaspoon = 4.92892159375e-6
  m^3
- const double canadian_gallon = 4.54609e-3

m^3

- const double uk_gallon = 4.546092e-3
  m^3
- const double miles_per_hour = 4.4704e-1
  m / s
- const double kilometers_per_hour = 2.77777777778e-1
  m / s
- const double knot = 5.14444444444e-1
  m / s
- const double pound_mass = 4.5359237e-1
  kg
- const double ounce_mass = 2.8349523125e-2
  kg
- const double ton = 9.0718474e2
  kg
- const double metric_ton = 1e3
  kg
- const double uk_ton = 1.0160469088e3
  kg
- const double troy_ounce = 3.1103475e-2
  kg
- const double carat = 2e-4
  kg
- const double unified_atomic_mass = 1.660538921e-27
  kg
- const double gram_force = 9.80665e-3
  kg m / s^2
- const double pound_force = 4.44822161526e0
  kg m / s^2
- const double kilopound_force = 4.44822161526e3
  kg m / s^2
- const double poundal = 1.38255e-1
  kg m / s^2
- const double calorie = 4.1868e0
  kg m^2 / s^2
- const double btu = 1.05505585262e3
  kg m^2 / s^2
- const double therm = 1.05506e8
  kg m^2 / s^2
- const double horsepower = 7.457e2
  kg m^2 / s^3
- const double bar = 1e5
  kg / m s^2
- const double std_atmosphere = 1.01325e5
  kg / m s^2
- const double torr = 1.33322368421e2
  kg / m s^2
- const double meter_of_mercury = 1.33322368421e5
  kg / m s^2
- const double inch_of_mercury = 3.38638815789e3
  kg / m s^2
- const double inch_of_water = 2.490889e2
  kg / m s^2
- const double psi = 6.89475729317e3
  kg / m s^2
- const double poise = 1e-1
  kg m^-1 s^-1
- const double stokes = 1e-4
  m^2 / s
- const double faraday = 9.64853429775e4
  A s / mol.

- const double electron_charge = 1.602176565e-19
  A s.
- const double gauss = 1e-4
  kg / A s^2
- const double stilb = 1e4
  cd / m^2
- const double lumen = 1e0
  cd sr
- const double lux = 1e0
  cd sr / m^2
- const double phot = 1e4
  cd sr / m^2
- const double footcandle = 1.076e1
  cd sr / m^2
- const double lambert = 1e4
  cd sr / m^2
- const double footlambert = 1.07639104e1
  cd sr / m^2
- const double curie = 3.7e10
  1 / s
- const double roentgen = 2.58e-4
  A s / kg.
- const double rad = 1e-2
  m^2 / s^2
- const double solar_mass = 1.98892e30
  kg
- const double bohr_radius = 5.291772083e-11
  m
- const double newton = 1e0
  kg m / s^2
- const double dyne = 1e-5
  kg m / s^2
- const double joule = 1e0
  kg m^2 / s^2
- const double erg = 1e-7
  kg m^2 / s^2
- const double stefan_boltzmann_constant = 5.67039934436e-8
  kg / K^4 s^3
- const double thomson_cross_section = 6.65245853542e-29
  m^2
- const double vacuum_permittivity = 8.854187817e-12
  A^2 s^4 / kg m^3.
- const double vacuum_permeability = 1.25663706144e-6
  kg m / A^2 s^2

45.4 gsl_mksa Namespace Reference

GSL constants in MKSA units.

Variables

- const double schwarzchild_radius = 2.95325008e3
  m
- const double speed_of_light = 2.99792458e8
  m / s
- const double gravitational_constant = 6.67384e-11
  m^3 / kg s^2
- const double plancks_constant_h = 6.62606876e-34

kg m^2 / s

- const double plancks_constant_hbar = 1.054571726e-34
  kg m^2 / s
- const double astronomical_unit = 1.49597870691e11
  m
- const double light_year = 9.46053620707e15
  m
- const double parsec = 3.08567758135e16
  m
- const double grav_accel = 9.80665e0
  m / s^2
- const double electron_volt = 1.602176487e-19
  kg m^2 / s^2
- const double mass_electron = 9.10938291e-31
  kg
- const double mass_muon = 1.88353109e-28
  kg
- const double mass_proton = 1.672621777e-27
  kg
- const double mass_neutron = 1.674927351e-27
  kg
- const double rydberg = 2.17987190389e-18
  kg m^2 / s^2
- const double boltzmann = 1.3806503e-23
  kg m^2 / K s^2
- const double bohr_magneton = 9.27400899e-24
  A m^2.
- const double nuclear_magneton = 5.05078317e-27
  A m^2.
- const double electron_magnetic_moment = 9.28476362e-24
  A m^2.
- const double proton_magnetic_moment = 1.410606633e-26
  A m^2.
- const double molar_gas = 8.314472e0
  kg m^2 / K mol s^2
- const double standard_gas_volume = 2.2710981e-2
  m^3 / mol
- const double minute = 6e1
  s
- const double hour = 3.6e3
  s
- const double day = 8.64e4
  s
- const double week = 6.048e5
  s
- const double inch = 2.54e-2
  m
- const double foot = 3.048e-1
  m
- const double yard = 9.144e-1
  m
- const double mile = 1.609344e3
  m
- const double nautical_mile = 1.852e3
  m
- const double fathom = 1.8288e0
  m
- const double mil = 2.54e-5
  m
- const double point = 3.52777777778e-4
  m

- const double texpoint = 3.51459803515e-4
  m
- const double micron = 1e-6
  m
- const double angstrom = 1e-10
  m
- const double hectare = 1e4
  m^2
- const double acre = 4.04685642241e3
  m^2
- const double barn = 1e-28
  m^2
- const double liter = 1e-3
  m^3
- const double us_gallon = 3.78541178402e-3
  m^3
- const double quart = 9.46352946004e-4
  m^3
- const double pint = 4.73176473002e-4
  m^3
- const double cup = 2.36588236501e-4
  m^3
- const double fluid_ounce = 2.95735295626e-5
  m^3
- const double tablespoon = 1.47867647813e-5
  m^3
- const double teaspoon = 4.92892159375e-6
  m^3
- const double canadian_gallon = 4.54609e-3
  m^3
- const double uk_gallon = 4.546092e-3
  m^3
- const double miles_per_hour = 4.4704e-1
  m / s
- const double kilometers_per_hour = 2.77777777778e-1
  m / s
- const double knot = 5.14444444444e-1
  m / s
- const double pound_mass = 4.5359237e-1
  kg
- const double ounce_mass = 2.8349523125e-2
  kg
- const double ton = 9.0718474e2
  kg
- const double metric_ton = 1e3
  kg
- const double uk_ton = 1.0160469088e3
  kg
- const double troy_ounce = 3.1103475e-2
  kg
- const double carat = 2e-4
  kg
- const double unified_atomic_mass = 1.660538921e-27
  kg
- const double gram_force = 9.80665e-3
  kg m / s^2
- const double pound_force = 4.44822161526e0
  kg m / s^2
- const double kilopound_force = 4.44822161526e3
  kg m / s^2
- const double poundal = 1.38255e-1

kg m / s^2

- const double calorie = 4.1868e0
  kg m^2 / s^2
- const double btu = 1.05505585262e3
  kg m^2 / s^2
- const double therm = 1.05506e8
  kg m^2 / s^2
- const double horsepower = 7.457e2
  kg m^2 / s^3
- const double bar = 1e5
  kg / m s^2
- const double std_atmosphere = 1.01325e5
  kg / m s^2
- const double torr = 1.33322368421e2
  kg / m s^2
- const double meter_of_mercury = 1.33322368421e5
  kg / m s^2
- const double inch_of_mercury = 3.38638815789e3
  kg / m s^2
- const double inch_of_water = 2.490889e2
  kg / m s^2
- const double psi = 6.89475729317e3
  kg / m s^2
- const double poise = 1e-1
  kg m^-1 s^-1
- const double stokes = 1e-4
  m^2 / s
- const double faraday = 9.64853429775e4
  A s / mol.
- const double electron_charge = 1.602176565e-19
  A s.
- const double gauss = 1e-4
  kg / A s^2
- const double stilb = 1e4
  cd / m^2
- const double lumen = 1e0
  cd sr
- const double lux = 1e0
  cd sr / m^2
- const double phot = 1e4
  cd sr / m^2
- const double footcandle = 1.076e1
  cd sr / m^2
- const double lambert = 1e4
  cd sr / m^2
- const double footlambert = 1.07639104e1
  cd sr / m^2
- const double curie = 3.7e10
  1 / s
- const double roentgen = 2.58e-4
  A s / kg.
- const double rad = 1e-2
  m^2 / s^2
- const double solar_mass = 1.98892e30
  kg
- const double bohr_radius = 5.291772083e-11
  m
- const double newton = 1e0
  kg m / s^2
- const double dyne = 1e-5
  kg m / s^2

- const double joule = 1e0
  kg m^2 / s^2
- const double erg = 1e-7
  kg m^2 / s^2
- const double stefan_boltzmann_constant = 5.67039934436e-8
  kg / K^4 s^3
- const double thomson_cross_section = 6.65245853542e-29
  m^2
- const double vacuum_permittivity = 8.854187817e-12
  A^2 s^4 / kg m^3.
- const double vacuum_permeability = 1.25663706144e-6
  kg m / A^2 s^2

45.5 gsl_num Namespace Reference

GSL numerical constants.

Variables

- const double **yotta** = 1e24
- const double **zetta** = 1e21
- const double **exa** = 1e18
- const double **peta** = 1e15
- const double **tera** = 1e12
- const double **giga** = 1e9
- const double **mega** = 1e6
- const double **kilo** = 1e3
- const double **milli** = 1e-3
- const double **micro** = 1e-6
- const double **nano** = 1e-9
- const double **pico** = 1e-12
- const double **femto** = 1e-15
- const double **atto** = 1e-18
- const double **zepto** = 1e-21
- const double **yocto** = 1e-24
- const double fine_structure = 7.2973525376e-3
  Fine structure constant (updated from `http://physics.nist.gov/cuu/Constants`*)*
- const double avogadro = 6.02214179e23
  Avogadro's number (updated from `http://physics.nist.gov/cuu/Constants`*)*

45.6 o2scl Namespace Reference

The main O$_2$scl namespace.

45.6.1 Detailed Description

By default, all O$_2$scl classes and functions which are not listed as being in one of O$_2$scl 's smaller specialized namespaces are in this namespace. This namespace has been removed from the documentation to simplify the formatting.

This namespace documentation is in the file `src/base/lib_settings.h`

45.7 o2scl_cblas Namespace Reference

Namespace for O2scl CBLAS function templates with operator[].

45.7.1 Detailed Description

Level-1 BLAS functions

Some functionality which would otherwise appear here is already given in vector.h.

- The equivalent of `dcopy()` is given in vector_copy() except that the ordering is reversed (in vector_copy() the source preceeds the destination in the function argument list).
- The equivalent of `dswap()` is given in vector_swap().
- The equivalent of `idamax()` is given in vector_max_index().

Level-2 BLAS functions

Currently only dgemv(), dtrmv(), and dtrsv() are implemented.

Level-3 BLAS functions

Currently only dgemm() is implemented.

Helper BLAS functions

There are several basic BLAS functions which are helpful to operate on only a part of a vector or matrix to ensure that the linear algebra routines are flexible with the types that they can handle.

The subvector functions operate only one set of adjacent vector elements. For a vector defined by with

$$\vec{x} = \begin{pmatrix} x_0 \\ x_1 \\ . \\ . \\ x_{\mathrm{ie}} \\ x_{\mathrm{ie}+1} \\ . \\ . \\ x_{\mathrm{N}-1} \\ x_{\mathrm{N}} \\ x_{\mathrm{N}+1} \\ . \\ . \end{pmatrix}$$

the functions with suffix `subvec` operate only on elements from x_{ie} to $x_{\mathrm{N}-1}$ (inclusive).

The subcolumn functions operate only on a part of a column of a matrix. For a matrix defined by

$$m = \begin{pmatrix} m_{0,0} & m_{0,1} & . & . & m_{0,\mathrm{ic}} & . & . \\ m_{1,0} & m_{1,1} & . & . & m_{1,\mathrm{ic}} & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & m_{\mathrm{ir},\mathrm{ic}} & . & . \\ . & . & . & . & m_{\mathrm{ir}+1,\mathrm{ic}} & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & . & . & . \\ . & . & . & . & m_{\mathrm{N}-1,\mathrm{ic}} & . & . \\ . & . & . & . & m_{\mathrm{N},\mathrm{ic}} & . & . \\ . & . & . & . & m_{\mathrm{N}+1,\mathrm{ic}} & . & . \\ . & . & . & . & . & . & . \end{pmatrix}$$

the functions with suffix `subcol` operate only on elements in the column from $m_{\mathrm{ir},\mathrm{ic}}$ to $m_{\mathrm{N}-1,\mathrm{ic}}$ inclusive.

The subrow functions operate only on a part of a row of a matrix. For a matrix defined by

$$m = \begin{pmatrix} m_{0,0} & m_{0,1} & . & . & . & . & . & . & . & . & . \\ m_{1,0} & m_{1,1} & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & . & . & . & . \\ m_{\mathrm{ir},0} & . & . & m_{\mathrm{ir,ic}} & m_{\mathrm{ir,ic}+1} & . & . & m_{\mathrm{ir,N}-1} & m_{\mathrm{ir,N}} & m_{\mathrm{ir,N}+1} & . \\ . & . & . & . & . & . & . & . & . & . & . \end{pmatrix}$$

the functions with suffix `subrow` operate only on elements in the column from $m_{\mathrm{ir,ic}}$ to $m_{\mathrm{ir,N}-1}$ inclusive.

This namespace is documented inside `src/linalg/cblas.h`.

Enumerations

- enum o2cblas_order { **o2cblas_RowMajor** = 101, **o2cblas_ColMajor** = 102 }
  Matrix order, either column-major or row-major.
- enum o2cblas_transpose { **o2cblas_NoTrans** = 111, **o2cblas_Trans** = 112, **o2cblas_ConjTrans** = 113 }
  Transpose operations.
- enum o2cblas_uplo { **o2cblas_Upper** = 121, **o2cblas_Lower** = 122 }
  Upper- or lower-triangular.
- enum o2cblas_diag { **o2cblas_NonUnit** = 131, **o2cblas_Unit** = 132 }
  Unit or generic diagonal.
- enum o2cblas_side { **o2cblas_Left** = 141, **o2cblas_Right** = 142 }
  Left or right sided operation.

Functions

Level-1 BLAS functions

- template<class vec_t >
  double dasum (const size_t N, const vec_t &X)
  Compute the absolute sum of vector elements.
- template<class vec_t , class vec2_t >
  void daxpy (const double alpha, const size_t N, const vec_t &X, vec2_t &Y)
  Compute $y = \alpha x + y$.
- template<class vec_t , class vec2_t >
  double ddot (const size_t N, const vec_t &X, const vec2_t &Y)
  Compute $r = x \cdot y$.
- template<class vec_t >
  double dnrm2 (const size_t N, const vec_t &X)
  Compute the norm of the vector X.
- template<class vec_t >
  void dscal (const double alpha, const size_t N, vec_t &X)
  Compute $x = \alpha x$.

Level-2 BLAS functions

- template<class mat_t , class vec_t >
  void dgemv (const enum o2cblas_order order, const enum o2cblas_transpose TransA, const size_t M, const size_t N, const double alpha, const mat_t &A, const vec_t &X, const double beta, vec_t &Y)
  Compute $y = \alpha [\mathrm{op}(A)] x + \beta y$.
- template<class mat_t , class vec_t >
  void dtrsv (const enum o2cblas_order order, const enum o2cblas_uplo Uplo, const enum o2cblas_transpose TransA, const enum o2cblas_diag Diag, const size_t M, const size_t N, const mat_t &A, vec_t &X)
  Compute $x = \mathrm{op}(A)^{-1} x$.
- template<class mat_t , class vec_t >
  void dtrmv (const enum o2cblas_order Order, const enum o2cblas_uplo Uplo, const enum o2cblas_transpose TransA, const enum o2cblas_diag Diag, const size_t N, const mat_t &A, vec_t &x)
  Compute $x = op(A)x$ for the triangular matrix A.

Level-3 BLAS functions

- template<class mat_t >
  void dgemm (const enum o2cblas_order Order, const enum o2cblas_transpose TransA, const enum o2cblas_transpose - TransB, const size_t M, const size_t N, const size_t K, const double alpha, const mat_t &A, const mat_t &B, const double beta, mat_t &C)
  Compute $y = \alpha \mathrm{op}(A)\mathrm{op}(B) + \beta C$.

Helper BLAS functions - Subvectors

- template<class vec_t , class vec2_t >
  void daxpy_subvec (const double alpha, const size_t N, const vec_t &X, vec2_t &Y, const size_t ie)
  Compute $y = \alpha x + y$ *beginning with index* `ie` *and ending with index* `N-1`.
- template<class vec_t , class vec2_t >
  double ddot_subvec (const size_t N, const vec_t &X, const vec2_t &Y, const size_t ie)
  Compute $r = x \cdot y$ *beginning with index* `ie` *and ending with index* `N-1`.
- template<class vec_t >
  double dnrm2_subvec (const size_t N, const vec_t &X, const size_t ie)
  Compute the norm of the vector `X` *beginning with index* `ie` *and ending with index* `N-1`.
- template<class vec_t >
  void dscal_subvec (const double alpha, const size_t N, vec_t &X, const size_t ie)
  Compute $x = \alpha x$ *beginning with index* `ie` *and ending with index* `N-1`.

Helper BLAS functions - Subcolums of a matrix

- template<class mat_t , class vec_t >
  void daxpy_subcol (const double alpha, const size_t M, const mat_t &X, const size_t ir, const size_t ic, vec_t &y)
  Compute $y = \alpha x + y$ *for a subcolumn of a matrix.*
- template<class mat_t , class vec_t >
  double ddot_subcol (const size_t M, const mat_t &X, const size_t ir, const size_t ic, const vec_t &y)
  Compute $r = x \cdot y$ *for a subcolumn of a matrix.*
- template<class mat_t >
  double dnrm2_subcol (const mat_t &A, const size_t ir, const size_t ic, const size_t M)
  Compute the norm of a subcolumn of a matrix.
- template<class mat_t >
  void dscal_subcol (mat_t &A, const size_t ir, const size_t ic, const size_t M, const double alpha)
  Compute $x = \alpha x$ *for a subcolumn of a matrix.*

Helper BLAS functions - Subrows of a matrix

- template<class mat_t , class vec_t >
  void daxpy_subrow (const double alpha, const size_t N, const mat_t &X, const size_t ir, const size_t ic, vec_t &Y)
  Compute $y = \alpha x + y$ *for a subrow of a matrix.*
- template<class mat_t , class vec_t >
  double ddot_subrow (const size_t N, const mat_t &X, const size_t ir, const size_t ic, const vec_t &Y)
  Compute $r = x \cdot y$ *for a subrow of a matrix.*
- template<class mat_t >
  double dnrm2_subrow (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
  Compute the norm of a subrow of a matrix.
- template<class mat_t >
  void dscal_subrow (mat_t &A, const size_t ir, const size_t ic, const size_t N, const double alpha)
  Compute $x = \alpha x$ *for a subrow of a matrix.*

45.7.2 Function Documentation

45.7.2.1 template<class vec_t > double o2scl_cblas::dasum (const size_t *N*, const vec_t & *X*)

If `alpha` is zero, this function returns and performs no computations.

Definition at line 84 of file cblas_base.h.

45.7.2.2 template<class vec_t , class vec2_t > void o2scl_cblas::daxpy (const double *alpha*, const size_t *N*, const vec_t & *X*, vec2_t & *Y*)

If `alpha` is zero, this function returns and performs no computations.

Definition at line 98 of file cblas_base.h.

45.7.2.3 template<class vec_t > double o2scl_cblas::dnrm2 (const size_t *N*, const vec_t & *X*)

Note

The suffix "2" on the function name indicates that this computes the "2-norm", not that the norm is squared.

If `N` is less than or equal to zero, this function returns zero without calling the error handler.

This function works only with vectors which hold `double`. For the norm of a general floating point vector, see vector_norm().

Definition at line 156 of file cblas_base.h.

45.7.2.4 template<class mat_t , class vec_t > void o2scl_cblas::dgemv (const enum o2cblas_order *order*, const enum o2cblas_transpose *TransA*, const size_t *M*, const size_t *N*, const double *alpha*, const mat_t & *A*, const vec_t & *X*, const double *beta*, vec_t & *Y*)

If `M` or `N` is zero, or if `alpha` is zero and `beta` is one, this function performs no calculations and returns without calling the error handler.

Definition at line 218 of file cblas_base.h.

45.7.2.5 template<class mat_t , class vec_t > void o2scl_cblas::dtrsv (const enum o2cblas_order *order*, const enum o2cblas_uplo *Uplo*, const enum o2cblas_transpose *TransA*, const enum o2cblas_diag *Diag*, const size_t *M*, const size_t *N*, const mat_t & *A*, vec_t & *X*)

If `N` is zero, this function does nothing and returns zero.

Definition at line 308 of file cblas_base.h.

45.7.2.6 template<class vec_t , class vec2_t > void o2scl_cblas::daxpy_subvec (const double *alpha*, const size_t *N*, const vec_t & *X*, vec2_t & *Y*, const size_t *ie*)

This function is used in householder_hv().

If `alpha` is identical with zero or `N==ie`, this function will perform no calculations and return without calling the error handler.

If `ie` is greater than `N-1` then the error handler will be called if `O2SCL_NO_RANGE_CHECK` is not defined.

Definition at line 797 of file cblas_base.h.

45.7.2.7 template<class vec_t , class vec2_t > double o2scl_cblas::ddot_subvec (const size_t *N*, const vec_t & *X*, const vec2_t & *Y*, const size_t *ie*)

This function is used in householder_hv().

If `ie` is greater than `N-1` then the error handler will be called if `O2SCL_NO_RANGE_CHECK` is not defined.

Definition at line 834 of file cblas_base.h.

45.7.2.8 template<class vec_t > double o2scl_cblas::dnrm2_subvec (const size_t *N*, const vec_t & *X*, const size_t *ie*)

Used in householder_transform().

Note

The suffix "2" on the function name indicates that this computes the "2-norm", not that the norm is squared.

If `ie` is greater than `N-1` then the error handler will be called if `O2SCL_NO_RANGE_CHECK` is not defined.

Definition at line 875 of file cblas_base.h.

45.7.2.9 template<class vec_t > void o2scl_cblas::dscal_subvec (const double *alpha*, const size_t *N*, vec_t & *X*, const size_t *ie*)

This function is used in householder_transform().

If `ie` is greater than `N-1` then the error handler will be called if `O2SCL_NO_RANGE_CHECK` is not defined.

Definition at line 920 of file cblas_base.h.

45.7.2.10 template<class mat_t , class vec_t > void o2scl_cblas::daxpy_subcol (const double *alpha*, const size_t *M*, const mat_t & *X*, const size_t *ir*, const size_t *ic*, vec_t & *y*)

Given the matrix `X`, define the vector `x` as the column with index `ic`. This function computes $y = \alpha x + y$ for elements in the vectors `x` and `y` from row `ir` to row `M-1` (inclusive). All other elements in `x` and `y` are not referenced.

Used in householder_hv_sub().

Definition at line 959 of file cblas_base.h.

45.7.2.11 template<class mat_t , class vec_t > double o2scl_cblas::ddot_subcol (const size_t *M*, const mat_t & *X*, const size_t *ir*, const size_t *ic*, const vec_t & *y*)

Given the matrix `X`, define the vector `x` as the column with index `ic`. This function computes $r = x \cdot y$ for elements in the vectors `x` and `y` from row `ir` to row `M-1` (inclusive). All other elements in `x` and `y` are not referenced.

Used in householder_hv_sub().

Definition at line 1001 of file cblas_base.h.

45.7.2.12 template<class mat_t > double o2scl_cblas::dnrm2_subcol (const mat_t & *A*, const size_t *ir*, const size_t *ic*, const size_t *M*)

Given the matrix `A`, define the vector `x` as the column with index `ic`. This function computes the norm of the part of `x` from row `ir` to row `M-1` (inclusive). All other elements in `x` are not referenced.

if `M` is zero, then this function silently returns zero without calling the error handler.

This function is used in householder_transform_subcol().

Note

The suffix "2" on the function name indicates that this computes the "2-norm", not that the norm is squared.

Definition at line 1044 of file cblas_base.h.

45.7.2.13 template<class mat_t > void o2scl_cblas::dscal_subcol (mat_t & *A*, const size_t *ir*, const size_t *ic*, const size_t *M*, const double *alpha*)

Given the matrix `A`, define the vector `x` as the column with index `ic`. This function computes $x = \alpha x$ for elements in the vectors `x` from row `ir` to row `M-1` (inclusive). All other elements in `x` are not referenced.

Used in householder_transform_subcol().

Definition at line 1093 of file cblas_base.h.

45.7.2.14 template<class mat_t , class vec_t > void o2scl_cblas::daxpy_subrow (const double *alpha*, const size_t *N*, const mat_t & *X*, const size_t *ir*, const size_t *ic*, vec_t & *Y*)

Given the matrix `X`, define the vector `x` as the row with index `ir`. This function computes $y = \alpha x + y$ for elements in the vectors `x` from column `ic` to column `N-1` (inclusive). All other elements in `x` and `y` are not referenced.

If `ic` is greater than `N-1` then the error handler will be called if `O2SCL_NO_RANGE_CHECK` is not defined.

Used in householder_hv_sub().

Definition at line 1138 of file cblas_base.h.

45.7.2.15 template<class mat_t , class vec_t > double o2scl_cblas::ddot_subrow (const size_t *N*, const mat_t & *X*, const size_t *ir*, const size_t *ic*, const vec_t & *Y*)

Given the matrix `X`, define the vector `x` as the row with index `ir`. This function computes $r = x \cdot y$ for elements in the vectors `x` from column `ic` to column `N-1` (inclusive). All other elements in `x` and `y` are not referenced.

If `ic` is greater than `N-1` then the error handler will be called if `O2SCL_NO_RANGE_CHECK` is not defined.

Used in householder_hv_sub().

Definition at line 1184 of file cblas_base.h.

45.7.2.16 template<class mat_t > double o2scl_cblas::dnrm2_subrow (const mat_t & *M*, const size_t *ir*, const size_t *ic*, const size_t *N*)

Given the matrix `X`, define the vector `x` as the row with index `ir`. This function computes the norm of the part of `x` from column `ic` to column `N-1` (inclusive). All other elements in `x` are not referenced.

Note

The suffix "2" on the function name indicates that this computes the "2-norm", not that the norm is squared.

Definition at line 1223 of file cblas_base.h.

45.7.2.17 template<class mat_t > void o2scl_cblas::dscal_subrow (mat_t & *A*, const size_t *ir*, const size_t *ic*, const size_t *N*, const double *alpha*)

Given the matrix `A`, define the vector `x` as the row with index `ir`. This function computes $x = \alpha x$ for elements in the vectors `x` from column `ic` to column `N-1` (inclusive). All other elements in `x` and `y` are not referenced.

If `ic` is greater than `N-1` then the error handler will be called if `O2SCL_NO_RANGE_CHECK` is not defined.

Definition at line 1266 of file cblas_base.h.

45.8 o2scl_cblas_paren Namespace Reference

Namespace for O2scl CBLAS function templates with operator()

45.8.1 Detailed Description

This namespace contains an identical copy of all the functions given in the o2scl_cblas namespace, but perform array indexing with `operator()` rather than `operator[]`. See o2scl_cblas for the function listing and documentation.

45.9 o2scl_const Namespace Reference

O2scl constants.

Variables

- const double **gsl_posinf** = GSL_POSINF
- const double **gsl_neginf** = GSL_NEGINF
- const double pi = acos(-1.0)
  π
- const double pi2 = pi∗pi
  π^2
- const double zeta32 = 2.6123753486854883433
  $\zeta(3/2)$
- const double zeta2 = 1.6449340668482264365
  $\zeta(2)$

- const double zeta52 = 1.3414872572509171798
  $\zeta(5/2)$
- const double zeta3 = 1.2020569031595942854
  $\zeta(3)$
- const double zeta5 = 1.0369277551433699263
  $\zeta(5)$
- const double zeta7 = 1.0083492773819228268
  $\zeta(7)$

Particle Physics Booklet

(see also D.E. Groom, et. al., Euro. Phys. J. C 15 (2000) 1.)

- const double sin2_theta_weak = 0.2224
  $\sin^2\theta_W$
- const double mev_kg = 1.782661731e-30
  1 MeV in kg
- const double mev_cgs = 1.60217733e-6
  1 MeV in $g\cdot cm^2/s^2$ *(ergs)*
- const double boltzmann_mev_K = 8.617342e-11
  1 MeV in Kelvin

From http://physics.nist.gov/cuu/Constants (7/27/11)

- const double hc_mev_fm = 197.3269718
  ħc in MeV fm
- const double gfermi_gev = 1.166364e-5
  Fermi coupling constant (G_F *) in* GeV^{-2}.
- const double hc_mev_cm = 1.973269718e-11
  ħc in MeV cm

Squared electron charge

- const double e2_gaussian = o2scl_const::hc_mev_fm∗gsl_num::fine_structure
  Electron charge squared in Gaussian units.
- const double e2_hlorentz = gsl_num::fine_structure∗4.0∗pi
  Electron charge sqaured in Heaviside-Lorentz units where $\hbar = c = 1$.
- const double e2_mksa = gsl_mksa::electron_charge
  Electron charge squared in SI(MKSA) units.

45.9.1 Variable Documentation

45.9.1.1 const double o2scl_const::e2_gaussian = o2scl_const::hc_mev_fm∗gsl_num::fine_structure

In Gaussian Units:

$$\vec{\nabla}\cdot\vec{E} = 4\pi\rho\,, \quad \vec{E} = -\vec{\nabla}\Phi, \quad \nabla^2\Phi = -4\pi\rho\,,$$
$$F = \frac{q_1 q_2}{r^2}\,, \quad W = \frac{1}{2}\int \rho V d^3x = \frac{1}{8\pi}\int |\vec{E}|^2 d^3x, \quad \alpha = \frac{e^2}{\hbar c} = \frac{1}{137}$$

Definition at line 952 of file constants.h.

45.9.1.2 const double o2scl_const::e2_hlorentz = gsl_num::fine_structure∗4.0∗pi

In Heaviside-Lorentz units:

$$\vec{\nabla}\cdot\vec{E} = \rho\,, \quad \vec{E} = -\vec{\nabla}\Phi, \quad \nabla^2\Phi = -\rho\,,$$
$$F = \frac{q_1 q_2}{4\pi r^2}\,, \quad W = \frac{1}{2}\int \rho V d^3x = \frac{1}{2}\int |\vec{E}|^2 d^3x, \quad \alpha = \frac{e^2}{4\pi} = \frac{1}{137}$$

Definition at line 972 of file constants.h.

45.9.1.3 const double o2scl_const::e2_mksa = gsl_mksa::electron_charge

In MKSA units:

$$\vec{\nabla}\cdot\vec{E}=\rho\,,\quad \vec{E}=-\vec{\nabla}\Phi\,,\quad \nabla^2\Phi=-\rho\,,$$
$$F=\frac{1}{4\pi\varepsilon_0}\frac{q_1q_2}{r^2}\,,\quad W=\frac{1}{2}\int\rho V d^3x=\frac{\varepsilon_0}{2}\int|\vec{E}|^2d^3x\,,\quad \alpha=\frac{e^2}{4\pi\varepsilon_0\hbar c}=\frac{1}{137}$$

Note the conversion formulas

$$q_H L=\sqrt{4\pi}q_G=\frac{1}{\sqrt{\varepsilon_0}}q_{SI}$$

as mentioned in pg. 13 of D. Griffiths Intro to Elem. Particles.

Definition at line 997 of file constants.h.

45.10 o2scl_fm Namespace Reference

Constants in units of fm.

45.10.1 Detailed Description

In nuclear physics is frequently convenient to work in units of fm with $\hbar=c=k_B=1$. Several useful constants are given here.

For example, mev gives 1 MeV in units of fm^{-1} (the solution to the equation $1\mathrm{MeV}=x\ \mathrm{fm}^{-1}$). If you have a number in MeV, you can multiply by mev to get a number in units of fm^{-1}. Alternatively, mev is a number with units $\mathrm{MeV}^{-1}\cdot\mathrm{fm}^{-1}$. These can be combined, so that erg divided by sec is 1 erg/sec in units of fm^{-2}.

/todo Make e.g. neutron mass consistent with the GSL ones somehow.

Variables

- const double mev = 1.0/o2scl_const::hc_mev_fm
  1 MeV *in* fm^{-1}
- const double kg = mev∗5.60958885e29
  1 kg *in* fm^{-1}
- const double Kelvin = 8.6173324e-11∗mev
  1 Kelvin *in* fm^{-1}
- const double joule = kg/gsl_mks::speed_of_light/gsl_mks::speed_of_light
  1 Joule *in* fm^{-1}
- const double msun_per_km3 = gsl_mks::solar_mass/1.0e54∗kg
  1 $M_\odot/km^3$ *in* fm^{-4}
- const double erg = kg/1.0e3/gsl_cgs::speed_of_light/gsl_cgs::speed_of_light
  1 erg *in* fm^{-1}
- const double sec = gsl_mks::speed_of_light∗1.0e15
  1 second *in* fm
- const double ec_gauss_fm2
  1 Gauss *times the electron charge in Gaussian units in* fm^{-2}
- const double gauss2_fm4 = ec_gauss_fm2∗ec_gauss_fm2/gsl_num::fine_structure
  Conversion factor from Gauss^2 *to* fm^{-4} *in Gaussian units.*

From http://physics.nist.gov/cuu/index.html (7/27/11)

- const double mass_neutron = 939.565379/o2scl_const::hc_mev_fm
  Neutron mass in fm^{-1}.
- const double mass_proton = 938.272046/o2scl_const::hc_mev_fm
  Proton mass in fm^{-1}.
- const double mass_electron = 0.510998928/o2scl_const::hc_mev_fm
  Electron mass in fm^{-1}.

- const double mass_muon = 105.6583715/o2scl_const::hc_mev_fm
  Muon mass in fm^{-1}.
- const double mass_amu = 931.494061/o2scl_const::hc_mev_fm
  Atomic mass unit in fm^{-1}.
- const double mass_alpha = 3727.379240/o2scl_const::hc_mev_fm
  Alpha particle mass in fm^{-1}.

Masses from Particle Physics Booklet

(see also D.E. Groom, et. al., Euro. Phys. J. C 15 (2000) 1.)

- const double mass_lambda = 1115.683/o2scl_const::hc_mev_fm
  Λ *mass in* fm^{-1}
- const double mass_sigmam = 1197.45/o2scl_const::hc_mev_fm
  Σ^- *mass in* fm^{-1}
- const double mass_sigma = 1192.642/o2scl_const::hc_mev_fm
  Σ^0 *mass in* fm^{-1}
- const double mass_sigmap = 1189.37/o2scl_const::hc_mev_fm
  Σ^+ *mass in* fm^{-1}
- const double mass_cascadem = 1321.3/o2scl_const::hc_mev_fm
  Ξ^- *mass in* fm^{-1}
- const double mass_cascade = 1314.8/o2scl_const::hc_mev_fm
  Ξ^0 *mass in* fm^{-1}
- const double mass_omega = 782.57/o2scl_const::hc_mev_fm
  ω *mass in* fm^{-1}
- const double mass_rho = 769.3/o2scl_const::hc_mev_fm
  ρ *mass in* fm^{-1}

45.10.2 Variable Documentation

45.10.2.1 const double o2scl_fm::mass_alpha = 3727.379240/o2scl_const::hc_mev_fm

This does not include the mass of the additional two electrons which are present in a helium atom.

Definition at line 1040 of file constants.h.

45.10.2.2 const double o2scl_fm::ec_gauss_fm2

Initial value:

```
gsl_mks::electron_charge*1.0e-34/
    gsl_mks::plancks_constant_hbar
```

Definition at line 1091 of file constants.h.

45.10.2.3 const double o2scl_fm::gauss2_fm4 = ec_gauss_fm2*ec_gauss_fm2/gsl_num::fine_structure

This is useful, e.g. in converting magnetic field squared to an energy density.

Definition at line 1100 of file constants.h.

45.11 o2scl_graph Namespace Reference

Namespace for experimental functions for use with Root.

45.11.1 Detailed Description

This namespace contains several functions, defined in graph.cpp, which simplify plotting with Root. They require that the Root libraries have been installed and are not compiled in with the normal O_2scl libraries. Some of the example plots created for the documentation are based on these functions.

Functions

- void arrow (double x1, double y1, double x2, double y2, TLine *&line, TPolyLine *&poly, double size=0.1, double size2=0.8, double alpha1=0.35)
  Draw a pretty arrow from (x1,y1) to (x2,y2)
- int new_graph (TCanvas *&c1, TPad *&p1, TH1 *&th1, std::string CanvasName, std::string WindowName, std::string PadName, double lleft, double lbottom, double lright, double ltop, int left=0, int top=0, int right=700, int bottom=700, bool logx=false, bool logy=false)
  Make a canvas and pad.
- int new_graph_ticks (TCanvas *&c1, TPad *&p1, TH1 *&th1, std::string CanvasName, std::string WindowName, std::string PadName, double lleft, double lbottom, double lright, double ltop, TGaxis *&a1, TGaxis *&a2, int left=0, int top=0, int right=700, int bottom=700, bool logx=false, bool logy=false)
  Make a canvas and pad with more tick marks.
- TGraph * table_graph (o2scl::table_units &at, std::string scolx, std::string scoly, int style=1, int color=1)
  Graph two columns from a data table.
- TGraphErrors * table_graph_errors (o2scl::table_units &at, std::string scolx, std::string scoly, std::string xerr, std::string yerr, int style=1, int color=1)
  Plot colums from a data table with error bars.
- int two_up_graph (TCanvas *&c1, TPad *&p1, TPad *&p2, TH1 *&th1, TH1 *&th2, std::string CanvasName, std::string WindowName, std::string Pad1Name, std::string Pad2Name, double lowx1, double highx1, double lowx2, double highx2, double lowy, double highy, int left=0, int top=0, int right=1000, int bottom=700, bool logx1=false, bool logx2=false, bool logy=false, double alpha=1.3, double margin=0.1)
  Make a canvas with a two-up graph, side-by-side.
- int two_up_graphy (TCanvas *&c1, TPad *&p1, TPad *&p2, TH1 *&th1, TH1 *&th2, std::string CanvasName, std::string WindowName, std::string Pad1Name, std::string Pad2Name, double lowx, double highx, double lowy1, double highy1, double lowy2, double highy2, int left=0, int top=0, int right=1000, int bottom=700, bool logx=false, bool logy1=false, bool logy2=false, double alpha=1.3, double margin=0.1)
  Make a canvas with two plots, one on top of the other.

Variables

Some useful colors

- const int **kGray20** = kGray
- const int **kGray40** = kGray+1
- const int **kGray60** = kGray+2
- const int **kGray80** = kGray+3

Markers

- const int **m_small_dot** = 1
- const int **m_plus** = 2
- const int **m_asterisk** = 3
- const int **m_circle** = 4
- const int **m_times** = 5
- const int **m_med_dot** = 6
- const int **m_large_dot** = 7
- const int **m_fill_circle** = 8
- const int **m_fill_square** = 21
- const int **m_fill_up_triangle** = 22
- const int **m_fill_dn_triangle** = 23
- const int **m_open_circle** = 24
- const int **m_open_square** = 25
- const int **m_open_up_triangle** = 26
- const int **m_open_diamond** = 27
- const int **m_open_plus** = 28
- const int **m_fill_star** = 29
- const int **m_open_star** = 30
- const int **m_asterisk2** = 31
- const int **m_open_dn_triangle** = 32
- const int **m_fill_diamond** = 33
- const int **m_fill_plus** = 34

45.11.2 Function Documentation

45.11.2.1 void o2scl_graph::arrow (double *x1*, double *y1*, double *x2*, double *y2*, TLine *& *line*, TPolyLine *& *poly*, double *size* = 0.1, double *size2* = 0.8, double *alpha1* = 0.35)

The parameter `size` determines the size of the head relative to the size of the arrow, `size2` determines the position of the niche (1.0 is no niche), and alpha1 determines the angle of the head relative to the line (default is about 20 degrees).

45.11.2.2 int o2scl_graph::new_graph (TCanvas *& *c1*, TPad *& *p1*, TH1 *& *th1*, std::string *CanvasName*, std::string *WindowName*, std::string *PadName*, double *lleft*, double *lbottom*, double *lright*, double *ltop*, int *left* = 0, int *top* = 0, int *right* = 700, int *bottom* = 700, bool *logx* = false, bool *logy* = false)

The CanvasName is the default output filename for graphs and macros also. The PadName is the name that comes up when you right-click on the pad. When making .eps files, the Title field is set to "CanvasName.eps: Windowname". This Title field is the title that appears, e.g., in a ghostview window. The x-axis limits are given by `lleft` and `lright` and the y-axis limits are given in `lbottom` and `ltop`. The upper left corner of the canvas window is at `(left,top)` and the lower right corner is `(right,bottom)`. The parameters `logx` and `logy` determine whether or not the x- or y-axes are logarithmic instead of linear (the default). The canvas and pad objects are created using `new`, and the histogram object `th1` is created with TPad::DrawFrame().

45.11.2.3 int o2scl_graph::new_graph_ticks (TCanvas *& *c1*, TPad *& *p1*, TH1 *& *th1*, std::string *CanvasName*, std::string *WindowName*, std::string *PadName*, double *lleft*, double *lbottom*, double *lright*, double *ltop*, TGaxis *& *a1*, TGaxis *& *a2*, int *left* = 0, int *top* = 0, int *right* = 700, int *bottom* = 700, bool *logx* = false, bool *logy* = false)

This is the same as new_graph(), except it adds tick marks to the right and top axes, which are created with `new` and returned in `a1` and `a2` respectively().

Idea for Future Modify this to just directly call new_graph() and add the new axes on top.

45.11.2.4 TGraph* o2scl_graph::table_graph (o2scl::table_units & *at*, std::string *scolx*, std::string *scoly*, int *style* = 1, int *color* = 1)

This function plots the function defined by `(scolx,scoly)` given in table `at` with line style `style` and line color `color`. A `TGraph` object created with `new` is returned. The `name` of the graph object is automatically set to be the same as the `scoly`, but this can always be changed afterwards, i.e.

```
g1=table_graph(at,scolx,scoly);
g1->SetName("Curve name");
g1->Draw();
```

45.11.2.5 int o2scl_graph::two_up_graph (TCanvas *& *c1*, TPad *& *p1*, TPad *& *p2*, TH1 *& *th1*, TH1 *& *th2*, std::string *CanvasName*, std::string *WindowName*, std::string *Pad1Name*, std::string *Pad2Name*, double *lowx1*, double *highx1*, double *lowx2*, double *highx2*, double *lowy*, double *highy*, int *left* = 0, int *top* = 0, int *right* = 1000, int *bottom* = 700, bool *logx1* = false, bool *logx2* = false, bool *logy* = false, double *alpha* = 1.3, double *margin* = 0.1)

This function creates a canvas and two pads for a side-by-side plot with two different x-axes and the same y-axis. The x-axis of the left-hand plot has limits `(lowx1,highx1)` and the the x-axis for the right-hand plot has limits `(lowx2,highx2)`.

45.11.2.6 int o2scl_graph::two_up_graphy (TCanvas *& *c1*, TPad *& *p1*, TPad *& *p2*, TH1 *& *th1*, TH1 *& *th2*, std::string *CanvasName*, std::string *WindowName*, std::string *Pad1Name*, std::string *Pad2Name*, double *lowx*, double *highx*, double *lowy1*, double *highy1*, double *lowy2*, double *highy2*, int *left* = 0, int *top* = 0, int *right* = 1000, int *bottom* = 700, bool *logx* = false, bool *logy1* = false, bool *logy2* = false, double *alpha* = 1.3, double *margin* = 0.1)

The variable `p1` is on bottom and `p2` is on top

45.12 o2scl_inte_gk_coeffs Namespace Reference

A namespace for the GSL adaptive Gauss-Kronrod integration coefficients.

45.12.1 Detailed Description

The storage scheme follows that of QUADPACK: symmetry about the origin permits storing only those abscissae in the interval $[0,1)$. Similiarly for the weights. The odd-indexed abscissae `xgk`[1], `xgk`[3],... belong to the low-order Gauss rule. For the full Gauss-Kronrod rule, the array `wgk`[] holds the weights corresponding to the abscissae `xgk`[]. For the low-order rule, `wg`[i] is the weight correpsonding to `xgk`[2∗i+1].

Documentation from GSL:

Gauss quadrature weights and kronrod quadrature abscissae and weights as evaluated with 80 decimal digit arithmetic by L. W. Fullerton, Bell Labs, Nov. 1981.

Variables

- static const double qk15_xgk [8]
  Abscissae of the 15-point Kronrod rule.
- static const double qk15_wg [4]
  Weights of the 7-point Gauss rule.
- static const double qk15_wgk [8]
  Weights of the 15-point Kronrod rule.
- static const double qk21_xgk [11]
  Abscissae of the 21-point Kronrod rule.
- static const double qk21_wg [5]
  Weights of the 10-point Gauss rule.
- static const double qk21_wgk [11]
  Weights of the 21-point Kronrod rule.
- static const double qk31_xgk [16]
  Abscissae of the 31-point Kronrod rule.
- static const double qk31_wg [8]
  Weights of the 15-point Gauss rule.
- static const double qk31_wgk [16]
  Weights of the 31-point Kronrod rule.
- static const double qk41_xgk [21]
  Abscissae of the 41-point Kronrod rule.
- static const double qk41_wg [11]
  Weights of the 20-point Gauss rule.
- static const double qk41_wgk [21]
  Weights of the 41-point Kronrod rule.
- static const double qk51_xgk [26]
  Abscissae of the 51-point Kronrod rule.
- static const double qk51_wg [13]
  Weights of the 25-point Gauss rule.
- static const double qk51_wgk [26]
  Weights of the 51-point Kronrod rule.
- static const double qk61_xgk [31]
  Abscissae of the 61-point Kronrod rule.
- static const double qk61_wg [15]
  Weights of the 30-point Gauss rule.
- static const double qk61_wgk [31]
  Weights of the 61-point Kronrod rule.

45.12.2 Variable Documentation

45.12.2.1 const double o2scl_inte_gk_coeffs::qk15_xgk[8] [static]

Initial value:

```
{
    0.991455371120812639206854697526329,
    0.949107912342758524526189684047851,
    0.864864423359769072789712788640926,
    0.741531185599394439863864773280788,
    0.586087235467691130294144838258730,
    0.405845151377397166906606412076961,
    0.207784955007898467600689403773245,
    0.000000000000000000000000000000000
  }
```

Definition at line 53 of file gsl_inte_kronrod.h.

45.12.2.2 const double o2scl_inte_gk_coeffs::qk15_wg[4] [static]

Initial value:

```
    {
      0.129484966168869693270611432679082,
      0.279705391489276667901467771423780,
      0.381830050505118944950369775488975,
      0.417959183673469387755102040816327
    }
```

Definition at line 65 of file gsl_inte_kronrod.h.

45.12.2.3 const double o2scl_inte_gk_coeffs::qk15_wgk[8] [static]

Initial value:

```
    {
      0.022935322010529224963732008058970,
      0.063092092629978553290700663189204,
      0.104790010322250183839876322541518,
      0.140653259715525918745189590510238,
      0.169004726639267902826583426598550,
      0.190350578064785409913256402421014,
      0.204432940075298892414161999234649,
      0.209482141084727828012999174891714
    }
```

Definition at line 74 of file gsl_inte_kronrod.h.

45.12.2.4 const double o2scl_inte_gk_coeffs::qk21_xgk[11] [static]

Initial value:

```
    {
      0.995657163025808080735527280689003,
      0.973906528517171720077964012084452,
      0.930157491355708226001207180059508,
      0.865063366688984510732096688423493,
      0.780817726586416897063717578345042,
      0.679409568299024406234327365114874,
      0.562757134668604683339000099272694,
      0.433395394129247190799265943165784,
      0.294392862701460198131126603103866,
      0.148874338981631210884826001129720,
      0.000000000000000000000000000000000
    }
```

Definition at line 87 of file gsl_inte_kronrod.h.

45.12.2.5 const double o2scl_inte_gk_coeffs::qk21_wg[5] [static]

Initial value:

```
{
  0.06667134430868813759356880989332,
  0.14945134915058059314577633965769 7,
  0.21908636251598204399553493422816 3,
  0.26926671930999635509122692156946 9,
  0.29552422471475287017389299465133 8
}
```

Definition at line 103 of file gsl_inte_kronrod.h.

45.12.2.6 const double o2scl_inte_gk_coeffs::qk21_wgk[11] [static]

Initial value:

```
{
  0.01169463886737187427806439606219 2,
  0.03255816230796472747881897245939 0,
  0.05475589657435199603138130024458 0,
  0.07503967481091995276704314091619 0,
  0.09312545458369760553506546508336 6,
  0.10938715880229764189921059032580 5,
  0.12349197626206585107795810983107 4,
  0.13470921731147332592805400177170 7,
  0.14277593857706008079709427313871 7,
  0.14773910490133849137484151597206 8,
  0.14944555400291690566493646838982 1
}
```

Definition at line 113 of file gsl_inte_kronrod.h.

45.12.2.7 const double o2scl_inte_gk_coeffs::qk31_xgk[16] [static]

Initial value:

```
{
  0.99800229869339706028517284015227 1,
  0.98799251802048542848956571858661 3,
  0.96773907567913913425734797878433 7,
  0.93727339240070590430775894771020 9,
  0.89726453234408190088250965645449 6,
  0.84820658341042721620064832077421 7,
  0.79041850144246593296764929481794 7,
  0.72441773136017004741618605461393 8,
  0.65099674129741697053373589531327 5,
  0.57097217260853884753722673725391 1,
  0.48508186364023968069365574023235 1,
  0.39415134707756336989720737098104 5,
  0.29918000715316881216678002426638 9,
  0.20119409399743452230062830339459 6,
  0.10114206691871749902707423144739 2,
  0.00000000000000000000000000000000 0
}
```

Definition at line 129 of file gsl_inte_kronrod.h.

45.12.2.8 const double o2scl_inte_gk_coeffs::qk31_wg[8] [static]

Initial value:

```
{
  0.03075324199611726835462839357720 4,
  0.07036604748810812470926741645066 7,
  0.10715922046717193501186954668586 9,
  0.13957067792615431444780479451102 8,
  0.16626920581699393355320086048120 9,
  0.18616100001556221102680056186642 3,
```

```
  0.19843148532711157645611832644383 9,
  0.20257824192556127288062019996751 9
}
```

Definition at line 150 of file gsl_inte_kronrod.h.

45.12.2.9 const double o2scl_inte_gk_coeffs::qk31_wgk[16] [static]

Initial value:

```
{
  0.00537747987292334898779205143012 8,
  0.01500794732931612253837476307580 7,
  0.02546084732671532018687400101965 3,
  0.03534636079137584622203794847836 0,
  0.04458975132476487660822729937328 0,
  0.05348152469092808726534314723943 0,
  0.06200956780067064028513923096080 3,
  0.06985412131872825870952007709914 7,
  0.07684968075772037889443277748265 9,
  0.08308050282313302103828924728610 4,
  0.08856444305621177064727544369377 4,
  0.09312659817082532122548687274734 6,
  0.09664272698362367850517990762758 9,
  0.09917359872179195933239317348460 3,
  0.10076984552387559504494666261757 0,
  0.10133000701479154901737479276749 3
}
```

Definition at line 163 of file gsl_inte_kronrod.h.

45.12.2.10 const double o2scl_inte_gk_coeffs::qk41_xgk[21] [static]

Initial value:

```
{
  0.99885903158827766383831557654586 3,
  0.99312859918509492478612238847132 0,
  0.98150787745025025919334299472021 7,
  0.96397192727791379126766613119727 7,
  0.94082263383175475351998272221244 3,
  0.91223442825132590586775244120329 8,
  0.87827681125228197607744299511307 8,
  0.83911697182221882339452906170152 1,
  0.79504142883755119835063883327278 8,
  0.74633190646015079261430507035564 2,
  0.69323765633475138480549071184593 2,
  0.63605368072651502545283669622628 6,
  0.57514044681971031534294603658642 5,
  0.51086700195082709800436405095525 1,
  0.44359317523872510319999221349264 0,
  0.37370608871541956067254817702492 7,
  0.30162786811491300432055535685859 2,
  0.22778585114164507808049619536857 5,
  0.15260546524092267550522024102267 8,
  0.07652652113349733375464040939883 8,
  0.00000000000000000000000000000000 0
}
```

Definition at line 184 of file gsl_inte_kronrod.h.

45.12.2.11 const double o2scl_inte_gk_coeffs::qk41_wg[11] [static]

Initial value:

```
{
```

```
  0.017614007139152118311861962351853,
  0.040601429800386941331039952274932,
  0.062672048334109063569506535187042,
  0.083276741576704748724758143222046,
  0.101930119817240435036750135480350,
  0.118194531961518417312377377711382,
  0.131688638449176626898494499748163,
  0.142096109318382051329298325067165,
  0.149172986472603746787828737001969,
  0.152753387130725850698084331955098
}
```

Definition at line 210 of file gsl_inte_kronrod.h.

45.12.2.12 const double o2scl_inte_gk_coeffs::qk41_wgk[21] [static]

Initial value:

```
{
  0.003073583718520531501218293246031,
  0.008600269855642942198661787950102,
  0.014626169256971252983787960308868,
  0.020388373461266523598010231432755,
  0.025882133604951158834505067096153,
  0.031287306777032798958543119323801,
  0.036600169758200798030557240707211,
  0.041668873327973686263788305936895,
  0.046434821867497674720231880926108,
  0.050944573923728691932707670050345,
  0.055195105348285994744832372419777,
  0.059111400880639572374967220648594,
  0.062653237554781168025870122174255,
  0.065834597133618422111563556969398,
  0.068648672928521619345623411885368,
  0.071054423553444068305790361723210,
  0.073030690332786667495189417658913,
  0.074582875400499188986581418362488,
  0.075704497684556674659542775376617,
  0.076377867672080736705502835038061,
  0.076600711917999656445049901530102
}
```

Definition at line 225 of file gsl_inte_kronrod.h.

45.12.2.13 const double o2scl_inte_gk_coeffs::qk51_xgk[26] [static]

Initial value:

```
{
  0.999262104992609834193457486540341,
  0.995556969790498097908784946893902,
  0.988035794534077247637331014577406,
  0.976663921459517511498315386479594,
  0.961614986425842512418130033660167,
  0.942974571228974339414011169658471,
  0.920747115281701561746346084546331,
  0.894991997878275368851042006782805,
  0.865847065293275595448996969588340,
  0.833442628760834001421021108693570,
  0.797873797998500059410410904994307,
  0.759259263037357630577282865204361,
  0.717766406813084388186654079773298,
  0.673566368473468364485120633247622,
  0.626810099010317412788122681624518,
  0.577662930241222967723689841612654,
  0.526325284334719182599623778158010,
  0.473002731445714960522182115009192,
```

```
  0.41788538219303774885181439459 4572,
  0.36117230580938783773582173012 7641,
  0.30308953893110783016747890998 0339,
  0.24386688372098843204519036279 7452,
  0.18371893942104889201596988875 9528,
  0.12286469261071039638735981880 8037,
  0.06154448300568507888654639236 6797,
  0.00000000000000000000000000000 0000
}
```

Definition at line 251 of file gsl_inte_kronrod.h.

45.12.2.14 const double o2scl_inte_gk_coeffs::qk51_wg[13] [static]

Initial value:

```
{
  0.01139379850102628794790296411 3235,
  0.02635498661503213726190181529 5299,
  0.04093915670130631265562348771 1646,
  0.05490469597583519192593689154 0473,
  0.06803833381235691720718718565 6708,
  0.08014070033500101801323495966 9111,
  0.09102826198296364981149722070 2892,
  0.10053594906705064420220689039 2686,
  0.10851962447426365311609395705 0117,
  0.11485825914571164833932554586 9556,
  0.11945576353578477222817812651 2901,
  0.12224244299031004168895951894 5852,
  0.12317605372671545120390287307 9050
}
```

Definition at line 282 of file gsl_inte_kronrod.h.

45.12.2.15 const double o2scl_inte_gk_coeffs::qk51_wgk[26] [static]

Initial value:

```
{
  0.00198738389233031592650785188 2843,
  0.00556193213535671375804023690 1066,
  0.00947397338617415160720771052 3655,
  0.01323622919557167481365640584 6976,
  0.01684781770912829823151666753 6336,
  0.02043537114588283545656829223 5939,
  0.02400994560695321622009248916 4881,
  0.02747531758785173780294845551 7811,
  0.03079230016738748889110902021 5229,
  0.03400213027432933783674879522 9551,
  0.03711627148341554356033062536 7620,
  0.04008382550403238207483928446 7076,
  0.04287284502017004947689579243 9495,
  0.04550291304992178890987058475 2660,
  0.04798253713883671390639225575 6915,
  0.05027767908071567196332525943 3440,
  0.05236288580640747586436671213 7873,
  0.05425112988854549014454337045 9876,
  0.05595081122041231730824068638 2747,
  0.05743711636156783285358269393 9506,
  0.05868968002239420796197417585 6788,
  0.05972034032417405997909929193 2562,
  0.06053945537604586294536026751 7565,
  0.06112850971705304830585903041 6293,
  0.06147118987142531666154413196 5264,
  0.06158081806783293507875982424 0066
}
```

Definition at line 300 of file gsl_inte_kronrod.h.

45.12.2.16 const double o2scl_inte_gk_coeffs::qk61_wg[15] [static]

Initial value:

```
{
  0.00796819249616660561546588347467 4,
  0.01846646831109095914230213191204 7,
  0.02878470788332336934971917961129 2,
  0.03879919256962704959680193644634 8,
  0.04840267283059405290293814042280 8,
  0.05749315621761906648172168940205 6,
  0.06597422988218049512812851511596 2,
  0.07375597473770520626824385002219 1,
  0.08075589522942021535469493846053 0,
  0.08689978720108297980238753071512 6,
  0.09212252223778612871763270708761 9,
  0.09636873717464425963946862635181 0,
  0.09959342058679526706278028210356 9,
  0.10176238974840550459642895216855 4,
  0.10285265289355884034128563670541 5
}
```

Definition at line 367 of file gsl_inte_kronrod.h.

45.13 o2scl_inte_qng_coeffs Namespace Reference

A namespace for the quadrature coefficients for non-adaptive integration.

45.13.1 Detailed Description

Documentation from GSL:

Gauss-Kronrod-Patterson quadrature coefficients for use in quadpack routine qng. These coefficients were calculated with 101 decimal digit arithmetic by L. W. Fullerton, Bell Labs, Nov 1981.

Variables

- static const double x1 [5]
- static const double w10 [5]
- static const double x2 [5]
- static const double w21a [5]
- static const double w21b [6]
- static const double x3 [11]
- static const double w43a [10]
- static const double w43b [12]
- static const double x4 [22]
- static const double w87a [21]
- static const double w87b [23]

45.13.2 Variable Documentation

45.13.2.1 const double o2scl_inte_qng_coeffs::x1[5] [static]

Initial value:

```
{
   0.97390652851717172007796401208445 2,
   0.86506336668898451073209668842349 3,
   0.67940956829902440623432736511487 4,
```

```
    0.43339539412924719079926594316578 4,
    0.14887433898163121088482600112972 0
  }
```

Abcissae common to the 10-, 21-, 43- and 87-point rule

Definition at line 44 of file gsl_inte_qng.h.

45.13.2.2 const double o2scl_inte_qng_coeffs::w10[5] [static]

Initial value:

```
{
    0.06667134430868813759356880989333 2,
    0.14945134915058059314577633965769 7,
    0.21908636251598204399553493422816 3,
    0.26926671930999635509122692156946 9,
    0.29552422471475287017389299465133 8
  }
```

Weights of the 10-point formula

Definition at line 53 of file gsl_inte_qng.h.

45.13.2.3 const double o2scl_inte_qng_coeffs::x2[5] [static]

Initial value:

```
{
    0.99565716302580808073552728068900 3,
    0.93015749135570822600120718005950 8,
    0.78081772658641689706371757834504 2,
    0.56275713466860468333900009927269 4,
    0.29439286270146019813112660310386 6
  }
```

Abcissae common to the 21-, 43- and 87-point rule

Definition at line 62 of file gsl_inte_qng.h.

45.13.2.4 const double o2scl_inte_qng_coeffs::w21a[5] [static]

Initial value:

```
{
    0.03255816230796472747881897245939 0,
    0.07503967481091995276704314091619 0,
    0.10938715880229764189921059032580 5,
    0.13470921731147332592805400177170 7,
    0.14773910490133849137484151597206 8
  }
```

Weights of the 21-point formula for abcissae x1

Definition at line 71 of file gsl_inte_qng.h.

45.13.2.5 const double o2scl_inte_qng_coeffs::w21b[6] [static]

Initial value:

```
{
    0.01169463886737187427806439606219 2,
    0.05475589657435199603138130024458 0,
    0.09312545458369760553506546508336 6,
    0.12349197626206585107795810983107 4,
```

```
   0.14277593857706008079709427313871 7,
   0.14944555400291690566493646838982 1
 }
```

Weights of the 21-point formula for abcissae x2

Definition at line 80 of file gsl_inte_qng.h.

45.13.2.6 const double o2scl_inte_qng_coeffs::x3[11] [static]

Initial value:

```
{
   0.99933336090193208139409932391991 1,
   0.98743340290808886979596147838120 9,
   0.95480793481426629925791920029047 3,
   0.90014869574832829362509949406909 2,
   0.82519831498311415084706673258852 0,
   0.73214838898930498261235484875546 1,
   0.62284797053772523864115912034432 3,
   0.49947957407105649995221488549975 5,
   0.36490166134658076804398954850264 4,
   0.22225491977660129649826092806621 2,
   0.07465061746138332204391443579650 6
 }
```

Abscissae common to the 43- and 87-point rule

Definition at line 90 of file gsl_inte_qng.h.

45.13.2.7 const double o2scl_inte_qng_coeffs::w43a[10] [static]

Initial value:

```
{
   0.01629673428966656492428197461766 3,
   0.03752287612086950146161379589811 5,
   0.05469490205825544214721268546500 5,
   0.06735541460947808607555316630217 4,
   0.07387019963239395343214069525136 7,
   0.00576855605976979618418432790865 5,
   0.02737189059324884208127606928915 1,
   0.04656082691042883074333915443382 4,
   0.06174499520144256449624033603088 3,
   0.07138726726869339776855911442551 6
 }
```

Weights of the 43-point formula for abscissae x1, x3

Definition at line 105 of file gsl_inte_qng.h.

45.13.2.8 const double o2scl_inte_qng_coeffs::w43b[12] [static]

Initial value:

```
{
   0.00184447764021241410038910655296 5,
   0.01079868958589165174046540674129 3,
   0.02189536386779542810252312307514 9,
   0.03259746397534568944388222252613 7,
   0.04216313793519181184762792432795 5,
   0.05074193960018457778018902009208 4,
   0.05837939554261924837547536933020 6,
   0.06474640495144588554468925951751 1,
   0.06956619791235648452863331503840 5,
   0.07282444147183320815093953519284 2,
```

```
    0.074507751014175118273571813842889,
    0.074722147517403005594425168280423
  }
```

Weights of the 43-point formula for abscissae x3

Definition at line 119 of file gsl_inte_qng.h.

45.13.2.9 const double o2scl_inte_qng_coeffs::x4[22] [static]

Initial value:

```
{
    0.999902977262729234490529830591582,
    0.997989895986678745427496322365960,
    0.992175497860687222808523352251425,
    0.981358163572712773571916941623894,
    0.965057623858384619128284110607926,
    0.943167613133670596816416634507426,
    0.915806414685507209591826430720050,
    0.883221657771316501372117548744163,
    0.845710748462415666605902011504855,
    0.803557658035230982788739474980964,
    0.757005730685495558328942793432020,
    0.706273209787321819824094274740840,
    0.651589466501177922534422205016736,
    0.593223374057961088875273770349144,
    0.531493605970831932285268948562671,
    0.466763623042022844871966781659270,
    0.399424847859218804732101665817923,
    0.329874877106188288265053371824597,
    0.258503559202161551802280975429025,
    0.185695396568346652015917141167606,
    0.111842213179907468172398359241362,
    0.037352123394619870814998165437704
  }
```

Abscissae of the 87-point rule

Definition at line 135 of file gsl_inte_qng.h.

45.13.2.10 const double o2scl_inte_qng_coeffs::w87a[21] [static]

Initial value:

```
{
    0.008148377384149172900002878448190,
    0.018761438201562822243935059003794,
    0.027347451050052286161582829741283,
    0.033677707311637930046581056957588,
    0.036935099820427907614589586742499,
    0.002884872430211530501334156248695,
    0.013685946022712701888950035273128,
    0.023280413502888311123409291030404,
    0.030872497611713358675466394126442,
    0.035693633639418770719351355457044,
    0.000915283345202241360843392549948,
    0.005399280219300471367738743391053,
    0.010947679601118931134327826856808,
    0.016298731696787335262665703223280,
    0.021081568889203835112433060188190,
    0.025370969769253827243467999831710,
    0.029189697756475752501446154084920,
    0.032373202467202789685788194889595,
    0.034783098950365142750781997949596,
    0.036412220731351787562801163687577,
    0.037253875503047708539592001191226
  }
```

Weights of the 87-point formula for abscissae x1, x2, x3

Definition at line 161 of file gsl_inte_qng.h.

45.13.2.11 const double o2scl_inte_qng_coeffs::w87b[23] [static]

Initial value:

```
{
   0.00027414556376207235001652709288 1,
   0.00180712415505794294834131175325 4,
   0.00409686928275916486445807068348 0,
   0.00675829005184737869981657789742 4,
   0.00954995767220164653605358132537 7,
   0.01232944765224485369462663996378 0,
   0.01501044734638895237669728604194 3,
   0.01754896798624319109966535292590 0,
   0.01993803778644088820227819273071 4,
   0.02219493596101228679633210295949 9,
   0.02433914712600080547036064704145 4,
   0.02637450541483920724150378655261 5,
   0.02828691078877120065996800298796 0,
   0.03005258112809269532252111034734 1,
   0.03164675137143992940458605107888 3,
   0.03305041341997850329078594486268 9,
   0.03425509970422606178708282104682 1,
   0.03526241266015668103378271799842 8,
   0.03607698962288870118550031800389 5,
   0.03669860449845609449801804744109 4,
   0.03712054926983257611411995841359 9,
   0.03733422875193504032123544909469 8,
   0.03736107376267902341032124176659 9
 }
```

Weights of the 87-point formula for abscissae x4

Definition at line 186 of file gsl_inte_qng.h.

45.14 o2scl_linalg Namespace Reference

The namespace for linear algebra classes and functions.

45.14.1 Detailed Description

This namespace documentation is in the file src/base/lib_settings.h

Idea for Future Move this documentation to the linalg directory.

Data Structures

- class linear_solver
  A generic solver for the linear system $Ax = b$ [abstract base].
- class linear_solver_lu
  Generic linear solver using LU decomposition.
- class linear_solver_qr
  Generic linear solver using QR decomposition.
- class linear_solver_hh
  Generic Householder linear solver.
- class gsl_solver_LU
  GSL solver by LU decomposition.

- class gsl_solver_QR
  GSL solver by QR decomposition.
- class gsl_solver_HH
  GSL Householder solver.
- class pointer_2_mem
  Allocation object for 2 C-style arrays of equal size.
- class pointer_4_mem
  Allocation object for 4 C-style arrays of equal size.
- class pointer_5_mem
  Allocation object for 5 C-style arrays of equal size.
- class uvector_2_mem
  Allocation object for 2 arrays of equal size.
- class uvector_4_mem
  Allocation object for 4 arrays of equal size.
- class uvector_5_mem
  Allocation object for 5 arrays of equal size.

Functions

- void create_givens (const double a, const double b, double &c, double &s)
  Desc.
- template<class mat1_t , class mat2_t >
  void apply_givens_qr (size_t M, size_t N, mat1_t &Q, mat2_t &R, size_t i, size_t j, double c, double s)
  Apply a rotation to matrices from the QR decomposition.
- template<class mat1_t , class mat2_t >
  void apply_givens_lq (size_t M, size_t N, mat1_t &Q, mat2_t &L, size_t i, size_t j, double c, double s)
  Apply a rotation to matrices from the LQ decomposition.
- template<class vec_t >
  void apply_givens_vec (vec_t &v, size_t i, size_t j, double c, double s)
  Apply a rotation to a vector, $v \to G^T v$.
- template<class mat_t , class vec_t , class vec2_t >
  int HH_solve (size_t n, mat_t &A, const vec_t &b, vec2_t &x)
  Solve linear system after Householder decomposition.
- template<class mat_t , class vec_t >
  int HH_svx (size_t N, size_t M, mat_t &A, vec_t &x)
  Solve a linear system after Householder decomposition in place.
- template<class vec_t >
  double householder_transform (const size_t n, vec_t &v)
  Replace the vector `v` with a Householder vector and a coefficient tau that annihilates `v[1]` through `v[n-1]` (inclusive)
- template<class mat_t >
  double householder_transform_subcol (mat_t &A, const size_t ir, const size_t ic, const size_t M)
  Compute the Householder transform of a vector formed with `n` rows of a column of a matrix.
- template<class mat_t >
  double householder_transform_subrow (mat_t &A, const size_t ir, const size_t ic, const size_t N)
  Compute the Householder transform of a vector formed with the last `n` columns of a row of a matrix.
- template<class vec_t , class mat_t >
  void householder_hm (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
  Apply a Householder transformation (v, τ) to matrix A of size M by N.
- template<class mat_t >
  void householder_hm_sub (mat_t &M, const size_t ir, const size_t ic, const size_t nr, const size_t nc, const mat_t &M2, const size_t ir2, const size_t ic2, double tau)
  Apply a Householder transformation to submatrix of a larger matrix.
- template<class mat1_t , class mat2_t >
  void householder_hm_sub2 (const size_t M, const size_t ic, double tau, const mat1_t &mv, mat2_t &A)
  Special version of Householder transformation for QR_unpack()
- template<class vec_t , class vec2_t >
  void householder_hv (const size_t N, double tau, const vec_t &v, vec2_t &w)
  Apply a Householder transformation `v` to vector `w`.

- template<class mat_t , class vec_t >
  void householder_hv_subcol (const mat_t &A, vec_t &w, double tau, const size_t ie, const size_t N)
  Apply a Householder transformation `v` to vector `w` where `v` is stored as a column in a matrix `A`.
- template<class mat_t >
  void householder_hm1 (const size_t M, const size_t N, double tau, mat_t &A)
  Apply a Householder transformation (v,τ) to a matrix being build up from the identity matrix, using the first column of A as a - Householder vector.
- template<class vec_t , class mat_t >
  void householder_mh (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
  Apply the Householder transformation `(v,tau)` to the right-hand side of the matrix `A`.
- template<class mat_t , class mat2_t >
  void householder_mh_sub (mat_t &M, const size_t ir, const size_t ic, const size_t nr, const size_t nc, const mat2_t &M2, const size_t ir2, const size_t ic2, double tau)
  Apply the Householder transformation `(v,tau)` to the right-hand side of the matrix `A`.
- template<size_t N>
  int LU_decomp_array_2d (const size_t n, double A[ ][N], o2scl::permutation &p, int &signum)
  Specialized version of LU_decomp for C-style 2D arrays.
- template<class mat_t >
  int diagonal_has_zero (const size_t N, mat_t &A)
  Return 1 if at least one of the elements in the diagonal is zero.
- template<class mat_t >
  int LU_decomp (const size_t N, mat_t &A, o2scl::permutation &p, int &signum)
  Compute the LU decomposition of the matrix `A`.
- template<class mat_t , class mat_row_t >
  int **LU_decomp_alt** (const size_t N, mat_t &A, o2scl::permutation &p, int &signum)
- template<class mat_t , class vec_t , class vec2_t >
  int LU_solve (const size_t N, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec2_t &x)
  Solve a linear system after LU decomposition.
- template<class mat_t , class vec_t >
  int LU_svx (const size_t N, const mat_t &LU, const o2scl::permutation &p, vec_t &x)
  Solve a linear system after LU decomposition in place.
- template<class mat_t , class mat2_t , class vec_t , class vec2_t , class vec3_t >
  int LU_refine (const size_t N, const mat_t &A, const mat2_t &LU, const o2scl::permutation &p, const vec_t &b, vec2_t &x, vec3_t &residual)
  Refine the solution of a linear system.
- template<class mat_t , class mat2_t , class mat_col_t >
  int LU_invert (const size_t N, const mat_t &LU, const o2scl::permutation &p, mat2_t &inverse)
  Compute the inverse of a matrix from its LU decomposition.
- template<class mat_t >
  double LU_det (const size_t N, const mat_t &LU, int signum)
  Compute the determinant of a matrix from its LU decomposition.
- template<class mat_t >
  double LU_lndet (const size_t N, const mat_t &LU)
  Compute the logarithm of the absolute value of the determinant of a matrix from its LU decomposition.
- template<class mat_t >
  int LU_sgndet (const size_t N, const mat_t &LU, int signum)
  Compute the sign of the determinant of a matrix from its LU decomposition.
- template<class mat_t , class vec_t >
  void QR_decomp (size_t M, size_t N, mat_t &A, vec_t &tau)
  Compute the QR decomposition of matrix `A`.
- template<class mat_t , class vec_t , class vec2_t , class vec3_t >
  void QR_solve (size_t N, const mat_t &QR, const vec_t &tau, const vec2_t &b, vec3_t &x)
  Solve the system A x = b using the QR factorization.
- template<class mat_t , class vec_t , class vec2_t >
  void QR_svx (size_t M, size_t N, const mat_t &QR, const vec_t &tau, vec2_t &x)
  Solve the system A x = b in place using the QR factorization.
- template<class mat_t , class vec_t , class vec2_t >
  void QR_QTvec (const size_t M, const size_t N, const mat_t &QR, const vec_t &tau, vec2_t &v)
  Form the product Q^T v from a QR factorized matrix.

- template<class mat1_t , class mat2_t , class mat3_t , class vec_t >
  void QR_unpack (const size_t M, const size_t N, const mat1_t &QR, const vec_t &tau, mat2_t &Q, mat3_t &R)
  Unpack the QR matrix to the individual Q and R components.
- template<class mat1_t , class mat2_t , class vec1_t , class vec2_t >
  void QR_update (size_t M, size_t N, mat1_t &Q, mat2_t &R, vec1_t &w, vec2_t &v)
  Update a QR factorisation for A= Q R, A' = A + u v^T,.
- template<class vec_t , class vec2_t >
  int chop_small_elements (size_t N, vec_t &d, vec2_t &f)
  Desc.
- template<class vec_t , class vec2_t >
  double trailing_eigenvalue (size_t n, const vec_t &d, const vec_t &f)
  Desc.
- int create_schur (double d0, double f0, double d1, double &c, double &s)
  Desc.
- template<class vec_t , class vec2_t , class mat_t , class mat2_t >
  int svd2 (size_t M, size_t N, vec_t &d, vec2_t &f, mat_t &U, mat2_t &V)
  Desc.
- template<class vec_t , class vec2_t , class mat_t >
  int chase_out_intermediate_zero (size_t M, size_t n, vec_t &d, vec2_t &f, mat_t &U, size_t k0)
  Desc.
- template<class vec_t , class vec2_t , class mat_t >
  int chase_out_trailing_zero (size_t N, size_t n, vec_t &d, vec2_t &f, mat_t &V)
  Desc.
- template<class vec_t , class vec2_t , class mat_t , class mat2_t >
  int qrstep (size_t M, size_t N, size_t n, vec_t &d, vec2_t &f, mat_t &U, mat2_t &V)
  Desc.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class mem_t , class mem_vec_t >
  void solve_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N, mem_t &m)
  Solve a symmetric tridiagonal linear system with user-specified memory.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t , class mem_t , class mem_vec_t >
  void solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N, mem_t &m)
  Solve an asymmetric tridiagonal linear system with user-specified memory.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class mem_t , class mem_vec_t >
  void solve_cyc_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N, mem_t &m)
  Solve a symmetric cyclic tridiagonal linear system with user specified memory.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t , class mem_t , class mem_vec_t >
  void solve_cyc_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N, mem_t &m)
  Solve an asymmetric cyclic tridiagonal linear system with user-specified memory.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t >
  void solve_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N)
  Solve a symmetric tridiagonal linear system.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t >
  void solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N)
  Solve an asymmetric tridiagonal linear system.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t >
  void solve_cyc_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N)
  Solve a symmetric cyclic tridiagonal linear system.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t >
  void solve_cyc_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N)
  Solve an asymmetric cyclic tridiagonal linear system.

45.14.2 Function Documentation

45.14.2.1 template<class mat1_t , class mat2_t > void o2scl_linalg::apply_givens_qr (size_t *M*, size_t *N*, mat1_t & *Q*, mat2_t & *R*, size_t *i*, size_t *j*, double *c*, double *s*)

This performs $Q \to QG$ and $R \to G^T R$.

Definition at line 58 of file givens_base.h.

45.14.2.2 template<class mat1_t , class mat2_t > void o2scl_linalg::apply_givens_lq (size_t *M*, size_t *N*, mat1_t & *Q*, mat2_t & *L*, size_t *i*, size_t *j*, double *c*, double *s*)

This performs $Q \to QG$ and $L \to LG^T$.

Definition at line 83 of file givens_base.h.

45.14.2.3 template<class mat_t , class vec_t > int o2scl_linalg::HH_svx (size_t *N*, size_t *M*, mat_t & *A*, vec_t & *x*)

Idea for Future Handle memory allocation like the tridiagonal functions.

Definition at line 68 of file hh_base.h.

45.14.2.4 template<class vec_t > double o2scl_linalg::householder_transform (const size_t *n*, vec_t & *v*)

On exit, this function returns the value of $\tau = 2/(v^T v)$. If `n` is less than or equal to 1 then this function returns zero without calling the error handler.

Definition at line 69 of file householder_base.h.

45.14.2.5 template<class mat_t > double o2scl_linalg::householder_transform_subcol (mat_t & *A*, const size_t *ir*, const size_t *ic*, const size_t *M*)

This performs a Householder transform of a vector defined by a column of a matrix `A` which starts at element `A[ir][ic]` and ends at element `A[N-1][ic]`.

Used in QR_decomp().

Definition at line 114 of file householder_base.h.

45.14.2.6 template<class mat_t > double o2scl_linalg::householder_transform_subrow (mat_t & *A*, const size_t *ir*, const size_t *ic*, const size_t *N*)

This performs a Householder transform of a vector defined by a row of a matrix `A` which starts at element `A[ir][ic]` and ends at element `A[ir][N-1]`

Definition at line 156 of file householder_base.h.

45.14.2.7 template<class mat_t > void o2scl_linalg::householder_hm_sub (mat_t & *M*, const size_t *ir*, const size_t *ic*, const size_t *nr*, const size_t *nc*, const mat_t & *M2*, const size_t *ir2*, const size_t *ic2*, double *tau*)

This applies a householder transformation `(v,tau)` to submatrix of `M`. The submatrix has `nr` rows and `nc` columns and starts at row `ir` of column `ic` of the original matrix `M`. The vector `v` is taken from a column of `M2` starting at row `ir2` and column `ic2`.

This function is used in QR_decomp().

Definition at line 237 of file householder_base.h.

45.14.2.8 template<class mat_t , class vec_t > void o2scl_linalg::householder_hv_subcol (const mat_t & *A*, vec_t & *w*, double *tau*, const size_t *ie*, const size_t *N*)

Used in QR_QTvec().

Definition at line 337 of file householder_base.h.

45.14.2.9 template<class mat_t , class mat2_t > void o2scl_linalg::householder_mh_sub (mat_t & *M,* const size_t *ir,* const size_t *ic,* const size_t *nr,* const size_t *nc,* const mat2_t & *M2,* const size_t *ir2,* const size_t *ic2,* double *tau*)

This is unfinished but needed for the bidiagonalization functions.

Definition at line 446 of file householder_base.h.

45.14.2.10 template<size_t N> int o2scl_linalg::LU_decomp_array_2d (const size_t *n,* double *A[][N],* o2scl::permutation & *p,* int & *signum*)

Note

Note that N and n must be equal, by no checking is done to ensure that this is the case

Definition at line 42 of file lu.h.

45.14.2.11 template<class mat_t > int o2scl_linalg::LU_decomp (const size_t *N,* mat_t & *A,* o2scl::permutation & *p,* int & *signum*)

On output the diagonal and upper triangular part of the input matrix A contain the matrix U. The lower triangular part of the input matrix (excluding the diagonal) contains L. The diagonal elements of L are unity, and are not stored.

The permutation matrix P is encoded in the permutation p. The j-th column of the matrix P is given by the k-th column of the identity matrix, where k = p_j the j-th element of the permutation vector. The sign of the permutation is given by signum. It has the value (-1)^n, where n is the number of interchanges in the permutation.

The algorithm used in the decomposition is Gaussian Elimination with partial pivoting (Golub & Van Loan, Matrix Computations, Algorithm 3.4.1).

Idea for Future The "swap rows j and i_pivot" section could probably be made more efficient using a "matrix_row"-like object as done in GSL. (7/16/09 - I've tried this, and it doesn't seem to improve the speed significantly.)

Definition at line 86 of file lu_base.h.

45.14.2.12 template<class mat_t , class vec_t , class vec2_t > int o2scl_linalg::LU_solve (const size_t *N,* const mat_t & *LU,* const o2scl::permutation & *p,* const vec_t & *b,* vec2_t & *x*)

This function solve the square system A x = b using the LU decomposition of A into (LU, p) given by gsl_linalg_LU_decomp or gsl_linalg_complex_LU_decomp.

Definition at line 207 of file lu_base.h.

45.14.2.13 template<class mat_t , class vec_t > int o2scl_linalg::LU_svx (const size_t *N,* const mat_t & *LU,* const o2scl::permutation & *p,* vec_t & *x*)

These functions solve the square system A x = b in-place using the LU decomposition of A into (LU,p). On input x should contain the right-hand side b, which is replaced by the solution on output.

Definition at line 230 of file lu_base.h.

45.14.2.14 template<class mat_t , class mat2_t , class vec_t , class vec2_t , class vec3_t > int o2scl_linalg::LU_refine (const size_t *N,* const mat_t & *A,* const mat2_t & *LU,* const o2scl::permutation & *p,* const vec_t & *b,* vec2_t & *x,* vec3_t & *residual*)

These functions apply an iterative improvement to x, the solution of A x = b, using the LU decomposition of A into (LU,p). The initial residual r = A x - b is also computed and stored in residual.

Definition at line 262 of file lu_base.h.

45.14.2.15 template<class mat_t , class mat2_t , class mat_col_t > int o2scl_linalg::LU_invert (const size_t *N,* const mat_t & *LU,* const o2scl::permutation & *p,* mat2_t & *inverse*)

These functions compute the inverse of a matrix A from its LU decomposition (LU,p), storing the result in the matrix inverse. The inverse is computed by solving the system A x = b for each column of the identity matrix. It is preferable to avoid direct use of the inverse whenever possible, as the linear solver functions can obtain the same result more efficiently and reliably.

Idea for Future Could rewrite to avoid mat_col_t, (9/16/09 - However, the function may be faster if mat_col_t is left in, so it's unclear what's best.)

Definition at line 300 of file lu_base.h.

45.14.2.16 template<class mat_t> double o2scl_linalg::LU_det (const size_t *N*, const mat_t & *LU*, int *signum*)

Compute the determinant of a matrix A from its LU decomposition, LU. The determinant is computed as the product of the diagonal elements of U and the sign of the row permutation signum.

Definition at line 339 of file lu_base.h.

45.14.2.17 template<class mat_t> double o2scl_linalg::LU_lndet (const size_t *N*, const mat_t & *LU*)

Compute the logarithm of the absolute value of the determinant of a matrix A, $\ln|\det(A)|$, from its LU decomposition, LU. This function may be useful if the direct computation of the determinant would overflow or underflow.

Definition at line 361 of file lu_base.h.

45.14.2.18 template<class mat_t> int o2scl_linalg::LU_sgndet (const size_t *N*, const mat_t & *LU*, int *signum*)

Compute the sign or phase factor of the determinant of a matrix A, $\det(A)/|\det(A)|$, from its LU decomposition, LU.

Definition at line 380 of file lu_base.h.

45.14.2.19 template<class mat1_t, class mat2_t, class vec1_t, class vec2_t> void o2scl_linalg::QR_update (size_t *M*, size_t *N*, mat1_t & *Q*, mat2_t & *R*, vec1_t & *w*, vec2_t & *v*)

The parameters `M` and `N` are the number of rows and columns of the matrix `R`.

```
* Q' R' = QR + u v^T
*       = Q (R + Q^T u v^T)
*       = Q (R + w v^T)
*
* where w = Q^T u.
*
* Algorithm from Golub and Van Loan, "Matrix Computations", Section
* 12.5 (Updating Matrix Factorizations, Rank-One Changes)
```

Definition at line 161 of file qr_base.h.

45.14.2.20 template<class vec_t, class vec2_t> int o2scl_linalg::chop_small_elements (size_t *N*, vec_t & *d*, vec2_t & *f*)

The parameter `N` is the size of `d`.

Definition at line 55 of file svdstep_base.h.

45.14.2.21 template<class vec_t, class vec2_t> double o2scl_linalg::trailing_eigenvalue (size_t *n*, const vec_t & *d*, const vec_t & *f*)

The parameter `n` is the size of the vector `d`.

Should be finished.

Definition at line 82 of file svdstep_base.h.

45.14.2.22 int o2scl_linalg::create_schur (double *d0*, double *f0*, double *d1*, double & *c*, double & *s*)

Should be finished.

Definition at line 117 of file svdstep_base.h.

45.14.2.23 template<class vec_t , class vec2_t , class mat_t , class mat2_t > int o2scl_linalg::svd2 (size_t *M,* size_t *N,* vec_t & *d,* vec2_t & *f,* mat_t & *U,* mat2_t & *V*)

The parameter M is the number of rows in U and N is the number of rows in V.

Definition at line 169 of file svdstep_base.h.

45.14.2.24 template<class vec_t , class vec2_t , class mat_t > int o2scl_linalg::chase_out_intermediate_zero (size_t *M,* size_t *n,* vec_t & *d,* vec2_t & *f,* mat_t & *U,* size_t *k0*)

Should be finished.

Definition at line 305 of file svdstep_base.h.

45.14.2.25 template<class vec_t , class vec2_t , class mat_t > int o2scl_linalg::chase_out_trailing_zero (size_t *N,* size_t *n,* vec_t & *d,* vec2_t & *f,* mat_t & *V*)

Should be finished.

Definition at line 352 of file svdstep_base.h.

45.14.2.26 template<class vec_t , class vec2_t , class mat_t , class mat2_t > int o2scl_linalg::qrstep (size_t *M,* size_t *N,* size_t *n,* vec_t & *d,* vec2_t & *f,* mat_t & *U,* mat2_t & *V*)

Should be finished.

The parameter M is the number of rows in U, N is the number of rows in V, and n is the length of the vector d.

Definition at line 402 of file svdstep_base.h.

45.14.2.27 template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class mem_t , class mem_vec_t > void o2scl_linalg::solve_tridiag_sym (const vec_t & *diag,* const vec2_t & *offdiag,* const vec3_t & *b,* vec4_t & *x,* size_t *N,* mem_t & *m*)

This function solves the system $Ax = b$ where A is a matrix of the form

```
*
*     diag[0]  offdiag[0]             0   .....
* offdiag[0]     diag[1]    offdiag[1]   .....
*           0  offdiag[1]       diag[2]
*           0           0    offdiag[2]   .....
```

given the N diagonal elements in diag, N-1 diagonal elements in offdiag, and the N elements b from the RHS.

See EngelnMullges96 .

Definition at line 208 of file tridiag_base.h.

45.14.2.28 template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t , class mem_t , class mem_vec_t > void o2scl_linalg::solve_tridiag_nonsym (const vec_t & *diag,* const vec2_t & *abovediag,* const vec3_t & *belowdiag,* const vec4_t & *rhs,* vec5_t & *x,* size_t *N,* mem_t & *m*)

This function solves the system $Ax = b$ where A is a matrix of the form

```
*
*       diag[0]  abovediag[0]             0   .....
* belowdiag[0]       diag[1]  abovediag[1]   .....
*             0  belowdiag[1]       diag[2]
*             0             0  belowdiag[2]   .....
```

This function uses plain Gauss elimination, not bothering with the zeroes.

Idea for Future Offer an option to avoid throwing on divide by zero?

Definition at line 273 of file tridiag_base.h.

45.14.2.29 template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class mem_t , class mem_vec_t > void o2scl_linalg::solve_cyc_tridiag_sym (const vec_t & *diag*, const vec2_t & *offdiag*, const vec3_t & *b*, vec4_t & *x*, size_t *N*, mem_t & *m*)

This function solves the system $Ax = b$ where A is a matrix of the form

```
*
*      diag[0]  offdiag[0]            0   .....  offdiag[N-1]
*   offdiag[0]     diag[1]    offdiag[1]   .....
*            0  offdiag[1]       diag[2]
*            0           0    offdiag[2]   .....
*          ...         ...
* offdiag[N-1]         ...
```

See EngelnMullges96 .

Definition at line 331 of file tridiag_base.h.

45.14.2.30 template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t , class mem_t , class mem_vec_t > void o2scl_linalg::solve_cyc_tridiag_nonsym (const vec_t & *diag*, const vec2_t & *abovediag*, const vec3_t & *belowdiag*, const vec4_t & *rhs*, vec5_t & *x*, size_t *N*, mem_t & *m*)

This function solves the system $Ax = b$ where A is a matrix of the form

```
*
*        diag[0]  abovediag[0]             0   .....  belowdiag[N-1]
*   belowdiag[0]       diag[1]  abovediag[1]   .....
*              0  belowdiag[1]       diag[2]
*              0             0  belowdiag[2]   .....
*            ...           ...
* abovediag[N-1]           ...
```

This function solves the following system without the corner elements and then use Sherman-Morrison formula to compensate for them.

Idea for Future Offer an option to avoid throwing on divide by zero?

Definition at line 427 of file tridiag_base.h.

45.15 o2scl_linalg_paren Namespace Reference

The namespace for linear algebra classes and functions with operator()

45.15.1 Detailed Description

This namespace contains an identical copy of all the functions given in the o2scl_cblas namespace, but perform array indexing with `operator()` rather than `operator[]`. See o2scl_linalg for the function listing and documentation.

This namespace documentation is in the file `src/base/lib_settings.h`

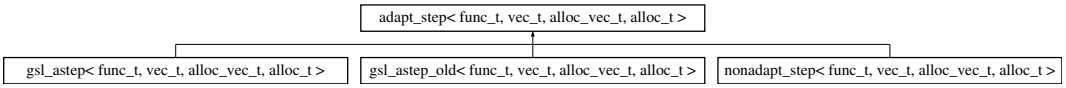
46 Data Structure Documentation

46.1 adapt_step< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Adaptive stepper [abstract base].

`#include <adapt_step.h>`

Inheritance diagram for adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >:



46.1.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >

The adaptive stepper routines are based on one or many applications of ordinary ODE steppers (implemented in ode_step). Each adaptive stepper (gsl_astep or nonadapt_step) can be used with any of the ODE stepper classes (e.g. gsl_rkck). By default, gsl_rkck is used. To modify the ODE stepper which is used, use the member function set_step() documented below.

Note

If you use gsl_rkck_fast or gsl_rk8pd_fast, you'll need to make sure that the argument `n` to astep() or astep_derivs() below matches the template size parameter given in the ODE stepper.

Definition at line 51 of file adapt_step.h.

Public Member Functions

- virtual int astep (double &x, double xlimit, double &h, size_t n, vec_t &y, vec_t &dydx_out, vec_t &yerr, func_t &derivs)=0
  Make an adaptive integration step of the system `derivs`.
- virtual int astep_derivs (double &x, double xlimit, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yerr, func_t &derivs)=0
  Make an adaptive integration step of the system `derivs` with derivatives.
- virtual int astep_full (double x, double xlimit, double &x_out, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)=0
  Make an adaptive integration step of the system `derivs` with derivatives.
- int set_step (ode_step< func_t, vec_t > &step)
  Set stepper.

Data Fields

- int verbose
  Set output level.
- gsl_rkck< func_t, vec_t, alloc_vec_t, alloc_t > def_step
  The default stepper.

Protected Attributes

- ode_step< func_t, vec_t > * stepp
  Pointer to the stepper being used.

46.1.2 Member Function Documentation

46.1.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::astep (double & *x*, double *xlimit*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx_out*, vec_t & *yerr*, func_t & *derivs*) [pure virtual]

This attempts to take a step of size `h` from the point `x` of an `n-dimensional` system `derivs` starting with `y`. On exit, `x` and `y` contain the new values at the end of the step, `h` contains the size of the step, `dydx_out` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

Implemented in gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >, gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >, and nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

46.1.2.2 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::astep_derivs (double & *x*, double *xlimit*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yerr*, func_t & *derivs*) [pure virtual]

This attempts to take a step of size `h` from the point `x` of an `n-dimensional` system `derivs` starting with `y` and given the initial derivatives `dydx`. On exit, `x`, `y` and `dydx` contain the new values at the end of the step, `h` contains the size of the step, `dydx` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

Implemented in gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >, gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >, and nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

46.1.2.3 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::astep_full (double *x*, double *xlimit*, double & *x_out*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [pure virtual]

This function performs an adaptive integration step with the `n-dimensional` system `derivs` and parameter `pa`. It Begins at `x` with initial stepsize `h`, ensuring that the step goes no farther than `xlimit`. At the end of the step, the size of the step taken is `h` and the new value of `x` is in `x_out`. Initially, the function values and derivatives should be specified in `y` and `dydx`. The function values, derivatives, and the error at the end of the step are given in `yout`, `yerr`, and `dydx_out`. Unlike in `ode_step` objects, the objects `y`, `yout`, `dydx`, and `dydx_out` must all be distinct.

Implemented in gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >, gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >, and nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

46.1.2.4 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::set_step (ode_step< func_t, vec_t > & *step*) [inline]

This sets the stepper for use in the adaptive step routine. If no stepper is specified, then the default (def_step of type gsl_rkck) is used.

Definition at line 120 of file adapt_step.h.

The documentation for this class was generated from the following file:

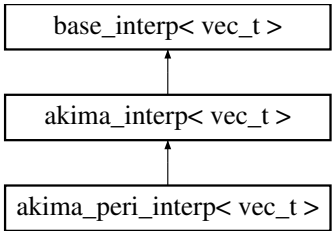
- adapt_step.h

46.2 akima_interp< vec_t > Class Template Reference

Akima spline interpolation (GSL)

`#include <interp.h>`

Inheritance diagram for akima_interp< vec_t >:



46.2.1 Detailed Description

template<class vec_t>class akima_interp< vec_t >

This class uses natural boundary conditions, where the second derivative vanishes at each end point. Extrapolation effectively assumes that the second derivative is linear outside of the endpoints. Use akima_peri_interp for perodic boundary conditions.

Definition at line 765 of file interp.h.

Public Member Functions

- akima_interp ()
  Create a base interpolation object with or without periodic boundary conditions.
- virtual int allocate (size_t size)
  Allocate memory, assuming x and y have size `size`.
- virtual int init (const vec_t &xa, const vec_t &ya, size_t size)
  Initialize interpolation routine.
- virtual int free ()
  Free allocated memory.
- virtual int interp (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &y)
  Give the value of the function $y(x = x_0)$.
- virtual int deriv (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &dydx)
  Give the value of the derivative $y'(x = x_0)$.
- virtual int deriv2 (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &d2ydx2)
  Give the value of the second derivative $y''(x = x_0)$.
- virtual int integ (const vec_t &x_array, const vec_t &y_array, size_t size, double aa, double bb, double &result)
  Give the value of the integral $\int_a^b y(x)\,dx$.
- virtual const char * type ()
  Return the type, `"akima_interp"`.

Protected Member Functions

- void akima_calc (const vec_t &x_array, size_t size, double m[ ])
  For initializing the interpolation.

Protected Attributes

Storage for Akima spline interpolation

- double * **b**
- double * **c**
- double * **d**
- double * **um**

Private Member Functions

- **akima_interp** (const akima_interp< vec_t > &)
- akima_interp< vec_t > & **operator=** (const akima_interp< vec_t > &)

46.2.2 Member Function Documentation

46.2.2.1 template<class vec_t> virtual int akima_interp< vec_t >::init (const vec_t & *xa*, const vec_t & *ya*, size_t *size*) [inline, virtual]

Non-periodic boundary conditions

Reimplemented from base_interp< vec_t >.

Reimplemented in akima_peri_interp< vec_t >.

Definition at line 862 of file interp.h.

The documentation for this class was generated from the following file:

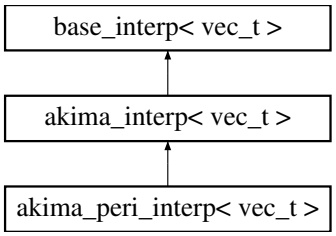
- interp.h

46.3 akima_peri_interp< vec_t > Class Template Reference

Akima spline interpolation with periodic boundary conditions (GSL)

`#include <interp.h>`

Inheritance diagram for akima_peri_interp< vec_t >:



46.3.1 Detailed Description

template<class vec_t>class akima_peri_interp< vec_t >

Definition at line 1025 of file interp.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "akima_peri_interp".
- virtual int init (const vec_t &xa, const vec_t &ya, size_t size)
  Initialize interpolation routine.

Private Member Functions

- **akima_peri_interp** (const akima_peri_interp< vec_t > &)
- akima_peri_interp< vec_t > & **operator=** (const akima_peri_interp< vec_t > &)

46.3.2 Member Function Documentation

46.3.2.1 template<class vec_t> virtual int akima_peri_interp< vec_t >::init (const vec_t & *xa*, const vec_t & *ya*, size_t *size*) [inline, virtual]

Periodic boundary conditions

Reimplemented from akima_interp< vec_t >.

Definition at line 1039 of file interp.h.

The documentation for this class was generated from the following file:

- interp.h

46.4 anneal_mt< param_t, param_vec_t, func_t, vec_t, alloc_vec_t, alloc_t, rng_t > Class Template Reference

Multidimensional minimization by simulated annealing (Boost multi-threaded version)

`#include <anneal_mt.h>`

46.4.1 Detailed Description

template<class param_t, class param_vec_t = std::vector<param_t>, class func_t = multi_funct<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class rng_t = gsl_rnga>class anneal_mt< param_t, param_vec_t, func_t, vec_t, alloc_vec_t, alloc_t, rng_t >

This header-only class additionally requires the Boost libraries. It performs simulated annealing using an arbitrary number of processors using `boost::thread`, which is closely related to the standard Unix pthread library. It works very similarly to gsl_anneal, it performs ntrial evaluations over each processor, then applying the metropolis algorithm to the results from all of the processors at the end.

Because `np` function calls are happening simultaneously, where `np` is the number of processors, `np` copies of the function parameters of type `param_t` must also be specified. The user-defined function to minimize must also be thread-safe, allowing multiple threads to call the function at once (albeit given different parameters). The default type for these `np` copies of the parameters of type `param_t` is `std::vector<param_t>`.

This works particularly well for functions which are not trivial to evaluate, i.e. functions where the execution time is more longer than the bookkeeping that anneal_mt performs between trials. For functions which satisfy this requirement, this algorithm scales nearly linearly with the number of processors.

Verbose I/O for this class happens only outside the theads unless the user places I/O in the streams in the function that is specified.

Idea for Future There may be a good way to remove the function indirection here to make this class a bit faster.

Definition at line 75 of file anneal_mt.h.

Public Member Functions

- virtual const char * type ()
  Return string denoting type ("anneal_mt")
- int set_verbose_stream (std::ostream &out, std::istream &in)
  Set streams for verbose I/O.
- virtual int print_iter (size_t nv, vec_t &xx, double y, int iter, double tptr, std::string comment)
  Print out iteration information.
- template<class vec2_t >
  int set_step (size_t nv, vec2_t &stepv)
  Set the step sizes.

Basic usage

- virtual int mmin (size_t nv, vec_t &x0, double &fmin, func_t &func, size_t np, param_vec_t &pars)
  Calculate the minimum `fmin` of `func` w.r.t the array `x0` of size `nv` using `np` threads.

Iteration control

- virtual int next (size_t nv, vec_t &x_old, double min_old, vec_t &x_new, double min_new, double &T, size_t n_moves, bool &finished)
  Determine how to change the minimization for the next iteration.
- virtual int start (size_t nv, double &T)
  Setup initial temperature and stepsize.

Data Fields

- rng_t def_rng
  The default random number generator.

Parameters

- double boltz
  Boltzmann factor (default 1.0).
- int ntrial
  Number of iterations.
- int verbose
  Output control.
- bool out_step_changes
  Output step size changes (default false)
- bool out_best
  Output best point (default false)
- double tolx
  The independent variable tolerance (default 10^{-6})
- double T_start
  Initial temperature (default 1.0)
- double T_dec
  Factor to decrease temperature by (default 1.5)
- double step_dec
  Factor to decrease step size by (default 1.5)
- double min_step_ratio
  Ratio between minimum step size and tolx (default 100.0)

Protected Member Functions

- void func_wrapper (size_t ip)
  The function wrapper executed by thread with index `ip`.
- virtual int allocate (size_t nv, size_t np)
  Allocate memory for a minimizer over `n` dimensions with stepsize `step`.
- virtual int free (size_t np)
  Free allocated memory.
- virtual int step (size_t nv, vec_t &sx)
  Make a step to a new attempted minimum.

Protected Attributes

- std::ostream * outs
  Stream for verbose output.
- std::istream * ins
  Stream for verbose input.
- size_t nproc
  The number of threads to run.

- param_vec_t * params
  The user-specified parameters.
- size_t nvar
  The number of variables over which we minimize.
- func_t * f
  The function to minimize.
- alloc_t ao
  Allocation object.
- ovector step_vec
  Vector of step sizes.

Storage for present, next, and best vectors

- alloc_vec_t **x**
- alloc_vec_t * **new_x**
- alloc_vec_t **best_x**
- alloc_vec_t **new_E**
- alloc_vec_t **old_x**

46.4.2 Member Function Documentation

46.4.2.1 template<class param_t , class param_vec_t = std::vector<param_t>, class func_t = multi_funct<param_t>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class rng_t = gsl_rnga> virtual int anneal_mt< param_t, param_vec_t, func_t, vec_t, alloc_vec_t, alloc_t, rng_t >::print_iter (size_t *nv*, vec_t & *xx*, double *y*, int *iter*, double *tptr*, std::string *comment*) [inline, virtual]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed. If verbose>0, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. Also, if verbose>0, every time a new smallest function value is found, the location and the function value is output. If verbose>=2 then each iteration waits for a character between each trial.

Definition at line 304 of file anneal_mt.h.

The documentation for this class was generated from the following file:

- anneal_mt.h

46.5 array_2d_alloc< mat_t > Class Template Reference

A simple class to provide an `allocate()` function for 2-dimensional arrays.

```
#include <array.h>
```

46.5.1 Detailed Description

template<class mat_t>class array_2d_alloc< mat_t >

The functions here are blank, as fixed-length arrays are automatically allocated and destroyed by the compiler. This class is present to provide an analog to pointer_2d_alloc and omatrix_alloc. This class is used in `gsl_mroot_hybrids_ts.cpp` .

Definition at line 121 of file array.h.

Public Member Functions

- void allocate (mat_t &v, size_t i, size_t j)
  Allocate `v` for `i` elements.
- void free (mat_t &v, size_t i)

Free memory.

The documentation for this class was generated from the following file:

- array.h

46.6 array_2d_col< R, C, data_t > Class Template Reference

Column of a 2d array.

`#include <array.h>`

46.6.1 Detailed Description

template<size_t R, size_t C, class data_t = double>class array_2d_col< R, C, data_t >

This works because two-dimensional arrays are always continguous (as indicated in appendix C of Soustroup's book).

Definition at line 367 of file array.h.

Public Member Functions

- array_2d_col (data_t mat[R][C], size_t i)
  Create an object as the `i`th column of `mat`.
- data_t & operator[ ] (size_t i)
  Array-like indexing.
- const data_t & operator[ ] (size_t i) const
  Array-like indexing.

Protected Attributes

- data_t * a
  The array pointer.

The documentation for this class was generated from the following file:

- array.h

46.7 array_2d_row< array_2d_t, data_t > Class Template Reference

Row of a 2d array.

`#include <array.h>`

46.7.1 Detailed Description

template<class array_2d_t, class data_t = double>class array_2d_row< array_2d_t, data_t >

Definition at line 401 of file array.h.

Public Member Functions

- array_2d_row (array_2d_t &mat, size_t i)
  Create an object as the ith row of mat.
- data_t & operator[ ] (size_t i)
  Array-like indexing, returns the element in the ith column of the chosen row.
- const data_t & operator[ ] (size_t i) const
  Array-like indexing, returns the element in the ith column of the chosen row.

Protected Attributes

- data_t * a
  The array pointer.

The documentation for this class was generated from the following file:

- array.h

46.8 array_alloc< vec_t > Class Template Reference

A simple class to provide an `allocate()` function for arrays.

`#include <array.h>`

46.8.1 Detailed Description

template<class vec_t>class array_alloc< vec_t >

The functions here are blank, as fixed-length arrays are automatically allocated and destroyed by the compiler. This class is present to provide an analog to pointer_alloc and ovector_alloc . This class is used, for example, for sma_interp_vec .

Idea for Future Might it be possible to rework this so that it does range checking and ensures that the user doesn't try to allocate more or less space? I.e. array_alloc<double[2]> complains if you try an allocate(x,3)?

Definition at line 104 of file array.h.

Public Member Functions

- void allocate (vec_t &v, size_t i)
  Allocate v for i elements.
- void free (vec_t &v)
  Free memory.

The documentation for this class was generated from the following file:

- array.h

46.9 array_const_reverse< sz > Class Template Reference

A simple class which reverses the order of an array.

`#include <array.h>`

46.9.1 Detailed Description

template<size_t sz>class array_const_reverse< sz >

See an example of usage in `interp_ts.cpp` and `smart_interp_ts.cpp`.

Definition at line 294 of file array.h.

Public Member Functions

- array_const_reverse (const double *arr)
  Create a reversed array from `arr` of size `sz`.
- const double & operator[ ] (size_t i) const
  Array-like indexing.

Protected Attributes

- double * a
  The array pointer.

The documentation for this class was generated from the following file:

- array.h

46.10 array_const_subvector Class Reference

A simple subvector class for a const array (without error checking)

`#include <array.h>`

46.10.1 Detailed Description

Idea for Future Make the member data truly const and remove the extra typecast.

Definition at line 441 of file array.h.

Public Member Functions

- array_const_subvector (const double *arr, size_t offset, size_t n)
  Create a reversed array from `arr` of size `sz`.
- const double & operator[ ] (size_t i) const
  Array-like indexing.

Protected Attributes

- double * a
  The array pointer.
- size_t off
  The offset.
- size_t len
  The subvector length.

The documentation for this class was generated from the following file:

- array.h

46.11 array_const_subvector_reverse Class Reference

Reverse a subvector of a const array.

`#include <array.h>`

46.11.1 Detailed Description

Definition at line 519 of file array.h.

Public Member Functions

- array_const_subvector_reverse (const double *arr, size_t offset, size_t n)
  Create a reversed array from `arr` of size `sz`.
- const double & operator[ ] (size_t i) const
  Array-like indexing.

Protected Attributes

- double * a
  The array pointer.
- size_t off
  The offset.
- size_t len
  The subvector length.

The documentation for this class was generated from the following file:

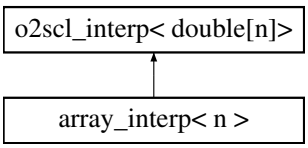
- array.h

46.12 array_interp< n > Class Template Reference

A specialization of o2scl_interp for C-style double arrays.

`#include <interp.h>`

Inheritance diagram for array_interp< n >:



46.12.1 Detailed Description

template<size_t n>class array_interp< n >

Definition at line 1434 of file interp.h.

Public Member Functions

- array_interp (base_interp_mgr< double[n]> &it)
  Create with base interpolation objects `it` and `rit`.
- array_interp ()
  Create an interpolator using the default base interpolation objects.

The documentation for this class was generated from the following file:

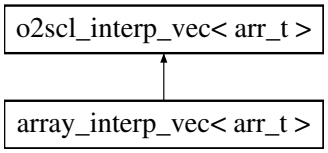
- interp.h

46.13 array_interp_vec< arr_t > Class Template Reference

A specialization of o2scl_interp_vec for C-style arrays.

`#include <interp.h>`

Inheritance diagram for array_interp_vec< arr_t >:



46.13.1 Detailed Description

template<class arr_t>class array_interp_vec< arr_t >

Definition at line 1451 of file interp.h.

Public Member Functions

- array_interp_vec (base_interp_mgr< arr_t > &it, size_t nv, const arr_t &x, const arr_t &y)
  Create with base interpolation object `it`.

The documentation for this class was generated from the following file:

- interp.h

46.14 array_reverse< sz > Class Template Reference

A simple class which reverses the order of an array.

`#include <array.h>`

46.14.1 Detailed Description

template<size_t sz>class array_reverse< sz >

See an example of usage in `interp_ts.cpp` and `smart_interp_ts.cpp`.

Definition at line 257 of file array.h.

Public Member Functions

- array_reverse (double *arr)
  Create a reversed array from `arr` of size `sz`.
- double & operator[ ] (size_t i)
  Array-like indexing.
- const double & operator[ ] (size_t i) const
  Array-like indexing.

Protected Attributes

- double * a
  The array pointer.

The documentation for this class was generated from the following file:

- array.h

46.15 array_subvector Class Reference

A simple subvector class for an array (without error checking)

`#include <array.h>`

46.15.1 Detailed Description

Definition at line 322 of file array.h.

Public Member Functions

- array_subvector (double *arr, size_t offset, size_t n)
  Create a reversed array from `arr` of size `sz`.
- double & operator[ ] (size_t i)
  Array-like indexing.
- const double & operator[ ] (size_t i) const
  Array-like indexing.

Protected Attributes

- double * a
  The array pointer.
- size_t off
  The offset.
- size_t len
  The subvector length.

The documentation for this class was generated from the following file:

- array.h

46.16 array_subvector_reverse Class Reference

Reverse a subvector of an array.

`#include <array.h>`

46.16.1 Detailed Description

Definition at line 477 of file array.h.

Public Member Functions

- array_subvector_reverse (double *arr, size_t offset, size_t n)
  Create a reversed array from `arr` of size `sz`.
- double & operator[ ] (size_t i)
  Array-like indexing.
- const double & operator[ ] (size_t i) const
  Array-like indexing.

Protected Attributes

- double * a
  The array pointer.
- size_t off
  The offset.
- size_t len
  The subvector length.

The documentation for this class was generated from the following file:

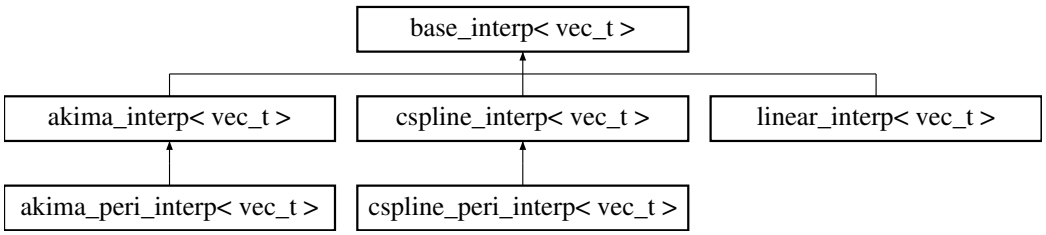
- array.h

46.17 base_interp< vec_t > Class Template Reference

Base low-level interpolation class [abstract base].

`#include <interp.h>`

Inheritance diagram for base_interp< vec_t >:



46.17.1 Detailed Description

template<class vec_t>class base_interp< vec_t >

For any pair of vectors `x` and `y` into which you would like to interpolate, you need to call allocate() and init() first, and then the interpolation functions, and then free(). If the next pair of vectors has the same size, then you need only to call init() before the next call to an interpolation function (you don't need to reallocate). If the `x` and `y` vectors do not change, then you may call the interpolation functions in succession.

These interpolation class work so long as the vector `x` is either monotonically increasing or monotonically decreasing. All of the descendants give identical results to the GSL interpolation routines when the vector is monotonically increasing.

Definition at line 56 of file interp.h.

Public Member Functions

- virtual int allocate (size_t size)
  Allocate memory, assuming x and y have size `size`.
- virtual int free ()
  Free allocated memory.
- virtual int init (const vec_t &x, const vec_t &y, size_t size)
  Initialize interpolation routine.
- virtual int interp (const vec_t &x, const vec_t &y, size_t size, double x0, double &y0)=0
  Give the value of the function $y(x=x_0)$.
- virtual int deriv (const vec_t &x, const vec_t &y, size_t size, double x0, double &dydx)=0
  Give the value of the derivative $y'(x=x_0)$.
- virtual int deriv2 (const vec_t &x, const vec_t &y, size_t size, double x0, double &d2ydx2)=0
  Give the value of the second derivative $y''(x=x_0)$.
- virtual int integ (const vec_t &x, const vec_t &y, size_t size, double a, double b, double &result)=0
  Give the value of the integral $\int_a^b y(x)\,dx$.
- virtual const char * type ()
  Return the type, `"base_interp"`.

Data Fields

- size_t min_size
  The minimum size of the vectors to interpolate between.

Protected Member Functions

- double integ_eval (double ai, double bi, double ci, double di, double xi, double a, double b)
  An internal function to assist in computing the integral for both the cspline and Akima types.

Protected Attributes

- search_vec< vec_t > * sv
  To perform binary searches.
- size_t n_sv
  The size of the allocated search_vec object.
- bool mem_alloc
  True if memory has been allocated.

Private Member Functions

- **base_interp** (const base_interp< vec_t > &)
- base_interp< vec_t > & **operator=** (const base_interp< vec_t > &)

46.17.2 Field Documentation

46.17.2.1 template<class vec_t> size_t base_interp< vec_t >::min_size

This needs to be set in the constructor of the children for access by the class user

Definition at line 109 of file interp.h.

The documentation for this class was generated from the following file:

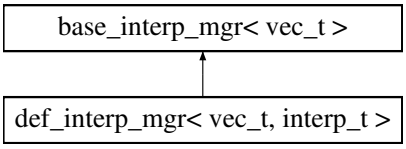
- interp.h

46.18 base_interp_mgr< vec_t > Class Template Reference

A base interpolation object manager [abstract base].

`#include <interp.h>`

Inheritance diagram for base_interp_mgr< vec_t >:



46.18.1 Detailed Description

template<class vec_t>class base_interp_mgr< vec_t >

Definition at line 1082 of file interp.h.

Public Member Functions

- virtual base_interp< vec_t > * new_interp ()=0
  Create a new interpolation object.
- virtual int free_interp (base_interp< vec_t > *b)
  Deallocate an interpolation object.

The documentation for this class was generated from the following file:

- interp.h

46.19 bin_size Class Reference

Determine bin size (CERNLIB)

`#include <bin_size.h>`

46.19.1 Detailed Description

This is adapted from the KERNLIB routine `binsiz.f` written by F. James.

This class computes an appropriate set of histogram bins given the upper and lower limits of the data and the maximum number of bins. The bin width is always an integral power of ten times 1, 2, 2.5 or 5. The bin width may also be specified by the user, in which case the class only computes the appropriate limits.

Note

This class is not working yet.

Idea for Future Finish this.

Definition at line 48 of file bin_size.h.

Public Member Functions

- int calc_bin (double al, double ah, int na, double &bl, double &bh, int &nb, double &bwid)
  Compute bin size.

Data Fields

- bool cern_mode
  (default true)

46.19.2 Member Function Documentation

46.19.2.1 int bin_size::calc_bin (double *al,* double *ah,* int *na,* double & *bl,* double & *bh,* int & *nb,* double & *bwid*)

- `al` - Lower limit of data
- `ah` - Upper limit of data
- `na` - Maximum number of bins desired.
- `bl` - Lower limit (BL<=AL)
- `bh` - Upper limit (BH>=AH)
- `nb` - Number of bins determined by BINSIZ (NA/2<NB<=NA)
- `bwid` - Bin width (BH-BL)/NB

If `na=0` or `na=-1`, this function always makes exactly one bin.

If `na=1`, this function takes `bwid` as input and determines only `bl`, `hb`, and `nb`. This is especially useful when it is desired to have the same bin width for several histograms (or for the two axes of a scatter-plot).

If `al` > `ah`, this function takes `al` to be the upper limit and `ah` to be the lower limit, so that in fact `al` and `ah` may appear in any order. They are not changed by calc_bin(). If `al` = `ah`, the lower limit is taken to be `al`, and the upper limit is set to `al+1`.

If cern_mode is true (which is the default) the starting guess for the number of bins is `na-1`. Otherwise, the starting guess for the number of bins is `na`.

The documentation for this class was generated from the following file:

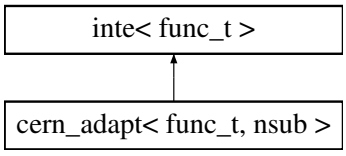
- bin_size.h

46.20 cern_adapt< func_t, nsub > Class Template Reference

Adaptive integration (CERNLIB)

`#include <cern_adapt.h>`

Inheritance diagram for cern_adapt< func_t, nsub >:



46.20.1 Detailed Description

template<class func_t = funct, size_t nsub = 100>class cern_adapt< func_t, nsub >

Uses a base integration object (default is cern_gauss56) to perform adaptive integration by automatically subdividing the integration interval. At each step, the interval with the largest absolute uncertainty is divided in half. The routine succeeds if the absolute tolerance is less than tol_abs or if the relative tolerance is less than tol_rel, i.e.

$$\text{err} \leq \text{tol\_abs} \text{ or } \text{err} \leq \text{tol\_rel} \cdot |I|$$

where I is the current estimate for the integral and err is the current estimate for the uncertainty. If the number of subdivisions exceeds the template parameter `nsub`, the error handler is called, since the integration may not have been successful. The number of subdivisions used in the last integration can be obtained from get_nsubdivisions().

The template parameter `nsub`, is the maximum number of subdivisions. It is automatically set to 100 in the original CERNLIB routine, and defaults to 100 here. The default base integration object is of type cern_gauss56. This is the CERNLIB default, but can be modified by calling set_inte().

This class is based on the CERNLIB routines RADAPT and DADAPT which are documented at `http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d102/top.html`

Idea for Future

- Allow user to set the initial subdivisions?
- It might be interesting to directly compare the performance of this class to gsl_inte_qag .
- There is a fixme entry in the code which could be resolved.
- Output the point where most subdividing was required?

Definition at line 71 of file cern_adapt.h.

Public Member Functions

Basic usage

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` giving result `res` and error `err`.

Protected Attributes

- double xlo [nsub]
  Lower end of subdivision.
- double xhi [nsub]
  High end of subdivision.
- double tval [nsub]
  Value of integral for subdivision.
- double ters [nsub]
  Squared error for subdivision.
- int prev_subdiv
  Previous number of subdivisions.
- inte< func_t > * it
  The base integration object.

Integration object

- cern_gauss56< func_t > def_inte
  Default integration object.
- int set_inte (inte< func_t > &i)
  Set the base integration object to use.

Subdivisions

- size_t nsubdiv
  Number of subdivisions.
- size_t get_nsubdivisions ()
  Return the number of subdivisions used in the last integration.
- int get_ith_subdivision (size_t i, double &xlow, double &xhigh, double &value, double &errsq)
  Return the ith subdivision.
- template<class vec_t >
  int get_subdivisions (vec_t &xlow, vec_t &xhigh, vec_t &value, vec_t &errsq)
  Return all of the subdivisions.

46.20.2 Field Documentation

46.20.2.1 template<class func_t = funct, size_t nsub = 100> size_t cern_adapt< func_t, nsub >::nsubdiv

The options are

- 0: Use previous binning and do not subdivide further
- 1: Automatic - adapt until tolerance is attained (default)
- n: (n>1) split first in n equal subdivisions, then adapt until tolerance is obtained.

Definition at line 266 of file cern_adapt.h.

The documentation for this class was generated from the following file:

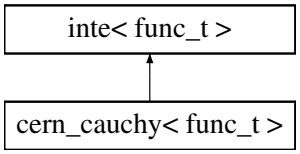
- cern_adapt.h

46.21 cern_cauchy< func_t > Class Template Reference

Cauchy principal value integration (CERNLIB)

`#include <cern_cauchy.h>`

Inheritance diagram for cern_cauchy< func_t >:



46.21.1 Detailed Description

template<class func_t>class cern_cauchy< func_t >

The location of the singularity must be specified before-hand in cern_cauchy::s, and the singularity must not be at one of the endpoints. Note that when integrating a function of the form $\frac{f(x)}{(x-s)}$, the denominator $(x-s)$ must be specified in the argument `func` to integ(). This is different from how the gsl_inte_qawc operates.

The method from Longman58 is used for the decomposition of the integral, and the resulting integrals are computed using a user-specified base integration object.

The uncertainty in the integral is not calculated, and is always given as zero. The default base integration object is of type cern_gauss. This is the CERNLIB default, but can be modified by calling set_inte(). If the singularity is outside the region of integration, then the result from the base integration object is returned without calling the error handler.

Possible errors for integ() and integ_err():

- gsl_einval - Singularity is on an endpoint
- gsl_efailed - Couldn't reach requested accuracy (occurs only if inte::err_nonconv is true)

This function is based on the CERNLIB routines RCAUCH and DCAUCH which are documented at `http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d104/top.html`

Definition at line 63 of file cern_cauchy.h.

Public Member Functions

- int set_inte (inte< func_t > &i)
  Set the base integration object to use (default is cern_cauchy::def_inte of type cern_gauss)
- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` giving result `res` and error `err`.
- virtual double integ (func_t &func, double a, double b)
  Integrate function `func` from `a` to `b`.

Data Fields

- double s
  The singularity (must be set before calling integ() or integ_err())
- cern_gauss< func_t > def_inte
  Default integration object.

Protected Attributes

- inte< func_t > * it
  The base integration object.

Integration constants

- double **x** [12]
- double **w** [12]

The documentation for this class was generated from the following file:

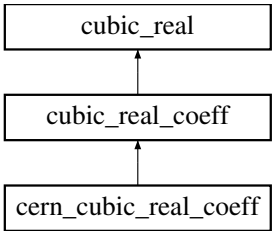
- cern_cauchy.h

46.22 cern_cubic_real_coeff Class Reference

Solve a cubic with real coefficients and complex roots (CERNLIB)

```
#include <poly.h>
```

Inheritance diagram for cern_cubic_real_coeff:



46.22.1 Detailed Description

Note

The function rrteq3() is based on the CERNLIB routine of the same name, but differs slightly. See the documtation of that function for details.

Definition at line 380 of file poly.h.

Public Member Functions

- virtual int solve_rc (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)
  Solves the polynomial $a_3x^3+b_3x^2+c_3x+d_3=0$ giving the real solution $x=x_1$ and two complex solutions $x=x_1$, $x=x_2$, and $x=x_3$.
- virtual int rrteq3 (double r, double s, double t, double x[], double &d)
  The CERNLIB-like interface.
- const char * type ()
  Return a string denoting the type ("cern_cubic_real_coeff")

Data Fields

- double eps
  Numerical tolerance (default 10^{-6})
- double delta
  Numerical tolerance (default 10^{-15})
- bool improve_scale
  Improve algorithm for poorly-scaled roots (default true)

46.22.2 Member Function Documentation

46.22.2.1 virtual int cern_cubic_real_coeff::rrteq3 (double *r*, double *s*, double *t*, double *x[]*, double & *d*) [virtual]

This function computes the roots of the cubic equation

$$x^3+rx^2+sx+t=0$$

returning the value of the discriminant in `d` and the roots in the array `x`. If the discriminant is negative, then all three real roots are stored in `x`. Otherwise, the real root is stored in `x[0]` and the real and imaginary parts of the complex conjugate roots are stored in `x[1]` and `x[2]`, respectively. This differs from the CERNLIB routine where the results were stored in `x[1]`, `x[2]`, and `x[3]` instead.

Another small change is that the discriminant for the resolvent cubic is evaluated slightly differently in order to improve the properties in the case where the roots are not all of order unity. The default CERNLIB behavior can be restored by setting improve_scale to `false`.

The documentation for this class was generated from the following file:

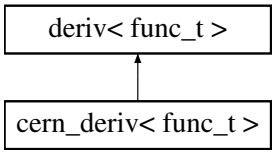
- poly.h

46.23 cern_deriv< func_t > Class Template Reference

Numerical differentiation routine (CERNLIB)

`#include <cern_deriv.h>`

Inheritance diagram for cern_deriv< func_t >:



46.23.1 Detailed Description

template<class func_t = funct>class cern_deriv< func_t >

This uses Romberg extrapolation to compute the derivative with the finite-differencing formula

$$f'(x) = [f(x+h) - f(x-h)]/(2h)$$

If deriv::verbose is greater than zero, then each iteration prints out the extrapolation table, and if deriv::verbose is greater than 1, then a keypress is required at the end of each iteration.

For sufficiently difficult functions, the derivative computation can fail, and will call the error handler and return zero with zero error.

Based on the CERNLIB routine DERIV, which was based on Rutishauser63 and is documented at `http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d401/top.html`

An example demonstrating the usage of this class is given in `examples/ex_deriv.cpp` and the Numerical differentiation .

If deriv::verbose is greater than zero, at each iteration this class prints something similar to

```
cern_deriv, iteration: 1
(hh, a[k], derivative) list:
-4.193459e-05 4.387643e-14 8.775286e-01
-2.995402e-05 4.387792e-14 8.775585e-01
-1.048405e-05 4.387845e-14 8.775690e-01
-7.488654e-06 4.387882e-14 8.775765e-01
-2.621038e-06 4.387895e-14 8.775791e-01
-1.872173e-06 4.387905e-14 8.775810e-01
-6.552611e-07 4.387908e-14 8.775817e-01
-4.680438e-07 4.387910e-14 8.775821e-01
-1.638153e-07 4.387911e-14 8.775823e-01
```

If deriv::verbose is greater than 1, a keypress is required after each iteration.

Note

Second and third derivatives are computed by naive nested applications of the formula for the first derivative. No uncertainty for these derivatives is provided.

Idea for Future All of the coefficients appear to be fractions which could be replaced with exact representation?

Idea for Future Record the number of function calls?

Definition at line 91 of file cern_deriv.h.

Public Member Functions

- virtual int calc_err (double x, func_t &func, double &dfdx, double &err)
  Calculate the first derivative of `func` w.r.t. x and the uncertainty.
- virtual const char * type ()
  Return string denoting type ("cern_deriv")

Data Fields

- double delta
  A scaling factor (default 1.0)
- double eps
  Extrapolation tolerance (default is 5×10^{14}*)*

Protected Member Functions

- template<class func2_t >
  int calc_base (double x, func2_t &func, double &dfdx, double &err)
  Internal template version of the derivative function.
- virtual int calc_err_int (double x, funct &func, double &dfdx, double &err)
  Calculate the first derivative of `func` w.r.t. x.

Protected Attributes

Storage for the fixed coefficients

- double **dx** [10]
- double **w** [10][4]

46.23.2 Member Function Documentation

46.23.2.1 template<class func_t = funct> virtual int cern_deriv< func_t >::calc_err_int (double *x*, funct & *func*, double & *dfdx*, double & *err*) [inline, protected, virtual]

This is an internal version of calc() which is used in computing second and third derivatives

Implements deriv< func_t >.

Definition at line 249 of file cern_deriv.h.

The documentation for this class was generated from the following file:

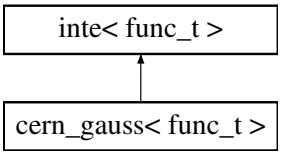
- cern_deriv.h

46.24 cern_gauss< func_t > Class Template Reference

Gaussian quadrature (CERNLIB)

`#include <cern_gauss.h>`

Inheritance diagram for cern_gauss< func_t >:



46.24.1 Detailed Description

template<class func_t = funct>class cern_gauss< func_t >

For any interval (a,b), we define $g_8(a,b)$ and $g_{16}(a,b)$ to be the 8- and 16-point Gaussian quadrature approximations to

$$I = \int_a^b f(x)\,dx$$

and define

$$r(a,b) = \frac{|g_{16}(a,b) - g_8(a,b)|}{1 + g_{16}(a,b)}$$

The function integ() returns G given by

$$G = \sum_{i=1}^{k} g_{16}(x_{i-1}, x_i)$$

where $x_0 = a$ and $x_k = b$ and the subdivision points x_i are given by

$$x_i = x_{i-1} + \lambda\,(B - x_{i-1})$$

where λ is the first number in the sequence $1, \frac{1}{2}, \frac{1}{4}, \ldots$ for which

$$r(x_{i-1}, x_i) < \text{eps}.$$

If, at any stage, the ratio

$$q = \left| \frac{x_i - x_{i-1}}{b - a} \right|$$

is so small so that $1 + 0.005q$ is indistinguishable from unity, then the accuracy is required is not reachable and the error handler is called.

Unless there is severe cancellation, inte::tol_rel may be considered as specifying a bound on the relative error of the integral in the case that $|I| > 1$ and an absolute error if $|I| < 1$. More precisely, if k is the number of subintervals from above, and if

$$I_{abs} = \int_a^b |f(x)|\,dx$$

then

$$\frac{|G - I|}{I_{abs} + k} < \text{tol}_\text{r}\text{el}$$

will nearly always be true when no error is returned. For functions with no singualarities in the interval, the accuracy will usually be higher than this.

If the desired accuracy is not achieved, the integration functions will call the error handler and return the best guess, unless inte::err_nonconv is false, in which case the error handler is not called.

This function is based on the CERNLIB routines GAUSS and DGAUSS which are documented at http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d103/top.html

Idea for Future Allow user to change `cst`?

Definition at line 93 of file cern_gauss.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` giving result `res` and error `err`.
- virtual double integ (func_t &func, double a, double b)
  Integrate function `func` from `a` to `b`.

Protected Attributes

Integration constants (Set in the constructor)

- double **x** [12]
- double **w** [12]

The documentation for this class was generated from the following file:

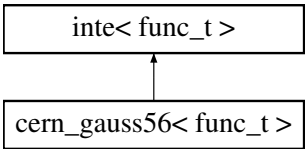
- cern_gauss.h

46.25 cern_gauss56< func_t > Class Template Reference

5,6-point Gaussian quadrature (CERNLIB)

`#include <cern_gauss56.h>`

Inheritance diagram for cern_gauss56< func_t >:



46.25.1 Detailed Description

template<class func_t>class cern_gauss56< func_t >

If I_5 is the 5-point approximation, and I_6 is the 6-point approximation to the integral, then integ_err() returns the result $\frac{1}{2}(I_5+I_6)$ with uncertainty $|I_5-I_6|$.

This class is based on the CERNLIB routines RGS56P and DGS56P which are documented at `http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d106/top.html`

Definition at line 44 of file cern_gauss56.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` giving result `res` and error `err`.

Protected Attributes

Integration constants

- double **x5** [5]
- double **w5** [5]
- double **x6** [6]
- double **w6** [6]

46.25.2 Member Function Documentation

46.25.2.1 template<class func_t > virtual int cern_gauss56< func_t >::integ_err (func_t & *func*, double *a*, double *b*, double & *res*, double & *err*) [inline, virtual]

This function always returns gsl_success.

Implements inte< func_t >.

Definition at line 79 of file cern_gauss56.h.

The documentation for this class was generated from the following file:

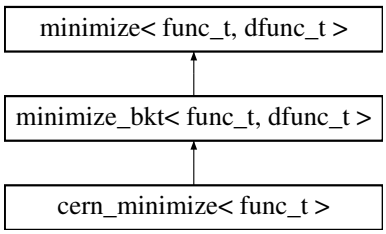
- cern_gauss56.h

46.26 cern_minimize< func_t > Class Template Reference

One-dimensional minimization (CERNLIB)

`#include <cern_minimize.h>`

Inheritance diagram for cern_minimize< func_t >:



46.26.1 Detailed Description

template<class func_t = funct>class cern_minimize< func_t >

The golden section search is applied in the interval (a,b) using a fixed number n of function evaluations where

$$n = \left[2.08 \ln(|a-b|/\text{tol\_abs}) + \frac{1}{2}\right] + 1$$

The accuracy depends on the function. A choice of tol_abs $> 10^{-8}$ usually results in a relative error of x which is smaller than or of the order of tol_abs .

This routine strictly searches the interval (a,b) . If the function is nowhere flat in this interval, then min_bkt() will return either a or b and min_type is set to 1. Note that if there are more than 1 local minima in the specified interval, there is no guarantee that this method will find the global minimum.

Note

The number of function evaluations can be quite large if multi_min::tol_abs is sufficiently small. If multi_min::tol_abs is exactly zero, then the error handler will be called.

Based on the CERNLIB routines RMINFC and DMINFC, which was based on Fletcher87, and Krabs83 and is documented at http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/d503/top.html

Definition at line 69 of file cern_minimize.h.

Public Member Functions

- virtual int min_bkt (double &x, double a, double b, double &y, func_t &func)
  Calculate the minimum `min` *of* `func` *between* `a` *and* `b`.
- int set_delta (double d)
  Set the value of δ.
- virtual const char * type ()
  Return string denoting type ("cern_minimize")

Data Fields

- int min_type
  Type of minimum found.

Protected Member Functions

- int nint (double x)
  C analog of Fortran's "Nearest integer" function.

Protected Attributes

- double delta
  The value of delta as specified by the user.
- bool delta_set
  True if the value of delta has been set.

46.26.2 Member Function Documentation

46.26.2.1 template<class func_t = funct> virtual int cern_minimize< func_t >::min_bkt (double & *x*, double *a*, double *b*, double & *y*, func_t & *func*) [inline, virtual]

The initial value of `x` is ignored.

If there is no minimum in the given interval, then on exit `x` will be equal to either `a` or `b` and min_type will be set to 1 instead of zero. The error handler is not called, as this need not be interpreted as an error.

Implements minimize_bkt< func_t, dfunc_t >.

Definition at line 106 of file cern_minimize.h.

46.26.2.2 template<class func_t = funct> int cern_minimize< func_t >::set_delta (double *d*) [inline]

If this is not called before min_bkt() is used, then the suggested value $\delta = 10\text{tol}_a\text{bs}$ is used.

Definition at line 183 of file cern_minimize.h.

The documentation for this class was generated from the following file:

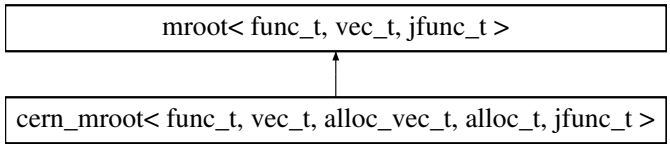
- cern_minimize.h

46.27 cern_mroot< func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t > Class Template Reference

Multi-dimensional mroot-finding routine (CERNLIB)

```
#include <cern_mroot.h>
```

Inheritance diagram for cern_mroot< func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >:



46.27.1 Detailed Description

template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class jfunc_t = jac_funct<vec_t,omatrix_base>>class cern_mroot< func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >

If x_i denotes the current iteration, and x_i' denotes the previous iteration, then the calculation is terminated if either of the following tests is successful

$$1: \quad \max|f_i(x)| \leq \text{tol\_rel}$$
$$2: \quad \max|x_i - x_i'| \leq \text{tol\_abs} \times \max|x_i|$$

This routine treats the functions specified as a mm_funct object slightly differently than gsl_mroot_hybrids. First the equations should be numbered (as much as is possible) in order of increasing nonlinearity. Also, instead of calculating all of the equations on each function call, only the equation specified by the `size_t` parameter needs to be calculated. If the equations are specified as

$$0 = f_0(x_0, x_1, ..., x_{n-1})$$
$$0 = f_1(x_0, x_1, ..., x_{n-1})$$
$$...$$
$$0 = f_{n-1}(x_0, x_1, ..., x_{n-1})$$

then when the `size_t` argument is given as `i`, then only the function f_i needs to be calculated.

Warning

This code has not been checked to ensure that it cannot fail to solve the equations without calling the error handler and returning a non-zero value. Until then, the solution may need to be checked explicitly by the caller.

There is an example for the usage of the multidimensional solver classes given in `examples/ex_mroot.cpp`, see Multi-dimensional solver .

Idea for Future Modify this so it handles functions which return non-zero values.

Idea for Future Move some of the memory allocation out of msolve()

Idea for Future Give the user access to the number of function calls

Idea for Future Rename nier6, nier7, and nier8 to something sensible.

Idea for Future It may be that the O_2scl native Householder transformations should be used here instead of the inline version given here.

Based on the CERNLIB routines RSNLEQ and DSNLEQ, which was based on More79 and More80 and is documented at http://wwwasdoc.web.cern.ch/wwwasdoc/shortwrupsdir/c201/top.html

Definition at line 92 of file cern_mroot.h.

Public Member Functions

- int get_info ()
  Get the value of `INFO` from the last call to msolve()
- std::string get_info_string ()
  Get the a string corresponding to the integer returned by cern_mroot::get_info().
- virtual const char * type ()
  Return the type, "`cern_mroot`".
- virtual int msolve (size_t nvar, vec_t &x, func_t &func)
  Solve `func` using `x` as an initial guess, returning `x`.

Data Fields

- int maxf
  Maximum number of function evaluations.
- double scale
  The original scale parameter from CERNLIB (default 10.0)
- double eps
  The smallest floating point number (default $\sim 1.49012 \times 10^{-8}$ *)*

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- umatrix w
  Desc.
- int info
  Internal storage for the value of `info`.
- int mpt [289]
  Store the number of function evaluations.

46.27.2 Member Function Documentation

46.27.2.1 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class jfunc_t = jac_funct<vec_t,omatrix_base>> int cern_mroot< func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >::get_info ( ) [inline]

The value of info is assigned according to the following list. The values 1-8 are the standard behavior from CERNLIB. 0 - The function solve() has not been called. 1 - Test 1 was successful.

2 - Test 2 was successful.

3 - Both tests were successful.

4 - Number of iterations is greater than cern_mroot_root::maxf.

5 - Approximate (finite difference) Jacobian matrix is singular.

6 - Iterations are not making good progress.

7 - Iterations are diverging.

8 - Iterations are converging, but either cern_mroot_root::tol_abs is too small or the Jacobian is nearly singular or the variables are badly scaled.

9 - Either root::tol_rel or root::tol_abs is not greater than zero or the specified number of variables is ≤ 0.

The return values returned by msolve() corresponding to the values of `INFO` above are 1 - gsl_success 2 - gsl_success 3 - gsl_success 4 - gsl_emaxiter 5 - gsl_esing 6 - gsl_enoprog 7 - gsl_erunaway 8 - gsl_efailed 9 - gsl_einval

Definition at line 165 of file cern_mroot.h.

46.27.3 Field Documentation

46.27.3.1 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class jfunc_t = jac_funct<vec_t,omatrix_base>> int cern_mroot< func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >::maxf

If maxf ≤ 0 , then $50(\mathrm{nv}+3)$ (which is the CERNLIB default) is used. The default value of `maxf` is zero which then implies the default from CERNLIB.

Definition at line 200 of file cern_mroot.h.

46.27.3.2 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class jfunc_t = jac_funct<vec_t,omatrix_base>> double cern_mroot< func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >::eps

The original prescription from CERNLIB for `eps` is given below:

```
#if !defined(CERNLIB_DOUBLE)
PARAMETER (EPS =  0.84293 69702 17878 97282 52636 392E-07)
#endif
#if defined(CERNLIB_IBM)
PARAMETER (EPS =  0.14901 16119 38476 562D-07)
#endif
#if defined(CERNLIB_VAX)
PARAMETER (EPS =  0.37252 90298 46191 40625D-08)
#endif
#if (defined(CERNLIB_UNIX))&&(defined(CERNLIB_DOUBLE))
PARAMETER (EPS =  0.14901 16119 38476 600D-07)
#endif
```

Definition at line 229 of file cern_mroot.h.

The documentation for this class was generated from the following file:

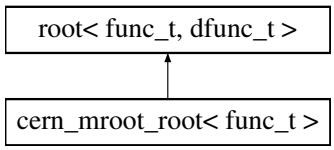
- cern_mroot.h

46.28 cern_mroot_root< func_t > Class Template Reference

One-dimensional version of cern_mroot.

`#include <cern_mroot_root.h>`

Inheritance diagram for cern_mroot_root< func_t >:



46.28.1 Detailed Description

template<class func_t = funct>class cern_mroot_root< func_t >

This one-dimensional root-finding routine, based on cern_mroot, is probably slower than the more typical 1-d routines, but also tends to converge for a larger class of functions than cern_root, gsl_root_brent, or gsl_root_stef. It has been modified from cern_mroot and slightly optimized, but has the same basic behavior.

If x_i denotes the current iteration, and x_i' denotes the previous iteration, then the calculation is terminated if either (or both) of the following tests is successful

$$1: \quad \max|f_i(x)| \leq \text{tol\_rel}$$

$$2: \quad \max|x_i - x_i'| \leq \text{tol\_abs} \times \max|x_i|$$

Note

This code has not been checked to ensure that it cannot fail to solve the equations without calling the error handler and returning a non-zero value. Until then, the solution may need to be checked explicitly by the caller.

Idea for Future Double-check this class to make sure it cannot fail while returning 0 for success.

Definition at line 64 of file cern_mroot_root.h.

Public Member Functions

- int get_info ()
  Get the value of `INFO` from the last call to solve() (default 0)
- virtual const char * type ()
  Return the type, "`cern_mroot_root`".
- virtual int solve (double &ux, func_t &func)
  Solve `func` using `x` as an initial guess, returning `x`.

Data Fields

- int maxf
  Maximum number of function evaluations.
- double scale
  The original scale parameter from CERNLIB (default 10.0)
- double eps
  The smallest floating point number (default $\sim 1.49012 \times 10^{-8}$*)*

Protected Attributes

- int info
  Internal storage for the value of `info`.

46.28.2 Member Function Documentation

46.28.2.1 template<class func_t = funct> int cern_mroot_root< func_t >::get_info ( ) [inline]

The value of info is assigned according to the following list. The values 1-8 are the standard behavior from CERNLIB. 0 - The function solve() has not been called. 1 - Test 1 was successful.

2 - Test 2 was successful.

3 - Both tests were successful.

4 - Number of iterations is greater than cern_mroot_root::maxf.

5 - Approximate (finite difference) Jacobian matrix is singular.

6 - Iterations are not making good progress.

7 - Iterations are diverging.

8 - Iterations are converging, but either cern_mroot_root::tol_abs is too small or the Jacobian is nearly singular or the variables are badly scaled.

9 - Either root::tol_rel or root::tol_abs is not greater than zero.

Definition at line 102 of file cern_mroot_root.h.

46.28.3 Field Documentation

46.28.3.1 template<class func_t = funct> int cern_mroot_root< func_t >::maxf

If maxf ≤ 0, then 200 (which is the CERNLIB default) is used. The default value of `maxf` is zero which then implies the default from CERNLIB.

Definition at line 110 of file cern_mroot_root.h.

46.28.3.2 template<class func_t = funct> double cern_mroot_root< func_t >::eps

The original prescription from CERNLIB for `eps` is given below:

```
#if !defined(CERNLIB_DOUBLE)
PARAMETER (EPS =  0.84293 69702 17878 97282 52636 392E-07)
#endif
#if defined(CERNLIB_IBM)
PARAMETER (EPS =  0.14901 16119 38476 562D-07)
#endif
#if defined(CERNLIB_VAX)
PARAMETER (EPS =  0.37252 90298 46191 40625D-08)
#endif
#if (defined(CERNLIB_UNIX))&&(defined(CERNLIB_DOUBLE))
PARAMETER (EPS =  0.14901 16119 38476 600D-07)
#endif
```

Idea for Future This number should probably default to one of the GSL tolerances.

Definition at line 142 of file cern_mroot_root.h.

The documentation for this class was generated from the following file:

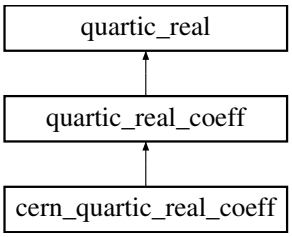
- cern_mroot_root.h

46.29 cern_quartic_real_coeff Class Reference

Solve a quartic with real coefficients and complex roots (CERNLIB)

`#include <poly.h>`

Inheritance diagram for cern_quartic_real_coeff:



46.29.1 Detailed Description

Definition at line 440 of file poly.h.

Public Member Functions

- virtual int solve_rc (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)

Solves the polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four complex solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.

- virtual int rrteq4 (double a, double b, double c, double d, std::complex< double > z[ ], double &dc, int &mt)
  The CERNLIB-like interface.
- const char * type ()
  Return a string denoting the type ("cern_quartic_real_coeff")

Protected Attributes

- cern_cubic_real_coeff cub_obj
  The object to solve for the associated cubic.

The documentation for this class was generated from the following file:

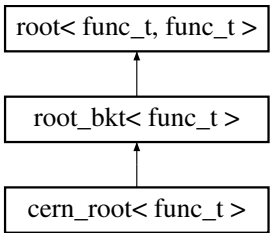
- poly.h

46.30 cern_root< func_t > Class Template Reference

One-dimensional root-finding routine (CERNLIB)

`#include <cern_root.h>`

Inheritance diagram for cern_root< func_t >:



46.30.1 Detailed Description

template<class func_t = funct>class cern_root< func_t >

This class attempts to find x_1 and x_2 in $[a,b]$ such that $f(x_1)f(x_2)\leq 0$, $|f(x_1)|\leq|f(x_2)|$, and $|x_1-x_2|\leq 2$ tol_abs $(1+|x_0|)$. The function solve_bkt() requires inputs `x1` and `x2` such that $f(x_1)f(x_2)\leq 0$.

The variable cern_root::tol_abs defaults to 10^{-8} and cern_root::ntrial defaults to 200.

The function solve_bkt() returns 0 for success, gsl_einval if the root is not initially bracketed, and gsl_emaxiter if the number of function evaluations is greater than cern_root::ntrial.

Based on the CERNLIB routines RZEROX and DZEROX, which was based on Bus75 and is documented at `http://wwwasdoc.-web.cern.ch/wwwasdoc/shortwrupsdir/c200/top.html`

Definition at line 58 of file cern_root.h.

Public Member Functions

- int set_mode (int m)
  Set mode of solution (1 or 2)
- virtual const char * type ()
  Return the type, "`cern_root`".
- virtual int solve_bkt (double &x1, double x2, func_t &func)
  Solve `func` in region $x_1<x<x_2$ returning x_1.

Protected Member Functions

- double sign (double a, double b)
  FORTRAN-like function for sign.

Protected Attributes

- int mode
  Internal storage for the mode.

46.30.2 Member Function Documentation

46.30.2.1 template<class func_t = funct> int cern_root< func_t >::set_mode (int *m*) [inline]

- `1` should be used for simple functions where the cost is inexpensive in comparison to one iteration of solve_bkt(), or functions which have a pole near the root (this is the default).
- `2` should be used for more time-consuming functions.

If an integer other than `1` or `2` is specified, the error handler is called.

Definition at line 100 of file cern_root.h.

46.30.2.2 template<class func_t = funct> virtual int cern_root< func_t >::solve_bkt (double & *x1*, double *x2*, func_t & *func*) [inline, virtual]

The parameters `x1` and `x2` should be set so that $f(x_1)f(x_2) \leq 0$ before calling solve_bkt(). If this is not the case, the error handler will be called and the solver will fail.

This function converges unless the number of iterations is larger than root::ntrial, in which case root::last_conv is set to gsl_emaxiter and the error handler is called if root::err_nonconv is true.

Implements root_bkt< func_t >.

Definition at line 125 of file cern_root.h.

46.30.3 Field Documentation

46.30.3.1 template<class func_t = funct> int cern_root< func_t >::mode [protected]

This internal variable is actually defined to be smaller by 1 than the "mode" as it is defined in the CERNLIB documentation in order to avoid needless subtraction in solve_bkt().

Definition at line 71 of file cern_root.h.

The documentation for this class was generated from the following file:

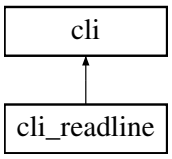
- cern_root.h

46.31 cli Class Reference

Configurable command-line interface.

`#include <cli.h>`

Inheritance diagram for cli:



46.31.1 Detailed Description

This class is experimental.

Default commands: help, get/set, quit, exit, '!', verbose, license, warranty, alias, run.

Note that if the shell command is allowed (as it is by default) there are some potential security issues which are not solved here.

Commands which begin with a '#' character are ignored.

Note

In interactive mode, commands are limited to 300 characters.

Todo Long options cannot be one letter long, or else process_args() will fail, thus the class should throw if a long option with only one letter is given.

Idea for Future Warn in run_interactive() when extra parameters are given

Idea for Future Include a "remove command" function

Idea for Future A replace command function, there's already some code in cli.cpp for this.

Idea for Future There's some code duplication between comm_option_run() and run_interactive()

Idea for Future Allow the user to set the tilde string

Idea for Future Disallow direct access to cli::par_list in order to ensure parameter names do not contain whitespace

Concepts

As a matter of definition, the command-line arguments are simply called arguments. These arguments may be options (in which case they begin with either one dash or two) or parameters to these options. When run in interactive mode, these options are also commands.

Definition at line 230 of file cli.h.

Data Structures

- class parameter
  Parameter for cli.
- class parameter_bool
  String parameter for cli.
- class parameter_double
  Double parameter for cli.
- class parameter_int
  Integer parameter for cli.
- class parameter_string
  String parameter for cli.

Public Member Functions

- int set_function (comm_option_funct &usf)
  Function to call when a `set` command is issued.
- virtual char * cli_gets (const char *c)
  The function which obtains input from the user.
- int call_args (std::vector< cmd_line_arg > &ca)
  Call functions corresponding to command-line args.
- int process_args (int argv, char *argc[ ], std::vector< cmd_line_arg > &ca, int debug=0)
  Process command-line arguments from a const char array.
- int process_args (std::string s, std::vector< cmd_line_arg > &ca, int debug=0)
  Process command-line arguments from a string.
- int set_verbose (int v)
  Set verbosity.
- int run_interactive ()
  Run the interactive mode.
- int set_alias (std::string alias, std::string str)
  Set an alias `alias` for the string `str`.
- std::string get_alias (std::string alias)
  Set an alias `alias` for the string `str`.

Basic operation

- int set_comm_option (comm_option_s &ic)
  Add a new command.
- template<class vec_t >
  int set_comm_option_vec (size_t list_size, vec_t &option_list)
  Add a vector containing new commands/options.
- int set_param_help (std::string param, std::string help)
  Set one-line help text for a parameter named `param`.
- int run_auto (int argv, char *argc[ ])
  Automatically parse arguments to main and call interactive mode if required.

Data Fields

- std::string tilde_string
  String to replace tildes with.
- bool gnu_intro
  If true, output the usual GNU intro when run_interactive() is called (default true).
- bool sync_verbose
  If true, then sync cli::verbose, with a parameter of the same name.
- bool shell_cmd_allowed
  If true, allow the user to use ! to execute a shell command (default true)
- std::string prompt
  The prompt (default "> ")
- std::string desc
  A one- or two-line description (default is empty string)
- std::string cmd_name
  The name of the command.
- std::string addl_help_cmd
  Additional help text for interactive mode (default is empty string)
- std::string addl_help_cli
  Additional help text for command-line (default is empty string)

The default command objects

- comm_option_s **c_commands**
- comm_option_s **c_help**
- comm_option_s **c_quit**
- comm_option_s **c_exit**
- comm_option_s **c_license**

- comm_option_s **c_warranty**
- comm_option_s **c_set**
- comm_option_s **c_get**
- comm_option_s **c_run**
- comm_option_s **c_no_intro**
- comm_option_s **c_alias**

Static Public Attributes

Value to indicate whether commands are also command-line options

- static const int **comm_option_command** = 0
- static const int **comm_option_cl_param** = 1
- static const int **comm_option_both** = 2

Protected Member Functions

- int output_param_list ()
  Output the parameter list.
- int expand_tilde (std::vector< std::string > &sv)
  Attempt to expand a tilde to a user's home directory.
- int apply_alias (std::vector< std::string > &sv, std::string sold, std::string snew)
  Replace all occurences of `sold` with `snew` in `sv`.
- int separate (std::string str, std::vector< std::string > &sv)
  Separate a string into words, handling quotes.
- bool string_equal_dash (std::string s1, std::string s2)
  Compare two strings, treating dashes and underscores as equivalent.

The hard-coded command functions

- int **comm_option_alias** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_commands** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_get** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_help** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_license** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_no_intro** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_run** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_set** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_warranty** (std::vector< std::string > &sv, bool itive_com)

Protected Attributes

- int verbose
  Control screen output.
- char buf [300]
  Storage for getline.
- comm_option_funct * user_set_func
  Storage for the function to call after setting a parameter.
- std::vector< comm_option_s > clist
  List of commands.

Help for parameters

- std::vector< std::string > **ph_name**
- std::vector< std::string > **ph_desc**

Parameter storage and associated iterator type

- typedef std::map< std::string, parameter *, string_comp > ::iterator par_t
  List iterator.
- std::map< std::string, parameter *, string_comp > par_list
  Parameter list.

Aliases

- typedef std::map< std::string, std::string, string_comp > ::iterator **al_it**
- std::map< std::string, std::string, string_comp > **als**

46.31.2 Member Function Documentation

46.31.2.1 int cli::expand_tilde (std::vector< std::string > & *sv*) [protected]

Experimental and currently unused.

46.31.2.2 int cli::separate (std::string *str,* std::vector< std::string > & *sv*) [protected]

This function separates a string into words, and handles words that begin with a "</tt> by adding more words until finding one which ends with another <tt>".

This is used to reformat command descriptions and help text for the screen width in comm_option_help(), to process lines read from a file in comm_option_run(), and to process input in run_interactive().

Note

This function does not understand nested quotes.

46.31.2.3 int cli::set_comm_option (comm_option_s & *ic*)

Each command/option must have either a short form in comm_option_s::shrt or a long from in comm_option_s::lng, which is unique from the other commands/options already present. You cannot add two commands/options with the same short form, even if they have different long forms, and vice versa.

46.31.2.4 virtual char* cli::cli_gets (const char * *c*) [virtual]

Idea for Future Think about whether or not this should be protected? (Possibly not, as it's extensively used by acolm.cpp)

Reimplemented in cli_readline.

46.31.2.5 int cli::process_args (int *argv,* char * *argc[],* std::vector< cmd_line_arg > & *ca,* int *debug* = 0)

This doesn't actually execute the functions for the corresponding options, but simply processes the parameters `argv` and `argc` and packs the information into `ca`.

This function assumes that `argc[0]` just contains the name of the command, and should thus be ignored.

46.31.2.6 int cli::process_args (std::string *s,* std::vector< cmd_line_arg > & *ca,* int *debug* = 0)

Todo There's a typecast in this function to (char *) from (const char *) which needs reworking.

46.31.2.7 int cli::set_verbose (int *v*)

Most errors are output to the screen even if verbose is zero.

46.31.2.8 int cli::set_alias (std::string *alias,* std::string *str*)

Aliases can also be set using the command `'alias'`, but that version allows only one-word aliases.

46.31.2.9 std::string cli::get_alias (std::string *alias*)

Aliases can also be set using the command `'alias'`, but that version allows only one-word aliases.

46.31.3 Field Documentation

46.31.3.1 bool cli::gnu_intro

In order to conform to GNU standards, this ought not be set to false by default.

Definition at line 440 of file cli.h.

The documentation for this class was generated from the following file:

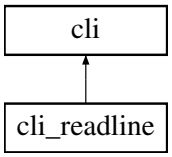
- cli.h

46.32 cli_readline Class Reference

An extension to cli which uses readline.

`#include <cli_readline.h>`

Inheritance diagram for cli_readline:



46.32.1 Detailed Description

This header-only class requires the GNU `readline` library for use, but is not referenced by O_2scl code at the moment to make the library usable without `readline`.

Definition at line 42 of file cli_readline.h.

Public Member Functions

- **cli_readline** (std::string fname=".cli_hist", size_t max_size=100)
- virtual char * cli_gets (const char *c)
  Function to get a string from the user.

Protected Attributes

- char * line_read
  Buffer for readline.
- std::string histfile
  String containing filename.
- size_t msize
  Maximum history file size.

46.32.2 Member Function Documentation

46.32.2.1 virtual char* cli_readline::cli_gets (const char * *c*) [inline, virtual]

Reimplemented from cli.

Definition at line 77 of file cli_readline.h.

The documentation for this class was generated from the following file:

- cli_readline.h

46.33 cmd_line_arg Struct Reference

A command-line argument for cli.

`#include <cli.h>`

46.33.1 Detailed Description

This is the internal structure that cli uses to package command-line arguments.

Definition at line 179 of file cli.h.

Data Fields

- std::string arg
  The argument.
- bool is_option
  Is an option?
- bool is_valid
  Is a properly formatted option.
- std::vector< std::string > parms
  List of parameters (empty, unless it's an option)
- comm_option_s * cop
  A pointer to the appropriate option (0, unless it's an option)

The documentation for this struct was generated from the following file:

- cli.h

46.34 table::col_s Struct Reference

Column structure for table [protected].

`#include <table.h>`

46.34.1 Detailed Description

This struct is used internally by table to organize the columns and need not be instantiated by the casual end-user.

Definition at line 1001 of file table.h.

Data Fields

- ovector_base * dat
  Pointer to column.
- bool owner
  Owner of column.
- int index
  Column index.

The documentation for this struct was generated from the following file:

- table.h

46.35 columnify Class Reference

Create nicely formatted columns from a table of strings.

`#include <columnify.h>`

46.35.1 Detailed Description

This is a brute-force approach of order ncols × nrows. The column widths and spacings of are computed by exhaustively examining all strings in every column.

Idea for Future Move the screenify() functionality from misc.h into this class?

Idea for Future It might be better to allow the string table to be specified with iterators.

Definition at line 51 of file columnify.h.

Public Member Functions

- template<class mat_string_t , class vec_string_t , class vec_int_t >
  int align (const mat_string_t &table, size_t ncols, size_t nrows, vec_string_t &ctable, vec_int_t &align_spec)
  Take `table` and create a new object `ctable` with appropriately formatted columns.

Static Public Attributes

- static const int align_left = 1
  Align the left-hand sides.
- static const int align_right = 2
  Align the right-hand sides.
- static const int align_lmid = 3
  Center, slightly to the left if spacing is uneven.
- static const int align_rmid = 4
  Center, slightly to the right if spacing is uneven.
- static const int align_dp = 5
  Align with decimal points.
- static const int align_lnum = 6
  Align negative numbers to the left and use a space for positive numbers.

46.35.2 Member Function Documentation

46.35.2.1 template<class mat_string_t , class vec_string_t , class vec_int_t > int columnify::align (const mat_string_t & *table*, size_t *ncols*, size_t *nrows*, vec_string_t & *ctable*, vec_int_t & *align_spec*) [inline]

The table of strings should be stored in `table` in "column-major" order (`table[ncols][nrows]`), so that `table` has the interpretation of a set of columns to be aligned. Before calling align(), `ctable` should be allocated so that at least the first `nrows` entries can be assigned, and `align_spec` should contain `ncols` entries specifying the style of alignment for each column.

The first argument can be any type which is accessible using two applications of `operator[]`, such as `string **`, `vector<string>[]`, or `vector<vector<string> >`

Definition at line 90 of file columnify.h.

The documentation for this class was generated from the following file:

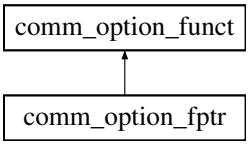
- columnify.h

46.36 comm_option_fptr Class Reference

Function pointer for cli command function.

`#include <cli.h>`

Inheritance diagram for comm_option_fptr:



46.36.1 Detailed Description

Definition at line 57 of file cli.h.

Public Member Functions

- comm_option_fptr (int(∗fp)(std::vector< std::string > &, bool))
  Create from a member function pointer from the specified class.
- virtual int operator() (std::vector< std::string > &cstr, bool itive_com)
  The basic function called by cli.

Protected Member Functions

- comm_option_fptr & operator= (const comm_option_fptr &f)
  Copy constructor.

Protected Attributes

- int(∗ fptr)(std::vector< std::string > &cstr, bool itive_com)
  The pointer to the member function.

The documentation for this class was generated from the following file:

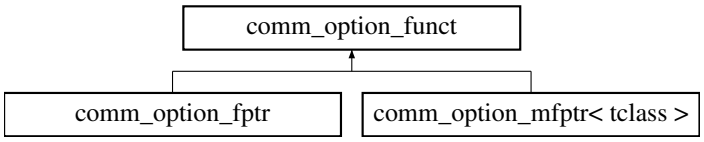
- cli.h

46.37 comm_option_funct Class Reference

Base for cli command function.

`#include <cli.h>`

Inheritance diagram for comm_option_funct:



46.37.1 Detailed Description

See the cli class for more details.

Definition at line 43 of file cli.h.

Public Member Functions

- virtual int operator() (std::vector< std::string > &cstr, bool itive_com)=0
  The basic function called by cli.

The documentation for this class was generated from the following file:

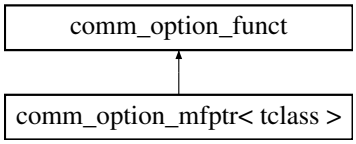
- cli.h

46.38 comm_option_mfptr< tclass > Class Template Reference

Member function pointer for cli command function.

`#include <cli.h>`

Inheritance diagram for comm_option_mfptr< tclass >:



46.38.1 Detailed Description

template<class tclass>class comm_option_mfptr< tclass >

Definition at line 96 of file cli.h.

Public Member Functions

- comm_option_mfptr (tclass *tp, int(tclass::*fp)(std::vector< std::string > &, bool))
  Create from a member function pointer from the specified class.
- virtual int operator() (std::vector< std::string > &cstr, bool itive_com)
  The basic function called by cli.

Protected Member Functions

- comm_option_mfptr (const comm_option_mfptr &f)

Copy constructor.

- comm_option_mfptr & operator= (const comm_option_mfptr &f)
  Copy constructor.

Protected Attributes

- int(tclass::* fptr)(std::vector< std::string > &cstr, bool itive_com)
  The pointer to the member function.
- tclass * tptr
  The pointer to the class.

The documentation for this class was generated from the following file:

- cli.h

46.39 comm_option_s Struct Reference

Command for interactive mode in cli.

```
#include <cli.h>
```

46.39.1 Detailed Description

See the cli class for more details.

Definition at line 151 of file cli.h.

Data Fields

- char shrt
  Short option ('\0' for none, must be unique if present)
- std::string lng
  Long option (must be specified and must be unique)
- std::string desc
  Description for help.
- int min_parms
  Minimum number of parameters (0 for none, -1 for variable)
- int max_parms
  Maximum number of parameters (0 for none, -1 for variable)
- std::string parm_desc
  Description of parameters.
- std::string help
  The help description.
- comm_option_funct * func
  The pointer to the function to be called (or 0 for no function)
- int type
  Type: command-line parameter, command, or both.

The documentation for this struct was generated from the following file:

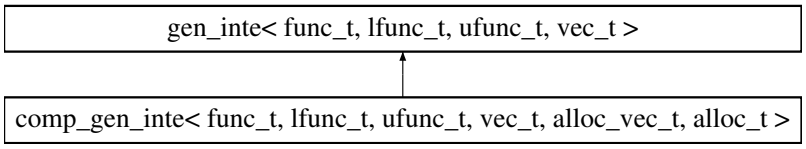
- cli.h

46.40 comp_gen_inte< func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Naive generalized multi-dimensional integration.

`#include <comp_gen_inte.h>`

Inheritance diagram for comp_gen_inte< func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t >:



46.40.1 Detailed Description

template<class func_t, class lfunc_t = func_t, class ufunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class comp_gen_inte< func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t >

Naively combine several one-dimensional integration objects from class inte in order to perform a multi-dimensional integration. The integration routines are specified in the function set_ptrs().

The integration routines are called in order of the index specified in the function set_oned_inte(). For `n`-dimensional integration, `n` one-dimensional integration objects should be specified, with indexes `0` through `n-1`. The integration routines are called in order of their index, so that the outermost integration is done by the routine specified with index 0. The integral performed is:

$$\int_{x_0=a_0}^{x_0=b_0} f(x_0) \int_{x_1=a_1(x_0)}^{x_1=b_1(x_0)} f(x_0,x_1) \ldots \int_{x_{n-1}=a_{n-1}(x_0,x_1,\ldots,x_{n-2})}^{x_{n-1}=b_{n-1}(x_0,x_1,\ldots,x_{n-2})} f(x_0,x_1,\ldots,x_{n-1})\, dx_{n-1} \ldots dx_1\, dx_0$$

This class is particularly useful if f_0 is time-consuming to evaluate, and separable from f_{n-1} .

See the discussion about the functions `func`, `lower` and `upper` in the documentation for the class gen_inte.

No error estimate is performed. Error estimation for multiple dimension integrals is provided by the Monte Carlo integration classes (see mcarlo_inte).

Idea for Future Provide an example of usage for this class.

Definition at line 73 of file comp_gen_inte.h.

Public Member Functions

- int set_oned_inte (inte< funct > &it, size_t i)
  Set the one-dimensional integration object with index `i`.
- virtual double ginteg (func_t &func, size_t n, func_t &lower, func_t &upper)
  Integrate function `func` from $\ell_i = f_i(x_i)$ to $u_i = g_i(x_i)$ for $0 < i < n-1$.
- virtual const char * type ()
  Return string denoting type ("comp_gen_inte")

Data Fields

- size_t max_dim
  The maxiumum number of integration dimensions (default 100)

Protected Member Functions

- double odfunc (double x, size_t &ix)
  The one-dimensional integration function.

Protected Attributes

- alloc_vec_t * cx
  The independent variable vector.
- lfunc_t * lowerp
  The function specifying the lower limits.
- ufunc_t * upperp
  The function specifying the upper limits.
- func_t * mf
  The function to be integrated.
- size_t ndim
  The number of dimensions.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- size_t nint
  The size of the integration object arrays.
- inte< funct > ** iptrs
  Pointers to the integration objects.
- bool * tptrs
  Flag indicating if integration object has been set.

The documentation for this class was generated from the following file:

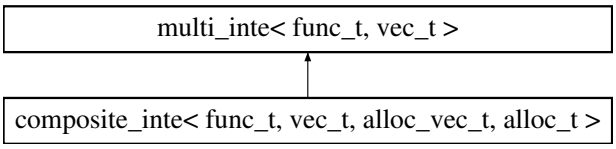
- comp_gen_inte.h

46.41 composite_inte< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Naive multi-dimensional integration over a hypercube.

`#include <composite_inte.h>`

Inheritance diagram for composite_inte< func_t, vec_t, alloc_vec_t, alloc_t >:



46.41.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class composite_inte< func_t, vec_t, alloc_vec_t, alloc_t >

Naively combine several one-dimensional integration objects from class inte in order to perform a multi-dimensional integration over a region defined by constant limits. For more general regions of integration, use children of the class gen_inte.

The 1-dimensional integration routines are specified in the function set_oned_inte().

The integration routines are called in order of the index specified in the function set_oned_inte(). For `n`-dimensional integration, `n` one-dimensional integration objects should be specified, with indexes `0` through `n-1`. The integration routines are called in order of their index, so that the outermost integration is done by the routine specified with index 0.

$$\int_{x_0=a_0}^{x_0=b_0} \int_{x_1=a_1}^{x_1=b_1} \ldots \int_{x_{n-1}=a_{n-1}}^{x_{n-1}=b_{n-1}} f(x_0, x_1, \ldots, x_n)$$

No error estimate is performed. Error estimation for multiple dimension integrals is provided by the Monte Carlo integration classes (see mcarlo_inte).

Idea for Future Create a function to set an entire array of one-dimensional integration objects at once

Definition at line 69 of file composite_inte.h.

Public Member Functions

- int set_oned_inte (inte< funct > &it, size_t i)
  Set the one-dimensional integration object with index `i`.
- virtual int minteg_err (func_t &func, size_t n, const vec_t &a, const vec_t &b, double &res, double &err)
  Integrate function `func` over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i <$ ndim-1.
- virtual const char * type ()
  Return string denoting type ("composite_inte")

Data Fields

- size_t max_dim
  The maxiumum number of integration dimensions (default 100)

Protected Member Functions

- double odfunc (double x, size_t &ix)
  The one-dimensional integration function.

Protected Attributes

- const vec_t * ax
  The user-specified upper limits.
- const vec_t * bx
  The user-specified lower limits.
- alloc_vec_t * cx
  The independent variable vector.
- func_t * mf
  The user-specified function.
- size_t ndim
  The user-specified number of dimensions.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- size_t nint
  The size of the integration object arrays.
- inte< funct > ** iptrs
  Pointers to the integration objects.
- bool * tptrs
  Flag indicating if integration object has been set.

46.41.2 Member Function Documentation

46.41.2.1 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int composite_inte< func_t, vec_t, alloc_vec_t, alloc_t >::minteg_err (func_t & *func,* size_t *n,* const vec_t & *a,* const vec_t & *b,* double & *res,* double & *err*) [inline, virtual]

The function set_oned_inte() must be used first to set the one-dimensional routines to use.

Implements multi_inte< func_t, vec_t >.

Definition at line 155 of file composite_inte.h.

46.41.2.2 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double composite_inte< func_t, vec_t, alloc_vec_t, alloc_t >::odfunc (double *x,* size_t & *ix*) [inline, protected]

This function to send to the integrators

Definition at line 209 of file composite_inte.h.

The documentation for this class was generated from the following file:

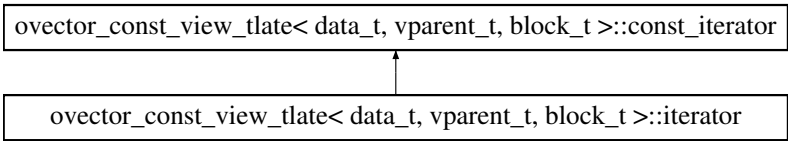
- composite_inte.h

46.42 ovector_const_view_tlate< data_t, vparent_t, block_t >::const_iterator Class Reference

A const iterator for ovectors.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_const_view_tlate< data_t, vparent_t, block_t >::const_iterator:



46.42.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_view_tlate< data_t, vparent_t, block_t >::const_iterator

Note

Experimental.

Todo Default constructor and iterator typedefs

Definition at line 123 of file ovector_tlate.h.

Public Member Functions

- const_iterator (const iterator &it)
  Copy-constructor for a const iterator.
- bool operator== (const const_iterator &it) const
  Equality.
- bool operator!= (const const_iterator &it) const
  Inequality.

- bool operator!= (const iterator &it) const
  Inequality.
- bool operator< (const const_iterator &it) const
  Less than.
- bool operator> (const const_iterator &it) const
  Greater than.
- const_iterator operator++ ()
  Prefix increment.
- const_iterator operator-- ()
  Prefix decrement.
- const_iterator operator++ (int)
  Postfix increment.
- const_iterator operator-- (int)
  Postfix decrement.
- const data_t operator* () const
  Dereference - return the corresponding vector element.
- const_iterator operator+= (size_t n)
  Move forward.
- const_iterator operator-= (size_t n)
  Move backwards.

Protected Member Functions

- const_iterator (data_t *p, size_t s)
  Internally create an iterator directly from a pointer.

Protected Attributes

- data_t * dp
  Pointer to current.
- size_t stride
  Stride.

Friends

- class ovector_const_view_tlate< data_t, vparent_t, block_t >
  Grant access to ovector classes for begin() and end() functions.
- class ovector_base_tlate< data_t, vparent_t, block_t >
  Grant access to ovector classes for begin() and end() functions.

The documentation for this class was generated from the following file:

- ovector_tlate.h

46.43 contour Class Reference

Calculate contour lines from a two-dimensional data set.

```
#include <contour.h>
```

46.43.1 Detailed Description

Basic Usage

- Specify the data as a two-dimensional square grid of "heights" with set_data().

- Specify the contour levels with set_levels().
- Compute the contours with calc_contours()

The contours are generated as a series of x- and y-coordinates, defining a line. If the contour is closed, then the first and the last set of coordinates will be equal.

The storage of the matrix to be specified in the function set_data() and this function is designed to follow the format:

$$\begin{array}{cccc} & x_0 & x_1 & x_2 \\ y_0 & M_{00} & M_{01} & M_{02} \\ y_1 & M_{10} & M_{11} & M_{12} \\ y_2 & M_{20} & M_{21} & M_{22} \end{array}$$

thus the matrix should be `M`[i][j] where `i` is the y index and `j` is the row index. (See also the discussion in the User's guide in the section called Rows and columns vs. x and y.)

The data is copied by set_data(), so changing the data will not change the contours unless set_data() is called again. The functions set_levels() and calc() can be called several times for the same data without calling set_data() again.

Note that in order to simplify the algorithm for computing contour lines, the calc_contours() function will adjust the user-specified contour levels slightly in order to ensure that no contour line passes exactly through any data point on the grid. The contours are adjusted by multiplying the original contour level by 1 plus a small number (10^{-8} by default), which is specified in lev_adjust.

Linear interpolation is used to decide whether or not a line segment and a contour cross. This choice is intentional, since (in addition to making the algorithm much simpler) it is the user (and not the class) which is likely best able to refine the data. In case a simple refinement scheme is desired, the method regrid_data() is provided which refines the data for any interpolation type.

Since linear interpolation is used, the contour calculation implicitly assumes that there is not more than one intersection of any contour level with any line segment. For contours which do not close inside the region of interest, the results will always end at either the minimum or maximum values of the x or y grid points (no extrapolation is ever done). Note also that the points defining the contour are not necessarily equally spaced, but two neighboring points will never be farther apart than the distance across opposite corners of one cell in the grid.

The Algorithm:

This works by viewing the data as defining a square two-dimensional grid. The function calc_contours() exhaustively enumerates every line segment in the grid which involves a level crossing and then organizes the points defined by the intersection of a line segment with a level curve into a full contour.

Idea for Future Rewrite the code which adjusts the contour levels to ensure contours don't go through the data to adjust the internal copy of the data instead.

Idea for Future It would be nice to have a function which creates a set of closed regions to fill which represent the data. However, this likely requires a completely new algorithm, because it's not easy to simply close the contours already generated by the calc_contours() function. There are, for example, several cases which are difficult to handle, such as filling a region in between several closed contours.

Definition at line 172 of file contour.h.

Public Member Functions

Basic usage

- template<class vec_t , class mat_t >
  int set_data (size_t sizex, size_t sizey, const vec_t &x_fun, const vec_t &y_fun, const mat_t &udata)
  Set the data.
- template<class vec_t >
  int set_levels (size_t nlevels, vec_t &ulevels)
  Set the contour levels.
- int calc_contours (std::vector< contour_line > &clines, bool debug=false)
  Calculate the contours.

Regrid function

- int regrid_data (size_t xfact, size_t yfact, base_interp_mgr< ovector_const_view > &bim1, base_interp_mgr< ovector_const_subvector > &bim2)
  Regrid the data.

Obtain internal data

- int get_data (size_t &sizex, size_t &sizey, ovector *&x_fun, ovector *&y_fun, omatrix *&udata)
  Get the data.
- int get_edges (std::vector< edge_crossings > &rt_edges, std::vector< edge_crossings > &bm_edges)
  Return the edges.
- int print_edges (edge_crossings &right, edge_crossings &bottom)
  Print out the edges to cout.

Data Fields

- int verbose
  Verbosity parameter.
- double lev_adjust
  (default 10^{-8}*)*

Protected Member Functions

- int find_next_point_right (int j, int k, int &jnext, int &knext, int &dir_next, int nsw, edge_crossings &right, edge_crossings &bottom)
  Find next point starting from a point on a right edge.
- int find_next_point_bottom (int j, int k, int &jnext, int &knext, int &dir_next, int nsw, edge_crossings &right, edge_crossings &bottom)
  Find next point starting from a point on a bottom edge.
- int find_intersections (size_t ilev, double &level, edge_crossings &right, edge_crossings &bottom)
  Find all of the intersections of the edges with the contour level.
- int right_edges (double level, o2scl::sm_interp *si, edge_crossings &right)
  Interpolate all right edge crossings.
- int bottom_edges (double level, o2scl::sm_interp *si, edge_crossings &bottom)
  Interpolate all bottom edge crossings.
- int process_line (int j, int k, int dir, ovector &x, ovector &y, bool first, edge_crossings &right, edge_crossings &bottom)
  Create a contour line from a starting edge.
- int check_data ()
  Check to ensure the x- and y-arrays are monotonic.

Protected Attributes

- std::vector< edge_crossings > red
  Right edge list.
- std::vector< edge_crossings > bed
  Bottom edge list.

User-specified data

- int **nx**
- int **ny**
- ovector **xfun**
- ovector **yfun**
- omatrix **data**

User-specified contour levels

- int **nlev**
- ovector **levels**
- bool **levels_set**

Static Protected Attributes

Edge direction

- static const int **dright** = 0
- static const int **dbottom** = 1

Edge status

- static const int **empty** = 0
- static const int **edge** = 1
- static const int **contourp** = 2
- static const int **endpoint** = 3

Edge found or not found

- static const int **efound** = 1
- static const int **enot_found** = 0

46.43.2 Member Function Documentation

46.43.2.1 template<class vec_t , class mat_t > int contour::set_data (size_t *sizex*, size_t *sizey*, const vec_t & *x_fun*, const vec_t & *y_fun*, const mat_t & *udata*) [inline]

The types `vec_t` and `mat_t` can be any types which have `operator`[] and `operator`[][] for array and matrix indexing.

Note that this method copies all of the user-specified data to local storage so that changes in the data after calling this function will not be reflected in the contours that are generated.

Definition at line 194 of file contour.h.

46.43.2.2 template<class vec_t > int contour::set_levels (size_t *nlevels*, vec_t & *ulevels*) [inline]

This is separate from the function calc_contours() so that the user can compute the contours for different data sets using the same levels.

Definition at line 224 of file contour.h.

46.43.2.3 int contour::calc_contours (std::vector< contour_line > & *clines*, bool *debug* = `false`)

The function calc_contours() returns the total number of contours found. Since there may be more than one disconnected contours for the same contour level, or no contours for a given level, the total number of contours may be less than or greater than the number of levels given by set_levels().

If an error occurs, zero is returned.

46.43.2.4 int contour::regrid_data (size_t *xfact*, size_t *yfact*, base_interp_mgr< ovector_const_view > & *bim1*, base_interp_mgr< ovector_const_subvector > & *bim2*)

Use interpolation to refine the data set. This can be called before calc_contours() in order to make the contour levels smoother by providing a smaller grid size. If the original number of data points is $(\mathrm{nx},\mathrm{ny})$, then the new number of data points is

$$(\mathrm{xfact}\,(\mathrm{nx}-1)+1,\mathrm{yfact}\,(\mathrm{ny}-1)+1)$$

The parameters `xfact` and `yfact` must both be larger than zero and they cannot both be 1.

46.43.2.5 int contour::get_data (size_t & *sizex*, size_t & *sizey*, ovector *& *x_fun*, ovector *& *y_fun*, omatrix *& *udata*) [inline]

This is useful to see how the data has changed after a call to regrid_data().

Idea for Future There is probably a better way than returning pointers to the internal data.

Definition at line 280 of file contour.h.

The documentation for this class was generated from the following file:

- contour.h

46.44 contour_line Class Reference

A contour line.

`#include <contour.h>`

46.44.1 Detailed Description

The contour lines generated by the contour class are given as objects of this type.

Idea for Future Write HDF I/O for contour lines?

Idea for Future Make this a subclass of contour .

Definition at line 43 of file contour.h.

Public Member Functions

- contour_line ()
  Create an empty line.
- contour_line (const contour_line &c)
  Copy constructor for STL allocator.

Data Fields

- double level
  The contour level.
- ovector x
  The line x coordinates.
- ovector y
  The line y coordinates.

The documentation for this class was generated from the following file:

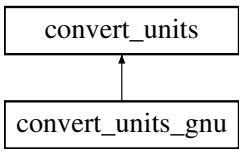
- contour.h

46.45 convert_units Class Reference

Convert units.

`#include <convert_units.h>`

Inheritance diagram for convert_units:



46.45.1 Detailed Description

Allow the user to convert between two different units after specifying a conversion factor. This class will also automatically combine two conversion factors to create a new unit conversion (but it cannot combine more than two).

Conversions are performed by the convert() function and the conversion factors must be specified beforehand using the insert_cache() function.

If the GNU units command is not in the local path, the user may modify units_cmd_string to specify the full pathname. One can also modify units_cmd_string to specify a different `units.dat` file.

Example:

```
convert_units cu;
cu.insert_cache("in","cm",2.54);
cout << "12 in is " << cu.convert("in","cm",12.0) << " cm. " << endl;
```

Note

Combining two conversions allows for some surprising apparent contradictions from numerical precision errors. If there are two matching unit conversion pairs which give the same requested conversion factor, then one can arrange a situation where the same conversion factor is reported with slightly different values after adding a related conversion to the table. One way to fix this is to force the class not to combine two conversions by setting combine_two_conv to false. Alternatively, one can ensure that no combination is necessary by manually adding the desired combination conversion to the cache after it is first computed.

Idea for Future Ideally, a real C++ API for the GNU units command would be better.

Definition at line 78 of file convert_units.h.

Data Structures

- struct unit_t
  The type for caching unit conversions.

Public Member Functions

- virtual double convert (std::string from, std::string to, double val)
  Return the value `val` after converting using units `from` and `to`.
- int insert_cache (std::string from, std::string to, double conv)
  Manually insert a unit conversion into the cache.
- int remove_cache (std::string from, std::string to)
  Manually remove a unit conversion into the cache.
- int print_cache ()
  Print the present unit cache to std::cout.
- int energy_conv ()
  Add conversion factors for energy equivalents.

Data Fields

- int verbose
  Verbosity (default 0)
- bool use_gnu_units
  (default true)
- bool err_on_fail
  (default true)
- bool combine_two_conv
  If true, allow combinations of two conversions (default true)
- std::string units_cmd_string
  Default 'units'.

Protected Types

- typedef std::map< std::string, unit_t, string_comp > ::iterator miter
  The iterator type.

Protected Attributes

- std::map< std::string, unit_t, string_comp > mcache
  The cache where unit conversions are stored.

The documentation for this class was generated from the following file:

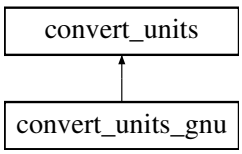
- convert_units.h

46.46 convert_units_gnu Class Reference

Convert units using a system call to GNU units.

`#include <convert_units_gnu.h>`

Inheritance diagram for convert_units_gnu:



46.46.1 Detailed Description

Experimental.

Definition at line 43 of file convert_units_gnu.h.

Public Member Functions

- virtual double convert (std::string from, std::string to, double val)
  Return the value `val` after converting using units `from` and `to`.
- int make_units_dat (std::string fname, bool c_1=false, bool hbar_1=false, bool K_1=false)
  Make a GNU `units.dat` file from the GSL constants.

Data Fields

Strings to form \c units command

- std::string **prefix**
- std::string **midfix**
- std::string **suffix**

46.46.2 Member Function Documentation

46.46.2.1 int convert_units_gnu::make_units_dat (std::string *fname*, bool *c_1* = false, bool *hbar_1* = false, bool *K_1* = false) [inline]

If c_1 is true, then the second is defined in terms of meters so that the speed of light is unitless. If hbar_1 is true, then the kilogram is defined in terms of s/m^2 so that $\hbar$ is unitless.

Note

Not all of the GSL constants or the canonical GNU units conversions are given here.

Definition at line 204 of file convert_units_gnu.h.

The documentation for this class was generated from the following file:

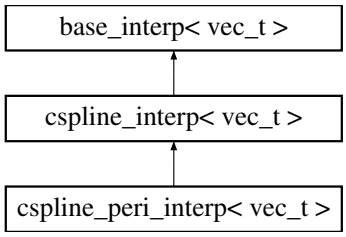
- convert_units_gnu.h

46.47 cspline_interp< vec_t > Class Template Reference

Cubic spline interpolation (GSL)

`#include <interp.h>`

Inheritance diagram for cspline_interp< vec_t >:



46.47.1 Detailed Description

template<class vec_t>class cspline_interp< vec_t >

By default, this class uses natural boundary conditions, where the second derivative vanishes at each end point. Extrapolation effectively assumes that the second derivative is linear outside of the endpoints.

Definition at line 300 of file interp.h.

Public Member Functions

- cspline_interp ()
  Create a base interpolation object with natural or periodic boundary conditions.
- virtual int allocate (size_t size)
  Allocate memory, assuming x and y have size size.
- virtual int init (const vec_t &xa, const vec_t &ya, size_t size)
  Initialize interpolation routine.
- virtual int free ()
  Free allocated memory.
- virtual int interp (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &y)
  Give the value of the function $y(x = x_0)$.

- virtual int deriv (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &dydx)
  Give the value of the derivative $y'(x = x_0)$.
- virtual int deriv2 (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &d2ydx2)
  Give the value of the second derivative $y''(x = x_0)$.
- virtual int integ (const vec_t &x_array, const vec_t &y_array, size_t size, double a, double b, double &result)
  Give the value of the integral $\int_a^b y(x)\,dx$.
- virtual const char * type ()
  Return the type, "cspline_interp".

Protected Member Functions

- void coeff_calc (const double c_array[ ], double dy, double dx, size_t index, double *b, double *c2, double *d)
  Compute coefficients for cubic spline interpolation.

Protected Attributes

- o2scl_linalg::uvector_4_mem p4m
  Memory for the tridiagonalization.

Storage for cubic spline interpolation

- double * **c**
- double * **g**
- double * **diag**
- double * **offdiag**

Private Member Functions

- **cspline_interp** (const cspline_interp< vec_t > &)
- cspline_interp< vec_t > & **operator=** (const cspline_interp< vec_t > &)

46.47.2 Member Function Documentation

46.47.2.1 template<class vec_t> virtual int cspline_interp< vec_t >::init (const vec_t & *xa*, const vec_t & *ya*, size_t *size*) [inline, virtual]

Natural boundary conditions

Reimplemented from base_interp< vec_t >.

Reimplemented in cspline_peri_interp< vec_t >.

Definition at line 383 of file interp.h.

The documentation for this class was generated from the following file:

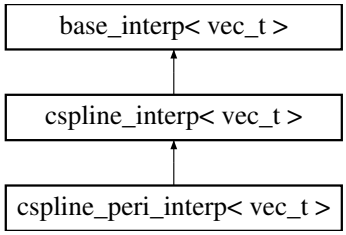
- interp.h

46.48 cspline_peri_interp< vec_t > Class Template Reference

Cubic spline interpolation with periodic boundary conditions (GSL)

```
#include <interp.h>
```

Inheritance diagram for cspline_peri_interp< vec_t >:



46.48.1 Detailed Description

template<class vec_t>class cspline_peri_interp< vec_t >

Definition at line 593 of file interp.h.

Public Member Functions

- virtual int allocate (size_t size)
  Allocate memory, assuming x and y have size `size`.
- virtual const char * type ()
  Return the type, `"cspline_peri_interp"`.
- virtual int init (const vec_t &xa, const vec_t &ya, size_t size)
  Initialize interpolation routine.
- virtual int free ()
  Free allocated memory.

Protected Attributes

- o2scl_linalg::uvector_5_mem p5m
  Desc.

Private Member Functions

- **cspline_peri_interp** (const cspline_peri_interp< vec_t > &)
- cspline_peri_interp< vec_t > & **operator=** (const cspline_peri_interp< vec_t > &)

46.48.2 Member Function Documentation

46.48.2.1 template<class vec_t> virtual int cspline_peri_interp< vec_t >::init (const vec_t & *xa*, const vec_t & *ya*, size_t *size*) [inline, virtual]

Periodic boundary conditions

Natural boundary conditions

Reimplemented from cspline_interp< vec_t >.

Definition at line 652 of file interp.h.

The documentation for this class was generated from the following file:

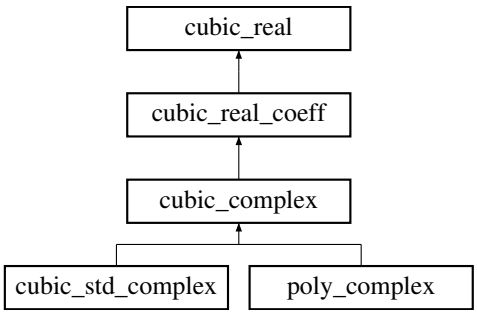
- interp.h

46.49 cubic_complex Class Reference

Solve a cubic polynomial with complex coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for cubic_complex:



46.49.1 Detailed Description

Definition at line 182 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a3, const double b3, const double c3, const double d3, double &x1, double &x2, double &x3)

  Solves the polynomial $a_3x^3+b_3x^2+c_3x+d_3=0$ giving the three solutions $x=x_1$, $x=x_2$, and $x=x_3$.
- virtual int solve_rc (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)

  Solves the polynomial $a_3x^3+b_3x^2+c_3x+d_3=0$ giving the real solution $x=x_1$ and two complex solutions $x=x_2$ and $x=x_3$.
- virtual int solve_c (const std::complex< double > a3, const std::complex< double > b3, const std::complex< double > c3, const std::complex< double > d3, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3)=0

  Solves the complex polynomial $a_3x^3+b_3x^2+c_3x+d_3=0$ giving the three complex solutions $x=x_1$, $x=x_2$, and $x=x_3$.
- const char * type ()

  Return a string denoting the type ("cubic_complex")

The documentation for this class was generated from the following file:

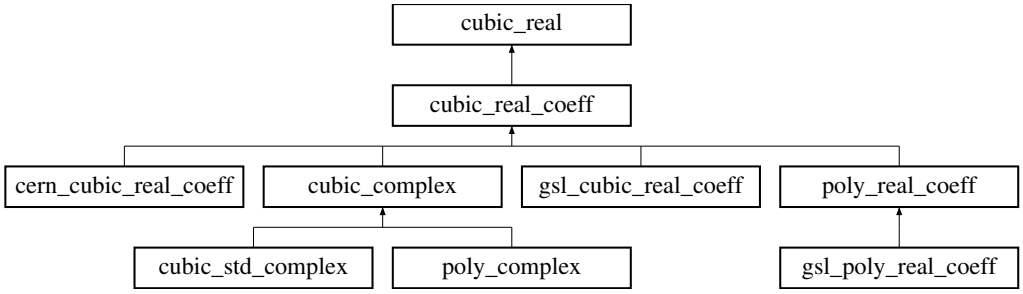
- poly.h

46.50 cubic_real Class Reference

Solve a cubic polynomial with real coefficients and real roots [abstract base].

`#include <poly.h>`

Inheritance diagram for cubic_real:



46.50.1 Detailed Description

Definition at line 133 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a3, const double b3, const double c3, const double d3, double &x1, double &x2, double &x3)=0
  Solves the polynomial $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$ giving the three solutions $x = x_1$, $x = x_2$, and $x = x_3$.
- const char * type ()
  Return a string denoting the type ("cubic_real")

The documentation for this class was generated from the following file:

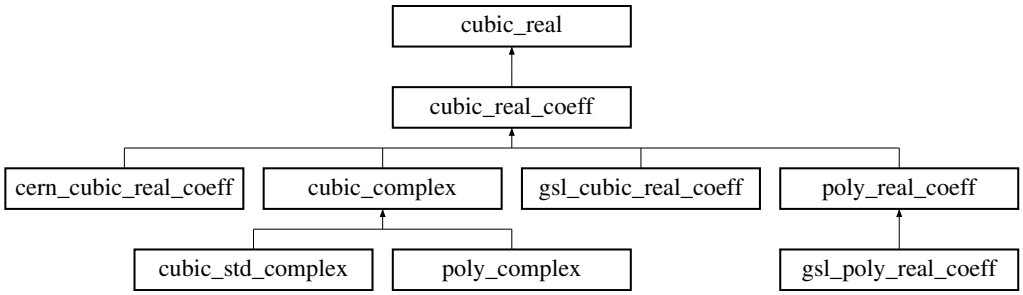
- poly.h

46.51 cubic_real_coeff Class Reference

Solve a cubic polynomial with real coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for cubic_real_coeff:



46.51.1 Detailed Description

Definition at line 153 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a3, const double b3, const double c3, const double d3, double &x1, double &x2, double &x3)

Solves the polynomial $a_3x^3+b_3x^2+c_3x+d_3=0$ giving the three solutions $x=x_1$, $x=x_2$, and $x=x_3$.

- virtual int solve_rc (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)=0
  Solves the polynomial $a_3x^3+b_3x^2+c_3x+d_3=0$ giving the real solution $x=x_1$ and two complex solutions $x=x_2$ and $x=x_3$.
- const char * type ()
  Return a string denoting the type ("cubic_real_coeff")

The documentation for this class was generated from the following file:

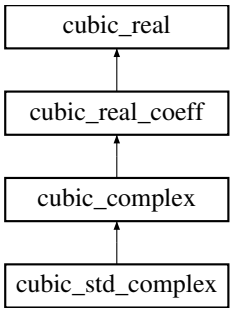
- poly.h

46.52 cubic_std_complex Class Reference

Solve a cubic with complex coefficients and complex roots.

`#include <poly.h>`

Inheritance diagram for cubic_std_complex:



46.52.1 Detailed Description

Definition at line 693 of file poly.h.

Public Member Functions

- virtual int solve_c (const std::complex< double > a3, const std::complex< double > b3, const std::complex< double > c3, const std::complex< double > d3, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3)
  Solves the complex polynomial $a_3x^3+b_3x^2+c_3x+d_3=0$ giving the three complex solutions $x=x_1$, $x=x_2$, and $x=x_3$.
- const char * type ()
  Return a string denoting the type ("cubic_std_complex")

The documentation for this class was generated from the following file:

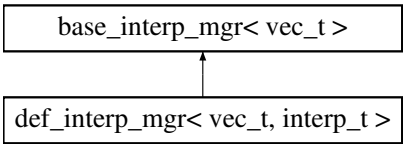
- poly.h

46.53 def_interp_mgr< vec_t, interp_t > Class Template Reference

A base interpolation object manager template.

`#include <interp.h>`

Inheritance diagram for def_interp_mgr< vec_t, interp_t >:



46.53.1 Detailed Description

template<class vec_t, template< class > class interp_t>class def_interp_mgr< vec_t, interp_t >

Definition at line 1100 of file interp.h.

Public Member Functions

- virtual base_interp< vec_t > * new_interp ()
  Create a new interpolation object.

The documentation for this class was generated from the following file:

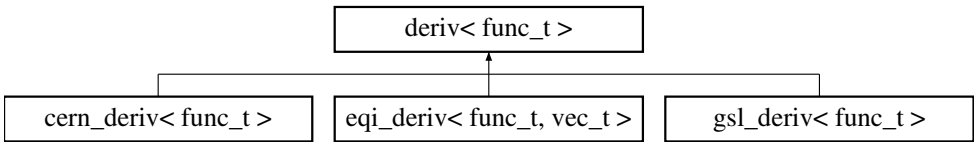
- interp.h

46.54 deriv< func_t > Class Template Reference

Numerical differentiation base [abstract base].

`#include <deriv.h>`

Inheritance diagram for deriv< func_t >:



46.54.1 Detailed Description

template<class func_t = funct>class deriv< func_t >

This base class does not perform any actual differentiation. Use one of the children cern_deriv, gsl_deriv, or eqi_deriv instead.

This base class contains some code to automatically apply the first derivative routines to compute second or third derivatives. The error estimates for these will likely be underestimated.

Note

Because this class template aims to automatically provide second and third derivatives, one must overload either both calc() and calc_int() or both calc_err() and calc_err_int().

Idea for Future Improve the methods for second and third derivatives

Definition at line 51 of file deriv.h.

Data Structures

- struct dpars
  A structure for passing the function to second and third derivatives [protected].

Public Member Functions

- virtual double calc (double x, func_t &func)
  Calculate the first derivative of `func` w.r.t. x.
- virtual double calc2 (double x, func_t &func)
  Calculate the second derivative of `func` w.r.t. x.
- virtual double calc3 (double x, func_t &func)
  Calculate the third derivative of `func` w.r.t. x.
- virtual double get_err ()
  Get uncertainty of last calculation.
- virtual int calc_err (double x, func_t &func, double &dfdx, double &err)=0
  Calculate the first derivative of `func` w.r.t. x and the uncertainty.
- virtual int calc2_err (double x, func_t &func, double &d2fdx2, double &err)
  Calculate the second derivative of `func` w.r.t. x and the uncertainty.
- virtual int calc3_err (double x, func_t &func, double &d3fdx3, double &err)
  Calculate the third derivative of `func` w.r.t. x and the uncertainty.
- virtual const char * type ()
  Return string denoting type ("deriv")

Data Fields

- bool err_nonconv
  If true, call the error handler if the routine does not "converge".
- int verbose
  Output control.

Protected Member Functions

- virtual double calc_int (double x, funct &func)
  Calculate the first derivative of `func` w.r.t. x.
- virtual int calc_err_int (double x, funct &func, double &dfdx, double &err)=0
  Calculate the first derivative of `func` w.r.t. x and the uncertainty.
- double derivfun (double x, dpars &dp)
  The function for the second derivative.
- double derivfun2 (double x, dpars &dp)
  The function for the third derivative.

Protected Attributes

- bool from_calc
  Avoids infinite loops in case the user calls the base class version.
- double derr
  The uncertainity in the most recent derivative computation.

46.54.2 Member Function Documentation

46.54.2.1 template<class func_t = funct> virtual double deriv< func_t >::calc (double *x*, func_t & *func*) [inline, virtual]

After calling calc(), the error may be obtained from get_err().

Definition at line 92 of file deriv.h.

46.54.2.2 template<class func_t = funct> virtual double deriv< func_t >::calc_int (double *x*, funct & *func*) [inline, protected, virtual]

This is an internal version of calc() which is used in computing second and third derivatives

Definition at line 180 of file deriv.h.

46.54.2.3 template<class func_t = funct> virtual int deriv< func_t >::calc_err_int (double *x*, funct & *func*, double & *dfdx*, double & *err*) [protected, pure virtual]

This is an internal version of calc_err() which is used in computing second and third derivatives

Implemented in cern_deriv< func_t >, gsl_deriv< func_t >, and gsl_deriv< funct >.

The documentation for this class was generated from the following file:

- deriv.h

46.55 deriv< func_t >::dpars Struct Reference

A structure for passing the function to second and third derivatives [protected].

`#include <deriv.h>`

46.55.1 Detailed Description

template<class func_t = funct>struct deriv< func_t >::dpars

Definition at line 60 of file deriv.h.

Data Fields

- func_t * func
  The pointer to the function.

The documentation for this struct was generated from the following file:

- deriv.h

46.56 edge_crossings Class Reference

Edges for the contour class.

`#include <contour.h>`

46.56.1 Detailed Description

The edge crossings generated by the contour class are given as objects of this type.

The status matrix contains one of four possible values

- 0 - empty (no edge)
- 1 - edge which has not yet been assigned to a contour
- 2 - edge assigned to contour point

- 3 - edge which has been designated as a contour endpoint

The matrices returned by contour are not square, their size depends on whether or not they contain the "bottom edges" or the "right edges".

Idea for Future Write HDF I/O object for edge crossings

Idea for Future Make this a subclass of contour .

Definition at line 82 of file contour.h.

Data Fields

- omatrix_int status
  Edge status.
- omatrix values
  Edge values.

The documentation for this class was generated from the following file:

- contour.h

46.57 exact_jacobian< func_t, vec_t, mat_t >::ej_parms Struct Reference

Parameter structure for passing information.

`#include <jacobian.h>`

46.57.1 Detailed Description

template<class func_t = mm_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base>struct exact_jacobian< func_t, vec_t, mat_t >::ej_parms

This class is primarily useful for specifying derivatives for using the jacobian::set_deriv() function.

Definition at line 395 of file jacobian.h.

Data Fields

- size_t nv
  The number of variables.
- size_t xj
  The current x value.
- size_t yi
  The current y value.
- vec_t * x
  The x vector.
- vec_t * y
  The y vector.

The documentation for this struct was generated from the following file:

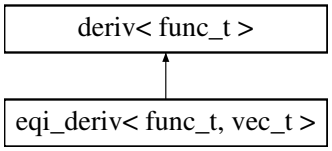
- jacobian.h

46.58 eqi_deriv< func_t, vec_t > Class Template Reference

Derivatives for equally-spaced abscissas.

`#include <eqi_deriv.h>`

Inheritance diagram for eqi_deriv< func_t, vec_t >:



46.58.1 Detailed Description

template<class func_t = funct, class vec_t = ovector_base>class eqi_deriv< func_t, vec_t >

This is an implementation of the formulas for equally-spaced abscissas as indicated below. The level of approximation is specified in set_npoints(). The value of $p \times h$ can be specified in `xoff` (default is zero).

Note

Uncertainties are not computed and the code for second and third derivatives is unfinished.
The derivatives given, for example, from the five-point formula can sometimes be more accurate than computing the derivative from the interpolation class. This is especially true near the boundaries of the interpolated region.

Idea for Future Finish the second and third derivative formulas.

Two-point formula (note that this is independent of p).

$$f'(x_0+ph) = \frac{1}{h}[f_1 - f_0]$$

Three-point formula from Abramowitz and Stegun

$$f'(x_0+ph) = \frac{1}{h}\left[\frac{2p-1}{2}f_{-1} - 2pf_0 + \frac{2p+1}{2}f_1\right]$$

Four-point formula from Abramowitz and Stegun

$$f'(x_0+ph) = \frac{1}{h}\left[-\frac{3p^2-6p+2}{6}f_{-1} + \frac{3p^2-4p-1}{2}f_0 - \frac{3p^2-2p-2}{2}f_1 + \frac{3p^2-1}{6}f_2\right]$$

Five-point formula from Abramowitz and Stegun

$$\begin{aligned} f'(x_0+ph) \quad = \quad & \frac{1}{h}\left[\frac{2p^3-3p^2-p+1}{12}f_{-2} - \frac{4p^3-3p^2-8p+4}{6}f_{-1}\right. \\ & + \frac{2p^3-5p}{2}f_0 - \frac{4p^3+3p^2-8p-4}{6}f_1 \\ & \left. + \frac{2p^3+3p^2-p-1}{12}f_2\right] \end{aligned}$$

The relations above can be confined to give formulas for second derivative formulas: Three-point formula

$$f'(x_0+ph) = \frac{1}{h^2}[f_{-1} - 2f_0 + f_1]$$

Four-point formula:

$$f'(x_0+ph) = \frac{1}{2h^2}\left[(1-2p)f_{-1} - (1-6p)f_0 - (1+6p)f_1 + (1+2p)f_2\right]$$

Five-point formula:

$$f'(x_0+ph) = \frac{1}{4h^2}\left[(1-2p)^2 f_{-2} + \left(8p-16p^2\right)f_{-1} - \left(2-24p^2\right)f_0 - \left(8p+16p^2\right)f_1 + (1+2p)^2 f_2\right]$$

Six-point formula:

$$\begin{aligned} f'(x_0+ph) &= \frac{1}{12h^2}\left[\left(2-10p+15p^2-6p^3\right)f_{-2} + \left(3+14p-57p^2+30p^3\right)f_{-1}\right. \\ &\quad + \left(-8+20p+78p^2-60p^3\right)f_0 + \left(-2-44p-42p^2+60p^3\right)f_1 \\ &\quad \left. + \left(6+22p+3p^2-30p^3\right)f_2 + \left(-1-2p+3p^2+6p^3\right)f_3\right] \end{aligned}$$

Seven-point formula:

$$\begin{aligned} f'(x_0+ph) &= \frac{1}{36h^2}\left[\left(4-24p+48p^2-36p^3+9p^4\right)f_{-3} + \left(12+12p-162p^2+180p^3-54p^4\right)f_{-2}\right. \\ &\quad + \left(-15+120p+162p^2-360p^3+135p^4\right)f_{-1} - 4\left(8+48p-3p^2-90p^3+45p^4\right)f_0 \\ &\quad +3\left(14+32p-36p^2-60p^3+45p^4\right)f_1 + \left(-12-12p+54p^2+36p^3-54p^4\right)f_2 \\ &\quad \left. + \left(1-6p^2+9p^4\right)f_3\right] \end{aligned}$$

Definition at line 137 of file eqi_deriv.h.

Public Member Functions

- int set_npoints (int npoints)
  Set the number of points to use for first derivatives (default 5)
- int set_npoints2 (int npoints)
  Set the number of points to use for second derivatives (default 5)
- virtual int calc_err (double x, func_t &func, double &dfdx, double &err)
  Calculate the first derivative of `func` w.r.t. x.
- virtual int calc2_err (double x, func_t &func, double &dfdx, double &err)
  Calculate the second derivative of `func` w.r.t. x.
- virtual int calc3_err (double x, func_t &func, double &dfdx, double &err)
  Calculate the third derivative of `func` w.r.t. x.
- double calc_vector (double x, double x0, double dx, size_t nx, const vec_t &y)
  Calculate the derivative at `x` given an array.
- double calc2_vector (double x, double x0, double dx, size_t nx, const vec_t &y)
  Calculate the second derivative at `x` given an array.
- double calc3_vector (double x, double x0, double dx, size_t nx, const vec_t &y)
  Calculate the third derivative at `x` given an array.
- int deriv_vector (size_t nv, double dx, const vec_t &y, vec_t &dydx)
  Calculate the derivative of an entire array.
- virtual const char * type ()
  Return string denoting type ("eqi_deriv")

Data Fields

- double h
  Stepsize (Default 10^{-4}).
- double xoff
  Offset (default 0.0)

46.58.2 Member Function Documentation

46.58.2.1 template<class func_t = funct, class vec_t = ovector_base> int eqi_deriv< func_t, vec_t >::set_npoints (int *npoints*) [inline]

Acceptable values are 2-5 (see above).

Definition at line 159 of file eqi_deriv.h.

46.58.2.2 template<class func_t = funct, class vec_t = ovector_base> int eqi_deriv< func_t, vec_t >::set_npoints2 (int *npoints*) [inline]

Acceptable values are 3-5 (see above).

Definition at line 185 of file eqi_deriv.h.

46.58.2.3 template<class func_t = funct, class vec_t = ovector_base> double eqi_deriv< func_t, vec_t >::calc_vector (double *x*, double *x0*, double *dx*, size_t *nx*, const vec_t & *y*) [inline]

This calculates the derivative at `x` given a function specified in an array `y` of size `nx` with equally spaced abscissas. The first abscissa should be given as `x0` and the distance between adjacent abscissas should be given as `dx`. The value `x` need not be one of the abscissas (i.e. it can lie in between an interval). The appropriate offset is calculated automatically.

Definition at line 240 of file eqi_deriv.h.

46.58.2.4 template<class func_t = funct, class vec_t = ovector_base> double eqi_deriv< func_t, vec_t >::calc2_vector (double *x*, double *x0*, double *dx*, size_t *nx*, const vec_t & *y*) [inline]

This calculates the second derivative at `x` given a function specified in an array `y` of size `nx` with equally spaced abscissas. The first abscissa should be given as `x0` and the distance between adjacent abscissas should be given as `dx`. The value `x` need not be one of the abscissas (i.e. it can lie in between an interval). The appropriate offset is calculated automatically.

Definition at line 256 of file eqi_deriv.h.

46.58.2.5 template<class func_t = funct, class vec_t = ovector_base> double eqi_deriv< func_t, vec_t >::calc3_vector (double *x*, double *x0*, double *dx*, size_t *nx*, const vec_t & *y*) [inline]

This calculates the third derivative at `x` given a function specified in an array `y` of size `nx` with equally spaced abscissas. The first abscissa should be given as `x0` and the distance between adjacent abscissas should be given as `dx`. The value `x` need not be one of the abscissas (i.e. it can lie in between an interval). The appropriate offset is calculated automatically.

Definition at line 273 of file eqi_deriv.h.

46.58.2.6 template<class func_t = funct, class vec_t = ovector_base> int eqi_deriv< func_t, vec_t >::deriv_vector (size_t *nv*, double *dx*, const vec_t & *y*, vec_t & *dydx*) [inline]

Right now this uses np=5.

Todo generalize to other values of npoints.

Definition at line 286 of file eqi_deriv.h.

The documentation for this class was generated from the following file:

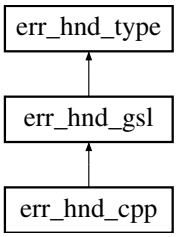
- eqi_deriv.h

46.59 err_hnd_cpp Class Reference

Error handler to throw C++ exceptions.

```
#include <exception.h>
```

Inheritance diagram for err_hnd_cpp:



46.59.1 Detailed Description

The default error handler, def_err_hnd, is of this type.

Definition at line 257 of file exception.h.

Public Member Functions

- virtual void set (const char *reason, const char *file, int line, int lerrno)
  Set an error.
- virtual const char * type () const
  Return type ("err_hnd_cpp")

The documentation for this class was generated from the following file:

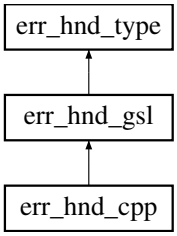
- exception.h

46.60 err_hnd_gsl Class Reference

The error handler.

`#include <err_hnd.h>`

Inheritance diagram for err_hnd_gsl:



46.60.1 Detailed Description

An error handler for use in O_2scl which replaces the GSL error handler

Note that the string arguments to set() can refer to temporary storage, since they are copied when the function is called and an error is set.

Definition at line 199 of file err_hnd.h.

Public Member Functions

- virtual void set (const char *reason, const char *file, int line, int lerrno)
  Set an error.
- virtual void get (const char *&reason, const char *&file, int &line, int &lerrno)
  Get the last error.
- virtual int get_errno () const
  Return the last error number.
- virtual int get_line () const
  Return the line number of the last error.
- virtual const char * get_reason () const
  Return the reason for the last error.
- virtual const char * get_file () const
  Return the file name of the last error.
- virtual const char * get_str ()
  Return a string summarizing the last error.
- virtual void reset ()
  Remove last error information.
- void set_mode (int m)
  Set error handling mode.
- int get_mode () const
  Return the error handling mode.
- virtual const char * type () const
  Return type ("err_hnd_gsl")

Data Fields

- bool array_abort
  If true, call exit() when an array index error is set (default true)
- size_t fname_size
  Number of characters from filename to print (default 28)

Protected Member Functions

- std::string errno_to_string (int errnox)
  Convert an error number to a string.

Protected Attributes

- int a_errno
  The error number.
- int a_line
  The line number.
- int mode
  The mode of error handling (default 2)
- char * a_file
  The filename.
- char a_reason [rsize]
  The error explanation.
- char fullstr [fsize]
  A full string with explanation and line and file info.

Static Protected Attributes

- static const int rsize = 300
  The maximum size of error explanations.
- static const int fsize = 400
  The maximum size of error explanations with the line and file info.

46.60.2 Member Function Documentation

46.60.2.1 void err_hnd_gsl::set_mode (int *m*) [inline]

- 0 - Continue execution after an error occurs
- 1 - Continue execution after an error occurs
- 2 - Abort execution after an error occurs, and print out the error information (default)

Definition at line 240 of file err_hnd.h.

The documentation for this class was generated from the following file:

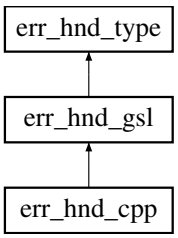
- err_hnd.h

46.61 err_hnd_type Class Reference

Class defining an error handler [abstract base].

`#include <err_hnd.h>`

Inheritance diagram for err_hnd_type:



46.61.1 Detailed Description

A global object of this type is defined, err_hnd .

Idea for Future There may be an issue associated with the string manipulations causing errors in the error handler.

Definition at line 142 of file err_hnd.h.

Public Member Functions

- virtual void set (const char *reason, const char *file, int line, int lerrno)=0
  Set an error.
- virtual void get (const char *&reason, const char *&file, int &line, int &lerrno)=0
  Get the last error.
- virtual int get_errno () const =0
  Return the last error number.
- virtual int get_line () const =0
  Return the line number of the last error.
- virtual const char * get_reason () const =0
  Return the reason for the last error.
- virtual const char * get_file () const =0
  Return the file name of the last error.
- virtual const char * get_str ()=0

Return a string summarizing the last error.
- virtual void reset ()=0
  Remove last error information.
- virtual const char * type () const =0
  Return type.

Static Public Member Functions

- static void gsl_hnd (const char *reason, const char *file, int line, int lerrno)
  Set an error.

46.61.2 Member Function Documentation

46.61.2.1 static void err_hnd_type::gsl_hnd (const char * *reason*, const char * *file*, int *line*, int *lerrno*) [inline, static]

This is separate from set(), since the gsl error handler needs to be a static function.

Definition at line 155 of file err_hnd.h.

The documentation for this class was generated from the following file:

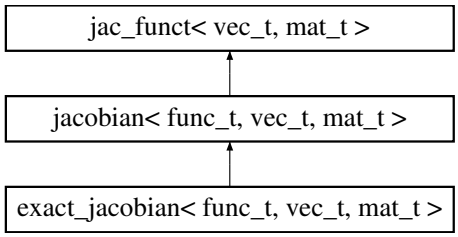
- err_hnd.h

46.62 exact_jacobian< func_t, vec_t, mat_t > Class Template Reference

A direct calculation of the jacobian using a deriv object.

`#include <jacobian.h>`

Inheritance diagram for exact_jacobian< func_t, vec_t, mat_t >:



46.62.1 Detailed Description

template<class func_t = mm_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base>class exact_jacobian< func_t, vec_t, mat_t >

Note that it is most often wasteful to use this Jacobian in a root-finding routine and using more approximate Jacobians is more efficient. This class is mostly useful for demonstration purposes.

Definition at line 374 of file jacobian.h.

Data Structures

- struct ej_parms
  Parameter structure for passing information.

Public Member Functions

- int set_deriv (deriv< funct > &de)
  Set the derivative object.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &jac)
  The operator()

Data Fields

- gsl_deriv< funct > def_deriv
  The default derivative object.

Protected Member Functions

- double dfn (double x, ej_parms &ejp)
  Function for the derivative object.

Protected Attributes

- deriv< funct > * dptr
  Pointer to the derivative object.

The documentation for this class was generated from the following file:

- jacobian.h

46.63 exc_exception Class Reference

Generic exception.

`#include <exception.h>`

46.63.1 Detailed Description

This class derives from `std::exception`.

The errors which are handled with this exception type are

- gsl_failure (-1) Failure
- gsl_efailed (5) Generic failure
- gsl_esanity (7) Sanity check failed
- gsl_eunsup (23) Requested feature is not supported by the hardware
- gsl_eunimpl (24) Requested feature not (yet) implemented

Definition at line 55 of file exception.h.

Public Member Functions

- exc_exception ()
  Create an exception.
- virtual const char * what () const throw ()
  Return the error string.

The documentation for this class was generated from the following file:

- exception.h

46.64 exc_invalid_argument Class Reference

Invalid argument exception.

`#include <exception.h>`

46.64.1 Detailed Description

This class derives from `std::invalid_argument`.

The errors which are handled with this exception type are

- gsl_einval (4) invalid argument supplied by user
- gsl_ebadtol (13) user specified an invalid tolerance
- gsl_ebadlen (19) matrix, vector lengths are not conformant
- gsl_enotsqr (20) matrix not square
- gsl_eindex (36) Invalid index for array or matrix

Definition at line 110 of file exception.h.

Public Member Functions

- exc_invalid_argument (const std::string &s)
  Create an exception with description provided in `s`.
- virtual const char * what () const throw ()
  Return the error string.

The documentation for this class was generated from the following file:

- exception.h

46.65 exc_ios_failure Class Reference

I/O failure error exception.

`#include <exception.h>`

46.65.1 Detailed Description

This class derives from `std::ios::failure`.

The errors which are handled with this exception type are

- gsl_eof=32 end of file
- gsl_efilenotfound=35 File not found

Definition at line 235 of file exception.h.

Public Member Functions

- exc_ios_failure (const std::string &s)
  Create an exception with description provided in s.
- virtual const char * what () const throw ()
  Return the error string.

The documentation for this class was generated from the following file:

- exception.h

46.66 exc_logic_error Class Reference

Logic error exception.

`#include <exception.h>`

46.66.1 Detailed Description

This class derives from `std::logic_error`.

The error which is handled with this exception type is

- gsl_ememtype (34) Incorrect type for memory object

Definition at line 81 of file exception.h.

Public Member Functions

- exc_logic_error (const std::string &s)
  Create an exception with description provided in s.
- virtual const char * what () const throw ()
  Return the error string.

The documentation for this class was generated from the following file:

- exception.h

46.67 exc_overflow_error Class Reference

Overflow error runtime exception.

`#include <exception.h>`

46.67.1 Detailed Description

This class derives from `std::overflow_error`.

The errors which are handled with this exception type are

- gsl_ezerodiv (12) tried to divide by zero
- gsl_eovrflw (16) overflow

Definition at line 209 of file exception.h.

Public Member Functions

- exc_overflow_error (const std::string &s)
  Create an exception with description provided in `s`.
- virtual const char * what () const throw ()
  Return the error string.

The documentation for this class was generated from the following file:

- exception.h

46.68 exc_range_error Class Reference

Range error runtime exception.

`#include <exception.h>`

46.68.1 Detailed Description

This class derives from `std::range_error`.

The errors which are handled with this exception type are

- gsl_edom (1) input domain error, e.g sqrt(-1)
- gsl_erange (2) output range error, e.g. exp(1e100)
- gsl_eundrflw (15) underflow

Definition at line 183 of file exception.h.

Public Member Functions

- exc_range_error (const std::string &s)
  Create an exception with description provided in `s`.
- virtual const char * what () const throw ()
  Return the error string.

The documentation for this class was generated from the following file:

- exception.h

46.69 exc_runtime_error Class Reference

Generic runtime error exception.

`#include <exception.h>`

46.69.1 Detailed Description

This class derives from `std::runtime_error`.

The errors which are handled with this exception type are

- gsl_efault (3) invalid pointer
- gsl_efactor (6) factorization failed
- gsl_enomem (8) malloc failed
- gsl_ebadfunc (9) problem with user-supplied function
- gsl_erunaway (10) iterative process is out of control
- gsl_emaxiter (11) exceeded max number of iterations
- gsl_etol (14) failed to reach the specified tolerance
- gsl_eloss (17) loss of accuracy
- gsl_eround (18) failed because of roundoff error
- gsl_esing (21) apparent singularity detected
- gsl_ediverge (22) integral or series is divergent
- gsl_ecache (25) cache limit exceeded
- gsl_etable (26) table limit exceeded
- gsl_enoprog (27) iteration is not making progress toward solution
- gsl_enoprogj (28) jacobian evaluations are not improving the solution
- gsl_etolf (29) cannot reach the specified tolerance in f
- gsl_etolx (30) cannot reach the specified tolerance in x
- gsl_etolg (31) cannot reach the specified tolerance in gradient
- gsl_enotfound (33) Generic "not found" result
- gsl_outsidecons (37) Outside constraint region

Definition at line 156 of file exception.h.

Public Member Functions

- **exc_runtime_error** (const std::string &s)
- virtual ~exc_runtime_error () throw ()
  Create an exception with description provided in s.
- virtual const char * what () const throw ()
  Return the error string.

The documentation for this class was generated from the following file:

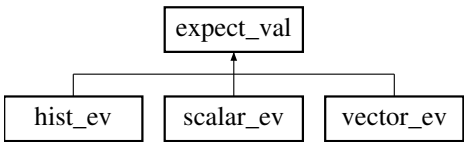
- exception.h

46.70 expect_val Class Reference

Expectation value base class.

`#include <expect_val.h>`

Inheritance diagram for expect_val:



46.70.1 Detailed Description

This class is experimental.

This is a base class for a set of classes useful for recording the outputs of several iterations of a numerical simulation, and summarizing the average, standard deviation, and error in the average over all iterations. After each iteration, some measurement is performed, and that measurement is added to the class with an `add()` functions (e.g. scalar_ev::add()). Autocorrelations are common in numerical simulations, and to handle this difficulty the measurements may be demarcated into 'blocks'. These blocks may have a variable or fixed number of measurements in each block.

The constructor needs as input the number of blocks to record (at least one) and the number of measurements per block. The number of measurements per block can be zero to indicate an unspecified number of blocks to record.

If the number of measurements per block is greater than zero, then blocks are filled one by one, moving to the next block when the requested of measurements (`n_per_block`) in the previous block has been provided. If the number of measurements per block is zero, then the blocks are filled evenly.

Current averages are always reported, though not all data is always used if there are incomplete blocks or the blocks have not yet been filled evenly. Two functions `current_avg()` and `current_avg_stats()` are provided in children. The latter provides the number of blocks and number of measurements per block used in the currently reported statistics. If only one block or one measurement is available, the standard deviation is reported as zero. If no data is available, then calls to `current_avg()` will call the error handler. See, e.g. scalar_ev::current_avg() and scalar_ev::current_avg_stats().

If `n_per_block` is nonzero, then children call the error handler if they get more than `n_per_block` times `n_blocks` measurements.

Idea for Future If `n_per_block` is nonzero and the user adds more than `n_blocks` times `n_per_block` data, set `n_per_block` to zero, allocate the `last` structure and fall back to the variable block size method.

Definition at line 81 of file expect_val.h.

Public Member Functions

- expect_val ()
  Create with one block of unspecified size.
- expect_val (const expect_val &ev)
  Copy constructor.
- expect_val & operator= (const expect_val &ev)
  Copy constructor.
- expect_val (size_t n_blocks, size_t n_per_block)
  Create with `n_blocks` blocks and `n_per_block` points per block.
- virtual void set_blocks (size_t n_blocks, size_t n_per_block)
  Reset for `n_blocks` blocks and `n_per_block` points per block.

- virtual void get_blocks (size_t &n_blocks, size_t &n_per_block) const
  Get the number of blocks and the number of points per block.
- virtual void free ()
  Free allocated data.
- virtual void reset ()
  Clear all the data.
- virtual void get_block_indices (size_t &i_block, size_t &i_curr_block) const
  Get the block index and the index within the current block.
- virtual bool finished () const
  Returns true if all blocks have been stored.
- virtual double progress () const
  Report progress as a fraction between zero to one (inclusive)

Data Fields

- std::string name
  The name of the expectation value.
- std::string short_name
  The shortened name.

Protected Attributes

- size_t iblock
  Index denoting the current block number.
- size_t i
  Index for the number of values in the current block (or blocks)
- size_t nblocks
  Total number of blocks (default 1)
- size_t nperblock
  Number of measurements per block (default 0)

46.70.2 Constructor & Destructor Documentation

46.70.2.1 expect_val::expect_val (size_t *n_blocks*, size_t *n_per_block*)

If this is called with a value of zero for `n_blocks`, then the error handler is called.

46.70.3 Member Function Documentation

46.70.3.1 virtual void expect_val::set_blocks (size_t *n_blocks*, size_t *n_per_block*) [virtual]

This function resets the currently stored data to zero by calling reset(). If this is called with a value of zero for `n_blocks`, then the value 1 is assumed.

Reimplemented in scalar_ev.

46.70.3.2 virtual bool expect_val::finished ( ) const [virtual]

If `n_per_block` is greater than zero, then this reports true only when all `n_blocks` times `n_per_block` data points have been added.

Otherwise, this reports true whenever the last addition of data filled up the last block.

46.70.3.3 virtual double expect_val::progress ( ) const [virtual]

When n_per_block is zero, this reports the progress in having filled a round of blocks (i.e. whenever the last addition of data filled up the last block). This reports zero if no data has been added.

When n_per_block is nonzero, this reports the total progress on all blocks, reporting 1.0 only when all n_blocks times n_per_block data points have been added.

46.70.4 Field Documentation

46.70.4.1 size_t expect_val::nblocks [protected]

This should never be zero.

Definition at line 95 of file expect_val.h.

46.70.4.2 size_t expect_val::nperblock [protected]

This is zero to indicate blocks of unspecified size

Definition at line 101 of file expect_val.h.

The documentation for this class was generated from the following file:

- expect_val.h

46.71 gsl_inte_singular< func_t >::extrapolation_table Struct Reference

A structure for extrapolation for gsl_inte_qags.

`#include <gsl_inte_singular.h>`

46.71.1 Detailed Description

template<class func_t>struct gsl_inte_singular< func_t >::extrapolation_table

Idea for Future Move this to a new class, with qelg() as a method

Definition at line 59 of file gsl_inte_singular.h.

Data Fields

- size_t n
  Index of new element in the first column.
- double rlist2 [52]
  Lower diagonals of the triangular epsilon table.
- size_t nres
  Number of calls.
- double res3la [3]
  Three most recent results.

The documentation for this struct was generated from the following file:

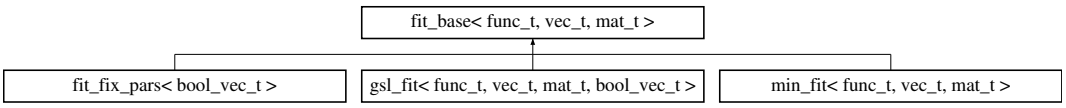
- gsl_inte_singular.h

46.72 fit_base< func_t, vec_t, mat_t > Class Template Reference

Non-linear least-squares fitting [abstract base].

`#include <fit_base.h>`

Inheritance diagram for fit_base< func_t, vec_t, mat_t >:



46.72.1 Detailed Description

template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base>class fit_base< func_t, vec_t, mat_t >

Definition at line 185 of file fit_base.h.

Public Member Functions

- virtual int print_iter (size_t nv, vec_t &x, double y, int iter, double value=0.0, double limit=0.0)
  Print out iteration information.
- virtual int fit (size_t ndat, vec_t &xdat, vec_t &ydat, vec_t &yerr, size_t npar, vec_t &par, mat_t &covar, double &chi2, func_t &fitfun)=0
  Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.
- virtual const char * type ()
  Return string denoting type ("fit_base")

Data Fields

- int verbose
  An integer describing the verbosity of the output.
- size_t n_dat
  The number of data points.
- size_t n_par
  The number of parameters.

46.72.2 Member Function Documentation

46.72.2.1 template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base> virtual int fit_base< func_t, vec_t, mat_t >::print_iter (size_t *nv*, vec_t & *x*, double *y*, int *iter*, double *value* = `0.0`, double *limit* = `0.0`) `[inline, virtual]`

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 204 of file fit_base.h.

46.72.2.2 template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base> virtual int fit_base< func_t, vec_t, mat_t >::fit (size_t *ndat*, vec_t & *xdat*, vec_t & *ydat*, vec_t & *yerr*, size_t *npar*, vec_t & *par*, mat_t & *covar*, double & *chi2*, func_t & *fitfun*) `[pure virtual]`

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Implemented in gsl_fit< func_t, vec_t, mat_t, bool_vec_t >, and min_fit< func_t, vec_t, mat_t >.

The documentation for this class was generated from the following file:

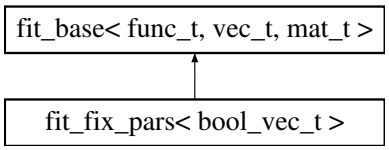
- fit_base.h

46.73 fit_fix_pars< bool_vec_t > Class Template Reference

Multidimensional fitting fixing some parameters and varying others.

`#include <fit_fix.h>`

Inheritance diagram for fit_fix_pars< bool_vec_t >:



46.73.1 Detailed Description

template<class bool_vec_t>class fit_fix_pars< bool_vec_t >

Definition at line 37 of file fit_fix.h.

Public Member Functions

- fit_fix_pars ()
  Specify the member function pointer.
- virtual int fit (size_t ndat, ovector_base &xdat, ovector_base &ydat, ovector_base &yerr, size_t npar, ovector_base &par, omatrix_base &covar, double &chi2, fit_funct<> &fitfun)
  Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.
- virtual int fit_fix (size_t ndat, ovector_base &xdat, ovector_base &ydat, ovector_base &yerr, size_t npar, ovector_base &par, bool_vec_t &fix, omatrix_base &covar, double &chi2, fit_funct<> &fitfun)
  Fit function `func` while fixing some parameters as specified in `fix`.
- int set_fit (fit_base< fit_funct_mfptr< fit_fix_pars< bool_vec_t > > > &fitter)
  Change the base minimizer.

Data Fields

- gsl_fit< fit_funct_mfptr < fit_fix_pars< bool_vec_t > > > def_fit
  The default base minimizer.

Protected Member Functions

- virtual double fit_func (size_t nv, const ovector_base &x, double xx)
  The new function to send to the minimizer.

Protected Attributes

- fit_base< fit_funct_mfptr < fit_fix_pars< bool_vec_t > > > * fitp
  The minimizer.
- fit_funct * funcp
  The user-specified function.

- size_t unv
  The user-specified number of variables.
- size_t nv_new
  The new number of variables.
- bool_vec_t * fixp
  Specify which parameters to fix.
- ovector_base * xp
  The user-specified initial vector.

Private Member Functions

- **fit_fix_pars** (const fit_fix_pars &)
- fit_fix_pars & **operator=** (const fit_fix_pars &)

46.73.2 Member Function Documentation

46.73.2.1 template<class bool_vec_t > virtual int fit_fix_pars< bool_vec_t >::fit (size_t *ndat*, ovector_base & *xdat*, ovector_base & *ydat*, ovector_base & *yerr*, size_t *npar*, ovector_base & *par*, omatrix_base & *covar*, double & *chi2*, fit_funct<> & *fitfun*) [inline, virtual]

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Definition at line 61 of file fit_fix.h.

The documentation for this class was generated from the following file:

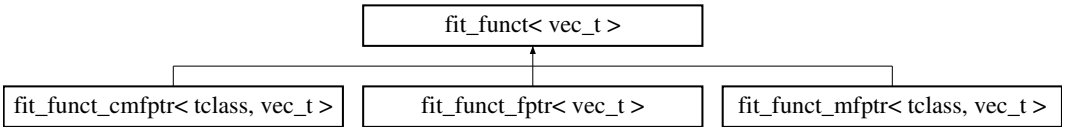
- fit_fix.h

46.74 fit_funct< vec_t > Class Template Reference

Fitting function [abstract base].

`#include <fit_base.h>`

Inheritance diagram for fit_funct< vec_t >:



46.74.1 Detailed Description

template<class vec_t = ovector_base>class fit_funct< vec_t >

Definition at line 35 of file fit_base.h.

Public Member Functions

- virtual double operator() (size_t np, const vec_t &p, double x)=0
  Using parameters in `p`, predict `y` given `x`.

Private Member Functions

- **fit_funct** (const fit_funct &)
- fit_funct & **operator=** (const fit_funct &)

The documentation for this class was generated from the following file:

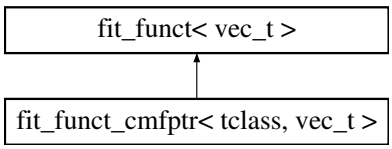
- fit_base.h

46.75 fit_funct_cmfptr< tclass, vec_t > Class Template Reference

Const member function pointer fitting function.

`#include <fit_base.h>`

Inheritance diagram for fit_funct_cmfptr< tclass, vec_t >:



46.75.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class fit_funct_cmfptr< tclass, vec_t >

Definition at line 139 of file fit_base.h.

Public Member Functions

- fit_funct_cmfptr (tclass *tp, double(tclass::*fp)(size_t np, const vec_t &p, double x) const)
  Specify the member function pointer.
- virtual double operator() (size_t np, const vec_t &p, double x)
  Using parameters in p, predict y given x.

Protected Attributes

- double(tclass::* fptr)(size_t np, const vec_t &p, double x) const
  Storage for the user-specified function pointer.
- tclass * tptr
  Storage for the class pointer.

Private Member Functions

- **fit_funct_cmfptr** (const fit_funct_cmfptr &)
- fit_funct_cmfptr & **operator=** (const fit_funct_cmfptr &)

The documentation for this class was generated from the following file:

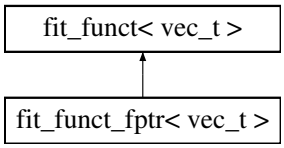
- fit_base.h

46.76 fit_funct_fptr< vec_t > Class Template Reference

Function pointer fitting function.

`#include <fit_base.h>`

Inheritance diagram for fit_funct_fptr< vec_t >:



46.76.1 Detailed Description

template<class vec_t = ovector_base>class fit_funct_fptr< vec_t >

Definition at line 58 of file fit_base.h.

Public Member Functions

- fit_funct_fptr (double(*fp)(size_t np, const vec_t &p, double x))
  Specify a fitting function by a function pointer.
- virtual double operator() (size_t np, const vec_t &p, double x)
  Using parameters in p, predict y given x.

Protected Member Functions

- **fit_funct_fptr** (const fit_funct_fptr &)
- fit_funct_fptr & **operator=** (const fit_funct_fptr &)

Protected Attributes

- double(* fptr)(size_t np, const vec_t &p, double x)
  Storage for the user-specified function pointer.

The documentation for this class was generated from the following file:

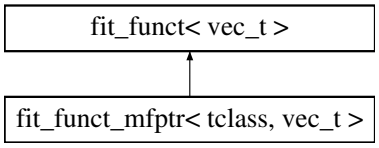
- fit_base.h

46.77 fit_funct_mfptr< tclass, vec_t > Class Template Reference

Member function pointer fitting function.

`#include <fit_base.h>`

Inheritance diagram for fit_funct_mfptr< tclass, vec_t >:



46.77.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class fit_funct_mfptr< tclass, vec_t >

Definition at line 96 of file fit_base.h.

Public Member Functions

- fit_funct_mfptr (tclass *tp, double(tclass::*fp)(size_t np, const vec_t &p, double x))
  Specify the member function pointer.
- virtual double operator() (size_t np, const vec_t &p, double x)
  Using parameters in p, predict y given x.

Protected Attributes

- double(tclass::* fptr)(size_t np, const vec_t &p, double x)
  Storage for the user-specified function pointer.
- tclass * tptr
  Storage for the class pointer.

Private Member Functions

- **fit_funct_mfptr** (const fit_funct_mfptr &)
- fit_funct_mfptr & **operator=** (const fit_funct_mfptr &)

The documentation for this class was generated from the following file:

- fit_base.h

46.78 format_float Class Reference

Format a floating point number into a Latex or HTML string.

`#include <format_float.h>`

46.78.1 Detailed Description

This class formats floating point strings into something useful for HTML or Latex documents. For basic use, simply call either html_mode() or latex_mode() and then use convert().

The base-10 logarithm of the smallest and largest numbers to be represented without a string akin to "times 10 to the nth power" can be specified in set_exp_limits(). The number of significant figures can be specified with set_sig_figs() (the default is 5).

To force convert() to adds zeros to the right side of the mantissa to guarantee that the requested number of significant digits is given, call set_pad_zeros() with a `true` argument.

To force scientific notation for all numbers, set the maximum exponent to be smaller than the minimum exponent.

Note

This function does not warn the user if the number of significant figures requested is larger than the machine precision.
If the absolute magnitude for either the minimum or maximum exponent is larger than or equal to the number of significant figures, then rounding will automatically occur.

The format for a normal number is

```
prefix (sign-string) number suffix
```

and in scientific notation is

```
sci-prefix (sci-sign-string) number times-string
exp-prefix (exp-sign-string) exponent exp-suffix sci-suffix
```

Examples

The code

```
format_float fd;
fd.latex_mode();
cout << fd.convert(-sqrt(2.0)*1.0e-5) << endl;
cout << fd.convert(sqrt(2.0)*1.0e-2) << endl;
cout << fd.convert(-sqrt(2.0)*1.0e-1) << endl;
```

outputs

```
$-$1.4142 $\\times 10^{-5}$
0.014142
$-$0.14142
```

and the code

```
format_float fd;
fd.html_mode();
fd.set_sig_figs(7);
fd.set_pad_zeros(true);
cout << fd.convert(1.414e-5) << endl;
cout << fd.convert(1.414e-2) << endl;
```

outputs

```
1.414000 &times; 10<sup>-5</sup>
0.01414000
```

Idea for Future Handle inf's and nan's correctly.

Idea for Future Allow change of string for the "+" sign for the exponent

Definition at line 110 of file format_float.h.

Public Member Functions

Basic usage

- int html_mode ()
  Set HTML mode.
- int latex_mode ()
  Set Latex mode.
- int c_mode ()
  C-like mode.
- std::string convert (double x, bool debug=false)
  Convert a floating point number to a string.

Set text settings

These are modified by the functions html_mode() and latex_mode()

- int set_prefix (std::string prefix)
  set prefix
- int set_suffix (std::string suffix)
  Set suffix.
- int set_sci_prefix (std::string sci_prefix)
  Set prefix for scientific notation.
- int set_sci_suffix (std::string sci_suffix)
  Set suffix for scientific notation.
- int set_exp_prefix (std::string exp_prefix)
  Set prefix for exponent.
- int set_exp_suffix (std::string exp_suffix)
  Set suffix for exponent.
- int set_sign (std::string sign)
  Set sign.
- int set_exp_sign (std::string exp_sign)
  Set sign for exponent.
- int set_show_exp_sign (bool b)
  Set policy for showing positive exponent sign.
- int set_sci_sign (std::string sci_sign)
  Set sign for scientific notation.
- int set_times (std::string times)
  Set times.
- int set_zero (std::string zero)
  Set zero.
- int set_not_finite (std::string not_finite)
  Set string for numbers which are not finite.

Set other settings

These are not modified by the functions html_mode() and latex_mode()

- int set_exp_limits (int min, int max)
  Set the exponent limits.
- int set_sig_figs (size_t sig_figs)
  Set the number of significant figures (argument has maximum of 15 and cannot be zero)
- int set_pad_zeros (bool pad)
  Set pad zeros.
- int set_dec_point (std::string dec_point)
  Set decimal point.
- int set_exp_digits (size_t d)
  Set minimum number of digits in the exponent.

Get text settings

These are modified by the functions html_mode() and latex_mode()

- std::string get_prefix ()
  Get prefix.
- std::string get_suffix ()
  Get suffix.
- std::string get_sci_prefix ()
  Get prefix for scientific notation.
- std::string get_sci_suffix ()
  Get suffix for scientific notation.
- std::string get_exp_prefix ()
  Get prefix for exponent.
- std::string get_exp_suffix ()
  Get suffix for exponent.
- std::string get_sign ()
  Get sign.

- std::string get_exp_sign ()
  Get sign for exponent.
- std::string get_sci_sign ()
  Get sign for scientific notation.
- std::string get_times ()
  Get times.
- std::string get_zero ()
  Get zero.
- std::string get_not_finite ()
  Get string for numbers which are not finite.

Get other settings

These are not modified by the functions html_mode() and latex_mode()

- int get_exp_min ()
  Get minimum exponent.
- int get_exp_max ()
  Get maximum exponent.
- size_t get_sig_figs ()
  Get sig_figs.
- bool get_pad_zeros ()
  Get pad_zeros.
- std::string get_dec_point ()
  Get decimal point.

Protected Member Functions

- int remove_zeros_dpt (std::string &s)
  Remove extra zeros and decimal point from mantisaa.

Protected Attributes

Base text settings

- std::string prefx
  Prefix (default "")
- std::string suffx
  Suffix (default "")
- std::string sgn
  Sign string (default "-")
- std::string sci_sgn
  Sign string in scientific mode (default "-")
- std::string exp_sgn
  Sign string for exponent in scientific mode (default "-")
- std::string sci_prefx
  Prefix in scientific mode (default "")
- std::string sci_suffx
  Suffix in scientific mode (default "")
- std::string exp_prefx
  Exponent prefix (default "")
- std::string exp_suffx
  Exponent suffix (default "")
- std::string tmes
  Times symbol for scientific mode (default " x ")
- std::string not_finte
  String for numbers which are not finite (default "Nan")
- std::string zeros
  String for zeros (default "0")

Other settings

- size_t sig_fgs
  Number of significant figures (default 5)
- size_t exp_dgs
  Number of digits in exponent (default 0 which prints the minimum)
- int ex_mn
  Lower limit for automatic mode (default -2)
- int ex_mx
  Upper limit for automatic mode (default 3)
- bool pad_zeros
  If true, pad with zeros (default false)
- bool show_exp_sgn
  If true, show the sign of the exponent when it's positive (default false)
- std::string dpt
  The decimal point (default '.')

46.78.2 Member Function Documentation

46.78.2.1 int format_float::html_mode ( )

This function is equivalent to the settings:

```
set_prefix("");
set_sign("-");
set_suffix("");
set_sci_prefix("");
set_times(" &times; ");
set_exp_prefix("10<sup>");
set_exp_sign("-");
set_sci_sign("-");
set_exp_suffix("</sup>");
set_sci_suffix("");
set_not_finite("Nan");
set_zero("0");
set_exp_digits(0);
set_show_exp_sign(false);
```

46.78.2.2 int format_float::latex_mode ( )

This function is equivalent to the settings:

```
set_prefix("");
set_sign("$-$");
set_suffix("");
set_sci_prefix("");
set_times(" $\\times ");
set_exp_prefix("10^{");
set_exp_sign("-");
set_sci_sign("$-$");
set_exp_suffix("}");
set_sci_suffix("");
set_not_finite("Nan");
set_zero("0");
set_exp_digits(0);
set_show_exp_sign(false);
```

Note

This setting assumes that the user is not in LaTeX's "math mode" already.

46.78.2.3 int format_float::c_mode ( )

This reproduces the default settings of `cout` in automatic mode. Obviously it is faster to use iostreams than to format numbers with this class. Nevertheless, this mode is very useful for testing to ensure that this class processes the numbers correctly at the requested precision.

This function is equivalent to the settings:

```
set_prefix("");
set_sign("-");
set_suffix("");
set_sci_prefix("");
set_times("e");
set_exp_prefix("");
set_exp_sign("-");
set_sci_sign("-");
set_exp_suffix("");
set_sci_suffix("");
set_not_finite("NaN");
set_zero("0");
set_exp_limits(-4,5);
set_exp_digits(2);
set_show_exp_sign(true);
```

The documentation for this class was generated from the following file:

- format_float.h

46.79 gsl_fit< func_t, vec_t, mat_t, bool_vec_t >::func_par Struct Reference

A structure for passing to the functions func(), dfunc(), and fdfunc()

`#include <gsl_fit.h>`

46.79.1 Detailed Description

template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class bool_vec_t = bool *>struct gsl_fit< func_t, vec_t, mat_t, bool_vec_t >::func_par

Definition at line 433 of file gsl_fit.h.

Data Fields

- func_t & f
  The function object.
- int ndat
  The number.
- vec_t * xdat
  The x values.
- vec_t * ydat
  The y values.
- vec_t * yerr
  The y uncertainties.
- int npar
  The number of parameters.

The documentation for this struct was generated from the following file:

- gsl_fit.h

46.80 min_fit< func_t, vec_t, mat_t >::func_par Struct Reference

A structure for passing information to the GSL functions for the min_fit class.

`#include <min_fit.h>`

46.80.1 Detailed Description

template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base>struct min_fit< func_t, vec_t, mat_t >::func_par

This structure is given so that the user can specify the minimizer to use.

Definition at line 75 of file min_fit.h.

Data Fields

- func_t & f
  The fitting function.
- int ndat
  The number of data.
- vec_t * xdat
  The x values.
- vec_t * ydat
  The y values.
- vec_t * yerr
  The y uncertainties.
- int npar
  The number of fitting parameters.

The documentation for this struct was generated from the following file:

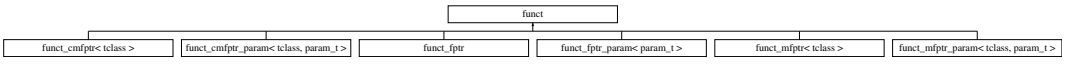
- min_fit.h

46.81 funct Class Reference

One-dimensional function [abstract base].

`#include <funct.h>`

Inheritance diagram for funct:



46.81.1 Detailed Description

This class generalizes a one-dimensional function y(x).

This class is one of a large number of function object classes in O_2scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See Function Objects for a general description.

Definition at line 41 of file funct.h.

Public Member Functions

- virtual double operator() (double x)=0
  Compute the function at point x, with result y.

The documentation for this class was generated from the following file:

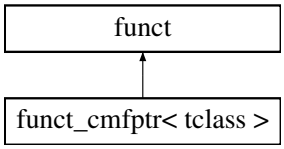
- funct.h

46.82 funct_cmfptr< tclass > Class Template Reference

Const member function pointer to a one-dimensional function.

`#include <funct.h>`

Inheritance diagram for funct_cmfptr< tclass >:



46.82.1 Detailed Description

template<class tclass>class funct_cmfptr< tclass >

Note

While this is designed to accept a pointer to a const member function, the choice of whether the class pointer given in the template type `tclass` is const or not is up to the user.

Definition at line 260 of file funct.h.

Public Member Functions

- funct_cmfptr (tclass *tp, double(tclass::*fp)(double x) const)
  Specify the member function pointer.
- virtual double operator() (double x)
  Compute the function at point x, with result y.

Protected Member Functions

- funct_cmfptr (const funct_cmfptr &f)
  Copy constructor.
- funct_cmfptr & operator= (const funct_cmfptr &f)
  Copy constructor.

Protected Attributes

- double(tclass::* fptr)(double x) const
  Storage for the const member function pointer.
- tclass * tptr

Store the pointer to the class instance.

The documentation for this class was generated from the following file:

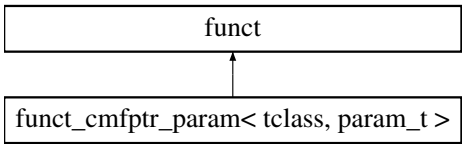
- funct.h

46.83 funct_cmfptr_param< tclass, param_t > Class Template Reference

Const member function pointer to a one-dimensional function with a parameter.

`#include <funct.h>`

Inheritance diagram for funct_cmfptr_param< tclass, param_t >:



46.83.1 Detailed Description

template<class tclass, class param_t>class funct_cmfptr_param< tclass, param_t >

Note

While this is designed to accept a pointer to a const member function, the choice of whether the class pointer given in the template type `tclass` is const or not is up to the user.

Definition at line 315 of file funct.h.

Public Member Functions

- funct_cmfptr_param (tclass *tp, double(tclass::*fp)(double x, param_t &pa) const, param_t &pa)
  Specify the member function pointer.
- virtual double operator() (double x, param_t &pa)
  Compute the function at point x, with result y.

Protected Member Functions

- funct_cmfptr_param (const funct_cmfptr_param &f)
  Copy constructor.
- funct_cmfptr_param & operator= (const funct_cmfptr_param &f)
  Copy constructor.

Protected Attributes

- double(tclass::* fptr)(double x, param_t &pa) const
  Storage for the const member function pointer.
- tclass * tptr
  Store the pointer to the class instance.
- param_t * pp
  The parameter.

The documentation for this class was generated from the following file:

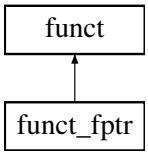
- funct.h

46.84 funct_fptr Class Reference

Function pointer to a function.

`#include <funct.h>`

Inheritance diagram for funct_fptr:



46.84.1 Detailed Description

Definition at line 61 of file funct.h.

Public Member Functions

- funct_fptr (double(∗fp)(double))
  Specify the function pointer.
- virtual double operator() (double x)
  Compute the function at point x, with result y.

Protected Member Functions

- funct_fptr (const funct_fptr &f)
  Copy constructor.
- funct_fptr & operator= (const funct_fptr &f)
  Copy constructor.

Protected Attributes

- double(∗ fptr)(double x)
  Function pointer.

The documentation for this class was generated from the following file:

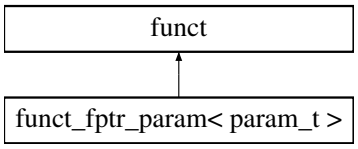
- funct.h

46.85 funct_fptr_param< param_t > Class Template Reference

Function pointer to a function with a parameter.

`#include <funct.h>`

Inheritance diagram for funct_fptr_param< param_t >:



46.85.1 Detailed Description

template<class param_t>class funct_fptr_param< param_t >

Definition at line 103 of file funct.h.

Public Member Functions

- funct_fptr_param (double(*fp)(double, param_t &), param_t &pa)
  Specify the function pointer.
- virtual double operator() (double x)
  Compute the function at point x, with result y.

Protected Member Functions

- funct_fptr_param (const funct_fptr_param &f)
  Copy constructor.
- funct_fptr_param & operator= (const funct_fptr_param &f)
  Copy constructor.

Protected Attributes

- double(* fptr)(double x, param_t &pa)
  The function pointer.
- param_t * pp
  The parameter.

The documentation for this class was generated from the following file:

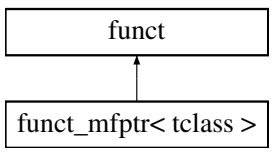
- funct.h

46.86 funct_mfptr< tclass > Class Template Reference

Member function pointer to a one-dimensional function.

`#include <funct.h>`

Inheritance diagram for funct_mfptr< tclass >:



46.86.1 Detailed Description

template<class tclass>class funct_mfptr< tclass >

Definition at line 149 of file funct.h.

Public Member Functions

- funct_mfptr (tclass *tp, double(tclass::*fp)(double x))
  Specify the member function pointer.
- virtual double operator() (double x)
  Compute the function at point x, with result y.

Protected Member Functions

- funct_mfptr (const funct_mfptr &f)
  Copy constructor.
- funct_mfptr & operator= (const funct_mfptr &f)
  Copy constructor.

Protected Attributes

- double(tclass::* fptr)(double x)
  Storage for the member function pointer.
- tclass * tptr
  Store the pointer to the class instance.

The documentation for this class was generated from the following file:

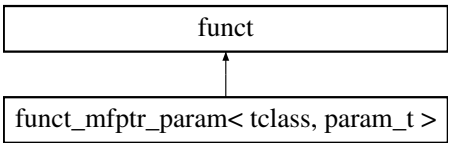
- funct.h

46.87 funct_mfptr_param< tclass, param_t > Class Template Reference

Member function pointer to a one-dimensional function with a parameter.

`#include <funct.h>`

Inheritance diagram for funct_mfptr_param< tclass, param_t >:



46.87.1 Detailed Description

template<class tclass, class param_t>class funct_mfptr_param< tclass, param_t >

Definition at line 198 of file funct.h.

Public Member Functions

- funct_mfptr_param (tclass *tp, double(tclass::*fp)(double x, param_t &pa), param_t &pa)
  Specify the member function pointer.
- virtual double operator() (double x)
  Compute the function at point x, with result y.

Protected Member Functions

- funct_mfptr_param (const funct_mfptr_param &f)
  Copy constructor.
- funct_mfptr_param & operator= (const funct_mfptr_param &f)
  Copy constructor.

Protected Attributes

- double(tclass::* fptr)(double x, param_t &pa)
  Storage for the member function pointer.
- tclass * tptr
  Store the pointer to the class instance.
- param_t * pp
  The parameter.

The documentation for this class was generated from the following file:

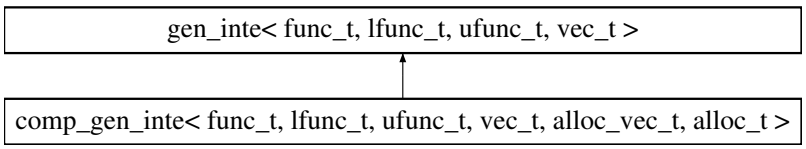
- funct.h

46.88 gen_inte< func_t, lfunc_t, ufunc_t, vec_t > Class Template Reference

Generalized multi-dimensional integration [abstract base].

`#include <gen_inte.h>`

Inheritance diagram for gen_inte< func_t, lfunc_t, ufunc_t, vec_t >:



46.88.1 Detailed Description

template<class func_t, class lfunc_t, class ufunc_t, class vec_t = ovector_base>class gen_inte< func_t, lfunc_t, ufunc_t, vec_t >

Perform the generalized multi-dimensional integral:

$$\int_{x_0=a_0}^{x_0=b_0} f(x_0) \int_{x_1=a_1(x_0)}^{x_1=b_1(x_0)} f(x_0,x_1) \ldots \int_{x_{n-1}=a_{n-1}(x_0,x_1,\ldots,x_{n-2})}^{x_{n-1}=b_{n-1}(x_0,x_1,\ldots,x_{n-2})} f(x_0,x_1,\ldots,x_{n-1})\, dx_{n-1} \ldots dx_1\, dx_0$$

The functions a_i and b_i are specified in the arguments `a` and `b` to the function ginteg() or ginteg_err() .

In order to allow the user to specify only three functions (for the integrand, the lower limits, and the upper limits) the first argument to the limit and integrand functions is used to distinguish among the limits for each separate integral. So first argument to `a` for $a_0()$

is 0, and the first argument to `a` for $a_1()$ is 1, etc., and similarly for the upper limits specified in `b` and the integrands specified in `func`.

At present, the only implementation of this abstract base is in comp_gen_inte .

Idea for Future It might be interesting to construct a child class of gen_inte which automatically transforms variables to a hypercube and then applies a child of multi_inte to do the integration.

Definition at line 62 of file gen_inte.h.

Public Member Functions

- virtual double ginteg (func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b)=0
  Integrate function `func` from $x_i = a_i(x_i)$ to $x_i = b_i(x_i)$ for $0 < i <$ ndim $- 1$.
- virtual int ginteg_err (func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, double &res, double &err)
  Integrate function `func` from $x_i = a_i(x_i)$ to $x_i = b_i(x_i)$ for $0 < i <$ ndim $- 1$.
- double get_error ()
  Return the error in the result from the last call to ginteg() or ginteg_err()
- const char * type ()
  Return string denoting type ("gen_inte")

Data Fields

- int verbose
  Verbosity.
- double tol_rel
  The maximum "uncertainty" in the value of the integral.
- bool err_nonconv
  If true, call the error handler if the routine does not "converge".

Protected Attributes

- double interror
  The uncertainty for the last integration computation.

46.88.2 Member Function Documentation

46.88.2.1 template<class func_t , class lfunc_t , class ufunc_t , class vec_t = ovector_base> double gen_inte< func_t, lfunc_t, ufunc_t, vec_t >::get_error ( ) [inline]

This will quietly return zero if no integrations have been performed.

Definition at line 108 of file gen_inte.h.

The documentation for this class was generated from the following file:

- gen_inte.h

46.89 gen_test_number< tot > Class Template Reference

Generate number sequence for testing.

```
#include <misc.h>
```

46.89.1 Detailed Description

template<size_t tot>class gen_test_number< tot >

A class which generates `tot` numbers from -1 to 1, making sure to include -1, 1, 0, and numbers near -1, 0 and 1 (so long as `tot` is sufficiently large). If gen() is called more than `tot` times, it just recycles through the list again.

This class is used to generate combinations of coefficients for testing the polynomial solvers.

For example, the first 15 numbers generated by an object of type gen_test_number<10> are:

```
0  -1.000000e+00
1  -9.975274e-01
2  -8.807971e-01
3  -1.192029e-01
4  -2.472623e-03
5  +0.000000e+00
6  +2.472623e-03
7  +1.192029e-01
8  +8.807971e-01
9  +1.000000e+00
10 -1.000000e+00
11 -9.975274e-01
12 -8.807971e-01
13 -1.192029e-01
14 -2.472623e-03
```

This function is used in `src/other/poly_ts.cpp` which tests the polynomial solvers.

Idea for Future Document what happens if `tot` is pathologically small.

Definition at line 219 of file misc.h.

Public Member Functions

- double gen ()
  Return the next number in the sequence.

Protected Attributes

- int n
  Count number of numbers generated.
- double fact
  A constant factor for the argument to `tanh()`, equal to `tot` divided by 20.

The documentation for this class was generated from the following file:

- misc.h

46.90 grad_funct< vec_t > Class Template Reference

Array of multi-dimensional functions [abstract base].

`#include <multi_min.h>`

Inheritance diagram for grad_funct< vec_t >:

46.90.1 Detailed Description

template< class vec_t = ovector_base > class grad_funct< vec_t >

This class generalizes `nv` functions of `nv` variables, i.e. $y_j(x_0, x_1, ..., x_{nv-1})$ for $0 \leq j \leq nv-1$.

This class is one of a large number of function object classes in O$_2$scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See Function Objects for a general description.

Note

This class is different from mm_funct in that the first vector argument is not const.

Definition at line 49 of file multi_min.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &y)=0
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

The documentation for this class was generated from the following file:

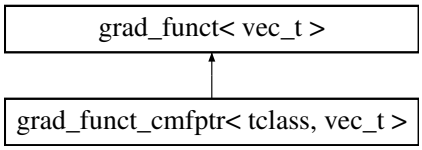
- multi_min.h

46.91 grad_funct_cmfptr< tclass, vec_t > Class Template Reference

Const member function pointer to an array of multi-dimensional functions.

`#include <multi_min.h>`

Inheritance diagram for grad_funct_cmfptr< tclass, vec_t >:



46.91.1 Detailed Description

template< class tclass, class vec_t = ovector_base > class grad_funct_cmfptr< tclass, vec_t >

Definition at line 303 of file multi_min.h.

Public Member Functions

- grad_funct_cmfptr ()
  Empty constructor.
- grad_funct_cmfptr (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y) const)

Specify the member function pointer.

- int set_function (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y) const)
  Specify the member function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nv, vec_t &x, vec_t &y) const
  The member function pointer.
- tclass * tptr
  The class pointer.

Private Member Functions

- **grad_funct_cmfptr** (const grad_funct_cmfptr &)
- grad_funct_cmfptr & **operator=** (const grad_funct_cmfptr &)

The documentation for this class was generated from the following file:

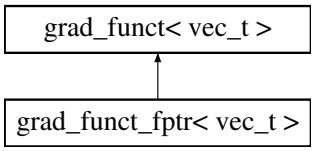
- multi_min.h

46.92 grad_funct_fptr< vec_t > Class Template Reference

Function pointer to array of multi-dimensional functions.

`#include <multi_min.h>`

Inheritance diagram for grad_funct_fptr< vec_t >:



46.92.1 Detailed Description

template<class vec_t = ovector_base>class grad_funct_fptr< vec_t >

Definition at line 74 of file multi_min.h.

Public Member Functions

- grad_funct_fptr (int(*fp)(size_t nv, vec_t &x, vec_t &y))
  Specify the function pointer.
- int set_function (int(*fp)(size_t nv, vec_t &x, vec_t &y))
  Specify the function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(∗ fptr)(size_t nv, vec_t &x, vec_t &y)
  The function pointer to the user-supplied function.

Private Member Functions

- **grad_funct_fptr** (const grad_funct_fptr &)
- grad_funct_fptr & **operator=** (const grad_funct_fptr &)

The documentation for this class was generated from the following file:

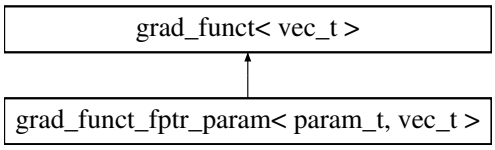
- multi_min.h

46.93 grad_funct_fptr_param< param_t, vec_t > Class Template Reference

Function pointer to array of multi-dimensional functions.

`#include <multi_min.h>`

Inheritance diagram for grad_funct_fptr_param< param_t, vec_t >:



46.93.1 Detailed Description

template<class param_t, class vec_t = ovector_base>class grad_funct_fptr_param< param_t, vec_t >

Definition at line 123 of file multi_min.h.

Public Member Functions

- grad_funct_fptr_param (int(∗fp)(size_t nv, vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the function pointer.
- int set_function (int(∗fp)(size_t nv, vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(∗ fptr)(size_t nv, vec_t &x, vec_t &y, param_t &)
  The function pointer to the user-supplied function.
- param_t ∗ pp
  The parameter.

Private Member Functions

- **grad_funct_fptr_param** (const grad_funct_fptr_param &)
- grad_funct_fptr_param & **operator=** (const grad_funct_fptr_param &)

The documentation for this class was generated from the following file:

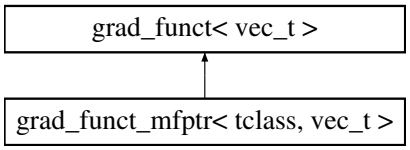
- multi_min.h

46.94 grad_funct_mfptr< tclass, vec_t > Class Template Reference

Member function pointer to an array of multi-dimensional functions.

`#include <multi_min.h>`

Inheritance diagram for grad_funct_mfptr< tclass, vec_t >:



46.94.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class grad_funct_mfptr< tclass, vec_t >

Definition at line 181 of file multi_min.h.

Public Member Functions

- grad_funct_mfptr ()
  Empty constructor.
- grad_funct_mfptr (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y))
  Specify the member function pointer.
- int set_function (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y))
  Specify the member function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nv, vec_t &x, vec_t &y)
  The member function pointer.
- tclass * tptr
  The class pointer.

Private Member Functions

- **grad_funct_mfptr** (const grad_funct_mfptr &)
- grad_funct_mfptr & **operator=** (const grad_funct_mfptr &)

The documentation for this class was generated from the following file:

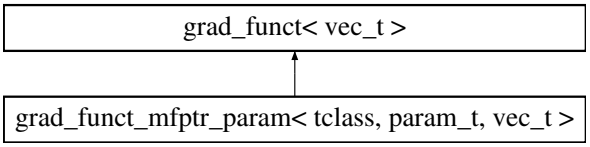
- multi_min.h

46.95 grad_funct_mfptr_param< tclass, param_t, vec_t > Class Template Reference

Member function pointer to an array of multi-dimensional functions.

`#include <multi_min.h>`

Inheritance diagram for grad_funct_mfptr_param< tclass, param_t, vec_t >:



46.95.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base>class grad_funct_mfptr_param< tclass, param_t, vec_t >

Definition at line 239 of file multi_min.h.

Public Member Functions

- grad_funct_mfptr_param ()
  Empty constructor.
- grad_funct_mfptr_param (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the member function pointer.
- int set_function (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the member function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nv, vec_t &x, vec_t &y, param_t &pa)
  The member function pointer.
- tclass * tptr
  The class pointer.
- param_t * pp
  Parameter.

Private Member Functions

- **grad_funct_mfptr_param** (const grad_funct_mfptr_param &)
- grad_funct_mfptr_param & **operator=** (const grad_funct_mfptr_param &)

The documentation for this class was generated from the following file:

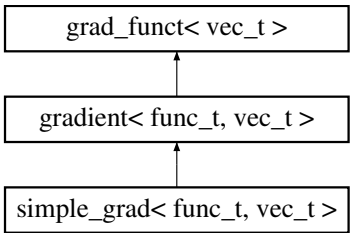
- multi_min.h

46.96 gradient< func_t, vec_t > Class Template Reference

Class for automatically computing gradients [abstract base].

`#include <multi_min.h>`

Inheritance diagram for gradient< func_t, vec_t >:



46.96.1 Detailed Description

template<class func_t, class vec_t = ovector_base>class gradient< func_t, vec_t >

Default template arguments

- `func_t` - (no default)
- `vec_t` - ovector_base

Idea for Future Consider making an exact_grad class for computing exact gradients.

Definition at line 368 of file multi_min.h.

Public Member Functions

- virtual int set_function (func_t &f)
  Set the function to compute the gradient of.
- virtual int operator() (size_t nv, vec_t &x, vec_t &g)=0
  Compute the gradient g at the point x.

Protected Attributes

- func_t * func
  A pointer to the user-specified function.

The documentation for this class was generated from the following file:

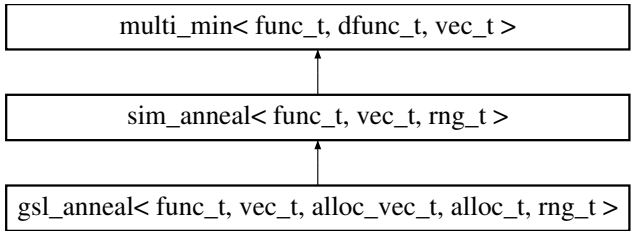
- multi_min.h

46.97 gsl_anneal< func_t, vec_t, alloc_vec_t, alloc_t, rng_t > Class Template Reference

Multidimensional minimization by simulated annealing (GSL)

`#include <gsl_anneal.h>`

Inheritance diagram for gsl_anneal< func_t, vec_t, alloc_vec_t, alloc_t, rng_t >:



46.97.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class rng_t = gsl_rnga>class gsl_anneal< func_t, vec_t, alloc_vec_t, alloc_t, rng_t >

This class is a modification of simulated annealing as implemented in GSL in the function `gsl_siman_solve()`. It acts as a generic multidimensional minimizer for any function given a generic temperature schedule specified by the user.

There are a large variety of strategies for choosing the temperature evolution. To offer the user the largest possible flexibility, the temperature evolution is controlled by a the virtual functions start() and next() which can be freely changed by creating a child class which overwrites these functions.

The simulated annealing algorithm proposes a displacement of one coordinate of the previous point by

$$x_{i,\mathrm{new}} = \mathrm{step\_size}_i(2u_i - 1) + x_{i,\mathrm{old}}$$

where the u_i are random numbers between 0 and 1. The displacement is accepted or rejected based on the Metropolis method. The random number generator is set in the parent, sim_anneal.

The default behavior is as follows: Initially, the step sizes are chosen to be 1.0 (or whatever was recently specified in set_step()) and the temperature to be T_start (default 1.0). Each iteration decreases the temperature by a factor of T_dec (default 1.5) for each step, and the minimizer is finished when the next decrease would bring the temperature below multi_min::tol_abs. If none of the multi_min::ntrial steps in a particular iteration changes the value of the minimum, and the step sizes are greater than min_step_ratio (default 100) times multi_min::tol_abs, then the step sizes are decreased by a factor of step_dec (default 1.5) for the next iteration.

If multi_min::verbose is greater than zero, then mmin() will print out information and/or request a keypress after the function iterations for each temperature.

An example demonstrating the usage of this class is given in `examples/ex_anneal.cpp` and in the Simulated annealing .

The form of the user-specified function is as in multi_funct has a "function value" which is the value of the function (given in the third argument as a number of type `double`), and a "return value" (the integer return value). The initial function evaluation which is performed at the user-specified initial guess must give 0 as the return value. If subsequent function evaluations have a non-zero return value, then the resulting point is ignored and a new point is selected.

This class thus can sometimes handle constrained minimization problems. If the user ensures that the function's return value is non-zero when the function is evaluated outside the allowed region, the minimizer will not accept any steps which take the minimizer outside the allowed region. Note that this should be done with care, however, as this approach may cause convergence problems with sufficiently difficult functions or constraints.

See also a multi-threaded version of this class in anneal_mt.

Idea for Future There's x0, old_x, new_x, best_x, and x? There's probably some duplication here which could be avoided.

Idea for Future

- Implement a more general simulated annealing routine which would allow the solution of discrete problems like the Traveling Salesman problem.

Definition at line 134 of file gsl_anneal.h.

Public Member Functions

- virtual int mmin (size_t nvar, vec_t &x0, double &fmin, func_t &func)
  Calculate the minimum `fmin` of `func` w.r.t the array `x0` of size `nvar`.
- virtual const char * type ()
  Return string denoting type ("gsl_anneal")
- template<class vec2_t >
  int set_step (size_t nv, vec2_t &stepv)
  Set the step sizes.

Data Fields

- double boltz
  Boltzmann factor (default 1.0).
- double T_start
  Initial temperature (default 1.0)
- double T_dec
  Factor to decrease temperature by (default 1.5)
- double step_dec
  Factor to decrease step size by (default 1.5)
- double min_step_ratio
  Ratio between minimum step size and tol_abs (default 100.0)

Protected Member Functions

- virtual int next (size_t nvar, vec_t &x_old, double min_old, vec_t &x_new, double min_new, double &T, size_t n_moves, bool &finished)
  Determine how to change the minimization for the next iteration.
- virtual int start (size_t nvar, double &T)
  Setup initial temperature and stepsize.
- virtual int allocate (size_t n, double boltz_factor=1.0)
  Allocate memory for a minimizer over `n` dimensions with stepsize `step` and Boltzmann factor `boltz_factor`.
- virtual int free ()
  Free allocated memory.
- virtual int step (vec_t &sx, int nvar)
  Make a step to a new attempted minimum.

Protected Attributes

- alloc_t ao
  Allocation object.
- ovector step_vec
  Vector of step sizes.

Storage for present, next, and best vectors

- alloc_vec_t **x**
- alloc_vec_t **new_x**
- alloc_vec_t **best_x**
- alloc_vec_t **old_x**

The documentation for this class was generated from the following file:

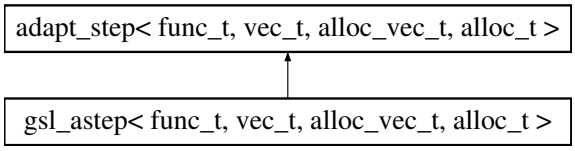
- gsl_anneal.h

46.98 gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Adaptive ODE stepper (GSL)

`#include <gsl_astep.h>`

Inheritance diagram for gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >:



46.98.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >

This class performs an adaptive step of a system of ODEs. To modify the ODE stepper which is used, use the function adapt_step::set_step().

Note, this has been updated to correspond to the `ode-initval2` functions in GSL.

There is an example for the usage of this class in `examples/ex_ode.cpp` documented in the Ordinary differential equations section.

Todo Document what happens when the stepper function returns a non-zero value, as it's different now with the ode-initval2 function.

Document count, failed_steps, etc.

Idea for Future Compare more directly to GSL

Default template arguments

- `func_t` - ode_funct
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc

Definition at line 259 of file gsl_astep.h.

Public Member Functions

- virtual int astep (double &x, double xmax, double &h, size_t n, vec_t &y, vec_t &dydx_out, vec_t &yerr, func_t &derivs)
  Make an adaptive integration step of the system `derivs`.
- virtual int astep_derivs (double &x, double xmax, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yerr, func_t &derivs)
  Make an adaptive integration step of the system `derivs` with derivatives.
- virtual int astep_full (double x, double xmax, double &x_out, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Make an adaptive integration step of the system `derivs`.

Data Fields

- gsl_ode_control< vec_t > con
  Control specification.

Protected Member Functions

- int evolve_apply (double t0, double t1, double &t, double &h, size_t nvar, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Apply the evolution for the next adaptive step.

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- alloc_vec_t yout_int
  Temporary storage for yout.
- alloc_vec_t dydx_int
  Internal storage for dydx.
- double last_step
  The size of the last step.
- unsigned long int count
  The number of steps.
- unsigned long int failed_steps
  The number of failed steps.
- size_t msize
  The size of the allocated vectors.

46.98.2 Member Function Documentation

46.98.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >::evolve_apply (double *t0*, double *t1*, double & *t*, double & *h*, size_t *nvar*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, protected]

This function is based on `gsl_odeiv2_evolve_apply`.

Note

This function requres that `y`, `yout`, `dydx` and `dydx_out` are all distinct vectors.

Definition at line 293 of file gsl_astep.h.

46.98.2.2 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >::astep (double & *x*, double *xmax*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx_out*, vec_t & *yerr*, func_t & *derivs*) [inline, virtual]

This attempts to take a step of size `h` from the point `x` of an `n-dimensional` system `derivs` starting with `y`. On exit, `x` and `y` contain the new values at the end of the step, `h` contains the size of the step, `dydx_out` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 452 of file gsl_astep.h.

46.98.2.3 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >::astep_derivs (double & *x*, double *xmax*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yerr*, func_t & *derivs*) [inline, virtual]

This attempts to take a step of size `h` from the point `x` of an `n-dimensional` system `derivs` starting with `y` and given the initial derivatives `dydx`. On exit, `x`, `y` and `dydx` contain the new values at the end of the step, `h` contains the size of the step, `dydx` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 497 of file gsl_astep.h.

46.98.2.4 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t >::astep_full (double *x*, double *xmax*, double & *x_out*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

This function performs an adaptive integration step with the `n-dimensional` system `derivs` and parameter `pa`. It Begins at `x` with initial stepsize `h`, ensuring that the step goes no farther than `xmax`. At the end of the step, the size of the step taken is `h` and the new value of `x` is in `x_out`. Initially, the function values and derivatives should be specified in `y` and `dydx`. The function values, derivatives, and the error at the end of the step are given in `yout`, `yerr`, and `dydx_out`. Unlike in `ode_step` objects, the objects `y`, `yout`, `dydx`, and `dydx_out` must all be distinct.

This adaptive stepper function is faster than astep() or astep_derivs() because it does not require any copying of vectors.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 548 of file gsl_astep.h.

The documentation for this class was generated from the following file:

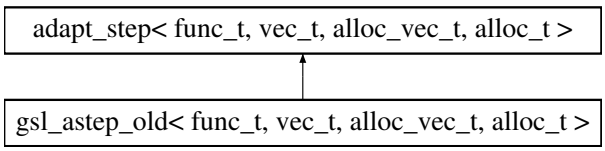
- gsl_astep.h

46.99 gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Adaptive ODE stepper (GSL)

`#include <gsl_astep_old.h>`

Inheritance diagram for gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >:



46.99.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >

[This is the stepper from O_2scl version 0.908 which is kept around for diagnostic purposes and some legacy code.]

This class performs an adaptive step of a system of ODEs. To modify the ODE stepper which is used, use the function adapt_step::set_step().

There is an example for the usage of this class in `examples/ex_ode.cpp` documented in the Ordinary differential equations section.

Default template arguments

- `func_t` - ode_funct
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc

Definition at line 78 of file gsl_astep_old.h.

Public Member Functions

- virtual int astep_derivs (double &x, double xmax, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yerr, func_t &derivs)
  Make an adaptive integration step of the system `derivs` with derivatives.
- virtual int astep_full (double x, double xmax, double &x_out, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout2, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Make an adaptive integration step of the system `derivs`.
- virtual int astep (double &x, double xmax, double &h, size_t n, vec_t &y, vec_t &dydx_out, vec_t &yerr, func_t &derivs)
  Make an adaptive integration step of the system `derivs`.

Data Fields

- gsl_ode_control< vec_t > con
  Control specification.

Protected Member Functions

- int evolve_apply (double t0, double &t, double &h, double t1, size_t nvar, vec_t &y, vec_t &dydx, vec_t &yout2, vec_t &yerr, vec_t &dydx_out2, func_t &derivs)
  Apply the evolution for the next adaptive step.

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- alloc_vec_t yout
  Temporary storage for yout.
- alloc_vec_t dydx_int
  Internal storage for dydx.
- double last_step
  The size of the last step.
- unsigned long int count
  The number of steps.
- unsigned long int failed_steps
  The number of failed steps.
- size_t msize
  The size of the allocated vectors.

46.99.2 Member Function Documentation

46.99.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >::astep_derivs (double & *x*, double *xmax*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yerr*, func_t & *derivs*) [inline, virtual]

This attempts to take a step of size `h` from the point `x` of an `n-dimensional` system `derivs` starting with `y` and given the initial derivatives `dydx`. On exit, `x`, `y` and `dydx` contain the new values at the end of the step, `h` contains the size of the step, `dydx` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

If the base stepper returns a non-zero value, the step is aborted and y is unmodified. The error handler is never called.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 209 of file gsl_astep_old.h.

46.99.2.2 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >::astep_full (double *x*, double *xmax*, double & *x_out*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout2*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

This function performs an adaptive integration step with the n-dimensional system derivs and parameter pa. It Begins at x with initial stepsize h, ensuring that the step goes no farther than xmax. At the end of the step, the size of the step taken is h and the new value of x is in x_out. Initially, the function values and derivatives should be specified in y and dydx. The function values, derivatives, and the error at the end of the step are given in yout, yerr, and dydx_out. Unlike in ode_step objects, the objects y, yout, dydx, and dydx_out must all be distinct.

If the base stepper returns a non-zero value, the step is aborted. The error handler is never called.

This adaptive stepper function is faster than astep() or astep_derivs() because it does not require any copying of vectors.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 263 of file gsl_astep_old.h.

46.99.2.3 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_astep_old< func_t, vec_t, alloc_vec_t, alloc_t >::astep (double & *x*, double *xmax*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx_out*, vec_t & *yerr*, func_t & *derivs*) [inline, virtual]

This attempts to take a step of size h from the point x of an n-dimensional system derivs starting with y. On exit, x and y contain the new values at the end of the step, h contains the size of the step, dydx_out contains the derivative at the end of the step, and yerr contains the estimated error at the end of the step.

If the system derivs or the base stepper return a non-zero value, the adaptive step is aborted and y is unmodified. The error handler is never called.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 298 of file gsl_astep_old.h.

The documentation for this class was generated from the following file:

- gsl_astep_old.h

46.100 gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t > Class Template Reference

Bulirsch-Stoer implicit ODE stepper (GSL)

```
#include <gsl_bsimp.h>
```

46.100.1 Detailed Description

template<class func_t, class jac_func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc>class gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >

Bader-Deuflhard implicit extrapolative stepper (Bader83).

Note

The variable `h_next` was defined in the original GSL version has been removed here, as it was unused by the stepper routine. At the moment, this class retains the GSL approach to handling non-integer return values in the user-specified derivative function. If the user-specified function returns `gsl_efailed`, then the stepper attempts to decrease the stepsize to continue. If the user-specified function returns a non-zero value other than `gsl_efailed`, or if the Jacobian evaluation returns any non-zero value, then the stepper aborts and returns the error value without calling the error handler. This behavior may change in the future.

Idea for Future Create an example with a stiff diff eq. which requires this kind of stepper

Idea for Future More detailed documentation about the algorithm

Idea for Future Figure out if this should be a child of ode_step or adapt_step. The function step_local() is actually its own ODE stepper and could be reimplemented as an object of type ode_step.

Idea for Future I don't like setting yerr to GSL_POSINF, there should be a better way to force an adaptive stepper which is calling this stepper to readjust the stepsize.

Idea for Future The functions deuf_kchoice() and compute_weights() can be moved out of this header file

Idea for Future Rework internal arrays as uvectors?

Idea for Future Rework linear solver to be amenable to using a sparse matrix solver

Definition at line 102 of file gsl_bsimp.h.

Public Member Functions

- int reset ()
  Reset stepper.
- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs, jac_func_t &jac)
  Perform an integration step.

Protected Member Functions

- int compute_weights (const double y[ ], double w[ ], size_t n)
  Compute weights.
- size_t deuf_kchoice (double eps2, size_t dimension)
  Calculate a choice for the "order" of the method, using the Deuflhard criteria.
- int poly_extrap (gsl_matrix *dloc, const double x[ ], const unsigned int i_step, const double x_i, const vec_t &y_i, vec_t &y_0, vec_t &y_0_err, double work[ ])
  Polynomial extrapolation.
- int step_local (const double t0, const double h_total, const unsigned int n_step, const double y[ ], const double yp_loc[ ], const vec_t &dfdt_loc, const mat_t &dfdy_loc, vec_t &y_out)
  Basic implicit Bulirsch-Stoer step.
- int allocate (size_t n)
  Allocate memory for a system of size `n`.
- void free ()
  Free allocated memory.

Protected Attributes

- size_t dim
  Size of allocated vectors.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- mat_alloc_t mo
  Memory allocator for objects of type `alloc_mat_t`.
- func_t * funcp
  Function specifying derivatives.
- jac_func_t * jfuncp
  Jacobian.
- gsl_matrix * d
  Workspace for extrapolation.
- gsl_matrix * a_mat
  Workspace for linear system matrix.
- double ex_wk [sequence_max]
  Workspace for extrapolation.
- size_t order
  Order of last step.

State info

- size_t **k_current**
- size_t **k_choice**
- double **eps**

Workspace for extrapolation step

- double * **yp**
- double * **y_save**
- double * **yerr_save**
- double * **y_extrap_save**
- alloc_vec_t **y_extrap_sequence**
- double * **extrap_work**
- alloc_vec_t **dfdt**
- alloc_vec_t **y_temp**
- alloc_vec_t **delta_temp**
- double * **weight**
- alloc_mat_t **dfdy**

Workspace for the basic stepper

- alloc_vec_t **rhs_temp**
- double * **delta**

Static Protected Attributes

- static const int sequence_count = 8
  Number of entries in the Bader-Deuflhard extrapolation sequence.
- static const int sequence_max = 7
  Largest index of the Bader-Deuflhard extrapolation sequence.

46.100.2 Member Function Documentation

46.100.2.1 template<class func_t, class jac_func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> size_t gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >::deuf_kchoice (double *eps2*, size_t *dimension*) [inline, protected]

Used in the allocate() function.

Definition at line 189 of file gsl_bsimp.h.

46.100.2.2 template<class func_t, class jac_func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> int gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >::poly_extrap (gsl_matrix * *dloc,* const double *x[],* const unsigned int *i_step,* const double *x_i,* const vec_t & *y_i,* vec_t & *y_0,* vec_t & *y_0_err,* double *work[]*) [inline, protected]

Compute the step of index `i_step` using polynomial extrapolation to evaulate functions by fitting a polynomial to estimates `(x_-i,y_i)` and output the result to `y_0` and `y_0_err`.

The index `i_step` begins with zero.

Definition at line 244 of file gsl_bsimp.h.

46.100.2.3 template<class func_t, class jac_func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> int gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >::step_local (const double *t0,* const double *h_total,* const unsigned int *n_step,* const double *y[],* const double *yp_loc[],* const vec_t & *dfdt_loc,* const mat_t & *dfdy_loc,* vec_t & *y_out*) [inline, protected]

Divide the step `h_total` into `n_step` smaller steps and do the Bader-Deuflhard semi-implicit iteration. This function starts at `t0` with function values `y`, derivatives `yp_loc`, and information from the Jacobian to compute the final value `y_out`.

Definition at line 293 of file gsl_bsimp.h.

46.100.2.4 template<class func_t, class jac_func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> virtual int gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >::step (double *x,* double *h,* size_t *n,* vec_t & *y,* vec_t & *dydx,* vec_t & *yout,* vec_t & *yerr,* vec_t & *dydx_out,* func_t & *derivs,* jac_func_t & *jac*) [inline, virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must generally be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` (at `x+h`) using function `derivs` to calculate derivatives. Implementations which do not calculate `yerr` and/or `dydx_out` do not reference these variables so that a blank `vec_t` can be given. All of the implementations allow `yout=y` and `dydx_out=dydx` if necessary.

Definition at line 504 of file gsl_bsimp.h.

46.100.3 Field Documentation

46.100.3.1 template<class func_t, class jac_func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> double gsl_bsimp< func_t, jac_func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >::ex_wk[sequence_max] [protected]

(This state variable was named 'x' in GSL.)

Definition at line 139 of file gsl_bsimp.h.

The documentation for this class was generated from the following file:

- gsl_bsimp.h

46.101 gsl_chebapp Class Reference

Chebyshev approximation (GSL)

`#include <gsl_chebapp.h>`

46.101.1 Detailed Description

Approximate a function on a finite interval using a Chebyshev series:

$$f(x) = \sum_n c_n T_n(x) \qquad \text{where} \qquad T_n(x) = \cos(n \arccos x)$$

Definition at line 44 of file gsl_chebapp.h.

Public Member Functions

- gsl_chebapp (const gsl_chebapp &gc)
  Copy constructor.
- gsl_chebapp & operator= (const gsl_chebapp &gc)
  Copy constructor.

Initialization methods

- template<class func_t >
  int init (func_t &func, size_t ord, double a1, double b1)
  Initialize a Chebyshev approximation of the function `func` over the interval from `a1` to `b1`.
- template<class vec_t >
  int init (double a1, double b1, size_t ord, vec_t &v)
  Create an approximation from a vector of coefficients.
- template<class vec_t >
  int init_func_values (double a1, double b1, size_t ord, vec_t &fval)
  Create an approximation from a vector of function values.

Evaulation methods

- double eval (double x) const
  Evaluate the approximation.
- double operator() (double x) const
  Evaluate the approximation.
- double eval_n (size_t n, double x) const
  Evaluate the approximation to a specified order.
- int eval_err (double x, double &result, double &abserr)
  Evaluate the approximation and give the uncertainty.
- int eval_n_err (size_t n, double x, double &result, double &abserr)
  Evaluate the approximation to a specified order and give the uncertainty.

Maniupulating coefficients and endpoints

- double get_coefficient (size_t ix) const
  Get a coefficient.
- int set_coefficient (size_t ix, double co)
  Set a coefficient.
- int get_endpoints (double &la, double &lb)
  Return the endpoints of the approximation.
- template<class vec_t >
  int get_coefficients (size_t n, vec_t &v) const
  Get the coefficients.
- template<class vec_t >
  int set_coefficients (size_t n, const vec_t &v)
  Set the coefficients.

Derivatives and integrals

- int deriv (gsl_chebapp &gc) const
  Make `gc` an approximation to the derivative.
- int integ (gsl_chebapp &gc) const
  Make `gc` an approximation to the integral.

Protected Attributes

- uvector c
  Coefficients.
- size_t order
  Order of the approximation.
- double a
  Lower end of the interval.
- double b
  Upper end of the interval.
- size_t order_sp
  Single precision order.
- uvector f
  Function evaluated at Chebyshev points.
- bool init_called
  True if init has been called.

46.101.2 Member Function Documentation

46.101.2.1 template<class func_t > int gsl_chebapp::init (func_t & *func*, size_t *ord*, double *a1*, double *b1*) [inline]

The interval must be specified so that $a < b$, so a and b are swapped if this is not the case.

Definition at line 107 of file gsl_chebapp.h.

46.101.2.2 double gsl_chebapp::get_coefficient (size_t *ix*) const [inline]

Legal values of the argument are 0 to `order` (inclusive)

Definition at line 326 of file gsl_chebapp.h.

46.101.2.3 int gsl_chebapp::set_coefficient (size_t *ix*, double *co*) [inline]

Legal values of the argument are 0 to `order` (inclusive)

Definition at line 340 of file gsl_chebapp.h.

The documentation for this class was generated from the following file:

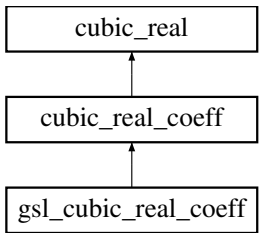
- gsl_chebapp.h

46.102 gsl_cubic_real_coeff Class Reference

Solve a cubic with real coefficients and complex roots (GSL)

`#include <poly.h>`

Inheritance diagram for gsl_cubic_real_coeff:



46.102.1 Detailed Description

Definition at line 495 of file poly.h.

Public Member Functions

- virtual int solve_rc (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)
  Solves the polynomial $a_3x^3 + b_3x^2 + c_3x + d_3 = 0$ giving the real solution $x = x_1$ and two complex solutions $x = x_2$ and $x = x_3$.
- const char * type ()
  Return a string denoting the type ("gsl_cubic_real_coeff")
- int gsl_poly_complex_solve_cubic2 (double a, double b, double c, gsl_complex *z0, gsl_complex *z1, gsl_complex *z2)
  An alternative to `gsl_poly_complex_solve_cubic()`

46.102.2 Member Function Documentation

46.102.2.1 int gsl_cubic_real_coeff::gsl_poly_complex_solve_cubic2 (double *a*, double *b*, double *c*, gsl_complex * *z0*, gsl_complex * *z1*, gsl_complex * *z2*)

This is an alternative to the function `gsl_poly_complex_solve_cubic()` with some small corrections to ensure finite values for some cubics. See `src/other/poly_ts.cpp` for more.

Idea for Future I think the GSL function is now fixed, so we can fall back to the original GSL function here.

The documentation for this class was generated from the following file:

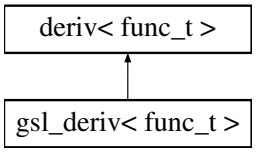
- poly.h

46.103 gsl_deriv< func_t > Class Template Reference

Numerical differentiation (GSL)

`#include <gsl_deriv.h>`

Inheritance diagram for gsl_deriv< func_t >:



46.103.1 Detailed Description

template<class func_t = funct>class gsl_deriv< func_t >

This class computes the numerical derivative of a function. The stepsize h should be specified before use. If similar functions are being differentiated in succession, the user may be able to increase the speed of later derivatives by setting the new stepsize equal to the optimized stepsize from the previous differentiation, by setting h to h_opt.

The derivative computation will never fail, but the results will be incorrect for sufficiently difficult functions or if the step size is not properly chosen.

Some successive derivative computations can be made more efficient by using the optimized stepsize in gsl_deriv::h_opt , which is set by the most recent last derivative computation.

Setting deriv::verbose to a number greater than zero results in output for each call to central_deriv() which looks like:

```
gsl_deriv:
step: 1.000000e-04
abscissas: 4.999500e-01 4.999000e-01 5.000500e-01 5.001000e-01
ordinates: 4.793377e-01 4.793816e-01 4.794694e-01 4.795132e-01
res: 8.775825e-01 trc: 1.462163e-09 rnd: 7.361543e-12
```

where the last line contains the result (`res`), the truncation error (`trc`) and the rounding error (`rnd`). If deriv::verbose is greater than 1, a keypress is required after each iteration.

Computing first derivatives requires either 1 or 2 calls to central_deriv() and thus either 4 or 8 function calls. This class never calls the error handler.

Note

Second and third derivatives are computed by naive nested applications of the formula for the first derivative. No uncertainty for these derivatives is provided.

An example demonstrating the usage of this class is given in `examples/ex_deriv.cpp` and the Numerical differentiation .

Idea for Future Include the forward and backward GSL derivatives?

Definition at line 105 of file gsl_deriv.h.

Public Member Functions

- virtual int calc_err (double x, func_t &func, double &dfdx, double &err)
  Calculate the first derivative of `func` w.r.t. x and uncertainty.
- virtual const char * type ()
  Return string denoting type ("gsl_deriv")

Data Fields

- double h
  Initial stepsize.
- double h_opt
  The last value of the optimized stepsize.

Protected Member Functions

- template<class func2_t >
  int calc_base (double x, func_t &func, double &dfdx, double &err)
  Internal template version of the derivative function.
- virtual int calc_err_int (double x, funct &func, double &dfdx, double &err)
  Internal version of calc_err() for second and third derivatives.
- template<class func2_t >
  int central_deriv (double x, double hh, double &result, double &abserr_round, double &abserr_trunc, func2_t &func)
  Compute derivative using 5-point rule.

46.103.2 Member Function Documentation

46.103.2.1 template<class func_t = funct> template<class func2_t > int gsl_deriv< func_t >::central_deriv (double *x*, double *hh*, double & *result*, double & *aberr_round*, double & *aberr_trunc*, func2_t & *func*) [inline, protected]

Compute the derivative using the 5-point rule (x-h, x-h/2, x, x+h/2, x+h) and the error using the difference between the 5-point and the 3-point rule (x-h,x,x+h). Note that the central point is not used for either.

This must be a class template because it is used by both calc_err() and calc_err_int().

Definition at line 207 of file gsl_deriv.h.

46.103.3 Field Documentation

46.103.3.1 template<class func_t = funct> double gsl_deriv< func_t >::h

This should be specified before a call to calc() or calc_err(). If it is zero, then $x10^{-4}$ will used, or if x is zero, then 10^{-4} will be used.

Definition at line 122 of file gsl_deriv.h.

46.103.3.2 template<class func_t = funct> double gsl_deriv< func_t >::h_opt

This is initialized to zero in the constructor and set by calc_err() to the most recent value of the optimized stepsize.

Definition at line 129 of file gsl_deriv.h.

The documentation for this class was generated from the following file:

- gsl_deriv.h

46.104 gsl_fft Class Reference

Real mixed-radix fast Fourier transform.

`#include <gsl_fft.h>`

46.104.1 Detailed Description

This is a simple wrapper for the GSL FFT functions which automatically allocates the necessary memory.

Definition at line 40 of file gsl_fft.h.

Public Member Functions

- int transform (int n, double *x)
  Perform the FFT transform.
- int inverse_transform (int n, double *x)
  Perform the inverse FFT transform.

Protected Member Functions

- int mem_resize (int new_size)
  Reallocate memory.

Protected Attributes

- int mem_size
  The current memory size.
- gsl_fft_real_workspace * work
  The GSL workspace.
- gsl_fft_real_wavetable * real
  The table for the forward transform.
- gsl_fft_halfcomplex_wavetable * hc
  The table for the inverse transform.

The documentation for this class was generated from the following file:

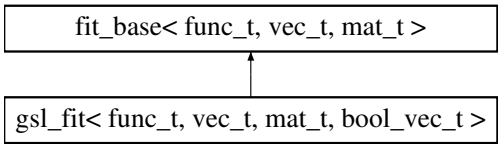
- gsl_fft.h

46.105 gsl_fit< func_t, vec_t, mat_t, bool_vec_t > Class Template Reference

Non-linear least-squares fitting class (GSL)

`#include <gsl_fit.h>`

Inheritance diagram for gsl_fit< func_t, vec_t, mat_t, bool_vec_t >:



46.105.1 Detailed Description

template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class bool_vec_t = bool *>class gsl_fit< func_t, vec_t, mat_t, bool_vec_t >

The GSL-based fitting class using a Levenberg-Marquardt type algorithm. The algorithm stops when

$$|dx_i| < \text{epsabs} + \text{epsrel} \times |x_i|$$

where dx is the last step and x is the current position. If test_gradient is true, then additionally fit() requires that

$$\sum_i |g_i| < \text{epsabs}$$

where g_i is the i-th component of the gradient of the function $\Phi(x)$ where

$$\Phi(x) = ||F(x)||^2$$

Default template arguments

- `func_t` - fit_funct<>
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `bool_vec_t` - `bool *`

Todo Properly generalize other vector types than ovector_base

Allow the user to specify the derivatives

Fix so that the user can specify automatic scaling of the fitting parameters, where the initial guess are used for scaling so that the fitting parameters are near unity.

Definition at line 91 of file gsl_fit.h.

Data Structures

- struct func_par
  A structure for passing to the functions func(), dfunc(), and fdfunc()

Public Member Functions

- virtual int fit (size_t ndat, vec_t &xdat, vec_t &ydat, vec_t &yerr, size_t npar, vec_t &par, mat_t &covar, double &chi2, func_t &fitfun)
  Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.
- virtual const char * type ()
  Return string denoting type ("gsl_fit")

Data Fields

- int max_iter
  (default 500)
- double epsabs
  (default 1.0e-4)
- double epsrel
  (default 1.0e-4)
- bool test_gradient
  If true, test the gradient also (default false)
- bool use_scaled
  Use the scaled routine if true (default true)

Protected Member Functions

- virtual int print_iter (size_t nv, gsl_vector *x, int iter, double val, double lim)
  Print the progress in the current iteration.

Static Protected Member Functions

- static int func (const gsl_vector *x, void *pa, gsl_vector *f)
  Evaluate the function.
- static int dfunc (const gsl_vector *x, void *pa, gsl_matrix *jac)
  Evaluate the jacobian.
- static int fdfunc (const gsl_vector *x, void *pa, gsl_vector *f, gsl_matrix *jac)
  Evaluate the function and the jacobian.

46.105.2 Member Function Documentation

46.105.2.1 template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class bool_vec_t = bool *> virtual int gsl_fit< func_t, vec_t, mat_t, bool_vec_t >::fit (size_t *ndat*, vec_t & *xdat*, vec_t & *ydat*, vec_t & *yerr*, size_t *npar*, vec_t & *par*, mat_t & *covar*, double & *chi2*, func_t & *fitfun*) [inline, virtual]

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Implements fit_base< func_t, vec_t, mat_t >.

Definition at line 261 of file gsl_fit.h.

The documentation for this class was generated from the following file:

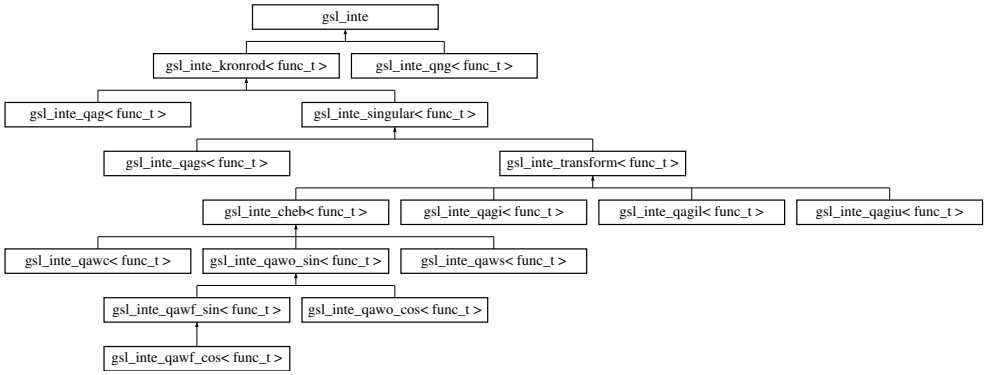
- gsl_fit.h

46.106 gsl_inte Class Reference

GSL integration base.

`#include <gsl_inte.h>`

Inheritance diagram for gsl_inte:



46.106.1 Detailed Description

This base class does not perform any actual integration, but just provides functions to be used in the integration classes based on GSL.

Definition at line 56 of file gsl_inte.h.

Protected Member Functions

- double rescale_error (double err, const double result_abs, const double result_asc)
  QUADPACK's nonlinear rescaling of the absolute-error estimate.

46.106.2 Member Function Documentation

46.106.2.1 double gsl_inte::rescale_error (double *err*, const double *result_abs*, const double *result_asc*) [inline, protected]

The values ρ_{abs} (stored in `result_abs`) and ρ_{abs} (stored in `result_asc`) are assumed to be

$$\begin{aligned} \rho_{\text{abs}} &= \int_a^b |f|\,dx, \\ \rho_{\text{asc}} &= \int_a^b |f-\mu(f)|\,dx, \qquad \mu(f) = \frac{1}{b-a}\int_a^b f\,dx, \end{aligned}$$

all of which are computed from the best (i.e., finest-grid) approximation of the integrals. The rescaled error, σ_{err}, is computed from the raw error, `err`, by

$$\sigma_{\text{err}} = \rho_{\text{asc}} \cdot \min\left\{1, \left(\frac{200|\text{err}|}{\rho_{\text{asc}}}\right)^{3/2}\right\},$$

or

$$\sigma_{\text{err}} = 50 \cdot \varepsilon_{\text{mach}} \cdot \rho_{\text{abs}},$$

whichever of the two is greater. The value $\varepsilon_{\text{mach}}$ denotes "machine epsilon." (In the case that the second value underflows, the first value is automatically accepted.)

This function is used in gsl_inte_qng and gsl_inte_kronrod::gauss_kronrod_base().

Definition at line 100 of file gsl_inte.h.

The documentation for this class was generated from the following file:

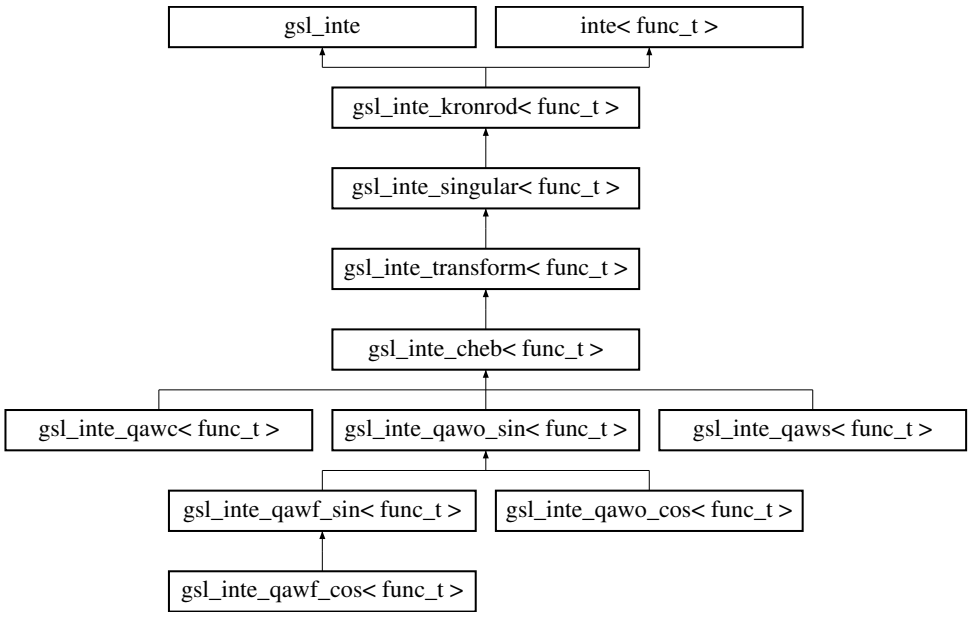
- gsl_inte.h

46.107 gsl_inte_cheb< func_t > Class Template Reference

Chebyshev integration base class (GSL)

`#include <gsl_inte_qawc.h>`

Inheritance diagram for gsl_inte_cheb< func_t >:



46.107.1 Detailed Description

template<class func_t>class gsl_inte_cheb< func_t >

This class provides the basic Chebyshev integration functions for use in the GSL-based integration classes which require them. See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Definition at line 42 of file gsl_inte_qawc.h.

Protected Member Functions

- void compute_moments (double cc, double *moment)
  Compute the Chebyshev moments.

- template<class func2_t >
  void inte_cheb_series (func2_t &f, double a, double b, double *cheb12, double *cheb24)
  Compute Chebyshev series expansion using a FFT method.

46.107.2 Member Function Documentation

46.107.2.1 template<class func_t > template<class func2_t > void gsl_inte_cheb< func_t >::inte_cheb_series (func2_t & *f*, double *a*, double *b*, double * *cheb12*, double * *cheb24*) [inline, protected]

The Chebyshev coefficients for the truncated expansions,

$$f(x) = \frac{a_0}{2} T_0(x) + \frac{a_d}{2} T_d(x) + \sum_{k=}^{d-1} a_k^{(d)} T_k(x),$$

are computed for $d = 12$ and $d = 24$ using an FFT algorithm from Tolstov62 that is adapted so that the both sets of coefficients are computed simultaneously.

Given the function specified in `f`, this function computes the 13 Chebyshev coefficients, C_k^{12} of degree 12 and 25 Chebyshev coefficients of degree 24, C_k^{24}, for the interval $[a,b]$ using a FFT method.

These coefficients are constructed to approximate the original function with

$$f = \sum_{k=1}^{13} C_k^{12} T_{k-1}(x)$$

and

$$f = \sum_{k=1}^{25} C_k^{24} T_{k-1}(x)$$

where $T_{k-1}(x)$ is the Chebyshev polynomial of degree $k-1$ evaluated at the point x.

It is assumed that memory for `cheb12` and `cheb24` has been allocated beforehand.

Originally written in QUADPACK by R. Piessens and E. de Doncker, translated into C for GSL by Brian Gough, and then rewritten for O_2scl .

Definition at line 111 of file gsl_inte_qawc.h.

The documentation for this class was generated from the following file:

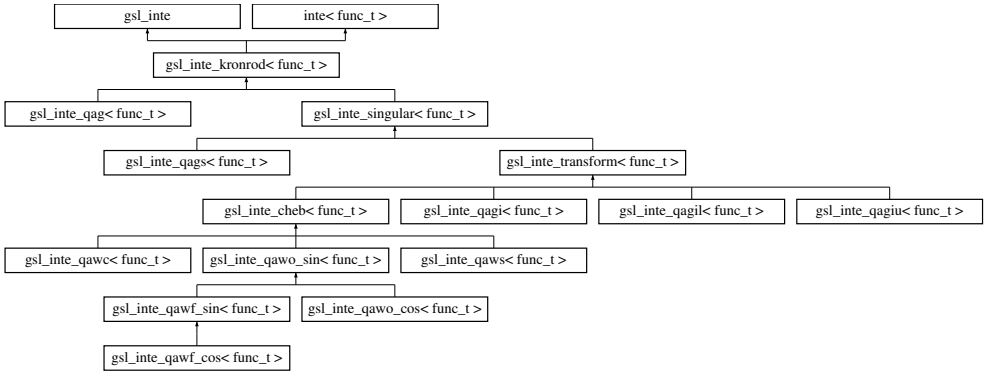
- gsl_inte_qawc.h

46.108 gsl_inte_kronrod< func_t > Class Template Reference

Basic Gauss-Kronrod integration class (GSL)

`#include <gsl_inte_kronrod.h>`

Inheritance diagram for gsl_inte_kronrod< func_t >:



46.108.1 Detailed Description

template<class func_t>class gsl_inte_kronrod< func_t >

This class provides the basic Gauss-Kronrod integration function and integration workspace for some of the GSL-based integration classes.

The main function of interest is set_rule(), which sets the integration rule for the GSL integration classes which inherit from this base class. The argument to set rule should be selected from the following list:

- 1: 15-point Gauss-Kronrod integration rule (default)
- 2: 21-point rule
- 3: 31-point rule
- 4: 41-point rule
- 5: 51-point rule
- 6: 61-point rule Any value other than 1-6 forces the error handler to be called. All classes default to the 15-point rule, except for gsl_inte_qags which defaults to the 21-point rule for singularities.

The integration coefficients for use with this class and associated children are stored in the o2scl_inte_gk_coeffs namespace.

Idea for Future Convert 'double *' objects to uvector

Definition at line 610 of file gsl_inte_kronrod.h.

Public Member Functions

- int get_rule ()
  Get the Gauss-Kronrod integration rule.
- void set_rule (int rule)
  Set the Gauss-Kronrod integration rule to be used.
- int set_limit (size_t lim)
  Set the limit for the number of subdivisions of the integration region (default 1000)
- template<class func2_t >
  void gauss_kronrod_base (func2_t &func, double a, double b, double *result, double *abserr, double *resabs, double *resasc)

  The base Gauss-Kronrod integration function template.
- virtual void gauss_kronrod (func_t &func, double a, double b, double *result, double *abserr, double *resabs, double *resasc)

  Integration wrapper for user-specified function type.

Protected Attributes

- gsl_inte_workspace * w
  The integration workspace.
- int n_gk
  Size of Gauss-Kronrod arrays.
- const double * x_gk
  Gauss-Kronrod abscissae pointer.
- const double * w_g
  Gauss weight pointer.
- const double * w_gk
  Gauss-Kronrod weight pointer.
- double * f_v1
  Scratch space.
- double * f_v2
  Scratch space.

46.108.2 Member Function Documentation

46.108.2.1 template<class func_t > int gsl_inte_kronrod< func_t >::get_rule ( ) [inline]

This returns the index of the GSL integration rule a number between 1 and 6 (inclusive)

Definition at line 659 of file gsl_inte_kronrod.h.

46.108.2.2 template<class func_t > int gsl_inte_kronrod< func_t >::set_limit (size_t *lim*) [inline]

If the value of `size` is zero, the error handler will be called.

Definition at line 741 of file gsl_inte_kronrod.h.

46.108.2.3 template<class func_t > template<class func2_t > void gsl_inte_kronrod< func_t >::gauss_kronrod_base (func2_t & *func*, double *a*, double *b*, double * *result*, double * *abserr*, double * *resabs*, double * *resasc*) [inline]

Given abcissas and weights, this performs the integration of `func` between `a` and `b`, providing a result with uncertainties.

The Gauss-Kronrod rule uses $2m+1$ abscissae $x_1,\ldots,x_{2m+1} \in (a,b)$ to estimate the integral of a function as a linear combination of values,

$$\int_a^b f\,dx \approx Q_m^{GK} f \equiv \sum_{k=1}^{2m+1} \beta_k f(x_k),$$

where the weights $\beta_1,\ldots,\beta_{2m+1}$ are intrinsic to the abscissae. The data are designed so that the even-indexed abscissae yield a coarser estimate,

$$Q_m^G f \equiv \sum_{j=1}^{m} \alpha_j f(x_{2j}),$$

and their difference $|Q_m^G f - Q_m^{GK} f|$ is the "raw" error estimate. The various quantities that the function computes are

$$\begin{aligned} \texttt{result} &= Q_m^{GK} f, \\ \texttt{resabs} &= Q_m^{GK} |f|, \\ \texttt{resasc} &= Q_m^{GK} (|f| - \mu), \quad \mu \equiv \frac{Q_m^{GK} f}{b-a}. \end{aligned}$$

The "absolute" error `abserr` is computed from the raw error value using the function gsl_inte::rescale_error.

This function is designed for use with the values given in the o2scl_inte_gk_coeffs namespace.

This function never calls the error handler.

Idea for Future This function, in principle, is an integration object in and of itself, and could be implemented separately as an object of type inte.

Definition at line 794 of file gsl_inte_kronrod.h.

The documentation for this class was generated from the following file:

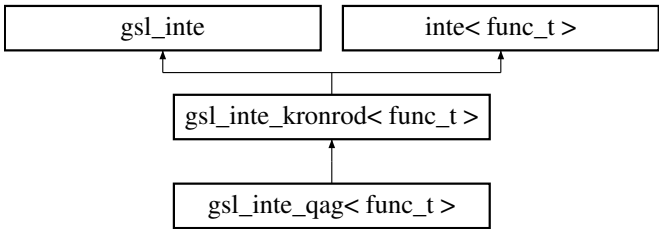
- gsl_inte_kronrod.h

46.109 gsl_inte_qag< func_t > Class Template Reference

Adaptive numerical integration of a function (without singularities) on a bounded interval (GSL)

`#include <gsl_inte_qag.h>`

Inheritance diagram for gsl_inte_qag< func_t >:



46.109.1 Detailed Description

template<class func_t = funct>class gsl_inte_qag< func_t >

Adaptive integration of a univariate function requires two main procedures: approximating the integral on a bounded interval, and estimating the approximation error. The algorithm recursively refines the interval, computing the integral and its error estimate on each subinterval, until the total error estimate over **all** subintervals falls within the user-specified tolerance. The value returned is the sum of the (approximated) integrals over all subintervals.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Idea for Future There are a few fine-tuned parameters which should be re-expressed as data members in the convergence tests.

Idea for Future Should QUADPACK parameters in round-off tests be subject to adjustments by the end user?

Idea for Future Add functions to examine the contents of the workspace to detect regions where the integrand may be problematic; possibly call these functions automatically depending on verbosity settings.

Definition at line 78 of file gsl_inte_qag.h.

Public Member Functions

- gsl_inte_qag ()
  Create an integrator with the specified rule.
- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.
- const char * type ()
  Return string denoting type ("gsl_inte_qag")

Protected Member Functions

- int qag (func_t &func, const double a, const double b, const double l_epsabs, const double l_epsrel, double *result, double *abserr)
  Perform an adaptive integration given the coefficients, and returning `result`.

46.109.2 Member Function Documentation

46.109.2.1 template<class func_t = funct> int gsl_inte_qag< func_t >::qag (func_t & *func*, const double *a*, const double *b*, const double *l_epsabs*, const double *l_epsrel*, double * *result*, double * *abserr*) [inline, protected]

Idea for Future Just move this function to integ_err().

Definition at line 107 of file gsl_inte_qag.h.

The documentation for this class was generated from the following file:

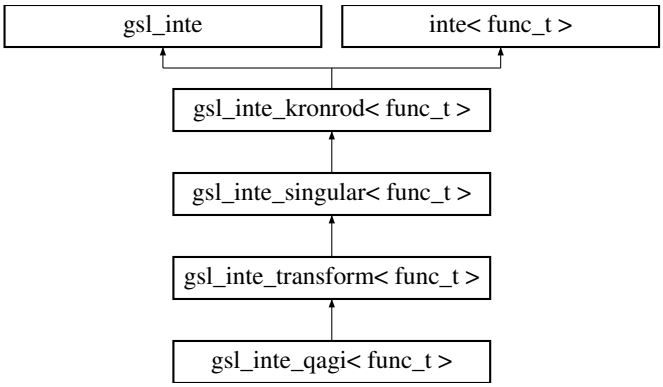
- gsl_inte_qag.h

46.110 gsl_inte_qagi< func_t > Class Template Reference

Integrate a function over the interval $(-\infty, \infty)$ (GSL)

`#include <gsl_inte_qagi.h>`

Inheritance diagram for gsl_inte_qagi< func_t >:



46.110.1 Detailed Description

template<class func_t = funct>class gsl_inte_qagi< func_t >

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Definition at line 40 of file gsl_inte_qagi.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` *from* ∞ *to* ∞ *giving result* `res` *and error* `err`.

Protected Member Functions

- virtual double transform (double t, func_t &func)
  Tranformation to $t \in (0,1]$.

46.110.2 Member Function Documentation

46.110.2.1 template<class func_t = funct> virtual int gsl_inte_qagi< func_t >::integ_err (func_t & *func*, double *a*, double *b*, double & *res*, double & *err*) [inline, virtual]

The values `a` and `b` are ignored

Implements inte< func_t >.

Definition at line 50 of file gsl_inte_qagi.h.

The documentation for this class was generated from the following file:

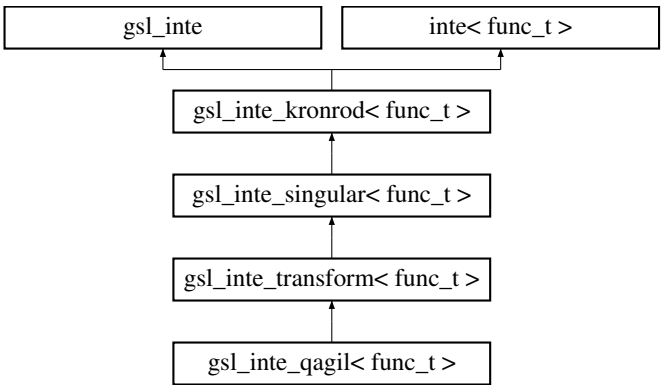
- gsl_inte_qagi.h

46.111 gsl_inte_qagil< func_t > Class Template Reference

Integrate a function over the interval $(-\infty, b]$ (GSL)

`#include <gsl_inte_qagil.h>`

Inheritance diagram for gsl_inte_qagil< func_t >:



46.111.1 Detailed Description

template<class func_t = funct>class gsl_inte_qagil< func_t >

The integral on the unbounded interval is rewritten over the semi-open interval $(0,1]$ via a variable transformation,

$$\int_{-\infty}^{b} f(x)\,dx = \int_{0}^{1} f(b-(1-t)/t)t^{-2}\,dt,$$

and the right hand side is evaluated with gsl_inte_qags.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Definition at line 48 of file gsl_inte_qagil.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` *from* $-\infty$ *to* `b` *and place the result in* `res` *and the error in* `err`.

Protected Member Functions

- virtual double transform (double t, func_t &func)
  Transform to $t \in (0,1]$.

Protected Attributes

- double upper_limit
  The upper limit.

46.111.2 Member Function Documentation

46.111.2.1 template<class func_t = funct> virtual int gsl_inte_qagil< func_t >::integ_err (func_t & *func*, double *a*, double *b*, double & *res*, double & *err*) [inline, virtual]

The value given in `a` is ignored.

Implements inte< func_t >.

Definition at line 67 of file gsl_inte_qagil.h.

The documentation for this class was generated from the following file:

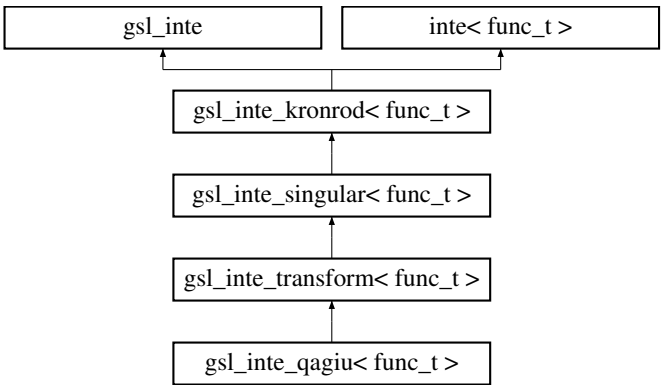
- gsl_inte_qagil.h

46.112 gsl_inte_qagiu< func_t > Class Template Reference

Integrate a function over the interval $[a,\infty)$ (GSL)

`#include <gsl_inte_qagiu.h>`

Inheritance diagram for gsl_inte_qagiu< func_t >:



46.112.1 Detailed Description

template<class func_t = funct>class gsl_inte_qagiu< func_t >

The integral on the unbounded interval is rewritten over the semi-open interval $(0,1]$ via a variable transformation,

$$\int_a^\infty f(x)\,dx = \int_0^1 f(a+(1-t)/t)t^{-2}\,dt,$$

and the right hand side is evaluated with gsl_inte_qags.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Definition at line 48 of file gsl_inte_qagiu.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate a function over the interval $[a,\infty)$ giving result `res` *and error* `err`.

Protected Member Functions

- virtual double transform (double t, func_t &func)
  Transform to $t \in (0,1]$.

Protected Attributes

- double lower_limit
  The lower limit.

46.112.2 Member Function Documentation

46.112.2.1 template<class func_t = funct> virtual int gsl_inte_qagiu< func_t >::integ_err (func_t & *func*, double *a*, double *b*, double & *res*, double & *err*) [inline, virtual]

The value `b` is ignored.

Implements inte< func_t >.

Definition at line 67 of file gsl_inte_qagiu.h.

The documentation for this class was generated from the following file:

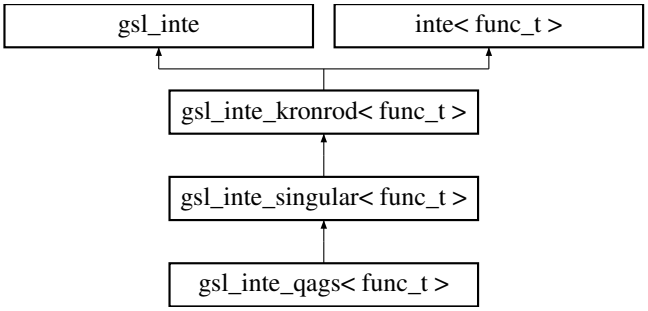
- gsl_inte_qagiu.h

46.113 gsl_inte_qags< func_t > Class Template Reference

Integrate a function with a singularity (GSL)

```
#include <gsl_inte_qags.h>
```

Inheritance diagram for gsl_inte_qags< func_t >:



46.113.1 Detailed Description

template<class func_t = funct>class gsl_inte_qags< func_t >

If a function is unbounded but has a finite integral, using the adaptive algorithm described for gsl_inte_qag to compute that integral (up to specified tolerance) will converge very slowly. The integration routine of this class remedies this by combining the adaptive algorithm with a series-acceleration method.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Definition at line 53 of file gsl_inte_qags.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` *from* `a` *to* `b` *and place the result in* `res` *and the error in* `err`.

The documentation for this class was generated from the following file:

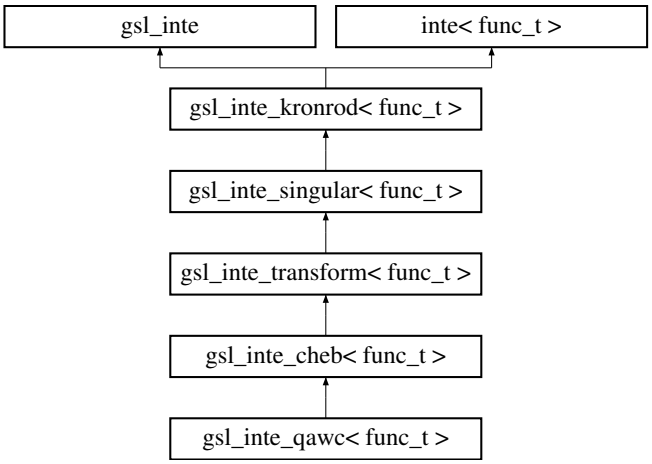
- gsl_inte_qags.h

46.114 gsl_inte_qawc< func_t > Class Template Reference

Adaptive Cauchy principal value integration (GSL)

`#include <gsl_inte_qawc.h>`

Inheritance diagram for gsl_inte_qawc< func_t >:



46.114.1 Detailed Description

template<class func_t>class gsl_inte_qawc< func_t >

The Cauchy principal value of the integral of

$$\int_a^b \frac{f(x)}{x-c}\,dx = \lim_{\varepsilon\to 0^+}\left\{\int_a^{c-\varepsilon}\frac{f(x)}{x-c}\,dx + \int_{c+\varepsilon}^b \frac{f(x)}{x-c}\,dx\right\}.$$

over (a,b), with a singularity at c, is computed. The adaptive refinement algorithm described for gsl_inte_qag is used with modifications to ensure that subdivisions do not occur at the singular point $x = c$. When a subinterval contains the point $x = c$ or is close to it, a special 25-point modified Clenshaw-Curtis rule is used to control the singularity. Further away from the singularity the algorithm uses a Gauss-Kronrod integration rule.

The location of the singularity must be specified before-hand in gsl_inte_qawc::s, and the singularity must not be at one of the endpoints. Note that when integrating a function of the form $\frac{f(x)}{(x-s)}$, the denominator $(x-s)$ must not be specified in the argument `func` to integ(). Note that this is different from how the cern_cauchy operates.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Idea for Future Make cern_cauchy and this class consistent in the way which they require the user to provide the denominator in the integrand

Definition at line 347 of file gsl_inte_qawc.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.

Data Fields

- double s
  The singularity.

Protected Member Functions

- int qawc (func_t &func, const double a, const double b, const double c, const double epsabs, const double epsrel, double *result, double *abserr)
  The full GSL integration routine called by integ_err()
- void qc25c (func_t &func, double a, double b, double c, double *result, double *abserr, int *err_reliable)
  25-point quadrature for Cauchy principal values
- virtual double transform (double t, func_t &func)
  Add the singularity to the function.
- const char * type ()
  Return string denoting type ("gsl_inte_qawc")

The documentation for this class was generated from the following file:

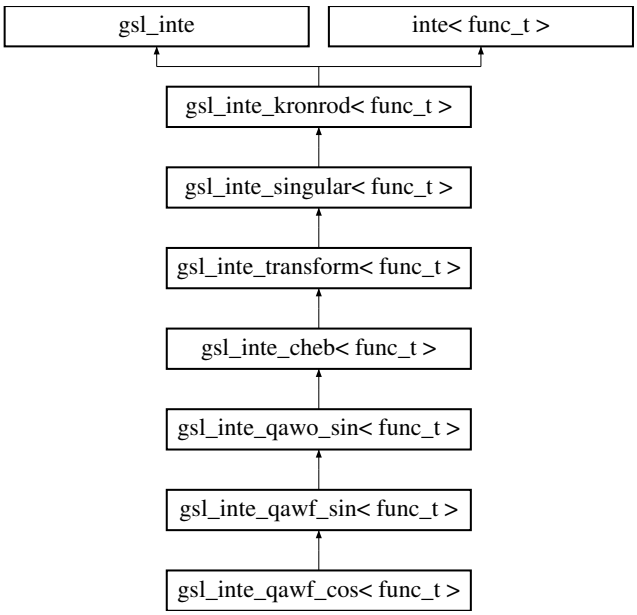
- gsl_inte_qawc.h

46.115 gsl_inte_qawf_cos< func_t > Class Template Reference

Adaptive integration a function with finite limits of integration (GSL)

`#include <gsl_inte_qawf.h>`

Inheritance diagram for gsl_inte_qawf_cos< func_t >:



46.115.1 Detailed Description

template<class func_t>class gsl_inte_qawf_cos< func_t >

The Fourier integral

$$\int_a^\infty f(x)\cos(\omega x)\,dx$$

is computed for some frequency parameter ω .

This class is exactly analogous to gsl_inte_qawf_sin . See that class documentation for more details.

Definition at line 385 of file gsl_inte_qawf.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.

Protected Member Functions

- virtual double transform (double t, func_t &func)
  Add the oscillating part to the integrand.
- const char * type ()
  Return string denoting type ("gsl_inte_qawf_cos")

The documentation for this class was generated from the following file:

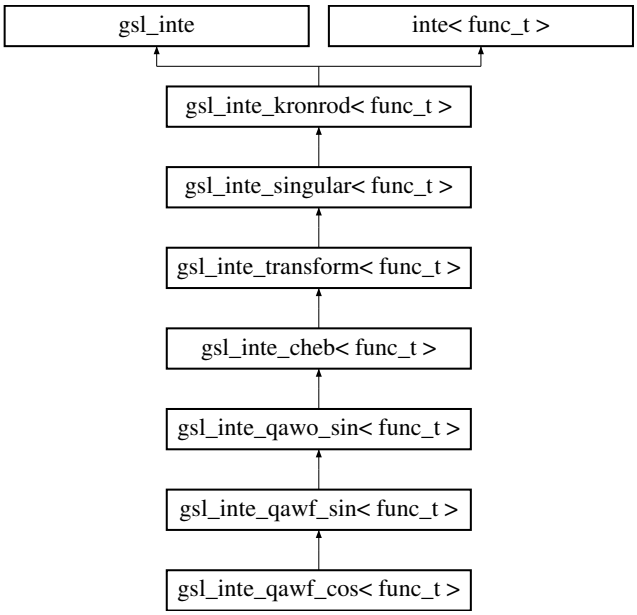
- gsl_inte_qawf.h

46.116 gsl_inte_qawf_sin< func_t > Class Template Reference

Adaptive integration for oscillatory integrals (GSL)

`#include <gsl_inte_qawf.h>`

Inheritance diagram for gsl_inte_qawf_sin< func_t >:



46.116.1 Detailed Description

template<class func_t>class gsl_inte_qawf_sin< func_t >

The Fourier integral

$$\int_a^\infty f(x)\sin(\omega x)\,dx$$

is computed for some frequency parameter ω, stored in gsl_inte_qawo_sin::omega .

The integral is computed using the same method as gsl_inte_qawo_sin and gsl_inte_qawo_cos over each of the subintervals,

$$\begin{aligned} C_1 &= [a, a+c] \\ C_2 &= [a+c, a+2c] \\ &\vdots \\ C_k &= [a+(k-1)c,\ a+kc], \end{aligned}$$

where $c = (2\text{floor}(|\omega|)+1)\pi/|\omega|$. This width is chosen to cover an odd number of periods so that the contributions from the intervals alternate in sign and are monotonically decreasing when f is positive and monotonically decreasing. The sum of this sequence of contributions is accelerated using the ε algorithm.

The algorithm uses zero for the relative tolerance inte::tol_rel and attempts to compute the integral to an overall absolute tolerance set by inte::tol_abs. The following strategy is used: on each interval C_k, the algorithm tries to achieve the tolerance

$$\text{TOL}_k = u_k \cdot \varepsilon_{\text{abs}}$$

where $u_k = (1-p)p^{k-1}$ and $p = 0.9$. The sum of the geometric series of contributions from each interval gives an overall tolerance of ε_{abs}. If the integration of a subinterval leads to difficulties then the accu racy requirement for subsequent intervals is relaxed,

$$\text{TOL}_k = u_k \cdot \max\{\varepsilon_{\text{abs}}, E_1, \ldots, E_{k-1}\}$$

where E_k is the estimated error on the interval C_k.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

When verbose output is enabled, this class outputs information from both the subintegrations performed by gsl_inte_qawo_sin and the overall integration progress in this class.

Definition at line 88 of file gsl_inte_qawf.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.

Protected Member Functions

- int qawf (func_t &func, const double a, const double epsabs, double *result, double *abserr)
  The full GSL integration routine called by integ_err()
- virtual double transform (double t, func_t &func)
  Add the oscillating part to the integrand.
- const char * type ()
  Return string denoting type ("gsl_inte_qawf_sin")

Protected Attributes

- gsl_inte_workspace * cyclew
  The integration workspace.

The documentation for this class was generated from the following file:

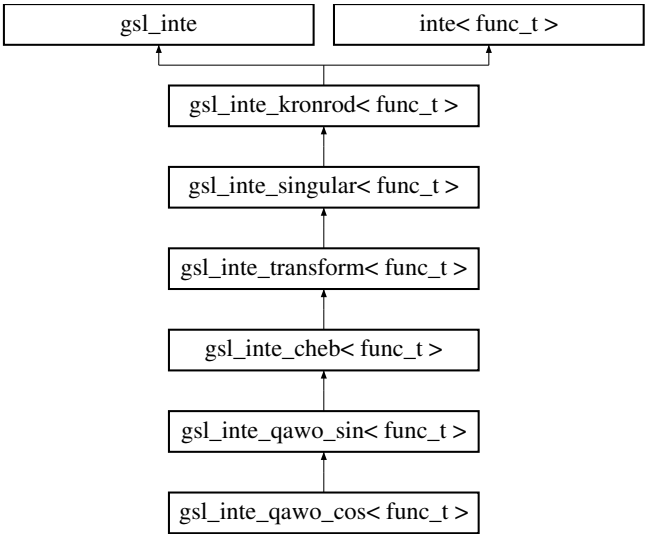
- gsl_inte_qawf.h

46.117 gsl_inte_qawo_cos< func_t > Class Template Reference

Adaptive integration a function with finite limits of integration (GSL)

`#include <gsl_inte_qawo.h>`

Inheritance diagram for gsl_inte_qawo_cos< func_t >:



46.117.1 Detailed Description

template<class func_t>class gsl_inte_qawo_cos< func_t >

The integral

$$\int_a^b f(x)\cos(\omega x)\,dx$$

is computed for some frequency parameter ω .

This class is exactly analogous to gsl_inte_qawo_sin . See that class documentation for more details.

Definition at line 645 of file gsl_inte_qawo.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.

Protected Member Functions

- virtual double transform (double t, func_t &func)
  Add the oscillating part to the integrand.
- const char * type ()
  Return string denoting type ("gsl_inte_qawo_cos")

The documentation for this class was generated from the following file:

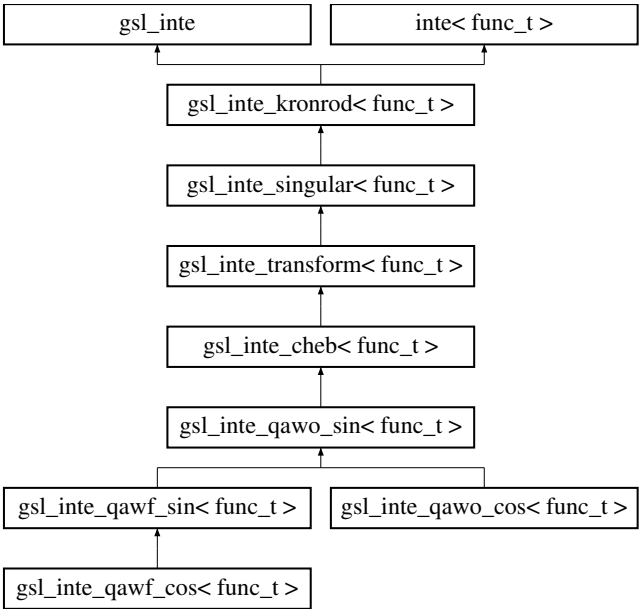
- gsl_inte_qawo.h

46.118 gsl_inte_qawo_sin< func_t > Class Template Reference

Adaptive integration for oscillatory integrals (GSL)

`#include <gsl_inte_qawo.h>`

Inheritance diagram for gsl_inte_qawo_sin< func_t >:



46.118.1 Detailed Description

template<class func_t>class gsl_inte_qawo_sin< func_t >

The integral

$$\int_a^b f(x)\sin(\omega x)\,dx$$

is computed for some frequency parameter ω, stored in gsl_inte_qawo_sin::omega .

An adaptive algorithm together with an series-acceleration method like that of gsl_inte_qags is used. Those subintervals with "large" widths $d \equiv b-a$ where $d\omega > 4$ are computed using a 25-point Clenshaw-Curtis integration rule to handle the oscillatory behavior. In order to work efficiently, the Chebyshev moments for the particular weight function W are computed in advance.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Definition at line 54 of file gsl_inte_qawo.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)
  Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.

Data Fields

- double omega
  The user-specified frequency (default 1.0)
- size_t n_levels
  The number of bisection levels (default 10)

Protected Member Functions

- int qawo (func_t &func, const double a, const double epsabs, const double epsrel, gsl_inte_workspace *loc_w, gsl_integration_qawo_table *wf, double *result, double *abserr)
  The full GSL integration routine called by integ_err()

- void qc25f (func_t &func, double a, double b, gsl_integration_qawo_table *wf, size_t level, double *result, double *abserr, double *resabs, double *resasc)
  25-point quadrature for oscillating functions
- virtual double transform (double t, func_t &func)
  Add the oscillating part to the integrand.
- const char * type ()
  Return string denoting type ("gsl_inte_qawo_sin")

Protected Attributes

- gsl_integration_qawo_table * otable
  The integration workspace.

46.118.2 Member Function Documentation

46.118.2.1 template<class func_t > int gsl_inte_qawo_sin< func_t >::qawo (func_t & *func*, const double *a*, const double *epsabs*, const double *epsrel*, gsl_inte_workspace * *loc_w*, gsl_integration_qawo_table * *wf*, double * *result*, double * *abserr*) [inline, protected]

Idea for Future Remove goto statements.

Definition at line 102 of file gsl_inte_qawo.h.

The documentation for this class was generated from the following file:

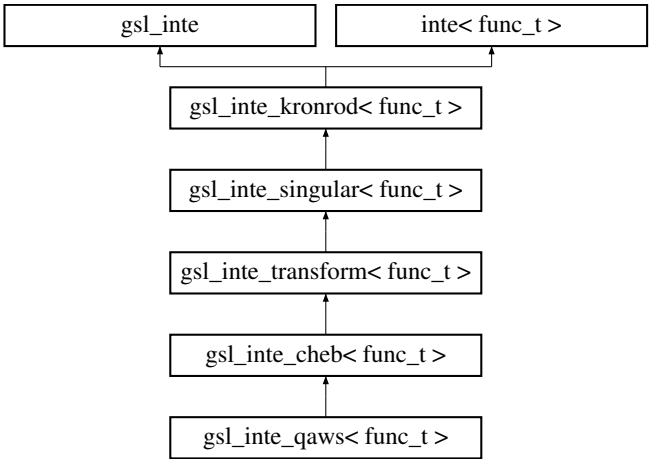
- gsl_inte_qawo.h

46.119 gsl_inte_qaws< func_t > Class Template Reference

Adaptive integration with with algebraic-logarithmic singularities at the end-points (GSL)

`#include <gsl_inte_qaws.h>`

Inheritance diagram for gsl_inte_qaws< func_t >:



46.119.1 Detailed Description

template<class func_t = funct>class gsl_inte_qaws< func_t >

This class computes the weighted integral

$$\int_a^b f(x)(x-a)^\alpha (b-x)^\beta \log^\mu (x-a) \log^\nu (b-x)\, dx$$

where the parameters of the weight function must satisfy

$$\alpha > -1, \quad \beta > -1, \quad \mu \in \{0,1\}, \quad \nu \in \{0,1\},$$

and which are set by set_weight(). Note that setting $\mu = 0$ or $\nu = 0$ removes the respective factor $\log^m u(\ldots)$ or $\log^\nu(\ldots)$ from the weight.

The adaptive refinement algorithm described for gsl_inte_qag is used. When a subinterval contains one of the endpoints, a special 25-point modified Clenshaw-Curtis rule is used to control the singularities. For subintervals which do not include the endpoints, a Gauss-Kronrod integration rule is used.

See GSL-based integration routines in the User's guide for general information about the GSL integration classes.

Definition at line 77 of file gsl_inte_qaws.h.

Public Member Functions

- gsl_inte_qaws ()
  Initialize the adptive workspace as with the constructor gsl_inte_qag::gsl_inte_qag.
- int set_weight (double u_alpha, double u_beta, int u_mu, int u_nu)
  Sets the exponents of singularites of the weight function.
- void get_weight (double &u_alpha, double &u_beta, int &u_mu, int &u_nu)
  Returns the current values (via reference) of the weight-function's parameters.
- virtual int integ_err (func_t &func, double a, double b, double &result, double &abserr)
  Integrate the function `func` on the interval (`a`, `b`) returning the `result` and error estimate `abserr`.
- const char * type ()
  Return string denoting type ("gsl_inte_qaws")

Protected Member Functions

- void initialise_qaws_table ()
  Set the array values `ri`, `rj`, `rg`, `rh` from the current values `alpha` and `beta`.
- virtual double transform (double t, func_t &func)
  Weighted integrand.
- void qc25s (func_t &func, double a, double b, double a1, double b1, double &result, double &abserr, int &err_reliable)
  Clenshaw-Curtis 25-point integration and error estimator for functions with an algebraic-logarithmic singularity at the endpoint(s).
- void compute_result (double *r, double *cheb12, double *cheb24, double &result12, double &result24)
  Compute the 13-point and 25-point approximations from the Chebyshev moments and coefficients.

Protected Attributes

- bool fn_qaws_R
  True if algebraic-logarithmic singularity is present at the right endpoint in the definition `f_trans`.
- bool fn_qaws_L
  True if algebraic-logarithmic singularity is present at the left endpoint in the definition `f_trans`.
- double left_endpoint
  Left endpoint in definition of `f_trans`.
- double right_endpoint
  Right endpoint in definition of `f_trans`.

Data from \c gsl_integration_qaws_table

- double **alpha**
- double **beta**
- int **mu**
- int **nu**
- double **ri** [25]
- double **rj** [25]
- double **rg** [25]
- double **rh** [25]

46.119.2 Constructor & Destructor Documentation

46.119.2.1 template<class func_t = funct> gsl_inte_qaws< func_t >::gsl_inte_qaws ( ) [inline]

The default paramters α, β, μ, ν of the weight function are all zero.

Definition at line 367 of file gsl_inte_qaws.h.

46.119.3 Member Function Documentation

46.119.3.1 template<class func_t = funct> void gsl_inte_qaws< func_t >::initialise_qaws_table ( ) [inline, protected]

This is the function from the GSL source code `integration/qmomo.c` that initializes `gsl_integration_qaws_table`.

Definition at line 104 of file gsl_inte_qaws.h.

46.119.3.2 template<class func_t = funct> int gsl_inte_qaws< func_t >::set_weight (double *u_alpha*, double *u_beta*, int *u_mu*, int *u_nu*) [inline]

The parameters determine the exponents of the weight function

$$W(x) = (x-a)^{\alpha}(b-x)^{\beta}\log^{\mu}(x-a)\log^{\nu}(b-x),$$

and must satsify

$$\alpha > -1, \quad \beta > -1, \quad \mu \in \{0,1\}, \quad \nu \in \{0,1\}.$$

In order for the adaptive algorithm to run quickly, a table of Chebyshev weights for the particular parameters are computed in advance.

Definition at line 388 of file gsl_inte_qaws.h.

The documentation for this class was generated from the following file:

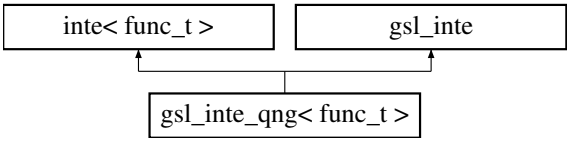
- gsl_inte_qaws.h

46.120 gsl_inte_qng< func_t > Class Template Reference

Non-adaptive integration from a to b (GSL)

`#include <gsl_inte_qng.h>`

Inheritance diagram for gsl_inte_qng< func_t >:



46.120.1 Detailed Description

template<class func_t = funct>class gsl_inte_qng< func_t >

The function integ() uses 10-point, 21-point, 43-point, and 87-point Gauss-Kronrod integration successively until the integral is returned within the accuracy specified by inte::tol_abs and inte::tol_rel. The 10-point rule is only used to estimate the error for the 21-point rule. The result of the 21-point calculation is used to estimate the error for the 43-point rule, and so forth.

The error handler is called if the 87-point integration fails to produce the desired accuracy. If inte::err_nonconv is false (the default is true), then the error handler is never called and when the desired accuracy is not obtained the result of the 87-point integration is returned along with the associated error.

The return value of the function to be integrated is ignored.

The integration fails and the error handler is called if the tolerances are too small, i.e. if either $\mathrm{tol}_{\mathrm{abs}} \leq 0$ or if $\mathrm{tol}_{\mathrm{rel}} < 50 \cdot \varepsilon_{\mathrm{double}}$ where $\varepsilon_{\mathrm{double}} \approx 1.11 \times 10^{-14}$).

The integration coefficients are stored in the o2scl_inte_qng_coeffs namespace.

Definition at line 251 of file gsl_inte_qng.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, double &res, double &err2)
  Integrate function `func` from `a` to `b` giving result `res` and error `err`.
- const char * type ()
  Return string denoting type ("gsl_inte_qng")

Data Fields

- size_t feval
  The number of function evalutions for the last integration.
- double min_rel_tol
  Minimum relative tolerance for integration (default 5×10^{-29})

46.120.2 Field Documentation

46.120.2.1 template<class func_t = funct> size_t gsl_inte_qng< func_t >::feval

Set to either 0, 21, 43, or 87, depending on the number of function evaluations that were used. This variable is zero if an error occurs before any function evaluations were performed and is never equal 10, since in the 10-point method, the 21-point result is used to estimate the error. If the function fails to achieve the desired precision, feval is 87.

Definition at line 270 of file gsl_inte_qng.h.

The documentation for this class was generated from the following file:

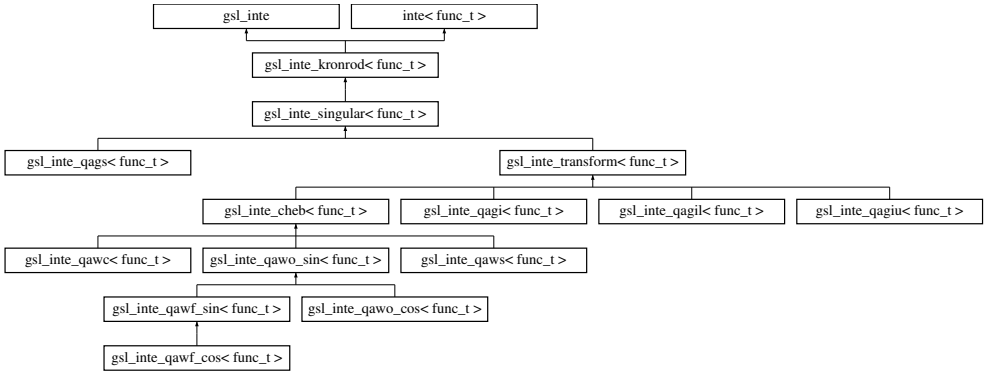
- gsl_inte_qng.h

46.121 gsl_inte_singular< func_t > Class Template Reference

Base class for integrating a function with a singularity (GSL)

`#include <gsl_inte_singular.h>`

Inheritance diagram for gsl_inte_singular< func_t >:



46.121.1 Detailed Description

template<class func_t>class gsl_inte_singular< func_t >

This class contains the extrapolation table mechanics and the base integration function for singular integrals from GSL. The casual end-user should use the classes explained in the Integration section of the User's guide.

Idea for Future Some of the functions inside this class could be moved out of header files?

Definition at line 50 of file gsl_inte_singular.h.

Data Structures

- struct extrapolation_table
  A structure for extrapolation for gsl_inte_qags.

Public Types

- typedef struct gsl_inte_singular::extrapolation_table extrap_table
  A structure for extrapolation for gsl_inte_qags.

Protected Member Functions

- void initialise_table (struct extrapolation_table *table)
  Initialize the table.
- void append_table (struct extrapolation_table *table, double y)
  Append a result to the table.
- int test_positivity (double result, double resabs)
  Test if the integrand satisfies $f = |f|$.
- void qelg (struct extrapolation_table *table, double *result, double *abserr)
  Determines the limit of a given sequence of approximations.
- int large_interval (gsl_inte_workspace *workspace)
  Determine if an interval is large.
- void reset_nrmax (gsl_inte_workspace *workspace)
  Reset workspace to work on the interval with the largest error.
- int increase_nrmax (gsl_inte_workspace *workspace)
  Increase workspace.
- int qags (func_t &func, const double a, const double b, const double l_epsabs, const double l_epsrel, double *result, double *abserr)
  Integration function.

46.121.2 Member Typedef Documentation

46.121.2.1 template<class func_t> typedef struct gsl_inte_singular::extrapolation_table gsl_inte_singular< func_t >::extrap_table

Idea for Future Move this to a new class, with qelg() as a method

46.121.3 Member Function Documentation

46.121.3.1 template<class func_t> void gsl_inte_singular< func_t >::qelg (struct extrapolation_table * *table*, double * *result*, double * *abserr*) [inline, protected]

For certain convergent series $\sum_k a_k$ whose error term $E_n = \sum_{k=n}^{\infty} a_k$ is well behaved, it is possible to find a transformation of the sequence that yields a faster converging series to the same limit. This method of extrapolation applies to some sequences of adaptive-approximation and error-estimation for numerical integration. This function implements the ϵ-algorithm (Wynn56, Piessens83) for an extrapolation table stored in `table`.

Quadpack documentation

```
c
c   list of major variables
c   -----------------------
c   e0     - the 4 elements on which the computation of a new
c   e1       element in the epsilon table is based
c   e2
c   e3                 e0
c                e3    e1    new
c                      e2
c   newelm - number of elements to be computed in the new
c            diagonal
c   error  - error = abs(e1-e0)+abs(e2-e1)+abs(new-e2)
c   result - the element in the new diagonal with least value
c            of error
c
c   machine dependent constants
c   ---------------------------
c
c   epmach is the largest relative spacing.
c   oflow is the largest positive magnitude.
c   limexp is the maximum number of elements the epsilon
c   table can contain. if this number is reached, the upper
c   diagonal of the epsilon table is deleted.
c
```

Definition at line 136 of file gsl_inte_singular.h.

46.121.3.2 template<class func_t> int gsl_inte_singular< func_t >::qags (func_t & *func*, const double *a*, const double *b*, const double *l_epsabs*, const double *l_epsrel*, double * *result*, double * *abserr*) [inline, protected]

Idea for Future Remove goto statements. Before this is done, it might be best to add some tests which fail in the various ways.

Definition at line 344 of file gsl_inte_singular.h.

The documentation for this class was generated from the following file:

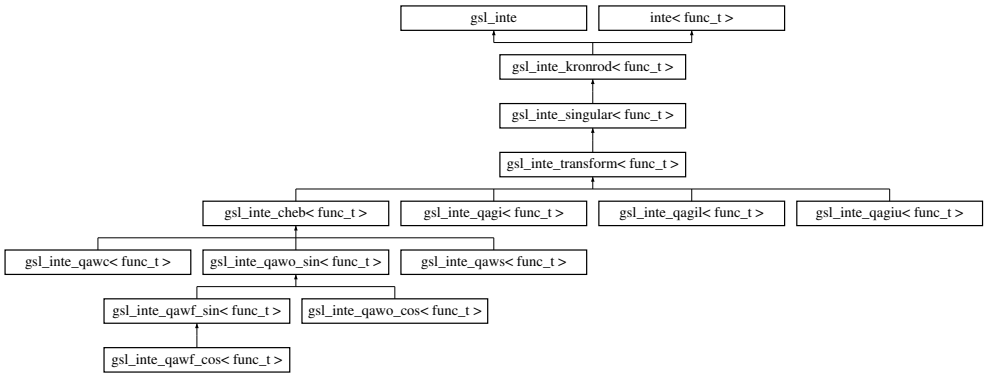
- gsl_inte_singular.h

46.122 gsl_inte_transform< func_t > Class Template Reference

Integrate a function with a singularity (GSL) [abstract base].

`#include <gsl_inte_singular.h>`

Inheritance diagram for gsl_inte_transform< func_t >:



46.122.1 Detailed Description

template<class func_t = funct>class gsl_inte_transform< func_t >

This class contains the GSL-based integration function for applying transformations to the user-defined integrand. The casual end-user should use the classes explained in the Integration section of the User's guide.

Definition at line 752 of file gsl_inte_singular.h.

Public Member Functions

- virtual double transform (double t, func_t &func)=0
  The transformation to apply to the user-supplied function.
- virtual void gauss_kronrod (func_t &func, double a, double b, double *result, double *abserr, double *resabs, double *resasc)

  Integration wrapper for internal transformed function type.

The documentation for this class was generated from the following file:

- gsl_inte_singular.h

46.123 gsl_inte_workspace Class Reference

Integration workspace for the GSL integrators.

`#include <gsl_inte_kronrod.h>`

46.123.1 Detailed Description

This is a simple rewrite of `gsl_integration_workspace` into a class.
QUADPACK workspace documentation:

```
c    parameters (meaning at output)
c       limit  - integer
c                maximum number of error estimates the list
c                can contain
c          last   - integer
c                number of error estimates currently in the list
c          maxerr - integer
```

```
c                maxerr points to the nrmax-th largest error
c                estimate currently in the list
c          ermax  - double precision
c                nrmax-th largest error estimate
c                ermax = elist(maxerr)
c          elist  - double precision
c                vector of dimension last containing
c                the error estimates
c          iord   - integer
c                vector of dimension last, the first k elements
c                of which contain pointers to the error
c                estimates, such that
c                elist(iord(1)),...,  elist(iord(k))
c                form a decreasing sequence, with
c                k = last if last.le.(limit/2+2), and
c                k = limit+1-last otherwise
c          nrmax  - integer
c      maxerr = iord(nrmax)


c     alist   - real
c               vector of dimension at least limit, the first
c                last  elements of which are the left
c               end points of the subintervals in the partition
c               of the given integration range (a,b)
c            blist   - real
c               vector of dimension at least limit, the first
c                last  elements of which are the right
c               end points of the subintervals in the partition
c               of the given integration range (a,b)
c            rlist   - real
c               vector of dimension at least limit, the first
c                last  elements of which are the
c               integral approximations on the subintervals
c            elist   - real
c               vector of dimension at least limit, the first
c                last  elements of which are the moduli of the
c               absolute error estimates on the subintervals
c            iord    - integer
c               vector of dimension at least limit, the first k
c               elements of which are pointers to the
c               error estimates over the subintervals,
c               such that elist(iord(1)), ...,
c               elist(iord(k)) form a decreasing sequence,
c               with k = last if last.le.(limit/2+2), and
c               k = limit+1-last otherwise
c            last    - integer
c               number of subintervals actually produced in the
c               subdivision process
```

Definition at line 494 of file gsl_inte_kronrod.h.

Public Member Functions

- int allocate (size_t sz)
  Allocate a workspace.
- int free ()
  Free allocated workspace memory.
- int initialise (double a, double b)
  Initialize the workspace for an integration with limits `a` and `b`.
- int set_initial_result (double result, double error)
  Update the workspace with the result and error from the first integration.
- int retrieve (double *a, double *b, double *r, double *e) const
  Retrieve the ith result from the workspace stack.

- int qpsrt ()
  Sort the workspace stack.
- int update (double a1, double b1, double area1, double error1, double a2, double b2, double area2, double error2)
  Determine which new subinterval to add to the workspace stack and perform update.
- double sum_results ()
  Add up all of the contributions to construct the final result.
- int subinterval_too_small (double a1, double a2, double b2)
  Test whether the proposed subdivision falls before floating-point precision.
- void append_interval (double a1, double b1, double area1, double error1)
  Push a new interval to the workspace stack.

Data Fields

- size_t limit
  Maximum number of subintervals allocated.
- size_t size
  Current number of subintervals being used.
- size_t nrmax
  Counter for extrapolation routine.
- size_t i
  Index of current subinterval.
- size_t maximum_level
  Depth of subdivisions reached.
- double * alist
  Left endpoints of subintervals.
- double * blist
  Right endpoints of subintervals.
- double * rlist
  Integral approximations on subintervals.
- double * elist
  Integral error estimates.
- size_t * order
  Linear ordering vector for sort routine.
- size_t * level
  Numbers of subdivisions made.

46.123.2 Member Function Documentation

46.123.2.1 int gsl_inte_workspace::retrieve (double * *a*, double * *b*, double * *r*, double * *e*) const

The workspace variable `i` is used to specify which interval is requested.

46.123.2.2 int gsl_inte_workspace::qpsrt ( )

This routine maintains the descending ordering in the list of the local error estimated resulting from the interval subdivision process. at each call two error estimates are inserted using the sequential search method, top-down for the largest error estimate and bottom-up for the smallest error estimate.

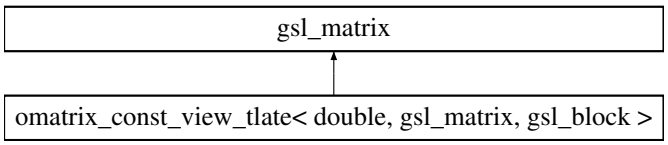
Originally written in QUADPACK by R. Piessens and E. de Doncker, translated into C for GSL by Brian Gough, and then rewritten for O_2scl .

The documentation for this class was generated from the following file:

- gsl_inte_kronrod.h

46.124 gsl_matrix Class Reference

Inheritance diagram for gsl_matrix:



The documentation for this class was generated from the following file:

- omatrix_tlate.h

46.125 gsl_matrix_int Class Reference

Inherited by omatrix_const_view_tlate< int, gsl_matrix_int, gsl_block_int >.

The documentation for this class was generated from the following file:

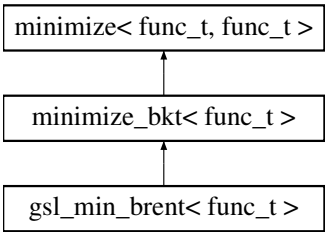
- omatrix_tlate.h

46.126 gsl_min_brent< func_t > Class Template Reference

One-dimensional minimization using Brent's method (GSL)

`#include <gsl_min_brent.h>`

Inheritance diagram for gsl_min_brent< func_t >:



46.126.1 Detailed Description

template<class func_t = funct>class gsl_min_brent< func_t >

The minimization in the function min_bkt() is complete when the bracketed interval is smaller than tol = tol_abs + tol_rel · min, where min = min(|lower|, |upper|).

Note that this algorithm requires that the initial guess already brackets the minimum, i.e. $x_1 < x_2 < x_3$, $f(x_1) > f(x_2)$ and $f(x_3) > f(x_2)$. This is different from cern_minimize, where the initial value of the first parameter to cern_minimize::min_bkt() is ignored.

The set functions throw an error if the initial bracket is not correctly specified. The function iterate() never calls the error handler. The function min_bkt() calls the error handler if the tolerances are negative or if the number of iterations is insufficient to give the specified tolerance.

Setting minimize::err_nonconv to false will force min_bkt() not to call the error handler when the number of iterations is insufficient.

Note that if there are more than 1 local minima in the specified interval, there is no guarantee that this method will find the global minimum.

See also gsl_root_brent for a similar algorithm applied as a solver rather than a minimizer.

Note

There was a bug in this minimizer which was fixed for GSL-1.11 which has also been fixed here.

Definition at line 87 of file gsl_min_brent.h.

Public Member Functions

- int set (func_t &func, double xmin, double lower, double upper)
  Set the function and the initial bracketing interval.
- int set_with_values (func_t &func, double xmin, double fmin, double lower, double fl, double upper, double fu)
  Set the function, the initial bracketing interval, and the corresponding function values.
- int iterate ()
  Perform an iteration.
- virtual int min_bkt (double &x2, double x1, double x3, double &fmin, func_t &func)
  Calculate the minimum `fmin` of `func` with `x2` bracketed between `x1` and `x3`.
- virtual const char * type ()
  Return string denoting type ("gsl_min_brent")

Data Fields

- double x_minimum
  Location of minimum.
- double x_lower
  Lower bound.
- double x_upper
  Upper bound.
- double f_minimum
  Minimum value.
- double f_lower
  Value at lower bound.
- double f_upper
  Value at upper bound.

Protected Member Functions

- int compute_f_values (func_t &func, double xminimum, double *fminimum, double xlower, double *flower, double xupper, double *fupper)
  Compute the function values at the various points.

Protected Attributes

- func_t * uf
  The function.

Temporary storage

- double **d**
- double **e**
- double **v**
- double **w**
- double **f_v**
- double **f_w**

46.126.2 Member Function Documentation

46.126.2.1 template<class func_t = funct> int gsl_min_brent< func_t >::iterate ( ) [inline]

Idea for Future It looks like x_left and x_right can be removed. Also, it would be great to replace the one-letter variable names with something more meaningful.

Definition at line 198 of file gsl_min_brent.h.

46.126.2.2 template<class func_t = funct> virtual int gsl_min_brent< func_t >::min_bkt (double & *x2*, double *x1*, double *x3*, double & *fmin*, func_t & *func*) [inline, virtual]

Note that this algorithm requires that the initial guess already brackets the minimum, i.e. $x_1 < x_2 < x_3$, $f(x_1) > f(x_2)$ and $f(x_3) > f(x_2)$. This is different from cern_minimize, where the initial value of the first parameter to cern_minimize::min_bkt() is ignored.

Implements minimize_bkt< func_t >.

Definition at line 332 of file gsl_min_brent.h.

The documentation for this class was generated from the following file:

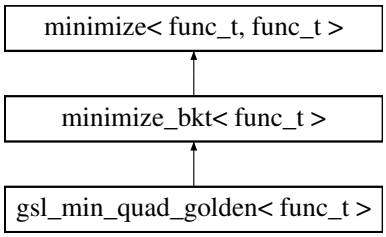
- gsl_min_brent.h

46.127 gsl_min_quad_golden< func_t > Class Template Reference

Minimization of a function using the safeguarded step-length algorithm of Gill and Murray [GSL].

`#include <gsl_min_quad_golden.h>`

Inheritance diagram for gsl_min_quad_golden< func_t >:



46.127.1 Detailed Description

template<class func_t = funct>class gsl_min_quad_golden< func_t >

This class is unfinished.

Documentation from GSL:

```
This algorithm performs univariate minimization (i.e., line
search). It requires only objective function values g(x) to
compute the minimum. The algorithm maintains an interval of
uncertainty [a,b] and a point x in the interval [a,b] such that
a < x < b, and g(a) > g(x) and g(x) < g(b). The algorithm also
maintains the three points with the smallest objective values x,
v and w such that g(x) < g(v) < g(w). The algorithm terminates
when max( x-a, b-x ) < 2(r |x| + t) where r and t are small
positive reals. At a given iteration, the algorithm first fits a
quadratic through the three points (x, g(x)), (v, g(v)) and (w,
g(w)) and computes the location of the minimum u of the
resulting quadratic. If u is in the interval [a,b] then g(u) is
```

```
computed. If u is not in the interval [a,b], and either v < x
and w < x, or v > x and w > x (i.e., the quadratic is
extrapolating), then a point u' is computed using a safeguarding
procedure and g(u') is computed. If u is not in the interval
[a,b], and the quadratic is not extrapolating, then a point u''
is computed using approximate golden section and g(u'') is
computed. After evaluating g() at the appropriate new point, a,
b, x, v, and w are updated and the next iteration is performed.
The algorithm is based on work presented in the following
references.

Algorithms for Minimization without derivatives
Richard Brent
Prentice-Hall Inc., Englewood Cliffs, NJ, 1973

Safeguarded Steplength Algorithms for Optimization using Descent Methods
Philip E. Gill and Walter Murray
Division of Numerical Analysis and Computing
National Physical Laboratory, Teddington, United Kingdom
NPL Report NAC 37, August 1974
```

Idea for Future Take common elements of this and gsl_min_brent and move to a generic GSL minimizer type?

Definition at line 100 of file gsl_min_quad_golden.h.

Public Member Functions

- int set (func_t &func, double xmin, double lower, double upper)
  Set the function and the initial bracketing interval.
- int set_with_values (func_t &func, double xmin, double fmin, double lower, double fl, double upper, double fu)
  Set the function, the initial bracketing interval, and the corresponding function values.
- int iterate ()
  Perform an iteration.
- virtual int min_bkt (double &x2, double x1, double x3, double &fmin, func_t &func)
  Calculate the minimum `fmin` of `func` with `x2` bracketed between `x1` and `x3`.
- virtual const char * type ()
  Return string denoting type ("gsl_min_quad_golden")

Data Fields

- double x_minimum
  Location of minimum.
- double x_lower
  Lower bound.
- double x_upper
  Upper bound.
- double f_minimum
  Minimum value.
- double f_lower
  Value at lower bound.
- double f_upper
  Value at upper bound.

Protected Member Functions

- int compute_f_values (func_t &func, double xminimum, double *fminimum, double xlower, double *flower, double xupper, double *fupper)
  Compute the function values at the various points.

Protected Attributes

- func_t * uf
  The function.
- double **x_prev_small**
- double **f_prev_small**
- double **x_small**
- double **f_small**
- double **step_size**
- double **stored_step**
- double **prev_stored_step**
- size_t **num_iter**
- double **rel_err_val**
- double **abs_err_val**
- double **golden_mean**
- double **golden_ratio**

46.127.2 Member Function Documentation

46.127.2.1 template<class func_t = funct> virtual int gsl_min_quad_golden< func_t >::min_bkt (double & *x2*, double *x1*, double *x3*, double & *fmin*, func_t & *func*) [inline, virtual]

Note that this algorithm requires that the initial guess already brackets the minimum, i.e. $x_1 < x_2 < x_3$, $f(x_1) > f(x_2)$ and $f(x_3) > f(x_2)$. This is different from cern_minimize, where the initial value of the first parameter to cern_minimize::min_bkt() is ignored.

Implements minimize_bkt< func_t >.

Definition at line 399 of file gsl_min_quad_golden.h.

The documentation for this class was generated from the following file:

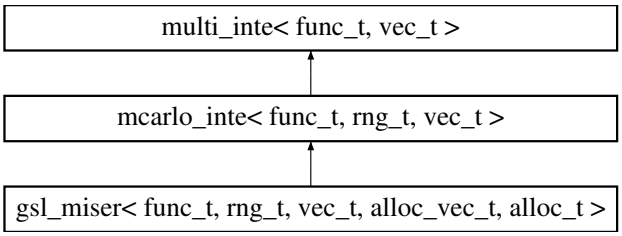
- gsl_min_quad_golden.h

46.128 gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Multidimensional integration using the MISER Monte Carlo algorithm (GSL)

```
#include <gsl_miser.h>
```

Inheritance diagram for gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >:



46.128.1 Detailed Description

template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

This class uses recursive stratified sampling to estimate the value of an integral over a hypercubic region.

By default the minimum number of calls to estimate the variance is 16 times the number of dimensions. This ratio is stored in calls_per_dim. By default the minimum number of calls per bisection is 32 times calls_per_dim times the number of dimensions. This ratio is stored in bisection_ratio. These ratios are employed by minteg_err().

Alternatively, the user can directly set these minimums by set_min_calls() and set_min_calls_per_bisection() and use miser_minteg_err() which ignores calls_per_dim and bisection_ratio.

If mcarlo_inte::verbose is greater than zero, then the status of the integration is output at every level of bisection less than n_levels_out. If it is greater than 1, then the boundaries of the current region are also output. Finally, if it is greater than 2, a keypress is required after each output.

Based on Press90 .

Definition at line 88 of file gsl_miser.h.

Public Member Functions

- int set_min_calls (size_t &mc)
  Minimum number of calls to estimate the variance.
- int set_min_calls_per_bisection (size_t &mcb)
  Minimum number of calls required to proceed with bisection.
- virtual int allocate (size_t ldim)
  Allocate memory.
- virtual int free ()
  Free allocated memory.
- virtual int miser_minteg_err (func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, size_t calls, size_t level, double &res, double &err)
  Integrate function `func` *over the hypercube from* $x_i = \mathrm{xl}_i$ *to* $x_i = \mathrm{xu}_i$ *for* $0 < i <$ *ndim-1.*
- virtual int minteg_err (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, double &res, double &err)
  Integrate function `func` *from x=a to x=b.*
- virtual double minteg (func_t &func, size_t ndim, const vec_t &a, const vec_t &b)
  Integrate function `func` *over the hypercube from* $x_i = a_i$ *to* $x_i = b_i$ *for* $0 < i <$ *ndim-1.*
- virtual const char * type ()
  Return string denoting type ("gsl_miser")

Data Fields

- size_t calls_per_dim
  Number of calls per dimension (default 16)
- size_t bisection_ratio
  Factor to set min_calls_per_bisection (default 32)
- double dither
  Introduce random variation into bisection (default 0.0)
- double estimate_frac
  Specify fraction of function calls for estimating variance (default 0.1)
- double alpha
  How estimated variances for two sub-regions are combined (default 2.0)
- size_t n_levels_out
  The number of recursive levels to output when verbose is greater than zero (default 5)

Protected Member Functions

- virtual int estimate_corrmc (func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, size_t calls, double &res, double &err, const uvector &lxmid, uvector &lsigma_l, uvector &lsigma_r)
  Estimate the variance.

Protected Attributes

- size_t min_calls
  Minimum number of calls to estimate the variance.
- size_t min_calls_per_bisection
  Minimum number of calls required to proceed with bisection.
- size_t dim
  The number of dimensions of memory allocated.
- alloc_t ao
  Memory allocator.
- alloc_vec_t x
  The most recent integration point.

Arrays which contain a value for each dimension

- uvector xmid
  The current midpoint.
- uvector sigma_l
  The left variance.
- uvector sigma_r
  The right variance.
- uvector fmax_l
  The maximum function value in the left half.
- uvector fmax_r
  The maximum function value in the right half.
- uvector fmin_l
  The minimum function value in the left half.
- uvector fmin_r
  The minimum function value in the right half.
- uvector fsum_l
  The sum in the left half.
- uvector fsum_r
  The sum in the right half.
- uvector fsum2_l
  The sum of the squares in the left half.
- uvector fsum2_r
  The sum of the squares in the right half.
- uvector_size_t hits_l
  The number of evaluation points in the left half.
- uvector_size_t hits_r
  The number of evaluation points in the right half.

46.128.2 Member Function Documentation

46.128.2.1 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::set_min_calls (size_t & *mc*) [inline]

This is set by minteg() and minteg_err() to be calls_per_dim times the number of dimensions in the problem. The default value of calls_per_dim is 16 (which is the GSL default).

From GSL documentation:

```
This parameter specifies the minimum number of function calls
required for each estimate of the variance. If the number of
function calls allocated to the estimate using ESTIMATE_FRAC falls
below MIN_CALLS then MIN_CALLS are used instead.  This ensures
that each estimate maintains a reasonable level of accuracy.
```

Definition at line 167 of file gsl_miser.h.

46.128.2.2 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::set_min_calls_per_bisection (size_t & *mcb*) [inline]

This is set by minteg() and minteg_err() to be calls_per_dim times bisection_ratio times the number of dimensions in the problem. The default values give 512 times the number of dimensions (also the GSL default).

From GSL documentation:

```
This parameter specifies the minimum number of function calls
required to proceed with a bisection step.  When a recursive step
has fewer calls available than MIN_CALLS_PER_BISECTION it performs
a plain Monte Carlo estimate of the current sub-region and
terminates its branch of the recursion.
```

Definition at line 188 of file gsl_miser.h.

46.128.2.3 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::estimate_corrmc (func_t & *func,* size_t *ndim,* const vec_t & *xl,* const vec_t & *xu,* size_t *calls,* double & *res,* double & *err,* const uvector & *lxmid,* uvector & *lsigma_l,* uvector & *lsigma_r*) [inline, protected, virtual]

Idea for Future Remove the reference to GSL_POSINF and replace with a function parameter.

Definition at line 246 of file gsl_miser.h.

46.128.2.4 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::miser_minteg_err (func_t & *func,* size_t *ndim,* const vec_t & *xl,* const vec_t & *xu,* size_t *calls,* size_t *level,* double & *res,* double & *err*) [inline, virtual]

Note

The values of min_calls and min_calls_per_bisection should be set before calling this function. The default values if not set are 100 and 3000 respectively, which correspond to the GSL default setting for a 6 dimensional problem.

Definition at line 428 of file gsl_miser.h.

46.128.2.5 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::minteg_err (func_t & *func,* size_t *ndim,* const vec_t & *a,* const vec_t & *b,* double & *res,* double & *err*) [inline, virtual]

This function is just a wrapper to miser_minteg_err() which allocates the memory if necessary, sets `min_calls` and `min_calls_per_bisection`, calls miser_minteg_err(), and then frees the previously allocated memory.

Implements multi_inte< func_t, vec_t >.

Definition at line 693 of file gsl_miser.h.

46.128.2.6 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual double gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::minteg (func_t & *func,* size_t *ndim,* const vec_t & *a,* const vec_t & *b*) [inline, virtual]

This function is just a wrapper to minteg_err() which allocates the memory, sets min_calls and min_calls_per_bisection, calls miser_minteg_err(), and then frees the previously allocated memory.

Reimplemented from multi_inte< func_t, vec_t >.

Definition at line 716 of file gsl_miser.h.

46.128.3 Field Documentation

46.128.3.1 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::dither

From GSL documentation:

```
This parameter introduces a random fractional variation of size
DITHER into each bisection, which can be used to break the
symmetry of integrands which are concentrated near the exact
center of the hypercubic integration region.  The default value of
dither is zero, so no variation is introduced. If needed, a
typical value of DITHER is 0.1.
```

Definition at line 113 of file gsl_miser.h.

46.128.3.2 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::estimate_frac

From GSL documentation:

```
This parameter specifies the fraction of the currently available
number of function calls which are allocated to estimating the
variance at each recursive step. The default value is 0.1.
```

Definition at line 125 of file gsl_miser.h.

46.128.3.3 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::alpha

The error handler will be called if this is less than zero.

From GSL documentation:

```
This parameter controls how the estimated variances for the two
sub-regions of a bisection are combined when allocating points.
With recursive sampling the overall variance should scale better
than 1/N, since the values from the sub-regions will be obtained
using a procedure which explicitly minimizes their variance.  To
accommodate this behavior the MISER algorithm allows the total
variance to depend on a scaling parameter \alpha,

\Var(f) = {\sigma_a \over N_a^\alpha} + {\sigma_b \over N_b^\alpha}.

The authors of the original paper describing MISER recommend the
value \alpha = 2 as a good choice, obtained from numerical
experiments, and this is used as the default value in this
implementation.
```

Definition at line 150 of file gsl_miser.h.

The documentation for this class was generated from the following file:

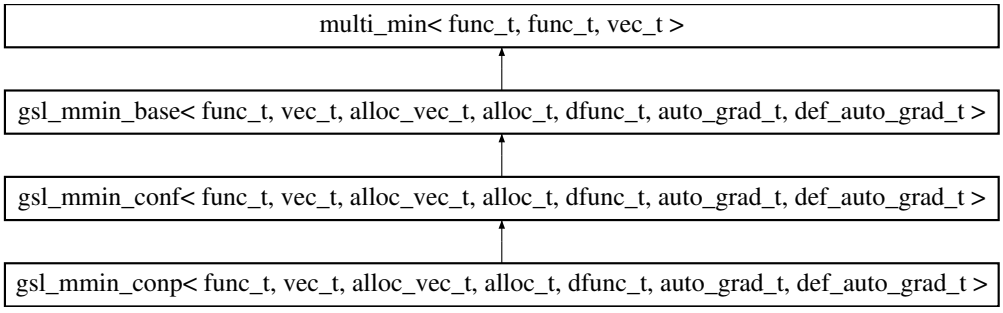
- gsl_miser.h

46.129 gsl_mmin_base< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t > Class Template Reference

Base minimization routines for gsl_mmin_conf and gsl_mmin_conp.

`#include <gsl_mmin_conf.h>`

Inheritance diagram for gsl_mmin_base< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >:



46.129.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>, class def_auto_grad_t = simple_grad<func_t,ovector_base>>class gsl_mmin_base< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

This class is used by the gsl_mmin_conf and gsl_mmin_conp minimizers to perform the line minimization along a specified direction. It is not intended for a casual end-user.

Default template arguments

- `func_t` - multi_funct< ovector_base >
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc
- `dfunc_t` - mm_funct< ovector_base >
- `auto_grad_t` - gradient<func_t, ovector_base >
- `def_auto_grad_t` - simple_grad<func_t,ovector_base >

Definition at line 61 of file gsl_mmin_conf.h.

Public Member Functions

- int base_set (func_t &ufunc, auto_grad_t &u_def_grad)
  Set the function.
- int base_set_de (func_t &ufunc, dfunc_t &udfunc)
  Set the function and the gradient .
- int base_allocate (size_t nn)
  Allocate memory.
- int base_free ()
  Clear allocated memory.

Data Fields

- double deriv_h
  Stepsize for finite-differencing (default 10^{-4} *)*
- int nmaxiter
  Maximum iterations for line minimization (default 10)
- def_auto_grad_t def_grad
  Default automatic gradient object.

Protected Member Functions

- void take_step (const vec_t &x, const vec_t &px, double stepx, double lambda, vec_t &x1x, vec_t &dx)
  Take a step.
- void intermediate_point (const vec_t &x, const vec_t &px, double lambda, double pg, double stepa, double stepc, double fa, double fc, vec_t &x1x, vec_t &dx, vec_t &gradient, double *stepx, double *f)
  Line minimization.
- void minimize (const vec_t &x, const vec_t &xp, double lambda, double stepa, double stepb, double stepc, double fa, double fb, double fc, double xtol, vec_t &x1x, vec_t &dx1x, vec_t &x2x, vec_t &dx2x, vec_t &gradient, double *xstep, double *f, double *gnorm_u)
  Perform the minimization.

Protected Attributes

- func_t * func
  User-specified function.
- dfunc_t * grad
  User-specified gradient.
- auto_grad_t * agrad
  Automatic gradient object.
- bool grad_given
  If true, a gradient has been specified.
- size_t dim
  Memory size.
- alloc_t ao
  Memory allocation.

46.129.2 Member Function Documentation

46.129.2.1 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>, class def_auto_grad_t = simple_grad<func_t,ovector_base>> void gsl_mmin_base< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >::intermediate_point (const vec_t & *x*, const vec_t & *px*, double *lambda*, double *pg*, double *stepa*, double *stepc*, double *fa*, double *fc*, vec_t & *x1x*, vec_t & *dx*, vec_t & *gradient*, double * *stepx*, double * *f*) [inline, protected]

Do a line minimisation in the region (xa,fa) (xc,fc) to find an intermediate (xb,fb) satisifying fa > fb < fc. Choose an initial xb based on parabolic interpolation.

Definition at line 105 of file gsl_mmin_conf.h.

46.129.2.2 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>, class def_auto_grad_t = simple_grad<func_t,ovector_base>> void gsl_mmin_base< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >::minimize (const vec_t & *x*, const vec_t & *xp*, double *lambda*, double *stepa*, double *stepb*, double *stepc*, double *fa*, double *fb*, double *fc*, double *xtol*, vec_t & *x1x*, vec_t & *dx1x*, vec_t & *x2x*, vec_t & *dx2x*, vec_t & *gradient*, double * *xstep*, double * *f*, double * *gnorm_u*) [inline, protected]

Starting at (x0, f0) move along the direction p to find a minimum f(x0 - lambda * p), returning the new point x1 = x0-lambda*p, f1=f(x1) and g1 = grad(f) at x1.

Definition at line 178 of file gsl_mmin_conf.h.

The documentation for this class was generated from the following file:

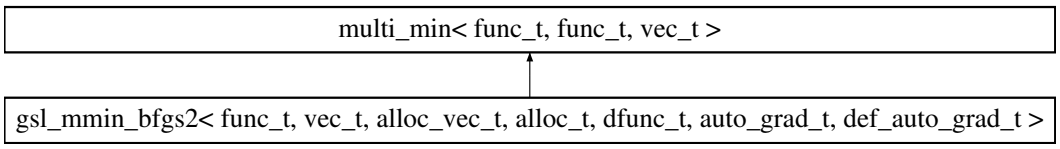
- gsl_mmin_conf.h

46.130 gsl_mmin_bfgs2< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t > Class Template Reference

Multidimensional minimization by the BFGS algorithm (GSL)

`#include <gsl_mmin_bfgs2.h>`

Inheritance diagram for gsl_mmin_bfgs2< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >:



46.130.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>, class def_auto_grad_t = simple_grad<func_t,ovector_base>>class gsl_mmin_bfgs2< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

The functions mmin() and mmin_de() minimize a given function until the gradient is smaller than the value of multi_min::tol_rel (which defaults to 10^{-4}).

See an example for the usage of this class in Multidimensional minimizer .

This class includes the optimizations from the GSL minimizer `vector_bfgs2`.

Default template arguments

- `func_t` - multi_funct< ovector_base >
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc
- `dfunc_t` - mm_funct< ovector_base >
- `auto_grad_t` - gradient<func_t, ovector_base >
- `def_auto_grad_t` - simple_grad<func_t,ovector_base >

Todo While BFGS does well in the `ex_mmin` example with the initial guess of $(1,0,7\pi)$ it seems to converge more poorly for the spring function than the other minimizers with other initial guesses, and I think this will happen in the GSL versions too. I need to examine this more closely with some code designed to clearly show this.

Definition at line 375 of file gsl_mmin_bfgs2.h.

Public Member Functions

- virtual int iterate ()
  Perform an iteration.
- virtual const char * type ()
  Return string denoting type("gsl_mmin_bfgs2")
- virtual int allocate (size_t n)
  Allocate the memory.
- virtual int free ()

Free the allocated memory.

- int restart ()
  Reset the minimizer to use the current point as a new starting point.
- virtual int set (vec_t &x, double u_step_size, double tol_u, func_t &ufunc)
  Set the function and initial guess.
- virtual int set_de (vec_t &x, double u_step_size, double tol_u, func_t &ufunc, dfunc_t &udfunc)
  Set the function, the gradient, and the initial guess.
- virtual int mmin (size_t nn, vec_t &xx, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nn`.
- virtual int mmin_de (size_t nn, vec_t &xx, double &fmin, func_t &ufunc, dfunc_t &udfunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nn`.

Data Fields

- double step_size
  The size of the first trial step (default 0.01)
- double lmin_tol
  The tolerance for the 1-dimensional minimizer.
- def_auto_grad_t def_grad
  Default automatic gradient object.

Protected Attributes

- gsl_mmin_linmin2 lm
  The line minimizer.
- size_t dim
  Memory size.
- alloc_t ao
  Memory allocation.
- auto_grad_t * agrad
  Automatic gradient object.

The original variables from the GSL state structure

- int **iter**
- double **step**
- double **g0norm**
- double **pnorm**
- double **delta_f**
- double **fp0**
- alloc_vec_t **x0**
- alloc_vec_t **g0**
- alloc_vec_t **p**
- alloc_vec_t **dx0**
- alloc_vec_t **dg0**
- gsl_mmin_wrapper< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t > **wrap**
- double **rho**
- double **sigma**
- double **tau1**
- double **tau2**
- double **tau3**
- int **order**

Store the arguments to set() so we can use them for iterate()

- vec_t * **st_x**
- alloc_vec_t **st_dx**
- alloc_vec_t **st_grad**
- double **st_f**

The documentation for this class was generated from the following file:

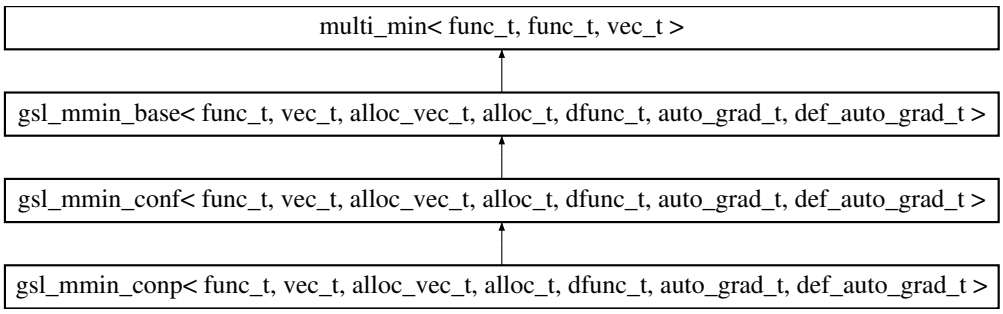
- gsl_mmin_bfgs2.h

46.131 gsl_mmin_conf< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t > Class Template Reference

Multidimensional minimization by the Fletcher-Reeves conjugate gradient algorithm (GSL)

`#include <gsl_mmin_conf.h>`

Inheritance diagram for gsl_mmin_conf< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >:



46.131.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>, class def_auto_grad_t = simple_grad<func_t,ovector_base>>class gsl_mmin_conf< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

This class performs multidimensional minimization by the Fletcher-Reeves conjugate gradient algorithm (GSL). The functions mmin() and mmin_de() minimize a given function until the gradient is smaller than the value of multi_min::tol_rel (which defaults to 10^{-4}).

This class has a high-level interface using mmin() or mmin_de() which automatically performs the memory allocation and minimization, or a GSL-like interface using allocate(), free(), interate() and set() or set_simplex().

See an example for the usage of this class in Multidimensional minimizer .

Default template arguments

- `func_t` - multi_funct< ovector_base >
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc
- `dfunc_t` - grad_funct< ovector_base >
- `auto_grad_t` - gradient<func_t, ovector_base >
- `def_auto_grad_t` - simple_grad<func_t, ovector_base >

Note that the state variable `max_iter` has not been included here, because it was not really used in the original GSL code for these minimizers.

Definition at line 401 of file gsl_mmin_conf.h.

Public Member Functions

- virtual const char * type ()
  Return string denoting type("gsl_mmin_conf")

GSL-like lower level interface

- virtual int iterate ()
  Perform an iteration.
- virtual int allocate (size_t n)
  Allocate the memory.
- virtual int free ()
  Free the allocated memory.
- int restart ()
  Reset the minimizer to use the current point as a new starting point.
- virtual int set (vec_t &x, double u_step_size, double tol_u, func_t &ufunc)
  Set the function and initial guess.
- virtual int set_de (vec_t &x, double u_step_size, double tol_u, func_t &ufunc, dfunc_t &udfunc)
  Set the function and initial guess.

Basic usage

- virtual int mmin (size_t nn, vec_t &xx, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`.
- virtual int mmin_de (size_t nn, vec_t &xx, double &fmin, func_t &ufunc, dfunc_t &udfunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`.

Data Fields

- double lmin_tol
  Tolerance for the line minimization (default 10^{-4})
- double step_size
  Size of the initial step (default 0.01)

Protected Attributes

The original variables from the GSL state structure

- int iter
  Iteration number.
- double step
  Stepsize.
- double tol
  Tolerance.
- alloc_vec_t x1
  Desc.
- alloc_vec_t dx1
  Desc.
- alloc_vec_t x2
  Desc.
- double pnorm
  Desc.
- alloc_vec_t p
  Desc.
- double g0norm
  Desc.
- alloc_vec_t g0
  Desc.

Store the arguments to set() so we can use them for iterate()

- alloc_vec_t ugx
  Proposed minimum.
- alloc_vec_t ugg
  Gradient.
- alloc_vec_t udx
  Proposed step.
- double it_min
  Desc.

46.131.2 Member Function Documentation

46.131.2.1 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>, class def_auto_grad_t = simple_grad<func_t,ovector_base>> virtual int gsl_mmin_conf< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >::set (vec_t & *x*, double *u_step_size*, double *tol_u*, func_t & *ufunc*) [inline, virtual]

Evaluate the function and its gradient

Definition at line 614 of file gsl_mmin_conf.h.

The documentation for this class was generated from the following file:

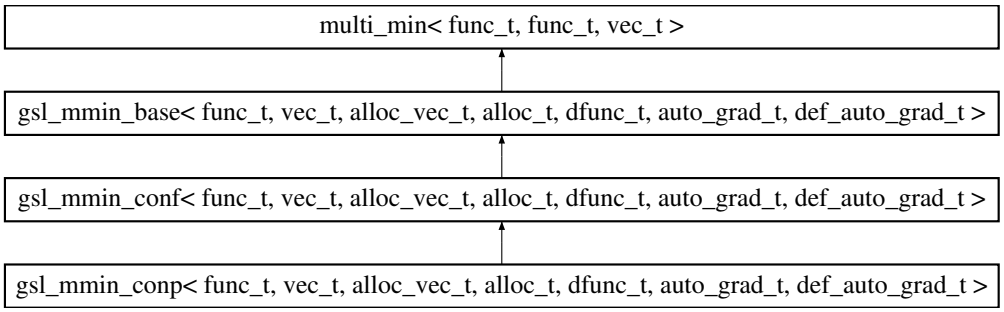
- gsl_mmin_conf.h

46.132 gsl_mmin_conp< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t > Class Template Reference

Multidimensional minimization by the Polak-Ribiere conjugate gradient algorithm (GSL)

`#include <gsl_mmin_conp.h>`

Inheritance diagram for gsl_mmin_conp< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >:



46.132.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>, class def_auto_grad_t = simple_grad<func_t,ovector_base>>class gsl_mmin_conp< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

The functions mmin() and mmin_de() minimize a given function until the gradient is smaller than the value of multi_min::tol_rel (which defaults to 10^{-4}).

See an example for the usage of this class in Multidimensional minimizer .

Definition at line 49 of file gsl_mmin_conp.h.

Public Member Functions

- virtual int iterate ()
  Perform an iteration.
- virtual const char * type ()
  Return string denoting type("gsl_mmin_conp")

The documentation for this class was generated from the following file:

- gsl_mmin_conp.h

46.133 gsl_mmin_linmin2 Class Reference

The line minimizer for gsl_mmin_bfgs2.

`#include <gsl_mmin_bfgs2.h>`

46.133.1 Detailed Description

Definition at line 286 of file gsl_mmin_bfgs2.h.

Public Member Functions

- int minimize (gsl_mmin_wrap_base &wrap, double rho, double sigma, double tau1, double tau2, double tau3, int order, double alpha1, double *alpha_new)
  The line minimization.

Protected Member Functions

- double interp_quad (double f0, double fp0, double f1, double zl, double zh)
  Minimize the interpolating quadratic.
- double cubic (double c0, double c1, double c2, double c3, double z)
  Minimize the interpolating cubic.
- void check_extremum (double c0, double c1, double c2, double c3, double z, double *zmin, double *fmin)
  Test to see curvature is positive.
- double interp_cubic (double f0, double fp0, double f1, double fp1, double zl, double zh)
  Interpolate using a cubic.
- double interpolate (double a, double fa, double fpa, double b, double fb, double fpb, double xmin, double xmax, int order)
  Perform the interpolation.

46.133.2 Member Function Documentation

46.133.2.1 double gsl_mmin_linmin2::interp_quad (double *f0*, double *fp0*, double *f1*, double *zl*, double *zh*) [protected]

Find a minimum in x=[0,1] of the interpolating quadratic through (0,f0) (1,f1) with derivative fp0 at x=0. The interpolating polynomial is q(x)=f0 + fp0 * z + (f1-f0-fp0) * z^2

46.133.2.2 double gsl_mmin_linmin2::cubic (double *c0*, double *c1*, double *c2*, double *c3*, double *z*) [protected]

Find a minimum in x=[0,1] of the interpolating cubic through (0,f0) (1,f1) with derivatives fp0 at x=0 and fp1 at x=1.

The interpolating polynomial is:

c(x)=f0 + fp0 * z + eta * z^2 + xi * z^3

where eta=3*(f1-f0)-2*fp0-fp1, xi=fp0+fp1-2*(f1-f0).

46.133.2.3 int gsl_mmin_linmin2::minimize (gsl_mmin_wrap_base & *wrap*, double *rho*, double *sigma*, double *tau1*, double *tau2*, double *tau3*, int *order*, double *alpha1*, double * *alpha_new*)

Recommended values from Fletcher87 are `rho=0.01, sigma=0.1, tau1=9, tau2=0.05, tau3=0.5`

The documentation for this class was generated from the following file:

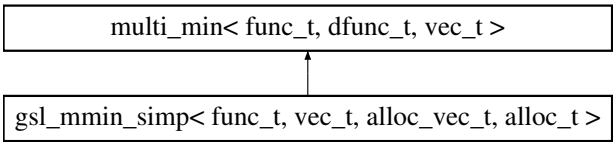
- gsl_mmin_bfgs2.h

46.134 gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Multidimensional minimization by the Simplex method (GSL)

`#include <gsl_mmin_simp.h>`

Inheritance diagram for gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >:



46.134.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >

This class minimizes a function using Nelder and Mead's Simplex algorithm. A simplex in a N-dimensional space is defined as a set of N+1 points which describe an N-dimensional volume surrounding the minimum. The algorithm proceeds by shifting the simplex points until the simplex is sufficiently small and thus the minimum is known with sufficient accuracy.

For a slightly improved method, see gsl_mmin_simp2 .

This class has a high-level interface using mmin(), mmin_twovec() or mmin_simplex() which automatically performs the memory allocation and minimization, or a GSL-like interface using allocate(), free(), interate() and set() or set_simplex().

The simplex can be completely specified by the user (see mmin_simplex() and set_simplex()). Alternatively, the simplex is automatically specified given initial guess x_j and a step size vector s_k for $0 \leq k < n_s$. The simplex p_{ij} with $0 \leq i \leq n$ and $0 \leq j < n$ is chosen with $p_{0j} = x_j$ and

$$\begin{aligned} p_{i+1,j} &= x_j \quad \text{for} \quad i \neq j \\ p_{i+1,j} &= x_j + s_{j \bmod n_s} \quad \text{for} \quad i = j \end{aligned}$$

for $0 < i < n$. The step size vector s is set by the set_step() member function. The prsence of mod in the recipe above just indicates that elements of the step size vector are automatically re-used if there are less step sizes than dimensions in the minimization problem.

Note

It is important that the initial simplex contains sufficient variation in every direction of the parameter space over which one is minimizing. For example, if all three points in a simplex for minimizing over a two-dimensional space contain nearly the same value for the second parameter, then the minimizer may only minimize the function with respect to the first parameter.

Default template arguments

- `func_t` - multi_funct<>
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc

Based on Nelder65 .

Definition at line 106 of file gsl_mmin_simp.h.

Public Member Functions

- template<class vec2_t >
  int set_step (size_t nv, vec2_t &step)
  Set the step sizes for each independent variable.
- virtual int mmin (size_t nn, vec_t &xx, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`.
- virtual int mmin_twovec (size_t nn, vec_t &xx, vec_t &xx2, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`, using `xx` and `xx2` to specify the simplex.
- template<class mat_t >
  int mmin_simplex (size_t nn, mat_t &sx, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`, given an initial simplex.
- virtual int allocate (size_t n)
  Allocate the memory.
- virtual int free ()
  Free the allocated memory.
- virtual int set (func_t &ufunc, size_t n, vec_t &ax, vec_t &step_size)
  Set the function and initial guess.
- template<class mat_t >
  int set_simplex (func_t &ufunc, mat_t &sx)
  Set the function and initial simplex.
- virtual int iterate ()
  Perform an iteration.
- virtual int print_iter (size_t nv, vec_t &xx, alloc_vec_t *simp, double y, int iter, double value, double limit, std::string comment)
  Print out iteration information.
- virtual const char * type ()
  Return string denoting type("gsl_mmin_simp")

Data Fields

- double size
  Size of current simplex computed by iterate()
- alloc_vec_t x
  Present minimum vector computed by iterate()
- double fval
  Function value at minimum computed by iterate()
- int print_simplex
  Print simplex information in print_iter() (default 0)

Protected Member Functions

- int nmsimplex_calc_center (vec_t &mp)
  Compute the center of the simplex and store in `mp`.
- double nmsimplex_size ()
  Compute the size of the simplex.
- virtual int move_corner_err (const double coeff, size_t corner, vec_t &xc, func_t &f, size_t nvar, double &newval)
  Move a corner of a simplex.
- virtual int contract_by_best (size_t best, vec_t &xc, func_t &f, size_t nvar)
  Contract the simplex towards the best point.

Protected Attributes

- alloc_vec_t * x1
  An array of n+1 vectors containing the simplex.
- ovector y1
  The n+1 function values at the simplex points.

- alloc_vec_t ws1
  Workspace vector 1.
- alloc_vec_t ws2
  Workspace vector 2.
- alloc_vec_t ws3
  Workspace vector 3.
- size_t dim
  Number of variables to be minimized over.
- func_t * func
  Function.
- bool set_called
  True if set() has been called.
- ovector step_vec
  Vector of step sizes.
- alloc_t ao
  Vector allocator.
- bool avoid_nonzero
  If true, try to automatically avoid regions where the function returns a non-zero value (default false)

46.134.2 Member Function Documentation

46.134.2.1 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >::nmsimplex_size ( ) [inline, protected]

Calculates simplex size as average sum of length of vectors from simplex center to corner points:

$(1/n)\sum||y - y_{\text{middlepoint}}||$

Definition at line 161 of file gsl_mmin_simp.h.

46.134.2.2 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >::move_corner_err (const double *coeff*, size_t *corner*, vec_t & *xc*, func_t & *f*, size_t *nvar*, double & *newval*) [inline, protected, virtual]

Moves a simplex corner scaled by coeff (negative value represents mirroring by the middle point of the "other" corner points) and gives new corner in xc and function value at xc in `newval`.

Definition at line 182 of file gsl_mmin_simp.h.

46.134.2.3 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >::contract_by_best (size_t *best*, vec_t & *xc*, func_t & *f*, size_t *nvar*) [inline, protected, virtual]

Function contracts the simplex in respect to best valued corner. All corners besides the best corner are moved.

The vector, `xc`, is used as work space.

Definition at line 212 of file gsl_mmin_simp.h.

46.134.2.4 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >::print_iter (size_t *nv*, vec_t & *xx*, alloc_vec_t * *simp*, double *y*, int *iter*, double *value*, double *limit*, std::string *comment*) [inline, virtual]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 763 of file gsl_mmin_simp.h.

46.134.3 Field Documentation

46.134.3.1 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> ovector gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >::y1 [protected]

Definition at line 129 of file gsl_mmin_simp.h.

46.134.3.2 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> bool gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >::avoid_nonzero [protected]

Note

This option doesn't work yet, so I've made the variable protected to prevent the user from changing it.

Definition at line 288 of file gsl_mmin_simp.h.

46.134.3.3 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int gsl_mmin_simp< func_t, vec_t, alloc_vec_t, alloc_t >::print_simplex

If this is 1 and verbose is greater than 0, then print_iter() will print the function values at all the simplex points. If this is 2 and verbose is greater than 0, then print_iter() will print the simplex coordinates in addition to the function values.

Definition at line 334 of file gsl_mmin_simp.h.

The documentation for this class was generated from the following file:

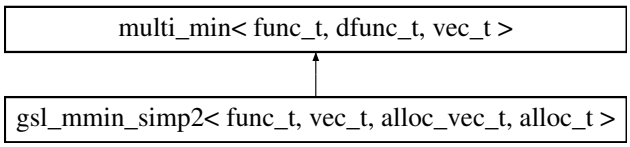
- gsl_mmin_simp.h

46.135 gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Multidimensional minimization by the Simplex method (v2) (GSL)

`#include <gsl_mmin_simp2.h>`

Inheritance diagram for gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >:



46.135.1 Detailed Description

template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >

This class minimizes a function using Nelder and Mead's Simplex algorithm. A simplex in a N-dimensional space is defined as a set of N+1 points which describe an N-dimensional volume surrounding the minimum. The algorithm proceeds by shifting the simplex points until the simplex is sufficiently small and thus the minimum is known with sufficient accuracy.

For the earlier method used in GSL, see gsl_mmin_simp .

This class has a high-level interface using mmin(), mmin_twovec() or mmin_simplex() which automatically performs the memory allocation and minimization, or a GSL-like interface using allocate(), free(), interate() and set() or set_simplex().

The simplex can be completely specified by the user (see mmin_simplex() and set_simplex()). Alternatively, the simplex is automatically specified given initial guess x_j and a step size vector s_k for $0 \leq k < n_s$. The simplex p_{ij} with $0 \leq i \leq n$ and $0 \leq j < n$ is chosen with $p_{0j} = x_j$ and

$$p_{i+1,j} \quad = \quad x_j \quad \text{for} \quad i \neq j$$

$$p_{i+1,j} = x_j + s_{j \bmod n_s} \quad \text{for} \quad i = j$$

for $0 < i < n$. The step size vector s is set by the set_step() member function. The prsence of mod in the recipe above just indicates that elements of the step size vector are automatically re-used if there are less step sizes than dimensions in the minimization problem.

Note

It is important that the initial simplex contains sufficient variation in every direction of the parameter space over which one is minimizing. For example, if all three points in a simplex for minimizing over a two-dimensional space contain nearly the same value for the second parameter, then the minimizer may only minimize the function with respect to the first parameter.

See an example for the usage of this class in Multidimensional minimizer .

Default template arguments

- `param_t` - no default
- `func_t` - multi_funct<param_t>
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc

Based on Nelder65 .

A variable `count` originally defined in the GSL simplex state is not present here, because it was unused.

Idea for Future Double check that the updates in gsl-1.13 are included here, and also add support for the nmsimplex2rand algorithm in GSL.

Definition at line 120 of file gsl_mmin_simp2.h.

Public Member Functions

- template<class vec2_t >
  int set_step (size_t nv, vec2_t &step)
  Set the step sizes for each independent variable.
- virtual int mmin (size_t nn, vec_t &xx, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`.
- virtual int mmin_twovec (size_t nn, vec_t &xx, vec_t &xx2, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`, using `xx` and `xx2` to specify the simplex.
- template<class mat_t >
  int mmin_simplex (size_t nn, mat_t &sx, double &fmin, func_t &ufunc)
  Calculate the minimum `min` of `func` w.r.t the array `x` of size `nvar`, given an initial simplex.
- virtual int allocate (size_t n)
  Allocate the memory.
- virtual int free ()
  Free the allocated memory.
- virtual int set (func_t &ufunc, size_t n, vec_t &ax, vec_t &step_size)
  Set the function and initial guess.
- template<class mat_t >
  int set_simplex (func_t &ufunc, mat_t &sx)
  Set the function and initial simplex.
- virtual int iterate ()
  Perform an iteration.
- virtual int print_iter (size_t nv, vec_t &xx, alloc_vec_t *simp, double y, int iter, double value, double limit, std::string comment)
  Print out iteration information.
- virtual const char * type ()
  Return string denoting type("gsl_mmin_simp2")

Data Fields

- double size
  Size of current simplex computed by iterate()
- alloc_vec_t x
  Present minimum vector computed by iterate()
- double fval
  Function value at minimum computed by iterate()
- int print_simplex
  Print simplex information in print_iter() (default 0)

Protected Member Functions

- int compute_center ()
  Compute the center of the simplex.
- double compute_size ()
  Compute the size of the simplex.
- virtual int try_corner_move (const double coeff, size_t corner, vec_t &xc, func_t &f, size_t nvar, double &newval)
  Move a corner of a simplex.
- virtual int update_point (size_t i, vec_t &xx, double val)
  Update point `i` in the simplex with values `xx`.
- virtual int contract_by_best (size_t best, func_t &f, size_t nvar)
  Contract the simplex towards the best point.

Protected Attributes

- alloc_vec_t * x1
  An array of n+1 vectors containing the simplex.
- ovector y1
  The n+1 function values at the simplex points.
- alloc_vec_t ws1
  Workspace vector 1.
- alloc_vec_t ws2
  Workspace vector 2.
- alloc_vec_t ws3
  Workspace vector 3.
- alloc_vec_t center
  Center of simplex.
- alloc_vec_t delta
  Desc.
- alloc_vec_t xmc
  Distance of vector from center.
- double S2
  Squared simplex size.
- size_t dim
  Number of variables to be minimized over.
- func_t * func
  Function.
- bool set_called
  True if set() has been called.
- ovector step_vec
  Vector of step sizes.
- alloc_t ao
  Vector allocator.
- bool avoid_nonzero
  If true, try to automatically avoid regions where the function returns a non-zero value (default false)

46.135.2 Member Function Documentation

46.135.2.1 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >::compute_size ( ) [inline, protected]

Calculates simplex size as average sum of length of vectors from simplex center to corner points:

$(1/n)\sum||y-y_{\text{center}}||$

Definition at line 183 of file gsl_mmin_simp2.h.

46.135.2.2 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >::try_corner_move (const double *coeff*, size_t *corner*, vec_t & *xc*, func_t & *f*, size_t *nvar*, double & *newval*) [inline, protected, virtual]

Moves a simplex corner scaled by coeff (negative value represents mirroring by the middle point of the "other" corner points) and gives new corner in xc and function value at xc in `newval`.

Definition at line 207 of file gsl_mmin_simp2.h.

46.135.2.3 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >::contract_by_best (size_t *best*, func_t & *f*, size_t *nvar*) [inline, protected, virtual]

Function contracts the simplex in respect to best valued corner. All corners besides the best corner are moved.

Definition at line 279 of file gsl_mmin_simp2.h.

46.135.2.4 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >::print_iter (size_t *nv*, vec_t & *xx*, alloc_vec_t * *simp*, double *y*, int *iter*, double *value*, double *limit*, std::string *comment*) [inline, virtual]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 842 of file gsl_mmin_simp2.h.

46.135.3 Field Documentation

46.135.3.1 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> ovector gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >::y1 [protected]

Definition at line 143 of file gsl_mmin_simp2.h.

46.135.3.2 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> bool gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >::avoid_nonzero [protected]

Note

This option doesn't work yet, so I've made the variable protected to prevent the casual user from changing it.

Definition at line 365 of file gsl_mmin_simp2.h.

46.135.3.3 template<class func_t = multi_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int gsl_mmin_simp2< func_t, vec_t, alloc_vec_t, alloc_t >::print_simplex

If this is 1 and verbose is greater than 0, then print_iter() will print the function values at all the simplex points. If this is 2 and verbose is greater than 0, then print_iter() will print the simplex coordinates in addition to the function values.

Definition at line 411 of file gsl_mmin_simp2.h.

The documentation for this class was generated from the following file:

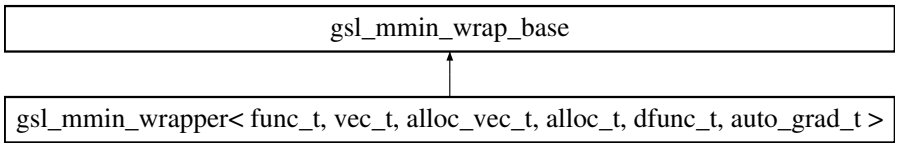
- gsl_mmin_simp2.h

46.136 gsl_mmin_wrap_base Class Reference

Virtual base for the gsl_mmin_bfgs2 wrapper.

`#include <gsl_mmin_bfgs2.h>`

Inheritance diagram for gsl_mmin_wrap_base:



46.136.1 Detailed Description

This class is useful so that the gsl_mmin_linmin class doesn't need to depend on any template parameters, even though it will need a wrapping object as an argument for the gsl_mmin_linmin::minimize() function.

Definition at line 43 of file gsl_mmin_bfgs2.h.

Public Member Functions

- virtual double wrap_f (double alpha)=0
  Function.
- virtual double wrap_df (double alpha)=0
  Derivative.
- virtual void wrap_fdf (double alpha, double *f, double *df)=0
  Function and derivative.

The documentation for this class was generated from the following file:

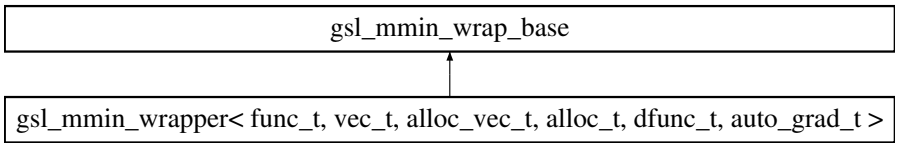
- gsl_mmin_bfgs2.h

46.137 gsl_mmin_wrapper< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t > Class Template Reference

Wrapper class for the gsl_mmin_bfgs2 minimizer.

`#include <gsl_mmin_bfgs2.h>`

Inheritance diagram for gsl_mmin_wrapper< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t >:



46.137.1 Detailed Description

template<class func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>>class gsl_mmin_wrapper< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t >

This is a reimplmentation of the internal GSL wrapper for function calls in the BFGS minimizer.

Definition at line 63 of file gsl_mmin_bfgs2.h.

Public Member Functions

- void prepare_wrapper (func_t &ufunc, dfunc_t *udfunc, vec_t &t_x, double f, vec_t &t_g, vec_t &t_p, auto_grad_t *ag)
  Initialize wrapper.
- void update_position (double alpha, vec_t &t_x, double *t_f, vec_t &t_g)
  Update position.
- void change_direction ()
  Convert cache values to the new minimizer direction.

Data Fields

- alloc_vec_t av_x_alpha
  Temporary storage.
- alloc_vec_t av_g_alpha
  Temporary storage.
- size_t dim
  Number of minimization dimensions.

Protected Member Functions

- void moveto (double alpha)
  Move to a new point, using the cached value if possible.
- double slope ()
  Compute the slope.
- virtual double wrap_f (double alpha)
  Evaluate the function.
- virtual double wrap_df (double alpha)
  Evaluate the derivative.
- virtual void wrap_fdf (double alpha, double *f, double *df)
  Evaluate the function and the derivative.

Protected Attributes

- func_t * func
  Function.
- dfunc_t * dfunc
  Derivative.
- auto_grad_t * agrad
  The automatic gradient object.
- bool grad_given
  True if the gradient was given by the user.

fixed values

- vec_t * **x**
- vec_t * **g**
- vec_t * **p**

cached values, for x(alpha)=x + alpha * p

- double **f_alpha**
- double **df_alpha**

cache keys

- double **f_cache_key**
- double **df_cache_key**
- double **x_cache_key**
- double **g_cache_key**

46.137.2 Member Function Documentation

46.137.2.1 template<class func_t , class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<ovector_base>, class auto_grad_t = gradient<func_t,ovector_base>> void gsl_mmin_wrapper< func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t >::change_direction ( ) [inline]

Convert the cache values from the end of the current minimisation to those needed for the start of the next minimisation, alpha=0

Definition at line 259 of file gsl_mmin_bfgs2.h.

The documentation for this class was generated from the following file:

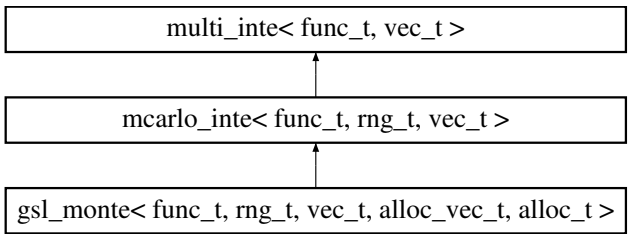
- gsl_mmin_bfgs2.h

46.138 gsl_monte< func_t, rng_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Multidimensional integration using plain Monte Carlo (GSL)

`#include <gsl_monte.h>`

Inheritance diagram for gsl_monte< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >:



46.138.1 Detailed Description

template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_monte< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

Definition at line 60 of file gsl_monte.h.

Public Member Functions

- virtual int minteg_err (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, double &res, double &err)
  Integrate function `func` from x=a to x=b.
- virtual double minteg (func_t &func, size_t ndim, const vec_t &a, const vec_t &b)
  Integrate function `func` over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i <$ ndim-1.
- virtual const char * type ()

Return string denoting type ("gsl_monte")

The documentation for this class was generated from the following file:

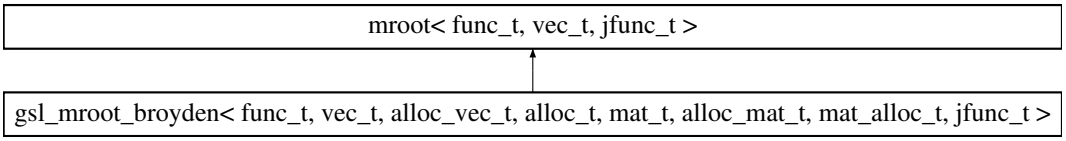
- gsl_monte.h

46.139 gsl_mroot_broyden< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t > Class Template - Reference

Multidimensional root-finding using Broyden's method (GSL)

`#include <gsl_mroot_broyden.h>`

Inheritance diagram for gsl_mroot_broyden< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >:



46.139.1 Detailed Description

template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>>class gsl_mroot_broyden< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >

Experimental.

See Broyden65.

Definition at line 45 of file gsl_mroot_broyden.h.

Public Member Functions

- void allocate (size_t n)
  Allocate memory.
- double enorm (size_t nvar, const vec_t &ff)
  Euclidean norm.
- void set (func_t &func, size_t nvar, vec_t &x, vec_t &f, vec_t &dx)
  Set the function, initial guess, and provide vectors to store function values and stepsize.
- int iterate ()
  Perform an iteration.
- virtual int msolve (size_t n, vec_t &x, func_t &func)
  Desc.
- void free ()
  Desc.

Data Fields

- simple_jacobian< func_t, vec_t, mat_t, alloc_vec_t, alloc_t > def_jac
  Default Jacobian object.

Protected Member Functions

- void clear ()
  Clear allocated vectors and matrices.

Protected Attributes

- omatrix H
  Desc.
- omatrix lu
  LU decomposition.
- permutation perm
  Permutation object for the LU decomposition.
- ovector v
  Desc.
- ovector w
  Desc.
- ovector y
  Desc.
- ovector p
  Desc.
- ovector fnew
  Desc.
- ovector x_trial
  Desc.
- double phi
  Desc.
- alloc_t ao
  Memory allocation object.
- alloc_vec_t dx_int
  Stepsize vector.
- alloc_vec_t f_int
  Function value vector.
- func_t * user_func
  A pointer to the user-specified function.
- vec_t * user_f
  Function values.
- vec_t * user_x
  Initial guess and current solution.
- vec_t * user_dx
  Initial and current step.
- size_t user_nvar
  Number of variables.
- size_t mem_size
  Size of memory allocated.
- jacobian< func_t, vec_t, mat_t > * ajac
  Jacobian.

46.139.2 Member Function Documentation

46.139.2.1 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>> void gsl_mroot_broyden< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >::clear ( ) [inline, protected]

This function is called by set() before each solve.

Definition at line 113 of file gsl_mroot_broyden.h.

46.139.2.2 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>> void gsl_mroot_broyden< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >::set (func_t & *func*, size_t *nvar*, vec_t & *x*, vec_t & *f*, vec_t & *dx*) [inline]

The initial values of `f` and `dx` are ignored.

Definition at line 187 of file gsl_mroot_broyden.h.

The documentation for this class was generated from the following file:

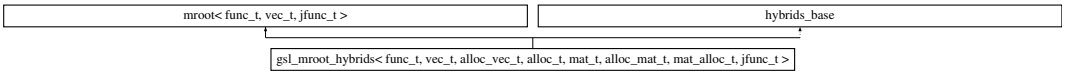
- gsl_mroot_broyden.h

46.140 gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t > Class Template - Reference

Multidimensional root-finding algorithm using Powell's Hybrid method (GSL)

`#include <gsl_mroot_hybrids.h>`

Inheritance diagram for gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >:



46.140.1 Detailed Description

template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>>class gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >

This is a recasted version of the GSL routines which use a modified version of Powell's Hybrid method as implemented in the HYBRJ algorithm in MINPACK (Garbow80). Both the scaled and unscaled options are available by setting int_scaling (the scaled version is the default). If derivatives are not provided, they will be computed automatically. This class provides the GSL-like interface using allocate(), set() (or set_de() in case where derivatives are available), iterate(), and free() and higher-level interfaces, msolve() and msolve_de(), which perform the solution and the memory allocation automatically. Some additional checking is performed in case the user calls the functions out of order (i.e. set() without allocate()).

The functions msolve() and msolve_de() use the condition $\sum_i |f_i| <$ mroot::tol_rel to determine if the solver has succeeded.

The original GSL algorithm has been modified to shrink the stepsize if a proposed step causes the function to return a non-zero value. This allows the routine to automatically try to avoid regions where the function is not defined. To return to the default GSL behavior, set shrink_step to false.

The default method for numerically computing the Jacobian is from simple_jacobian. This default is identical to the GSL approach, except that the default value of simple_jacobian::epsmin is non-zero. See simple_jacobian for more details.

There is an example for the usage of the multidimensional solver classes given in `examples/ex_mroot.cpp`, see the Multi-dimensional solver .

Todo Are all the set_all() statements really necessary? Do they need to be executed even if memory hasn't been recently allocated?

Idea for Future It's kind of strange that set() sets jac_given to false and set_de() has to reset it to true. Can this be simplified?

Idea for Future It might be good to make sure that we're directly testing to make sure that GSL and O2SCL are more or less identical.

Idea for Future Many of these minpack functions could be put in their own "minpack_tools" class, or possibly moved to be linear algebra routines instead.

Idea for Future There are still some numbers in here which the user could have control over, for example, the `nslow2` threshold which indicates failure.

Definition at line 485 of file gsl_mroot_hybrids.h.

Public Member Functions

- virtual int set_jacobian (jacobian< func_t, vec_t, mat_t > &j)
  Set the automatic Jacobian object.
- int iterate ()
  Perform an iteration.
- int allocate (size_t n)
  Allocate the memory.
- int free ()
  Free the allocated memory.
- virtual const char * type ()
  Return the type, "`gsl_mroot_hybrids`".
- virtual int msolve_de (size_t nn, vec_t &xx, func_t &ufunc, jfunc_t &dfunc)
  Solve `func` with derivatives `dfunc` using `x` as an initial guess, returning `x`.
- virtual int msolve (size_t nn, vec_t &xx, func_t &ufunc)
  Solve `ufunc` using `xx` as an initial guess, returning `xx`.
- int set (size_t nn, vec_t &ax, func_t &ufunc)
  Set the function, the parameters, and the initial guess.
- int set_de (size_t nn, vec_t &ax, func_t &ufunc, jfunc_t &dfunc)
  Set the function, the Jacobian, the parameters, and the initial guess.

Data Fields

- bool shrink_step
  If true, iterate() will shrink the step-size automatically if the function returns a non-zero value (default true)
- bool int_scaling
  If true, use the internal scaling method (default true)
- simple_jacobian< func_t, vec_t, mat_t, alloc_vec_t, alloc_t > def_jac
  Default automatic Jacobian object.
- alloc_vec_t f
  The value of the function at the present iteration.
- alloc_vec_t x
  The present solution.

Protected Member Functions

- int solve_set (size_t nn, vec_t &xx, func_t &ufunc)
  Finish the solution after set() or set_de() has been called.

Protected Attributes

- int iter
  Number of iterations.
- size_t ncfail
  Compute the number of failures.
- size_t ncsuc
  Compute the number of successes.
- size_t nslow1

The number of times the actual reduction is less than 0.001.

- size_t nslow2
  The number of times the actual reduction is less than 0.1.
- double fnorm
  The norm of the current function value.
- double delta
  The limit of the Nuclidean norm.
- alloc_mat_t J
  Jacobian.
- omatrix q
  Q matrix from QR decomposition.
- omatrix r
  R matrix from QR decomposition.
- ovector tau
  The tau vector from QR decomposition.
- ovector diag
  The diagonal elements.
- ovector qtf
  The value of $Q^T f$.
- ovector newton
  The Newton direction.
- ovector gradient
  The gradient direction.
- ovector df
  The change in the function value.
- ovector qtdf
  The value of $Q^T \cdot$ df.
- ovector rdx
  The value of $R \cdot$ dx.
- ovector w
  The value of $w = (Q^T df - Rdx)/|dx|$.
- ovector v
  The value of $v = D^2 dx/|dx|$.
- jfunc_t * jac
  The user-specified Jacobian.
- jacobian< func_t, vec_t, mat_t > * ajac
  The automatic Jacobian.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- mat_alloc_t am
  Memory allocator for objects of type `alloc_mat_t`.
- alloc_vec_t dx
  The value of the derivative.
- alloc_vec_t x_trial
  Trial root.
- alloc_vec_t f_trial
  Trial function value.
- size_t dim
  The number of equations and unknowns.
- bool jac_given
  True if the jacobian has been given.
- func_t * fnewp
  The user-specified function.
- bool set_called
  True if "set" has been called.

46.140.2 Member Function Documentation

46.140.2.1 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>> int gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >::iterate ( ) [inline]

At the end of the iteration, the current value of the solution is stored in x.

Definition at line 675 of file gsl_mroot_hybrids.h.

46.140.2.2 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>> virtual int gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >::msolve_de (size_t *nn*, vec_t & *xx*, func_t & *ufunc*, jfunc_t & *dfunc*) [inline, virtual]

Reimplemented from mroot< func_t, vec_t, jfunc_t >.

Definition at line 972 of file gsl_mroot_hybrids.h.

46.140.3 Field Documentation

46.140.3.1 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>> bool gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >::shrink_step

The original GSL behavior can be obtained by setting this to `false`.

Definition at line 644 of file gsl_mroot_hybrids.h.

46.140.3.2 template<class func_t = mm_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_base, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<vec_t,mat_t>> alloc_vec_t gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >::f

Definition at line 665 of file gsl_mroot_hybrids.h.

The documentation for this class was generated from the following file:

- gsl_mroot_hybrids.h

46.141 gsl_ode_control< vec_t > Class Template Reference

Control structure for gsl_astep.

`#include <gsl_astep.h>`

46.141.1 Detailed Description

template<class vec_t = ovector_base>class gsl_ode_control< vec_t >

This class implements both the "standard" and "scaled" step control methods from GSL. The standard control method is the default. To use the scaled control, set standard to `false` and set the scale for each component using set_scale().

The control object is a four parameter heuristic based on absolute and relative errors eps_abs and eps_rel, and scaling factors a_y and a_dydt for the system state $y(t)$ and derivatives $y'(t)$ respectively.

The step-size adjustment procedure for this method begins by computing the desired error level D_i for each component. In the unscaled version,

$$D_i = \text{eps\_abs} + \text{eps\_rel} \times \left(\text{a\_y}|y_i| + \text{a\_dydt}\, h|y_i'|\right)$$

while in the scaled version the user specifies the scale for each component, s_i,

$$D_i = \text{eps\_abs}\, s_i + \text{eps\_rel} \times \left(\text{a\_y}|y_i| + \text{a\_dydt}\, h|y_i'|\right)$$

The desired error level D_i is compared to then observed error $E_i = |\text{yerr}_i|$. If the observed error E exceeds the desired error level D by more than 10 percent for any component then the method reduces the step-size by an appropriate factor,

$$h_{\text{new}} = S\, h_{\text{old}} \left(\frac{E}{D}\right)^{-1/q}$$

where q is the consistency order of the method (e.g. $q = 4$ for 4(5) embedded RK), and S is a safety factor of 0.9. The ratio E/D is taken to be the maximum of the ratios E_i/D_i.

If the observed error E is less than 50 percent of the desired error level D for the maximum ratio E_i/D_i then the algorithm takes the opportunity to increase the step-size to bring the error in line with the desired level,

$$h_{\text{new}} = S\, h_{\text{old}} \left(\frac{E}{D}\right)^{-1/(q+1)}$$

This encompasses all the standard error scaling methods. To avoid uncontrolled changes in the stepsize, the overall scaling factor is limited to the range 1/5 to 5.

If the user specified fewer scaling parameters than the number of ODEs, then the scaling parameters are reused as follows. If there are N ODEs and M scaling parameters, then for $i > M$, the ith scaling parameter s_i is set to be $s_{i\%M}$. If the user selects the scaled control by setting standard to `false` and no scale parameters are specified, this class reverts to the standard control.

Todo Double check that the improvements in the ode-initval2 routines are available here

Definition at line 122 of file gsl_astep.h.

Public Member Functions

- template<class svec_t >
  int set_scale (size_t nscal, const svec_t &scale)
  Set the scaling for each differential equation.
- virtual int hadjust (size_t dim, unsigned int ord, const vec_t &y, vec_t &yerr, vec_t &yp, double &h)
  Automatically adjust step-size.

Data Fields

- double eps_abs
  Absolute precision (default 10^{-6})
- double eps_rel
  Relative precision (default 0)
- double a_y
  Function scaling factor (default 1)
- double a_dydt
  Derivative scaling factor (default 0)
- bool standard
  Use standard or scaled algorithm (default true)

Static Public Attributes

Adjustment specification

- static const size_t hadj_nil = 0
  No adjustment required.
- static const size_t hadj_dec = 1
  Recommend step decrease.
- static const size_t hadj_inc = 2
  Recommend step increase.

Protected Attributes

- uvector scale_abs
  Scalings.

The documentation for this class was generated from the following file:

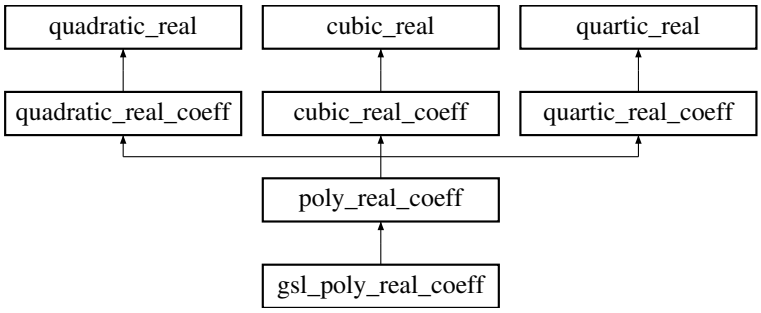
- gsl_astep.h

46.142 gsl_poly_real_coeff Class Reference

Solve a general polynomial with real coefficients (GSL)

`#include <poly.h>`

Inheritance diagram for gsl_poly_real_coeff:



46.142.1 Detailed Description

Definition at line 606 of file poly.h.

Public Member Functions

- virtual int solve_rc (int n, const double co[ ], std::complex< double > ro[ ])
  Solve a generic polynomial given `n+1` coefficients.
- virtual int solve_rc (const double a3, const double b3, const double c3, const double d3, double &x1, std::complex< double > &x2, std::complex< double > &x3)
  Solve a cubic polynomial with real coefficients.
- virtual int solve_rc (const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)
  Solve a quadratic polynomial with real coefficients.
- virtual int solve_rc (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)
  Solve a quartic polynomial with real coefficients.
- const char * type ()
  Return a string denoting the type ("gsl_poly_real_coeff")

Protected Attributes

- gsl_poly_complex_workspace * w2
  Workspace for quadratic polynomials.
- gsl_poly_complex_workspace * w3
  Workspace for cubic polynomials.

- gsl_poly_complex_workspace * w4
  Workspace for quartic polynomials.
- gsl_poly_complex_workspace * wgen
  Workspace for general polynomials.
- int gen_size
  The size of the workspace wgen.

46.142.2 Member Function Documentation

46.142.2.1 virtual int gsl_poly_real_coeff::solve_rc (int *n*, const double *co[]*, std::complex< double > *ro[]*) [virtual]

Note

In order to be consistent with the other solve_rc() functions, the ordering of the coefficients is reversed with respect to gsl_poly_complex_solve(). The leading coefficient is stored in `co[0]` and the constant term is stored in `co[n]`.

Implements poly_real_coeff.

The documentation for this class was generated from the following file:

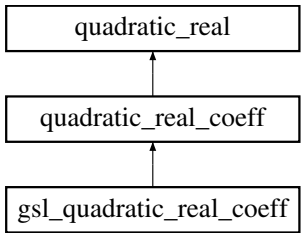
- poly.h

46.143 gsl_quadratic_real_coeff Class Reference

Solve a quadratic with real coefficients and complex roots (GSL)

`#include <poly.h>`

Inheritance diagram for gsl_quadratic_real_coeff:



46.143.1 Detailed Description

Definition at line 476 of file poly.h.

Public Member Functions

- virtual int solve_rc (const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)
  Solves the polynomial $a_2x^2 + b_2x + c_2 = 0$ giving the two complex solutions $x = x_1$ and $x = x_2$.
- const char * type ()
  Return a string denoting the type ("gsl_quadratic_real_coeff")

The documentation for this class was generated from the following file:

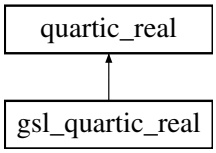
- poly.h

46.144 gsl_quartic_real Class Reference

Solve a quartic with real coefficients and real roots (GSL)

`#include <poly.h>`

Inheritance diagram for gsl_quartic_real:



46.144.1 Detailed Description

This class internally uses the GSL functions to solve the resolvent cubic and associated quadratics, while gsl_quartic_real2 contains explicit code to solve them instead.

Idea for Future Optimize value of `cube_root_tol` and compare more clearly to gsl_quartic_real2

Definition at line 539 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)
  Solves the polynomial $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$ giving the four real solutions $x = x_1$, $x = x_2$, $x = x_3$, and $x = x_4$.
- const char * type ()
  Return a string denoting the type ("gsl_quartic_real")

Data Fields

- double cube_root_tol
  A tolerance for determining the proper cube root (default 10^{-4})

The documentation for this class was generated from the following file:

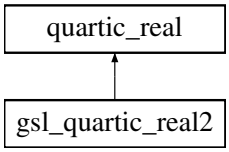
- poly.h

46.145 gsl_quartic_real2 Class Reference

Solve a quartic with real coefficients and real roots (GSL)

`#include <poly.h>`

Inheritance diagram for gsl_quartic_real2:



46.145.1 Detailed Description

This class directly solves resolvent cubic and associated quadratics without using the GSL functions (as done in gsl_quartic_real).

Idea for Future Optimize value of `cube_root_tol` and compare more clearly to gsl_quartic_real

Definition at line 576 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)
  Solves the polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four real solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- const char * type ()
  Return a string denoting the type ("gsl_quartic_real2")

Data Fields

- double cube_root_tol
  A tolerance for determining the proper cube root (default 10^{-7})

The documentation for this class was generated from the following file:

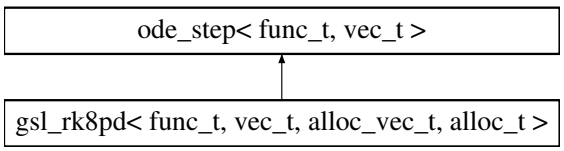
- poly.h

46.146 gsl_rk8pd< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Embedded Runge-Kutta Prince-Dormand ODE stepper (GSL)

`#include <gsl_rk8pd.h>`

Inheritance diagram for gsl_rk8pd< func_t, vec_t, alloc_vec_t, alloc_t >:



46.146.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_rk8pd< func_t, vec_t, alloc_vec_t, alloc_t >

Based on Prince81 .

There is an example for the usage of this class in `examples/ex_ode.cpp` documented in the Ordinary differential equations section.

Definition at line 62 of file gsl_rk8pd.h.

Public Member Functions

- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Perform an integration step.

Protected Attributes

- size_t ndim
  Size of allocated vectors.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

Storage for the intermediate steps

- alloc_vec_t **k2**
- alloc_vec_t **k3**
- alloc_vec_t **k4**
- alloc_vec_t **k5**
- alloc_vec_t **k6**
- alloc_vec_t **k7**
- alloc_vec_t **ytmp**
- alloc_vec_t **k8**
- alloc_vec_t **k9**
- alloc_vec_t **k10**
- alloc_vec_t **k11**
- alloc_vec_t **k12**
- alloc_vec_t **k13**

Storage for the coefficients

- double **Abar** [13]
- double **A** [12]
- double **ah** [10]
- double **b21**
- double **b3** [2]
- double **b4** [3]
- double **b5** [4]
- double **b6** [5]
- double **b7** [6]
- double **b8** [7]
- double **b9** [8]
- double **b10** [9]
- double **b11** [10]
- double **b12** [11]
- double **b13** [12]

46.146.2 Member Function Documentation

46.146.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_rk8pd< func_t, vec_t, alloc_vec_t, alloc_t >::step (double *x*, double *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

If `derivs` always returns zero, then this function will also return zero. If not, `step()` will return the first non-zero value which was obtained in a call to `derivs` . The error handler is never called.

Implements ode_step< func_t, vec_t >.

Definition at line 256 of file gsl_rk8pd.h.

The documentation for this class was generated from the following file:

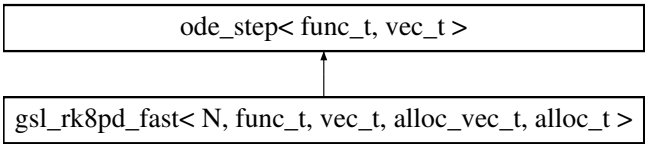
- gsl_rk8pd.h

46.147 gsl_rk8pd_fast< N, func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Faster embedded Runge-Kutta Prince-Dormand ODE stepper (GSL)

`#include <gsl_rk8pd.h>`

Inheritance diagram for gsl_rk8pd_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >:



46.147.1 Detailed Description

template<size_t N, class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_rk8pd_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >

This a fast version of gsl_rk8pd, which is a stepper for a fixed number of ODEs. It ignores the error values returned by the `derivs` argument. The argument `n` to step() should always be equal to the template parameter `N`, and the vector parameters to step must have space allocated for at least `N` elements. No error checking is performed to ensure that this is the case.

Based on Prince81 .

Definition at line 427 of file gsl_rk8pd.h.

Public Member Functions

- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Perform an integration step.

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

Storage for the intermediate steps

- alloc_vec_t **k2**
- alloc_vec_t **k3**
- alloc_vec_t **k4**
- alloc_vec_t **k5**
- alloc_vec_t **k6**
- alloc_vec_t **k7**
- alloc_vec_t **ytmp**
- alloc_vec_t **k8**
- alloc_vec_t **k9**
- alloc_vec_t **k10**
- alloc_vec_t **k11**
- alloc_vec_t **k12**
- alloc_vec_t **k13**

Storage for the coefficients

- double **Abar** [13]
- double **A** [12]
- double **ah** [10]
- double **b21**
- double **b3** [2]
- double **b4** [3]
- double **b5** [4]
- double **b6** [5]
- double **b7** [6]
- double **b8** [7]
- double **b9** [8]
- double **b10** [9]
- double **b11** [10]
- double **b12** [11]
- double **b13** [12]

46.147.2 Member Function Documentation

46.147.2.1 template<size_t N, class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_rk8pd_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >::step (double *x*, double *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

Note

The value of the parameter `n` should be equal to the template parameter `N`.

Implements ode_step< func_t, vec_t >.

Definition at line 628 of file gsl_rk8pd.h.

The documentation for this class was generated from the following file:

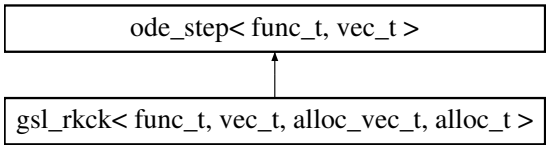
- gsl_rk8pd.h

46.148 gsl_rkck< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Cash-Karp embedded Runge-Kutta ODE stepper (GSL)

`#include <gsl_rkck.h>`

Inheritance diagram for gsl_rkck< func_t, vec_t, alloc_vec_t, alloc_t >:



46.148.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_rkck< func_t, vec_t, alloc_vec_t, alloc_t >

Based on Cash90 .

There is an example for the usage of this class in `examples/ex_ode.cpp` documented in the Ordinary differential equations section.

Definition at line 63 of file gsl_rkck.h.

Public Member Functions

- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Perform an integration step.

Protected Attributes

- size_t ndim
  Size of allocated vectors.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

Storage for the intermediate steps

- alloc_vec_t **k2**
- alloc_vec_t **k3**
- alloc_vec_t **k4**
- alloc_vec_t **k5**
- alloc_vec_t **k6**
- alloc_vec_t **ytmp**

Storage for the coefficients

- double **ah** [5]
- double **b3** [2]
- double **b4** [3]
- double **b5** [4]
- double **b6** [5]
- double **ec** [7]
- double **b21**
- double **c1**
- double **c3**
- double **c4**
- double **c6**

46.148.2 Member Function Documentation

46.148.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_rkck< func_t, vec_t, alloc_vec_t, alloc_t >::step (double *x*, double *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

If `derivs` always returns zero, then this function will also return zero. If not, `step()` will return the first non-zero value which was obtained in a call to `derivs` . The error handler is never called.

Implements ode_step< func_t, vec_t >.

Definition at line 160 of file gsl_rkck.h.

The documentation for this class was generated from the following file:

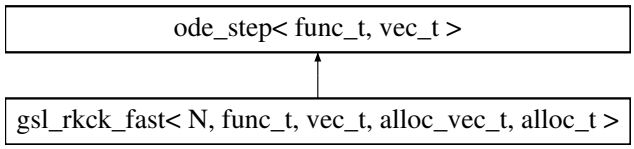
- gsl_rkck.h

46.149 gsl_rkck_fast< N, func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Faster Cash-Karp embedded Runge-Kutta ODE stepper (GSL)

`#include <gsl_rkck.h>`

Inheritance diagram for gsl_rkck_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >:



46.149.1 Detailed Description

template<size_t N, class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_rkck_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >

This a faster version of gsl_rkck, which is a stepper for a fixed number of ODEs. It ignores the error values returned by the `derivs` argument. The argument `n` to step() should always be equal to the template parameter `N`, and the vector parameters to step must have space allocated for at least `N` elements. No error checking is performed to ensure that this is the case.

Based on Cash90 .

Definition at line 253 of file gsl_rkck.h.

Public Member Functions

- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Perform an integration step.

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

Storage for the intermediate steps

- alloc_vec_t **k2**
- alloc_vec_t **k3**
- alloc_vec_t **k4**
- alloc_vec_t **k5**
- alloc_vec_t **k6**
- alloc_vec_t **ytmp**

Storage for the coefficients

- double **ah** [5]
- double **b3** [2]
- double **b4** [3]
- double **b5** [4]
- double **b6** [5]
- double **ec** [7]
- double **b21**
- double **c1**
- double **c3**
- double **c4**
- double **c6**

46.149.2 Member Function Documentation

46.149.2.1 template<size_t N, class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_rkck_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >::step (double *x*, double *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

Note

The value of the parameter `n` must be equal to the template parameter `N`.

Implements ode_step< func_t, vec_t >.

Definition at line 344 of file gsl_rkck.h.

The documentation for this class was generated from the following file:

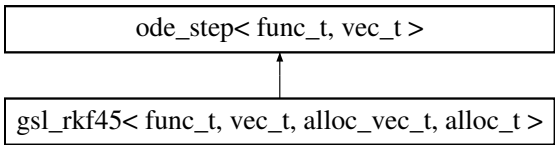
- gsl_rkck.h

46.150 gsl_rkf45< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Runge-Kutta-Fehlberg embedded Runge-Kutta ODE stepper (GSL)

`#include <gsl_rkf45.h>`

Inheritance diagram for gsl_rkf45< func_t, vec_t, alloc_vec_t, alloc_t >:



46.150.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_rkf45< func_t, vec_t, alloc_vec_t, alloc_t >

Based on Hairer00 .

Todo Check this because it may not give exact dydt_out.

Definition at line 60 of file gsl_rkf45.h.

Public Member Functions

- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Perform an integration step.

Protected Attributes

- size_t ndim
  Size of allocated vectors.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

Storage for the intermediate steps

- alloc_vec_t **k2**
- alloc_vec_t **k3**
- alloc_vec_t **k4**
- alloc_vec_t **k5**
- alloc_vec_t **k6**
- alloc_vec_t **ytmp**

Storage for the coefficients

- double **ah** [5]
- double **b3** [2]
- double **b4** [3]
- double **b5** [4]
- double **b6** [5]
- double **c1**
- double **c3**
- double **c4**
- double **c5**
- double **c6**
- double **ec** [7]

46.150.2 Member Function Documentation

46.150.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_rkf45< func_t, vec_t, alloc_vec_t, alloc_t >::step (double *x*, double *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

If `derivs` always returns zero, then this function will also return zero. If not, `step()` will return the first non-zero value which was obtained in a call to `derivs` . The error handler is never called.

Implements ode_step< func_t, vec_t >.

Definition at line 156 of file gsl_rkf45.h.

The documentation for this class was generated from the following file:

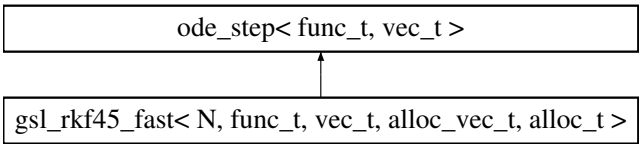
- gsl_rkf45.h

46.151 gsl_rkf45_fast< N, func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Faster Runge-Kutta-Fehlberg embedded Runge-Kutta ODE stepper (GSL)

`#include <gsl_rkf45.h>`

Inheritance diagram for gsl_rkf45_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >:



46.151.1 Detailed Description

template<size_t N, class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_rkf45_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >

This a faster version of gsl_rkf45, which is a stepper for a fixed number of ODEs. It ignores the error values returned by the `derivs` argument. The argument `n` to step() should always be equal to the template parameter `N`, and the vector parameters to step must have space allocated for at least `N` elements. No error checking is performed to ensure that this is the case.

Based on Hairer00 .

Definition at line 256 of file gsl_rkf45.h.

Public Member Functions

- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Perform an integration step.

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

Storage for the intermediate steps

- alloc_vec_t **k2**
- alloc_vec_t **k3**
- alloc_vec_t **k4**
- alloc_vec_t **k5**
- alloc_vec_t **k6**
- alloc_vec_t **ytmp**

Storage for the coefficients

- double **ah** [5]
- double **b3** [2]
- double **b4** [3]
- double **b5** [4]
- double **b6** [5]
- double **c1**
- double **c3**
- double **c4**
- double **c5**
- double **c6**
- double **ec** [7]

46.151.2 Member Function Documentation

46.151.2.1 template<size_t N, class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_rkf45_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >::step (double *x*, double *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty in `yerr`, and the new derivative in `dydx_out` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

Note

The value of the parameter `n` should be equal to the template parameter `N`.

Implements ode_step< func_t, vec_t >.

Definition at line 352 of file gsl_rkf45.h.

The documentation for this class was generated from the following file:

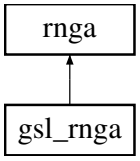
- gsl_rkf45.h

46.152 gsl_rnga Class Reference

Random number generator (GSL)

`#include <gsl_rnga.h>`

Inheritance diagram for gsl_rnga:



46.152.1 Detailed Description

If `seed` is zero, or is not given, then the default seed specific to the particular random number generator is used. No virtual functions are used in this class or its parent, rnga. This should be as fast as the original GSL version.

An interesting application of this class to generate an arbitrary distribution through a Markov chain Monte Carlo method is in Generate an arbitrary distribution.

Definition at line 45 of file gsl_rnga.h.

Public Member Functions

- gsl_rnga (const gsl_rng_type *gtype=gsl_rng_mt19937)
  Initialize the random number generator with type `gtype` and the default seed.
- gsl_rnga (unsigned long int seed, const gsl_rng_type *gtype=gsl_rng_mt19937)
  Initialize the random number generator with `seed`.
- const gsl_rng_type * get_type ()
  Return generator type.
- double random ()
  Return a random number in $(0, 1]$.

- unsigned long int get_max ()
  Return the maximum integer for random_int()
- unsigned long int random_int (unsigned long int n=0)
  Return random integer in $[0, \max - 1]$.
- void set_seed (unsigned long int s)
  Set the seed.
- void clock_seed ()
  Set the seed.

Protected Attributes

- gsl_rng * gr
  The GSL random number generator.
- const gsl_rng_type * rng
  The GSL random number generator type.

The documentation for this class was generated from the following file:

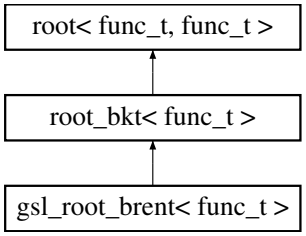
- gsl_rnga.h

46.153 gsl_root_brent< func_t > Class Template Reference

One-dimensional root-finding (GSL)

`#include <gsl_root_brent.h>`

Inheritance diagram for gsl_root_brent< func_t >:



46.153.1 Detailed Description

template<class func_t = funct>class gsl_root_brent< func_t >

This class finds the root of a user-specified function. If test_form is 0, then solve_bkt() stops when the size of the bracket is smaller than root::tol_abs. If test_form is 1, then the function stops when the residual is less than root::tol_rel. If test_form is 2, then both tests are applied.

An example demonstrating the usage of this class is given in `examples/ex_fptr.cpp` and the Function object example .

Idea for Future There is some duplication in the variables `x_lower`, `x_upper`, `a`, and `b`, which could be removed. Some better variable names would also be helpful.

Idea for Future Create a meaningful enum list for gsl_root_brent::test_form.

Definition at line 85 of file gsl_root_brent.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "gsl_root_brent".
- int iterate (func_t &f)
  Perform an iteration.
- virtual int solve_bkt (double &x1, double x2, func_t &f)
  Solve func in region $x_1 < x < x_2$ returning x_1.
- double get_root ()
  Get the most recent value of the root.
- double get_lower ()
  Get the lower limit.
- double get_upper ()
  Get the upper limit.
- int set (func_t &ff, double lower, double upper)
  Set the information for the solver.

Data Fields

- int test_form
  The type of convergence test applied: 0, 1, or 2 (default 0)

Protected Attributes

- double root
  The present solution estimate.
- double x_lower
  The present lower limit.
- double x_upper
  The present upper limit.

Storage for solver state

- double **a**
- double **b**
- double **c**
- double **d**
- double **e**
- double **fa**
- double **fb**
- double **fc**

46.153.2 Member Function Documentation

46.153.2.1 template<class func_t = funct> int gsl_root_brent< func_t >::iterate (func_t & *f*) [inline]

This function currently always returns gsl_success.

Definition at line 101 of file gsl_root_brent.h.

46.153.2.2 template<class func_t = funct> int gsl_root_brent< func_t >::set (func_t & *ff*, double *lower*, double *upper*) [inline]

This function currently always returns gsl_success.

Definition at line 335 of file gsl_root_brent.h.

The documentation for this class was generated from the following file:

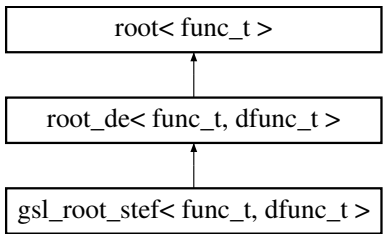
- gsl_root_brent.h

46.154 gsl_root_stef< func_t, dfunc_t > Class Template Reference

Steffenson equation solver (GSL)

`#include <gsl_root_stef.h>`

Inheritance diagram for gsl_root_stef< func_t, dfunc_t >:



46.154.1 Detailed Description

template<class func_t, class dfunc_t = func_t>class gsl_root_stef< func_t, dfunc_t >

This is Newton's method with an Aitken "delta-squared" acceleration of the iterates. This can improve the convergence on multiple roots where the ordinary Newton algorithm is slow.

Defining the next iteration with

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

the accelerated value is

$$x_{\mathrm{acc},i} = x_i - (x_{i+1} - x_i)^2/(x_{i+2} - 2x_{i+1} + x_i)$$

We can only use the accelerated estimate after three iterations, and use the unaccelerated value until then.

This class finds a root of a function a derivative. If the derivative is not analytically specified, it is most likely preferable to use of the alternatives, gsl_root_brent, cern_root, or cern_mroot_root. The function solve_de() performs the solution automatically, and a lower-level GSL-like interface with set() and iterate() is also provided.

By default, this solver compares the present value of the root (root) to the previous value (x), and returns success if $|\mathrm{root} - \mathrm{x}| < \mathrm{tol}$, where $\mathrm{tol} = \mathrm{tol\_abs} + \mathrm{tolf2\ root}$.

If test_residual is set to true, then the solver additionally requires that the absolute value of the function is less than root::tol_rel.

The original variable `x_2` has been removed as it was unused in the original GSL code.

Idea for Future There's some extra copying here which can probably be removed.

Idea for Future Compare directly to GSL.

Idea for Future This can probably be modified to shorten the step if the function goes out of bounds as in gsl_mroot_hybrids.

Definition at line 104 of file gsl_root_stef.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "gsl_root_stef".
- int iterate ()
  Perform an iteration.
- virtual int solve_de (double &xx, func_t &fun, dfunc_t &dfun)

Solve `func` using `x` as an initial guess using derivatives `df`.

- int set (func_t &fun, dfunc_t &dfun, double guess)
  Set the information for the solver.

Data Fields

- double root
  The present solution estimate.
- double tolf2
  The relative tolerance for subsequent solutions (default 10^{-12})
- bool test_residual
  True if we should test the residual also (default false)

Protected Attributes

- double f
  Function value.
- double df
  Derivative value.
- double x_1
  Previous value of root.
- double x
  Root.
- int count
  Number of iterations.
- func_t * fp
  The function to solve.
- dfunc_t * dfp
  The derivative.

46.154.2 Member Function Documentation

46.154.2.1 template<class func_t , class dfunc_t = func_t> int gsl_root_stef< func_t, dfunc_t >::iterate ( ) [inline]

After a successful iteration, root contains the most recent value of the root.

Definition at line 158 of file gsl_root_stef.h.

46.154.2.2 template<class func_t , class dfunc_t = func_t> int gsl_root_stef< func_t, dfunc_t >::set (func_t & *fun*, dfunc_t & *dfun*, double *guess*) [inline]

Set the function, the derivative, the initial guess and the parameters.

Definition at line 277 of file gsl_root_stef.h.

The documentation for this class was generated from the following file:

- gsl_root_stef.h

46.155 gsl_series Class Reference

Series acceleration by Levin u-transform (GSL)

`#include <gsl_series.h>`

46.155.1 Detailed Description

Given an array of terms in a sum, this attempts to evaluate the entire sum with an estimate of the error.

Idea for Future Move the worspaces to classes?

Definition at line 64 of file gsl_series.h.

Public Member Functions

- gsl_series (size_t size=0)
  `size` *is the number of terms in the series*
- template<class vec_t >
  double series_accel (size_t na, vec_t &array, double &abserr_trunc)
  Return the accelerated sum of the series with a simple error estimate.
- template<class vec_t >
  double series_accel_err (size_t na, vec_t &array, double &abserr)
  Return the accelerated sum of the series with an accurate error estimate.
- void set_size (size_t new_size)
  Set the number of terms.

46.155.2 Member Function Documentation

46.155.2.1 template<class vec_t > double gsl_series::series_accel (size_t *na*, vec_t & *array*, double & *abserr_trunc*) [inline]

The input vector `x` should be an array with `n` values from `x[0]` to `x[n-1]`.

Definition at line 81 of file gsl_series.h.

46.155.2.2 template<class vec_t > double gsl_series::series_accel_err (size_t *na*, vec_t & *array*, double & *abserr*) [inline]

The input vector `x` should be an array with `n` values from `x[0]` to `x[n-1]`.

Definition at line 204 of file gsl_series.h.

The documentation for this class was generated from the following file:

- gsl_series.h

46.156 gsl_smooth Class Reference

Smooth a GSL vector using GSL bsplines.

`#include <gsl_smooth.h>`

46.156.1 Detailed Description

Todo Needs a bit more error checking and more documentation.

Idea for Future Generalize to generic vector types. (Does this require reworking the GSL linear fitting routines?) In the meantime, make a ovector interface.

Idea for Future Possibly create a new gsl_bspline class which replaces the GSL bspline workspace

Idea for Future Allow user to probe chi squared and the covariance?

Definition at line 53 of file gsl_smooth.h.

Public Member Functions

- gsl_smooth (const gsl_vector ∗ix)
  Begin using x-values from vector `ix`.
- void set_ncoeff (int incoeffs)
  Set the number of coefficients.
- void set_order (int order)
  Set order.
- void set_pars (int incoeffs, int order)
  Set parameters.
- void set_x (const gsl_vector ∗ix)
  Set the x-values.
- int smooth_data (const gsl_vector ∗y, const gsl_vector ∗e, gsl_vector ∗ys)
  Smooth data in `y` with errors `e` returning result `ys`.
- int smooth_data (const gsl_vector ∗y, gsl_vector ∗ys)
  Smooth data in `y` returning result `ys`.

Protected Member Functions

- int fit (const gsl_vector ∗y)
  Construct un-weighted fit.
- int fit_errors (const gsl_vector ∗y, const gsl_vector ∗e)
  Construct weighted fit.
- double calc_for_x (double xi)
  calculate smoothed curve value for a certain xi
- int init ()
  Allocate memory and initialize splines.
- void init_pointers_and_defs ()
  Set default values and zero pointers.
- int free ()
  Free memory.

Protected Attributes

- size_t ncoeffs
  Number of free coefficients for spline.
- size_t norder
  Order of spline to be used (4=cubic)
- size_t nbreak
  internally calculated, number of "segment" to split the data into
- bool x_set
  True of the x values have been set.
- gsl_bspline_workspace ∗ bw
  Spline workspace.
- gsl_vector ∗ B
  Spline temporary vector.
- gsl_vector ∗ c
  Parameters of linear fit, y=X∗c.
- gsl_multifit_linear_workspace ∗ mw
  Linear fit workspace.
- gsl_matrix ∗ X
  Workspace for spline fitting.
- const gsl_vector ∗ x
  Values of the independent variable.
- gsl_matrix ∗ cov
  Covariance matrix.

46.156.2 Member Function Documentation

46.156.2.1 void gsl_smooth::set_x (const gsl_vector * *ix*) [inline]

Definition at line 95 of file gsl_smooth.h.

46.156.2.2 int gsl_smooth::init ( ) [protected]

The documentation for this class was generated from the following file:

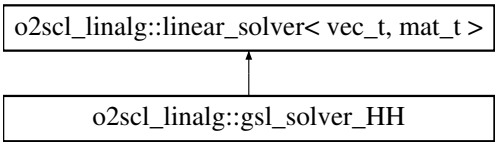
- gsl_smooth.h

46.157 o2scl_linalg::gsl_solver_HH Class Reference

GSL Householder solver.

`#include <linear_solver.h>`

Inheritance diagram for o2scl_linalg::gsl_solver_HH:



46.157.1 Detailed Description

Definition at line 174 of file linear_solver.h.

Public Member Functions

- virtual int solve (size_t n, o2scl::omatrix_base &A, o2scl::ovector_base &b, o2scl::ovector_base &x)
  Solve square linear system $Ax = b$ of size n.

The documentation for this class was generated from the following file:

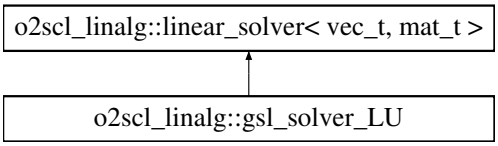
- linear_solver.h

46.158 o2scl_linalg::gsl_solver_LU Class Reference

GSL solver by LU decomposition.

`#include <linear_solver.h>`

Inheritance diagram for o2scl_linalg::gsl_solver_LU:



46.158.1 Detailed Description

Definition at line 119 of file linear_solver.h.

Public Member Functions

- virtual int solve (size_t n, o2scl::omatrix_base &A, o2scl::ovector_base &b, o2scl::ovector_base &x)
  Solve square linear system $Ax = b$ of size n.

The documentation for this class was generated from the following file:

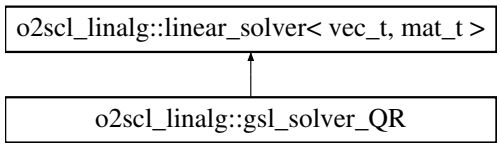
- linear_solver.h

46.159 o2scl_linalg::gsl_solver_QR Class Reference

GSL solver by QR decomposition.

`#include <linear_solver.h>`

Inheritance diagram for o2scl_linalg::gsl_solver_QR:



46.159.1 Detailed Description

Definition at line 148 of file linear_solver.h.

Public Member Functions

- virtual int solve (size_t n, o2scl::omatrix_base &A, o2scl::ovector_base &b, o2scl::ovector_base &x)
  Solve square linear system $Ax = b$ of size n.

The documentation for this class was generated from the following file:

- linear_solver.h

46.160 gsl_vector_norm Class Reference

Norm object for gsl vectors.

`#include <ovector_tlate.h>`

Inherited by ovector_const_view_tlate< double, gsl_vector_norm, gsl_block >.

46.160.1 Detailed Description

This object is principally for use inside the ovector templates e.g.. ovector_tlate . For most applications, this object need not be instantiated directly by the end-user.

For an empty vector, this function returns zero and does not call the error handler.

Definition at line 64 of file ovector_tlate.h.

Public Member Functions

- double norm () const
  Compute the norm of a gsl_vector.

The documentation for this class was generated from the following file:

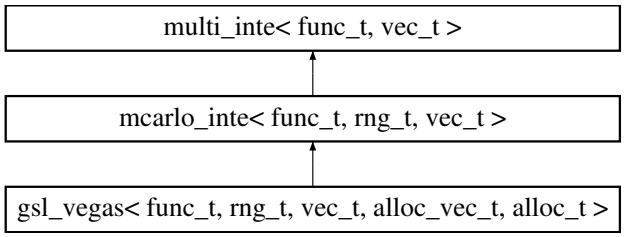
- ovector_tlate.h

46.161 gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Multidimensional integration using Vegas Monte Carlo (GSL)

`#include <gsl_vegas.h>`

Inheritance diagram for gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >:



46.161.1 Detailed Description

template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

The output options are a little different than the original GSL routine. The default setting of mcarlo_inte::verbose is 0, which turns off all output. A verbose value of 1 prints summary information about the weighted average and final result, while a value of 2 also displays the grid coordinates. A value of 3 prints information from the rebinning procedure for each iteration.

Some original documentation from GSL:

```
The input coordinates are x[j], with upper and lower limits
xu[j] and xl[j]. The integration length in the j-th direction is
delx[j]. Each coordinate x[j] is rescaled to a variable y[j] in
the range 0 to 1. The range is divided into bins with boundaries
xi[i][j], where i=0 corresponds to y=0 and i=bins to y=1. The
grid is refined (ie, bins are adjusted) using d[i][j] which is
some variation on the squared sum. A third parameter used in
defining the real coordinate using random numbers is called z.
It ranges from 0 to bins. Its integer part gives the lower index
of the bin into which a call is to be placed, and the remainder
gives the location inside the bin.

When stratified sampling is used the bins are grouped into
boxes, and the algorithm allocates an equal number of function
calls to each box.

The variable alpha controls how "stiff" the rebinning algorithm
```

```
is. alpha = 0 means never change the grid. Alpha is typically
set between 1 and 2.
```

Todo Mode = importance only doesn't give the same answer as GSL yet.

Idea for Future Prettify the verbose output

Idea for Future Allow the user to get information about the how the sampling was done, possibly by converting the bins and boxes into a structure or class.

Idea for Future Allow the user to change the maximum number of bins.

Based on Lepage78 . The current version of the algorithm was described in the Cornell preprint CLNS-80/447 of March, 1980. The GSL code follows most closely the C version by D. R. Yennie, coded in 1984.

Definition at line 110 of file gsl_vegas.h.

Public Member Functions

- virtual int allocate (size_t ldim)
  Allocate memory.
- virtual int free ()
  Free allocated memory.
- virtual int vegas_minteg_err (int stage, func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, double &res, double &err)
  Integrate function `func` from x=a to x=b.
- virtual int minteg_err (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, double &res, double &err)
  Integrate function `func` from x=a to x=b.
- virtual double minteg (func_t &func, size_t ndim, const vec_t &a, const vec_t &b)
  Integrate function `func` over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i <$ ndim-1.
- virtual const char * type ()
  Return string denoting type ("gsl_vegas")

Data Fields

- double result
  Result from last iteration.
- double sigma
  Uncertainty from last iteration.
- double alpha
  The stiffness of the rebinning algorithm (default 1.5)
- unsigned int iterations
  Set the number of iterations (default 5)
- double chisq
  The chi-squared per degree of freedom for the weighted estimate of the integral.
- std::ostream * outs
  The output stream to send output information (default `std::cout`)

Protected Member Functions

- virtual void init_box_coord (uvector_int &boxt)
  Initialize box coordinates.
- int change_box_coord (uvector_int &boxt)
  Change box coordinates.
- virtual void init_grid (const vec_t &xl, const vec_t &xu, size_t ldim)

Initialize grid.

- virtual void reset_grid_values ()
  Reset grid values.
- void accumulate_distribution (uvector_int &lbin, double y)
  Add the most recently generated result to the distribution.
- void random_point (vec_t &lx, uvector_int &lbin, double &bin_vol, const uvector_int &lbox, const vec_t &xl, const vec_t &xu)
  Generate a random position in a given box.
- virtual void resize_grid (unsigned int lbins)
  Resize the grid.
- virtual void refine_grid ()
  Refine the grid.
- virtual void print_lim (const vec_t &xl, const vec_t &xu, unsigned long ldim)
  Print limits of integration.
- virtual void print_head (unsigned long num_dim, unsigned long calls, unsigned int lit_num, unsigned int lbins, unsigned int lboxes)
  Print header.
- virtual void print_res (unsigned int itr, double res, double err, double cum_res, double cum_err, double chi_sq)
  Print results.
- virtual void print_dist (unsigned long ldim)
  Print distribution.
- virtual void print_grid (unsigned long ldim)
  Print grid.

Protected Attributes

- size_t dim
  Number of dimensions.
- unsigned int bins
  Number of bins.
- unsigned int boxes
  Number of boxes.
- uvector xi
  Boundaries for each bin.
- uvector xin
  Storage for grid refinement.
- uvector delx
  The iteration length in each direction.
- uvector weight
  The weight for each bin.
- double vol
  The volume of the current bin.
- uvector_int bin
  The bins for each direction.
- uvector_int box
  The boxes for each direction.
- uvector d
  Distribution.
- unsigned int it_start
  The starting iteration number.
- unsigned int it_num
  The total number of iterations.
- unsigned int samples
  Number of samples for computing chi squared.
- unsigned int calls_per_box
  Number of function calls per box.
- alloc_t ao
  Memory allocation object.
- alloc_vec_t x
  Point for function evaluation.

Scratch variables preserved between calls to

vegas_minteg_err()

- double **jac**
- double **wtd_int_sum**
- double **sum_wgts**
- double **chi_sum**

Static Protected Attributes

- static const size_t bins_max = 50
  Maximum number of bins.

Integration mode (default is mode_importance)

- int **mode**
- static const int **mode_importance** = 1
- static const int **mode_importance_only** = 0
- static const int **mode_stratified** = -1

46.161.2 Member Function Documentation

46.161.2.1 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::change_box_coord (uvector_int & *boxt*) [inline, protected]

Steps through the box coordinates, e.g.

```
{0,0}, {0,1}, {0,2}, {0,3}, {1,0}, {1,1}, {1,2}, ...
```

This is among the member functions that is not virtual because it is part of the innermost loop.

Definition at line 222 of file gsl_vegas.h.

46.161.2.2 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> void gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::accumulate_distribution (uvector_int & *lbin,* double *y*) [inline, protected]

This is among the member functions that is not virtual because it is part of the innermost loop.

Definition at line 274 of file gsl_vegas.h.

46.161.2.3 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> void gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::random_point (vec_t & *lx,* uvector_int & *lbin,* double & *bin_vol,* const uvector_int & *lbox,* const vec_t & *xl,* const vec_t & *xu*) [inline, protected]

Use the random number generator mcarlo_inte::def_rng to return a random position x in a given box. The value of bin gives the bin location of the random position (there may be several bins within a given box)

This is among the member functions that is not virtual because it is part of the innermost loop.

Definition at line 294 of file gsl_vegas.h.

46.161.2.4 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::vegas_minteg_err (int *stage,* func_t & *func,* size_t *ndim,* const vec_t & *xl,* const vec_t & *xu,* double & *res,* double & *err*) [inline, virtual]

Original documentation from GSL:

Normally, `stage = 0` which begins with a new uniform grid and empty weighted average. Calling vegas with `stage = 1` retains the grid from the previous run but discards the weighted average, so that one can "tune" the grid using a relatively small number of points and then do a large run with `stage = 1` on the optimized grid. Setting `stage = 2` keeps the grid and the weighted average from the previous run, but may increase (or decrease) the number of histogram bins in the grid depending on the number of calls available. Choosing `stage = 3` enters at the main loop, so that nothing is changed, and is equivalent to performing additional iterations in a previous call.

Todo Should stage be passed by reference?

There was an update between gsl-1.12 and 1.15 which has not been implemented here yet.

Definition at line 618 of file gsl_vegas.h.

46.161.3 Field Documentation

46.161.3.1 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::alpha

This usual range is between 1 and 2.

Definition at line 133 of file gsl_vegas.h.

46.161.3.2 template<class func_t = multi_funct<>, class rng_t = gsl_rnga, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >::chisq

After an integration, this should be close to 1. If it is not, then this indicates that the values of the integral from different iterations are inconsistent, and the error may be underestimated. Further iterations of the algorithm may enable one to obtain more reliable results.

Definition at line 147 of file gsl_vegas.h.

The documentation for this class was generated from the following file:

- gsl_vegas.h

46.162 hdf_file Class Reference

Store data in an HDF5 file.

`#include <hdf_file.h>`

46.162.1 Detailed Description

The member functions which write or get data from an HDF file begin with either `get` or `set`. Where appropriate, the next character is either `c` for character, `d` for double, `f` for float, or `i` for int.

The get functions are:

- getc() - get a single character
- getc_def() - get a character or set to a default value
- getc_arr() - get an array
- getc_arr_alloc() - allocate memory with `new` and get an array

The set functions are:

- setc() - set a single character

- setc_arr() - set an extensible array
- setc_arr_fixed() - set a fixed array

The string I/O functions:

- gets() - get a string
- gets_def() - get a string or set to a default value
- gets_arr() - Get an extensible string
- gets_fixed() - get a fixed-length string
- sets() - set a string
- sets_fixed() - set a fixed-length string
- sets_arr() - set an extensible array of strings

The vector I/O functions:

- setd_vec() and seti_vec() - set a vector
- getd_vec() and geti_vec() - get a vector
- getd_vec_prealloc() and geti_vec_prealloc - get a pre-allocated vector

The matrix I/O functions:

- setd_mat() and seti_mat() - set a matrix
- getd_mat() and geti_mat() - get a matrix

By default, vectors and matrices are written to HDF files in a chunked format, so their length can be changed later as necessary. The chunk size is chosen in def_chunk() to be the closest power of 10 to the current vector size.

All files not closed by the user are closed in the destructor, but the destructor does not automatically close groups.

Warning

This class is still in development. Because of this, hdf5 files generated by this class may not be easily read by future versions. Later versions of O_2scl will have stronger guarantees on backwards compatibility.

Todo Error handling with HDF functions is a particular concern. Modify to throw O_2scl exceptions when appropriate.

Idea for Future Automatically close groups, e.g. by storing hid_t's in a stack?

Definition at line 94 of file hdf_file.h.

Public Member Functions

- int getd_ten3 (std::string name, o2scl::tensor3 &t3)
  Desc.

Open and close files

- int open (std::string fname)
  Open a file named `fname`.
- int close ()
  Close the file.

Manipulate ids

- hid_t get_file_id ()
  Get the current file id.
- int set_current_id (hid_t cur)
  Set the current working id.
- hid_t get_current_id ()
  Retrieve the current working it.

Set functions

- int setc (std::string name, char c)
  Set a character named `name` to value `d`.
- int setd (std::string name, double d)
  Set a double named `name` to value `d`.
- int setf (std::string name, float f)
  Set a float named `name` to value `d`.
- int seti (std::string name, int i)
  Set an integer named `name` to value `d`.
- int sets (std::string name, std::string s)
  Set a string named `name` to value `s`.
- int sets_fixed (std::string name, std::string s)
  Set a string named `name` to value `s`.

Group manipulation

- hid_t open_group (hid_t init_id, std::string path)
  Open a group relative to the location specified in `init_id`.
- hid_t open_group (std::string path)
  Open a group relative to the current location.
- int close_group (hid_t group)
  Close a previously created group.

Vector set functions

These functions automatically write all of the vector elements to the HDF file, if necessary extending the data that is already present.

- int setd_vec (std::string name, const std::vector< double > &v)
  Set vector dataset named `name` with `v`.
- int setd_vec (std::string name, const o2scl::ovector_const_view &v)
  Set vector dataset named `name` with `v`.
- int setd_vec (std::string name, const o2scl::uvector_const_view &v)
  Set vector dataset named `name` with `v`.
- int seti_vec (std::string name, const std::vector< int > &v)
  Set vector dataset named `name` with `v`.
- int seti_vec (std::string name, const o2scl::ovector_int_const_view &v)
  Set vector dataset named `name` with `v`.
- int seti_vec (std::string name, const o2scl::uvector_int_const_view &v)
  Set vector dataset named `name` with `v`.

Vector get functions

These functions automatically free any previously allocated memory in `v` and then allocate the proper space required to read the information from the HDF file.

- int getd_vec (std::string name, std::vector< double > &v)
  Get vector dataset and place data in `v`.
- int getd_vec (std::string name, o2scl::ovector &v)
  Get vector dataset and place data in `v`.
- int getd_vec (std::string name, o2scl::uvector &v)
  Get vector dataset and place data in `v`.
- int geti_vec (std::string name, std::vector< int > &v)
  Get vector dataset and place data in `v`.
- int geti_vec (std::string name, o2scl::ovector_int &v)
  Get vector dataset and place data in `v`.
- int geti_vec (std::string name, o2scl::uvector_int &v)
  Get vector dataset and place data in `v`.

Matrix set functions

These functions automatically write all of the vector elements to the HDF file, if necessary extending the data that is already present.

- int setd_mat (std::string name, const o2scl::omatrix_const_view &m)
  Set matrix dataset named `name` with `m`.
- int setd_mat (std::string name, const o2scl::umatrix_const_view &m)
  Set matrix dataset named `name` with `m`.
- int seti_mat (std::string name, const o2scl::omatrix_int_const_view &m)
  Set matrix dataset named `name` with `m`.
- int seti_mat (std::string name, const o2scl::umatrix_int_const_view &m)
  Set matrix dataset named `name` with `m`.

Matrix get functions

These functions automatically free any previously allocated memory in `m` and then allocate the proper space required to read the information from the HDF file.

- int getd_mat (std::string name, o2scl::omatrix &m)
  Get matrix dataset and place data in `m`.
- int getd_mat (std::string name, o2scl::umatrix &m)
  Get matrix dataset and place data in `m`.
- int geti_mat (std::string name, o2scl::omatrix_int &m)
  Get matrix dataset and place data in `m`.
- int geti_mat (std::string name, o2scl::umatrix_int &m)
  Get matrix dataset and place data in `m`.

Array set functions

- int setc_arr (std::string name, size_t n, const char *c)
  Set a character array named `name` of size `n` to value `c`.
- int setd_arr (std::string name, size_t n, const double *d)
  Set a double array named `name` of size `n` to value `d`.
- int setf_arr (std::string name, size_t n, const float *f)
  Set a float array named `name` of size `n` to value `f`.
- int seti_arr (std::string name, size_t n, const int *i)
  Set a integer array named `name` of size `n` to value `i`.
- int sets_arr (std::string name, std::vector< std::string > &s)
  Set a vector of strings named `name`.

Fixed-length array set functions

If a dataset named `name` is already present, then the user-specified array must not be longer than the array already present in the HDF file.

- int setc_arr_fixed (std::string name, size_t n, const char *c)
  Set a character array named `name` of size `n` to value `c`.
- int setd_arr_fixed (std::string name, size_t n, const double *c)
  Set a double array named `name` of size `n` to value `d`.
- int setf_arr_fixed (std::string name, size_t n, const float *f)
  Set a float array named `name` of size `n` to value `f`.
- int seti_arr_fixed (std::string name, size_t n, const int *i)
  Set an integer array named `name` of size `n` to value `i`.

Get functions

If the specified object is not found, the O_2scl error handler will be called.

- int getc (std::string name, char &c)
  Get a character named `name`.
- int getd (std::string name, double &d)
  Get a double named `name`.
- int getf (std::string name, float &f)

Get a float named `name`.
- int geti (std::string name, int &i)
  Get a integer named `name`.
- int gets (std::string name, std::string &s)
  Get a string named `name`.
- int gets_fixed (std::string name, std::string &s)
  Get a string named `name`.
- int gets_def_fixed (std::string name, std::string def, std::string &s)
  Get a string named `name`.

Array get functions

All of these functions (except gets_arr()) assume that the pointer allocated beforehand, and matches the size of the array in the HDF file. If the specified object is not found, the O_2scl error handler will be called.

- int getc_arr (std::string name, size_t n, char *c)
  Get a character array named `name` *of size* `n`.
- int getd_arr (std::string name, size_t n, double *d)
  Get a double array named `name` *of size* `n`.
- int getf_arr (std::string name, size_t n, float *f)
  Get a float array named `name` *of size* `n`.
- int geti_arr (std::string name, size_t n, int *i)
  Get an integer array named `name` *of size* `n`.
- int gets_arr (std::string name, std::vector< std::string > &s)
  Get a string array named `name` *and store it in* `s`.

Array get functions with memory allocation

These functions allocate memory with `new`*, which should be freed by the user with* `delete`.

- int getc_arr_alloc (std::string name, size_t &n, char *c)
  Get a character array named `name` *of size* `n`.
- int getd_arr_alloc (std::string name, size_t &n, double *d)
  Get a double array named `name` *of size* `n`.
- int getf_arr_alloc (std::string name, size_t &n, float *f)
  Get a float array named `name` *of size* `n`.
- int geti_arr_alloc (std::string name, size_t &n, int *i)
  Get an integer array named `name` *of size* `n`.

Get functions with default values

If the requested dataset is not found in the HDF file, the object is set to the specified default value and the error handler is not called.

- int getc_def (std::string name, char def, char &c)
  Get a character named `name`.
- int getd_def (std::string name, double def, double &d)
  Get a double named `name`.
- int getf_def (std::string name, float def, float &f)
  Get a float named `name`.
- int geti_def (std::string name, int def, int &i)
  Get a integer named `name`.
- int gets_def (std::string name, std::string def, std::string &s)
  Get a string named `name`.

Get functions with pre-allocated pointer

- int getd_vec_prealloc (std::string name, size_t n, double *d)
  Get a double array `d` *pre-allocated to have size* `n`.
- int geti_vec_prealloc (std::string name, size_t n, int *i)
  Get an integer array `i` *pre-allocated to have size* `n`.
- int getd_mat_prealloc (std::string name, size_t n, size_t m, double *d)
  Get a double matrix `d` *pre-allocated to have size* `(n,m)`
- int geti_mat_prealloc (std::string name, size_t n, size_t m, int *i)
  Get an integer matrix `i` *pre-allocated to have size* `(n,m)`

Protected Member Functions

- virtual hsize_t def_chunk (size_t n)
  Default chunk size.

Protected Attributes

- hid_t file
  File ID.
- bool file_open
  True if a file has been opened.
- hid_t current
  Current file or group location.

46.162.2 Member Function Documentation

46.162.2.1 virtual hsize_t hdf_file::def_chunk (size_t *n*) [inline, protected, virtual]

Choose the closest power of 10 which is greater than or equal to 10 and less than or equal to 10^6.

Definition at line 114 of file hdf_file.h.

46.162.2.2 int hdf_file::sets (std::string *name*, std::string *s*)

The string is stored in the HDF file as an extensible character array rather than a string.

46.162.2.3 int hdf_file::sets_fixed (std::string *name*, std::string *s*)

This function stores `s` as a fixed-length string in the HDF file. If a dataset named `name` is already present, then `s` must not be longer than the string length already specified in the HDF file.

46.162.2.4 hid_t hdf_file::open_group (hid_t *init_id*, std::string *path*)

Note

In order to ensure that future objects are written to the newly-created group, the user must use set_current_id() using the newly-created group ID for the argument.

46.162.2.5 hid_t hdf_file::open_group (std::string *path*)

Note

In order to ensure that future objects are written to the newly-created group, the user must use set_current_id() using the newly-created group ID for the argument.

46.162.2.6 int hdf_file::setd_vec (std::string *name*, const o2scl::ovector_const_view & *v*)

Note

In the case that the stride of the vector `v` is not unity, this function requires copying the data twice: once from the vector to a pointer, and once from the pointer to the file. Keep this in mind when performing I/O with larger data sets.

46.162.2.7 int hdf_file::seti_vec (std::string *name*, const o2scl::ovector_int_const_view & *v*)

Note

In the case that the stride of the vector `v` is not unity, this function requires copying the data twice: once from the vector to a pointer, and once from the pointer to the file. Keep this in mind when performing I/O with larger data sets.

46.162.2.8 int hdf_file::setd_mat (std::string *name*, const o2scl::omatrix_const_view & *m*)

Note

In the case that the trailing dimension (as reported by omatrix_const_view_tlate::tda()) of the matrix `m` is greater than the number of columns, this function requires copying the data twice: once from the matrix to a pointer, and once from the pointer to the file. Keep this in mind when performing I/O with larger data sets.

46.162.2.9 int hdf_file::seti_mat (std::string *name*, const o2scl::omatrix_int_const_view & *m*)

Note

In the case that the trailing dimension (as reported by omatrix_const_view_tlate::tda()) of the matrix `m` is greater than the number of columns, this function requires copying the data twice: once from the matrix to a pointer, and once from the pointer to the file. Keep this in mind when performing I/O with larger data sets.

46.162.2.10 int hdf_file::sets_arr (std::string *name*, std::vector< std::string > & *s*)

Developer note:

String vectors are reformatted as a single character array, in order to allow each string to have different length and to make each string extensible. The size of the vector `s` is stored as an integer named `nw`.

46.162.2.11 int hdf_file::gets (std::string *name*, std::string & *s*)

Note

Strings are stored as character arrays and thus retrieving a string from a file requires loading the information from the file into a character array, and then copying it to the string. This will be slow for very long strings.

46.162.2.12 int hdf_file::getc_arr (std::string *name*, size_t *n*, char * *c*)

Note

The pointer `c` must be allocated beforehand to hold `n` entries, and `n` must match the size of the array in the HDF file.

46.162.2.13 int hdf_file::getd_arr (std::string *name*, size_t *n*, double * *d*)

Note

The pointer `d` must be allocated beforehand to hold `n` entries, and `n` must match the size of the array in the HDF file.

46.162.2.14 int hdf_file::getf_arr (std::string *name*, size_t *n*, float * *f*)

Note

The pointer `f` must be allocated beforehand to hold `n` entries, and `n` must match the size of the array in the HDF file.

46.162.2.15 int hdf_file::geti_arr (std::string *name*, size_t *n*, int * *i*)

Note

The pointer `i` must be allocated beforehand to hold `n` entries, and `n` must match the size of the array in the HDF file.

The documentation for this class was generated from the following file:

- hdf_file.h

46.163 hist Class Reference

A one-dimensional histogram class.

`#include <hist.h>`

46.163.1 Detailed Description

Experimental.

One may set the histogram bins using set_bins() or one may manually set the limit of one bin using the reference returned by get_bin_low(), get_bin_low_i(), get_bin_high(), or get_bin_high_i(). Note that if one attempts to set the bins on a histogram where the bins have already been set, one must ensure that the new and old binnings have the same size. This requirement is designed to help prevent accidental data loss.

The upper edge of bin `i` is always equal to the lower edge of bin `i+1`.

By convention, all functions which take a bin index as an argument have an extra `"_i"` suffix to distinguish them from functions which take a floating-point value to be binned as their argument.

Empty histograms have zero size.

The set_bins() functions can only be used if (i) the current histogram is empty or (ii) the new number of bins is equal to the histogram size so that no reallocation is necessary.

To save space, representative vectors are not allocated until they are used.

Bin Representatives

One way to operate on a histogram as a function is to designate a particular value inside each bin which represents the coordinate associated with that bin. These "bin representative values" are automatically created and can be used by the function evaluation and interpolation functions operator(double), interp(), deriv(), integ() for some particular interpolation type. By default, these representative values are taken to be the midpoint of each bin, but this option is configurable and the representative values may be set by the user for each individual bin.

Todo Check that the actions of set_bins(), clear() and other functions perform the correct actions on the user_rep vector. I'm a bit concerned that set_bins_auto() shouldn't call user_rep.allocate().

Todo More documentation and testing.

Idea for Future Would be nice not to have to create a new search_vec object in get_bin_index().

Idea for Future Consider adding the analogs of the GSL histogram sampling functions

Idea for Future Add a function which computes the bin sizes?

Definition at line 94 of file hist.h.

Public Member Functions

- hist (const hist &h)
  Copy constructor.
- hist & operator= (const hist &h)
  Copy constructor.
- size_t size () const
  The histogram size.

- double get_max_wgt ()
  Get maximum weight.
- size_t get_max_index ()
  Get the index of the maximum weight.
- double get_max_rep ()
  Get the representative for the bin with maximum weight.

Initial bin setup

- int set_bins (uniform_grid< double > g)
  Set bins from a uniform_grid object.
- template<class vec_t >
  int set_bins (size_t n, const vec_t &v)
  Set the bins from a vector.

Weight functions

- int update (double x, double val=1.0)
  Increment bin for `x` by value `val`.
- int update_i (size_t i, double val=1.0)
  Increment bin with index `i` by value `val`.
- const double & get_wgt_i (size_t i) const
  Return contents of bin with index `i`.
- double & get_wgt_i (size_t i)
  Return contents of bin with index `i`.
- const double & get_wgt (double x) const
  Return contents of bin for `x`.
- double & get_wgt (double x)
  Return contents of bin for `x`.
- int set_wgt_i (size_t i, double val)
  Set contents of bin with index `i` to value `val`.
- int set_wgt (double x, double val)
  Set contents of bin for `x` to value `val`.
- const uvector & get_wgts () const
  Get a reference to the full y vector.
- const double & operator[ ] (size_t i) const
  Get a reference to the weight for the bin at index `i`.
- double & operator[ ] (size_t i)
  Get a reference to the weight for the bin at index `i`.

Bin manipulation

- size_t get_bin_index (double x) const
  Get the index of the bin which holds `x`.
- double & get_bin_low_i (size_t i)
  Get the lower edge of bin of index `i`.
- const double & get_bin_low_i (size_t i) const
  Get the lower edge of bin of index `i`.
- double & get_bin_high_i (size_t i)
  Get the upper edge of bin of index `i`.
- const double & get_bin_high_i (size_t i) const
  Get the upper edge of bin of index `i`.
- double & get_bin_low (double x)
  Get the lower edge of bin of index `i`.
- const double & get_bin_low (double x) const
  Get the lower edge of bin of index `i`.
- double & get_bin_high (double x)
  Get the upper edge of bin of index `i`.
- const double & get_bin_high (double x) const
  Get the upper edge of bin of index `i`.
- const uvector & get_bins () const
  Get a reference to the full vector of bin specifications.

Delete functions

- int clear_wgts ()
  Clear the data, but leave the bins as is.
- int clear ()
  Clear the entire histogram.

Representative functions

- template<class vec_t >
  int set_reps (size_t n, const vec_t &v)
  Set the representative x-values for each bin.
- int set_rep_mode (size_t mode)
  Set mode used to compute bin representatives.
- size_t get_rep_mode () const
  Get mode used to compute bin representatives.
- double & get_rep_i (size_t i)
  Return the representative of bin of index `i`.
- double & get_rep (double x)
  Return the representative of bin containing `x`.
- const uvector & get_reps ()
  Get a reference to the full representative vector.
- const uvector & get_user_reps () const
  Get a reference to the full data vector.

Evaluation and interpolation functions

- double operator() (double x)
  Return the value of the function at `x`.
- double interp (double x)
  Return the value of the function at `x`.
- double deriv (double x)
  Return the derivative of the function at `x`.
- double deriv2 (double x)
  Return the second derivative of the function at `x`.
- double integ (double x, double y)
  Return the integral of the function between `x` and `y`.
- int set_interp (base_interp_mgr< uvector_const_view > &bi1, base_interp_mgr< uvector_const_subvector > &bi2)
  Set the base interpolation objects.

Data Fields

- bool extend_rhs
  If true, allow abcissa larger than largest bin limit to correspond to the highest bin (default false)

Static Public Attributes

Rep modes (default is \c rmode_avg)

- static const size_t **rmode_avg** = 0
- static const size_t **rmode_user** = 1
- static const size_t **rmode_low** = 2
- static const size_t **rmode_high** = 3
- static const size_t **rmode_gmean** = 4

Protected Types

- typedef o2scl_interp_vec < uvector_const_view > interp_t
  Interpolation typedef.

Protected Member Functions

- void set_reps_auto ()
  Set the representative array according to current rmode.
- void allocate (size_t n)
  Allocate vectors for a histogram of size `n`.

Protected Attributes

- uvector ubin
  Bin locations (N+1)
- uvector uwgt
  Bin contents (N)
- uvector urep
  Bin representative values (N)
- uvector user_rep
  User-defined representative values (N)
- size_t hsize
  Number of bins.
- size_t rmode
  Representative mode.
- base_interp_mgr < uvector_const_view > * bim1
  A pointer to the interpolation manager.
- base_interp_mgr < uvector_const_subvector > * bim2
  A pointer to the subvector interpolation manager.
- def_interp_mgr < uvector_const_view, cspline_interp > dim1
  Default interpolation manager.
- def_interp_mgr < uvector_const_subvector, cspline_interp > dim2
  Default interpolation manager.

46.163.2 Member Function Documentation

46.163.2.1 void hist::allocate (size_t *n*) [protected]

This function also sets all the weights to zero.

46.163.2.2 int hist::set_bins (uniform_grid< double > *g*)

Note

If the size of the histogram uniform_grid is not equal to the current histogram size, this function requires the histogram be empty so it can reallocate the vectors for both the bins and the weights.

46.163.2.3 template<class vec_t > int hist::set_bins (size_t *n*, const vec_t & *v*) [inline]

Note

If `n` is not equal to the current histogram size, this function requires the histogram be empty so it can reallocate the vectors for both the bins and the weights.

Definition at line 179 of file hist.h.

The documentation for this class was generated from the following file:

- hist.h

46.164 hist_2d Class Reference

A two-dimensional histogram class.

`#include <hist_2d.h>`

46.164.1 Detailed Description

Experimental.

Idea for Future Write a function to create a 1-d histogram from a 2-d histogram either by selecting one bin in one axis or by marginalizing over one direction.

Definition at line 45 of file hist_2d.h.

Public Member Functions

- **hist_2d** (const hist_2d &h)
- hist_2d & **operator=** (const hist_2d &h)
- int setup_interp (twod_intp &ti, bool x_first=true)
  Set up a twod_intp object for interpolation.

Initial bin setup

- int **set_bins** (uniform_grid< double > gx, uniform_grid< double > gy)
- template<class vec_t >
  int set_bins (size_t nx, vec_t &vx, size_t ny, vec_t &vy)
  Set the bins from a vector.

Weight functions

- int update_i (size_t i, size_t j, double val=1.0)
  Increment bin for `x` by value `val`.
- int update (double x, double y, double val=1.0)
  Increment bin for `x` by value `val`.
- const double & get_wgt_i (size_t i, size_t j) const
  Return contents of bin with index `i`.
- const double & get_wgt (double x, double y) const
  Return contents of bin for `x`.
- double & get_wgt_i (size_t i, size_t j)
  Return contents of bin with index `i`.
- double & get_wgt (double x, double y)
  Return contents of bin for `x`.
- int set_wgt_i (size_t i, size_t j, double val)
  Set contents of bin with index `i` to value `val`.
- int set_wgt (double x, double y, double val)
  Set contents of bin for `x` to value `val`.
- const omatrix & get_wgts () const
  Get a const reference to the full data vector.
- omatrix & get_wgts ()
  Get a reference to the full data vector.

Delete functions

- int clear_wgts ()
  Clear the data, but leave the bins as is.
- int clear ()
  Clear the entire histogram.

Bin manipulation

- void get_bin_indices (double x, double y, size_t &i, size_t &j) const
  Get the index of the bin which holds `x`.
- size_t get_x_bin_index (double x) const
  Get the index of the bin which holds `x`.
- size_t get_y_bin_index (double y) const
  Get the indey of the bin which holds `y`.
- double & get_x_low_i (size_t i)
  Get the lower edge of bin of index `i`.
- const double & get_x_low_i (size_t i) const
  Get the lower edge of bin of index `i`.
- double & get_x_high_i (size_t i)
  Get the upper edge of bin of index `i`.
- const double & get_x_high_i (size_t i) const
  Get the upper edge of bin of index `i`.
- double & get_y_low_i (size_t j)
  Get the lower edge of bin of index `j`.
- const double & get_y_low_i (size_t j) const
  Get the lower edge of bin of index `j`.
- double & get_y_high_i (size_t j)
  Get the upper edge of bin of index `j`.
- const double & get_y_high_i (size_t j) const
  Get the upper edge of bin of index `j`.

Rep functions

- template<class vec_t >
  int set_reps (size_t nx, vec_t &vx, size_t ny, vec_t &vy)
  Set the representative x-values for each bin.
- template<class vec_t >
  int set_x_reps (size_t nx, vec_t &vx)
  Set the representative x-values for each bin.
- template<class vec_t >
  int set_y_reps (size_t ny, vec_t &vy)
  Set the representative y-values for each bin.
- int set_rep_mode (size_t x_mode, size_t y_mode)
  Set mode used to compute bin reps.
- size_t get_x_rep_mode () const
  Get mode used to compute bin reps.
- size_t get_y_rep_mode () const
  Get mode used to compute bin reps.
- const ovector & get_x_bins () const
  Get a reference to the full vector of bin specifications.
- const ovector & get_y_bins () const
  Get a reference to the full vector of bin specifications.
- size_t size_x () const
  Return the histogram size of the x coordinate.
- size_t size_y () const
  Return the histogram size of the y coordinate.
- const ovector & get_user_reps_x () const
  Get a reference to the user-specified reps for x coordinates.
- const ovector & get_user_reps_y () const
  Get a reference to the user-specified reps for y coordinates.
- double & get_x_rep_i (size_t i)
  Return the rep of bin of index `i`.
- const double & get_x_rep_i (size_t i) const
  Return the rep of bin of index `i`.
- double & get_y_rep_i (size_t j)
  Return the rep of bin of index `j`.
- const double & get_y_rep_i (size_t j) const
  Return the rep of bin of index `j`.

Data Fields

- bool extend_rhs
  If true, allow abcissa larger than largest bin limit to correspond to the highest bin (default false).

Static Public Attributes

Rep modes (default is \c rmode_avg)

- static const size_t **rmode_avg** = 0
- static const size_t **rmode_user** = 1
- static const size_t **rmode_low** = 2
- static const size_t **rmode_high** = 3
- static const size_t **rmode_gmean** = 4

Protected Member Functions

- void allocate (size_t nx, size_t ny)
  Allocate for a histogram of size `nx`, `ny`.
- void set_reps_auto ()
  An internal function to automatically set xrep and yrep.

Protected Attributes

- ovector xa
  Bin locations (Nx+1)
- ovector ya
  Bin locations (Ny+1)
- omatrix wgt
  Values (Nx,Ny)
- ovector xrep
  "Central" values for x-axis (N)
- ovector yrep
  "Central" values for y-axis (N)
- ovector user_xrep
  User-defined central values for x-axis (N)
- ovector user_yrep
  User-defined central values for y-axis (N)
- size_t hsize_x
  Number of x-bins.
- size_t hsize_y
  Number of y-bins.
- size_t xrmode
  Rep mode for x.
- size_t yrmode
  Rep mode for y.
- base_interp_mgr < ovector_const_view > * bim1
  A pointer to the interpolation manager.
- base_interp_mgr < ovector_const_subvector > * bim2
  A pointer to the subvector interpolation manager.
- def_interp_mgr < ovector_const_view, cspline_interp > dim1
  Default interpolation manager.
- def_interp_mgr < ovector_const_subvector, cspline_interp > dim2
  Default interpolation manager.

46.164.2 Member Function Documentation

46.164.2.1 void hist_2d::allocate (size_t *nx*, size_t *ny*) [protected]

This function also sets all the weights to zero.

The documentation for this class was generated from the following file:

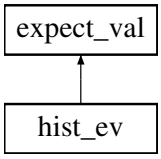
- hist_2d.h

46.165 hist_ev Class Reference

A set expectation values for histogram bins.

`#include <hist_ev.h>`

Inheritance diagram for hist_ev:



46.165.1 Detailed Description

See expect_val for some general notes on this and related classes.

This class is experimental.

This class computes expectation values of bins in a histogram. It is most useful in cases where one does not know a priori how many measurements one is going to get for each bin. The class automatically arranges each bin into blocks, so that the standard deviation and error and the average can be computed even when not all bins have been filled with the same number of measurements.

Note

This class always assumes that `n_per_block` is zero. See expect_val for more details.

Todo Test set_grid_blocks() function.

Todo Create copy constructors as in the scalar_ev class.

Definition at line 58 of file hist_ev.h.

Public Member Functions

- hist_ev (uniform_grid< double > g, size_t n_blocks)
  Create a histogram expectation value.
- void reset ()
  Remove all currently stored data, but keep same block size.
- void set_grid_blocks (uniform_grid< double > g, size_t n_blocks)
  Set the histogram grid and the number of blocks.
- virtual void add (double x, double val)
  Add measurement of value `val` at location `x`.
- virtual void current_avg_stats (uvector &reps, uvector &avg, uvector &std_dev, uvector &avg_err, uvector_int &m_block, uvector_int &m_per_block)

Report current average, standard deviation, and the error in the average.

- virtual void current_avg (uvector &reps, uvector &avg, uvector &std_dev, uvector &avg_err)
  Report current average, standard deviation, and the error in the average.

Protected Attributes

- umatrix last
  Last measurement for each block and each bin.
- umatrix vals
  Running average for each block and each bin.
- uvector_int iblock_bins
  The value of `iblock` for each bin.
- uvector_int i_bins
  The value of `i` for each bin.
- size_t hsize
  This should always be the same as the size as the histogram.
- hist h
  The associated histogram.

46.165.2 Member Function Documentation

46.165.2.1 virtual void hist_ev::current_avg_stats (uvector & *reps*, uvector & *avg*, uvector & *std_dev*, uvector & *avg_err*, uvector_int & *m_block*, uvector_int & *m_per_block*) [virtual]

This function deallocates any space already allocated for the vector parameters and reallocates space as needed. Information previously stored in these vectors will be lost.

46.165.2.2 virtual void hist_ev::current_avg (uvector & *reps*, uvector & *avg*, uvector & *std_dev*, uvector & *avg_err*) [virtual]

This function deallocates any space already allocated for the vector parameters and reallocates space as needed. Information previously stored in these vectors will be lost.

46.165.3 Field Documentation

46.165.3.1 hist hist_ev::h [protected]

This object is only currently used internally for binning. No data is added to it.

Definition at line 84 of file hist_ev.h.

The documentation for this class was generated from the following file:

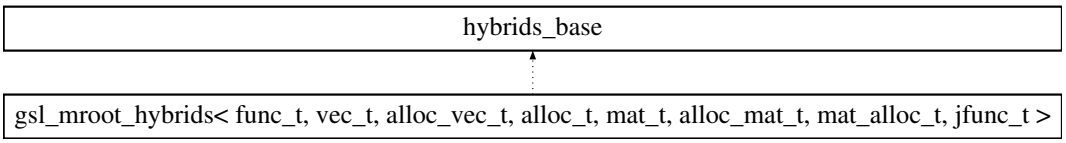
- hist_ev.h

46.166 hybrids_base Class Reference

Base functions for gsl_mroot_hybrids.

`#include <gsl_mroot_hybrids.h>`

Inheritance diagram for hybrids_base:



46.166.1 Detailed Description

Definition at line 62 of file gsl_mroot_hybrids.h.

Public Member Functions

- double compute_actual_reduction (double fnorm0, double fnorm1)
  Compute the actual reduction.
- double compute_predicted_reduction (double fnorm0, double fnorm1)
  Compute the predicted reduction phi1p=|Q^T f + R dx|.
- template<class vec2_t >
  void compute_Rg (size_t N, const omatrix_base &r2, const ovector_base &gradient2, vec2_t &Rg)
  Compute $R \cdot g$ where g is the gradient .
- template<class vec2_t >
  void compute_wv (size_t n, const ovector_base &qtdf2, const ovector_base &rdx2, const vec2_t &dx2, const ovector_base &diag2, double pnorm, ovector_base &w2, ovector_base &v2)
  Compute w and v.
- template<class vec2_t >
  void compute_rdx (size_t N, const omatrix_base &r2, const vec2_t &dx2, ovector_base &rdx2)
  Compute $R \cdot$ dx.
- template<class vec2_t , class vec3_t >
  double scaled_enorm (size_t n, const vec2_t &d, const vec3_t &ff)
  Compute the norm of the vector $\vec{v}$ defined by $v_i = d_i f f_i$.
- template<class vec2_t >
  double compute_delta (size_t n, ovector_base &diag2, vec2_t &x2)
  Compute delta.
- template<class vec2_t >
  double enorm (size_t N, const vec2_t &ff)
  Compute the Euclidean norm of ff.
- double enorm_sum (size_t n, const ovector_base &a, const ovector_base &b)
  Compute the Euclidean norm of the sum of a and b.
- template<class vec2_t >
  int compute_trial_step (size_t N, vec2_t &xl, vec2_t &dxl, vec2_t &xx_trial)
  Compute a trial step and put the result in xx_trial.
- template<class vec2_t >
  int compute_df (size_t n, const vec2_t &ff_trial, const vec2_t &fl, ovector_base &dfl)
  Compute the change in the function value.
- template<class mat2_t >
  void compute_diag (size_t n, const mat2_t &J2, ovector_base &diag2)
  Compute diag, the norm of the columns of the Jacobian.
- template<class vec2_t , class vec3_t , class vec4_t >
  void compute_qtf (size_t N, const vec2_t &q2, const vec3_t &f2, vec4_t &qtf2)
  Compute $Q^T f$.
- template<class mat2_t >
  void update_diag (size_t n, const mat2_t &J2, ovector_base &diag2)
  Update diag.
- template<class vec2_t >
  void scaled_addition (size_t N, double alpha, ovector_base &newton2, double beta, ovector_base &gradient2, vec2_t &pp)
  Form appropriate convex combination of the Gauss-Newton direction and the scaled gradient direction.
- int newton_direction (const size_t N, const omatrix_base &r2, const ovector_base &qtf2, ovector_base &p)
  Compute the Gauss-Newton direction.
- void gradient_direction (const size_t M, const size_t N, const omatrix_base &r2, const ovector_base &qtf2, const ovector_base &diag2, ovector_base &g)
  Compute the gradient direction.
- void minimum_step (const size_t N, double gnorm, const ovector_base &diag2, ovector_base &g)
  Compute the point at which the gradient is minimized.
- template<class vec2_t >
  int dogleg (size_t n, const omatrix_base &r2, const ovector_base &qtf2, const ovector_base &diag2, double delta2, ovector_base &newton2, ovector_base &gradient2, vec2_t &p)

Take a dogleg step.

46.166.2 Member Function Documentation

46.166.2.1 template<class vec2_t > double hybrids_base::enorm (size_t *N*, const vec2_t & *ff*) [inline]

Idea for Future Replace this with `dnrm2` from cblas_base.h

Definition at line 162 of file gsl_mroot_hybrids.h.

46.166.2.2 template<class vec2_t > int hybrids_base::compute_trial_step (size_t *N*, vec2_t & *xl*, vec2_t & *dxl*, vec2_t & *xx_trial*) [inline]

Idea for Future Replace with daxpy.

Definition at line 187 of file gsl_mroot_hybrids.h.

46.166.2.3 template<class vec2_t > int hybrids_base::compute_df (size_t *n*, const vec2_t & *ff_trial*, const vec2_t & *fl*, ovector_base & *dfl*) [inline]

Idea for Future Replace with daxpy?

Definition at line 200 of file gsl_mroot_hybrids.h.

46.166.2.4 template<class vec2_t , class vec3_t , class vec4_t > void hybrids_base::compute_qtf (size_t *N*, const vec2_t & *q2*, const vec3_t & *f2*, vec4_t & *qtf2*) [inline]

Idea for Future This is just right-multiplication, so we could use the O_2scl cblas routines instead.

Definition at line 233 of file gsl_mroot_hybrids.h.

46.166.2.5 template<class vec2_t > void hybrids_base::scaled_addition (size_t *N*, double *alpha*, ovector_base & *newton2*, double *beta*, ovector_base & *gradient2*, vec2_t & *pp*) [inline]

Using the Gauss-Newton direction given in `newton` (a vector of size N), and the gradient direction in `gradient` (also a vector of size N), this computes

$$\mathrm{pp} = \alpha \mathrm{newton} + \beta \mathrm{gradient}$$

Definition at line 279 of file gsl_mroot_hybrids.h.

46.166.2.6 template<class vec2_t > int hybrids_base::dogleg (size_t *n*, const omatrix_base & *r2*, const ovector_base & *qtf2*, const ovector_base & *diag2*, double *delta2*, ovector_base & *newton2*, ovector_base & *gradient2*, vec2_t & *p*) [inline]

Given the QR decomposition of an `n` by `n` matrix "A", a diagonal matrix `diag`, a vector `b`, and a positive number `delta`, this function determines the convex combination `x` of the Gauss-Newton and scaled gradient directions which minimizes $Ax - b$ in the least squares sense, subject to the restriction that the Euclidean norm of dx is smaller than the value of `delta`.

Definition at line 344 of file gsl_mroot_hybrids.h.

The documentation for this class was generated from the following file:

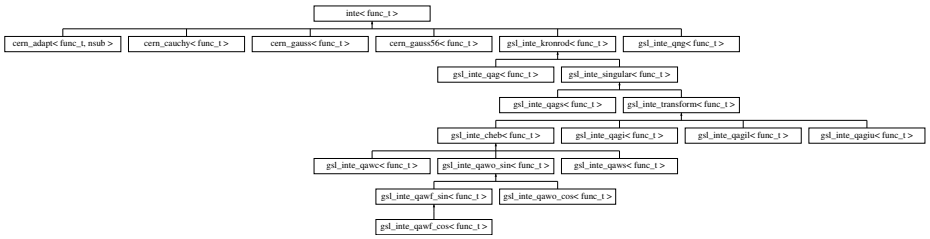
- gsl_mroot_hybrids.h

46.167 inte< func_t > Class Template Reference

Base integration class [abstract base].

`#include <inte.h>`

Inheritance diagram for inte< func_t >:



46.167.1 Detailed Description

template<class func_t = funct>class inte< func_t >

Idea for Future It might be useful to have all of the integration classes report the number of function evaluations used in addition to the number of iterations which were taken

Definition at line 41 of file inte.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b)
  Integrate function `func` from `a` to `b`.
- virtual int integ_err (func_t &func, double a, double b, double &res, double &err)=0
  Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.
- double get_error ()
  Return the numerically estimated error in the result from the last call to integ()
- virtual const char * type ()
  Return string denoting type ("inte")

Data Fields

- int verbose
  Verbosity.
- size_t last_iter
  The most recent number of iterations taken.
- double tol_rel
  The maximum relative uncertainty in the value of the integral (default 10^{-8})
- double tol_abs
  The maximum absolute uncertainty in the value of the integral (default 10^{-8})
- bool err_nonconv
  If true, call the error handler if the routine does not converge or reach the desired tolerance (default true)
- int last_conv
  Integer indicating convergence properties of the last call to integ() or integ_err().

Protected Attributes

- double interror
  The uncertainty for the last integration computation.

46.167.2 Member Function Documentation

46.167.2.1 template<class func_t = funct> double inte< func_t >::get_error ( ) [inline]

This will quietly return zero if no integrations have been performed or if the integrator does not estimate the error.

Definition at line 111 of file inte.h.

46.167.3 Field Documentation

46.167.3.1 template<class func_t = funct> bool inte< func_t >::err_nonconv

If this is false, the function proceeds normally and may provide convergence information in last_conv.

Definition at line 79 of file inte.h.

46.167.3.2 template<class func_t = funct> int inte< func_t >::last_conv

This is zero if the last call to integ() or integ_err() converged successfully. This variable is particularly useful, in combination with setting err_nonconv to false, to test for non-convergence without calling the error handler.

Definition at line 89 of file inte.h.

The documentation for this class was generated from the following file:

- inte.h

46.168 iterate_parms Struct Reference

A structure to pass information to and from iterate_func()

`#include <hdf_io.h>`

46.168.1 Detailed Description

Definition at line 103 of file hdf_io.h.

Data Fields

- hdf_file * **hf**
- std::string **type**
- std::string **group_name**
- bool **found**
- int **verbose**

The documentation for this struct was generated from the following file:

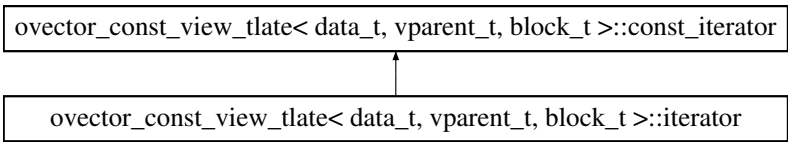
- hdf_io.h

46.169 ovector_const_view_tlate< data_t, vparent_t, block_t >::iterator Class Reference

An iterator for ovectors.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_const_view_tlate< data_t, vparent_t, block_t >::iterator:



46.169.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_view_tlate< data_t, vparent_t, block_t >::iterator

Note

Experimental.

Definition at line 223 of file ovector_tlate.h.

Public Member Functions

- iterator (const const_iterator &cit)
  Copy constructor.
- const data_t operator* () const
  Dereference - return the corresponding vector element.
- data_t operator* ()
  Dereference - return the corresponding vector element.

Protected Member Functions

- iterator (data_t *p, size_t s)
  Internally create an iterator directly from a pointer.

Friends

- class ovector_const_view_tlate< data_t, vparent_t, block_t >
  Grant access to ovector classes for begin() and end() functions.
- class ovector_base_tlate< data_t, vparent_t, block_t >
  Grant access to ovector classes for begin() and end() functions.

The documentation for this class was generated from the following file:

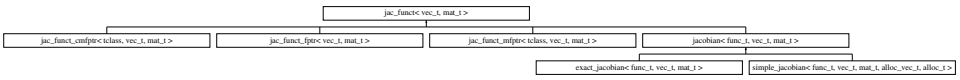
- ovector_tlate.h

46.170 jac_funct< vec_t, mat_t > Class Template Reference

Base for a square Jacobian where J is computed at x given y=f(x) [abstract base].

`#include <jacobian.h>`

Inheritance diagram for jac_funct< vec_t, mat_t >:



46.170.1 Detailed Description

template<class vec_t = ovector_base, class mat_t = omatrix_base>class jac_funct< vec_t, mat_t >

Compute

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

The `vec_t` objects in operator() could have been written to be `const`, but they are not `const` so that they can be used as temporary workspace. They are typically restored to their original values before operator() exits. The Jacobian is stored in the order `J`[i][j], i.e. the rows have index `i` and the columns with index `j`.

Definition at line 52 of file jacobian.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j)=0
  The operator()

The documentation for this class was generated from the following file:

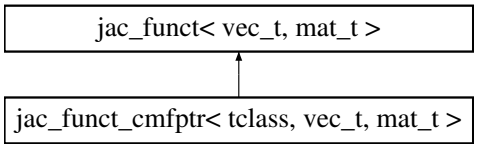
- jacobian.h

46.171 jac_funct_cmfptr< tclass, vec_t, mat_t > Class Template Reference

Const member function pointer to a Jacobian.

`#include <jacobian.h>`

Inheritance diagram for jac_funct_cmfptr< tclass, vec_t, mat_t >:



46.171.1 Detailed Description

template<class tclass, class vec_t = ovector_base, class mat_t = omatrix_base>class jac_funct_cmfptr< tclass, vec_t, mat_t >

Definition at line 166 of file jacobian.h.

Public Member Functions

- jac_funct_cmfptr (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y, mat_t &j) const)
  Specify the member function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j)
  The operator()

Protected Attributes

- int(tclass::* fptr)(size_t nv, vec_t &x, vec_t &y, mat_t &j) const
  Member function pointer.
- tclass * tptr
  Class pointer.

The documentation for this class was generated from the following file:

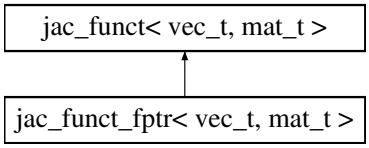
- jacobian.h

46.172 jac_funct_fptr< vec_t, mat_t > Class Template Reference

Function pointer to jacobian.

`#include <jacobian.h>`

Inheritance diagram for jac_funct_fptr< vec_t, mat_t >:



46.172.1 Detailed Description

template<class vec_t = ovector_base, class mat_t = omatrix_base>class jac_funct_fptr< vec_t, mat_t >

Definition at line 78 of file jacobian.h.

Public Member Functions

- jac_funct_fptr (int(*fp)(size_t nv, vec_t &x, vec_t &y, mat_t &j))
  Specify the function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j)
  The operator()

The documentation for this class was generated from the following file:

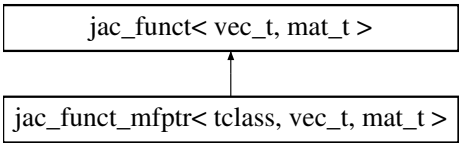
- jacobian.h

46.173 jac_funct_mfptr< tclass, vec_t, mat_t > Class Template Reference

Member function pointer to a Jacobian.

`#include <jacobian.h>`

Inheritance diagram for jac_funct_mfptr< tclass, vec_t, mat_t >:



46.173.1 Detailed Description

template<class tclass, class vec_t = ovector_base, class mat_t = omatrix_base>class jac_funct_mfptr< tclass, vec_t, mat_t >

Definition at line 118 of file jacobian.h.

Public Member Functions

- jac_funct_mfptr (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &y, mat_t &j))
  Specify the member function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j)
  The operator()

Protected Attributes

- int(tclass::* fptr)(size_t nv, vec_t &x, vec_t &y, mat_t &j)
  Member function pointer.
- tclass * tptr
  Class pointer.

The documentation for this class was generated from the following file:

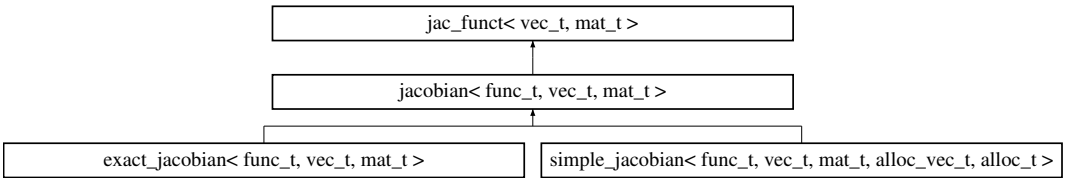
- jacobian.h

46.174 jacobian< func_t, vec_t, mat_t > Class Template Reference

Base for providing a numerical jacobian [abstract base].

`#include <jacobian.h>`

Inheritance diagram for jacobian< func_t, vec_t, mat_t >:



46.174.1 Detailed Description

template<class func_t = mm_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base>class jacobian< func_t, vec_t, mat_t >

This is provides a Jacobian which is numerically determined by differentiating a user-specified function (typically of the form of mm_funct).

Definition at line 218 of file jacobian.h.

Public Member Functions

- virtual int set_function (func_t &f)
  Set the function to compute the Jacobian of.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j)=0
  Evaluate the Jacobian `j` at point `y(x)`

Protected Attributes

- func_t * func
  A pointer to the user-specified function.

Private Member Functions

- **jacobian** (const jacobian &)
- jacobian & **operator=** (const jacobian &)

The documentation for this class was generated from the following file:

- jacobian.h

46.175 lanczos< vec_t, mat_t, alloc_vec_t, alloc_t > Class Template Reference

Lanczos diagonalization.

`#include <lanczos_base.h>`

46.175.1 Detailed Description

template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base, class alloc_vec_t = o2scl::ovector, class alloc_t = o2scl::ovector_alloc>class lanczos< vec_t, mat_t, alloc_vec_t, alloc_t >

This class approximates the largest eigenvalues of a symmetric matrix.

The vector and matrix types can be any type which provides access via `operator[]` or `operator()`, given a suitable vector allocation type. See Vector types for template classes in the User's guide for more details.

The tridiagonalization routine was rewritten from the EISPACK routines `imtql1.f` (but uses `gsl_hypot()` instead of `pythag.f`).

Idea for Future The function eigen_tdiag() automatically sorts the eigenvalues, which may not be necessary.

Idea for Future Do something better than the naive matrix-vector product. For example, use dgemm() and allow user to specify column or row-major.

Idea for Future Rework memory allocation to perform as needed.

Definition at line 54 of file lanczos_base.h.

Public Member Functions

- int eigenvalues (size_t size, mat_t &mat, size_t n_iter, vec_t &eigen, vec_t &diag, vec_t &off_diag)
  Approximate the largest eigenvalues of a symmetric matrix `mat` using the Lanczos method.
- int eigen_tdiag (size_t n, vec_t &diag, vec_t &off_diag)
  In-place diagonalization of a tridiagonal matrix.

Data Fields

- size_t td_iter
  Number of iterations for finding the eigenvalues of the tridiagonal matrix (default 30)
- size_t td_lasteval
  The index for the last eigenvalue not determined if tridiagonalization fails.

Protected Member Functions

- void product (size_t n, mat_t &a, vec_t &w, vec_t &prod)
  Naive matrix-vector product.

46.175.2 Member Function Documentation

46.175.2.1 template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base, class alloc_vec_t = o2scl::ovector, class alloc_t = o2scl::ovector_alloc> int lanczos< vec_t, mat_t, alloc_vec_t, alloc_t >::eigenvalues (size_t *size,* mat_t & *mat,* size_t *n_iter,* vec_t & *eigen,* vec_t & *diag,* vec_t & *off_diag*) [inline]

Given a square matrix `mat` with size `size`, this function applies `n_iter` iterations of the Lanczos algorithm to produce `n_iter` approximate eigenvalues stored in `eigen`. As a by-product, this function also partially tridiagonalizes the matrix placing the result in `diag` and `off_diag`. Before calling this function, space must have already been allocated for `eigen`, `diag`, and `off_diag`. All three of these arrays must have at least enough space for `n_iter` elements.

Choosing /c n_iter = `size` will produce all of the exact eigenvalues and the corresponding tridiagonal matrix, but this may be slower than diagonalizing the matrix directly.

Definition at line 90 of file lanczos_base.h.

46.175.2.2 template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base, class alloc_vec_t = o2scl::ovector, class alloc_t = o2scl::ovector_alloc> int lanczos< vec_t, mat_t, alloc_vec_t, alloc_t >::eigen_tdiag (size_t *n,* vec_t & *diag,* vec_t & *off_diag*) [inline]

On input, the vectors `diag` and `off_diag` should both be vectors of size `n`. The diagonal entries stored in `diag`, and the $n-1$ off-diagonal entries should be stored in `off_diag`, starting with `off_diag`[1]. The value in `off_diag`[0] is unused. The vector `off_diag` is destroyed by the computation.

This uses an implict QL method from the EISPACK routine `imtql1`. The value of `ierr` from the original Fortran routine is stored in td_lasteval.

Definition at line 174 of file lanczos_base.h.

46.175.2.3 template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base, class alloc_vec_t = o2scl::ovector, class alloc_t = o2scl::ovector_alloc> void lanczos< vec_t, mat_t, alloc_vec_t, alloc_t >::product (size_t *n,* mat_t & *a,* vec_t & *w,* vec_t & *prod*) [inline, protected]

It is assumed that memory is already allocated for `prod`.

Definition at line 311 of file lanczos_base.h.

The documentation for this class was generated from the following file:

- lanczos_base.h

46.176 lib_settings_class Class Reference

A class to manage global library settings.

`#include <lib_settings.h>`

46.176.1 Detailed Description

A global object of this type is defined in `lib_settings.h` named lib_settings .

Definition at line 91 of file lib_settings.h.

Public Member Functions

- std::string get_data_dir ()
  Return the data directory.
- int set_data_dir (std::string dir)
  Set the data directory.
- std::string get_tmp_dir ()
  Return the temp file directory.
- int set_tmp_dir (std::string dir)
  Set the temp file directory.
- bool eos_installed ()
  Return true if the EOS library was installed.
- bool part_installed ()
  Return true if the particle library was installed.
- bool hdf_support ()
  Return true if O_2scl was installed with HDF support.
- bool range_check ()
  Return true if range checking was turned on during installation (default true)
- std::string o2scl_version ()
  Return the library version.

Protected Attributes

- std::string data_dir
  The present data directory.
- std::string tmp_dir
  The present temp file directory.

The documentation for this class was generated from the following file:

- lib_settings.h

46.177 pinside::line Struct Reference

Internal line definition for pinside.

```
#include <pinside.h>
```

46.177.1 Detailed Description

Definition at line 77 of file pinside.h.

Data Fields

- point **p1**
- point **p2**

The documentation for this struct was generated from the following file:

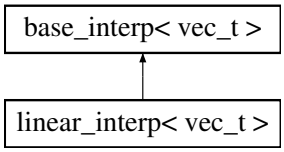
- pinside.h

46.178 linear_interp< vec_t > Class Template Reference

Linear interpolation (GSL)

`#include <interp.h>`

Inheritance diagram for linear_interp< vec_t >:



46.178.1 Detailed Description

template<class vec_t>class linear_interp< vec_t >

Linear interpolation requires no calls to allocate() or free() as there is no internal storage required.

Definition at line 159 of file interp.h.

Public Member Functions

- virtual int init (const vec_t &x, const vec_t &y, size_t size)
  Initialize interpolation routine.
- virtual int interp (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &y)
  Give the value of the function $y(x = x_0)$.
- virtual int deriv (const vec_t &x_array, const vec_t &y_array, size_t size, double x, double &dydx)
  Give the value of the derivative $y'(x = x_0)$.
- virtual int deriv2 (const vec_t &x, const vec_t &y, size_t size, double x0, double &d2ydx2)
  Give the value of the second derivative $y''(x = x_0)$ *(always zero)*
- virtual int integ (const vec_t &x_array, const vec_t &y_array, size_t size, double a, double b, double &result)
  Give the value of the integral $\int_a^b y(x)\,dx$.
- virtual const char * type ()
  Return the type, `"linear_interp"`.

Private Member Functions

- **linear_interp** (const linear_interp< vec_t > &)
- linear_interp< vec_t > & **operator=** (const linear_interp< vec_t > &)

The documentation for this class was generated from the following file:

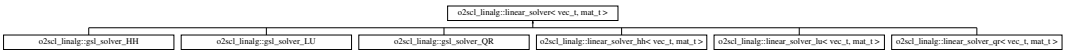
- interp.h

46.179 o2scl_linalg::linear_solver< vec_t, mat_t > Class Template Reference

A generic solver for the linear system $Ax = b$ [abstract base].

`#include <linear_solver.h>`

Inheritance diagram for o2scl_linalg::linear_solver< vec_t, mat_t >:



46.179.1 Detailed Description

template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base>class o2scl_linalg::linear_solver< vec_t, mat_t >

A generic solver for dense linear systems.

Those writing production level code should consider calling LAPACK directly using O_2scl objects as described in the Linear Algebra section of the User's Guide.

Idea for Future The test code uses a Hilbert matrix, which is known to be ill-conditioned, especially for the larger sizes. This should probably be changed.

Definition at line 52 of file linear_solver.h.

Public Member Functions

- virtual int solve (size_t n, mat_t &a, vec_t &b, vec_t &x)=0
  Solve square linear system $Ax = b$ of size n.

The documentation for this class was generated from the following file:

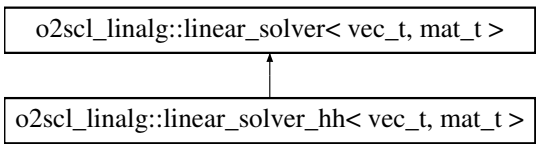
- linear_solver.h

46.180 o2scl_linalg::linear_solver_hh< vec_t, mat_t > Class Template Reference

Generic Householder linear solver.

`#include <linear_solver.h>`

Inheritance diagram for o2scl_linalg::linear_solver_hh< vec_t, mat_t >:



46.180.1 Detailed Description

template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base>class o2scl_linalg::linear_solver_hh< vec_t, mat_t >

Definition at line 102 of file linear_solver.h.

Public Member Functions

- virtual int solve (size_t n, mat_t &A, vec_t &b, vec_t &x)
  Solve square linear system $Ax = b$ of size n.

The documentation for this class was generated from the following file:

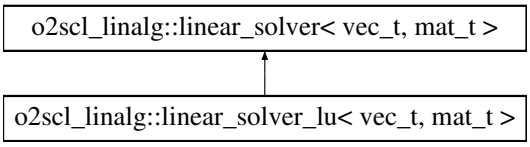
- linear_solver.h

46.181 o2scl_linalg::linear_solver_lu< vec_t, mat_t > Class Template Reference

Generic linear solver using LU decomposition.

`#include <linear_solver.h>`

Inheritance diagram for o2scl_linalg::linear_solver_lu< vec_t, mat_t >:



46.181.1 Detailed Description

template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base>class o2scl_linalg::linear_solver_lu< vec_t, mat_t >

Definition at line 64 of file linear_solver.h.

Public Member Functions

- virtual int solve (size_t n, mat_t &A, vec_t &b, vec_t &x)
  Solve square linear system $Ax = b$ of size n.

The documentation for this class was generated from the following file:

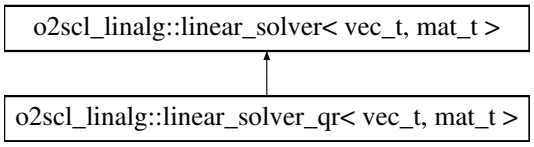
- linear_solver.h

46.182 o2scl_linalg::linear_solver_qr< vec_t, mat_t > Class Template Reference

Generic linear solver using QR decomposition.

`#include <linear_solver.h>`

Inheritance diagram for o2scl_linalg::linear_solver_qr< vec_t, mat_t >:



46.182.1 Detailed Description

template<class vec_t = o2scl::ovector_base, class mat_t = o2scl::omatrix_base>class o2scl_linalg::linear_solver_qr< vec_t, mat_t >

Definition at line 84 of file linear_solver.h.

Public Member Functions

- virtual int solve (size_t n, mat_t &A, vec_t &b, vec_t &x)

Solve square linear system $Ax = b$ of size n.

The documentation for this class was generated from the following file:

- linear_solver.h

46.183 matrix_row< mat_t > Class Template Reference

Create a vector-like class from a row of a matrix.

`#include <vector.h>`

46.183.1 Detailed Description

template<class mat_t>class matrix_row< mat_t >

Idea for Future Generalize to operator(,)

Definition at line 769 of file vector.h.

Public Member Functions

- matrix_row (mat_t &m, size_t r)
  Create an object which represents row r of matrix m.
- double & operator[ ] (size_t c)
  Access element of column c from the specified row.
- const double & operator[ ] (size_t c) const
  Access element of column c from the specified row.

Protected Attributes

- size_t r2
  The row index.
- mat_t * m2
  The pointer to the matrix.

The documentation for this class was generated from the following file:

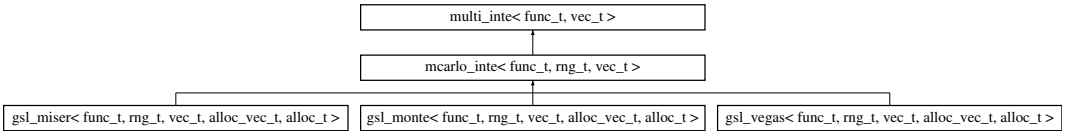
- vector.h

46.184 mcarlo_inte< func_t, rng_t, vec_t > Class Template Reference

Monte-Carlo integration [abstract base].

`#include <mcarlo_inte.h>`

Inheritance diagram for mcarlo_inte< func_t, rng_t, vec_t >:



46.184.1 Detailed Description

template<class func_t, class rng_t = gsl_rnga, class vec_t = ovector_base>class mcarlo_inte< func_t, rng_t, vec_t >

This class provides the generic Monte Carlo parameters and the random number generator. The default type for the random number generator is a gsl_rnga object.

Definition at line 42 of file mcarlo_inte.h.

Public Member Functions

- virtual const char * type ()
  Return string denoting type ("mcarlo_inte")

Data Fields

- unsigned long n_points
  Number of integration points (default 1000)
- rng_t def_rng
  The random number generator.

The documentation for this class was generated from the following file:

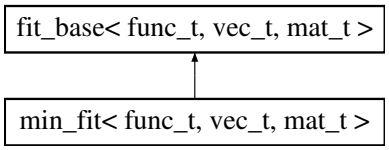
- mcarlo_inte.h

46.185 min_fit< func_t, vec_t, mat_t > Class Template Reference

Non-linear least-squares fitting class with generic minimizer.

`#include <min_fit.h>`

Inheritance diagram for min_fit< func_t, vec_t, mat_t >:



46.185.1 Detailed Description

template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base>class min_fit< func_t, vec_t, mat_t >

This minimizes a generic fitting function using any multi_min object, and then uses the GSL routines to calculate the uncertainties in the parameters and the covariance matrix.

This can be useful for fitting problems which might be better handled by more complex minimizers than those that are used in gsl_fit. For problems with many local minima near the global minimum, using a sim_anneal object with this class can sometimes produce better results than gsl_fit.

Default template arguments

- `func_t` - fit_funct<>
- `vec_t` - ovector_base

- `alloc_vec_t` - ovector

Definition at line 57 of file min_fit.h.

Data Structures

- struct func_par
  A structure for passing information to the GSL functions for the min_fit class.

Public Member Functions

- virtual int fit (size_t ndat, vec_t &xdat, vec_t &ydat, vec_t &yerr, size_t npar, vec_t &par, mat_t &covar, double &chi2, func_t &fitfun)
  Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.
- int set_multi_min (multi_min< multi_funct< vec_t > > &mm)
  Set the multi_min object to use (default is gsl_mmin_nmsimplex)
- virtual const char * type ()
  Return string denoting type ("min_fit")

Data Fields

- gsl_mmin_simp< multi_funct < vec_t > > def_multi_min
  The default minimizer.

Protected Member Functions

- double min_func (size_t np, const vec_t &xp, func_par *&fp)
  The function to minimize.

Static Protected Member Functions

- static int func (const gsl_vector *x, void *pa, gsl_vector *f)
  Evaluate the function.
- static int dfunc (const gsl_vector *x, void *pa, gsl_matrix *jac)
  Evaluate the jacobian.
- static int fdfunc (const gsl_vector *x, void *pa, gsl_vector *f, gsl_matrix *jac)
  Evaluate the function and the jacobian.

Protected Attributes

- multi_min< multi_funct< vec_t > > * mmp
  The minimizer.
- bool min_set
  True if the minimizer has been set by the user.

46.185.2 Member Function Documentation

46.185.2.1 template<class func_t = fit_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base> virtual int min_fit< func_t, vec_t, mat_t >::fit (size_t *ndat*, vec_t & *xdat*, vec_t & *ydat*, vec_t & *yerr*, size_t *npar*, vec_t & *par*, mat_t & *covar*, double & *chi2*, func_t & *fitfun*) [inline, virtual]

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Implements fit_base< func_t, vec_t, mat_t >.

Definition at line 96 of file min_fit.h.

The documentation for this class was generated from the following file:

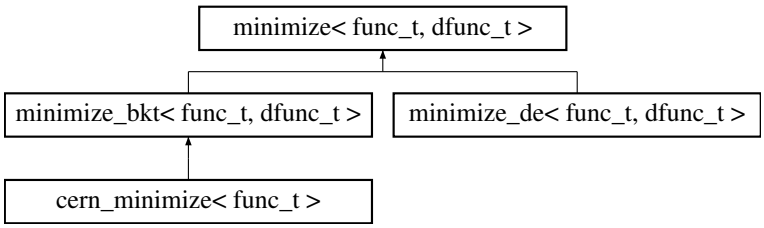
- min_fit.h

46.186 minimize< func_t, dfunc_t > Class Template Reference

One-dimensional minimization [abstract base].

`#include <minimize.h>`

Inheritance diagram for minimize< func_t, dfunc_t >:



46.186.1 Detailed Description

template<class func_t = funct, class dfunc_t = func_t>class minimize< func_t, dfunc_t >

Idea for Future Add a class for the quad_golden algorithm from GSL.

Definition at line 44 of file minimize.h.

Public Member Functions

- virtual int print_iter (double x, double y, int iter, double value=0.0, double limit=0.0, std::string comment="")
  Print out iteration information.
- virtual int min (double &x, double &fmin, func_t &func)=0
  Calculate the minimum `min` of `func` w.r.t 'x'.
- virtual int min_bkt (double &x2, double x1, double x3, double &fmin, func_t &func)=0
  Calculate the minimum `min` of `func` with x2 bracketed between x1 and x3.
- virtual int min_de (double &x, double &fmin, func_t &func, dfunc_t &df)=0
  Calculate the minimum `min` of `func` with derivative `dfunc` w.r.t 'x'.
- virtual int bracket (double &ax, double &bx, double &cx, double &fa, double &fb, double &fc, func_t &func)
  Given interval `(ax,bx)`, attempt to bracket a minimum for function `func`.
- virtual const char * type ()
  Return string denoting type ("minimize")

Data Fields

- int verbose
  Output control.
- int ntrial
  Maximum number of iterations.
- double tol_rel
  The tolerance for the minimum function value.

- double tol_abs
  The tolerance for the location of the minimum.
- int last_ntrial
  The number of iterations for in the most recent minimization.
- int bracket_iter
  The number of iterations for automatically bracketing a minimum (default 20)
- bool err_nonconv
  If true, call the error handler if the routine does not "converge".
- int last_conv
  Zero if last call to min(), min_bkt(), or min_de() converged.

46.186.2 Member Function Documentation

46.186.2.1 template<class func_t = funct, class dfunc_t = func_t> virtual int minimize< func_t, dfunc_t >::print_iter (double *x*, double *y*, int *iter*, double *value* = 0.0, double *limit* = 0.0, std::string *comment* = " ") [inline, virtual]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 102 of file minimize.h.

46.186.2.2 template<class func_t = funct, class dfunc_t = func_t> virtual int minimize< func_t, dfunc_t >::min (double & *x*, double & *fmin*, func_t & *func*) [pure virtual]

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt() with the newly bracketed minimum.

Implemented in minimize_de< func_t, dfunc_t >, minimize_bkt< func_t, dfunc_t >, and minimize_bkt< func_t >.

46.186.2.3 template<class func_t = funct, class dfunc_t = func_t> virtual int minimize< func_t, dfunc_t >::min_bkt (double & *x2*, double *x1*, double *x3*, double & *fmin*, func_t & *func*) [pure virtual]

If this is not overloaded, it ignores the bracket and calls min().

Implemented in gsl_min_quad_golden< func_t >, minimize_de< func_t, dfunc_t >, gsl_min_brent< func_t >, minimize_bkt< func_t, dfunc_t >, minimize_bkt< func_t >, and cern_minimize< func_t >.

46.186.2.4 template<class func_t = funct, class dfunc_t = func_t> virtual int minimize< func_t, dfunc_t >::min_de (double & *x*, double & *fmin*, func_t & *func*, dfunc_t & *df*) [pure virtual]

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt_de() with the newly bracketed minimum.

Implemented in minimize_de< func_t, dfunc_t >, minimize_bkt< func_t, dfunc_t >, and minimize_bkt< func_t >.

46.186.2.5 template<class func_t = funct, class dfunc_t = func_t> virtual int minimize< func_t, dfunc_t >::bracket (double & *ax*, double & *bx*, double & *cx*, double & *fa*, double & *fb*, double & *fc*, func_t & *func*) [inline, virtual]

Upon success, `fa=func(ax)`, `fb=func(bx)`, and `fc=func(cx)` with `fb<fa`, `fb<fc` and `ax<bx<cx`. The initial values of `cx`, `fa`, `fb`, and `fc` are all ignored.

The number of iterations is controlled by bracket_iter.

Note

This algorithm can fail if there's a minimum which has a much smaller size than $bx - ax$, or if the function has the same value at `ax`, `bx`, and the midpoint `(ax+bx)/2`.

Idea for Future Improve this algorithm with the golden ratio method in gsl/min/bracketing.c?

Definition at line 169 of file minimize.h.

46.186.3 Field Documentation

46.186.3.1 template<class func_t = funct, class dfunc_t = func_t> int minimize< func_t, dfunc_t >::last_conv

This is particularly useful if err_nonconv is false to test if the last call to min(), min_bkt(), or min_de() converged.

Definition at line 91 of file minimize.h.

The documentation for this class was generated from the following file:

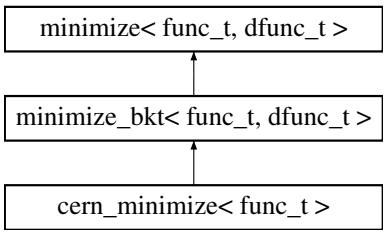
- minimize.h

46.187 minimize_bkt< func_t, dfunc_t > Class Template Reference

One-dimensional bracketing minimization [abstract base].

`#include <minimize.h>`

Inheritance diagram for minimize_bkt< func_t, dfunc_t >:



46.187.1 Detailed Description

template<class func_t, class dfunc_t = func_t>class minimize_bkt< func_t, dfunc_t >

Definition at line 241 of file minimize.h.

Public Member Functions

- virtual int min (double &x, double &fmin, func_t &func)
  Calculate the minimum `min` of `func` w.r.t 'x'.
- virtual int min_bkt (double &x2, double x1, double x3, double &fmin, func_t &func)=0
  Calculate the minimum `min` of `func` with x2 bracketed between x1 and x3.
- virtual int min_de (double &x, double &fmin, func_t &func, dfunc_t &df)
  Calculate the minimum `min` of `func` with derivative `dfunc` w.r.t 'x'.
- virtual const char * type ()
  Return string denoting type ("minimize_bkt")

Data Fields

- int bracket_iter
  The number of iterations for automatically bracketing a minimum (default 20)

46.187.2 Member Function Documentation

46.187.2.1 template<class func_t, class dfunc_t = func_t> virtual int **minimize_bkt**< func_t, dfunc_t >::min (double & *x*, double & *fmin*, func_t & *func*) [inline, virtual]

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt() with the newly bracketed minimum.

Implements minimize< func_t, dfunc_t >.

Definition at line 262 of file minimize.h.

46.187.2.2 template<class func_t, class dfunc_t = func_t> virtual int **minimize_bkt**< func_t, dfunc_t >::min_bkt (double & *x2*, double *x1*, double *x3*, double & *fmin*, func_t & *func*) [pure virtual]

If this is not overloaded, it ignores the bracket and calls min().

Implements minimize< func_t, dfunc_t >.

Implemented in gsl_min_quad_golden< func_t >, gsl_min_brent< func_t >, and cern_minimize< func_t >.

46.187.2.3 template<class func_t, class dfunc_t = func_t> virtual int **minimize_bkt**< func_t, dfunc_t >::min_de (double & *x*, double & *fmin*, func_t & *func*, dfunc_t & *df*) [inline, virtual]

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt_de() with the newly bracketed minimum.

Implements minimize< func_t, dfunc_t >.

Definition at line 288 of file minimize.h.

The documentation for this class was generated from the following file:

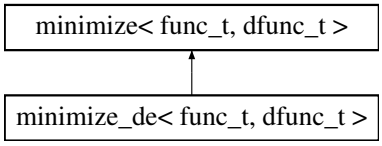
- minimize.h

46.188 minimize_de< func_t, dfunc_t > Class Template Reference

One-dimensional minimization using derivatives [abstract base].

`#include <minimize.h>`

Inheritance diagram for minimize_de< func_t, dfunc_t >:



46.188.1 Detailed Description

template<class func_t, class dfunc_t = func_t>class minimize_de< func_t, dfunc_t >

At the moment there are no minimizers of this type implemented in O_2scl .

Idea for Future Create a version of gsl_mmin_conf which implements a minimizer with this interface.

Definition at line 314 of file minimize.h.

Public Member Functions

- virtual int min (double &x, double &fmin, func_t &func)=0
  Calculate the minimum `min` *of* `func` *w.r.t 'x'.*
- virtual int min_bkt (double &x2, double x1, double x3, double &fmin, func_t &func)=0
  Calculate the minimum `min` *of* `func` *with x2 bracketed between x1 and x3.*
- virtual int min_de (double &x, double &fmin, func_t &func, dfunc_t &df)=0
  Calculate the minimum `min` *of* `func` *with derivative* `dfunc` *w.r.t 'x'.*
- virtual const char * type ()
  Return string denoting type ("minimize_de")

46.188.2 Member Function Documentation

46.188.2.1 template<class func_t , class dfunc_t = func_t> virtual int minimize_de< func_t, dfunc_t >::min (double & *x*, double & *fmin*, func_t & *func*) [pure virtual]

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt() with the newly bracketed minimum.

Implements minimize< func_t, dfunc_t >.

46.188.2.2 template<class func_t , class dfunc_t = func_t> virtual int minimize_de< func_t, dfunc_t >::min_bkt (double & *x2*, double *x1*, double *x3*, double & *fmin*, func_t & *func*) [pure virtual]

If this is not overloaded, it ignores the bracket and calls min().

Implements minimize< func_t, dfunc_t >.

46.188.2.3 template<class func_t , class dfunc_t = func_t> virtual int minimize_de< func_t, dfunc_t >::min_de (double & *x*, double & *fmin*, func_t & *func*, dfunc_t & *df*) [pure virtual]

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt_de() with the newly bracketed minimum.

Implements minimize< func_t, dfunc_t >.

The documentation for this class was generated from the following file:

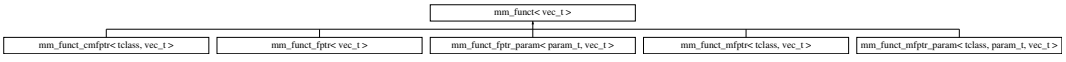
- minimize.h

46.189 mm_funct< vec_t > Class Template Reference

Array of multi-dimensional functions [abstract base].

`#include <mm_funct.h>`

Inheritance diagram for mm_funct< vec_t >:



46.189.1 Detailed Description

template<class vec_t = ovector_base>class mm_funct< vec_t >

This class generalizes `nv` functions of `nv` variables, i.e. $y_j(x_0, x_1, ..., x_{nv-1})$ for $0 \leq j \leq nv-1$.

This class is one of a large number of function object classes in O_2scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See Function Objects for a general description.

Definition at line 44 of file mm_funct.h.

Public Member Functions

- virtual int operator() (size_t nv, const vec_t &x, vec_t &y)=0
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

The documentation for this class was generated from the following file:

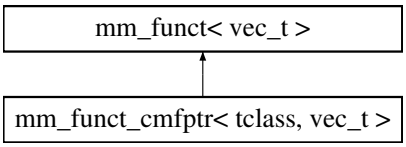
- mm_funct.h

46.190 mm_funct_cmfptr< tclass, vec_t > Class Template Reference

Const member function pointer to an array of multi-dimensional functions.

`#include <mm_funct.h>`

Inheritance diagram for mm_funct_cmfptr< tclass, vec_t >:



46.190.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class mm_funct_cmfptr< tclass, vec_t >

Definition at line 297 of file mm_funct.h.

Public Member Functions

- mm_funct_cmfptr ()
  Empty constructor.
- mm_funct_cmfptr (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y) const)
  Specify the member function pointer.
- int set_function (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y) const)
  Specify the member function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nv, const vec_t &x, vec_t &y) const
  The member function pointer.
- tclass * tptr
  The class pointer.

Private Member Functions

- **mm_funct_cmfptr** (const mm_funct_cmfptr &)
- mm_funct_cmfptr & **operator=** (const mm_funct_cmfptr &)

The documentation for this class was generated from the following file:

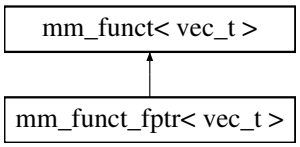
- mm_funct.h

46.191 mm_funct_fptr< vec_t > Class Template Reference

Function pointer to array of multi-dimensional functions.

`#include <mm_funct.h>`

Inheritance diagram for mm_funct_fptr< vec_t >:



46.191.1 Detailed Description

template<class vec_t = ovector_base>class mm_funct_fptr< vec_t >

Definition at line 69 of file mm_funct.h.

Public Member Functions

- mm_funct_fptr (int(*fp)(size_t nv, const vec_t &x, vec_t &y))
  Specify the function pointer.
- int set_function (int(*fp)(size_t nv, const vec_t &x, vec_t &y))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(* fptr)(size_t nv, const vec_t &x, vec_t &y)
  The function pointer to the user-supplied function.

Private Member Functions

- **mm_funct_fptr** (const mm_funct_fptr &)
- mm_funct_fptr & **operator=** (const mm_funct_fptr &)

The documentation for this class was generated from the following file:

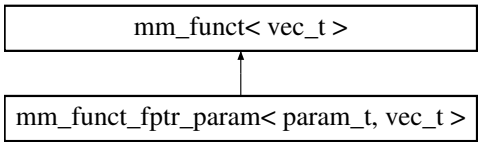
- mm_funct.h

46.192 mm_funct_fptr_param< param_t, vec_t > Class Template Reference

Function pointer to array of multi-dimensional functions.

`#include <mm_funct.h>`

Inheritance diagram for mm_funct_fptr_param< param_t, vec_t >:



46.192.1 Detailed Description

template<class param_t, class vec_t = ovector_base>class mm_funct_fptr_param< param_t, vec_t >

Definition at line 118 of file mm_funct.h.

Public Member Functions

- mm_funct_fptr_param (int(*fp)(size_t nv, const vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the function pointer.
- int set_function (int(*fp)(size_t nv, const vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(* fptr)(size_t nv, const vec_t &x, vec_t &y, param_t &)
  The function pointer to the user-supplied function.
- param_t * pp
  The parameter.

Private Member Functions

- **mm_funct_fptr_param** (const mm_funct_fptr_param &)
- mm_funct_fptr_param & **operator=** (const mm_funct_fptr_param &)

The documentation for this class was generated from the following file:

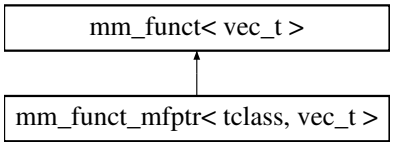
- mm_funct.h

46.193 mm_funct_mfptr< tclass, vec_t > Class Template Reference

Member function pointer to an array of multi-dimensional functions.

`#include <mm_funct.h>`

Inheritance diagram for mm_funct_mfptr< tclass, vec_t >:



46.193.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class mm_funct_mfptr< tclass, vec_t >

Definition at line 175 of file mm_funct.h.

Public Member Functions

- mm_funct_mfptr ()
  Empty constructor.
- mm_funct_mfptr (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y))
  Specify the member function pointer.
- int set_function (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y))
  Specify the member function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nv, const vec_t &x, vec_t &y)
  The member function pointer.
- tclass * tptr
  The class pointer.

Private Member Functions

- **mm_funct_mfptr** (const mm_funct_mfptr &)
- mm_funct_mfptr & **operator=** (const mm_funct_mfptr &)

The documentation for this class was generated from the following file:

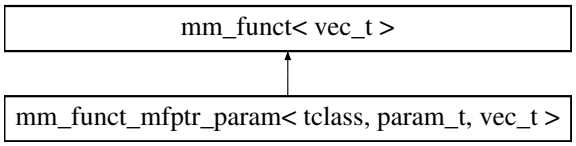
- mm_funct.h

46.194 mm_funct_mfptr_param< tclass, param_t, vec_t > Class Template Reference

Member function pointer to an array of multi-dimensional functions.

`#include <mm_funct.h>`

Inheritance diagram for mm_funct_mfptr_param< tclass, param_t, vec_t >:



46.194.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base>class mm_funct_mfptr_param< tclass, param_t, vec_t >

Definition at line 233 of file mm_funct.h.

Public Member Functions

- mm_funct_mfptr_param ()
  Empty constructor.
- mm_funct_mfptr_param (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the member function pointer.
- int set_function (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, vec_t &y, param_t &), param_t &pa)
  Specify the member function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y)
  Compute `nv` functions, `y`, of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nv, const vec_t &x, vec_t &y, param_t &pa)
  The member function pointer.
- tclass * tptr
  The class pointer.
- param_t * pp
  Parameter.

Private Member Functions

- **mm_funct_mfptr_param** (const mm_funct_mfptr_param &)
- mm_funct_mfptr_param & **operator=** (const mm_funct_mfptr_param &)

The documentation for this class was generated from the following file:

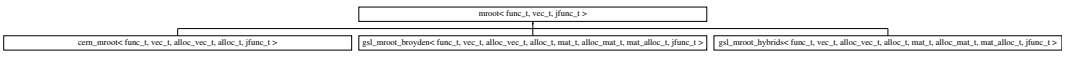
- mm_funct.h

46.195 mroot< func_t, vec_t, jfunc_t > Class Template Reference

Multidimensional root-finding [abstract base].

`#include <mroot.h>`

Inheritance diagram for mroot< func_t, vec_t, jfunc_t >:



46.195.1 Detailed Description

template<class func_t, class vec_t = ovector_base, class jfunc_t = jac_funct<vec_t,omatrix_base>>class mroot< func_t, vec_t, jfunc_t >

The template parameters: The template parameter `func_t` specifies the functions to solve and should be a class containing a definition

```
func_t::operator()(size_t nv, const vec_t &x, vec_t &y);
```

where y is the value of the functions at x with parameter pa and x and y are a array-like classes defining operator[] of size nv. If the Jacobian matrix is to be specified by the user, then the parameter jfunc_t specifies the jacobian and should contain the definition

```
jfunc_t::operator(size_t nv, vec_t &x, vec_t &y, mat_t &j);
```

where x is the independent variables, y is the array of function values, and j is the Jacobian matrix. This template parameter can be ignored if only the function msolve() will be used.

Warning

Many of the routines assume that the scale of the functions and their variables is of order unity. The solution routines may lose precision if this is not the case.

There is an example for the usage of the multidimensional solver classes given in examples/ex_mroot.cpp, see Multi-dimensional solver .

Idea for Future Change ntrial to size_t?

Definition at line 65 of file mroot.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "mroot".
- virtual int msolve (size_t n, vec_t &x, func_t &func)=0
  Solve func using x as an initial guess, returning x.
- virtual int msolve_de (size_t n, vec_t &x, func_t &func, jfunc_t &dfunc)
  Solve func with derivatives dfunc using x as an initial guess, returning x.
- template<class vec2_t , class vec3_t >
  int print_iter (size_t n, const vec2_t &x, const vec3_t &y, int iter, double value=0.0, double limit=0.0, std::string comment="")
  Print out iteration information.

Data Fields

- double tol_rel
  The maximum value of the functions for success (default 1.0e-8)
- double tol_abs
  The minimum allowable stepsize (default 1.0e-12)
- int verbose
  Output control (default 0)
- int ntrial
  Maximum number of iterations (default 100)
- int last_ntrial
  The number of iterations for in the most recent minimization.
- bool err_nonconv
  If true, call the error handler if msolve() or msolve_de() does not converge (default true)
- int last_conv
  Zero if last call to msolve() or msolve_de() converged.

46.195.2 Member Function Documentation

46.195.2.1 template<class func_t, class vec_t = ovector_base, class jfunc_t = jac_funct<vec_t,omatrix_base>> virtual int mroot< func_t, vec_t, jfunc_t >::msolve_de (size_t *n*, vec_t & *x*, func_t & *func*, jfunc_t & *dfunc*) [inline, virtual]

By default, this function just calls msolve() and ignores the last argument.

Reimplemented in gsl_mroot_hybrids< func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >.

Definition at line 119 of file mroot.h.

46.195.2.2 template<class func_t, class vec_t = ovector_base, class jfunc_t = jac_funct<vec_t,omatrix_base>> template<class vec2_t , class vec3_t > int mroot< func_t, vec_t, jfunc_t >::print_iter (size_t *n*, const vec2_t & *x*, const vec3_t & *y*, int *iter*, double *value* = 0.0, double *limit* = 0.0, std::string *comment* = "") [inline]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

This is implemented as a template class using a new vector type because sometimes the internal vector class is distinct from the user-specified vector class (e.g. in gsl_mroot_hybrids).

Definition at line 138 of file mroot.h.

46.195.3 Field Documentation

46.195.3.1 template<class func_t, class vec_t = ovector_base, class jfunc_t = jac_funct<vec_t,omatrix_base>> int mroot< func_t, vec_t, jfunc_t >::last_conv

This is particularly useful if err_nonconv is false to test if the last call to msolve() or msolve_de() converged.

Definition at line 105 of file mroot.h.

The documentation for this class was generated from the following file:

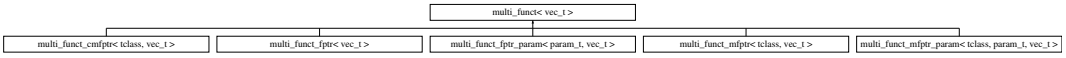
- mroot.h

46.196 multi_funct< vec_t > Class Template Reference

Multi-dimensional function [abstract base].

`#include <multi_funct.h>`

Inheritance diagram for multi_funct< vec_t >:



46.196.1 Detailed Description

template<class vec_t = ovector_base>class multi_funct< vec_t >

This class generalizes one function of several variables, i.e. $y(x_0, x_1, ..., x_{nv-1})$ where nv is the number of variables in the function y.

This class is one of a large number of function object classes in O_2scl designed to provide a mechanism for the user to supply functions to solvers, minimizers, integrators, etc. See Function Objects for a general description.

Definition at line 47 of file multi_funct.h.

Public Member Functions

- virtual double operator() (size_t nv, const vec_t &x)=0
  Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.

The documentation for this class was generated from the following file:

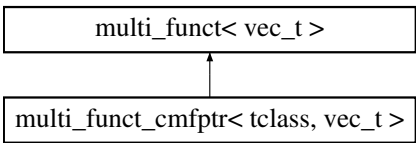
- multi_funct.h

46.197 multi_funct_cmfptr< tclass, vec_t > Class Template Reference

Const member function pointer to a multi-dimensional function.

`#include <multi_funct.h>`

Inheritance diagram for multi_funct_cmfptr< tclass, vec_t >:



46.197.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class multi_funct_cmfptr< tclass, vec_t >

Definition at line 260 of file multi_funct.h.

Public Member Functions

- multi_funct_cmfptr (tclass *tp, double(tclass::*fp)(size_t nv, const vec_t &x) const)
  Specify the member function pointer.
- virtual double operator() (size_t nv, const vec_t &x)
  Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- double(tclass::* fptr)(size_t nv, const vec_t &x) const
  Store the function pointer.
- tclass * tptr
  Store a pointer to the class instance.

Private Member Functions

- **multi_funct_cmfptr** (const multi_funct_cmfptr &)
- multi_funct_cmfptr & **operator=** (const multi_funct_cmfptr &)

The documentation for this class was generated from the following file:

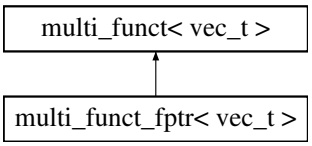
- multi_funct.h

46.198 multi_funct_fptr< vec_t > Class Template Reference

Function pointer to a multi-dimensional function.

`#include <multi_funct.h>`

Inheritance diagram for multi_funct_fptr< vec_t >:



46.198.1 Detailed Description

template<class vec_t = ovector_base>class multi_funct_fptr< vec_t >

Definition at line 74 of file multi_funct.h.

Public Member Functions

- multi_funct_fptr (double(*fp)(size_t nv, const vec_t &x))
  Specify the function pointer.
- virtual double operator() (size_t nv, const vec_t &x)
  Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- double(* fptr)(size_t nv, const vec_t &x)
  Store the function pointer.

Private Member Functions

- **multi_funct_fptr** (const multi_funct_fptr &)
- multi_funct_fptr & **operator=** (const multi_funct_fptr &)

The documentation for this class was generated from the following file:

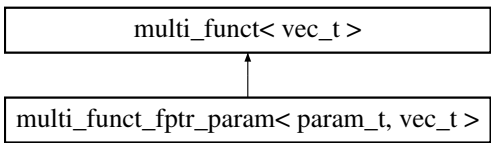
- multi_funct.h

46.199 multi_funct_fptr_param< param_t, vec_t > Class Template Reference

Function pointer to a multi-dimensional function.

`#include <multi_funct.h>`

Inheritance diagram for multi_funct_fptr_param< param_t, vec_t >:



46.199.1 Detailed Description

template<class param_t, class vec_t = ovector_base>class multi_funct_fptr_param< param_t, vec_t >

Definition at line 117 of file multi_funct.h.

Public Member Functions

- multi_funct_fptr_param (double(*fp)(size_t nv, const vec_t &x, param_t &), param_t &pa)
  Specify the function pointer.
- virtual double operator() (size_t nv, const vec_t &x)
  Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- double(* fptr)(size_t nv, const vec_t &x, param_t &)
  Store the function pointer.
- param_t * pp
  Parameter.

Private Member Functions

- **multi_funct_fptr_param** (const multi_funct_fptr_param &)
- multi_funct_fptr_param & **operator=** (const multi_funct_fptr_param &)

The documentation for this class was generated from the following file:

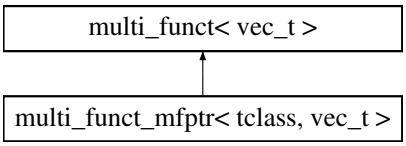
- multi_funct.h

46.200 multi_funct_mfptr< tclass, vec_t > Class Template Reference

Member function pointer to a multi-dimensional function.

`#include <multi_funct.h>`

Inheritance diagram for multi_funct_mfptr< tclass, vec_t >:



46.200.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class multi_funct_mfptr< tclass, vec_t >

Definition at line 165 of file multi_funct.h.

Public Member Functions

- multi_funct_mfptr (tclass *tp, double(tclass::*fp)(size_t nv, const vec_t &x))
  Specify the member function pointer.
- virtual double operator() (size_t nv, const vec_t &x)
  Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- double(tclass::* fptr)(size_t nv, const vec_t &x)
  Store the function pointer.
- tclass * tptr
  Store a pointer to the class instance.

Private Member Functions

- **multi_funct_mfptr** (const multi_funct_mfptr &)
- multi_funct_mfptr & **operator=** (const multi_funct_mfptr &)

The documentation for this class was generated from the following file:

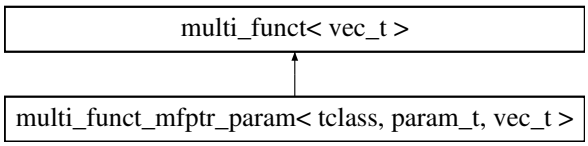
- multi_funct.h

46.201 multi_funct_mfptr_param< tclass, param_t, vec_t > Class Template Reference

Member function pointer to a multi-dimensional function.

`#include <multi_funct.h>`

Inheritance diagram for multi_funct_mfptr_param< tclass, param_t, vec_t >:



46.201.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base>class multi_funct_mfptr_param< tclass, param_t, vec_t >

Definition at line 210 of file multi_funct.h.

Public Member Functions

- multi_funct_mfptr_param (tclass *tp, double(tclass::*fp)(size_t nv, const vec_t &x, param_t &), param_t &pa)
  Specify the member function pointer.
- virtual double operator() (size_t nv, const vec_t &x)
  Compute a function `y` of `nv` variables stored in `x` with parameter `pa`.

Protected Attributes

- double(tclass::* fptr)(size_t nv, const vec_t &x, param_t &)
  Store the function pointer.
- param_t * pp
  Parameter.
- tclass * tptr
  Store a pointer to the class instance.

Private Member Functions

- **multi_funct_mfptr_param** (const multi_funct_mfptr_param &)
- multi_funct_mfptr_param & **operator=** (const multi_funct_mfptr_param &)

The documentation for this class was generated from the following file:

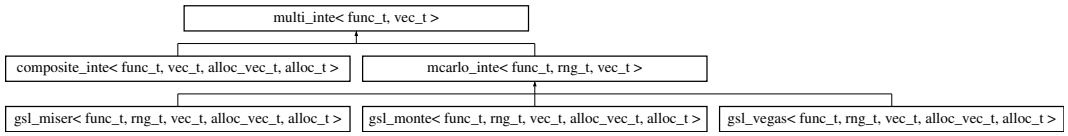
- multi_funct.h

46.202 multi_inte< func_t, vec_t > Class Template Reference

Multi-dimensional integration over a hypercube [abstract base].

`#include <multi_inte.h>`

Inheritance diagram for multi_inte< func_t, vec_t >:



46.202.1 Detailed Description

template<class func_t, class vec_t = ovector_base>class multi_inte< func_t, vec_t >

Multi-dimensional integration over a region defined by constant limits. For more general regions of integration, use children of the class gen_inte.

Definition at line 40 of file multi_inte.h.

Public Member Functions

- virtual double minteg (func_t &func, size_t ndim, const vec_t &a, const vec_t &b)
  Integrate function `func` over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i <$ ndim-1.
- virtual int minteg_err (func_t &func, size_t ndim, const vec_t &a, const vec_t &b, double &res, double &err)=0
  Integrate function `func` over the hypercube from $x_i = a_i$ to $x_i = b_i$ for $0 < i <$ ndim-1.
- double get_error ()
  Return the error in the result from the last call to minteg() or minteg_err()
- const char * type ()
  Return string denoting type ("multi_inte")

Data Fields

- bool err_nonconv
  If true, call the error handler if the routine does not "converge".
- int verbose
  Verbosity.
- double tol_rel
  The maximum "uncertainty" in the value of the integral (default 10^{-8}*).*

Protected Attributes

- double interror
  The uncertainty for the last integration computation.

46.202.2 Member Function Documentation

46.202.2.1 template<class func_t, class vec_t = ovector_base> virtual double multi_inte< func_t, vec_t >::minteg (func_t & *func*, size_t *ndim*, const vec_t & *a*, const vec_t & *b*) [inline, virtual]

Reimplemented in gsl_vegas< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >, gsl_miser< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >, and gsl_monte< func_t, rng_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 74 of file multi_inte.h.

46.202.2.2 template<class func_t, class vec_t = ovector_base> double multi_inte< func_t, vec_t >::get_error ( ) [inline]

This will quietly return zero if no integrations have been performed.

Definition at line 94 of file multi_inte.h.

The documentation for this class was generated from the following file:

- multi_inte.h

46.203 multi_min< func_t, dfunc_t, vec_t > Class Template Reference

Multidimensional minimization [abstract base].

`#include <multi_min.h>`

Inheritance diagram for multi_min< func_t, dfunc_t, vec_t >:

46.203.1 Detailed Description

template<class func_t, class dfunc_t = func_t, class vec_t = ovector_base>class multi_min< func_t, dfunc_t, vec_t >

The template parameters: The template parameter `func_t` specifies the function to minimize and should be a class containing a definition

```
func_t::operator()(size_t nv, const vec_t &x, double &f);
```

where `f` is the value of the function at `x` , where `x` is a array-like class defining `operator[]` of size `nv`. The parameter `dfunc_t` (if used) should provide the gradient with

```
func_t::operator()(size_t nv, vec_t &x, vec_t &g);
```

where `g` is the gradient of the function at `x`.

Verbose I/O is sent through `std::cout` and `std::cin` by default, but this can be modified using set_verbose_stream().

Definition at line 463 of file multi_min.h.

Public Member Functions

- int set_verbose_stream (std::ostream &out, std::istream &in)
  Set streams for verbose I/O.
- virtual int mmin (size_t nvar, vec_t &x, double &fmin, func_t &func)=0
  Calculate the minimum `min` of `func` w.r.t. the array `x` of size `nvar`.
- virtual int mmin_de (size_t nvar, vec_t &x, double &fmin, func_t &func, dfunc_t &dfunc)
  Calculate the minimum `min` of `func` w.r.t. the array `x` of size `nvar` with gradient `dfunc`.
- template<class vec2_t >
  int print_iter (size_t nv, vec2_t &x, double y, int iter, double value, double limit, std::string comment)
  Print out iteration information.
- const char * type ()
  Return string denoting type ("multi_min")

Data Fields

- int verbose
  Output control.
- int ntrial
  Maximum number of iterations.
- double tol_rel
  Function value tolerance.
- double tol_abs
  The independent variable tolerance.
- int last_ntrial
  The number of iterations for in the most recent minimization.
- bool err_nonconv
  If true, call the error handler if the routine does not "converge".
- int last_conv
  Zero if last call to mmin() or mmin_de() converged.

Protected Attributes

- std::ostream * outs
  Stream for verbose output.
- std::istream * ins
  Stream for verbose input.

46.203.2 Member Function Documentation

46.203.2.1 template<class func_t, class dfunc_t = func_t, class vec_t = ovector_base> template<class vec2_t > int multi_min< func_t, dfunc_t, vec_t >::print_iter (size_t *nv*, vec2_t & *x*, double *y*, int *iter*, double *value*, double *limit*, std::string *comment*) [inline]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 551 of file multi_min.h.

46.203.3 Field Documentation

46.203.3.1 template<class func_t, class dfunc_t = func_t, class vec_t = ovector_base> int multi_min< func_t, dfunc_t, vec_t >::last_conv

This is particularly useful if err_nonconv is false to test if the last call to mmin() or mmin_de() converged.

Definition at line 516 of file multi_min.h.

The documentation for this class was generated from the following file:

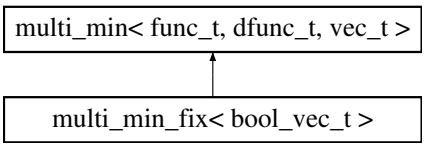
- multi_min.h

46.204 multi_min_fix< bool_vec_t > Class Template Reference

Multidimensional minimizer fixing some variables and varying others.

`#include <multi_min_fix.h>`

Inheritance diagram for multi_min_fix< bool_vec_t >:



46.204.1 Detailed Description

template<class bool_vec_t>class multi_min_fix< bool_vec_t >

See an example for the usage of this class in Minimizer fixing variables .

Todo Generalize to all vector types

Generalize to minimizers which require derivatives

At the moment, the user has to change def_mmin::ntrial instead of multi_min_fix::ntrial, which is a bit confusing. Fix this.

Definition at line 46 of file multi_min_fix.h.

Public Member Functions

- multi_min_fix ()
  Specify the member function pointer.
- virtual int mmin (size_t nvar, ovector_base &x, double &fmin, multi_funct<> &func)
  Calculate the minimum `min` of `func` w.r.t. the array `x` of size `nvar`.
- virtual int mmin_fix (size_t nvar, ovector_base &x, double &fmin, bool_vec_t &fix, multi_funct<> &func)
  Calculate the minimum of `func` while fixing some parameters as specified in `fix`.
- int set_mmin (multi_min< multi_funct_mfptr< multi_min_fix > > &min)
  Change the base minimizer.

Data Fields

- gsl_mmin_simp2 < multi_funct_mfptr < multi_min_fix > > def_mmin
  The default base minimizer.

Protected Member Functions

- virtual double min_func (size_t nv, const ovector_base &x)
  The new function to send to the minimizer.

Protected Attributes

- multi_min< multi_funct_mfptr < multi_min_fix > > * mmp
  The minimizer.
- multi_funct * funcp
  The user-specified function.
- size_t unv
  The user-specified number of variables.
- size_t nv_new
  The new number of variables.
- bool_vec_t * fixp
  Specify which parameters to fix.
- ovector_base * xp
  The user-specified initial vector.

Private Member Functions

- **multi_min_fix** (const multi_min_fix &)
- multi_min_fix & **operator=** (const multi_min_fix &)

46.204.2 Member Function Documentation

46.204.2.1 template<class bool_vec_t> virtual int multi_min_fix< bool_vec_t >::mmin_fix (size_t *nvar,* ovector_base & *x,* double & *fmin,* bool_vec_t & *fix,* multi_funct<> & *func*) [inline, virtual]

If all of entries `fix[0], fix[1], ... fix[nvar-1]` are true, then this function assumes all of the parameters are fixed and that there is no minimization to be performed. In this case, it will return 0 for success without calling the error handler.

Definition at line 107 of file multi_min_fix.h.

The documentation for this class was generated from the following file:

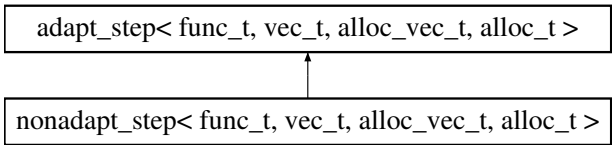
- multi_min_fix.h

46.205 nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

An non-adaptive stepper implementation of adapt_step.

`#include <nonadapt_step.h>`

Inheritance diagram for nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >:



46.205.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >

This class simply calls the specified ODE stepper without any attempt to modify the size of the step. It is primarily useful to allow for simple comparisons between adaptive and non-adaptive solution or to proceed with a solution in places where an adaptive stepper is failing.

To modify the ODE stepper which is used, use the function adapt_step::set_step().

Definition at line 48 of file nonadapt_step.h.

Public Member Functions

- virtual int astep (double &x, double xlimit, double &h, size_t n, vec_t &y, vec_t &dydx_out, vec_t &yerr, func_t &derivs)
  Make an adaptive integration step of the system `derivs`.
- virtual int astep_derivs (double &x, double xlimit, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yerr, func_t &derivs)
  Make an adaptive integration step of the system `derivs` *with derivatives.*
- virtual int astep_full (double x, double xlimit, double &x_out, double &h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)
  Make an adaptive integration step of the system `derivs`.

Protected Attributes

- size_t msize
  The allocated vector size.
- alloc_vec_t dydx_int
  Internal storage for dydx.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

46.205.2 Member Function Documentation

46.205.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::astep (double & *x*, double *xlimit*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx_out*, vec_t & *yerr*, func_t & *derivs*) [inline, virtual]

This attempts to take a step of size `h` from the point `x` of an `n-dimensional` system `derivs` starting with `y`. On exit, `x` and `y` contain the new values at the end of the step, `h` contains the size of the step, `dydx_out` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

If the base stepper returns a non-zero value, that value is passed on as the return value of `astep()`, though the input `y` may still have been modified by the base stepper.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 99 of file nonadapt_step.h.

46.205.2.2 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::astep_derivs (double & *x*, double *xlimit*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yerr*, func_t & *derivs*) [inline, virtual]

This attempts to take a step of size `h` from the point `x` of an `n-dimensional` system `derivs` starting with `y` and given the initial derivatives `dydx`. On exit, `x`, `y` and `dydx` contain the new values at the end of the step, `h` contains the size of the step, `dydx` contains the derivative at the end of the step, and `yerr` contains the estimated error at the end of the step.

If the base stepper returns a non-zero value, that value is passed on as the return value of `astep()`, though the inputs `y` and `dydx` may still have been modified by the base stepper.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 135 of file nonadapt_step.h.

46.205.2.3 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::astep_full (double *x*, double *xlimit*, double & *x_out*, double & *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [inline, virtual]

This function performs an adaptive integration step with the `n-dimensional` system `derivs` and parameter `pa`. It Begins at `x` with initial stepsize `h`, ensuring that the step goes no farther than `xlimit`. At the end of the step, the size of the step taken is `h` and the new value of `x` is in `x_out`. Initially, the function values and derivatives should be specified in `y` and `dydx`. The function values, derivatives, and the error at the end of the step are given in `yout`, `yerr`, and `dydx_out`. Unlike in `ode_step` objects, the objects `y`, `yout`, `dydx`, and `dydx_out` must all be distinct.

If the base stepper returns a non-zero value, that value is passed on as the return value of `astep()`, though the output parameters may still have been modified by the base stepper.

Implements adapt_step< func_t, vec_t, alloc_vec_t, alloc_t >.

Definition at line 170 of file nonadapt_step.h.

46.205.3 Field Documentation

46.205.3.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> alloc_vec_t nonadapt_step< func_t, vec_t, alloc_vec_t, alloc_t >::dydx_int [protected]

Definition at line 65 of file nonadapt_step.h.

The documentation for this class was generated from the following file:

- nonadapt_step.h

46.206 o2_int_shared_ptr< T > Class Template Reference

The internal shared pointer class used if no TR1 or Boost implementation is available.

`#include <shared_ptr.h>`

46.206.1 Detailed Description

template<class T>class o2_int_shared_ptr< T >

See also o2_shared_ptr .

This class was based on the Aegis `shared_ptr` class from `http://aegis.sourceforge.net`, but is implemented in O_2scl using a template struct rather than a define constant.

Definition at line 60 of file shared_ptr.h.

Public Member Functions

- o2_int_shared_ptr ()
  Create an empty shared_ptr.
- template<class Y >
  o2_int_shared_ptr (Y *rhs)

Create a shared pointer.

- o2_int_shared_ptr (const o2_int_shared_ptr &rhs)
  Copy constructor.
- template<class Y >
  o2_int_shared_ptr (const o2_int_shared_ptr< Y > &rhs)
  Copy constructor for compatible pointer types.
- bool valid () const
  Test if the shared_ptr object is valid.
- void swap (o2_int_shared_ptr &rhs)
  Swap.
- void reset ()
  Drop the reference and set the pointer to zero.
- o2_int_shared_ptr & operator= (const o2_int_shared_ptr &rhs)
  Assignment operator.
- template<class Y >
  o2_int_shared_ptr & operator= (const o2_int_shared_ptr< Y > &rhs)
  Assignment operator for compatible pointer types.
- T & operator* () const
  Dereference operator.
- T * operator-> () const
  Pointing at operator.
- T * get () const
  Return a pointer to the object.
- operator bool () const
  Return true if the pointer is not zero.
- bool operator! () const
  Logical not operator.
- bool operator== (const o2_int_shared_ptr &rhs) const
  Test equality.
- template<class U >
  bool operator== (const o2_int_shared_ptr< U > &rhs) const
  Test equality for compatible pointer types.
- bool operator!= (o2_int_shared_ptr &rhs) const
  Test inequality.
- template<class U >
  bool operator!= (o2_int_shared_ptr< U > &rhs) const
  Test inequality for compatible pointer types.
- bool operator< (o2_int_shared_ptr &rhs) const
  Less than operator.
- template<class U >
  bool operator< (o2_int_shared_ptr< U > &rhs) const
  Less than operator for compatible pointer types.

Private Attributes

- long * reference_count
  Reference count.
- T * subject
  Object being pointed to.

Friends

- class **o2_int_shared_ptr**

46.206.2 Constructor & Destructor Documentation

46.206.2.1 template<class T> template<class Y > o2_int_shared_ptr< T >::o2_int_shared_ptr (Y * *rhs*) [inline, explicit]

This constructor permits initialization from any compatable pointer where `Y` must be a complete type.

Definition at line 105 of file shared_ptr.h.

46.206.3 Member Function Documentation

46.206.3.1 template<class T> bool o2_int_shared_ptr< T >::valid ( ) const [inline]

This method verifies that the internal state of the shared pointer object is valid.

Definition at line 163 of file shared_ptr.h.

46.206.3.2 template<class T> void o2_int_shared_ptr< T >::swap (o2_int_shared_ptr< T > & *rhs*) [inline]

Swap the contents of two o2_int_shared_ptr<T> objects. This method is more efficient than manually swapping shared_ptr objects using a copy constructor or temporary.

Definition at line 174 of file shared_ptr.h.

46.206.3.3 template<class T> bool o2_int_shared_ptr< T >::operator! ( ) const [inline]

Return true if and only if the pointer is zero.

Definition at line 282 of file shared_ptr.h.

46.206.4 Field Documentation

46.206.4.1 template<class T> long* o2_int_shared_ptr< T >::reference_count [private]

Documentation from Aegis:

```
The reference_count instance variable is used to remember the
location of the reference count. By having the reference count
separate from the subject, we can cope with compatible pointers,
not just exact pointers.

This is not ideal because it allocates huge numbers of small
objects. Some heap implementations go slowly when faced with
many small allocations. Some heap implementations waste a lot of
memory when faced with many small allocations.
```

Definition at line 79 of file shared_ptr.h.

46.206.4.2 template<class T> T* o2_int_shared_ptr< T >::subject [private]

Documentation from Aegis:

```
The subject instance variable is used to remember the location
of the object being reference counted.  By having it separate
from the reference count, we can skip one indirection.
```

Definition at line 90 of file shared_ptr.h.

The documentation for this class was generated from the following file:

- shared_ptr.h

46.207 o2_shared_ptr< T > Struct Template Reference

A struct to provide the shared_ptr type.

`#include <shared_ptr.h>`

46.207.1 Detailed Description

template<class T>struct o2_shared_ptr< T >

This object exists in order to provide the shared_ptr template type used in O_2scl . The full specification of a shared pointer in O_2scl for an object of type `T` is thus

```
o2scl::o2_shared_ptr<T>::type
```

In a default O_2scl installation, type (as given below) is a typedef of

```
std::tr1::shared_ptr<T>
```

If `O2SCL_NO_TR1_MEMORY` is defined, then type is a typedef of o2_int_shared_ptr, unless `O2SCL_HAVE_BOOST` is defined, in which case type is a typedef of

```
boost::shared_ptr<T>
```

See also the discussion at `http://www.gotw.ca/gotw/079.htm` . This struct won't be necessary when C++ allows template typedef's as part of the C++11 standard `http://en.wikipedia.org/wiki/C%2B%2B11` , but very few compilers have implemented this standard yet.

Definition at line 356 of file shared_ptr.h.

Public Types

- typedef std::tr1::shared_ptr< T > type
  The actual shared_ptr type.

The documentation for this struct was generated from the following file:

- shared_ptr.h

46.208 o2scl_interp< vec_t > Class Template Reference

Interpolation class.

`#include <interp.h>`

46.208.1 Detailed Description

template<class vec_t = ovector_const_view>class o2scl_interp< vec_t >

Interpolation of ovector like objects is performed with the default template parameters, and array_interp is provided for simple interpolation on C-style arrays.

The type of interpolation to be performed can be specified using the set_type() function or in the constructor. The default is cubic splines with natural boundary conditions.

Definition at line 1125 of file interp.h.

Public Member Functions

- o2scl_interp (base_interp_mgr< vec_t > &it)
  Create with base interpolation object `it`.
- o2scl_interp ()
  Create an interpolator using the default cubic spline interpolation.
- virtual double interp (const double x0, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the function $y(x=x_0)$.
- virtual double deriv (const double x0, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the derivative $y'(x=x_0)$.
- virtual double deriv2 (const double x0, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the second derivative $y''(x=x_0)$.
- virtual double integ (const double x1, const double x2, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the integral $\int_a^b y(x)\,dx$.
- int set_type (base_interp_mgr< vec_t > &it)
  Set base interpolation object.

Data Fields

- def_interp_mgr< vec_t, cspline_interp > dim1
  Default base interpolation object (cubic spline with natural boundary conditions)

Protected Attributes

- base_interp< vec_t > * itp
  Pointer to base interpolation object.
- base_interp_mgr< vec_t > * bim1
  Pointer to base interpolation manager.

Private Member Functions

- **o2scl_interp** (const o2scl_interp< vec_t > &)
- o2scl_interp< vec_t > & **operator=** (const o2scl_interp< vec_t > &)

The documentation for this class was generated from the following file:

- interp.h

46.209 o2scl_interp_vec< vec_t > Class Template Reference

Interpolation class for pre-specified vector.

```
#include <interp.h>
```

46.209.1 Detailed Description

template<class vec_t = ovector_const_view>class o2scl_interp_vec< vec_t >

This interpolation class is intended to be sufficiently general to handle any vector type. Interpolation of ovector like objects is performed with the default template parameters, and array_interp_vec is provided for simple interpolation on C-style arrays.

This class does not double check the vector to ensure that all of the intervals for the abcissa are all increasing or all decreasing.

The type of interpolation to be performed can be specified using the set_type() function. The default is cubic splines with natural boundary conditions.

Definition at line 1293 of file interp.h.

Public Member Functions

- o2scl_interp_vec (base_interp_mgr< vec_t > &it, size_t n, const vec_t &x, const vec_t &y)
  Create with base interpolation object `it`.
- virtual double interp (const double x0)
  Give the value of the function $y(x=x_0)$.
- virtual double deriv (const double x0)
  Give the value of the derivative $y'(x=x_0)$.
- virtual double deriv2 (const double x0)
  Give the value of the second derivative $y''(x=x_0)$.
- virtual double integ (const double x1, const double x2)
  Give the value of the integral $\int_a^b y(x)\,dx$.
- int set_type (base_interp_mgr< vec_t > &it)
  Set base interpolation object.

Protected Attributes

- base_interp< vec_t > * itp
  Base interpolation object.
- base_interp_mgr< vec_t > * bim
  The interpolation manager.
- bool inc
  True if the original array was increasing.
- const vec_t * lx
  Pointer to the user-specified x vector.
- const vec_t * ly
  Pointer to the user-specified x vector.
- size_t ln
  Size of user-specified vectors.

Private Member Functions

- **o2scl_interp_vec** (const o2scl_interp_vec< vec_t > &)
- o2scl_interp_vec< vec_t > & **operator=** (const o2scl_interp_vec< vec_t > &)

The documentation for this class was generated from the following file:

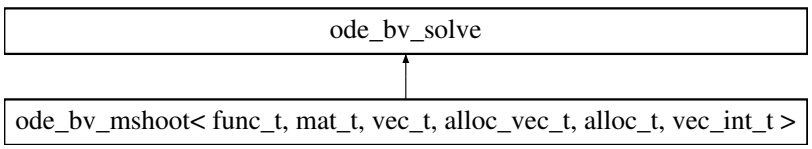
- interp.h

46.210 ode_bv_mshoot< func_t, mat_t, vec_t, alloc_vec_t, alloc_t, vec_int_t > Class Template Reference

Solve boundary-value ODE problems by multishooting with a generic nonlinear solver.

`#include <ode_bv_mshoot.h>`

Inheritance diagram for ode_bv_mshoot< func_t, mat_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >:



46.210.1 Detailed Description

template<class func_t = ode_funct<>, class mat_t = omatrix_base, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_base>class ode_bv_mshoot< func_t, mat_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >

This class is experimental.

Default template arguments

- `func_t` - ode_funct
- `mat_t` - omatrix_base
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc
- `vec_int_t` - ovector_int_base

Idea for Future Make a class which performs an iterative linear solver which uses sparse matrices like ode_it_solve?

Definition at line 52 of file ode_bv_mshoot.h.

Public Member Functions

- int solve_final_value (double h, size_t n, size_t n_bound, vec_t &x_bound, mat_t &y_bound, vec_int_t &index, func_t &derivs)
  Solve the boundary-value problem and store the solution.
- template<class mat_row_t >
  int solve_store (double h, size_t n, size_t n_bound, vec_t &x_bound, mat_t &y_bound, vec_int_t &index, size_t &n_sol, vec_t &x_sol, mat_t &y_sol, mat_t &dydx_sol, mat_t &yerr_sol, func_t &derivs)
  Solve the boundary-value problem and store the solution.
- int set_iv (ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > &ois)
  Set initial value solver.
- int set_mroot (mroot< mm_funct<> > &root)
  Set the equation solver.

Data Fields

- ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > def_ois
  The default initial value solver.
- gsl_mroot_hybrids< mm_funct<> > def_mroot
  The default equation solver.

Protected Member Functions

- int solve_fun (size_t nv, const vec_t &tx, vec_t &ty)
  The shooting function to be solved by the multidimensional solver.

Protected Attributes

- ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > * oisp
  The solver for the initial value problem.
- mroot< mm_funct<> > * mrootp
  The equation solver.
- vec_int_t * l_index
  The index defining the boundary conditions.
- vec_t * l_xbound
  Storage for the starting vector.
- mat_t * l_ybound
  Storage for the ending vector.
- double l_h
  Storage for the stepsize.
- func_t * l_derivs
  The functions to integrate.
- size_t l_n
  The number of functions.
- size_t l_nbound
  The number of boundaries.
- size_t l_lhs_unks
  The number of unknowns on the LHS.
- alloc_t ao
  Vector allocation object.
- size_t mem_size
  Size of recent allocation.

Temporary storage for \ref solve_fun()

- alloc_vec_t **sy**
- alloc_vec_t **sy2**
- alloc_vec_t **syerr**
- alloc_vec_t **sdydx**
- alloc_vec_t **sdydx2**

The documentation for this class was generated from the following file:

- ode_bv_mshoot.h

46.211 ode_bv_multishoot< func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t, mat_t > Class Template Reference

Multishooting.

```
#include <ode_bv_multishoot.h>
```

46.211.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_base, class mat_t = omatrix>class ode_bv_multishoot< func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t, mat_t >

This class is experimental.

Todo Improve documentation a little and create testing code

Definition at line 48 of file ode_bv_multishoot.h.

Public Member Functions

- virtual int **solve** (vec_t &mesh, int &n_func, vec_t &y_start, func_t &left_b, func_t &right_b, func_t &extra_b, func_t &derivs, vec_t &x_save, mat_t &y_save)
- int **set_iv** (ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > &ois)
- int **set_mroot** (mroot< mm_funct<> > &root)

Data Fields

- ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > **def_ois**
- gsl_mroot_hybrids< mm_funct<> > **def_mroot**

Protected Member Functions

- int solve_fun (size_t nv, const vec_t &sx, vec_t &sy)
  Function to solve.

Protected Attributes

- ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > * oisp
  The initial value solver.
- gsl_mroot_hybrids< mm_funct<> > * mrootp
  The equation solver.
- vec_t * l_mesh
  Desc.
- vec_t * l_y_start
  Desc.
- func_t * l_left_b
  Desc.
- func_t * l_right_b
  Desc.
- func_t * l_extra_b
  Desc.
- func_t * l_derivs
  Desc.
- int * l_n_func
  Desc.
- vec_t * l_x_save
  Desc.
- mat_t * l_y_save
  Desc.
- bool save
  Desc.

The documentation for this class was generated from the following file:

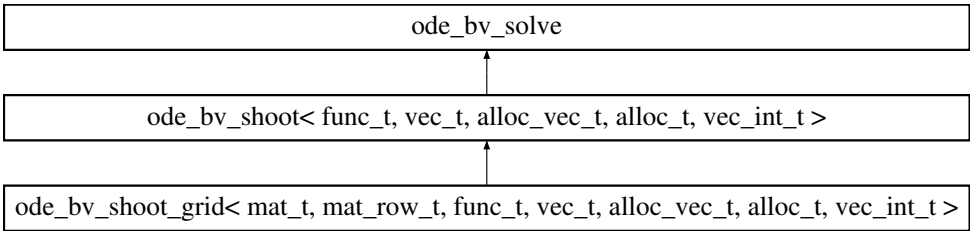
- ode_bv_multishoot.h

46.212 ode_bv_shoot< func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t > Class Template Reference

Solve boundary-value ODE problems by shooting from one boundary to the other.

`#include <ode_bv_solve.h>`

Inheritance diagram for ode_bv_shoot< func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >:



46.212.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_base>class ode_bv_shoot< func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >

This class is experimental.

Default template arguments

- `func_t` - ode_funct
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc
- `vec_int_t` - ovector_int_base

Definition at line 82 of file ode_bv_solve.h.

Public Member Functions

- void allocate (size_t n)
  Allocate internal storage.
- int solve_final_value (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_int_t &index, vec_t &yerr, vec_t &dydx_end, func_t &derivs)
  Solve the boundary-value problem and store the solution.
- template<class mat_t , class mat_row_t >
  int solve_store (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_int_t &index, size_t &n_sol, vec_t &x_sol, mat_t &y_sol, mat_t &yerr_sol, mat_t &dydx_sol, func_t &derivs)
  Solve the boundary-value problem and store the solution.
- int set_iv (ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > &ois)
  Set initial value solver.
- int set_mroot (mroot< mm_funct<> > &root)
  Set the equation solver.

Data Fields

- ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > def_ois
  The default initial value solver.
- gsl_mroot_hybrids< mm_funct<> > def_mroot
  The default equation solver.

Protected Member Functions

- int solve_fun (size_t nv, const vec_t &tx, vec_t &ty)
  The shooting function to be solved by the multidimensional solver.

Protected Attributes

- ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > * oisp
  The solver for the initial value problem.
- mroot< mm_funct<> > * mrootp
  The equation solver.
- vec_int_t * l_index
  The index defining the boundary conditions.
- vec_t * l_ystart
  Storage for the starting vector.
- vec_t * l_yend
  Storage for the ending vector.
- vec_t * l_yerr
  Storage for the starting vector.
- vec_t * l_dydx_end
  Storage for the ending vector.
- double l_x0
  Storage for the starting point.
- double l_x1
  Storage for the ending abcissa.
- double l_h
  Storage for the stepsize.
- func_t * l_derivs
  The functions to integrate.
- size_t l_n
  The number of functions.
- alloc_t ao
  Vector allocation object.
- size_t mem_size
  Size of recent allocation.

Temporary storage for \ref solve_fun()

- alloc_vec_t **sy**
- alloc_vec_t **sy2**
- alloc_vec_t **syerr**
- alloc_vec_t **sdydx**

46.212.2 Member Function Documentation

46.212.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_base> int ode_bv_shoot< func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >::solve_final_value (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, vec_t & *yend*, vec_int_t & *index*, vec_t & *yerr*, vec_t & *dydx_end*, func_t & *derivs*) [inline]

Given the `n` initial values of the functions in `ystart`, this function integrates the ODEs specified in `derivs` over the interval from `x0` to `x1` with an initial stepsize of `h`. The final values of the function are given in `yend`, the derivatives in `dydx_end`, and the associated errors are given in `yerr`. The initial values of `yend` and `yerr` are ignored.

Definition at line 126 of file ode_bv_solve.h.

The documentation for this class was generated from the following file:

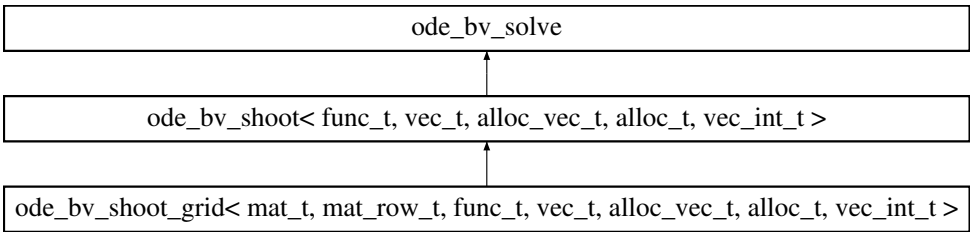
- ode_bv_solve.h

46.213 ode_bv_shoot_grid< mat_t, mat_row_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t > Class Template Reference

Solve boundary-value ODE problems by shooting from one boundary to the other on a grid.

`#include <ode_bv_solve.h>`

Inheritance diagram for ode_bv_shoot_grid< mat_t, mat_row_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >:



46.213.1 Detailed Description

template<class mat_t = omatrix, class mat_row_t = omatrix_row, class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_base>class ode_bv_shoot_grid< mat_t, mat_row_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >

This class is experimental.

Default template arguments

- `func_t` - ode_funct
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc
- `vec_int_t` - ovector_int_base

Definition at line 365 of file ode_bv_solve.h.

Public Member Functions

- int solve_grid (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_int_t &index, size_t nsol, vec_t &xsol, mat_t &ysol, mat_t &err_sol, mat_t &dydx_sol, func_t &derivs)
  Desc.

Protected Member Functions

- int solve_grid_fun (size_t nv, const vec_t &tx, vec_t &ty)
  The shooting function to be solved by the multidimensional solver.

Protected Attributes

- size_t l_nsol
  Desc.
- vec_t * l_xsol
  Desc.
- mat_t * l_ysol
  Desc.
- mat_t * l_dydxsol
  Desc.
- mat_t * l_errsol
  Desc.

The documentation for this class was generated from the following file:

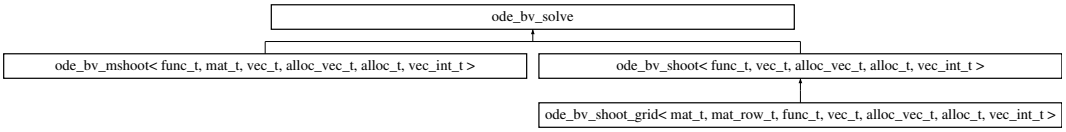
- ode_bv_solve.h

46.214 ode_bv_solve Class Reference

Base class for boundary-value ODE solvers.

`#include <ode_bv_solve.h>`

Inheritance diagram for ode_bv_solve:



46.214.1 Detailed Description

This class is experimental.

Definition at line 41 of file ode_bv_solve.h.

Data Fields

- int verbose
  Set output level.

Static Public Attributes

Values for index arrays

- static const int unk = 0
  Unknown on both the left and right boundaries.
- static const int right = 1
  Known on the right boundary.
- static const int left = 2
  Known on the left boundary.
- static const int both = 3
  Known on both the left and right boundaries.

The documentation for this class was generated from the following file:

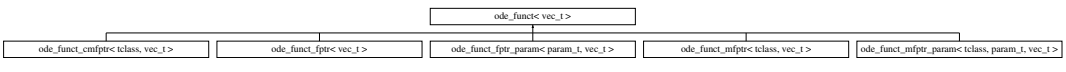
- ode_bv_solve.h

46.215 ode_funct< vec_t > Class Template Reference

Ordinary differential equation function [abstract base].

`#include <ode_funct.h>`

Inheritance diagram for ode_funct< vec_t >:



46.215.1 Detailed Description

template<class vec_t = ovector_base>class ode_funct< vec_t >

This base class provides the basic format for specifying ordinary differential equations to integrate with the O_2scl ODE solvers. Select the appropriate child of this class according to the kind of functions which are to be given to the solver.

Definition at line 40 of file ode_funct.h.

Public Member Functions

- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx)=0
  Compute the `nv` *derivatives as a function of the* `nv` *functions specified in* `y` *at the point* `x`.

Private Member Functions

- **ode_funct** (const ode_funct &)
- ode_funct & **operator=** (const ode_funct &)

The documentation for this class was generated from the following file:

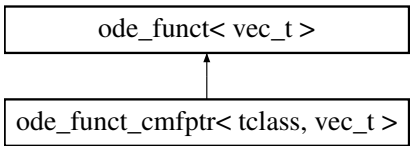
- ode_funct.h

46.216 ode_funct_cmfptr< tclass, vec_t > Class Template Reference

Provide ODE functions in the form of const member function pointers.

```
#include <ode_funct.h>
```

Inheritance diagram for ode_funct_cmfptr< tclass, vec_t >:



46.216.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class ode_funct_cmfptr< tclass, vec_t >

Definition at line 256 of file ode_funct.h.

Public Member Functions

- ode_funct_cmfptr (tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, vec_t &dydx) const)
  Create an object given a class and member function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx)
  Compute the `nv` *derivatives as a function of the* `nv` *functions specified in* `y` *at the point* `x`.

Protected Attributes

- int(tclass::* fptr)(double x, size_t nv, const vec_t &y, vec_t &dydx) const
  The pointer to the member function.
- tclass * tptr
  The pointer to the class.

Private Member Functions

- **ode_funct_cmfptr** (const ode_funct_cmfptr &)
- ode_funct_cmfptr & **operator=** (const ode_funct_cmfptr &)

The documentation for this class was generated from the following file:

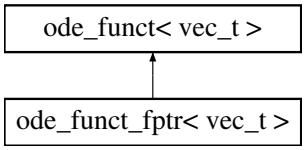
- ode_funct.h

46.217 ode_funct_fptr< vec_t > Class Template Reference

Provide ODE functions in the form of function pointers.

`#include <ode_funct.h>`

Inheritance diagram for ode_funct_fptr< vec_t >:



46.217.1 Detailed Description

template<class vec_t = ovector_base>class ode_funct_fptr< vec_t >

Definition at line 68 of file ode_funct.h.

Public Member Functions

- ode_funct_fptr (int(*fp)(double, size_t, const vec_t &, vec_t &))
  Create an object given a function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx)
  Compute the `nv` derivatives as a function of the `nv` functions specified in `y` at the point `x`.

Protected Attributes

- int(* fptr)(double x, size_t nv, const vec_t &y, vec_t &dydx)
  The function pointer.

Private Member Functions

- **ode_funct_fptr** (const ode_funct_fptr &)
- ode_funct_fptr & **operator=** (const ode_funct_fptr &)

The documentation for this class was generated from the following file:

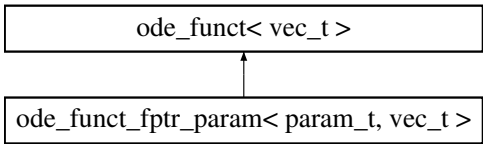
- ode_funct.h

46.218 ode_funct_fptr_param< param_t, vec_t > Class Template Reference

Provide ODE functions in the form of function pointers.

`#include <ode_funct.h>`

Inheritance diagram for ode_funct_fptr_param< param_t, vec_t >:



46.218.1 Detailed Description

template<class param_t, class vec_t = ovector_base>class ode_funct_fptr_param< param_t, vec_t >

Definition at line 110 of file ode_funct.h.

Public Member Functions

- ode_funct_fptr_param (int(*fp)(double, size_t, const vec_t &, vec_t &, param_t &), param_t &pa)
  Create an object given a function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx)
  Compute the `nv` derivatives as a function of the `nv` functions specified in `y` at the point `x`.

Protected Attributes

- int(* fptr)(double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &)
  The function pointer.
- param_t * pp
  The parameter.

Private Member Functions

- **ode_funct_fptr_param** (const ode_funct_fptr_param &)
- ode_funct_fptr_param & **operator=** (const ode_funct_fptr_param &)

The documentation for this class was generated from the following file:

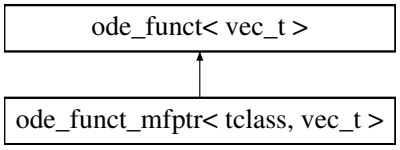
- ode_funct.h

46.219 ode_funct_mfptr< tclass, vec_t > Class Template Reference

Provide ODE functions in the form of member function pointers.

`#include <ode_funct.h>`

Inheritance diagram for ode_funct_mfptr< tclass, vec_t >:



46.219.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class ode_funct_mfptr< tclass, vec_t >

Definition at line 157 of file ode_funct.h.

Public Member Functions

- ode_funct_mfptr (tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, vec_t &dydx))
  Create an object given a class and member function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx)
  Compute the `nv` derivatives as a function of the `nv` functions specified in `y` at the point `x`.

Protected Attributes

- int(tclass::* fptr)(double x, size_t nv, const vec_t &y, vec_t &dydx)
  The pointer to the member function.
- tclass * tptr
  The pointer to the class.

Private Member Functions

- **ode_funct_mfptr** (const ode_funct_mfptr &)
- ode_funct_mfptr & **operator=** (const ode_funct_mfptr &)

The documentation for this class was generated from the following file:

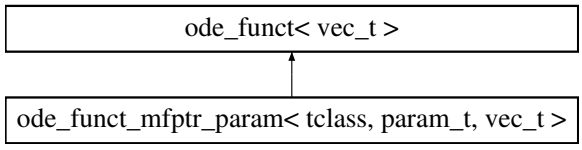
- ode_funct.h

46.220 ode_funct_mfptr_param< tclass, param_t, vec_t > Class Template Reference

Provide ODE functions in the form of member function pointers.

`#include <ode_funct.h>`

Inheritance diagram for ode_funct_mfptr_param< tclass, param_t, vec_t >:



46.220.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_base>class ode_funct_mfptr_param< tclass, param_t, vec_t >

Definition at line 204 of file ode_funct.h.

Public Member Functions

- ode_funct_mfptr_param (tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &), param_t &pa)
  Create an object given a class and member function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx)
  Compute the `nv` derivatives as a function of the `nv` functions specified in `y` at the point `x`.

Protected Attributes

- int(tclass::* fptr)(double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &)
  The pointer to the member function.
- tclass * tptr
  The pointer to the class.
- param_t * pp
  The parameter.

Private Member Functions

- **ode_funct_mfptr_param** (const ode_funct_mfptr_param &)
- ode_funct_mfptr_param & **operator=** (const ode_funct_mfptr_param &)

The documentation for this class was generated from the following file:

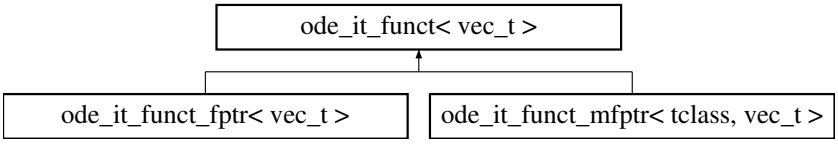
- ode_funct.h

46.221 ode_it_funct< vec_t > Class Template Reference

Function class for ode_it_solve.

`#include <ode_it_solve.h>`

Inheritance diagram for ode_it_funct< vec_t >:



46.221.1 Detailed Description

template<class vec_t = ovector_base>class ode_it_funct< vec_t >

Definition at line 42 of file ode_it_solve.h.

Public Member Functions

- virtual double operator() (size_t ieq, double x, vec_t &y)
  Using `x` and `y`, return the value of function number `ieq`.

The documentation for this class was generated from the following file:

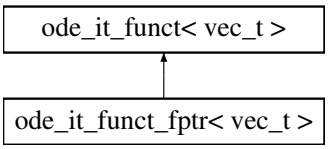
- ode_it_solve.h

46.222 ode_it_funct_fptr< vec_t > Class Template Reference

Function pointer for ode_it_solve.

`#include <ode_it_solve.h>`

Inheritance diagram for ode_it_funct_fptr< vec_t >:



46.222.1 Detailed Description

template<class vec_t = ovector_base>class ode_it_funct_fptr< vec_t >

Definition at line 60 of file ode_it_solve.h.

Public Member Functions

- ode_it_funct_fptr (double(*fp)(size_t, double, vec_t &))
  Create using a function pointer.
- virtual double operator() (size_t ieq, double x, vec_t &y)
  Using `x` and `y`, return the value of function number `ieq`.

Protected Attributes

- double(* fptr)(size_t ieq, double x, vec_t &y)
  The function pointer.

The documentation for this class was generated from the following file:

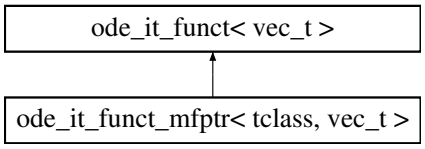
- ode_it_solve.h

46.223 ode_it_funct_mfptr< tclass, vec_t > Class Template Reference

Member function pointer for ode_it_solve.

`#include <ode_it_solve.h>`

Inheritance diagram for ode_it_funct_mfptr< tclass, vec_t >:



46.223.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class ode_it_funct_mfptr< tclass, vec_t >

Definition at line 87 of file ode_it_solve.h.

Public Member Functions

- ode_it_funct_mfptr (tclass *tp, double(tclass::*fp)(size_t, double, vec_t &))
  Create using a class instance and member function.
- virtual double operator() (size_t ieq, double x, vec_t &y)
  Using `x` and `y`, return the value of function number `ieq`.

Protected Attributes

- tclass * tptr
  The class pointer.
- double(tclass::* fptr)(size_t ieq, double x, vec_t &y)
  The member function pointer.

The documentation for this class was generated from the following file:

- ode_it_solve.h

46.224 ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t > Class Template Reference

ODE solver using a generic linear solver to solve finite-difference equations.

`#include <ode_it_solve.h>`

46.224.1 Detailed Description

template<class func_t = ode_it_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class matrix_row_t = omatrix_row, class solver_vec_t = ovector_base, class solver_mat_t = omatrix_base>class ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >

Note

This class does not work with all matrix and vector types because using operator[][] would require it to be inefficient for use with sparse matrices.

Idea for Future Create a GSL-like set() and iterate() interface

Idea for Future Implement as a child of ode_bv_solve ?

Idea for Future Max and average tolerance?

Definition at line 130 of file ode_it_solve.h.

Public Member Functions

- int set_solver (o2scl_linalg::linear_solver< solver_vec_t, solver_mat_t > &ls)
  Set the linear solver.
- int solve (size_t n_grid, size_t n_eq, size_t nb_left, vec_t &x, mat_t &y, func_t &derivs, func_t &left, func_t &right, solver_mat_t &mat, solver_vec_t &rhs, solver_vec_t &dy)
  Solve `derivs` *with boundary conditions* `left` *and* `right`*.*

Data Fields

- bool **make_mats**
- int verbose
  Set level of output (default 0)
- double h
  Stepsize for finite differencing (default 10^{-4}*)*
- double tol_rel
  Tolerance (default 10^{-8}*)*
- size_t niter
  Maximum number of iterations (default 30)
- double alpha
  Size of correction to apply (default 1.0)
- o2scl_linalg::linear_solver_hh < solver_vec_t, solver_mat_t > def_solver
  Default linear solver.

Protected Member Functions

- virtual double fd_left (size_t ieq, size_t ivar, double x, vec_t &y)
  Compute the derivatives of the LHS boundary conditions.
- virtual double fd_right (size_t ieq, size_t ivar, double x, vec_t &y)
  Compute the derivatives of the RHS boundary conditions.
- virtual double fd_derivs (size_t ieq, size_t ivar, double x, vec_t &y)
  Compute the finite-differenced part of the differential equations.

Protected Attributes

- o2scl_linalg::linear_solver < solver_vec_t, solver_mat_t > * solver
  Solver.

Storage for functions

- ode_it_funct< vec_t > * **fl**
- ode_it_funct< vec_t > * **fr**
- ode_it_funct< vec_t > * **fd**

46.224.2 Member Function Documentation

46.224.2.1 template<class func_t = ode_it_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class matrix_row_t = omatrix_row, class solver_vec_t = ovector_base, class solver_mat_t = omatrix_base> int ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >::solve (size_t *n_grid*, size_t *n_eq*, size_t *nb_left*, vec_t & *x*, mat_t & *y*, func_t & *derivs*, func_t & *left*, func_t & *right*, solver_mat_t & *mat*, solver_vec_t & *rhs*, solver_vec_t & *dy*) [inline]

Given a grid of size `n_grid` and `n_eq` differential equations, solve them by relaxation. The grid is specified in `x`, which is a vector of size `n_grid`. The differential equations are given in `derivs`, the boundary conditions on the left hand side in `left`, and the boundary conditions on the right hand side in `right`. The number of boundary conditions on the left hand side is `nb_left`, and the number of boundary conditions on the right hand side should be `n_eq-nb_left`. The initial guess for the solution, a matrix of size `[n_grid][n_eq]` should be given in `y`. Upon success, `y` will contain an approximate solution of the differential

equations. The matrix `mat` is workspace of size `[n_grid*n_eq][n_grid*n_eq]`, and the vectors `rhs` and `y` are workspace of size `[n_grid*n_eq]`.

Definition at line 189 of file ode_it_solve.h.

46.224.2.2 template<class func_t = ode_it_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class matrix_row_t = omatrix_row, class solver_vec_t = ovector_base, class solver_mat_t = omatrix_base> virtual double ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >::fd_left (size_t *ieq*, size_t *ivar*, double *x*, vec_t & *y*) [inline, protected, virtual]

This function computes $\partial f_{left,\mathrm{ieq}}/\partial y_{\mathrm{ivar}}$

Definition at line 356 of file ode_it_solve.h.

46.224.2.3 template<class func_t = ode_it_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class matrix_row_t = omatrix_row, class solver_vec_t = ovector_base, class solver_mat_t = omatrix_base> virtual double ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >::fd_right (size_t *ieq*, size_t *ivar*, double *x*, vec_t & *y*) [inline, protected, virtual]

This function computes $\partial f_{right,\mathrm{ieq}}/\partial y_{\mathrm{ivar}}$

Definition at line 375 of file ode_it_solve.h.

46.224.2.4 template<class func_t = ode_it_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class matrix_row_t = omatrix_row, class solver_vec_t = ovector_base, class solver_mat_t = omatrix_base> virtual double ode_it_solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >::fd_derivs (size_t *ieq*, size_t *ivar*, double *x*, vec_t & *y*) [inline, protected, virtual]

This function computes $\partial f_{\mathrm{ieq}}/\partial y_{\mathrm{ivar}}$

Definition at line 395 of file ode_it_solve.h.

The documentation for this class was generated from the following file:

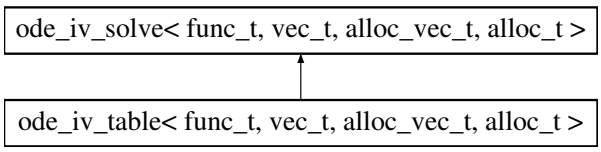
- ode_it_solve.h

46.225 ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Solve an initial-value ODE problems given an adaptive ODE stepper.

`#include <ode_iv_solve.h>`

Inheritance diagram for ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >:



46.225.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >

This class is experimental.

This class gives several functions which solve an initial value ODE problem. The functions solve_final_value() give only the final value of the functions at the end of the ODE integration and are relatively fast.

The function solve_store() stores the solution of the ODE over the full range into a set of vectors and matrices which are allocated

and specified by the user. This function is designed to give exactly the same results (though this cannot be guaranteed) as solve_final_value() and additionally records some or all of the results from the adaptive steps which were taken.

The functions solve_grid() works as in solve_store() except that the solution is stored on a grid of points in the independent variable specified by the user, at the cost of taking extra steps to ensure that function values, derivatives, and errors are computed at each grid point.

All of these functions automatically evaluate the derivatives from the specified function at the initial point and user-specified initial derivatives are ignored. The total number of steps taken is limited by ntrial and nsteps stores the number of steps taken by the most recent solution. The variable nsteps_out is the maximum number of points in the interval for which verbose output will be given when verbose is greater than zero.

There is an example for the usage of this class in `examples/ex_ode.cpp` documented in the Ordinary differential equations section.

Default template arguments

- `func_t` - ode_funct
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc

The default adaptive stepper is an object of type gsl_astep.

Idea for Future The form of solve_final_value() is very similar to that of adapt_step::astep_full(), but not quite the same. Maybe should probably be made to be consistent with each other?

Definition at line 84 of file ode_iv_solve.h.

Public Member Functions

- virtual const char * type ()
  Return the type, `"ode_iv_solve"`.

Main solver functions

- int solve_final_value (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, func_t &derivs)
  Solve the initial-value problem to get the final value.
- int solve_final_value (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_t &yerr, func_t &derivs)
  Solve the initial-value problem to get the final value with errors.
- int solve_final_value (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_t &yerr, vec_t &dydx_end, func_t &derivs)
  Solve the initial-value problem to get the final value, derivatives, and errors.
- template<class mat_t , class mat_row_t >
  int solve_store (double x0, double x1, double h, size_t n, size_t &n_sol, vec_t &x_sol, mat_t &y_sol, mat_t &yerr_sol, mat_t &dydx_sol, func_t &derivs, size_t istart=0)
  Solve the initial-value problem and store the associated output.
- template<class mat_t , class mat_row_t >
  int solve_grid (double h, size_t n, size_t nsol, vec_t &xsol, mat_t &ysol, mat_t &err_sol, mat_t &dydx_sol, func_t &derivs)
  Solve the initial-value problem from `x0` to `x1` over a grid storing derivatives and errors.

Data Fields

- int verbose
  Set output level.

- size_t nsteps_out
  Number of output points if verbose is greater than zero (default 10)
- size_t ntrial
  Maximum number of applications of the adaptive stepper (default 1000)
- size_t nsteps
  Number of adaptive steps employed.

Protected Member Functions

- virtual int print_iter (double x, size_t nv, vec_t &y)
  Print out iteration information.
- void free ()
  Free allocated memory.
- void allocate (size_t n)
  Allocate space for temporary vectors.

Protected Attributes

- alloc_t ao
  Memory allocator.
- size_t mem_size
  Desc.
- adapt_step< func_t, vec_t, alloc_vec_t, alloc_t > * astp
  The adaptive stepper.

Desc

- alloc_vec_t **vtemp**
- alloc_vec_t **vtemp2**
- alloc_vec_t **vtemp3**
- alloc_vec_t **vtemp4**

The adaptive stepper

- bool exit_on_fail
  If true, stop the solution if the adaptive stepper fails (default true)
- gsl_astep< func_t, vec_t, alloc_vec_t, alloc_t > gsl_astp
  The default adaptive stepper.
- int set_adapt_step (adapt_step< func_t, vec_t, alloc_vec_t, alloc_t > &as)
  Set the adaptive stepper to use.

46.225.2 Member Function Documentation

46.225.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_final_value (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, vec_t & *yend*, func_t & *derivs*) [inline]

Given the `n` initial values of the functions in `ystart`, this function integrates the ODEs specified in `derivs` over the interval from `x0` to `x1` with an initial stepsize of `h`. The final values of the function are given in `yend` and the initial values of `yend` are ignored.

If verbose is greater than zero, The solution at less than or approximately equal to nsteps_out points will be written to `std::cout`. If verbose is greater than one, a character will be required after each selected point.

Todo Document if yend can be the same as ystart.

Definition at line 172 of file ode_iv_solve.h.

46.225.2.2 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_final_value (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, vec_t & *yend*, vec_t & *yerr*, func_t & *derivs*) [inline]

Given the n initial values of the functions in ystart, this function integrates the ODEs specified in derivs over the interval from x0 to x1 with an initial stepsize of h. The final values of the function are given in yend and the associated errors are given in yerr. The initial values of yend and yerr are ignored.

If verbose is greater than zero, The solution at less than or approximately equal to nsteps_out points will be written to std::cout. If verbose is greater than one, a character will be required after each selected point.

Definition at line 195 of file ode_iv_solve.h.

46.225.2.3 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_final_value (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, vec_t & *yend*, vec_t & *yerr*, vec_t & *dydx_end*, func_t & *derivs*) [inline]

Given the n initial values of the functions in ystart, this function integrates the ODEs specified in derivs over the interval from x0 to x1 with an initial stepsize of h. The final values of the function are given in yend, the derivatives in dydx_end, and the associated errors are given in yerr. The initial values of yend and yerr are ignored.

This function is designed to be relatively fast, avoiding extra copying of vectors back and forth.

If verbose is greater than zero, The solution at less than or approximately equal to nsteps_out points will be written to std::cout. If verbose is greater than one, a character will be required after each selected point.

This function computes dydx_start automatically and the values given by the user are ignored.

The solution fails if more than ntrial steps are required. This function will also fail if x1>x0 and h<0 or if x1<x0 but h>0.

Definition at line 230 of file ode_iv_solve.h.

46.225.2.4 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> template<class mat_t , class mat_row_t > int ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_store (double *x0*, double *x1*, double *h*, size_t *n*, size_t & *n_sol*, vec_t & *x_sol*, mat_t & *y_sol*, mat_t & *yerr_sol*, mat_t & *dydx_sol*, func_t & *derivs*, size_t *istart* = 0) [inline]

Initially, x_sol should be a vector of size n_sol, and y_sol, dydx_sol, and yerr_sol should be matrices with size [n_sol][n]. On exit, n_sol will will be number of points store, less than or equal to the original value of n_sol. This function avoids performing extra calls to the adaptive stepper, and the table will be approximately evenly spaced.

This function is also designed to give the exactly the same results as solve_final_value(). This cannot be strictly guaranteed, but will likely hold in most applications.

This template function works with any matrix class mat_t which can be accessed using successive applications of operator[] and which has an associated class mat_row_t which returns a row of a matrix of type mat_t as an object with type vec_t.

If verbose is greater than zero, The solution at each internal point will be written to std::cout. If verbose is greater than one, a character will be required after each point.

Idea for Future It might be possible to remove some extra copying by removing the yerrl and dydx vectors

Definition at line 425 of file ode_iv_solve.h.

46.225.2.5 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> template<class mat_t , class mat_row_t > int ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::solve_grid (double *h*, size_t *n*, size_t *nsol*, vec_t & *xsol*, mat_t & *ysol*, mat_t & *err_sol*, mat_t & *dydx_sol*, func_t & *derivs*) [inline]

Initially, xsol should be an array of size nsol, and ysol should be a omatrix of size [nsol][n]. This function never takes a step larger than the grid size.

If verbose is greater than zero, The solution at each grid point will be written to std::cout. If verbose is greater than one, a

character will be required after each point.

Idea for Future Consider making a version of grid which gives the same answers as solve_final_value(). After each proposed step, it would go back and fill in the grid points if the proposed step was past the next grid point.

Definition at line 598 of file ode_iv_solve.h.

46.225.3 Field Documentation

46.225.3.1 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> size_t ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::nsteps_out

This is used in functions solve_table() and solve_final_value() to control how often steps are output when verbose is greater than zero.

Definition at line 702 of file ode_iv_solve.h.

46.225.3.2 template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> bool ode_iv_solve< func_t, vec_t, alloc_vec_t, alloc_t >::exit_on_fail

If this is false, then failures in the adaptive stepper are ignored.

Definition at line 724 of file ode_iv_solve.h.

The documentation for this class was generated from the following file:

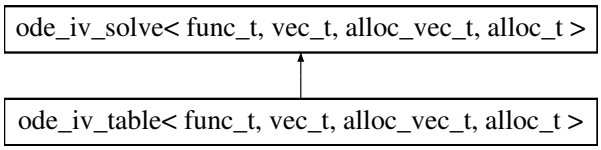
- ode_iv_solve.h

46.226 ode_iv_table< func_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Solve an initial-value ODE problem and store the result in a table object.

`#include <ode_iv_table.h>`

Inheritance diagram for ode_iv_table< func_t, vec_t, alloc_vec_t, alloc_t >:



46.226.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class ode_iv_table< func_t, vec_t, alloc_vec_t, alloc_t >

This class is experimental.

Definition at line 43 of file ode_iv_table.h.

Public Member Functions

- int solve_grid_table (size_t n, vec_t &ystart, table &t, std::string x_col, std::string y_prefix, std::string dydx_prefix, std::string yerr_prefix, func_t &derivs)

Desc.

- int solve_store_table (double x0, double x1, double h, size_t n, vec_t &ystart, size_t &n_sol, table &t, std::string x_col, std::string y_prefix, std::string dydx_prefix, std::string yerr_prefix, func_t &derivs)
  Desc.

The documentation for this class was generated from the following file:

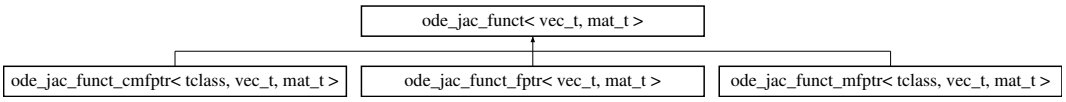
- ode_iv_table.h

46.227 ode_jac_funct< vec_t, mat_t > Class Template Reference

Ordinary differential equation Jacobian [abstract base].

`#include <ode_jac_funct.h>`

Inheritance diagram for ode_jac_funct< vec_t, mat_t >:



46.227.1 Detailed Description

template<class vec_t = ovector_base, class mat_t = omatrix_base>class ode_jac_funct< vec_t, mat_t >

This base class provides the basic format for specifying the Jacobian for ordinary differential equations to integrate with the O_2scl ODE solvers. Select the appropriate child of this class according to the kind of functions which are to be given to the solver.

Definition at line 43 of file ode_jac_funct.h.

Public Member Functions

- virtual int operator() (double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx)=0
  Compute the derivatives `dfdx` and the Jacobian matrix `dfdy` given `y` at the point `x`.

Private Member Functions

- **ode_jac_funct** (const ode_jac_funct &)
- ode_jac_funct & **operator=** (const ode_jac_funct &)

The documentation for this class was generated from the following file:

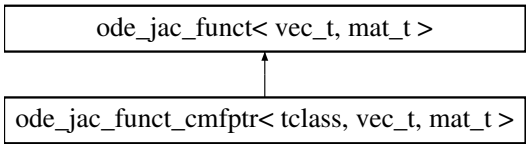
- ode_jac_funct.h

46.228 ode_jac_funct_cmfptr< tclass, vec_t, mat_t > Class Template Reference

Provide ODE Jacobian in the form of const member function pointers.

`#include <ode_jac_funct.h>`

Inheritance diagram for ode_jac_funct_cmfptr< tclass, vec_t, mat_t >:



46.228.1 Detailed Description

template<class tclass, class vec_t = ovector_base, class mat_t = omatrix_base>class ode_jac_funct_cmfptr< tclass, vec_t, mat_t >

Definition at line 177 of file ode_jac_funct.h.

Public Member Functions

- ode_jac_funct_cmfptr (tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx) const)
  Create an object given a class and member function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx)
  Compute the derivatives `dfdx` and the Jacobian matrix `dfdy` given `y` at the point `x`.

Protected Attributes

- int(tclass::* fptr)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx) const
  The pointer to the member function.
- tclass * tptr
  The pointer to the class.

Private Member Functions

- **ode_jac_funct_cmfptr** (const ode_jac_funct_cmfptr &)
- ode_jac_funct_cmfptr & **operator=** (const ode_jac_funct_cmfptr &)

The documentation for this class was generated from the following file:

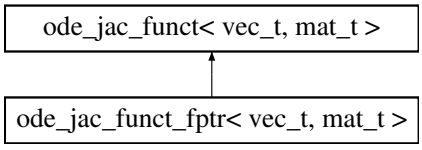
- ode_jac_funct.h

46.229 ode_jac_funct_fptr< vec_t, mat_t > Class Template Reference

Provide ODE Jacobian in the form of function pointers.

```
#include <ode_jac_funct.h>
```

Inheritance diagram for ode_jac_funct_fptr< vec_t, mat_t >:



46.229.1 Detailed Description

template<class vec_t = ovector_base, class mat_t = omatrix_base>class ode_jac_funct_fptr< vec_t, mat_t >

Definition at line 72 of file ode_jac_funct.h.

Public Member Functions

- ode_jac_funct_fptr (int(*fp)(double, size_t, const vec_t &, mat_t &, vec_t &))
  Create an object given a function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx)
  Compute the derivatives `dfdx` and the Jacobian matrix `dfdy` given `y` at the point `x`.

Protected Attributes

- int(* fptr)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx)
  The function pointer.

Private Member Functions

- **ode_jac_funct_fptr** (const ode_jac_funct_fptr &)
- ode_jac_funct_fptr & **operator=** (const ode_jac_funct_fptr &)

The documentation for this class was generated from the following file:

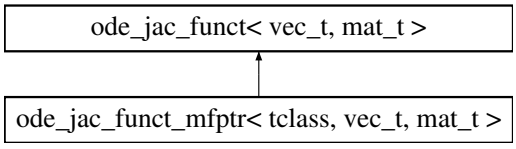
- ode_jac_funct.h

46.230 ode_jac_funct_mfptr< tclass, vec_t, mat_t > Class Template Reference

Provide ODE Jacobian in the form of member function pointers.

`#include <ode_jac_funct.h>`

Inheritance diagram for ode_jac_funct_mfptr< tclass, vec_t, mat_t >:



46.230.1 Detailed Description

template<class tclass, class vec_t = ovector_base, class mat_t = omatrix_base>class ode_jac_funct_mfptr< tclass, vec_t, mat_t >

Definition at line 122 of file ode_jac_funct.h.

Public Member Functions

- ode_jac_funct_mfptr (tclass *tp, int(tclass::*fp)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx))
  Create an object given a class and member function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx)
  Compute the derivatives `dfdx` and the Jacobian matrix `dfdy` given `y` at the point `x`.

Protected Attributes

- int(tclass::* fptr)(double x, size_t nv, const vec_t &y, mat_t &dfdy, vec_t &dfdx)
  The pointer to the member function.
- tclass * tptr
  The pointer to the class.

Private Member Functions

- **ode_jac_funct_mfptr** (const ode_jac_funct_mfptr &)
- ode_jac_funct_mfptr & **operator=** (const ode_jac_funct_mfptr &)

The documentation for this class was generated from the following file:

- ode_jac_funct.h

46.231 ode_step< func_t, vec_t > Class Template Reference

ODE stepper base [abstract base].

`#include <ode_step.h>`

Inheritance diagram for ode_step< func_t, vec_t >:

46.231.1 Detailed Description

template<class func_t = ode_funct<>, class vec_t = ovector_base>class ode_step< func_t, vec_t >

Definition at line 37 of file ode_step.h.

Public Member Functions

- virtual int get_order ()
  Return the order of the ODE stepper.
- virtual int step (double x, double h, size_t n, vec_t &y, vec_t &dydx, vec_t &yout, vec_t &yerr, vec_t &dydx_out, func_t &derivs)=0
  Perform an integration step.

Protected Attributes

- int order
  The order of the ODE stepper.

46.231.2 Member Function Documentation

46.231.2.1 template<class func_t = ode_funct<>, class vec_t = ovector_base> virtual int ode_step< func_t, vec_t >::get_order ( ) [inline, virtual]

This is used, for example, by gsl_astep to adaptively adjust the stepsize.

Definition at line 62 of file ode_step.h.

46.231.2.2 template<class func_t = ode_funct<>, class vec_t = ovector_base> virtual int ode_step< func_t, vec_t >::step (double *x*, double *h*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yout*, vec_t & *yerr*, vec_t & *dydx_out*, func_t & *derivs*) [pure virtual]

Given initial value of the n-dimensional function in `y` and the derivative in `dydx` (which must generally be computed beforehand) at the point `x`, take a step of size `h` giving the result in `yout`, the uncertainty at $x+h$ in `yerr`, and the new derivative at $x+h$ in `dydx_out` using function `derivs` to calculate derivatives. Implementations which do not calculate `yerr` and/or `dydx_out` do not reference these variables so that a blank `vec_t` can be given. All of the current O_2scl implementations allow `yout=y` and `dydx_out=dydx` if necessary

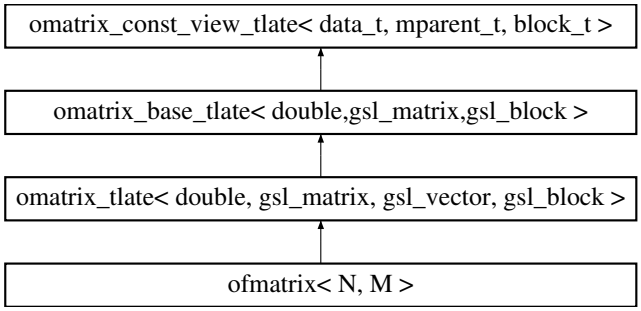
Implemented in gsl_rk8pd_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >, gsl_rkf45_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >, gsl_rkck_fast< N, func_t, vec_t, alloc_vec_t, alloc_t >, gsl_rk8pd< func_t, vec_t, alloc_vec_t, alloc_t >, gsl_rkck< func_t, vec_t, alloc_vec_t, alloc_t >, and gsl_rkf45< func_t, vec_t, alloc_vec_t, alloc_t >.

The documentation for this class was generated from the following file:

- ode_step.h

46.232 ofmatrix< N, M > Class Template Reference

Inheritance diagram for ofmatrix< N, M >:



46.232.1 Detailed Description

template<size_t N, size_t M>class ofmatrix< N, M >

Definition at line 1334 of file omatrix_tlate.h.

The documentation for this class was generated from the following file:

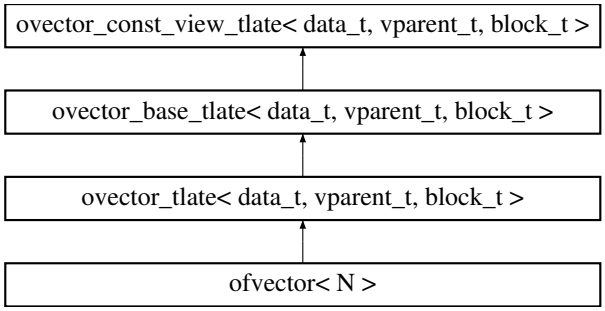
- omatrix_tlate.h

46.233 ofvector< N > Class Template Reference

A vector where the memory allocation is performed in the constructor.

`#include <ovector_tlate.h>`

Inheritance diagram for ofvector< N >:



46.233.1 Detailed Description

template<size_t N = 0>class ofvector< N >

This can be useful, for example to easily make C-style arrays of ovector objects with a fixed size. For example,

```
ofvector<8> x[10];
```

would mean x is a 10-dimensional array of ovector object with initial length 8.

Idea for Future Consider making allocate() and free() functions private for this class?

Definition at line 1896 of file ovector_tlate.h.

The documentation for this class was generated from the following file:

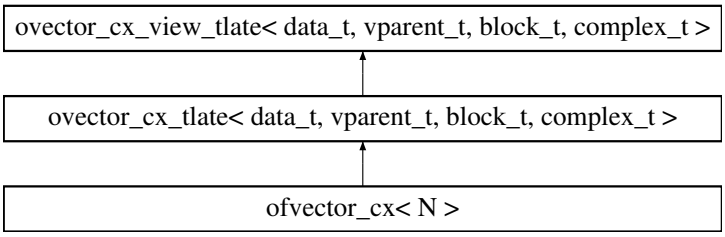
- ovector_tlate.h

46.234 ofvector_cx< N > Class Template Reference

A complex vector where the memory allocation is performed in the constructor.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ofvector_cx< N >:



46.234.1 Detailed Description

template<size_t N = 0>class ofvector_cx< N >

Definition at line 996 of file ovector_cx_tlate.h.

The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

46.235 omatrix_alloc Class Reference

A simple class to provide an `allocate()` function for omatrix.

`#include <omatrix_tlate.h>`

46.235.1 Detailed Description

Definition at line 1311 of file omatrix_tlate.h.

Public Member Functions

- void allocate (omatrix &o, size_t i, size_t j)
  Allocate `v` for `i` elements.
- void free (omatrix &o, size_t i)
  Free memory.

The documentation for this class was generated from the following file:

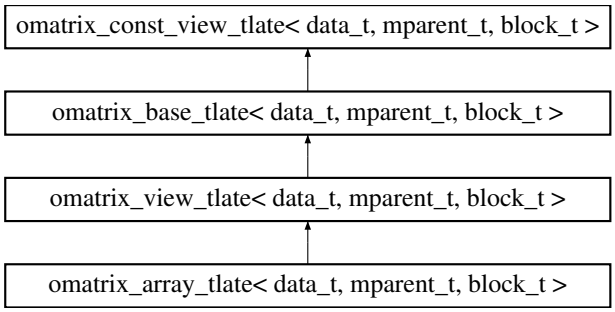
- omatrix_tlate.h

46.236 omatrix_array_tlate< data_t, mparent_t, block_t > Class Template Reference

Create a matrix from an array.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_array_tlate< data_t, mparent_t, block_t >:



46.236.1 Detailed Description

template<class data_t, class mparent_t, class block_t>class omatrix_array_tlate< data_t, mparent_t, block_t >

Definition at line 1015 of file omatrix_tlate.h.

Public Member Functions

- omatrix_array_tlate (data_t *dat, size_t ltda, size_t sz1, size_t sz2)
  Create a vector from `dat` with size `n`.

The documentation for this class was generated from the following file:

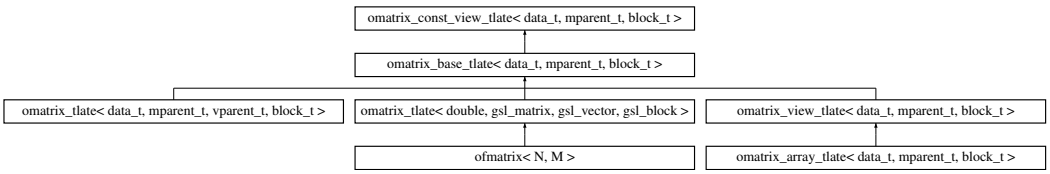
- omatrix_tlate.h

46.237 omatrix_base_tlate< data_t, mparent_t, block_t > Class Template Reference

A base class for omatrix and omatrix_view.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_base_tlate< data_t, mparent_t, block_t >:



46.237.1 Detailed Description

template<class data_t, class mparent_t, class block_t>class omatrix_base_tlate< data_t, mparent_t, block_t >

Definition at line 260 of file omatrix_tlate.h.

Public Member Functions

Copy constructors

- omatrix_base_tlate (const omatrix_base_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- omatrix_base_tlate & operator= (const omatrix_base_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- data_t * operator[ ] (size_t i)
  Array-like indexing.
- const data_t * operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i, size_t j)
  Array-like indexing.
- const data_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- data_t * get_ptr (size_t i, size_t j)
  Get pointer (with optional range-checking)
- int set (size_t i, size_t j, data_t val)
  Set (with optional range-checking)
- int set_all (double val)
  Set all of the value to be the value `val`.

Other methods

- mparent_t * get_gsl_matrix ()
  Return a gsl matrix.

Arithmetic

- omatrix_base_tlate< data_t, mparent_t, block_t > & operator+= (const omatrix_base_tlate< data_t, mparent_t, block_t > &x)
  operator+=
- omatrix_base_tlate< data_t, mparent_t, block_t > & operator-= (const omatrix_base_tlate< data_t, mparent_t, block_t > &x)

operator-=
- omatrix_base_tlate< data_t, mparent_t, block_t > & operator+= (const data_t &y)
  operator+=
- omatrix_base_tlate< data_t, mparent_t, block_t > & operator-= (const data_t &y)
  operator-=
- omatrix_base_tlate< data_t, mparent_t, block_t > & operator*= (const data_t &y)
  operator=*

Protected Member Functions

- omatrix_base_tlate ()
  The default constructor.

The documentation for this class was generated from the following file:

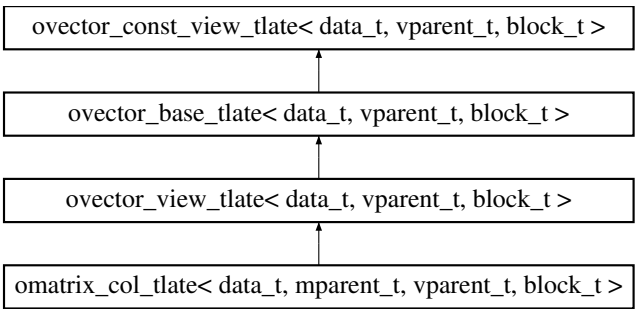
- omatrix_tlate.h

46.238 omatrix_col_tlate< data_t, mparent_t, vparent_t, block_t > Class Template Reference

Create a vector from a column of a matrix.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_col_tlate< data_t, mparent_t, vparent_t, block_t >:



46.238.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t>class omatrix_col_tlate< data_t, mparent_t, vparent_t, block_t >

Idea for Future Also do a umatrix constructor since we can't do that with uvectors

Definition at line 1104 of file omatrix_tlate.h.

Public Member Functions

- omatrix_col_tlate (omatrix_base_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from col `i` of matrix `m`.

The documentation for this class was generated from the following file:

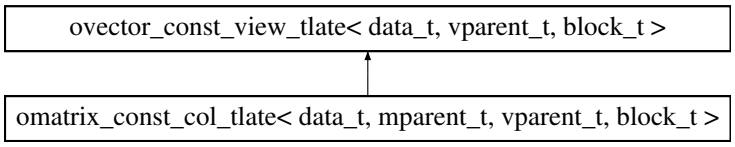
- omatrix_tlate.h

46.239 omatrix_const_col_tlate< data_t, mparent_t, vparent_t, block_t > Class Template Reference

Create a const vector from a column of a matrix.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_const_col_tlate< data_t, mparent_t, vparent_t, block_t >:



46.239.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t>class omatrix_const_col_tlate< data_t, mparent_t, vparent_t, block_t >

Definition at line 1131 of file omatrix_tlate.h.

Public Member Functions

- omatrix_const_col_tlate (const omatrix_base_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from col `i` of matrix `m`.

The documentation for this class was generated from the following file:

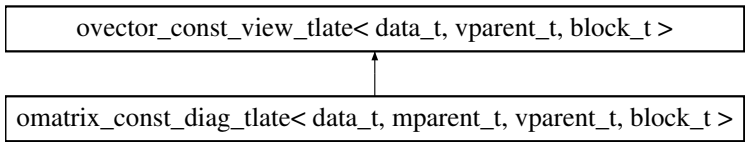
- omatrix_tlate.h

46.240 omatrix_const_diag_tlate< data_t, mparent_t, vparent_t, block_t > Class Template Reference

Create a vector from the main diagonal.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_const_diag_tlate< data_t, mparent_t, vparent_t, block_t >:



46.240.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t>class omatrix_const_diag_tlate< data_t, mparent_t, vparent_t, block_t >

Definition at line 1176 of file omatrix_tlate.h.

Public Member Functions

- omatrix_const_diag_tlate (const omatrix_view_tlate< data_t, mparent_t, block_t > &m)

Create a vector of the diagonal of matrix `m`.

The documentation for this class was generated from the following file:

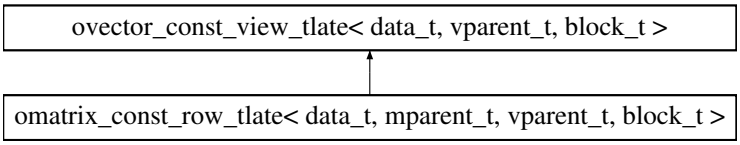
- omatrix_tlate.h

46.241 omatrix_const_row_tlate< data_t, mparent_t, vparent_t, block_t > Class Template Reference

Create a const vector from a row of a matrix.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_const_row_tlate< data_t, mparent_t, vparent_t, block_t >:



46.241.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t>class omatrix_const_row_tlate< data_t, mparent_t, vparent_t, block_t >

Definition at line 1073 of file omatrix_tlate.h.

Public Member Functions

- omatrix_const_row_tlate (const omatrix_base_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from row `i` *of matrix* `m`.

The documentation for this class was generated from the following file:

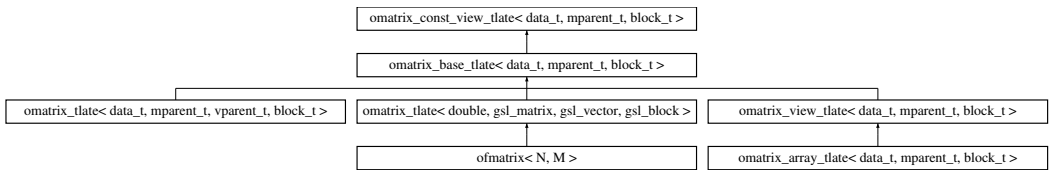
- omatrix_tlate.h

46.242 omatrix_const_view_tlate< data_t, mparent_t, block_t > Class Template Reference

A const matrix view of omatrix objects.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_const_view_tlate< data_t, mparent_t, block_t >:



46.242.1 Detailed Description

template<class data_t, class mparent_t, class block_t>class omatrix_const_view_tlate< data_t, mparent_t, block_t >

Definition at line 55 of file omatrix_tlate.h.

Public Member Functions

Copy constructors

- omatrix_const_view_tlate (const omatrix_const_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- omatrix_const_view_tlate & operator= (const omatrix_const_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- const data_t * operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- data_t get (size_t i, size_t j) const
  Get (with optional range-checking)
- data_t * get_ptr (size_t i, size_t j)
  Get pointer (with optional range-checking)
- const data_t * get_const_ptr (size_t i, size_t j) const
  Get pointer (with optional range-checking)
- size_t rows () const
  Method to return number of rows.
- size_t cols () const
  Method to return number of columns.
- size_t tda () const
  Method to return matrix tda.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.
- data_t max () const
  The largest matrix element.
- data_t min () const
  The smallest matrix element.
- const mparent_t * get_gsl_matrix_const () const
  Return a `const` gsl matrix.

Protected Member Functions

- omatrix_const_view_tlate ()
  The default constructor.

46.242.2 Member Function Documentation

46.242.2.1 template<class data_t, class mparent_t, class block_t> size_t omatrix_const_view_tlate< data_t, mparent_t, block_t >::rows ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 179 of file omatrix_tlate.h.

46.242.2.2 template<class data_t, class mparent_t, class block_t> size_t omatrix_const_view_tlate< data_t, mparent_t, block_t >::cols ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 188 of file omatrix_tlate.h.

46.242.2.3 template<class data_t, class mparent_t, class block_t> size_t omatrix_const_view_tlate< data_t, mparent_t, block_t >::tda ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 197 of file omatrix_tlate.h.

46.242.2.4 template<class data_t, class mparent_t, class block_t> bool omatrix_const_view_tlate< data_t, mparent_t, block_t >::is_owner ( ) const [inline]

This can be used to determine if an object is a "matrix_view", or a "matrix". If is_owner() is true, then it is an omatrix_tlate object.

Definition at line 210 of file omatrix_tlate.h.

The documentation for this class was generated from the following file:

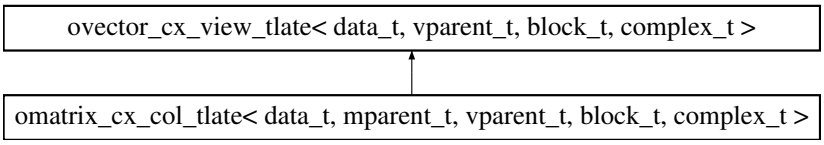
- omatrix_tlate.h

46.243 omatrix_cx_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from a column of a matrix.

`#include <omatrix_cx_tlate.h>`

Inheritance diagram for omatrix_cx_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >:



46.243.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t, class complex_t>class omatrix_cx_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >

Definition at line 659 of file omatrix_cx_tlate.h.

Public Member Functions

- omatrix_cx_col_tlate (omatrix_cx_view_tlate< data_t, mparent_t, block_t, complex_t > &m, size_t i)
  Create a vector from col `i` of matrix `m`.

The documentation for this class was generated from the following file:

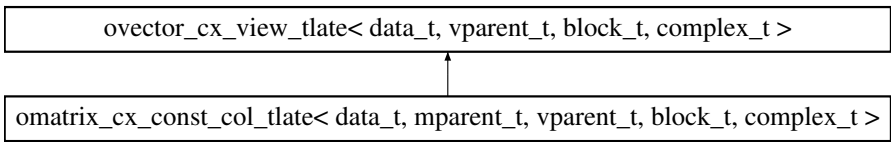
- omatrix_cx_tlate.h

46.244 omatrix_cx_const_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from a column of a matrix.

`#include <omatrix_cx_tlate.h>`

Inheritance diagram for omatrix_cx_const_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >:



46.244.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t, class complex_t>class omatrix_cx_const_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >

Definition at line 679 of file omatrix_cx_tlate.h.

Public Member Functions

- omatrix_cx_const_col_tlate (omatrix_cx_view_tlate< data_t, mparent_t, block_t, complex_t > &m, size_t i)
  Create a vector from col `i` of matrix `m`.

The documentation for this class was generated from the following file:

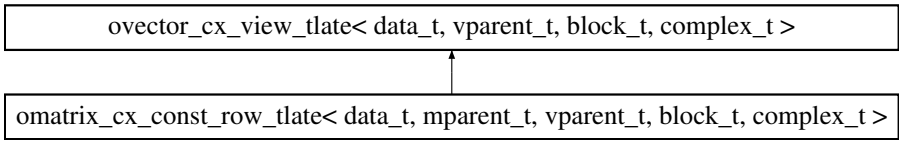
- omatrix_cx_tlate.h

46.245 omatrix_cx_const_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from a row of a matrix.

`#include <omatrix_cx_tlate.h>`

Inheritance diagram for omatrix_cx_const_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >:



46.245.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t, class complex_t>class omatrix_cx_const_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >

Definition at line 639 of file omatrix_cx_tlate.h.

Public Member Functions

- omatrix_cx_const_row_tlate (const omatrix_cx_view_tlate< data_t, mparent_t, block_t, complex_t > &m, size_t i)
  Create a vector from row `i` of matrix `m`.

The documentation for this class was generated from the following file:

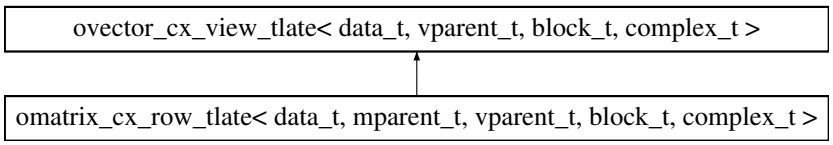
- omatrix_cx_tlate.h

46.246 omatrix_cx_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from a row of a matrix.

`#include <omatrix_cx_tlate.h>`

Inheritance diagram for omatrix_cx_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >:



46.246.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t, class complex_t>class omatrix_cx_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >

Definition at line 619 of file omatrix_cx_tlate.h.

Public Member Functions

- omatrix_cx_row_tlate (omatrix_cx_view_tlate< data_t, mparent_t, block_t, complex_t > &m, size_t i)
  Create a vector from row `i` of matrix `m`.

The documentation for this class was generated from the following file:

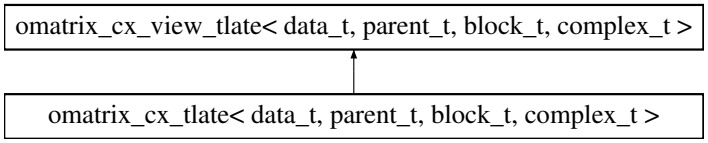
- omatrix_cx_tlate.h

46.247 omatrix_cx_tlate< data_t, parent_t, block_t, complex_t > Class Template Reference

A matrix of double-precision numbers.

`#include <omatrix_cx_tlate.h>`

Inheritance diagram for omatrix_cx_tlate< data_t, parent_t, block_t, complex_t >:



46.247.1 Detailed Description

template<class data_t, class parent_t, class block_t, class complex_t>class omatrix_cx_tlate< data_t, parent_t, block_t, complex_t >

Definition at line 389 of file omatrix_cx_tlate.h.

Public Member Functions

Standard constructor

- omatrix_cx_tlate (size_t r=0, size_t c=0)
  Create an omatrix of size n with owner as 'true'.

Copy constructors

- omatrix_cx_tlate (const omatrix_cx_tlate &v)
  Deep copy constructor, allocate new space and make a copy.
- omatrix_cx_tlate (const omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t > &v)
  Deep copy constructor, allocate new space and make a copy.

Memory allocation

- int allocate (size_t nrows, size_t ncols)
  Allocate memory for size n after freeing any memory presently in use.
- int free ()
  Free the memory.

Other methods

- omatrix_cx_tlate< data_t, parent_t, block_t, complex_t > transpose ()
  Compute the transpose.
- omatrix_cx_tlate< data_t, parent_t, block_t, complex_t > htranspose ()
  Compute the conjugate transpose.

46.247.2 Member Function Documentation

46.247.2.1 template<class data_t, class parent_t, class block_t, class complex_t> int omatrix_cx_tlate< data_t, parent_t, block_t, complex_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 573 of file omatrix_cx_tlate.h.

The documentation for this class was generated from the following file:

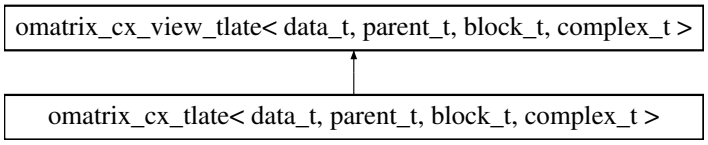
- omatrix_cx_tlate.h

46.248 omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t > Class Template Reference

A matrix view of double-precision numbers.

`#include <omatrix_cx_tlate.h>`

Inheritance diagram for omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t >:



46.248.1 Detailed Description

template<class data_t, class parent_t, class block_t, class complex_t>class omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t >

Definition at line 48 of file omatrix_cx_tlate.h.

Public Member Functions

Copy constructors

- omatrix_cx_view_tlate (const omatrix_cx_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- omatrix_cx_view_tlate & operator= (const omatrix_cx_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- complex_t * operator[ ] (size_t i)
  Array-like indexing.
- const complex_t * operator[ ] (size_t i) const
  Array-like indexing.
- complex_t & operator() (size_t i, size_t j)
  Array-like indexing.
- const complex_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- complex_t get (size_t i, size_t j) const
  Get (with optional range-checking)
- std::complex< data_t > get_stl (size_t i, size_t j) const
  Get STL-like complex number (with optional range-checking)
- data_t real (size_t i, size_t j) const
  Get real part (with optional range-checking)
- data_t imag (size_t i, size_t j) const
  Get imaginary part(with optional range-checking)
- complex_t * get_ptr (size_t i, size_t j)
  Get pointer (with optional range-checking)
- const complex_t * get_const_ptr (size_t i, size_t j) const
  Get pointer (with optional range-checking)
- int set (size_t i, size_t j, complex_t &val)
  Set (with optional range-checking)
- int set (size_t i, size_t j, data_t vr, data_t vi)
  Set (with optional range-checking)
- int set_real (size_t i, size_t j, data_t vr)
  Set (with optional range-checking)
- int set_imag (size_t i, size_t j, data_t vi)
  Set (with optional range-checking)
- int set_all (complex_t &val)
  Set all.
- size_t rows () const
  Method to return number of rows.
- size_t cols () const
  Method to return number of columns.
- size_t tda () const
  Method to return matrix tda.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.

46.248.2 Member Function Documentation

46.248.2.1 template<class data_t, class parent_t, class block_t, class complex_t> size_t omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t >::rows ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 340 of file omatrix_cx_tlate.h.

46.248.2.2 template<class data_t, class parent_t, class block_t, class complex_t> size_t omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t >::cols ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 349 of file omatrix_cx_tlate.h.

46.248.2.3 template<class data_t, class parent_t, class block_t, class complex_t> size_t omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t >::tda ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 358 of file omatrix_cx_tlate.h.

The documentation for this class was generated from the following file:

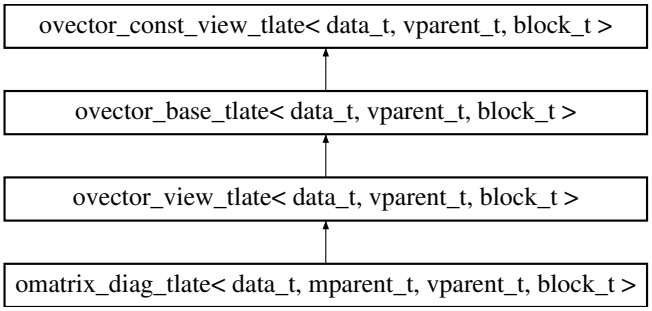
- omatrix_cx_tlate.h

46.249 omatrix_diag_tlate< data_t, mparent_t, vparent_t, block_t > Class Template Reference

Create a vector from the main diagonal.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_diag_tlate< data_t, mparent_t, vparent_t, block_t >:



46.249.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t>class omatrix_diag_tlate< data_t, mparent_t, vparent_t, block_t >

Definition at line 1158 of file omatrix_tlate.h.

Public Member Functions

- omatrix_diag_tlate (omatrix_view_tlate< data_t, mparent_t, block_t > &m)
  Create a vector of the diagonal of matrix m.

The documentation for this class was generated from the following file:

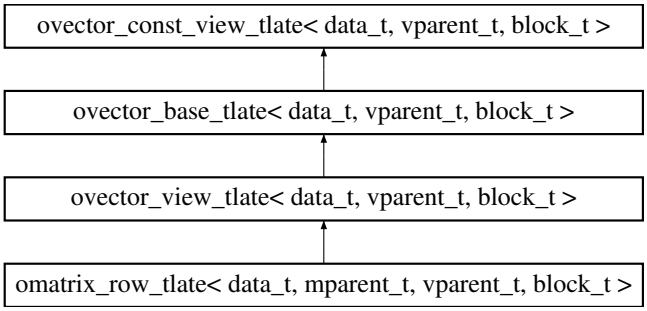
- omatrix_tlate.h

46.250 omatrix_row_tlate< data_t, mparent_t, vparent_t, block_t > Class Template Reference

Create a vector from a row of a matrix.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_row_tlate< data_t, mparent_t, vparent_t, block_t >:



46.250.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t>class omatrix_row_tlate< data_t, mparent_t, vparent_t, block_t >

Definition at line 1045 of file omatrix_tlate.h.

Public Member Functions

- omatrix_row_tlate (omatrix_base_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from row `i` of matrix `m`.

The documentation for this class was generated from the following file:

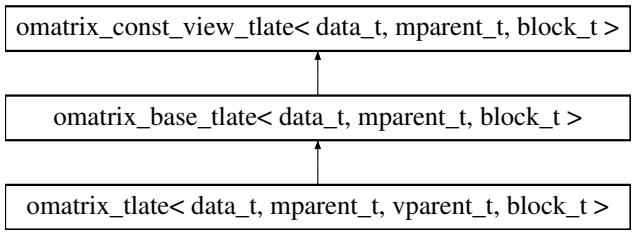
- omatrix_tlate.h

46.251 omatrix_tlate< data_t, mparent_t, vparent_t, block_t > Class Template Reference

A matrix of double-precision numbers.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_tlate< data_t, mparent_t, vparent_t, block_t >:



46.251.1 Detailed Description

template<class data_t, class mparent_t, class vparent_t, class block_t>class omatrix_tlate< data_t, mparent_t, vparent_t, block_t >

The basic matrix classes are built upon this template. A matrix of double-precision numbers is an object of type omatrix , which is just a `typedef` defined using this class template. See Arrays, Vectors, Matrices and Tensors in the User's Guide for more information.

Definition at line 665 of file omatrix_tlate.h.

Public Member Functions

Standard constructor

- omatrix_tlate (size_t r=0, size_t c=0)
  Create an omatrix of size `n` with owner as `true`.

Copy constructors

- omatrix_tlate (const omatrix_tlate &v)
  Deep copy constructor, allocate new space and make a copy.
- omatrix_tlate (const omatrix_const_view_tlate< data_t, mparent_t, block_t > &v)
  Deep copy constructor, allocate new space and make a copy.
- omatrix_tlate & operator= (const omatrix_tlate &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- omatrix_tlate & operator= (const omatrix_const_view_tlate< data_t, mparent_t, block_t > &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- omatrix_tlate (size_t n, ovector_base_tlate< data_t, vparent_t, block_t > ova[])
  Deep copy from an array of ovectors.
- omatrix_tlate (size_t n, uvector_base_tlate< data_t > uva[])
  Deep copy from an array of uvectors.
- omatrix_tlate (size_t n, size_t n2, data_t **csa)
  Deep copy from a C-style 2-d array.

Memory allocation

- int allocate (size_t nrows, size_t ncols)
  Allocate memory after freeing any memory presently in use.
- int free ()
  Free the memory.

Other methods

- omatrix_tlate< data_t, mparent_t, vparent_t, block_t > transpose ()
  Compute the transpose (even if matrix is not square)

46.251.2 Member Function Documentation

46.251.2.1 template<class data_t, class mparent_t, class vparent_t, class block_t> int omatrix_tlate< data_t, mparent_t, vparent_t, block_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 975 of file omatrix_tlate.h.

The documentation for this class was generated from the following file:

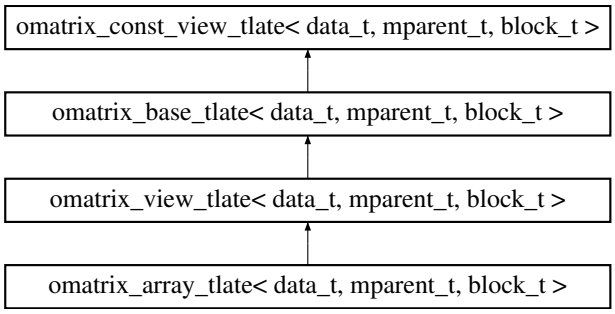
- omatrix_tlate.h

46.252 omatrix_view_tlate< data_t, mparent_t, block_t > Class Template Reference

A matrix view of double-precision numbers.

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_view_tlate< data_t, mparent_t, block_t >:



46.252.1 Detailed Description

template<class data_t, class mparent_t, class block_t>class omatrix_view_tlate< data_t, mparent_t, block_t >

This is a matrix view, which views a matrix stored somewhere else. For a full matrix template class with its own memory, see omatrix_tlate . The basic matrix classes are built upon these templates. A matrix of double-precision numbers is an object of type omatrix . See Arrays, Vectors, Matrices and Tensors in the User's Guide for more information.

Definition at line 506 of file omatrix_tlate.h.

Public Member Functions

Copy constructors

- omatrix_view_tlate (const omatrix_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- omatrix_view_tlate & operator= (const omatrix_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- omatrix_view_tlate (omatrix_base_tlate< data_t, mparent_t, block_t > &v)
  Shallow copy constructor - create a new view of the same matrix.
- omatrix_view_tlate & operator= (omatrix_base_tlate< data_t, mparent_t, block_t > &v)
  Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- data_t * operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- data_t * get_ptr (size_t i, size_t j) const
  Get pointer (with optional range-checking)
- int set (size_t i, size_t j, data_t val) const
  Set (with optional range-checking)
- int set_all (double val) const
  Set all of the value to be the value `val`.

Protected Member Functions

- omatrix_view_tlate ()

The default constructor.

The documentation for this class was generated from the following file:

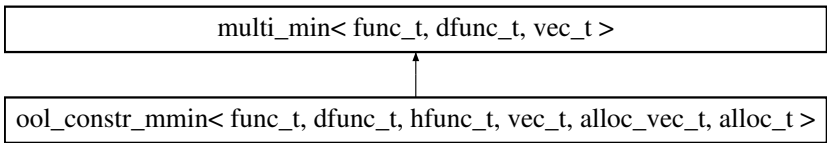
- omatrix_tlate.h

46.253 ool_constr_mmin< func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Constrained multidimensional minimization (OOL) [abstract base].

`#include <ool_constr_mmin.h>`

Inheritance diagram for ool_constr_mmin< func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >:



46.253.1 Detailed Description

template<class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class ool_constr_mmin< func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >

Idea for Future Implement automatic computations of gradient and Hessian

Idea for Future Construct a more difficult example for the "examples" directory

Idea for Future Finish mmin() interface

Idea for Future Implement a direct computation of the hessian as the jacobian of the gradient

Definition at line 187 of file ool_constr_mmin.h.

Public Member Functions

- virtual int allocate (const size_t n)
  Allocate memory.
- virtual int free ()
  Free previously allocated memory.
- virtual int restart ()
  Restart the minimizer.
- virtual int set (func_t &fn, dfunc_t &dfn, vec_t &init)
  Set the function, the gradient, and the initial guess.
- virtual int set_hess (func_t &fn, dfunc_t &dfn, hfunc_t &hfn, vec_t &init)
  Set the function, the gradient, the Hessian product, and the initial guess.
- virtual int set_constraints (size_t nc, vec_t &lower, vec_t &upper)
  Set the constraints.
- virtual int iterate ()=0
  Perform an iteration.
- virtual int is_optimal ()=0
  See if we're finished.
- virtual int mmin (size_t nvar, vec_t &xx, double &fmin, func_t &ff)

Calculate the minimum `min` *of* `func` *w.r.t. the array* `x` *of size* `nvar`.

- virtual int mmin_hess (size_t nvar, vec_t &xx, double &fmin, func_t &ff, dfunc_t &df, hfunc_t &hf)
  Calculate the minimum `min` *of* `ff` *w.r.t. the array* `x` *of size* `nvar` *with gradient* `df` *and hessian vector product* `hf`.
- virtual int mmin_de (size_t nvar, vec_t &xx, double &fmin, func_t &ff, dfunc_t &df)
  Calculate the minimum `min` *of* `func` *w.r.t. the array* `x` *of size* `nvar` *with gradient* `dfunc`.
- const char * type ()
  Return string denoting type ("ool_constr_mmin")

Protected Member Functions

- void shrink (const size_t nind, gsl_vector_uint *Ind, const vec_t &V)
  Shrink vector `V` *from the full to the reduced space.*
- void expand (const size_t nind, gsl_vector_uint *Ind, const vec_t &V)
  Expand vector `V` *from the reduced to the full space.*
- double calc_f (const size_t nind, gsl_vector_uint *Ind, vec_t &X, vec_t &Xc)
  Evaluate the objective function from the reduced space.
- int calc_g (const size_t nind, gsl_vector_uint *Ind, vec_t &X, vec_t &Xc, vec_t &G)
  Compute gradient in the reduced space.
- int calc_Hv (const size_t nind, gsl_vector_uint *Ind, vec_t &X, vec_t &Xc, vec_t &V, vec_t &Hv)
  Evaluate a hessian times a vector from the reduced space.

Protected Attributes

- double f
  The current function value.
- double size
  Desc.
- alloc_t ao
  Memory allocation object.
- alloc_vec_t x
  The current minimum vector.
- alloc_vec_t gradient
  The current gradient vector.
- alloc_vec_t dx
  Desc.
- size_t fcount
  Number of function evaluations.
- size_t gcount
  Number of gradient evaluations.
- size_t hcount
  Number of Hessian evaluations.
- size_t dim
  Number of parameters.
- size_t nconstr
  Number of constraints.
- func_t * func
  User-supplied function.
- dfunc_t * dfunc
  Gradient function.
- hfunc_t * hfunc
  Hessian function.
- alloc_vec_t L
  Lower bound constraints.
- alloc_vec_t U
  Upper bound constraints.
- bool requires_hess
  If true, the algorithm requires the hessian vector product.

46.253.2 Member Function Documentation

46.253.2.1 template<class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> int ool_constr_mmin< func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >::calc_Hv (const size_t *nind*, gsl_vector_uint * *Ind*, vec_t & *X*, vec_t & *Xc*, vec_t & *V*, vec_t & *Hv*) [inline, protected]

Expand to full space

Definition at line 326 of file ool_constr_mmin.h.

46.253.2.2 template<class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> virtual int ool_constr_mmin< func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >::mmin (size_t *nvar*, vec_t & *xx*, double & *fmin*, func_t & *ff*) [inline, virtual]

Note

This is unimplemented.

Implements multi_min< func_t, dfunc_t, vec_t >.

Definition at line 476 of file ool_constr_mmin.h.

The documentation for this class was generated from the following file:

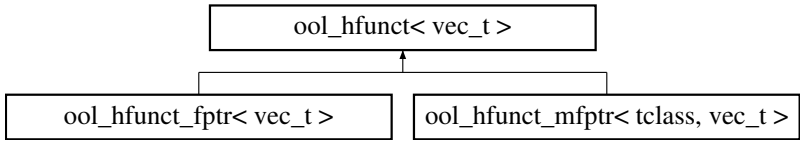
- ool_constr_mmin.h

46.254 ool_hfunct< vec_t > Class Template Reference

Hessian product function for ool_constr_mmin [abstract base].

`#include <ool_constr_mmin.h>`

Inheritance diagram for ool_hfunct< vec_t >:



46.254.1 Detailed Description

template<class vec_t = ovector_base>class ool_hfunct< vec_t >

Definition at line 60 of file ool_constr_mmin.h.

Public Member Functions

- virtual int operator() (size_t nv, const vec_t &x, const vec_t &v, vec_t &hv)=0
  Evaluate $H(x) \cdot v$.

The documentation for this class was generated from the following file:

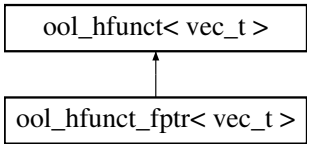
- ool_constr_mmin.h

46.255 ool_hfunct_fptr< vec_t > Class Template Reference

A hessian product supplied by a function pointer.

`#include <ool_constr_mmin.h>`

Inheritance diagram for ool_hfunct_fptr< vec_t >:



46.255.1 Detailed Description

template<class vec_t = ovector_base>class ool_hfunct_fptr< vec_t >

Definition at line 87 of file ool_constr_mmin.h.

Public Member Functions

- ool_hfunct_fptr (int(∗fp)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, const vec_t &v, vec_t &hv)
  Evaluate $H(x) \cdot v$.

Protected Attributes

- int(∗ fptr)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv)
  Store the function pointer.

Private Member Functions

- **ool_hfunct_fptr** (const ool_hfunct_fptr &)
- ool_hfunct_fptr & **operator=** (const ool_hfunct_fptr &)

The documentation for this class was generated from the following file:

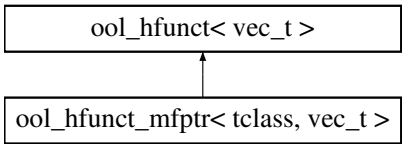
- ool_constr_mmin.h

46.256 ool_hfunct_mfptr< tclass, vec_t > Class Template Reference

A hessian product supplied by a member function pointer.

`#include <ool_constr_mmin.h>`

Inheritance diagram for ool_hfunct_mfptr< tclass, vec_t >:



46.256.1 Detailed Description

template<class tclass, class vec_t = ovector_base>class ool_hfunct_mfptr< tclass, vec_t >

Definition at line 132 of file ool_constr_mmin.h.

Public Member Functions

- ool_hfunct_mfptr (tclass *tp, int(tclass::*fp)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv))
  Specify the class instance and member function pointer.
- virtual int operator() (size_t nv, const vec_t &x, const vec_t &v, vec_t &hv)
  Evaluate $H(x) \cdot v$.

Protected Attributes

- int(tclass::* fptr)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv)
  Store the function pointer.
- tclass * tptr
  Store a pointer to the class instance.

Private Member Functions

- **ool_hfunct_mfptr** (const ool_hfunct_mfptr &)
- ool_hfunct_mfptr & **operator=** (const ool_hfunct_mfptr &)

The documentation for this class was generated from the following file:

- ool_constr_mmin.h

46.257 ool_mmin_gencan< param_t, func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Constrained minimization by the "GENCAN" method (OOL)

`#include <ool_mmin_gencan.h>`

46.257.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class ool_mmin_gencan< param_t, func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >

Note

Not working yet

Definition at line 68 of file ool_mmin_gencan.h.

Public Member Functions

- virtual int alloc (const size_t n)
  Allocate memory.
- virtual int free ()
  Free previously allocated memory.
- virtual int set (func_t &fn, dfunc_t &dfn, hfunc_t &hfn, vec_t &init, param_t &par)

Set the function, the initial guess, and the parameters.

- virtual int restart ()
  Restart the minimizer.
- virtual int iterate ()
  Perform an iteration.
- virtual int is_optimal ()
  See if we're finished.
- const char * type ()
  Return string denoting type ("ool_mmin_gencan")

Data Fields

- double epsgpen
  Tolerance on Euclidean norm of projected gradient (default 1.0e-5)
- double epsgpsn
  Tolerance on infinite norm of projected gradient (default 1.0e-5)
- double fmin
  Minimum function value (default 10^{-99})
- double udelta0
  Trust-region radius (default -1.0)
- double ucgmia
  Maximum interations per variable (default -1.0)
- double ucgmib
  Extra maximum iterations (default -1.0)
- int cg_scre
  Conjugate gradient condition type (default 1)
- double cg_gpnf
  Projected gradient norm (default 1.0e-5)
- double cg_epsi
  Desc (default 1.0e-1)
- double cg_epsf
  Desc (default 1.0e-5)
- double cg_epsnqmp
  Stopping fractional tolerance for conjugate gradient (default 1.0e-4)
- int cg_maxitnqmp
  Maximum iterations for conjugate gradient (default 5)
- int nearlyq
  Set to 1 if the function is nearly quadratic (default 0)
- double nint
  Interpolation constant (default 2.0)
- double next
  Extrapolation constant (default 2.0)
- int mininterp
  Minimum interpolation size (default 4)
- int maxextrap
  Maximum extrapolations in truncated Newton direction (default 100)
- int trtype
  Type of trust region (default 0)
- double eta
  Threshold for abandoning current face (default 0.9)
- double delmin
  Minimum trust region for truncated Newton direction (default 0.1)
- double lspgmi
  Minimum spectral steplength (default 1.0e-10)
- double lspgma
  Maximum spectral steplength (default 1.0e10)
- double theta
  Constant for the angle condition (default 1.0e-6)
- double gamma
  Constant for Armijo condition (default 1.0e-4)

- double beta
  Constant for beta condition (default 0.5)
- double sigma1
  Lower bound to the step length reduction (default 0.1)
- double sigma2
  Upper bound to the step length reduction (default 0.9)
- double epsrel
  Relative small number (default 1.0e-7)
- double epsabs
  Absolute small number (default 1.0e-10)
- double infrel
  Relative infinite number (default 1.0e20)
- double infabs
  Absolute infinite number (default 1.0e99)

Protected Attributes

- double cg_src
  Desc (default 1.0)
- alloc_vec_t S
  Temporary vector.
- alloc_vec_t Y
  Temporary vector.
- alloc_vec_t D
  Temporary vector.
- alloc_vec_t cg_W
  Temporary vector.
- alloc_vec_t cg_R
  Temporary vector.
- alloc_vec_t cg_D
  Temporary vector.
- alloc_vec_t cg_Sprev
  Temporary vector.
- alloc_vec_t Xtrial
  Temporary vector.
- alloc_vec_t tnls_Xtemp
  Temporary vector.
- alloc_vec_t near_l
  Temporary vector.
- alloc_vec_t near_u
  Temporary vector.
- int * Ind
  Desc.

46.257.2 Field Documentation

46.257.2.1 template<class param_t , class func_t , class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double ool_mmin_gencan< param_t, func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >::fmin

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 705 of file ool_mmin_gencan.h.

The documentation for this class was generated from the following file:

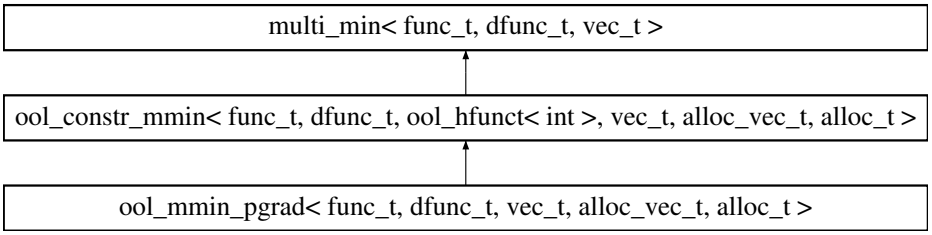
- ool_mmin_gencan.h

46.258 ool_mmin_pgrad< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Constrained minimization by the projected gradient method (OOL)

`#include <ool_mmin_pgrad.h>`

Inheritance diagram for ool_mmin_pgrad< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >:



46.258.1 Detailed Description

template<class func_t = multi_funct<>, class dfunc_t = grad_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class ool_mmin_pgrad< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >

This is the simple extension of the steepest descent algorithm to constrained minimization. Each step is a line search is performed along the projected gradient direction subject to the specified constraints.

This algorithm is likely not ideal for most problems and is provided mostly for demonstration and educational purposes. Based on implementation of Kelley99 in OOL.

Default template arguments

- `func_t` - multi_funct<>
- `dfunc_t` - grad_funct<>
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc

Idea for Future Replace the explicit norm computation below with the more accurate dnrm2 from linalg

Idea for Future Replace the generic variable 'tol' with 'tolf' or 'tolx' from multi_min.

Definition at line 85 of file ool_mmin_pgrad.h.

Public Member Functions

- virtual int allocate (const size_t n)
  Allocate memory.
- virtual int free ()
  Free previously allocated memory.
- virtual int set (func_t &fn, dfunc_t &dfn, vec_t &init)
  Set the function, the initial guess, and the parameters.
- virtual int restart ()
  Restart the minimizer.
- virtual int iterate ()

Perform an iteration.
- virtual int is_optimal ()
  See if we're finished.
- const char * type ()
  Return string denoting type ("ool_mmin_pgrad")

Data Fields

- double fmin
  Minimum function value (default 10^{-99})
- double tol
  Tolerance on infinite norm.
- double alpha
  Constant for the sufficient decrease condition (default 10^{-4})
- double sigma1
  Lower bound to the step length reduction.
- double sigma2
  Upper bound to the step length reduction.

Protected Types

- typedef ool_hfunct< int > hfunc_t
  A convenient typedef for the unused Hessian product type.

Protected Member Functions

- int proj (vec_t &xt)
  Project into feasible region.
- int line_search ()
  Line search.

Protected Attributes

- alloc_vec_t xx
  Temporary vector.

46.258.2 Field Documentation

46.258.2.1 template<class func_t = multi_funct<>, class dfunc_t = grad_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double ool_mmin_pgrad< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >::fmin

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 174 of file ool_mmin_pgrad.h.

The documentation for this class was generated from the following file:

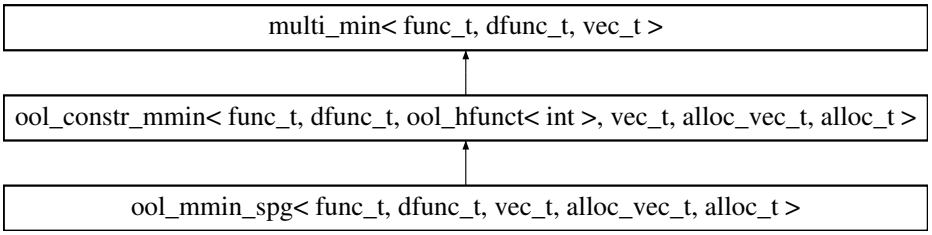
- ool_mmin_pgrad.h

46.259 ool_mmin_spg< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t > Class Template Reference

Constrained minimization by the spectral projected gradient method (OOL)

`#include <ool_mmin_spg.h>`

Inheritance diagram for ool_mmin_spg< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >:



46.259.1 Detailed Description

template<class func_t = multi_funct<>, class dfunc_t = grad_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class ool_mmin_spg< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >

This class applies a non-monotone line search strategy to the classical projected gradient method.

As in Birgin00, this class applies a nonmonotone Armijo sufficient decrease condition for accepting trial points as an improvement over the classical spectral projected gradient method. This method may be competitive with large problems because it has low memory requirements.

Default template arguments

- `func_t` - multi_funct<>
- `dfunc_t` - grad_funct<>
- `vec_t` - ovector_base
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc

Idea for Future There is some memory allocation which isn't deallocated until the destructor, and should be handled a bit more elegantly.

Definition at line 84 of file ool_mmin_spg.h.

Public Member Functions

- virtual int allocate (const size_t n)
  Allocate memory.
- virtual int free ()
  Free previously allocated memory.
- virtual int set (func_t &fn, dfunc_t &dfn, vec_t &init)
  Set the function, the initial guess, and the parameters.
- virtual int restart ()
  Restart the minimizer.
- virtual int iterate ()
  Perform an iteration.
- virtual int is_optimal ()
  See if we're finished.
- const char * type ()
  Return string denoting type ("ool_mmin_spg")

Data Fields

- double fmin
  Minimum function value (default 10^{-99})
- double tol
  Tolerance on infinite norm (default 10^{-4})
- double alphamin
  Lower bound to spectral step size (default 10^{-30})
- double alphamax
  Upper bound to spectral step size (default 10^{30})
- double gamma
  Sufficient decrease parameter (default 10^{-4})
- double sigma1
  Lower bound to the step length reduction (default 0.1)
- double sigma2
  Upper bound to the step length reduction (default 0.9)
- size_t M
  Monotonicity parameter (M=1 forces monotonicity) (default 10)

Protected Types

- typedef ool_hfunct< int > hfunc_t
  A convenient typedef for the unused Hessian product type.

Protected Member Functions

- int line_search ()
  Line search.
- int proj (vec_t &xt)
  Project into feasible region.

Protected Attributes

- double alpha
  Armijo parameter.
- alloc_vec_t xx
  Temporary vector.
- alloc_vec_t d
  Temporary vector.
- alloc_vec_t s
  Temporary vector.
- alloc_vec_t y
  Temporary vector.
- alloc_vec_t fvec
  Temporary vector.
- size_t m
  Non-monotone parameter.
- int tail
  Desc.

46.259.2 Field Documentation

46.259.2.1 template<class func_t = multi_funct<>, class dfunc_t = grad_funct<>, class vec_t = ovector_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> double ool_mmin_spg< func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >::fmin

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 237 of file ool_mmin_spg.h.

The documentation for this class was generated from the following file:

- ool_mmin_spg.h

46.260 other_todos_and_bugs Class Reference

An empty class to add some items to the todo and bug lists.

46.260.1 Detailed Description

Todo
- Make sure an error message is printed to ensure users do 'make o2scl-test' and not 'make check'
- Make sure cstdlib is included wherever exit() is used [5/22/11 - Most of these are taken care of now.]

Idea for Future
- Make sure we have a uvector_cx_alloc, ovector_cx_const_reverse, ovector_cx_const_subvector_reverse, uvector_reverse, uvector_const_reverse, uvector_subvector_reverse, uvector_const_subvector_reverse, omatrix_cx_diag, blah_const_diag, umatrix_diag, and umatrix_cx_diag
- ovector_cx_view::operator=(uvector_cx_view &) is missing
- ovector_cx::operator=(uvector_cx_view &) is missing
- uvector_c_view::operator+=(complex) is missing
- uvector_c_view::operator-=(complex) is missing
- uvector_c_view::operator*=(complex) is missing

Idea for Future Currently, I believe the testing code requires the shared libraries and may not work if the static libraries only are installed. It would be nice to ensure the tests could be complied in either case.

Definition at line 159 of file main.dox.

The documentation for this class was generated from the following file:

- main.dox

46.261 ovector_alloc Class Reference

A simple class to provide an `allocate()` function for ovector.

```
#include <ovector_tlate.h>
```

46.261.1 Detailed Description

Idea for Future Could (or should?) the allocate() functions be adapted to return an integer error value?

Definition at line 1862 of file ovector_tlate.h.

Public Member Functions

- void allocate (ovector &o, size_t i)
  Allocate `v` for `i` elements.
- void free (ovector &o)
  Free memory.

The documentation for this class was generated from the following file:

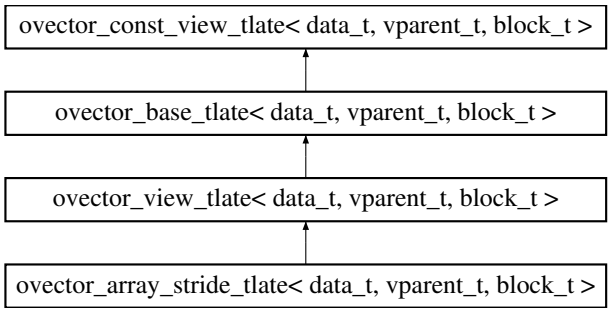
- ovector_tlate.h

46.262 ovector_array_stride_tlate< data_t, vparent_t, block_t > Class Template Reference

Create a vector from an array with a stride.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_array_stride_tlate< data_t, vparent_t, block_t >:



46.262.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_array_stride_tlate< data_t, vparent_t, block_t >

Definition at line 1615 of file ovector_tlate.h.

Public Member Functions

- ovector_array_stride_tlate (size_t n, data_t *dat, size_t s)
  Create a vector from `dat` with size `n` and stride `s`.

The documentation for this class was generated from the following file:

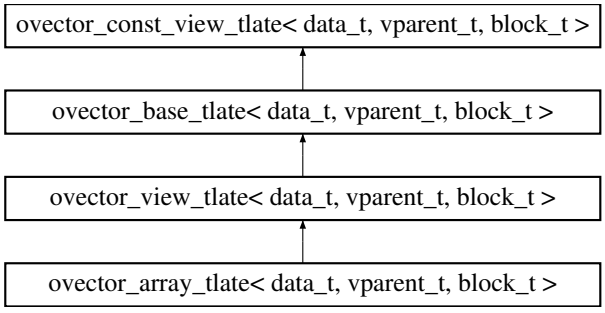
- ovector_tlate.h

46.263 ovector_array_tlate< data_t, vparent_t, block_t > Class Template Reference

Create a vector from an array.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_array_tlate< data_t, vparent_t, block_t >:



46.263.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_array_tlate< data_t, vparent_t, block_t >

Definition at line 1590 of file ovector_tlate.h.

Public Member Functions

- ovector_array_tlate (size_t n, data_t *dat)
  Create a vector from `dat` *with size* `n`.

The documentation for this class was generated from the following file:

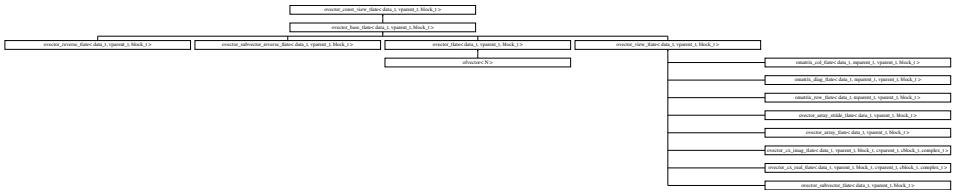
- ovector_tlate.h

46.264 ovector_base_tlate< data_t, vparent_t, block_t > Class Template Reference

A base class for ovector and ovector_view.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_base_tlate< data_t, vparent_t, block_t >:



46.264.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_base_tlate< data_t, vparent_t, block_t >

This class provides a base class for ovector and ovector_view, mostly useful for creating function arguments which accept either ovector or ovector_view.

Definition at line 640 of file ovector_tlate.h.

Public Types

- typedef ovector_const_view_tlate < data_t, vparent_t, block_t > ::const_iterator const_iterator
  Desc.
- typedef ovector_const_view_tlate < data_t, vparent_t, block_t > ::iterator iterator
  Desc.

Public Member Functions

- const_iterator begin () const
  Desc.
- const_iterator end () const
  Desc.
- iterator begin ()
  Desc.
- iterator end ()
  Desc.

Copy constructors

- ovector_base_tlate (ovector_base_tlate &v)
  Shallow copy constructor - create a new view of the same vector.
- ovector_base_tlate & operator= (ovector_base_tlate &v)
  Shallow copy constructor - create a new view of the same vector.

Get and set methods

- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing with operator()
- data_t & operator[ ] (size_t i)
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing with operator()
- data_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- int set (size_t i, data_t val)
  Set (with optional range-checking)
- int set_all (data_t val)
  Set all of the value to be the value `val`.

Arithmetic

- ovector_base_tlate< data_t, vparent_t, block_t > & operator+= (const ovector_base_tlate< data_t, vparent_t, block_t > &x)
  operator+=
- ovector_base_tlate< data_t, vparent_t, block_t > & operator-= (const ovector_base_tlate< data_t, vparent_t, block_t > &x)
  operator-=
- ovector_base_tlate< data_t, vparent_t, block_t > & operator+= (const data_t &x)
  operator+=
- ovector_base_tlate< data_t, vparent_t, block_t > & operator-= (const data_t &x)
  operator-=
- ovector_base_tlate< data_t, vparent_t, block_t > & operator*= (const data_t &y)
  operator=*

Other methods

- vparent_t * get_gsl_vector ()
  Return a gsl vector.
- const vparent_t * get_gsl_vector_const () const
  Return a `const` gsl vector.

Protected Member Functions

- ovector_base_tlate ()
  Empty constructor for use by children.

46.264.2 Constructor & Destructor Documentation

46.264.2.1 template<class data_t, class vparent_t, class block_t> ovector_base_tlate< data_t, vparent_t, block_t >::ovector_base_tlate ( ) [inline, protected]

Definition at line 898 of file ovector_tlate.h.

46.264.3 Member Function Documentation

46.264.3.1 template<class data_t, class vparent_t, class block_t> int ovector_base_tlate< data_t, vparent_t, block_t >::set_all (data_t *val*) [inline]

If the vector is empty, this function does not perform any assignment and does not call the error handler.

Definition at line 801 of file ovector_tlate.h.

46.264.3.2 template<class data_t, class vparent_t, class block_t> ovector_base_tlate<data_t,vparent_t,block_t>& ovector_base_tlate< data_t, vparent_t, block_t >::operator+= (const ovector_base_tlate< data_t, vparent_t, block_t > & *x*) [inline]

This operator only operates on elements which are present in both vectors, i.e. elements which are present in one vector but missing in the other will be ignored. If one of the two vectors is empty, this function does nothing and does not call the error handler.

Definition at line 820 of file ovector_tlate.h.

46.264.3.3 template<class data_t, class vparent_t, class block_t> ovector_base_tlate<data_t,vparent_t,block_t>& ovector_base_tlate< data_t, vparent_t, block_t >::operator-= (const ovector_base_tlate< data_t, vparent_t, block_t > & *x*) [inline]

This operator only operates on elements which are present in both vectors, i.e. elements which are present in one vector but missing in the other will be ignored. If one of the two vectors is empty, this function does nothing and does not call the error handler.

Definition at line 837 of file ovector_tlate.h.

46.264.3.4 template<class data_t, class vparent_t, class block_t> ovector_base_tlate<data_t,vparent_t,block_t>& ovector_base_tlate< data_t, vparent_t, block_t >::operator+= (const data_t & *x*) [inline]

If the vector is empty, this function does not perform any modifications and does not call the error handler.

Definition at line 850 of file ovector_tlate.h.

46.264.3.5 template<class data_t, class vparent_t, class block_t> ovector_base_tlate<data_t,vparent_t,block_t>& ovector_base_tlate< data_t, vparent_t, block_t >::operator-= (const data_t & *x*) [inline]

If the vector is empty, this function does not perform any modifications and does not call the error handler.

Definition at line 860 of file ovector_tlate.h.

46.264.3.6 template<class data_t, class vparent_t, class block_t> ovector_base_tlate<data_t,vparent_t,block_t>& ovector_base_tlate< data_t, vparent_t, block_t >::operator*= (const data_t & *y*) [inline]

If the vector is empty, this function does not perform any modifications and does not call the error handler.

Definition at line 871 of file ovector_tlate.h.

The documentation for this class was generated from the following file:

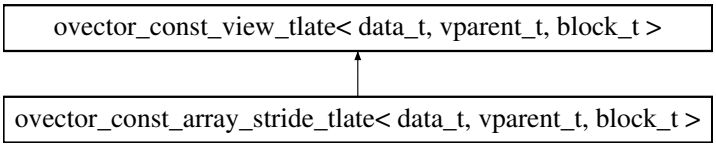
- ovector_tlate.h

46.265 ovector_const_array_stride_tlate< data_t, vparent_t, block_t > Class Template Reference

Create a const vector from an array with a stride.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_const_array_stride_tlate< data_t, vparent_t, block_t >:



46.265.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_array_stride_tlate< data_t, vparent_t, block_t >

The constructor will fail if the size argument `n` is zero, the stride `s` is zero, or if the pointer `dat` is 0.

Definition at line 1716 of file ovector_tlate.h.

Public Member Functions

- ovector_const_array_stride_tlate (size_t n, const data_t *dat, size_t s)
  Create a vector from `dat` with size `n`.

The documentation for this class was generated from the following file:

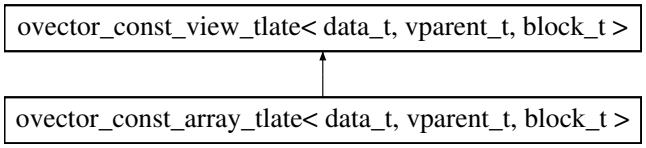
- ovector_tlate.h

46.266 ovector_const_array_tlate< data_t, vparent_t, block_t > Class Template Reference

Create a const vector from an array.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_const_array_tlate< data_t, vparent_t, block_t >:



46.266.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_array_tlate< data_t, vparent_t, block_t >

The constructor will fail if the size argument `n` is zero or if the pointer `dat` is 0.

Definition at line 1680 of file ovector_tlate.h.

Public Member Functions

- ovector_const_array_tlate (size_t n, const data_t *dat)
  Create a vector from `dat` *with size* `n`*.*

The documentation for this class was generated from the following file:

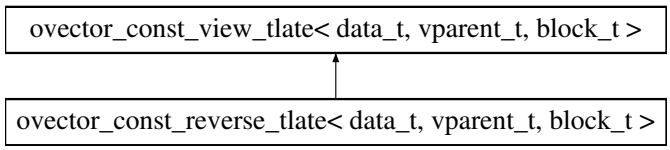
- ovector_tlate.h

46.267 ovector_const_reverse_tlate< data_t, vparent_t, block_t > Class Template Reference

Reversed view of a vector.

`#include <ovector_rev_tlate.h>`

Inheritance diagram for ovector_const_reverse_tlate< data_t, vparent_t, block_t >:



46.267.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_reverse_tlate< data_t, vparent_t, block_t >

Warning

At present, reversing a reversed vector does not give the original ordering.

Idea for Future I think that maybe in order to ensure that this isn't created from an already reversed vector, this class has to be separated from the hierarchy altogether. However, I think this might break the smart interpolation stuff.

Definition at line 229 of file ovector_rev_tlate.h.

Public Member Functions

- ovector_const_reverse_tlate (const ovector_const_view_tlate< data_t, vparent_t, block_t > &v)
  Create a vector from `dat` *with size* `n` *and stride* `s`*.*

Get and set methods

- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.
- data_t get (size_t i) const
  Get (with optional range-checking)
- const data_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)

The documentation for this class was generated from the following file:

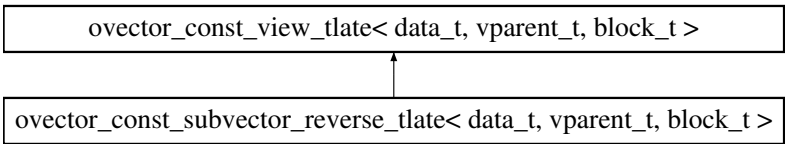
- ovector_rev_tlate.h

46.268 ovector_const_subvector_reverse_tlate< data_t, vparent_t, block_t > Class Template Reference

Reversed view of a const subvector.

`#include <ovector_rev_tlate.h>`

Inheritance diagram for ovector_const_subvector_reverse_tlate< data_t, vparent_t, block_t >:



46.268.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_subvector_reverse_tlate< data_t, vparent_t, block_t >

Warning

At present, reversing a reversed vector does not give the original ordering.

Definition at line 483 of file ovector_rev_tlate.h.

Public Member Functions

- ovector_const_subvector_reverse_tlate (const ovector_const_view_tlate< data_t, vparent_t, block_t > &v, size_t offset, size_t n)
  Create a vector from `dat` with size `n` and stride `s`.

Get and set methods

- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.
- data_t get (size_t i) const
  Get (with optional range-checking)
- const data_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)

The documentation for this class was generated from the following file:

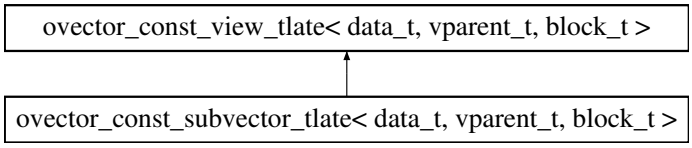
- ovector_rev_tlate.h

46.269 ovector_const_subvector_tlate< data_t, vparent_t, block_t > Class Template Reference

Create a const vector from a subvector of another vector.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_const_subvector_tlate< data_t, vparent_t, block_t >:



46.269.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_subvector_tlate< data_t, vparent_t, block_t >

Definition at line 1747 of file ovector_tlate.h.

Public Member Functions

- ovector_const_subvector_tlate (const ovector_const_view_tlate< data_t, vparent_t, block_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

The documentation for this class was generated from the following file:

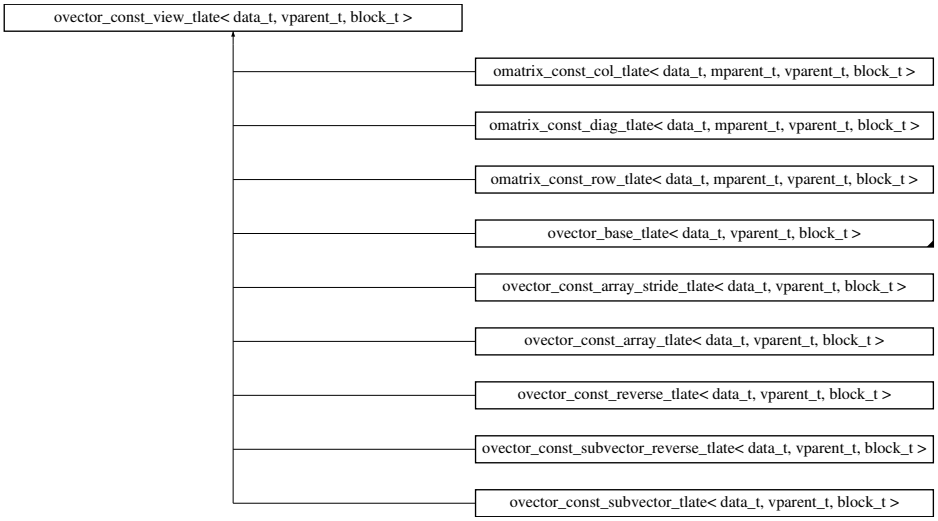
- ovector_tlate.h

46.270 ovector_const_view_tlate< data_t, vparent_t, block_t > Class Template Reference

A const vector view with finite stride.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_const_view_tlate< data_t, vparent_t, block_t >:



46.270.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_const_view_tlate< data_t, vparent_t, block_t >

Definition at line 109 of file ovector_tlate.h.

Data Structures

- class const_iterator
  A const iterator for ovectors.
- class iterator
  An iterator for ovectors.

Public Member Functions

- const_iterator begin () const
  An iterator for the beginning of the vector.
- const_iterator end () const
  An iterator for the end of the vector.

Copy constructors

- ovector_const_view_tlate (const ovector_const_view_tlate &v)
  Shallow copy constructor - create a new view of the same vector.
- ovector_const_view_tlate & operator= (const ovector_const_view_tlate &v)
  Shallow copy constructor - create a new view of the same vector.
- ovector_const_view_tlate (const uvector_const_view_tlate< data_t > &v)
  Shallow copy constructor - view a unit-stride vector.
- ovector_const_view_tlate & operator= (const uvector_const_view_tlate< data_t > &v)
  Shallow copy constructor - view a unit-stride vector.

Get methods

- const data_t & operator[ ] (size_t i) const
  Array-like indexing (with optional range-checking)
- const data_t & operator() (size_t i) const
  Array-like indexing (with optional range-checking)
- data_t get (size_t i) const
  Get (with optional range-checking)
- const data_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)
- size_t size () const
  Method to return vector size.
- size_t capacity () const
  Method to return capacity.
- size_t stride () const
  Method to return vector stride.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.
- size_t lookup (const data_t x0) const
  Exhaustively look through the vector for a particular value and return the closest match.
- data_t max () const
  Find the maximum element.
- size_t max_index () const
  Find the location of the maximum element.
- data_t min () const
  Find the minimum element.
- size_t min_index () const
  Find the location of the minimum element.

Protected Member Functions

- ovector_const_view_tlate ()
  Empty constructor provided for use by ovector_view_tlate(const ovector_view_tlate &v)

46.270.2 Member Function Documentation

46.270.2.1 template<class data_t, class vparent_t, class block_t> size_t ovector_const_view_tlate< data_t, vparent_t, block_t >::size ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 383 of file ovector_tlate.h.

46.270.2.2 template<class data_t, class vparent_t, class block_t> size_t ovector_const_view_tlate< data_t, vparent_t, block_t >::capacity ( ) const [inline]

Analogous to `std::vector<>.capacity()`.

Definition at line 391 of file ovector_tlate.h.

46.270.2.3 template<class data_t, class vparent_t, class block_t> size_t ovector_const_view_tlate< data_t, vparent_t, block_t >::stride ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 401 of file ovector_tlate.h.

46.270.2.4 template<class data_t, class vparent_t, class block_t> bool ovector_const_view_tlate< data_t, vparent_t, block_t >::is_owner ( ) const [inline]

This can be used to determine if an object is a "vector_view", or a "vector". If is_owner() is true, then it is an ovector_tlate object.

If any O_2scl class creates a ovector_tlate object in which is_owner() returns false, then it is a bug and should be reported.

Definition at line 418 of file ovector_tlate.h.

46.270.2.5 template<class data_t, class vparent_t, class block_t> size_t ovector_const_view_tlate< data_t, vparent_t, block_t >::lookup (const data_t *x0*) const [inline]

This can only fail if the vector is empty or if *all* of the entries in the vector are not finite. In these cases the function calls the error handler and returns 0.

If more than one entry is the same distance from `x0`, this function returns the entry with smallest index.

Definition at line 433 of file ovector_tlate.h.

46.270.2.6 template<class data_t, class vparent_t, class block_t> data_t ovector_const_view_tlate< data_t, vparent_t, block_t >::max ( ) const [inline]

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 464 of file ovector_tlate.h.

46.270.2.7 template<class data_t, class vparent_t, class block_t> size_t ovector_const_view_tlate< data_t, vparent_t, block_t >::max_index ( ) const [inline]

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 498 of file ovector_tlate.h.

46.270.2.8 template<class data_t, class vparent_t, class block_t> data_t ovector_const_view_tlate< data_t, vparent_t, block_t >::min ( ) const [inline]

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 544 of file ovector_tlate.h.

46.270.2.9 template<class data_t, class vparent_t, class block_t> size_t ovector_const_view_tlate< data_t, vparent_t, block_t >::min_index ( ) const [inline]

This can only fail if *all* of the entries in the array are not finite or if the vector is empty, in which case it calls the error handler and returns 0.

Definition at line 578 of file ovector_tlate.h.

The documentation for this class was generated from the following file:

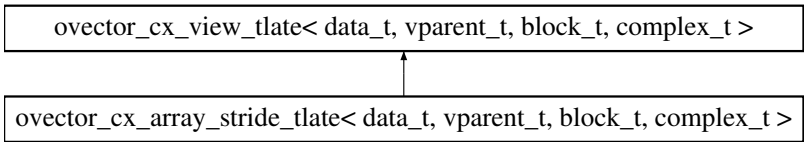
- ovector_tlate.h

46.271 ovector_cx_array_stride_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from an array with a stride.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_array_stride_tlate< data_t, vparent_t, block_t, complex_t >:



46.271.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_array_stride_tlate< data_t, vparent_t, block_t, complex_t >

Definition at line 740 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_array_stride_tlate (size_t n, complex_t *dat, size_t s)
  Create a vector from `dat` with size `n` and stride `s`.

The documentation for this class was generated from the following file:

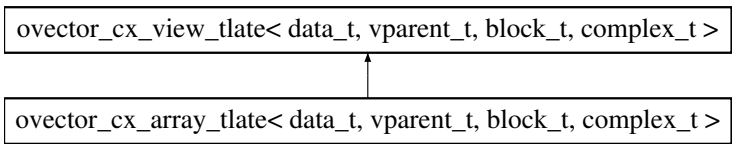
- ovector_cx_tlate.h

46.272 ovector_cx_array_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from an array.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_array_tlate< data_t, vparent_t, block_t, complex_t >:



46.272.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_array_tlate< data_t, vparent_t, block_t, complex_t >

Definition at line 720 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_array_tlate (size_t n, complex_t *dat)
  Create a vector from `dat` *with size* `n`.

The documentation for this class was generated from the following file:

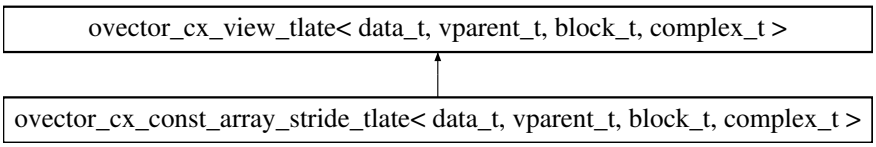
- ovector_cx_tlate.h

46.273 ovector_cx_const_array_stride_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from an array_stride.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_const_array_stride_tlate< data_t, vparent_t, block_t, complex_t >:



46.273.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_const_array_stride_tlate< data_t, vparent_t, block_t, complex_t >

Definition at line 842 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_const_array_stride_tlate (size_t n, const complex_t *dat, size_t s)
  Create a vector from `dat` *with size* `n`.

Protected Member Functions

These are inaccessible to ensure the vector is \c const.

- data_t & operator[ ] (size_t i)
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing.
- data_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- int **set** (size_t i, data_t val)
- int **set_all** (double val)
- vparent_t * **get_gsl_vector** ()
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator+= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator+=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator-= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator-=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator*= (const data_t &y)

operator∗=

The documentation for this class was generated from the following file:

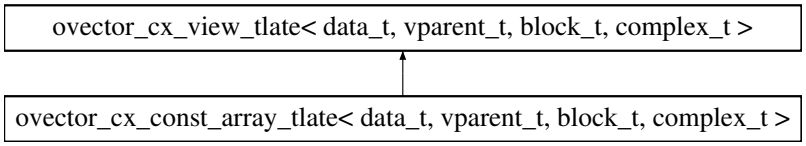
- ovector_cx_tlate.h

46.274 ovector_cx_const_array_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from an array.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_const_array_tlate< data_t, vparent_t, block_t, complex_t >:



46.274.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_const_array_tlate< data_t, vparent_t, block_t, complex_t >

Definition at line 790 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_const_array_tlate (size_t n, const complex_t ∗dat)
  Create a vector from `dat` *with size* `n`.

Protected Member Functions

These are inaccessible to ensure the vector is \c const.

- data_t & operator[ ] (size_t i)
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing.
- data_t ∗ get_ptr (size_t i)
  Get pointer (with optional range-checking)
- int **set** (size_t i, data_t val)
- int **set_all** (double val)
- vparent_t ∗ **get_gsl_vector** ()
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator+= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator+=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator-= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator-=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator∗= (const data_t &y)
  operator∗=

The documentation for this class was generated from the following file:

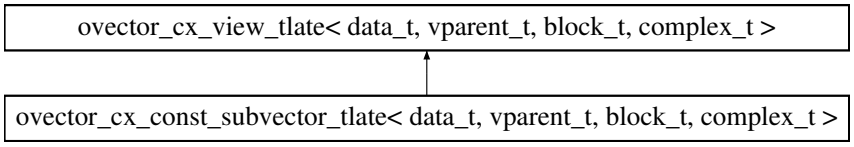
- ovector_cx_tlate.h

46.275 ovector_cx_const_subvector_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from a subvector of another.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_const_subvector_tlate< data_t, vparent_t, block_t, complex_t >:



46.275.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_const_subvector_tlate< data_t, vparent_t, block_t, complex_t >

Definition at line 895 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_const_subvector_tlate (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

Protected Member Functions

These are inaccessible to ensure the vector is \c const.

- data_t & operator[ ] (size_t i)
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing.
- data_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- int **set** (size_t i, data_t val)
- int **set_all** (double val)
- vparent_t * **get_gsl_vector** ()
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator+= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator+=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator-= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator-=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator*= (const data_t &y)
  operator=*

The documentation for this class was generated from the following file:

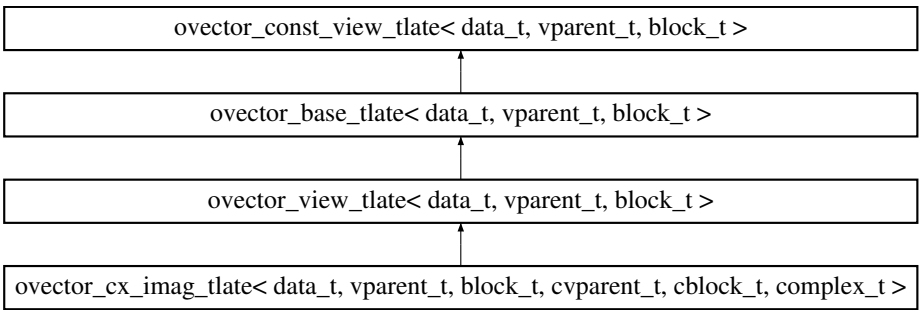
- ovector_cx_tlate.h

46.276 ovector_cx_imag_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t > Class Template Reference

Create a imaginary vector from the imaginary parts of a complex vector.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_imag_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t >:



46.276.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class cvparent_t, class cblock_t, class complex_t>class ovector_cx_imag_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t >

Definition at line 974 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_imag_tlate (ovector_cx_view_tlate< data_t, cvparent_t, cblock_t, complex_t > &x)
  Create a imaginary vector from the imaginary parts of a complex vector.

The documentation for this class was generated from the following file:

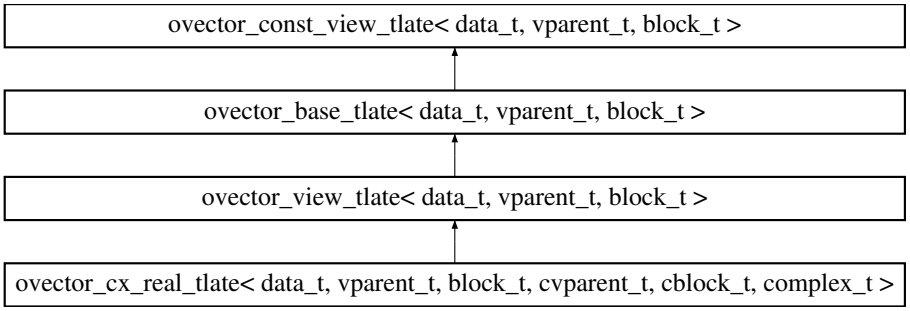
- ovector_cx_tlate.h

46.277 ovector_cx_real_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t > Class Template Reference

Create a real vector from the real parts of a complex vector.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_real_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t >:



46.277.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class cvparent_t, class cblock_t, class complex_t>class ovector_cx_real_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t >

Definition at line 952 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_real_tlate (ovector_cx_view_tlate< data_t, cvparent_t, cblock_t, complex_t > &x)
  Create a real vector from the real parts of a complex vector.

The documentation for this class was generated from the following file:

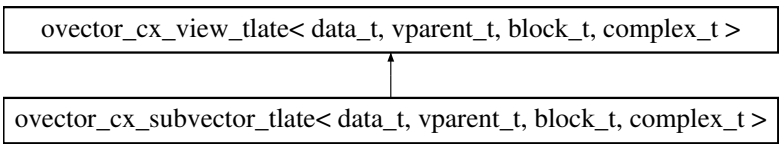
- ovector_cx_tlate.h

46.278 ovector_cx_subvector_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

Create a vector from a subvector of another.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_subvector_tlate< data_t, vparent_t, block_t, complex_t >:



46.278.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_subvector_tlate< data_t, vparent_t, block_t, complex_t >

Definition at line 761 of file ovector_cx_tlate.h.

Public Member Functions

- ovector_cx_subvector_tlate (ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

The documentation for this class was generated from the following file:

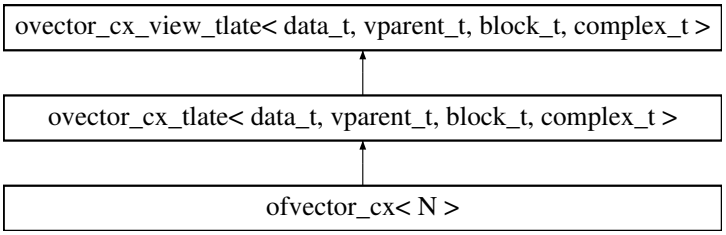
- ovector_cx_tlate.h

46.279 ovector_cx_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

A vector of double-precision numbers.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_tlate< data_t, vparent_t, block_t, complex_t >:



46.279.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_tlate< data_t, vparent_t, block_t, complex_t >

If the memory allocation fails, either in the constructor or in allocate(), then the error handler will be called, partially allocated memory will be freed, and the size will be reset to zero. You can test to see if the allocation succeeded using something like

```
const size_t n=10;
ovector_cx x(10);
if (x.size()==0) cout << "Failed." << endl;
```

Todo Add subvector_stride, const_subvector_stride

Definition at line 479 of file ovector_cx_tlate.h.

Public Member Functions

Standard constructor

- ovector_cx_tlate (size_t n=0)
  Create an ovector_cx of size n with owner as 'true'.

Copy constructors

- ovector_cx_tlate (const ovector_cx_tlate &v)
  Deep copy constructor, allocate new space and make a copy.
- ovector_cx_tlate (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &v)
  Deep copy constructor, allocate new space and make a copy.
- ovector_cx_tlate & operator= (const ovector_cx_tlate &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- ovector_cx_tlate & operator= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.

Memory allocation

- int allocate (size_t nsize)
  Allocate memory for size n after freeing any memory presently in use.
- int free ()
  Free the memory.

Other methods

- vparent_t * get_gsl_vector_complex ()
  Return a gsl vector_cx.
- const vparent_t * get_gsl_vector_complex_const () const
  Return a gsl vector_cx.

46.279.2 Member Function Documentation

46.279.2.1 template<class data_t, class vparent_t, class block_t, class complex_t> int ovector_cx_tlate< data_t, vparent_t, block_t, complex_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 690 of file ovector_cx_tlate.h.

The documentation for this class was generated from the following file:

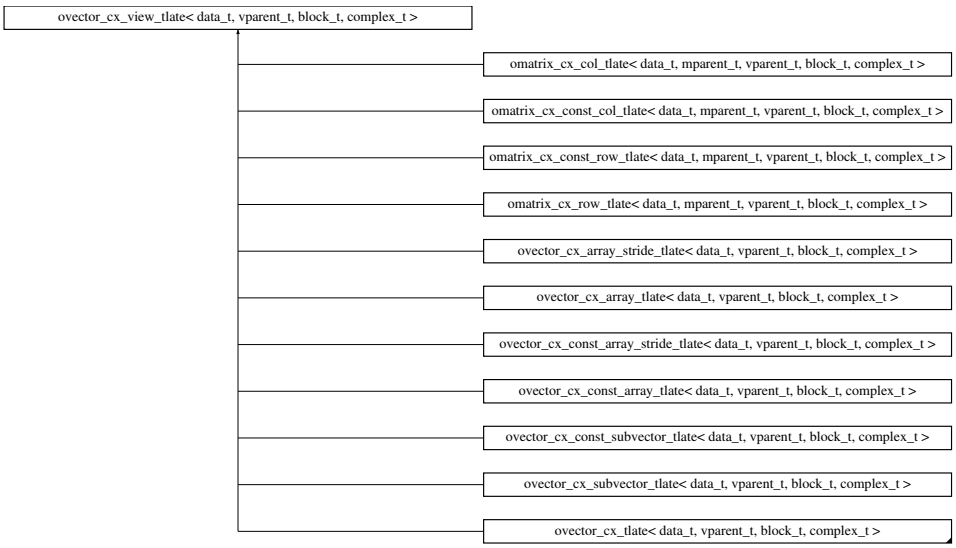
- ovector_cx_tlate.h

46.280 ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > Class Template Reference

A vector view of double-precision numbers.

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t >:



46.280.1 Detailed Description

template<class data_t, class vparent_t, class block_t, class complex_t>class ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t >

Definition at line 51 of file ovector_cx_tlate.h.

Public Member Functions

- int conjugate ()
  Conjugate the vector.
- data_t norm () const
  Complex norm $v^{\dagger}v$.

Copy constructors

- ovector_cx_view_tlate (const ovector_cx_view_tlate &v)

Shallow copy constructor - create a new view of the same vector.

- ovector_cx_view_tlate & operator= (const ovector_cx_view_tlate &v)
  Shallow copy constructor - create a new view of the same vector.

Get and set methods

- complex_t & operator[ ] (size_t i)
  Array-like indexing.
- const complex_t & operator[ ] (size_t i) const
  Array-like indexing.
- complex_t & operator() (size_t i)
  Array-like indexing.
- const complex_t & operator() (size_t i) const
  Array-like indexing.
- complex_t get (size_t i) const
  Get (with optional range-checking)
- data_t real (size_t i) const
  Get real part (with optional range-checking)
- data_t imag (size_t i) const
  Get imaginary part (with optional range-checking)
- std::complex< data_t > get_stl (size_t i) const
  Get STL-like complex number (with optional range-checking)
- complex_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- const complex_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)
- int set (size_t i, const complex_t &val)
  Set (with optional range-checking)
- int set_stl (size_t i, const std::complex< data_t > &d)
  Set (with optional range-checking)
- int set (size_t i, data_t vr, data_t vi)
  Set (with optional range-checking)
- int set_all (const complex_t &g)
  Set all of the value to be the value `val`.
- size_t size () const
  Method to return vector size.
- size_t stride () const
  Method to return vector stride.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.

Arithmetic

- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator+= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator+=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator-= (const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
  operator-=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator+= (const complex_t &x)
  operator+=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator-= (const complex_t &x)
  operator-=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator*= (const complex_t &x)
  operator=*
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator+= (const data_t &x)
  operator+=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator-= (const data_t &x)
  operator-=
- ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & operator*= (const data_t &x)
  operator=*

Protected Member Functions

- ovector_cx_view_tlate ()
  Empty constructor provided for use by ovector_cx_tlate(const ovector_cx_tlate &v) [protected].

46.280.2 Member Function Documentation

46.280.2.1 template<class data_t, class vparent_t, class block_t, class complex_t> size_t ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t >::size ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 303 of file ovector_cx_tlate.h.

46.280.2.2 template<class data_t, class vparent_t, class block_t, class complex_t> size_t ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t >::stride ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 312 of file ovector_cx_tlate.h.

The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

46.281 ovector_int_alloc Class Reference

A simple class to provide an `allocate()` function for ovector_int.

`#include <ovector_tlate.h>`

46.281.1 Detailed Description

Definition at line 1873 of file ovector_tlate.h.

Public Member Functions

- void allocate (ovector_int &o, size_t i)
  Allocate `v` for `i` elements.
- void free (ovector_int &o)
  Free memory.

The documentation for this class was generated from the following file:

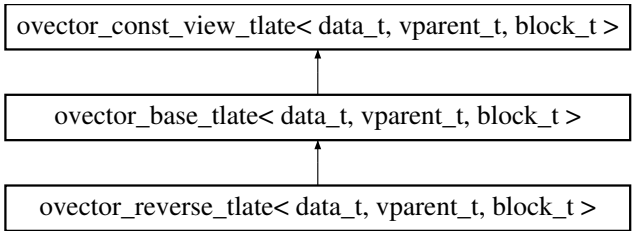
- ovector_tlate.h

46.282 ovector_reverse_tlate< data_t, vparent_t, block_t > Class Template Reference

Reversed view of a vector.

`#include <ovector_rev_tlate.h>`

Inheritance diagram for ovector_reverse_tlate< data_t, vparent_t, block_t >:



46.282.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_reverse_tlate< data_t, vparent_t, block_t >

Note

Note that you can't reverse a reversed vector, and this is why this class does not have a constructor of the form `ovector_reverse(ovector_reverse &v)`.

Definition at line 55 of file ovector_rev_tlate.h.

Public Member Functions

- ovector_reverse_tlate (ovector_tlate< data_t, vparent_t, block_t > &v)
  Create a vector from `dat` with size `n` and stride `s`.
- ovector_reverse_tlate (ovector_view_tlate< data_t, vparent_t, block_t > &v)
  Create a vector from `dat` with size `n` and stride `s`.

Get and set methods

- data_t & operator[ ] (size_t i)
  Array-like indexing.
- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.
- data_t get (size_t i) const
  Get (with optional range-checking)
- data_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- const data_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)
- int set (size_t i, data_t val)
  Set (with optional range-checking)
- int set_all (double val)
  Set all of the value to be the value `val`.

The documentation for this class was generated from the following file:

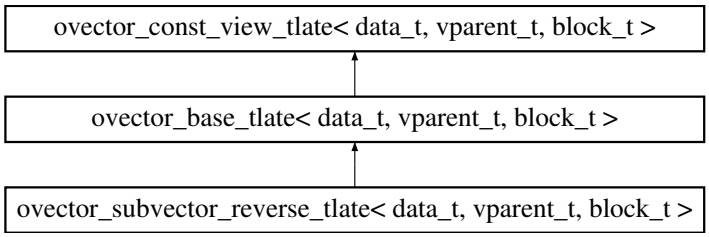
- ovector_rev_tlate.h

46.283 ovector_subvector_reverse_tlate< data_t, vparent_t, block_t > Class Template Reference

Reversed view of a subvector.

`#include <ovector_rev_tlate.h>`

Inheritance diagram for ovector_subvector_reverse_tlate< data_t, vparent_t, block_t >:



46.283.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_subvector_reverse_tlate< data_t, vparent_t, block_t >

Warning

At present, reversing a reversed vector does not give the original ordering.

Definition at line 317 of file ovector_rev_tlate.h.

Public Member Functions

- ovector_subvector_reverse_tlate (ovector_base_tlate< data_t, vparent_t, block_t > &v, size_t offset, size_t n)
  Create a vector from `dat` with size `n` and stride `s`.

Get and set methods

- data_t & operator[ ] (size_t i)
  Array-like indexing.
- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.
- data_t get (size_t i) const
  Get (with optional range-checking)
- data_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- const data_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)
- int set (size_t i, data_t val)
  Set (with optional range-checking)
- int set_all (double val)
  Set all of the value to be the value `val`.

The documentation for this class was generated from the following file:

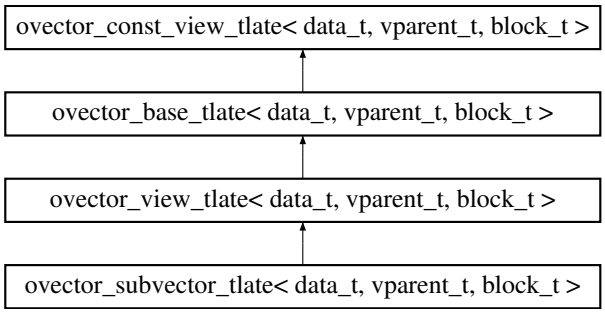
- ovector_rev_tlate.h

46.284 ovector_subvector_tlate< data_t, vparent_t, block_t > Class Template Reference

Create a vector from a subvector of another.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_subvector_tlate< data_t, vparent_t, block_t >:



46.284.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_subvector_tlate< data_t, vparent_t, block_t >

The constructor will fail if the original vector is empty, or if the user requests a subvector which includes elements beyond the end of the original vector.

Definition at line 1645 of file ovector_tlate.h.

Public Member Functions

- ovector_subvector_tlate (ovector_base_tlate< data_t, vparent_t, block_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

The documentation for this class was generated from the following file:

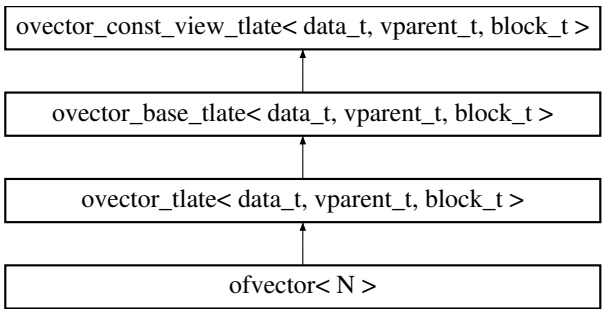
- ovector_tlate.h

46.285 ovector_tlate< data_t, vparent_t, block_t > Class Template Reference

A vector with finite stride.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_tlate< data_t, vparent_t, block_t >:



46.285.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_tlate< data_t, vparent_t, block_t >

There are several global binary operators associated with objects of type uvector_tlate. The are documented in the "Functions" section of ovector_tlate.h.

Design notes

At present, the owner variable can be used to distinguish between ovectors and ovector_views: for views, owner is always zero, but for normal ovectors, owner is 1 (even for empty vectors).

The reserve(), pop_back(), and push_back() methods require the ability to have empty vectors which still have memory allocated for them, so the proper test for whether or not memory is allocated is to test whether or not 'block' is zero. This means we should never have a case where the block is non-zero (i.e. there is memory allocated there) but the size of the block is zero. This is the test used in the destructor and the free() method. The free method also sets block to zero accordingly.

Definition at line 1101 of file ovector_tlate.h.

Public Member Functions

Standard constructor

- ovector_tlate (size_t n=0)
  Create an ovector of size `n` with owner as 'true'.

Copy constructors

- ovector_tlate (const ovector_tlate &v)
  Deep copy constructor.
- template<class alt_vec_t >
  ovector_tlate (size_t nv, alt_vec_t &v)
  Deep copy constructor for generic vectors.
- ovector_tlate (const ovector_const_view_tlate< data_t, vparent_t, block_t > &v)
  Deep copy constructor for other related vectors.
- ovector_tlate & operator= (const ovector_tlate &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- ovector_tlate & operator= (const ovector_const_view_tlate< data_t, vparent_t, block_t > &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- ovector_tlate & operator= (const uvector_const_view_tlate< data_t > &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.

Memory allocation

- int allocate (size_t nsize)
  Allocate memory for size `n` after freeing any memory currently in use.
- int free ()
  Free the memory.

Stack-like operations

- int push_back (data_t val)
  Add a value to the end of the vector.
- int reserve (size_t cap)
  Reserve memory by increasing capacity.
- data_t pop_back ()
  Return the last value and shrink the vector size by one.

Other methods

- int erase (size_t ix)
  Remove element with index `ix` and decrease the vector size by one.
- int sort_unique ()
  Sort the vector and ensure all elements are unique by removing duplicates.

Protected Member Functions

- void intl_sanity_check (size_t ix)
  An internal sanity check to ensure correctness.
- int intl_allocate (size_t nsize)
  An internal allocate function.
- int intl_free ()
  The internal free function.
- template<class alt_vec_t >
  int intl_init (size_t n, alt_vec_t &v)
  An internal init() function.
- template<class alt_vec_t >
  ovector_tlate & intl_assign (size_t n, alt_vec_t &v)
  An internal assignment function for `operator=()`

46.285.2 Constructor & Destructor Documentation

46.285.2.1 template<class data_t, class vparent_t, class block_t> ovector_tlate< data_t, vparent_t, block_t >::ovector_tlate (const ovector_tlate< data_t, vparent_t, block_t > & *v*) [inline]

Definition at line 1257 of file ovector_tlate.h.

46.285.3 Member Function Documentation

46.285.3.1 template<class data_t, class vparent_t, class block_t> int ovector_tlate< data_t, vparent_t, block_t >::intl_allocate (size_t *nsize*) [inline, protected]

Note

This function does nothing if `nsize` is zero. Also, this does not free any already allocated memory first (unlike the user-interface version allocate()).

Definition at line 1129 of file ovector_tlate.h.

46.285.3.2 template<class data_t, class vparent_t, class block_t> template<class alt_vec_t > int ovector_tlate< data_t, vparent_t, block_t >::intl_init (size_t *n*, alt_vec_t & *v*) [inline, protected]

This function calls intl_allocate() first, and then copies the data from the vector `v`.

Definition at line 1194 of file ovector_tlate.h.

46.285.3.3 template<class data_t, class vparent_t, class block_t> int ovector_tlate< data_t, vparent_t, block_t >::allocate (size_t *nsize*) [inline]

Note that this automatically deallocates any previously allocated memory before attempting to allocate more. If this allocation fails (i.e. because we ran out of memory) then the original vector will still have been deallocated.

Definition at line 1360 of file ovector_tlate.h.

46.285.3.4 template<class data_t, class vparent_t, class block_t> int ovector_tlate< data_t, vparent_t, block_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 1373 of file ovector_tlate.h.

46.285.3.5 template<class data_t, class vparent_t, class block_t> int ovector_tlate< data_t, vparent_t, block_t >::reserve (size_t *cap*) [inline]

Increase the maximum capacity of the vector so that calls to push_back() do not need to automatically increase the capacity.

If the argument `cap` is smaller than the present vector size given by size(), then this function does nothing and the error handler is not called.

Definition at line 1445 of file ovector_tlate.h.

The documentation for this class was generated from the following file:

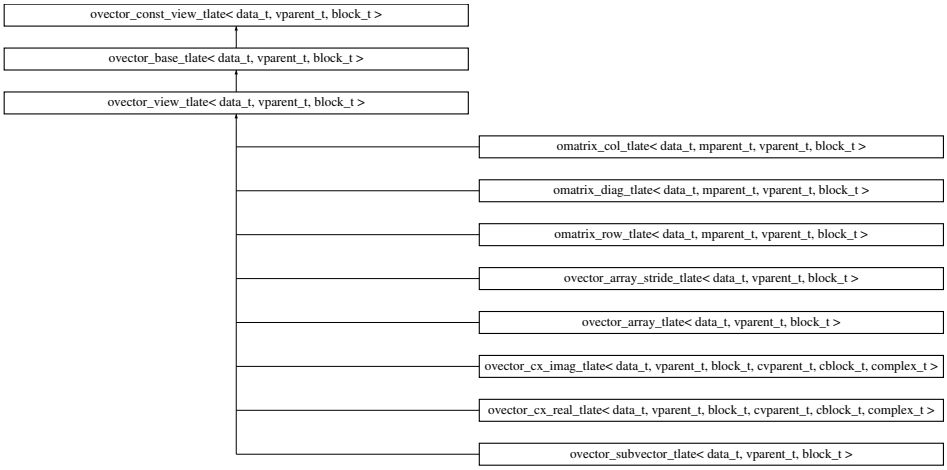
- ovector_tlate.h

46.286 ovector_view_tlate< data_t, vparent_t, block_t > Class Template Reference

A vector view with finite stride.

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_view_tlate< data_t, vparent_t, block_t >:



46.286.1 Detailed Description

template<class data_t, class vparent_t, class block_t>class ovector_view_tlate< data_t, vparent_t, block_t >

Definition at line 905 of file ovector_tlate.h.

Public Member Functions

Copy constructors

- ovector_view_tlate (const ovector_view_tlate &v)
  Shallow copy constructor - create a new view of the same vector.
- ovector_view_tlate & operator= (const ovector_view_tlate &v)
  Shallow copy constructor - create a new view of the same vector.
- ovector_view_tlate (ovector_base_tlate< data_t, vparent_t, block_t > &v)
  Shallow copy constructor for non-views.
- ovector_view_tlate & operator= (ovector_base_tlate< data_t, vparent_t, block_t > &v)
  Shallow copy constructor for non-views.
- ovector_view_tlate (uvector_base_tlate< data_t > &v)
  Shallow copy constructor for unit-stride vectors.
- ovector_view_tlate & operator= (uvector_base_tlate< data_t > &v)
  Shallow copy constructor for unit-stride vectors.

Get and set methods

- data_t & operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i) const
  Array-like indexing with operator()
- data_t * get_ptr (size_t i) const
  Get pointer (with optional range-checking)
- int set (size_t i, data_t val) const
  Set (with optional range-checking)
- int set_all (double val) const
  Set all of the value to be the value `val`.

Protected Member Functions

- ovector_view_tlate ()
  Empty constructor provided for use by ovector_view_tlate(const ovector_view_tlate &v)

46.286.2 Member Function Documentation

46.286.2.1 template<class data_t, class vparent_t, class block_t> int ovector_view_tlate< data_t, vparent_t, block_t >::set_all (double *val*) const [inline]

If the vector is empty, this function does not perform any assignment and does not call the error handler.

Definition at line 1056 of file ovector_tlate.h.

The documentation for this class was generated from the following file:

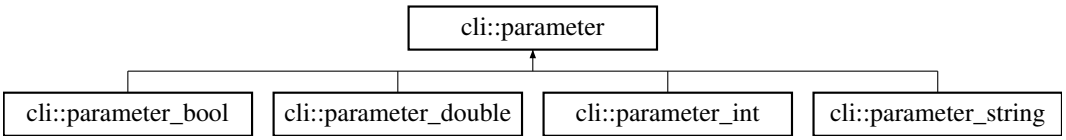
- ovector_tlate.h

46.287 cli::parameter Class Reference

Parameter for cli.

`#include <cli.h>`

Inheritance diagram for cli::parameter:



46.287.1 Detailed Description

Definition at line 235 of file cli.h.

Public Member Functions

- virtual int set (std::string s)=0
  Set from string.
- virtual std::string get ()=0
  Convert to string.

Data Fields

- std::string help
  Help description.

The documentation for this class was generated from the following file:

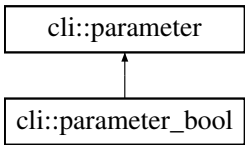
- cli.h

46.288 cli::parameter_bool Class Reference

String parameter for cli.

`#include <cli.h>`

Inheritance diagram for cli::parameter_bool:



46.288.1 Detailed Description

Definition at line 276 of file cli.h.

Public Member Functions

- virtual int set (std::string s)
  Set from string.
- virtual std::string get ()
  Convert to string.

Data Fields

- bool * b
  Parameter.

The documentation for this class was generated from the following file:

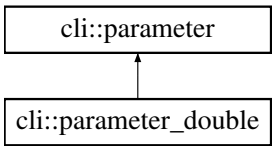
- cli.h

46.289 cli::parameter_double Class Reference

Double parameter for cli.

`#include <cli.h>`

Inheritance diagram for cli::parameter_double:



46.289.1 Detailed Description

Definition at line 299 of file cli.h.

Public Member Functions

- virtual int set (std::string s)
  Set from string.
- virtual std::string get ()
  Convert to string.

Data Fields

- double * d
  Parameter.

The documentation for this class was generated from the following file:

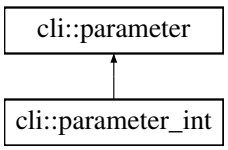
- cli.h

46.290 cli::parameter_int Class Reference

Integer parameter for cli.

`#include <cli.h>`

Inheritance diagram for cli::parameter_int:



46.290.1 Detailed Description

Definition at line 322 of file cli.h.

Public Member Functions

- virtual int set (std::string s)
  Set from string.
- virtual std::string get ()
  Convert to string.

Data Fields

- int * i
  Parameter.

The documentation for this class was generated from the following file:

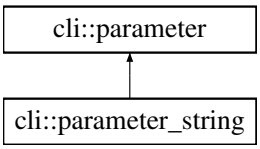
- cli.h

46.291 cli::parameter_string Class Reference

String parameter for cli.

`#include <cli.h>`

Inheritance diagram for cli::parameter_string:



46.291.1 Detailed Description

Definition at line 253 of file cli.h.

Public Member Functions

- virtual int set (std::string s)
  Set from string.
- virtual std::string get ()
  Convert to string.

Data Fields

- std::string * str
  Parameter.

The documentation for this class was generated from the following file:

- cli.h

46.292 permutation Class Reference

A class for representing permutations.

`#include <permutation.h>`

46.292.1 Detailed Description

A class for representing permutations. This permutation class is completely compatible with the GSL permutation object since it is derived from `gsl_permutation_struct` (and thus also `gsl_permutation`). For example, the last line in the code below is a trivial upcast, and permitted since `gsl_permutation` is a base type of this class.

```
permuation p(5);
p.init();
gsl_permutation *gp=&p;
```

See also the Permutations section of the User's guide.

Definition at line 84 of file permutation.h.

Public Member Functions

- permutation (size_t dim=0)
  Create a permutation of size `dim`.
- size_t & operator[ ] (size_t i)
  Array-like indexing.
- const size_t & operator[ ] (size_t i) const
  Array-like indexing.
- size_t & operator() (size_t i)
  Array-like indexing.
- const size_t & operator() (size_t i) const
  Array-like indexing.
- size_t get (size_t i) const
  Get (with optional range-checking)
- int set (size_t i, size_t val)
  Set (with optional range-checking)
- int init ()
  Initialize permutation to the identity.
- size_t size () const
  Return permutation size.
- int allocate (size_t dim)
  Allocate memory for a permutation of size `dim`.
- int free ()
  Free the memory.
- int swap (const size_t i, const size_t j)
  Swap two elements of a permutation.
- bool valid () const
  Check to see that a permutation is valid.
- int reverse ()
  Reverse the permutation.
- permutation inverse () const
  Compute the inverse of a permutation.
- template<class vec_t >
  int apply (vec_t &v) const
  Apply the permutation to a vector.
- template<class vec_t >
  int apply_inverse (vec_t &v) const
  Apply the inverse permutation to a vector.

Copy constructors

- **permutation** (const permutation &v)
- permutation & **operator=** (const permutation &v)

46.292.2 Member Function Documentation

46.292.2.1 size_t permutation::size ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 234 of file permutation.h.

46.292.2.2 int permutation::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 255 of file permutation.h.

The documentation for this class was generated from the following file:

- permutation.h

46.293 pinside Class Reference

Test line intersection and point inside polygon.

`#include <pinside.h>`

46.293.1 Detailed Description

This is a fast and dirty implementation of the point inside polygon test from Jerome L. Lewis, SIGSCE Bulletin, 34 (2002) 81.

Note that an error in that article ("count-" should have been "count--") has been corrected here.

Idea for Future The inside() functions actually copy the points twice. This can be made more efficient.

Definition at line 66 of file pinside.h.

Data Structures

- struct line
  Internal line definition for pinside.
- struct point
  Internal point definition for pinside.

Public Member Functions

- int intersect (double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)
  Determine if two line segments intersect.
- int inside (double x, double y, const o2scl::ovector_base &xa, const o2scl::ovector_base &ya)
  Determine if point (x,y) is inside a polygon.
- template<class vec_t >
  int inside (double x, double y, size_t n, const vec_t &xa, const vec_t &ya)
  Determine if point (x,y) is inside a polygon.
- int test (test_mgr &t)
  Perform some simple testing.

Protected Member Functions

- int intersect (line P, line Q)
  Test if line segments `P` and `Q` intersect.
- int inside (point t, point p[ ], int N)
  Test if point `t` is inside polygon `p` of size `N`.

46.293.2 Member Function Documentation

46.293.2.1 int pinside::intersect (double *x1*, double *y1*, double *x2*, double *y2*, double *x3*, double *y3*, double *x4*, double *y4*) [inline]

The function returns 1 if the line segment determined by the endpoints (x_1, y_1) and (x_2, y_2) and the line segment determined by the endpoints (x_3, y_3) and (x_4, y_4) intersect, and 0 otherwise.

Definition at line 98 of file pinside.h.

46.293.2.2 int pinside::inside (double *x*, double *y*, const o2scl::ovector_base & *xa*, const o2scl::ovector_base & *ya*)

This returns 1 if the point (x,y) is inside the polygon defined by `xa` and `ya`, and 0 otherwise.

Note that if the point (x,y) is exactly on the polygon, then the result of this function is not well-defined and it will return either 0 or 1.

46.293.2.3 template<class vec_t > int pinside::inside (double *x*, double *y*, size_t *n*, const vec_t & *xa*, const vec_t & *ya*) [inline]

This returns 1 if the point (x,y) is inside the polygon defined by `xa` and `ya`, and 0 otherwise.

The parameter `n` should be the number of polygon points specified in vectors `xa` and `ya`.

Note that if the point (x,y) is exactly on the polygon, then the result of this function is not well-defined and it will return either 0 or 1.

Definition at line 130 of file pinside.h.

The documentation for this class was generated from the following file:

- pinside.h

46.294 planar_intp< vec_t, mat_t > Class Template Reference

Interpolate among two independent variables with planes.

```
#include <planar_intp.h>
```

46.294.1 Detailed Description

template<class vec_t, class mat_t>class planar_intp< vec_t, mat_t >

This class is experimental.

This is an analog of 1-d linear interpolation for two dimensions, which is particularly useful with the data points are not arranged in a specified order (i.e. on a grid). For a set of data $x_i, y_i, f_{j,i}$, the values of f_j are predicted given a value of x and y. In contrast to twod_intp, the data need not be presented in a grid. This interpolation is performed by finding the plane that goes through three closest points in the data set. Distances are determined with

$$d_{ij} = \sqrt{\left(\frac{x_i - x_j}{\Delta x}\right)^2 + \left(\frac{y_i - y_j}{\Delta y}\right)^2}$$

where $\Delta x = x_{\max} - x_{\min}$ and $\Delta y = y_{\max} - y_{\min}$.

This procedure will fail if the three closest points are co-linear, and interp() will then call the error handler.

There is no caching so the numeric values of the data may be freely changed between calls to interp().

The vector and matrix types can be any types which have suitably defined functions `operator`[].

For example, with 10 random points in the x-y plane with $-1 < x < 1$ and $-1 < y < 1$, the planes are demarcated according to

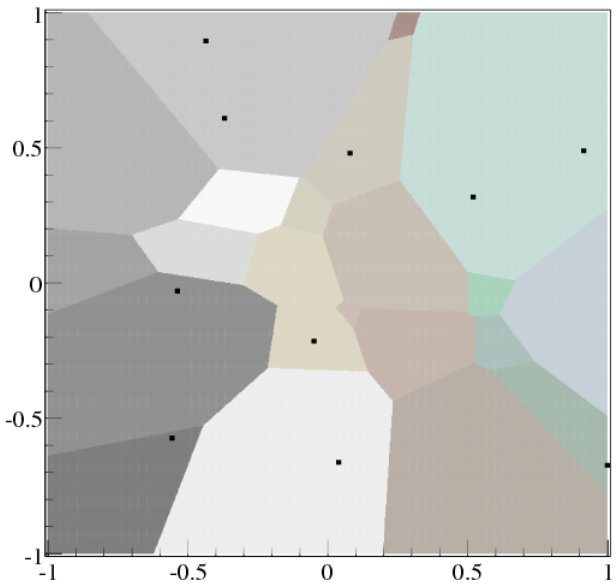


Figure 12: in_planar_intp.pdf

where each polygonal region represents the set of all points in the domain which will be mapped to the same plane to approximate the function.

Idea for Future Rewrite so that it never fails unless all the points in the data set lie on a line. This would probably demand sorting all of the points by distance from desired location. (11/1/11 - Maybe not. We might be able to get away with just storing an vector which records the index j of the point in the data set which is the nearest neighbor for each point i in the data set.)

Idea for Future Instead of comparing `den` with zero, add the option of comparing it with a small number.

Idea for Future Allow the user to specify the scales used for x and y, instead of computing max-min every time.

Idea for Future The generalization to more dimensions might be straight forward.

Definition at line 90 of file planar_intp.h.

Public Member Functions

- int set_data (size_t n_points, vec_t &x, vec_t &y, size_t n_dat, mat_t &dat)
  Initialize the data for the planar interpolation.
- int interp (double x, double y, vec_t &ip)
  Perform the planar interpolation.
- int interp (double x, double y, vec_t &ip, size_t &i1, double &x1, double &y1, size_t &i2, double &x2, double &y2, size_t &i3, double &x3, double &y3)
  Planar interpolation returning the closest points.

Protected Member Functions

- int swap (size_t &i1, double &c1, size_t &i2, double &c2)
  Swap points 1 and 2.

Protected Attributes

- size_t np
  The number of points.
- size_t nd
  The number of functions.
- vec_t * ux
  The x-values.
- vec_t * uy
  The y-values.
- mat_t * udat
  The data.
- bool data_set
  True if the data has been specified.

46.294.2 Member Function Documentation

46.294.2.1 template<class vec_t , class mat_t > int planar_intp< vec_t, mat_t >::interp (double *x*, double *y*, vec_t & *ip*) [inline]

It is assumed that `ip` is properly allocated beforehand.

Definition at line 118 of file planar_intp.h.

46.294.2.2 template<class vec_t , class mat_t > int planar_intp< vec_t, mat_t >::interp (double *x*, double *y*, vec_t & *ip*, size_t & *i1*, double & *x1*, double & *y1*, size_t & *i2*, double & *x2*, double & *y2*, size_t & *i3*, double & *x3*, double & *y3*) [inline]

This function interpolates `x` and `y` into the data returning `ip`. It also returns the three closest x- and y-values used for computing the plane.

It is assumed that `ip` is properly allocated beforehand. Put in initial points

Sort initial points

Definition at line 132 of file planar_intp.h.

The documentation for this class was generated from the following file:

- planar_intp.h

46.295 pinside::point Struct Reference

Internal point definition for pinside.

`#include <pinside.h>`

46.295.1 Detailed Description

Definition at line 71 of file pinside.h.

Data Fields

- double **x**

- double **y**

The documentation for this struct was generated from the following file:

- pinside.h

46.296 o2scl_linalg::pointer_2_mem Class Reference

Allocation object for 2 C-style arrays of equal size.

`#include <tridiag_base.h>`

46.296.1 Detailed Description

This class is used in solve_tridiag_nonsym().

Definition at line 58 of file tridiag_base.h.

Public Member Functions

- void **allocate** (size_t n)
- double * **get_pointer1** ()
- double * **get_pointer2** ()
- void **free** ()

Data Fields

- double * **v1**
- double * **v2**

The documentation for this class was generated from the following file:

- tridiag_base.h

46.297 pointer_2d_alloc< base_t > Class Template Reference

A simple class to provide an `allocate()` function for pointers.

`#include <array.h>`

46.297.1 Detailed Description

template<class base_t>class pointer_2d_alloc< base_t >

This class uses `new` and `delete` and may throw a C++ exception in the usual way. It contains a simple garbage collection mechanism to assist in exception safety.

This class is used in columnify and cern_mroot.

Idea for Future There may be a slight issue here if the pointer allocation succeeds and the list insertion fails, and the user catches the exception thrown by the list insertion failure, then a memory leak will ensue. This might be partially ameliorated by counting allocations and insertions. This failure mechanism is rarely encountered in practice.

Definition at line 200 of file array.h.

Data Structures

- struct pointer_comp

Public Member Functions

- void allocate (base_t **&v, size_t i, size_t j)
  Allocate `v` for `i` elements.
- void free (base_t **&v, size_t i)
  Free memory.

Protected Types

- typedef std::map< base_t **, size_t, pointer_comp > ::iterator iter
  Iterator type.

Protected Attributes

- std::map< base_t **, size_t, pointer_comp > list
  Store allocated pointers.

The documentation for this class was generated from the following file:

- array.h

46.298 o2scl_linalg::pointer_4_mem Class Reference

Allocation object for 4 C-style arrays of equal size.

```
#include <tridiag_base.h>
```

46.298.1 Detailed Description

This class is used in solve_tridiag_sym() and solve_cyc_tridiag_nonsym().

Definition at line 78 of file tridiag_base.h.

Public Member Functions

- void **allocate** (size_t n)
- double * **get_pointer1** ()
- double * **get_pointer2** ()
- double * **get_pointer3** ()
- double * **get_pointer4** ()
- void **free** ()

Data Fields

- double * **v1**
- double * **v2**
- double * **v3**
- double * **v4**

The documentation for this class was generated from the following file:

- tridiag_base.h

46.299 o2scl_linalg::pointer_5_mem Class Reference

Allocation object for 5 C-style arrays of equal size.

`#include <tridiag_base.h>`

46.299.1 Detailed Description

This class is used in solve_cyc_tridiag_sym().

Definition at line 103 of file tridiag_base.h.

Public Member Functions

- void **allocate** (size_t n)
- double * **get_pointer1** ()
- double * **get_pointer2** ()
- double * **get_pointer3** ()
- double * **get_pointer4** ()
- double * **get_pointer5** ()
- void **free** ()

Data Fields

- double * **v1**
- double * **v2**
- double * **v3**
- double * **v4**
- double * **v5**

The documentation for this class was generated from the following file:

- tridiag_base.h

46.300 pointer_alloc< base_t > Class Template Reference

A simple class to provide an `allocate()` function for pointers.

`#include <array.h>`

46.300.1 Detailed Description

template<class base_t>class pointer_alloc< base_t >

This class uses `new` and `delete` and may throw a C++ exception in the usual way. It contains a simple garbage collection mechanism to assist in exception safety.

This class is used in tensor_grid .

Idea for Future There may be a slight issue here if the pointer allocation succeeds and the list insertion fails, and the user catches the exception thrown by the list insertion failure, then a memory leak will ensue. This might be partially ameliorated by counting allocations and insertions. This failure mechanism is rarely encountered in practice.

Definition at line 145 of file array.h.

Public Member Functions

- void allocate (base_t *&v, size_t i)
  Allocate `v` for `i` elements.
- void free (base_t *&v)
  Free memory.

Protected Attributes

- std::vector< base_t * > list
  Store allocated pointers.

The documentation for this class was generated from the following file:

- array.h

46.301 pointer_2d_alloc< base_t >::pointer_comp Struct Reference

46.301.1 Detailed Description

template<class base_t>struct pointer_2d_alloc< base_t >::pointer_comp

Definition at line 204 of file array.h.

Public Member Functions

- bool operator() (base_t **const &p1, base_t **const &p2) const
  Return `p1<p2`.

The documentation for this struct was generated from the following file:

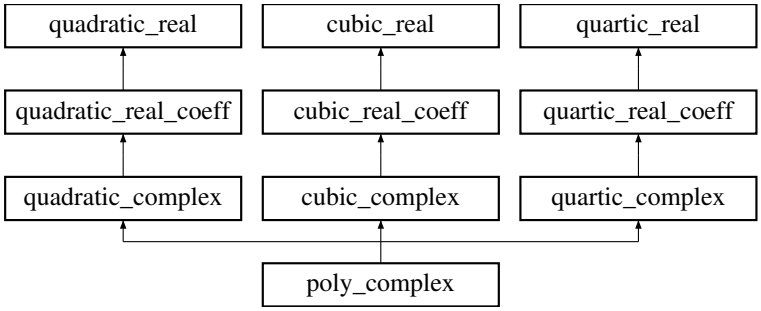
- array.h

46.302 poly_complex Class Reference

Solve a general polynomial with complex coefficients [abstract base].

`#include <poly.h>`

Inheritance diagram for poly_complex:



46.302.1 Detailed Description

Definition at line 349 of file poly.h.

Public Member Functions

- virtual int solve_c (int n, const std::complex< double > co[ ], std::complex< double > ro[ ])=0
  Solve the n-th order polynomial.
- virtual int polish_c (int n, const std::complex< double > co[ ], std::complex< double > *ro)=0
  Polish the roots.
- const char * type ()
  Return a string denoting the type ("poly_complex")

46.302.2 Member Function Documentation

46.302.2.1 virtual int poly_complex::solve_c (int *n*, const std::complex< double > *co[]*, std::complex< double > *ro[]*) [pure virtual]

The coefficients are stored in co[], with the leading coefficient as co[0] and the constant term as co[n]. The roots are returned in ro[0],...,ro[n-1].

The documentation for this class was generated from the following file:

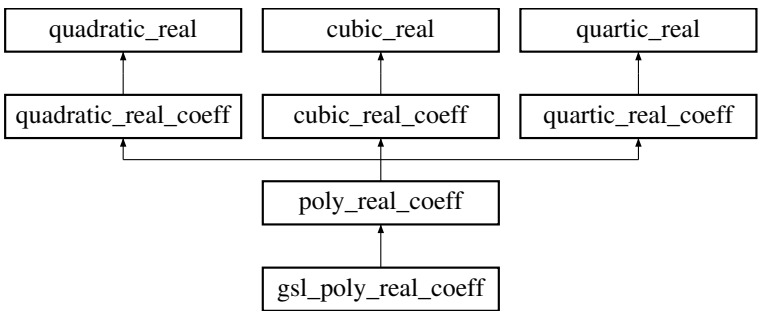
- poly.h

46.303 poly_real_coeff Class Reference

Solve a general polynomial with real coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for poly_real_coeff:



46.303.1 Detailed Description

Definition at line 326 of file poly.h.

Public Member Functions

- virtual int solve_rc (int n, const double co[ ], std::complex< double > ro[ ])=0
  Solve the n-th order polynomial.
- const char * type ()
  Return a string denoting the type ("poly_real_coeff")

46.303.2 Member Function Documentation

46.303.2.1 virtual int poly_real_coeff::solve_rc (int *n*, const double *co[]*, std::complex< double > *ro[]*) [pure virtual]

The coefficients are stored in co[], with the leading coefficient as co[0] and the constant term as co[n]. The roots are returned in ro[0],...,ro[n-1].

Implemented in gsl_poly_real_coeff.

The documentation for this class was generated from the following file:

- poly.h

46.304 polylog Class Reference

Polylogarithms (approximate) $Li_n(x)$.

`#include <polylog.h>`

46.304.1 Detailed Description

This class is experimental.

This gives an approximation to the polylogarithm functions.

Only works at present for $n = 0, 1, ..., 6$. Uses GSL library for n=2.

Uses linear interpolation for $-1 < x < 0$ and a series expansion for $x < -1$

Idea for Future

- Give error estimate?
- Improve accuracy?
- Use more sophisticated interpolation?
- Add the series $Li(n,x) = x + 2^{-n}x^2 + 3^{-n}x^3 + ...$ for $x \to 0$?
- Implement for positive arguments < 1.0
- Make another polylog class which implements series acceleration?

For reference, there are exact relations

$$\mathrm{Li}_2\left(\frac{1}{2}\right) = \frac{1}{12}\left[\pi^2 - 6(\ln 2)^2\right]$$

$$\mathrm{Li}_3\left(\frac{1}{2}\right) = \frac{1}{24}\left[4(\ln 2)^3 - 2\pi^2 \ln 2 + 21\zeta(3)\right]$$

$$\mathrm{Li}_{-1}(x) = \frac{x}{(1-x)^2}$$

$$\mathrm{Li}_{-2}(x) = \frac{x(x+1)}{(1-x)^3}$$

Definition at line 78 of file polylog.h.

Public Member Functions

- double li0 (double x)
  0-th order polylogarithm = $x/(1-x)$
- double li1 (double x)
  1-st order polylogarithm = $-\ln(1-x)$
- double li2 (double x)
  2-nd order polylogarithm
- double li3 (double x)
  3-rd order polylogarithm
- double li4 (double x)
  4-th order polylogarithm
- double li5 (double x)
  5-th order polylogarithm
- double li6 (double x)
  6-th order polylogarithm

The documentation for this class was generated from the following file:

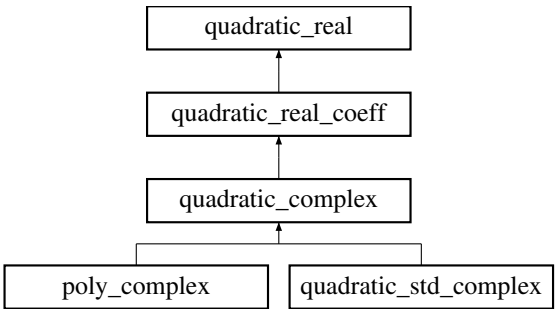
- polylog.h

46.305 quadratic_complex Class Reference

Solve a quadratic polynomial with complex coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for quadratic_complex:



46.305.1 Detailed Description

Definition at line 101 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a2, const double b2, const double c2, double &x1, double &x2)
  Solves the polynomial $a_2x^2 + b_2x + c_2 = 0$ *giving the two solutions* $x = x_1$ *and* $x = x_2$.
- virtual int solve_rc (const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)

Solves the polynomial $a_2x^2+b_2x+c_2=0$ giving the two complex solutions $x=x_1$ and $x=x_2$.

- virtual int solve_c (const std::complex< double > a2, const std::complex< double > b2, const std::complex< double > c2, std::complex< double > &x1, std::complex< double > &x2)=0
  Solves the complex polynomial $a_2x^2+b_2x+c_2=0$ giving the two complex solutions $x=x_1$ and $x=x_2$.
- const char * type ()
  Return a string denoting the type ("quadratic_complex")

The documentation for this class was generated from the following file:

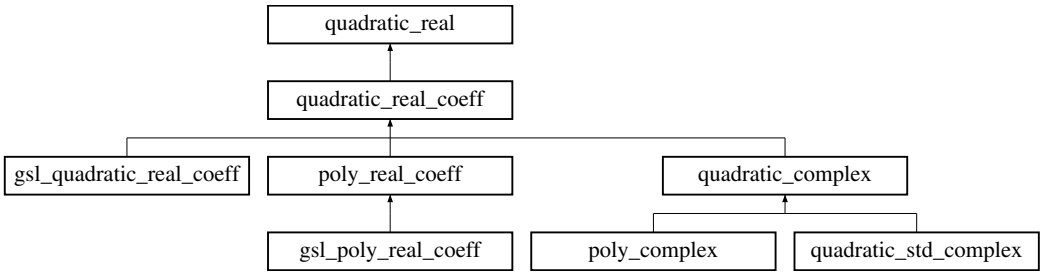
- poly.h

46.306 quadratic_real Class Reference

Solve a quadratic polynomial with real coefficients and real roots [abstract base].

`#include <poly.h>`

Inheritance diagram for quadratic_real:



46.306.1 Detailed Description

Definition at line 57 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a2, const double b2, const double c2, double &x1, double &x2)=0
  Solves the polynomial $a_2x^2+b_2x+c_2=0$ giving the two solutions $x=x_1$ and $x=x_2$.
- const char * type ()
  Return a string denoting the type ("quadratic_real")

The documentation for this class was generated from the following file:

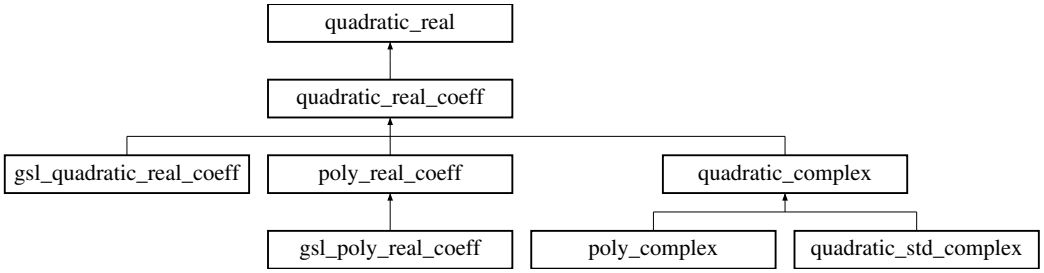
- poly.h

46.307 quadratic_real_coeff Class Reference

Solve a quadratic polynomial with real coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for quadratic_real_coeff:



46.307.1 Detailed Description

Definition at line 75 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a2, const double b2, const double c2, double &x1, double &x2)
  Solves the polynomial $a_2x^2 + b_2x + c_2 = 0$ giving the two solutions $x = x_1$ and $x = x_2$.
- virtual int solve_rc (const double a2, const double b2, const double c2, std::complex< double > &x1, std::complex< double > &x2)=0
  Solves the polynomial $a_2x^2 + b_2x + c_2 = 0$ giving the two complex solutions $x = x_1$ and $x = x_2$.
- const char * type ()
  Return a string denoting the type ("quadratic_real_coeff")

The documentation for this class was generated from the following file:

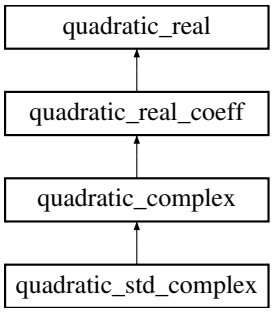
- poly.h

46.308 quadratic_std_complex Class Reference

Solve a quadratic with complex coefficients and complex roots.

`#include <poly.h>`

Inheritance diagram for quadratic_std_complex:



46.308.1 Detailed Description

Definition at line 673 of file poly.h.

Public Member Functions

- virtual int solve_c (const std::complex< double > a2, const std::complex< double > b2, const std::complex< double > c2, std::complex< double > &x1, std::complex< double > &x2)
  Solves the complex polynomial $a_2x^2+b_2x+c_2=0$ giving the two complex solutions $x=x_1$ and $x=x_2$.
- const char * type ()
  Return a string denoting the type ("quadratic_std_complex")

The documentation for this class was generated from the following file:

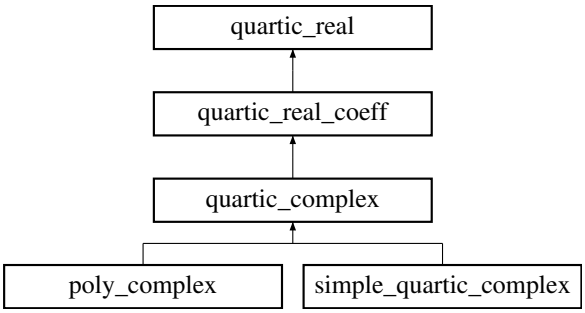
- poly.h

46.309 quartic_complex Class Reference

Solve a quartic polynomial with complex coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for quartic_complex:



46.309.1 Detailed Description

Definition at line 279 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)
  Solves the polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- virtual int solve_rc (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)
  Solves the polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four complex solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- virtual int solve_c (const std::complex< double > a4, const std::complex< double > b4, const std::complex< double > c4, const std::complex< double > d4, const std::complex< double > e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)=0
  Solves the complex polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four complex solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- const char * type ()
  Return a string denoting the type ("quartic_complex")

The documentation for this class was generated from the following file:

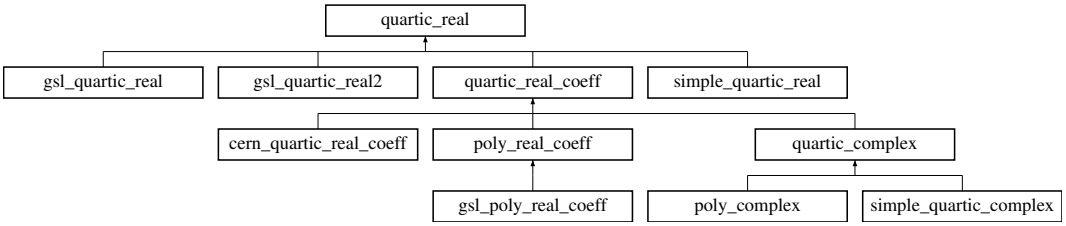
- poly.h

46.310 quartic_real Class Reference

Solve a quartic polynomial with real coefficients and real roots [abstract base].

`#include <poly.h>`

Inheritance diagram for quartic_real:



46.310.1 Detailed Description

Definition at line 223 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)=0
  Solves the polynomial $a_4x^4 + b_4x^3 + c_4x^2 + d_4x + e_4 = 0$ giving the four solutions $x = x_1$, $x = x_2$, $x = x_3$, and $x = x_4$.
- const char * type ()
  Return a string denoting the type ("quartic_real")

The documentation for this class was generated from the following file:

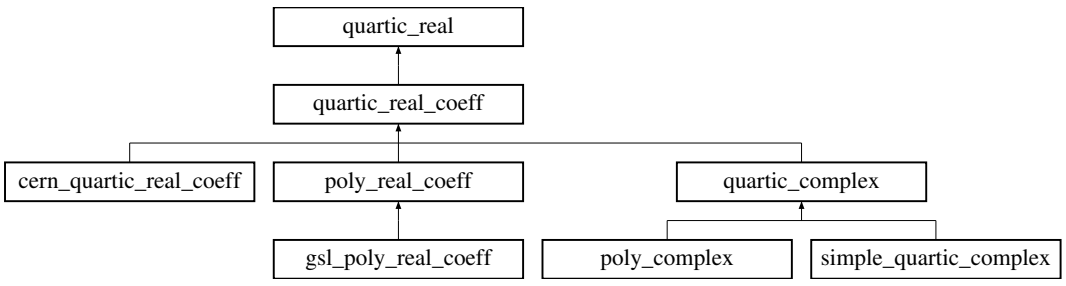
- poly.h

46.311 quartic_real_coeff Class Reference

Solve a quartic polynomial with real coefficients and complex roots [abstract base].

`#include <poly.h>`

Inheritance diagram for quartic_real_coeff:



46.311.1 Detailed Description

Definition at line 246 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)
  Solves the polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- virtual int solve_rc (const double a4, const double b4, const double c4, const double d4, const double e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)=0
  Solves the polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four complex solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- const char * type ()
  Return a string denoting the type ("quartic_real_coeff")

The documentation for this class was generated from the following file:

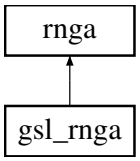
- poly.h

46.312 rnga Class Reference

Random number generator base.

`#include <rnga.h>`

Inheritance diagram for rnga:



46.312.1 Detailed Description

Using virtual functions is not recommended for random number generators, as speed is often an important issue. In order to facilitate the use of templates for routines which require random number generators, all descendants ought to provides the following functions:

- double random() - Provide a random number in [0.0,1.0)
- unsigned long int random_int(unsigned long int n) - Provide a random integer in [0,n-1]
- unsigned long int get_max() - Return the maximum integer for the random number generator. The argument to random_int() should be less than the value returned by get_max().

Idea for Future Consider some analog of the GSL function `gsl_rng_uniform_pos()`, i.e. as used in the GSL Monte Carlo classes.

Definition at line 52 of file rnga.h.

Public Member Functions

- void clock_seed ()
  Initialize the seed with a value taken from the computer clock.
- unsigned long int get_seed ()
  Get the seed.
- void set_seed (unsigned long int s)
  Set the seed.

Protected Member Functions

- **rnga** (const rnga &)
- rnga & **operator=** (const rnga &)

Protected Attributes

- unsigned long int seed
  The seed.

46.312.2 Member Function Documentation

46.312.2.1 void rnga::clock_seed ( )

This is a naive seed generator which uses `seed=time(0)` to generate a seed.

Idea for Future Figure out a better way of computing a random seed in a platform-independent way.

Reimplemented in gsl_rnga.

The documentation for this class was generated from the following file:

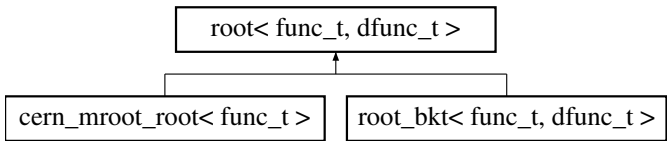
- rnga.h

46.313 root< func_t, dfunc_t > Class Template Reference

One-dimensional solver [abstract base].

`#include <root.h>`

Inheritance diagram for root< func_t, dfunc_t >:



46.313.1 Detailed Description

template<class func_t = funct, class dfunc_t = func_t>class root< func_t, dfunc_t >

Idea for Future Maybe consider allowing the user to specify the stream to which 'verbose' information is sent.

Definition at line 40 of file root.h.

Public Member Functions

- virtual const char * type ()
  Return the type, `"root"`.
- virtual int print_iter (double x, double y, int iter, double value=0.0, double limit=0.0, std::string comment="")
  Print out iteration information.
- virtual int solve (double &x, func_t &func)=0

Solve `func` *using* `x` *as an initial guess.*

- virtual int solve_bkt (double &x1, double x2, func_t &func)
  Solve `func` *in region* $x_1 < x < x_2$ *returning* x_1 .
- virtual int solve_de (double &x, func_t &func, dfunc_t &df)
  Solve `func` *using* `x` *as an initial guess using derivatives* `df`.

Data Fields

- double tol_rel
  The maximum value of the functions for success (default 10^{-8} *)*
- double tol_abs
  The minimum allowable stepsize (default 10^{-12} *)*
- int verbose
  Output control (default 0)
- int ntrial
  Maximum number of iterations (default 100)
- bool err_nonconv
  If true, call the error handler if the routine does not "converge".
- int last_conv
  Zero if last call to solve(), solve_bkt(), or solve_de() converged.
- int last_ntrial
  The number of iterations for in the most recent minimization.

46.313.2 Member Function Documentation

46.313.2.1 template<class func_t = funct, class dfunc_t = func_t> virtual int root< func_t, dfunc_t >::print_iter (double *x*, double *y*, int *iter*, double *value* = 0.0, double *limit* = 0.0, std::string *comment* = " ") [inline, virtual]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character before continuing.

Definition at line 101 of file root.h.

46.313.3 Field Documentation

46.313.3.1 template<class func_t = funct, class dfunc_t = func_t> int root< func_t, dfunc_t >::last_conv

This is particularly useful if err_nonconv is false to test if the last call to solve(), solve_bkt(), or solve_de() converged.

Definition at line 82 of file root.h.

The documentation for this class was generated from the following file:

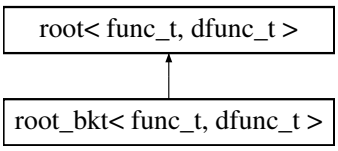
- root.h

46.314 root_bkt< func_t, dfunc_t > Class Template Reference

One-dimensional bracketing solver [abstract base].

`#include <root.h>`

Inheritance diagram for root_bkt< func_t, dfunc_t >:



46.314.1 Detailed Description

template<class func_t = funct, class dfunc_t = func_t>class root_bkt< func_t, dfunc_t >

Definition at line 147 of file root.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "root_bkt".
- virtual int solve_bkt (double &x1, double x2, func_t &func)=0
  Solve func in region $x_1 < x < x_2$ returning x_1 .
- virtual int solve (double &x, func_t &func)
  Solve func using x as an initial guess.
- virtual int solve_de (double &x, func_t &func, dfunc_t &df)
  Solve func using x as an initial guess using derivatives df.

Data Fields

- double bracket_step
  The stepsize for automatic bracketing (default 10^{-4})
- size_t bracket_iters
  The number of iterations in attempt to bracket root (default 10)

46.314.2 Field Documentation

46.314.2.1 template<class func_t = funct, class dfunc_t = func_t> double root_bkt< func_t, dfunc_t >::bracket_step

If this is exactly zero, it will be reset to 10^{-4} by solve().

Definition at line 165 of file root.h.

The documentation for this class was generated from the following file:

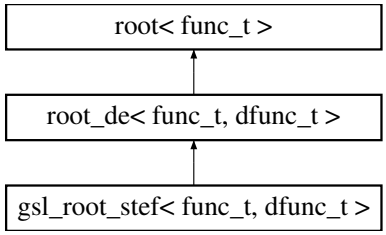
- root.h

46.315 root_de< func_t, dfunc_t > Class Template Reference

One-dimensional with solver with derivatives [abstract base].

`#include <root.h>`

Inheritance diagram for root_de< func_t, dfunc_t >:



46.315.1 Detailed Description

template<class func_t = funct, class dfunc_t = func_t>class root_de< func_t, dfunc_t >

Idea for Future At the moment, the functions solve() and solve_bkt() are not implemented for derivative solvers.

Definition at line 259 of file root.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "root_de".
- virtual int solve_bkt (double &x1, double x2, func_t &func)
  Solve func in region $x_1 < x < x_2$ returning x_1 .
- virtual int solve (double &x, func_t &func)
  Solve func using x as an initial guess.
- virtual int solve_de (double &x, func_t &func, dfunc_t &df)=0
  Solve func using x as an initial guess using derivatives df.

The documentation for this class was generated from the following file:

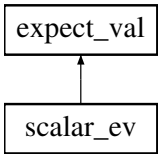
- root.h

46.316 scalar_ev Class Reference

Scalar expectation value.

`#include <expect_val.h>`

Inheritance diagram for scalar_ev:



46.316.1 Detailed Description

See expect_val for some general notes on this and related classes.

This class is experimental.

Implementation notes

In the case where nperblock is zero, the vector last is used to store the most recent measurements, copying the results to vals only when nblocks measurements are stored in last. The variable i is only incremented when the vector last is full and the information is copied over to vals. This ensures that vals always contains averages with an equal number of measurements. The information in last is never used by reblock_avg().

Definition at line 201 of file expect_val.h.

Public Member Functions

- scalar_ev ()
  Create with one unspecified block.
- scalar_ev (size_t n_blocks, size_t n_per_block)
  Create with n_blocks blocks and n_per_block points block.
- scalar_ev (const scalar_ev &ev)
  Copy constructor.
- scalar_ev & operator= (const scalar_ev &ev)
  Copy constructor.
- virtual void set_blocks (size_t n_blocks, size_t n_per_block)
  Reset for n_blocks blocks and n_per_block points block.
- virtual void free ()
  Free allocated data.
- virtual void reset ()
  Clear all the data.
- virtual void add (double val)
  Add measurement of value val.

Report statistics

- virtual void current_avg_stats (double &avg, double &std_dev, double &avg_err, size_t &m_block, size_t &m_per_block) const
  Report current average, standard deviation, and the error in the average and include block information.
- virtual void current_avg (double &avg, double &std_dev, double &avg_err) const
  Report current average, standard deviation, and the error in the average.
- virtual void reblock_avg_stats (size_t new_blocks, double &avg, double &std_dev, double &avg_err, size_t &m_per_block) const
  Report average, standard deviation, and the error in the average assuming a new block size.
- virtual void reblock_avg (size_t new_blocks, double &avg, double &std_dev, double &avg_err) const
  Report average, standard deviation, and the error in the average assuming a new block size.

Direct manipulation of the stored data

- const uvector & get_data () const
  Return the current data for all blocks.
- const double & operator[ ] (size_t i_block) const
  Return the current data for block with index i_block.
- double & operator[ ] (size_t i_block)
  Return the current data for block with index i_block.
- template<class vec_t >
  void set_data (vec_t &v)
  Set the data for all blocks.

Protected Attributes

- uvector vals
  The average for each block.
- double current
  The current rolling average.
- uvector last
  The most recent values for each block.

46.316.2 Member Function Documentation

46.316.2.1 virtual void scalar_ev::reblock_avg_stats (size_t *new_blocks*, double & *avg*, double & *std_dev*, double & *avg_err*, size_t & *m_per_block*) const [virtual]

Idea for Future Use recurrence relation for averages here rather than dividing at the end.

46.316.3 Field Documentation

46.316.3.1 uvector scalar_ev::vals [protected]

This is a vector of length `nblocks`.

Definition at line 209 of file expect_val.h.

46.316.3.2 uvector scalar_ev::last [protected]

This is a vector of length `nblocks` used for when `n_per_block` is set to zero.

Definition at line 219 of file expect_val.h.

The documentation for this class was generated from the following file:

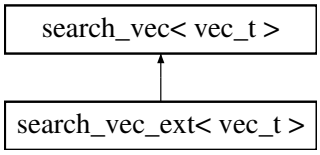
- expect_val.h

46.317 search_vec< vec_t > Class Template Reference

Searching class for monotonic data with caching.

`#include <search_vec.h>`

Inheritance diagram for search_vec< vec_t >:



46.317.1 Detailed Description

template<class vec_t>class search_vec< vec_t >

A searching class for monotonic vectors. A caching system similar to `gsl_interp_accel` is used.

To find the interval containing a value, use find(). If you happen to know in advance that the vector is increasing or decreasing, then you can use find_inc() or find_dec() instead. To ignore the caching and just use straight binary search, you can use the functions in vector.h .

Alternatively, if you just want to find the index with the element closest to a specified value, use ordered_lookup(). Note that ordered_lookup() is slightly different than ovector_tlate::lookup(), since the latter does not assume the data is monotonic.

The functions find_inc(), find_dec() and find() are designed to return the lower index of an interval containing the desired point, and as a result will never return the last index of a vector, e.g. for a vector of size `n` they always return a number between `0` and `n-2` inclusive. See search_vec_ext for an alternative.

Note

The behavior of these functions is undefined if some of the user-specified data is not finite or not strictly monotonic. Two adjacent data points should not be equal. This class does not verify that the user-specified data has these properties.
This class does not store a copy of the data, but only a pointer to it. This means that one can safely modify the data after the constructor is called, so long as one does not make the vector smaller (as the cache might then point to a value outside the new vector) and so long as the new vector is still monotonic.

Idea for Future Consider a sanity check to ensure that the cache is never larger than the vector length.

Definition at line 77 of file search_vec.h.

Public Member Functions

- search_vec (size_t nn, const vec_t &x)
  Create a searching object with vector `x` of size `nn`.
- size_t find (const double x0) const
  Search an increasing or decreasing vector for the interval containing `x0`
- size_t find_inc (const double x0) const
  Search an increasing vector for the interval containing `x0`
- size_t find_dec (const double x0) const
  Search a decreasing vector for the interval containing `x0`
- size_t ordered_lookup (const double x0) const
  Find the index of x0 in the ordered array `x`.

Protected Attributes

- size_t cache
  Storage for the most recent index.
- const vec_t * v
  The vector to be searched.
- const size_t n
  The vector size.

46.317.2 Member Function Documentation

46.317.2.1 template<class vec_t> size_t search_vec< vec_t >::find (const double *x0*) const [inline]

This function is identical to find_inc() if the data is increasing, and find_dec() if the data is decreasing.

Reimplemented in search_vec_ext< vec_t >.

Definition at line 116 of file search_vec.h.

46.317.2.2 template<class vec_t> size_t search_vec< vec_t >::find_inc (const double *x0*) const [inline]

This function is a cached version of vector_bsearch_inc() , analogous to `gsl_interp_accel_find()`, except that it does not internally record cache hits and misses.

Reimplemented in search_vec_ext< vec_t >.

Definition at line 130 of file search_vec.h.

46.317.2.3 template<class vec_t> size_t search_vec< vec_t >::find_dec (const double *x0*) const [inline]

This function is a cached version of vector_bsearch_dec() . The operation of this function is undefined if the data is not strictly monotonic, i.e. if some of the data elements are equal.

Reimplemented in search_vec_ext< vec_t >.

Definition at line 147 of file search_vec.h.

46.317.2.4 template<class vec_t> size_t search_vec< vec_t >::ordered_lookup (const double *x0*) const [inline]

This returns the index i for which x[i] is as close as possible to x0 if x[i] is either increasing or decreasing.

If you have a non-monotonic vector, you can use vector_lookup() instead, or if you a non-monotonic ovector or uvector object, consider using ovector_view_tlate::lookup() or uvector_view_tlate::lookup() instead of this function.

Generally, if there are two adjacent entries with the same value, this function will return the entry with the smaller index. (This is the same as the convention in ovector_view_tlate::lookup() and uvector_view_tlate::lookup()).

Idea for Future This function just uses the `find` functions and then adjusts the answer at the end if necessary. It might be possible to improve the speed by rewriting this from scratch.

Definition at line 178 of file search_vec.h.

46.317.3 Field Documentation

46.317.3.1 template<class vec_t> size_t search_vec< vec_t >::cache [mutable, protected]

Note

This is marked mutable to ensure const-correctness is straightforward.

Definition at line 88 of file search_vec.h.

The documentation for this class was generated from the following file:

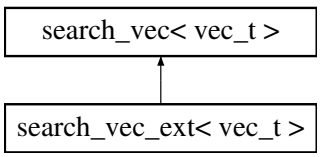
- search_vec.h

46.318 search_vec_ext< vec_t > Class Template Reference

An extended search_vec which is allowed to return the last element.

`#include <search_vec.h>`

Inheritance diagram for search_vec_ext< vec_t >:



46.318.1 Detailed Description

template<class vec_t>class search_vec_ext< vec_t >

Definition at line 217 of file search_vec.h.

Public Member Functions

- search_vec_ext (size_t nn, vec_t &x)
  Create a searching object for vector x of size nn.

- size_t find (const double x0) const
  Search an increasing or decreasing vector for the interval containing x0
- size_t find_inc (const double x0) const
  Search an increasing vector for the interval containing x0
- size_t find_dec (const double x0) const
  Search a decreasing vector for the interval containing x0

46.318.2 Constructor & Destructor Documentation

46.318.2.1 template<class vec_t > search_vec_ext< vec_t >::search_vec_ext (size_t *nn*, vec_t & *x*) [inline]

Idea for Future This could be rewritten to allow vectors with only one element (5/2/11: It looks like this might have been done already?)

Create a searching object with vector x of size nn

Definition at line 230 of file search_vec.h.

The documentation for this class was generated from the following file:

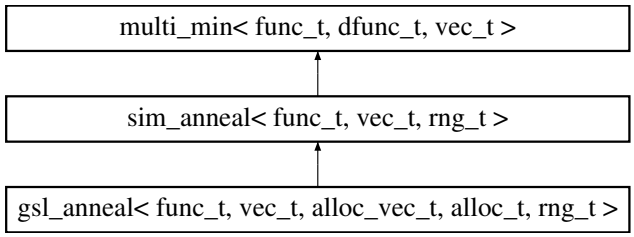
- search_vec.h

46.319 sim_anneal< func_t, vec_t, rng_t > Class Template Reference

Simulated annealing base.

`#include <sim_anneal.h>`

Inheritance diagram for sim_anneal< func_t, vec_t, rng_t >:



46.319.1 Detailed Description

template<class func_t = multi_funct, class vec_t = ovector_base, class rng_t = gsl_rnga>class sim_anneal< func_t, vec_t, rng_t >

The seed of the generator is not fixed initially by calls to mmin(), so if successive calls should reproduce the same results, then the random seed should be set by the user before each call.

For the algorithms here, it is important that all of the inputs x[i] to the function are scaled similarly relative to the temperature. For example, if the inputs x[i] are all of order 1, one might consider a temperature schedule which begins with $T = 1$.

The number of iterations at each temperature is controlled by multi_min::ntrial which defaults to 100.

Definition at line 55 of file sim_anneal.h.

Public Member Functions

- virtual int mmin (size_t nvar, vec_t &x, double &fmin, func_t &func)=0

Calculate the minimum `fmin` *of* `func` *w.r.t the array* `x` *of size* `nvar`.

- virtual int print_iter (size_t nv, vec_t &x, double y, int iter, double tptr, std::string comment)
  Print out iteration information.
- virtual const char * type ()
  Return string denoting type, `"sim_anneal"`.

Data Fields

- rng_t def_rng
  The default random number generator.

46.319.2 Member Function Documentation

46.319.2.1 template<class func_t = multi_funct, class vec_t = ovector_base, class rng_t = gsl_rnga> virtual int sim_anneal< func_t, vec_t, rng_t >::print_iter (size_t *nv*, vec_t & *x*, double *y*, int *iter*, double *tptr*, std::string *comment*) [inline, virtual]

Depending on the value of the variable verbose, this prints out the iteration information. If verbose=0, then no information is printed, while if verbose>1, then after each iteration, the present values of x and y are output to std::cout along with the iteration number. If verbose>=2 then each iteration waits for a character.

Definition at line 87 of file sim_anneal.h.

The documentation for this class was generated from the following file:

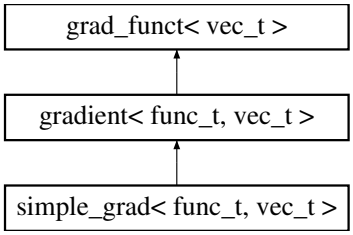
- sim_anneal.h

46.320 simple_grad< func_t, vec_t > Class Template Reference

Simple automatic computation of gradient by finite differencing.

`#include <multi_min.h>`

Inheritance diagram for simple_grad< func_t, vec_t >:



46.320.1 Detailed Description

template<class func_t, class vec_t>class simple_grad< func_t, vec_t >

Definition at line 398 of file multi_min.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &g)
  Compute the gradient `g` *at the point* `x`.

Data Fields

- double epsrel
  The relative stepsize for finite-differencing (default 10^{-6})
- double epsmin
  The minimum stepsize (default 10^{-15})

The documentation for this class was generated from the following file:

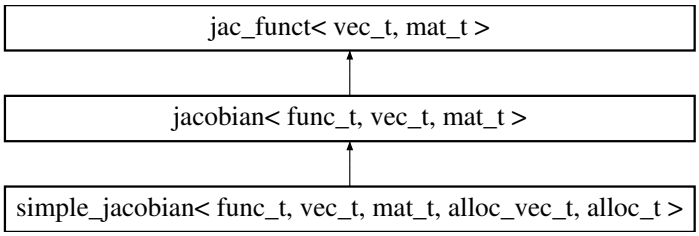
- multi_min.h

46.321 simple_jacobian< func_t, vec_t, mat_t, alloc_vec_t, alloc_t > Class Template Reference

Simple automatic Jacobian.

`#include <jacobian.h>`

Inheritance diagram for simple_jacobian< func_t, vec_t, mat_t, alloc_vec_t, alloc_t >:



46.321.1 Detailed Description

template<class func_t = mm_funct<>, class vec_t = ovector_base, class mat_t = omatrix_base, class alloc_vec_t = ovector, class alloc_t = ovector_alloc>class simple_jacobian< func_t, vec_t, mat_t, alloc_vec_t, alloc_t >

This class computes a numerical Jacobian by finite differencing. The stepsize is chosen to be $h_j =$ epsrel x_j or $h_j =$ epsmin if epsrel $x_j <$ epsmin.

This is nearly equivalent to the GSL method for computing Jacobians as in `multiroots/fdjac.c`. To obtain the GSL behavior, set epsrel to `GSL_SQRT_DBL_EPSILON` and set epsmin to zero. The gsl_mroot_hybrids class sets epsrel to `GSL_SQRT_DBL_EPSILON` in its constructor, but does not set epsmin to zero.

This class does not separately check the vector and matrix sizes to ensure they are commensurate.

Definition at line 273 of file jacobian.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &jac)
  The operator()

Data Fields

- double epsrel
  The relative stepsize for finite-differencing (default 10^{-4})
- double epsmin
  The minimum stepsize (default 10^{-15})
- bool err_nonconv
  If true, call the error handler if the routine does not "converge".

Protected Attributes

- alloc_t ao
  For memory allocation.
- alloc_vec_t f
  Function values.
- alloc_vec_t xx
  Function arguments.
- size_t mem_size
  Size of allocated memory.

The documentation for this class was generated from the following file:

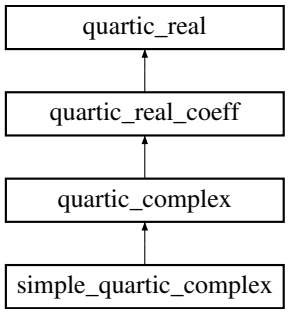
- jacobian.h

46.322 simple_quartic_complex Class Reference

Solve a quartic with complex coefficients and complex roots.

`#include <poly.h>`

Inheritance diagram for simple_quartic_complex:



46.322.1 Detailed Description

Definition at line 742 of file poly.h.

Public Member Functions

- virtual int solve_c (const std::complex< double > a4, const std::complex< double > b4, const std::complex< double > c4, const std::complex< double > d4, const std::complex< double > e4, std::complex< double > &x1, std::complex< double > &x2, std::complex< double > &x3, std::complex< double > &x4)
  Solves the complex polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four complex solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- const char * type ()
  Return a string denoting the type ("simple_quartic_complex")

The documentation for this class was generated from the following file:

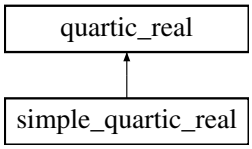
- poly.h

46.323 simple_quartic_real Class Reference

Solve a quartic with real coefficients and real roots.

`#include <poly.h>`

Inheritance diagram for simple_quartic_real:



46.323.1 Detailed Description

Definition at line 717 of file poly.h.

Public Member Functions

- virtual int solve_r (const double a4, const double b4, const double c4, const double d4, const double e4, double &x1, double &x2, double &x3, double &x4)
  Solves the polynomial $a_4x^4+b_4x^3+c_4x^2+d_4x+e_4=0$ giving the four solutions $x=x_1$, $x=x_2$, $x=x_3$, and $x=x_4$.
- const char * type ()
  Return a string denoting the type ("simple_quartic_real")

Data Fields

- double cube_root_tol
  A tolerance for determining the proper cube root (default 10^{-6})

The documentation for this class was generated from the following file:

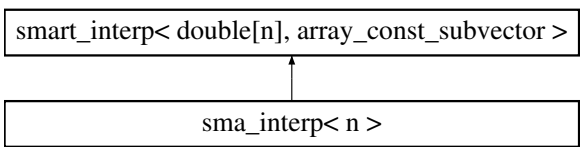
- poly.h

46.324 sma_interp< n > Class Template Reference

A specialization of smart_interp for C-style double arrays.

`#include <smart_interp.h>`

Inheritance diagram for sma_interp< n >:



46.324.1 Detailed Description

template<size_t n>class sma_interp< n >

Definition at line 807 of file smart_interp.h.

Public Member Functions

- sma_interp (base_interp_mgr< double[n]> &it1, base_interp_mgr< array_const_subvector > &it3)
  Create an interpolation object with user-specified interpolation types.
- sma_interp ()
  Create an interpolation object with the default interpolation types.

The documentation for this class was generated from the following file:

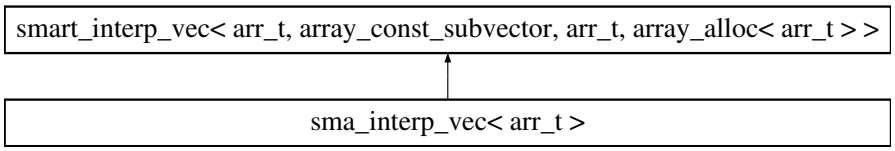
- smart_interp.h

46.325 sma_interp_vec< arr_t > Class Template Reference

A specialization of smart_interp_vec for C-style double arrays.

`#include <smart_interp.h>`

Inheritance diagram for sma_interp_vec< arr_t >:



46.325.1 Detailed Description

template<class arr_t>class sma_interp_vec< arr_t >

Definition at line 828 of file smart_interp.h.

Public Member Functions

- sma_interp_vec (base_interp_mgr< arr_t > &it, base_interp_mgr< array_const_subvector > &it2, size_t n, const arr_t &x, const arr_t &y)
  Create an interpolation object with user-specified interpolation types.
- sma_interp_vec (size_t n, const arr_t &x, const arr_t &y)
  Create an interpolation object with the default interpolation types.

The documentation for this class was generated from the following file:

- smart_interp.h

46.326 smart_interp< vec_t, svec_t > Class Template Reference

Smart interpolation class.

`#include <smart_interp.h>`

46.326.1 Detailed Description

template<class vec_t = ovector_const_view, class svec_t = ovector_const_subvector>class smart_interp< vec_t, svec_t >

This class can semi-intelligently handle arrays which are not-well formed outside the interpolation region. In particular, if an initial interpolation or derivative calculation fails, the arrays are searched for the largest neighborhood around the point x for which an interpolation or differentiation will likely produce a finite result.

Definition at line 51 of file smart_interp.h.

Public Member Functions

- smart_interp (base_interp_mgr< vec_t > &im1, base_interp_mgr< svec_t > &im3)
  Create a new interpolation object with the specified interpolation managers.
- smart_interp ()
  Create an interpolation object with the default cubic spline interpolation managers.
- virtual double interp (const double x0, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the function $y(x=x_0)$.
- virtual double deriv (const double x0, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the derivative $y^{prime}(x=x_0)$.
- virtual double deriv2 (const double x0, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the second derivative $y^{primet}(x=x_0)$.
- virtual double integ (const double a, const double b, size_t n, const vec_t &x, const vec_t &y)
  Give the value of the integral $\int_a^b y(x)\,dx$.

Data Fields

Default interpolation managers

- def_interp_mgr< vec_t, cspline_interp > **dim1**
- def_interp_mgr< svec_t, cspline_interp > **dim3**

Protected Member Functions

- size_t local_lookup (size_t n, const vec_t &x, double x0)
  A lookup function for generic vectors.
- int find_subset (const double a, const double b, size_t sz, const vec_t &x, const vec_t &y, size_t &nsz)
  Try to find the largest monotonic and finite region around the desired location.

Protected Attributes

Storage internally created subvectors

- bool **sxalloc**
- const svec_t * **sx**
- const svec_t * **sy**

Pointers to interpolation objects

- base_interp< vec_t > * **rit1**
- base_interp< svec_t > * **rit3**

Pointers to interpolation managers

- base_interp_mgr< vec_t > * **bim1**
- base_interp_mgr< svec_t > * **bim3**

46.326.2 Member Function Documentation

46.326.2.1 template<class vec_t = ovector_const_view, class svec_t = ovector_const_subvector> virtual double smart_interp< vec_t, svec_t >::interp (const double *x0*, size_t *n*, const vec_t & *x*, const vec_t & *y*) [inline, virtual]

Todo After calling find_subset, I think we might need to double check that `nn` is larger than the minimum interpolation size.

Definition at line 134 of file smart_interp.h.

46.326.2.2 template<class vec_t = ovector_const_view, class svec_t = ovector_const_subvector> int smart_interp< vec_t, svec_t >::find_subset (const double *a*, const double *b*, size_t *sz*, const vec_t & *x*, const vec_t & *y*, size_t & *nsz*) [inline, protected]

This function tries to find the largest monotonic region enclosing both `a` and `b` in the vector `x`. If it succeeds, it returns gsl_success, and if it fails, it returns gsl_efailed. It does not call the error handler.

Todo The error handling is a bit off here, as it can return a non-zero value even with there is no real "error". We should just make a new bool reference paramter.

Definition at line 409 of file smart_interp.h.

The documentation for this class was generated from the following file:

- smart_interp.h

46.327 smart_interp_vec< vec_t, svec_t, alloc_vec_t, alloc_t > Class Template Reference

Smart interpolation class with pre-specified vectors.

`#include <smart_interp.h>`

46.327.1 Detailed Description

template<class vec_t, class svec_t, class alloc_vec_t, class alloc_t>class smart_interp_vec< vec_t, svec_t, alloc_vec_t, alloc_t >

This class can semi-intelligently handle arrays which are not-well formed outside the interpolation region. In particular, if an initial interpolation or derivative calculation fails, the arrays are searched for the largest neighborhood around the point `x` for which an interpolation or differentiation will likely produce a finite result.

Idea for Future Properly implement handling of non-monotonic regions in the derivative functions as well as the interpolation function.

Definition at line 534 of file smart_interp.h.

Public Member Functions

- smart_interp_vec (size_t n, const vec_t &x, const vec_t &y)
  Create with base interpolation objects `it` and `rit`.
- smart_interp_vec (base_interp_mgr< vec_t > &it1, base_interp_mgr< svec_t > &it2, size_t n, const vec_t &x, const vec_t &y)
  Create with base interpolation objects `it` and `rit`.
- virtual double operator() (const double x0)
  Give the value of the function $y(x = x_0)$.
- virtual double interp (const double x0)
  Give the value of the function $y(x = x_0)$.

- virtual double deriv (const double x0)
  Give the value of the derivative $y^{prime}(x = x_0)$.
- virtual double deriv2 (const double x0)
  Give the value of the second derivative $y^{prime\prime}(x = x_0)$.
- virtual double integ (const double x1, const double x2)
  Give the value of the integral $\int_a^b y(x)\,dx$.

Data Fields

- def_interp_mgr< vec_t, cspline_interp > dim1
  Default interpolation manager.
- def_interp_mgr< svec_t, cspline_interp > dim2
  Default interpolation manager.

Protected Member Functions

- size_t local_lookup (size_t n, const vec_t &x, double x0)
  A lookup function for generic vectors.
- int find_inc_subset (const double x0, size_t sz, const vec_t &x, const vec_t &y, size_t &nsz)
  Try to find the largest monotonically increasing and finite region around the desired location.

Protected Attributes

- bool sxalloc
  If true, then sx and sy have been allocated.
- svec_t * sx
  Storage for internally created subvector.
- svec_t * sy
  Storage for internally created subvector.
- base_interp< vec_t > * rit1
  Pointer to base interpolation object.
- base_interp< svec_t > * rit2
  Pointer to base interpolation object.
- base_interp_mgr< vec_t > * bim1
  Pointer to base interpolation manager.
- base_interp_mgr< svec_t > * bim2
  Pointer to base interpolation manager.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- bool inc
  True if the user-specified x vector is increasing.
- const vec_t * lx
  Pointer to user-specified vector.
- const vec_t * ly
  Pointer to user-specified vector.
- size_t ln
  Size of user-specifed vector.

46.327.2 Member Function Documentation

46.327.2.1 template<class vec_t, class svec_t, class alloc_vec_t, class alloc_t> int smart_interp_vec< vec_t, svec_t, alloc_vec_t, alloc_t >::find_inc_subset (const double *x0*, size_t *sz*, const vec_t & *x*, const vec_t & *y*, size_t & *nsz*) [inline, protected]

This function looks through the vector `x` near the element closest to `x0` to find the largest possible monotonic region. If it succeeds, it returns gsl_success, and if it fails, it returns gsl_efailed. It does not call the error handler.

Todo Variables `row` and `row` appear to be unused? (2/17/11)

Definition at line 761 of file smart_interp.h.

The documentation for this class was generated from the following file:

- smart_interp.h

46.328 table::sortd_s Struct Reference

A structure for sorting in table [protected].

`#include <table.h>`

46.328.1 Detailed Description

This struct is used internally by table to perform sorting and need not be instantiated by the casual end-user.

Definition at line 1052 of file table.h.

Data Fields

- double val
  Value to sort.
- int indx
  Sorted index.

The documentation for this struct was generated from the following file:

- table.h

46.329 string_comp Struct Reference

Simple string comparison.

`#include <misc.h>`

46.329.1 Detailed Description

This struct is used internally by O_2scl for the STL routines which require a way to compare strings in the class table and in the I/O classes.

Definition at line 169 of file misc.h.

Public Member Functions

- bool operator() (const std::string s1, const std::string s2) const
  Return `s1<s2`.

The documentation for this struct was generated from the following file:

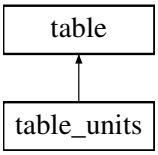
- misc.h

46.330 table Class Reference

Data table class.

`#include <table.h>`

Inheritance diagram for table:



46.330.1 Detailed Description

Summary

A class to contain and manipulate several equally-sized columns of data. The purpose of this class is to provide a structure which allows one to refer to the columns using a name represented by a string. Thus for a table object named `t` with 3 columns (named "colx", "coly" and "colz") and three rows, one could do the following:

```
// Set the 1st row of column "colx" to 1.0
t.set("colx",0,1.0);
// Set the 2nd row of column "colz" to 2.0
t.set("colz",1,2.0);
// Set the 3rd row of column "coly" to 4.0
t.set("coly",2,4.0);
// This will print out 2.0
cout << t.get("colz",1) << endl;
```

Note that the rows are numbered starting with 0 instead of starting with 1. To output all the rows of entire column, one can use

```
for(size_t i=0;i<t.get_nlines();i++) {
  cout << i << " " << t.get("colx",i) << endl;
}
```

To output all the columns of an entire row (in the following example it is the second row), labeled by their column name, one can use:

```
for(size_t i=0;i<t.get_ncolumns();i++) {
  cout << t.get_column_name(i) << " ";
}
cout << endl;
for(size_t i=0;i<t.get_ncolumns();i++) {
  cout << t.get(i,1) << " ";
}
cout << endl;
```

Methods are provided for interpolating columns, sorting columns, finding data points, and several other manipulations of the data.

Column size

The columns grow automatically (similar to the STL <vector>) in reponse to an attempt to call set() for a row that does not presently exist or in a call to line_of_data() when the table is already full. However, this forces memory rearrangments that are expensive, O(R∗C). If the user has a good estimate of the number of rows beforehand, it is best to either specify this in the constructor, or in an explicit call to inc_maxlines().

Lookup, differentiation, integration, and interpolation

Lookup, differentiation, integration, and interpolation are automatically implemented using splines from the class smart_interp_vec. A caching mechanism is implemented so that successive interpolations, derivative evaluations or integrations over the same two columns are fast.

Sorting

The columns are automatically sorted by name for speed, the results can be accessed by table::get_sorted_name(i). Individual columns can be sorted (sort_column()), or the entire table can be sorted by one column (sort_table()).

Data representation

Each individual column is just an ovector_base object. The columns can be referred to in one of two ways:

- A numerical index from 0 to C-1 (where C is the number of columns). For example, data can be accessed through table::get() and table::set(size_t c, size_t r, double val), or the overloaded [] operator, `table[c][r]`.
- A name of the column which is a string with no whitespace. For example, data can be accessed with table::get(string cname, int r) and table::set(string cname, int r, double val).

The columns are organized in a both a <map> and a <vector> structure so that finding a column by its index, using either of

```
std::string table::get_column_name(size_t index);
ovector_base &table::get_column(int index);
```

takes only constant time, and finding a column by its name using either of

```
int lookup_column(std::string name, int &ix) const;
const ovector_base &get_column(std::string col) const;
```

is O(log(C)). Insertion of a column (new_column()) is O(log(C)), but deletion (delete_column()) is O(C). Adding a row of data can be either O(1) or O(C), but row insertion and deletion is slow, since the all of the rows must be shifted accordingly.

Because of the structure, this class is not suitable for the matrix manipulation. The classes omatrix and umatrix are better used for that purpose.

Thread-safety

Generally, the member functions are thread-safe in the sense that one would expect: it is always permitted to have many threads accessing many distinct tables, but care must be taken if many threads are accessing only one table. Simple get() and set() functions are thread-safe (unless a set() function forces memory rearrangement), while insertion and deletion operations are not. Only the `const` version of the interpolation routines are thread-safe.

I/O and command-line manipulation

When data from an object of type table is output to a file through a `hdf_output()` function (see hdf_io.h), the table can be manipulated on the command-line through the `acol` utility.

There is an example for the usage of this class given in `examples/ex_table.cpp`.

Todo Document or fix the fact that the table copy constructors do not copy the interpolation objects. Maybe def_interp_mgr objects can have their own copy constructors?

Idea for Future

- Rewrite the table::create_array() and table::insert_data() functions for generic vector type
- Return the empty column in the operator[] functions as is done for the get_column() functions.
- A "delete rows" method to delete a range of several rows
- The present structure, `std::map<std::string,col,string_comp>` atree and `std::vector<aiter>` alist; could be replaced with `std::vector<col>` list and `std::map<std::string,int>` tree where the map just stores the index of the the column in the list.

Definition at line 191 of file table.h.

Data Structures

- struct col_s
  Column structure for table [protected].
- struct sortd_s
  A structure for sorting in table [protected].

Public Member Functions

- table (int cmaxlines=0)
  Create a new table with space for nlines<=cmaxlines.
- table (const table &t)
  Copy constructor.
- table & operator= (const table &t)
  Copy constructor.
- virtual const char * type ()
  Return the type, "table".

Basic get and set methods

- int set (std::string col, size_t row, double val)
  Set row row of column named col to value val - O(log(C)).
- int set (size_t icol, size_t row, double val)
  Set row row of column number icol to value val.
- double get (std::string col, size_t row) const
  Get value from row row of column named col - O(log(C)).
- double get (size_t icol, size_t row) const
  Get value from row row of column number icol - O(1).
- size_t get_ncolumns () const
  Return the number of columns.
- size_t get_nlines () const
  Return the number of lines.
- int set_nlines (size_t il)
  Set the number of lines.
- int set_nlines_auto (size_t il)
  Set the number of lines, increasing the size more agressively.
- int get_maxlines ()
  Return the maximum number of lines.
- ovector_base & get_column (std::string col)
  Returns a reference to the column named col - O(log(C))
- const ovector_base & get_column (std::string col) const
  Returns a reference to the column named col - O(log(C))
- const ovector_base & operator[ ] (size_t icol) const
  Returns the column of index icol - O(1) (const version)
- ovector_base & operator[ ] (size_t icol)
  Returns the column of index icol - O(1).
- const ovector_base & operator[ ] (std::string scol) const
  Returns the column named scol - O(log(C)) (const version)
- ovector_base & operator[ ] (std::string scol)
  Returns the column named scol - O(log(C))
- int get_row (std::string col, double val, ovector &row) const
  *Returns a copy of the row with value val in column col - O(R*C)*
- int get_row (size_t irow, ovector &row) const
  Returns a copy of row number irow - O(C)

Histogram-like functions

- int hist_update (std::string ix_col, std::string scol, double ix, double inc=1.0)
  Update a histogram entry.
- double hist_get (std::string ix_col, std::string scol, double ix)
  Get a histogram entry.
- int hist_set (std::string ix_col, std::string scol, double ix, double value)
  Set a histogram entry.

Column manipulation

- std::string get_column_name (size_t col) const
  Returns the name of column col - O(1).
- std::string get_sorted_name (size_t col)
  Returns the name of column col in sorted order - O(1).
- int new_column (std::string name)

Add a new column owned by the table - O(log(C)).

- template<class vec_t >
  int new_column (std::string name, size_t sz, vec_t &v)
  Add a new column by copying data from another vector.
- bool is_column (std::string scol)
  Return true if `scol` is a column in the current table .
- int lookup_column (std::string name, int &ix) const
  Find the index for column named `name` - O(log(C)).
- int copy_column (std::string src, std::string dest)
  Make a new column named `dest` equal to `src` - O(log(C)∗R).
- double ∗ create_array (std::string col) const
  Create (using `new`) a generic array from column `col`.
- int init_column (std::string scol, double val)
  Initialize all values of column named `scol` to `val` - O(log(C)∗R).
- const gsl_vector ∗ get_gsl_vector (std::string name) const
  Get a gsl_vector from column `name` - O(log(C)).
- int add_col_from_table (std::string loc_index, table &source, std::string src_index, std::string src_col, std::string dest_col="")
  Insert a column from a separate table, interpolating it into a new column.

Row maninpulation and data input

- int new_row (size_t n)
  Insert a row before row `n`.
- int copy_row (size_t src, size_t dest)
  Copy the data in row `src` to row `dest`.
- int insert_data (size_t n, size_t nv, double ∗v)
  Insert a row of data before row `n`.
- int insert_data (size_t n, size_t nv, double ∗∗v)
  Insert a row of data before row `n`.
- int line_of_names (std::string newheads)
  Read a new set of names from `newheads`.
- template<class vec_t >
  int line_of_data (size_t nv, const vec_t &v)
  Read a line of data from an array.

Lookup and search methods

- size_t ordered_lookup (std::string col, double val)
  Look for a value in an ordered column.
- size_t lookup (std::string col, double val) const
  Exhaustively search column `col` for the value `val` - O(log(C)∗R).
- double lookup_val (std::string col, double val, std::string col2) const
  Search column `col` for the value `val` and return value in `col2`.
- size_t lookup (int col, double val) const
  Exhaustively search column `col` for the value `val` - O(log(C)∗R).
- size_t mlookup (std::string col, double val, std::vector< double > &results, double threshold=0.0) const
  Exhaustively search column `col` for many occurences of `val` - O(log(C)∗R).

Interpolation, differentiation, and integration, max,

and min

- int set_interp (base_interp_mgr< ovector_const_view > &bi1, base_interp_mgr< ovector_const_subvector > &bi2)
  Set the base interpolation objects.
- double interp (std::string sx, double x0, std::string sy)
  Interpolate `x0` from `sx` into `sy`.
- double interp_const (std::string sx, double x0, std::string sy) const
  Interpolate `x0` from `sx` into `sy`.
- double interp (size_t ix, double x0, size_t iy)
  Interpolate `x0` from `ix` into `iy`.
- double interp_const (size_t ix, double x0, size_t iy) const
  Interpolate `x0` from `ix` into `iy`.
- int deriv (std::string x, std::string y, std::string yp)
  Make a new column `yp` which is the derivative y'(x) - O(log(C)∗R).

- double deriv (std::string sx, double x0, std::string sy)
  The first derivative of the function sy(sx) at sx=x0.
- double deriv_const (std::string sx, double x0, std::string sy) const
  The first derivative of the function sy(sx) at sx=x0.
- double deriv (size_t ix, double x0, size_t iy)
  The first derivative of the function iy(ix) at ix=x0.
- double deriv_const (size_t ix, double x0, size_t iy) const
  The first derivative of the function iy(ix) at ix=x0.
- int deriv2 (std::string x, std::string y, std::string yp)
  Make a new column `yp` which is $y''(x)$ - O(log(C)∗R).
- double deriv2 (std::string sx, double x0, std::string sy)
  The second derivative of the function sy(sx) at sx=x0.
- double deriv2_const (std::string sx, double x0, std::string sy) const
  The second derivative of the function sy(sx) at sx=x0.
- double deriv2 (size_t ix, double x0, size_t iy)
  The second derivative of the function iy(ix) at ix=x0.
- double deriv2_const (size_t ix, double x0, size_t iy) const
  The second derivative of the function iy(ix) at ix=x0.
- double integ (std::string sx, double x1, double x2, std::string sy)
  The integral of the function sy(sx) from sx=x1 to sx=x2.
- double integ_const (std::string sx, double x1, double x2, std::string sy) const
  The integral of the function sy(sx) from sx=x1 to sx=x2.
- double integ (size_t ix, double x1, double x2, size_t iy)
  The integral of the function iy(ix) from ix=x1 to ix=x2.
- double integ_const (size_t ix, double x1, double x2, size_t iy) const
  The integral of the function iy(ix) from ix=x1 to ix=x2.
- int integ (std::string x, std::string y, std::string ynew)
  The integral of the function iy(ix)
- double max (std::string col) const
  Return column maximum. Makes no assumptions about ordering - O(R).
- double min (std::string col) const
  Return column minimum. Makes no assumptions about ordering - O(R).

Subtable method

- table ∗ subtable (std::string list, size_t top, size_t bottom) const
  Make a subtable.

Add space

- int inc_maxlines (size_t llines)
  Manually increase the maximum number of lines.

Delete methods

- virtual int delete_column (std::string scol)
  Delete column named `scol` - O(C).
- int delete_row (std::string scol, double val)
  Delete the row with the entry closest to the value `val` in column `scol` - O(R∗C)
- int delete_row (size_t irow)
  Delete the row of index `irow` - O(R∗C)

Clear methods

- void zero_table ()
  Zero the data entries but keep the column names and nlines fixed.
- void clear_all ()
  Clear everything.
- void clear_table ()
  Clear the table and the column names (but leave constants)
- void clear_data ()
  Remove all of the data by setting the number of lines to zero.
- void clear_constants ()
  CLear all constants.

Sorting methods

- int sort_table (std::string scol)
  Sort the entire table by the column `scol`.
- int sort_column (std::string scol)
  Individually sort the column `scol`.

Summary method

- virtual int summary (std::ostream *out, int ncol=79) const
  Output a summary of the information stored.

Constant manipulation

- virtual int add_constant (std::string name, double val)
  Add a constant, or if the constant already exists, change its value.
- virtual int set_constant (std::string name, double val, bool err_on_notfound=true)
  Add a constant.
- virtual double get_constant (std::string name) const
  Get a constant.
- virtual size_t get_nconsts () const
  Get the number of constants.
- virtual int get_constant (size_t ix, std::string &name, double &val) const
  Get a constant by index.
- virtual int remove_constant (std::string name)
  Remove a constant.

Miscellaneous methods

- int read_generic (std::istream &fin, int verbose=0)
  Clear the current table and read from a generic data file.
- int check_synchro () const
  Return 0 if the tree and list are properly synchronized.
- bool get_owner (std::string name) const
  Get ownership - O(log(C)).

Data Fields

- def_interp_mgr < ovector_const_view, cspline_interp > dim1
  Default interpolation manager.
- def_interp_mgr < ovector_const_subvector, cspline_interp > dim2
  Default interpolation manager.
- bool intp_set
  True if the interpolation type has been set.

Protected Types

- typedef struct table::col_s col
  Column structure for table [protected].
- typedef struct table::sortd_s sortd
  A structure for sorting in table [protected].

Iterator types

- typedef std::map< std::string, col, string_comp >::iterator **aiter**
- typedef std::map< std::string, col, string_comp > ::const_iterator **aciter**
- typedef std::vector< aiter > ::iterator **aviter**

Protected Member Functions

- int reset_list ()
  Set the elements of alist with the appropriate iterators from atree - O(C).
- int make_fp_varname (std::string &s)
  Ensure a variable name does not match a function or contain non-alphanumeric characters.
- int make_unique_name (std::string &col, std::vector< std::string > &cnames)
  Make sure a name is unique.

Column manipulation methods

- aiter get_iterator (std::string lname)
  Return the iterator for a column.
- col * get_col_struct (std::string lname)
  Return the column structure for a column.
- aiter begin ()
  Return the beginning of the column tree.
- aiter end ()
  Return the end of the column tree.

Static Protected Member Functions

- static int sortd_comp (const void *a, const void *b)
  The sorting function.

Protected Attributes

- std::map< std::string, double > constants
  The list of constants.
- ovector empty_col
  An empty vector for get_column()

Actual data

- size_t maxlines
  The size of allocated memory.
- size_t nlines
  The size of presently used memory.
- std::map< std::string, col, string_comp > atree
  The tree of columns.
- std::vector< aiter > alist
  The list of tree iterators.

Interpolation

- sm_interp_vec * si
  The interpolation object.
- base_interp_mgr < ovector_const_view > * bim1
  A pointer to the interpolation manager.
- base_interp_mgr < ovector_const_subvector > * bim2
  A pointer to the subvector interpolation manager.
- std::string intp_colx
  The last x-column interpolated.
- std::string intp_coly
  The last y-column interpolated.

46.330.2 Member Typedef Documentation

46.330.2.1 typedef struct table::col_s table::col [protected]

This struct is used internally by table to organize the columns and need not be instantiated by the casual end-user.

46.330.2.2 typedef struct table::sortd_s table::sortd [protected]

This struct is used internally by table to perform sorting and need not be instantiated by the casual end-user.

46.330.3 Member Function Documentation

46.330.3.1 int table::set (std::string *col*, size_t *row*, double *val*)

This function adds the column `col` if it does not already exist and adds rows using inc_maxlines() and set_nlines() to create at least (row+1) rows if they do not already exist.

46.330.3.2 int table::set (size_t *icol*, size_t *row*, double *val*)

- O(1).

46.330.3.3 int table::set_nlines (size_t *il*)

This function is stingy about increasing the table memory space and will only increase it enough to fit `il` lines. Using it in succession to slowly increase the number of lines in the table is likely to be inefficient compared to set_nlines_auto() in this case.

46.330.3.4 int table::set_nlines_auto (size_t *il*)

This function is like set_nlines(), but doubles the maximum column size if an increase in the maximum size is required instead of simply making enough room for the current number of lines. This function is used internally by set() to ensure that the cost of setting lines in sequence is linear and not quadratic.

46.330.3.5 const ovector_base& table::operator[] (size_t *icol*) const [inline]

This does not do any sort of bounds checking and is quite fast.

Note that several of the methods require reallocation of memory and refereces previously returned by this function will be incorrect.

Unlike set(), this function will not automatically result in an increase in the size of the table if the user attempts to set an element beyond the current column range.

Definition at line 291 of file table.h.

46.330.3.6 ovector_base& table::operator[] (size_t *icol*) [inline]

This does not do any sort of bounds checking and is quite fast.

Note that several of the methods require reallocation of memory and refereces previously returned by this function will be incorrect.

Unlike set(), this function will not automatically result in an increase in the size of the table if the user attempts to set an element beyond the current column range.

Definition at line 308 of file table.h.

46.330.3.7 const ovector_base& table::operator[] (std::string *scol*) const [inline]

No error checking is performed.

Note that several of the methods require reallocation of memory and refereces previously returned by this function will be incorrect.

Unlike set(), this function will not automatically result in an increase in the size of the table if the user attempts to set an element beyond the current column range.

Definition at line 324 of file table.h.

46.330.3.8 ovector_base& table::operator[] (std::string *scol*) [inline]

No error checking is performed.

Note that several of the methods require reallocation of memory and refereces previously returned by this function will be incorrect.

Unlike set(), this function will not automatically result in an increase in the size of the table if the user attempts to set an element beyond the current column range.

Definition at line 341 of file table.h.

46.330.3.9 int table::get_row (std::string *col,* double *val,* ovector & *row*) const

This function searches the entire table for the row which has the entry in column `col` which is closest to the value `val`, and copies that row to the vector `row`.

If the ovector object `row` previously contains any data, it will be lost.

46.330.3.10 int table::get_row (size_t *irow,* ovector & *row*) const

This function returns a copy of row with index `irow`, where `irow` ranges from 0 to `get_nlines()-1`, inclusive.

If the ovector object `row` previously contains any data, it will be lost.

46.330.3.11 int table::hist_update (std::string *ix_col,* std::string *scol,* double *ix,* double *inc* = `1.0`)

Assuming that the histogram bins are given in column named `ix_col`, this updates the histogram found in column `scol` at index location `ix` by incrementing it by an amount given in `inc`.

46.330.3.12 double table::hist_get (std::string *ix_col,* std::string *scol,* double *ix*)

Assuming that the histogram bins are given in column named `ix_col`, this returns the histogram entry found in column `scol` at index location `ix` .

46.330.3.13 int table::hist_set (std::string *ix_col,* std::string *scol,* double *ix,* double *value*)

Assuming that the histogram bins are given in column named `ix_col`, this sets the histogram entry found in column `scol` at index location `ix` to the value given in `value` .

46.330.3.14 int table::new_column (std::string *name*)

Note

This function does not set all the column entries to zero in the case that a new column is added to a table which already contains data.

46.330.3.15 template<class vec_t > int table::new_column (std::string *name,* size_t *sz,* vec_t & *v*) [inline]

This function copies `sz` elements of vector `v` into the table in a new column named `name`. If `sz` is larger than the current number of lines (as given, e.g. in get_nlines()), then only the first part of the vector `v` is copied, up to the current number of lines.

This function calls the error handler if `sz` is zero.

Definition at line 434 of file table.h.

46.330.3.16 bool table::is_column (std::string *scol*)

This function does not call the error handler if the column is not found, but just silently returns false.

46.330.3.17 int table::lookup_column (std::string *name,* int & *ix*) const

If the column is not present, this does not call the error handler, but quietly sets `ix` to zero and returns gsl_enotfound.

46.330.3.18 int table::init_column (std::string *scol*, double *val*)

Note that this does not initialize elements beyond nlines so that if the number of rows is increased afterwards, the new rows will have uninitialized values.

46.330.3.19 int table::add_col_from_table (std::string *loc_index*, table & *source*, std::string *src_index*, std::string *src_col*, std::string *dest_col* = "")

Given a pair of columns (src_index, src_col) in a separate table (source), this creates a new column in the present table named src_col which interpolates loc_index into src_index. The interpolation objects from the source table will be used. If there is already a column in the present table named src_col, then this will fail.

If there is an error in the interpolation for any particular row, then the value of src_col in that row will be set to zero.

46.330.3.20 int table::new_row (size_t *n*)

Acceptable values for n are between 0 and get_nlines() inclusive, with the maximum value denoting the addition of a row after the last row presently in the table.

46.330.3.21 int table::line_of_names (std::string *newheads*)

This function reads a set of white-space delimited column names from the string newheads, and creates a new column for each name which is specified.

For example

```
table t;
t.line_of_names("position velocity acceleration");
```

will create three new columns with the names ""position", "velocity", and "acceleration".

46.330.3.22 size_t table::ordered_lookup (std::string *col*, double *val*)

O(log(C)∗log(R))

This uses the function search_vec::ordered_lookup(), which offers caching and assumes the vector is monotonic. If you don't have monotonic data, you can still use the table::lookup() function, which is more general.

46.330.3.23 double table::interp (std::string *sx*, double *x0*, std::string *sy*)

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.24 double table::interp_const (std::string *sx*, double *x0*, std::string *sy*) const

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.25 double table::interp (size_t *ix*, double *x0*, size_t *iy*)

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.26 double table::interp_const (size_t *ix*, double *x0*, size_t *iy*) const

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.27 double table::deriv (std::string *sx*, double *x0*, std::string *sy*)

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.28 double table::deriv_const (std::string *sx*, double *x0*, std::string *sy*) const

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.29 double table::deriv (size_t *ix*, double *x0*, size_t *iy*)

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.30 double table::deriv_const (size_t *ix*, double *x0*, size_t *iy*) const

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.31 double table::deriv2 (std::string *sx*, double *x0*, std::string *sy*)

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.32 double table::deriv2_const (std::string *sx*, double *x0*, std::string *sy*) const

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.33 double table::deriv2 (size_t *ix*, double *x0*, size_t *iy*)

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.34 double table::deriv2_const (size_t *ix*, double *x0*, size_t *iy*) const

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.35 double table::integ (std::string *sx*, double *x1*, double *x2*, std::string *sy*)

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.36 double table::integ_const (std::string *sx*, double *x1*, double *x2*, std::string *sy*) const

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

46.330.3.37 double table::integ (size_t *ix*, double *x1*, double *x2*, size_t *iy*)

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.38 double table::integ_const (size_t *ix*, double *x1*, double *x2*, size_t *iy*) const

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.39 int table::integ (std::string *x*, std::string *y*, std::string *ynew*)

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

46.330.3.40 table∗ table::subtable (std::string *list*, size_t *top*, size_t *bottom*) const

Uses the columns specified in `list` from the row `top` to the row of index `bottom` to generate a new table which is a copy of part of the original.

46.330.3.41 virtual int table::delete_column (std::string *scol*) [virtual]

This is slow because the iterators in alist are mangled and we have to call reset_list() to get them back.

Reimplemented in table_units.

46.330.3.42 void table::clear_data ( ) [inline]

This leaves the column names intact and does not remove the constants.

Definition at line 868 of file table.h.

46.330.3.43 int table::sort_table (std::string *scol*)

Note

This function works by allocating space for an entirely new chunk of memory for the data in the table.

Todo Use vector_sort() rather than qsort().

46.330.3.44 int table::sort_column (std::string *scol*)

Todo Use vector_sort() rather than qsort().

46.330.3.45 virtual int table::summary (std::ostream * *out*, int *ncol* = 79) const [virtual]

Outputs the number of constants, the number of columns, a list of the column names, and the number of lines of data.

Reimplemented in table_units.

46.330.3.46 int table::reset_list ( ) [protected]

Generally, the end-user shouldn't need this method. It is only used in delete_column() to rearrange the list when a column is deleted from the tree.

The documentation for this class was generated from the following file:

- table.h

46.331 table3d Class Reference

A data structure containing many slices of two-dimensional data points defined on a grid.

`#include <table3d.h>`

46.331.1 Detailed Description

Idea for Future Improve interpolation and derivative caching

Idea for Future Make a 'const' version of the interpolation functions

Idea for Future Should there be a clear_grid() function separate from clear_data() and clear_table()?

Definition at line 53 of file table3d.h.

Public Member Functions

- table3d ()
  Create a new 3D table .
- bool **is_size_set** () const
- bool **is_xy_set** () const
- virtual const char * type ()
  Return the type, "table3d".

Initialization

- template<class vec2_t >
  int set_xy (std::string x_name, size_t nx, const vec2_t &x, std::string y_name, size_t ny, const vec2_t &y)
  Initialize the x-y grid.
- int set_size (size_t nx, size_t ny)
  Initialize table size.

On-grid get and set methods

- int set (size_t ix, size_t iy, std::string name, double val)
  Set element in slice `name` at location `ix,iy` to value `val`.
- int set (size_t ix, size_t iy, size_t z, double val)
  Set element in slice of index `z` at location `ix,iy` to value `val` .
- double & get (size_t ix, size_t iy, std::string name)
  Get element in slice `name` at location `ix,iy`
- const double & get (size_t ix, size_t iy, std::string name) const
  Get element in slice `name` at location `ix,iy` (const version)
- double & get (size_t ix, size_t iy, size_t z)
  Get element in slice of index `z` at location `ix,iy`
- const double & get (size_t ix, size_t iy, size_t z) const
  Get element in slice of index `z` at location `ix,iy` (const version)

Off-grid get and set methods

These methods return the value of a slice on the grid point nearest to a user-specified location. For interpolation into a point off the grid, use table3d::interp().

- int set_val (double x, double y, std::string name, double val)
  Set element in slice `name` at the nearest location to `x,y` to value `val`.
- int set_val (double x, double y, size_t z, double val)
  Set element in slice of index `z` at the nearest location to `x,y` to value `val`.
- double & get_val (double x, double y, std::string name)
  Get element in slice `name` at location closest to `x,y`
- const double & get_val (double x, double y, std::string name) const
  Get element in slice `name` at location closest to `x,y`
- double & get_val (double x, double y, size_t z)
  Get element in slice of index `z` at location closest to `x,y`
- const double & get_val (double x, double y, size_t z) const
  Get element in slice of index `z` at location closest to `x,y`
- template<class vec_t >
  int set_slices (double x, double y, size_t nv, vec_t &vals)
  Set elements in the first `nv` slices at the nearest location to `x,y` to value `val`.
- template<class vec_t >
  int get_slices (double x, double y, size_t nv, vec_t &v)
  Get the data for every slice at the nearest location to `x,y`

Off-grid get and set methods returning nearest point

- int set_val_ret (double &x, double &y, std::string name, double val)
  Set element in slice `name` at the nearest location to `x,y` to value `val`.
- int set_val_ret (double &x, double &y, size_t z, double val)
  Set element in slice of index `z` at the nearest location to `x,y` to value `val`.
- double & get_val_ret (double &x, double &y, std::string name)
  Get element in slice `name` at location closest to `x,y`, and also return the corresponding values of `x` and `y`.
- const double & get_val_ret (double &x, double &y, std::string name) const
  Get element in slice `name` at location closest to `x,y`, and also return the corresponding values of `x` and `y`.
- double & get_val_ret (double &x, double &y, size_t z)
  Get element in slice of index `z` at location closest to `x,y`, and also return the corresponding values of `x` and `y`.
- const double & get_val_ret (double &x, double &y, size_t z) const
  Get element in slice of index `z` at location closest to `x,y`, and also return the corresponding values of `x` and `y`.
- template<class vec_t >
  int set_slices_ret (double &x, double &y, size_t nv, vec_t &vals)
  Set elements in the first `nv` slices at the nearest location to `x,y` to values `vals`.
- template<class vec_t >
  int get_slices_ret (double &x, double &y, size_t nv, vec_t &vals)
  Get elements in the first `nv` slices at the nearest location to `x,y` to value `val`.

Grid information get and set methods

- int set_grid_x (size_t ix, double val)
  Set x grid point at index `ix`.
- int set_grid_y (size_t iy, double val)
  Set y grid point at index `iy`.
- double get_grid_x (size_t ix)
  Get x grid point at index `ix`.
- double get_grid_y (size_t iy)
  Get y grid point at index `iy`.
- std::string get_x_name () const
  Get the name of the x grid variable.
- std::string get_y_name () const
  Get the name of the y grid variable.
- int set_x_name (std::string name)
  Set the name of the x grid variable.
- int set_y_name (std::string name)
  Set the name of the y grid variable.
- const ovector & get_x_data () const
  Get a const reference to the full x grid.
- const ovector & get_y_data () const
  Get a const reference to the full y grid.

Size get methods

- int get_size (size_t &nx, size_t &ny) const
  Get the size of the slices.
- int get_nx () const
  Get the x size.
- int get_ny () const
  Get the y size.
- int get_nslices () const
  Get the number of slices.

Slice manipulation

- int line_of_names (std::string names)
  Create a set of new slices specified in the string `names`.
- std::string get_slice_name (size_t z) const
  Returns the name of slice with index `z`.
- int new_slice (std::string name)
  Add a new slice.
- int set_slice_all (std::string name, double val)
  Set all of the values in slice `name` to `val`.
- int lookup_slice (std::string name, size_t &ix) const
  Find the index for column named `name`.
- bool is_slice (std::string name, int &ix)
  Return true if slice is already present.
- int rename_slice (std::string olds, std::string news)
  Rename slice named `olds` to `news`.
- int copy_slice (std::string src, std::string dest)
  Make a new slice named `dest` which is a copy of the slice with name given in `src`.
- int init_slice (std::string scol, double val)
  Initialize all values of slice named `scol` to `val`.
- const omatrix & get_slice (std::string scol) const
  Return a constant reference to a slice.
- const omatrix & get_slice (size_t iz) const
  Return a constant reference to a slice.
- omatrix & get_slice (std::string scol)
  Return a constant reference to a slice.
- omatrix & get_slice (size_t iz)
  Return a constant reference to a slice.
- const std::vector< omatrix > & get_data ()
  Return a constant reference to all the slice data.

Lookup and search methods

- int lookup_x (double val, size_t &ix) const
  Look for a value in the x grid.
- int lookup_y (double val, size_t &iy) const
  Look for a value in the y grid.
- int lookup (double val, std::string slice, size_t &ix, size_t &iy) const
  Look for a value in a specified slice.

Interpolation, differentiation, and integration

- int set_interp (base_interp_mgr< ovector_const_view > &b1, base_interp_mgr< ovector_const_subvector > &b2)
  Specify the base interpolation objects to use.
- double interp (double x, double y, std::string name)
  Interpolate `x` and `y` in slice named `name`.
- double deriv_x (double x, double y, std::string name)
  Interpolate the derivative of the data with respect to the x grid at point `x` and `y` in slice named `name`.
- double deriv_y (double x, double y, std::string name)
  Interpolate the derivative of the data with respect to the y grid at point `x` and `y` in slice named `name`.
- double deriv_xy (double x, double y, std::string name)
  Interpolate the mixed second derivative of the data at point `x` and `y` in slice named `name`.
- double integ_x (double x1, double x2, double y, std::string name)
  Interpolate the integral of the data respect to the x grid.
- double integ_y (double x, double y1, double y2, std::string name)
  Interpolate the integral of the data respect to the y grid.
- template<class vec_t >
  int interp_slices (double x, double y, size_t nv, vec_t &v)
  Fill a vector of interpolated values from each slice at the point `x,y`

Extract 2-dimensional tables

- int extract_x (double x, table &t)
  Extract a table at a fixed x grid point.
- int extract_y (double y, table &t)
  Extract a table at a fixed y grid point.

Clear methods

- int zero_table ()
  Zero the data entries but keep the slice names and grid.
- void clear_table ()
  Clear the table and the slice names.
- void clear_data ()
  Remove all of the data by setting the number of lines to zero.

Summary method

- int summary (std::ostream *out, int ncol=79) const
  Output a summary of the information stored.

Histogram-like functions

- int hist_update (double ix, double iy, std::string scol, double inc=1.0)
  Update a histogram entry.
- double hist_get (double ix, double iy, std::string scol) const
  Get a histogram entry.
- int hist_set (double ix, double iy, std::string scol, double value)
  Set a histogram entry.

Manipulating constants

- virtual int add_constant (std::string name, double val)
  Add a constant.
- virtual int remove_constant (std::string name)
  Remove a constant.

- virtual int set_constant (std::string name, double val)
  Add a constant.
- virtual double get_constant (std::string name)
  Get a constant.
- virtual int get_constant (size_t ix, std::string &name, double &val) const
  Get a constant by index.
- virtual size_t get_nconsts () const
  Get the number of constants.

Data Fields

Default interpolation objects

- def_interp_mgr < ovector_const_view, cspline_interp > **dim1**
- def_interp_mgr < ovector_const_subvector, cspline_interp > **dim2**

Protected Types

Iterator types

- typedef std::map< std::string, size_t, string_comp > ::iterator **map_iter**
- typedef std::map< std::string, size_t, string_comp > ::const_iterator **map_const_iter**

Protected Member Functions

Tree iterator boundaries

- map_iter begin ()
  Return the beginning of the slice tree.
- map_iter end ()
  Return the end of the slice tree.
- map_const_iter const_begin () const
  Return the beginning of the slice tree.
- map_const_iter const_end () const
  Return the end of the slice tree.

Protected Attributes

Interpolation data

- base_interp_mgr < ovector_const_view > * bimp1
  The base interpolation object.
- base_interp_mgr < ovector_const_subvector > * bimp2
  The subvector base interpolation object.
- sm_interp_vec ** si
  The array of sm_interp_vec pointers.
- omatrix_col ** aci
  Matrices for interpolation.

Data storage

- std::map< std::string, double > constants
  The list of constants.
- size_t numx
  The size of the x grid.
- size_t numy
  The size of the y grid.
- std::map< std::string, size_t, string_comp > tree
  A tree connecting column names to list indexes.

- std::string xname
  The name for the x grid.
- std::string yname
  The name for the y grid.
- std::vector< omatrix > list
  The pointers to the matrices.
- ovector xval
  The x grid.
- ovector yval
  The y grid.
- bool xy_set
  True if the grid has been set.
- bool size_set
  True if the size of the grid has been set.
- bool has_slice
  True if the table has at least one slice.

46.331.2 Member Function Documentation

46.331.2.1 template<class vec2_t > int table3d::set_xy (std::string *x_name*, size_t *nx*, const vec2_t & *x*, std::string *y_name*, size_t *ny*, const vec2_t & *y*) [inline]

This function will not allow you to redefine the grid when there is data in the table if a grid of a different size was already set from a previous call to either set_xy() or set_size(). However, you may freely redefine the grid after a call to clear_data() or clear_table(). You may change individual grid points at any time with set_grid_x() and set_grid_y().

Definition at line 76 of file table3d.h.

46.331.2.2 int table3d::set_size (size_t *nx*, size_t *ny*)

This function will not allow you to resize the table if it already has data or if the size has already been set with the set_xy() function, unless you clear the data with clear_data() or the table with clear_table() first.

46.331.2.3 int table3d::rename_slice (std::string *olds*, std::string *news*)

This is slow since we have to delete the column and re-insert it. This process in turn mangles all of the iterators in the list.

46.331.2.4 const std::vector<omatrix>& table3d::get_data ( )

46.331.2.5 int table3d::extract_x (double *x*, table & *t*)

Note

All of the information previously stored in `t` will be lost.

46.331.2.6 int table3d::extract_y (double *y*, table & *t*)

Note

All of the information previously stored in `t` will be lost.

46.331.2.7 void table3d::clear_data ( )

This leaves the column names intact and does not remove the constants.

46.331.2.8 int table3d::summary (std::ostream * *out*, int *ncol* = 79) const

Outputs the number of constants, the grid information, and a list of the slice names

46.331.2.9 int table3d::hist_update (double *ix*, double *iy*, std::string *scol*, double *inc* = 1.0)

Assuming that the histogram bins are given in column named `ix_col`, this updates the histogram found in column `scol` at index location `ix` by incrementing it by an amount given in `inc`.

46.331.2.10 double table3d::hist_get (double *ix*, double *iy*, std::string *scol*) const

Assuming that the histogram bins are given in column named `ix_col`, this returns the histogram entry found in column `scol` at index location `ix` .

46.331.2.11 int table3d::hist_set (double *ix*, double *iy*, std::string *scol*, double *value*)

Assuming that the histogram bins are given in column named `ix_col`, this sets the histogram entry found in column `scol` at index location `ix` to the value given in `value` .

The documentation for this class was generated from the following file:

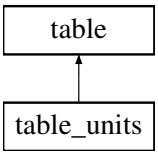
- table3d.h

46.332 table_units Class Reference

Data table class with units.

`#include <table_units.h>`

Inheritance diagram for table_units:



46.332.1 Detailed Description

Idea for Future Make table methods virtual? (not necessary yet since delete_column() isn't referred to internally)

Definition at line 38 of file table_units.h.

Public Member Functions

- table_units (int cmaxlines=0)
  Create a new table_units with space for nlines<=cmaxlines.
- std::string get_unit (std::string scol) const
  Get the unit for column `scol`.
- int remove_unit (std::string scol)
  Remove the unit for column `scol`.
- int set_unit (std::string scol, std::string unit)
  Set the unit for column `scol` to `unit`.
- int convert_to_unit (std::string scol, std::string unit, bool err_on_fail=true)
  Convert the units of column `scol` to `unit`.
- double get_conv (std::string old_unit, std::string new_unit)
  Get the conversion factor from `old_unit` to `new_unit`.
- virtual int delete_column (std::string scol)
  Delete column named `scol`.
- virtual const char * type ()

Return the type, "table_units".

- virtual int summary (std::ostream *out, int ncol=79) const
  Output a summary of the information stored.
- int set_convert (convert_units &c)
  Set the convert units object.
- int show_units ()
  Show the unit cache as given by convert_units::print_cache()
- size_t get_nunits ()
  Return the number of columns with units.

Copy constructors

- **table_units** (const table_units &t)
- **table_units** (const table &t)
- table_units & **operator=** (const table_units &t)
- table_units & operator= (const table &t)
  Copy constructor.

Data Fields

- convert_units def_cu
  The default object for unit conversions.

Protected Types

Unit map iterator types

- typedef std::map< std::string, std::string, string_comp > ::iterator **uiter**
- typedef std::map< std::string, std::string, string_comp > ::const_iterator **uciter**

Protected Attributes

- convert_units * cup
  The pointer to the convert units object.
- std::map< std::string, std::string, string_comp > utree
  Unit map.

The documentation for this class was generated from the following file:

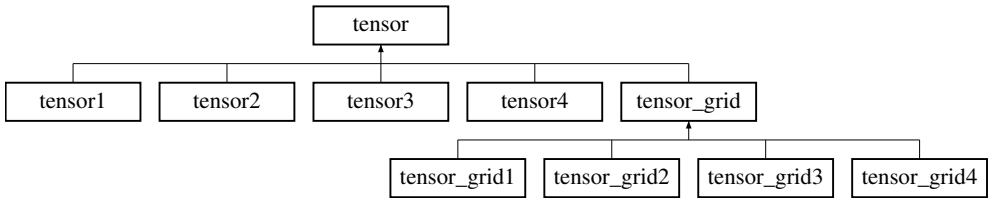
- table_units.h

46.333 tensor Class Reference

Tensor class with arbitrary dimensions.

`#include <tensor.h>`

Inheritance diagram for tensor:



46.333.1 Detailed Description

The elements of a tensor are typically specified as a list of `size_t` numbers with length equal to the tensor rank. For a rank-4 tensor named `t`, the element `t[1][2][0][3]` can be obtained with something similar to

```
size_t ix[4]={1,2,0,3};
double x=t.get(ix);
```

Empty tensors have zero rank.

Slices of tensors are subsets obtained from fixing the index of several dimensions, while letting other dimensions vary. For a slice with one dimension not fixed, see vector_slice(). For a slice with two dimensions not fixed, see matrix_slice().

For I/O with tensors, see hdf_io.h .

Idea for Future Could implement arithmetic operators + and - and some different products.

Idea for Future Implement copies to and from vector and matrices

Idea for Future Implement tensor contractions, i.e. tensor = tensor $*$ tensor

Idea for Future Consider making a template type to replace double, e.g. `value_t`.

Idea for Future Could be interesting to write an iterator for this class.

Definition at line 93 of file tensor.h.

Public Member Functions

- tensor ()
  Create an empty tensor with zero rank.
- template<class size_vec_t >
  tensor (size_t rank, const size_vec_t &dim)
  Create a tensor of rank `rank` with sizes given in `dim`.

Set functions

- template<class size_vec_t >
  void set (const size_vec_t &index, double val)
  Set the element indexed by `index` to value `val`.
- void set_all (double x)
  Set all elements in a tensor to some fixed value.

Get functions

- template<class size_vec_t >
  double & get (const size_vec_t &index)
  Get the element indexed by `index`.
- template<class size_vec_t >
  double const & get (const size_vec_t &index) const
  Get a const reference to the element indexed by `index`.

Slice functions

- template<class size_vec_t >
  ovector_array_stride vector_slice (size_t ix, const size_vec_t &index)
  Fix all but one index to create a vector.
- template<class size_vec_t >
  omatrix_array matrix_slice (size_t ix, const size_vec_t &index)
  Fix all but two indices to create a matrix.

Memory allocation

- template<class size_vec_t >
  void allocate (size_t rank, const size_vec_t &dim)
  Allocate space for a tensor of rank `rank` with sizes given in `dim`.
- void free ()
  Free allocated space (also sets rank to zero)

Size functions

- size_t get_rank () const
  Return the rank of the tensor.
- size_t get_size (size_t i) const
  Returns the size of the ith index.
- const uvector_size_t & get_size_arr () const
  Return the full array of sizes.
- const uvector & get_data () const
  Return the full array of sizes.
- size_t total_size () const
  Returns the size of the tensor (the product of the sizes over every index)

Index manipulation

- template<class size_vec_t >
  size_t pack_indices (const size_vec_t &index)
  Pack the indices into a single array index.
- template<class size_vec_t >
  void unpack_indices (size_t ix, size_vec_t &index)
  Unpack the single array index into indices.

Protected Attributes

- uvector data
  The data.
- uvector_size_t size
  A rank-sized array of the sizes of each dimension.
- size_t rk
  Rank.

46.333.2 Constructor & Destructor Documentation

46.333.2.1 template<class size_vec_t > tensor::tensor (size_t *rank,* const size_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`. If the user requests any of the sizes to be zero, this constructor will call the error handler, create an empty tensor, and will allocate no memory.

Definition at line 125 of file tensor.h.

46.333.3 Member Function Documentation

46.333.3.1 template<class size_vec_t > ovector_array_stride tensor::vector_slice (size_t *ix,* const size_vec_t & *index*) [inline]

This fixes all of the indices to the values given in `index` except for the index number `ix`, and returns the corresponding vector, whose length is equal to the size of the tensor in that index. The value `index[ix]` is ignored.

For example, for a rank 3 tensor allocated with

```
tensor t;
size_t dim[3]={3,4,5};
t.allocate(3,dim);
```

the following code

```
size_t index[3]={1,0,3};
ovector_view v=t.vector_slice(1,index);
```

Gives a vector `v` of length 4 which refers to the values `t(1,0,3)`, `t(1,1,3)`, `t(1,2,3)`, and `t(1,3,3)`.

Definition at line 275 of file tensor.h.

46.333.3.2 template<class size_vec_t> omatrix_array tensor::matrix_slice (size_t *ix*, const size_vec_t & *index*) [inline]

This fixes all of the indices to the values given in `index` except for the index number `ix` and the last index, and returns the corresponding matrix, whose size is equal to the size of the tensor in the two indices which are not fixed.

Since the last index is always fixed, the parameter `ix` is limited to be $\in [0, \text{rank}-2]$.

Definition at line 303 of file tensor.h.

46.333.3.3 template<class size_vec_t> void tensor::allocate (size_t *rank*, const size_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`.

If memory was previously allocated, it will be freed before the new allocation.

If the user requests any of the sizes to be zero, this function will call the error handler and will allocate no memory. If memory was previously allocated, the tensor is left unmodified and no deallocation is performed.

Reimplemented in tensor_grid.

Definition at line 348 of file tensor.h.

The documentation for this class was generated from the following file:

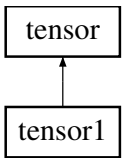
- tensor.h

46.334 tensor1 Class Reference

Rank 1 tensor.

`#include <tensor.h>`

Inheritance diagram for tensor1:



46.334.1 Detailed Description

Definition at line 924 of file tensor.h.

Public Member Functions

- tensor1 ()
  Create an empty tensor.
- tensor1 (size_t sz)
  Create a rank 1 tensory of size `sz`.

- double & get (size_t ix)
  Get the element indexed by `ix`.
- const double & get (size_t ix) const
  Get the element indexed by `ix`.
- void set (size_t index, double val)
  Set the element indexed by `index` to value `val`.
- void set (size_t *index, double val)
  Set the element indexed by `index` to value `val`.
- double & operator[ ] (size_t ix)
  Get an element using array-like indexing.
- double & operator() (size_t ix)
  Get an element using operator()

46.334.2 Member Function Documentation

46.334.2.1 void tensor1::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor.)

Definition at line 952 of file tensor.h.

The documentation for this class was generated from the following file:

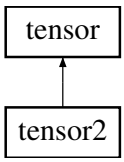
- tensor.h

46.335 tensor2 Class Reference

Rank 2 tensor.

`#include <tensor.h>`

Inheritance diagram for tensor2:



46.335.1 Detailed Description

Definition at line 1010 of file tensor.h.

Public Member Functions

- tensor2 ()
  Create an empty tensor.
- tensor2 (size_t sz, size_t sz2)
  Create a rank 2 tensor of size (sz,sz2)
- double & get (size_t ix1, size_t ix2)
  Get the element indexed by (ix1,ix2)
- const double & get (size_t ix1, size_t ix2) const
  Get the element indexed by (ix1,ix2)
- void set (size_t ix1, size_t ix2, double val)
  Set the element indexed by (ix1,ix2) to value `val`.
- void set (size_t *index, double val)

Set the element indexed by `index` *to value* `val`.

- double & operator() (size_t ix, size_t iy)
  Get the element indexed by (ix1,ix2)

46.335.2 Member Function Documentation

46.335.2.1 void tensor2::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor.)

Definition at line 1050 of file tensor.h.

The documentation for this class was generated from the following file:

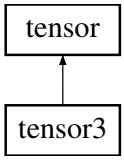
- tensor.h

46.336 tensor3 Class Reference

Rank 3 tensor.

`#include <tensor.h>`

Inheritance diagram for tensor3:



46.336.1 Detailed Description

Definition at line 1111 of file tensor.h.

Public Member Functions

- tensor3 ()
  Create an empty tensor.
- tensor3 (size_t sz, size_t sz2, size_t sz3)
  Create a rank 3 tensor of size (sz,sz2,sz3)
- double & get (size_t ix1, size_t ix2, size_t ix3)
  Get the element indexed by (ix1,ix2,ix3)
- const double & get (size_t ix1, size_t ix2, size_t ix3) const
  Get the element indexed by (ix1,ix2,ix3)
- void set (size_t ix1, size_t ix2, size_t ix3, double val)
  Set the element indexed by (ix1,ix2,ix3) to value `val`.
- void set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`.

46.336.2 Member Function Documentation

46.336.2.1 void tensor3::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor.)

Definition at line 1152 of file tensor.h.

The documentation for this class was generated from the following file:

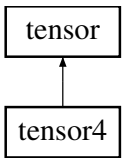
- tensor.h

46.337 tensor4 Class Reference

Rank 4 tensor.

`#include <tensor.h>`

Inheritance diagram for tensor4:



46.337.1 Detailed Description

Definition at line 1210 of file tensor.h.

Public Member Functions

- tensor4 ()
  Create an empty tensor.
- tensor4 (size_t sz, size_t sz2, size_t sz3, size_t sz4)
  Create a rank 4 tensor of size (sz,sz2,sz3,sz4)
- double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4)
  Get the element indexed by (ix1,ix2,ix3,ix4)
- const double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4) const
  Get the element indexed by (ix1,ix2,ix3,ix4)
- void set (size_t ix1, size_t ix2, size_t ix3, size_t ix4, double val)
  Set the element indexed by (ix1,ix2,ix3,ix4) to value `val`.
- void set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`.

46.337.2 Member Function Documentation

46.337.2.1 void tensor4::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor.)

Definition at line 1256 of file tensor.h.

The documentation for this class was generated from the following file:

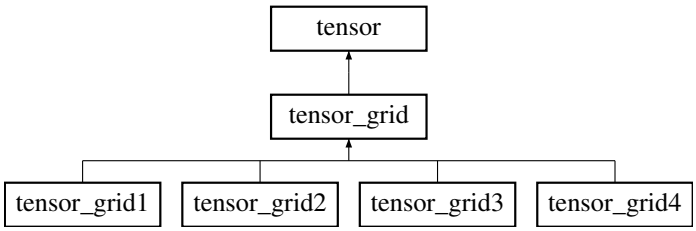
- tensor.h

46.338 tensor_grid Class Reference

Tensor class with arbitrary dimensions with a grid.

`#include <tensor.h>`

Inheritance diagram for tensor_grid:



46.338.1 Detailed Description

This tensor class allows one to assign the indexes to numerical scales, so that n-dimensional interpolation can be performed. To set the grid, use set_grid() and then interpolation can be done using interpolate().

By convention, member functions ending in the _val suffix return the closest grid-point to some user-specified values.

Idea for Future Only allocate space for grid if it is set

Idea for Future Could implement arithmetic operators + and - and some different products.

Idea for Future Consider creating a set_grid() function which takes grids from an object like hist_grid. Maybe make a constructor for a tensor_grid object which just takes as input a set of grids?

Definition at line 501 of file tensor.h.

Public Member Functions

- tensor_grid ()
  Create an empty tensor with zero rank.
- template<class size_vec_t >
  tensor_grid (size_t rank, const size_vec_t &dim)
  Create a tensor of rank `rank` with sizes given in `dim`.

Set functions

- template<class vec_t >
  int set_val (const vec_t &grdp, double val)
  Set the element closest to grid point `grdp` to value `val`.
- template<class vec_t , class vec2_t >
  int set_val (const vec_t &grdp, double val, vec2_t &closest)
  Set the element closest to grid point `grdp` to value `val`.

Get functions

- template<class vec_t >
  double get_val (const vec_t &grdp)
  Get the element closest to grid point `grdp`.
- template<class vec_t , class vec2_t >
  double get_val (const vec_t &grdp, const vec2_t &closest)
  Get the element closest to grid point `grdp` to value `val`.

Memory allocation

- template<class size_vec_t >
  int allocate (size_t rank, const size_vec_t &dim)
  Allocate space for a tensor of rank `rank` with sizes given in `dim`.
- int free ()
  Free allocated space (also sets rank to zero)

Grid manipulation

- bool is_grid_set () const
  Return true if the grid has been set.
- template<class vec_t >
  int set_grid (const vec_t &grid_vec)
  Set the grid.
- size_t lookup_grid (size_t i, double val)
  Lookup index for grid closest to `val`.
- double get_grid (size_t i, size_t j) const
  Lookup jth value on the ith grid.
- template<class vec_t , class size_vec_t >
  int lookup_grid_vec (const vec_t &vals, const size_vec_t &indices) const
  Lookup indices for grid closest point to `vals`.
- size_t lookup_grid_val (size_t i, double &val, double &val2)
  Lookup index for grid closest to `val`, returning the grid point.

Protected Attributes

- uvector grid
  A rank-sized set of arrays for the grid points.
- bool grid_set
  If true, the grid has been set by the user.
- base_interp_mgr< uvector_base > * bim1
  The interpolation manager.
- base_interp_mgr < uvector_const_subvector > * bim2
  The subvector interpolation manager.

Interpolation

- def_interp_mgr< uvector_base, linear_interp > **dim1**
- def_interp_mgr < uvector_const_subvector, linear_interp > **dim2**
- int set_interp (base_interp_mgr< uvector_base > &bi1, base_interp_mgr< uvector_const_subvector > &bi2)
  Set interpolation managers.
- double interpolate (double *vals)
  Interpolate values `vals` into the tensor, returning the result.

46.338.2 Constructor & Destructor Documentation

46.338.2.1 template<class size_vec_t > tensor_grid::tensor_grid (size_t *rank*, const size_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`. If the user requests any of the sizes to be zero, this constructor will call the error handler, create an empty tensor, and will allocate no memory.

Definition at line 538 of file tensor.h.

46.338.3 Member Function Documentation

46.338.3.1 template<class vec_t , class vec2_t > int tensor_grid::set_val (const vec_t & *grdp*, double *val*, vec2_t & *closest*) [inline]

The parameters `closest` and `grdp` may be identical, allowing one to update a vector `grdp` with the closest grid point.

Definition at line 584 of file tensor.h.

46.338.3.2 template<class size_vec_t > int tensor_grid::allocate (size_t *rank*, const size_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`.

If memory was previously allocated, it will be freed before the new allocation and previously specified grid data will be lost.

If the user requests any of the sizes to be zero, this function will call the error handler and will allocate no memory. If memory was previously allocated, the tensor is left unmodified and no deallocation is performed.

Reimplemented from tensor.

Definition at line 667 of file tensor.h.

46.338.3.3 template<class vec_t > int tensor_grid::set_grid (const vec_t & *grid_vec*) [inline]

The grid must be specified for all of the dimensions at once. Denote $(\text{size})_0$ as the size of the first dimension, $(\text{size})_1$ as the size of the second dimesion, and so on. Then the first $(\text{size})_0$ entries in `grid` must be the grid for the first dimension, the next $(\text{size})_1$ entries must be the grid for the second dimension, and so on. Thus `grid` must be a vector of size

$$\sum_{i=0}^{\text{rank}} (\text{size})_i$$

Note that the grid is copied so the function argument may be destroyed by the user after calling set_grid() without affecting the tensor grid.

Idea for Future Define a more generic interface for matrix types

Definition at line 725 of file tensor.h.

46.338.3.4 template<class vec_t , class size_vec_t > int tensor_grid::lookup_grid_vec (const vec_t & *vals*, const size_vec_t & *indices*) const [inline]

The values in `vals` are not modified by this function.

Definition at line 779 of file tensor.h.

46.338.3.5 double tensor_grid::interpolate (double * *vals*) [inline]

This is a quick and dirty implementation of n-dimensional interpolation by recursive application of the 1-dimensional routine from smart_interp_vec, using the base interpolation object specified in the template parameter `base_interp_t`. This will be slow for sufficiently large data sets.

Idea for Future Maybe make this a template as well?

Idea for Future It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor. However, this should probably be superceded by a more generic alternative which avoids explicit use of the 1-d interpolation types.

Definition at line 839 of file tensor.h.

The documentation for this class was generated from the following file:

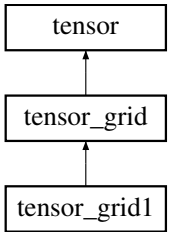
- tensor.h

46.339 tensor_grid1 Class Reference

Rank 1 tensor with a grid.

`#include <tensor.h>`

Inheritance diagram for tensor_grid1:



46.339.1 Detailed Description

Definition at line 965 of file tensor.h.

Public Member Functions

- tensor_grid1 ()
  Create an empty tensor.
- tensor_grid1 (size_t sz)
  Create a rank 2 tensor of size (sz,sz2,sz3)
- double & get (size_t ix1)
  Get the element indexed by (ix1)
- const double & get (size_t ix1) const
  Get the element indexed by (ix1)
- void set (size_t ix1, double val)
  Set the element indexed by (ix1) to value `val`.
- double interp (double x)
  Interpolate `x` *and return the results.*

The documentation for this class was generated from the following file:

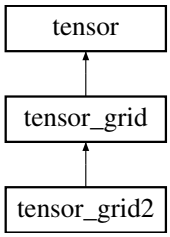
- tensor.h

46.340 tensor_grid2 Class Reference

Rank 2 tensor with a grid.

`#include <tensor.h>`

Inheritance diagram for tensor_grid2:



46.340.1 Detailed Description

Definition at line 1062 of file tensor.h.

Public Member Functions

- tensor_grid2 ()
  Create an empty tensor.
- tensor_grid2 (size_t sz, size_t sz2)
  Create a rank 2 tensor of size (sz,sz2)
- double & get (size_t ix1, size_t ix2)
  Get the element indexed by (ix1,ix2)
- const double & get (size_t ix1, size_t ix2) const
  Get the element indexed by (ix1,ix2)
- void set (size_t ix1, size_t ix2, double val)
  Set the element indexed by (ix1,ix2) to value `val`.
- double interp (double x, double y)
  Interpolate (x,y) and return the results.

The documentation for this class was generated from the following file:

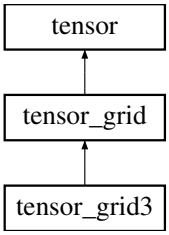
- tensor.h

46.341 tensor_grid3 Class Reference

Rank 3 tensor with a grid.

`#include <tensor.h>`

Inheritance diagram for tensor_grid3:



46.341.1 Detailed Description

Definition at line 1160 of file tensor.h.

Public Member Functions

- tensor_grid3 ()
  Create an empty tensor.
- tensor_grid3 (size_t sz, size_t sz2, size_t sz3)
  Create a rank 3 tensor of size (sz,sz2,sz3)
- double & get (size_t ix1, size_t ix2, size_t ix3)
  Get the element indexed by (ix1,ix2,ix3)
- const double & get (size_t ix1, size_t ix2, size_t ix3) const
  Get the element indexed by (ix1,ix2,ix3)
- void set (size_t ix1, size_t ix2, size_t ix3, double val)
  Set the element indexed by (ix1,ix2,ix3) to value `val`.
- double interp (double x, double y, double z)
  Interpolate (x,y,z) and return the results.

The documentation for this class was generated from the following file:

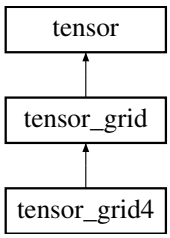
- tensor.h

46.342 tensor_grid4 Class Reference

Rank 4 tensor with a grid.

`#include <tensor.h>`

Inheritance diagram for tensor_grid4:



46.342.1 Detailed Description

Definition at line 1264 of file tensor.h.

Public Member Functions

- tensor_grid4 ()
  Create an empty tensor.
- tensor_grid4 (size_t sz, size_t sz2, size_t sz3, size_t sz4)
  Create a rank 4 tensor of size (sz,sz2,sz3,sz4)
- double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4)
  Get the element indexed by (ix1,ix2,ix3,ix4)
- const double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4) const
  Get the element indexed by (ix1,ix2,ix3,ix4)
- void set (size_t ix1, size_t ix2, size_t ix3, size_t ix4, double val)
  Set the element indexed by (ix1,ix2,ix3,ix4) to value `val`.
- double interp (double x, double y, double z, double a)
  Interpolate (x,y,z,a) and return the results.

The documentation for this class was generated from the following file:

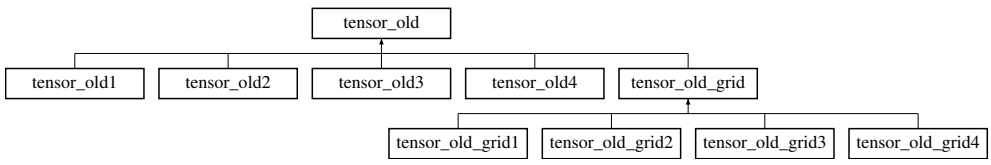
- tensor.h

46.343 tensor_old Class Reference

Tensor_Old class with arbitrary dimensions.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old:



46.343.1 Detailed Description

The elements of a tensor_old are typically specified as a list of `size_t` numbers with length equal to the tensor_old rank. For a rank-4 tensor_old named `t`, the element `t[1][2][0][3]` can be obtained with something similar to

```
size_t ix[4]={1,2,0,3};
double x=t.get(ix);
```

Empty tensor_olds have zero rank.

Slices of tensor_olds are subsets obtained from fixing the index of several dimensions, while letting other dimensions vary. For a slice with one dimension not fixed, see vector_slice(). For a slice with two dimensions not fixed, see matrix_slice().

For I/O with tensor_olds, see hdf_io.h .

Idea for Future Could implement arithmetic operators + and - and some different products.

Idea for Future Implement copies to and from vector and matrices

Idea for Future Implement tensor_old contractions, i.e. tensor_old = tensor_old $*$ tensor_old

Idea for Future Consider making a template type to replace double, e.g. `value_t`.

Idea for Future Could be interesting to write an iterator for this class.

Definition at line 93 of file tensor_old.h.

Public Member Functions

- tensor_old ()
  Create an empty tensor_old with zero rank.
- template<class uint_vec_t >
  tensor_old (size_t rank, const uint_vec_t &dim)
  Create a tensor_old of rank `rank` with sizes given in `dim`.

Set functions

- template<class uint_vec_t >
  void set (const uint_vec_t &index, double val)
  Set the element indexed by `index` to value `val`.
- void set_all (double x)
  Set all elements in a tensor_old to some fixed value.

Get functions

- template<class uint_vec_t >
  double & get (const uint_vec_t &index)
  Get the element indexed by `index`.
- template<class uint_vec_t >
  double const & get (const uint_vec_t &index) const
  Get a const reference to the element indexed by `index`.

Slice functions

- template<class uint_vec_t >
  ovector_array_stride vector_slice (size_t ix, const uint_vec_t &index)
  Fix all but one index to create a vector.
- template<class uint_vec_t >
  omatrix_array matrix_slice (size_t ix, const uint_vec_t &index)
  Fix all but two indices to create a matrix.

Memory allocation

- template<class uint_vec_t >
  void tensor_old_allocate (size_t rank, const uint_vec_t &dim)
  Allocate space for a tensor_old of rank `rank` with sizes given in `dim`.
- void tensor_old_free ()
  Free allocated space (also sets rank to zero)

Size functions

- size_t get_rank () const
  Return the rank of the tensor_old.
- size_t get_size (size_t i) const
  Returns the size of the ith index.
- const size_t * get_size_arr () const
  Return the full array of sizes.
- const double * get_data () const
  Return the full array of sizes.
- size_t total_size () const
  Returns the size of the tensor_old (the product of the sizes over every index)

Index manipulation

- template<class uint_vec_t >
  size_t pack_indices (const uint_vec_t &index)
  Pack the indices into a single array index.
- template<class uint_vec_t >
  void unpack_indices (size_t ix, uint_vec_t &index)
  Unpack the single array index into indices.

Protected Attributes

- double * data
  The data.
- size_t * size
  A rank-sized array of the sizes of each dimension.
- size_t rk
  Rank.

46.343.2 Constructor & Destructor Documentation

46.343.2.1 template<class uint_vec_t > tensor_old::tensor_old (size_t *rank*, const uint_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`. If the user requests any of the sizes to be zero, this constructor will call the error handler, create an empty tensor_old, and will allocate no memory.

Definition at line 127 of file tensor_old.h.

46.343.3 Member Function Documentation

46.343.3.1 template<class uint_vec_t > ovector_array_stride tensor_old::vector_slice (size_t *ix*, const uint_vec_t & *index*) [inline]

This fixes all of the indices to the values given in `index` except for the index number `ix`, and returns the corresponding vector, whose length is equal to the size of the tensor_old in that index. The value `index[ix]` is ignored.

For example, for a rank 3 tensor_old allocated with

```
tensor_old t;
size_t dim[3]={3,4,5};
t.tensor_old_allocate(3,dim);
```

the following code

```
size_t index[3]={1,0,3};
ovector_view v=t.vector_slice(1,index);
```

Gives a vector `v` of length 4 which refers to the values `t(1,0,3)`, `t(1,1,3)`, `t(1,2,3)`, and `t(1,3,3)`.

Definition at line 282 of file tensor_old.h.

46.343.3.2 template<class uint_vec_t> omatrix_array tensor_old::matrix_slice (size_t *ix*, const uint_vec_t & *index*) [inline]

This fixes all of the indices to the values given in `index` except for the index number `ix` and the last index, and returns the corresponding matrix, whose size is equal to the size of the tensor_old in the two indices which are not fixed.

Since the last index is always fixed, the parameter `ix` is limited to be $\in [0, \text{rank}-2]$.

Definition at line 310 of file tensor_old.h.

46.343.3.3 template<class uint_vec_t> void tensor_old::tensor_old_allocate (size_t *rank*, const uint_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`.

If memory was previously allocated, it will be freed before the new allocation.

If the user requests any of the sizes to be zero, this function will call the error handler and will allocate no memory. If memory was previously allocated, the tensor_old is left unmodified and no deallocation is performed.

Reimplemented in tensor_old_grid.

Definition at line 355 of file tensor_old.h.

The documentation for this class was generated from the following file:

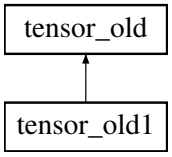
- tensor_old.h

46.344 tensor_old1 Class Reference

Rank 1 tensor_old.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old1:



46.344.1 Detailed Description

Definition at line 970 of file tensor_old.h.

Public Member Functions

- tensor_old1 ()
  Create an empty tensor_old.
- tensor_old1 (size_t sz)
  Create a rank 1 tensor_oldy of size `sz`.

- double & get (size_t ix)
  Get the element indexed by `ix`.
- const double & get (size_t ix) const
  Get the element indexed by `ix`.
- void set (size_t index, double val)
  Set the element indexed by `index` *to value* `val`.
- void set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`.
- double & operator[ ] (size_t ix)
  Get an element using array-like indexing.
- double & operator() (size_t ix)
  Get an element using operator()

46.344.2 Member Function Documentation

46.344.2.1 void tensor_old1::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor_old.)

Definition at line 994 of file tensor_old.h.

The documentation for this class was generated from the following file:

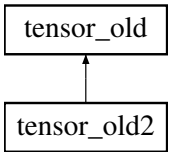
- tensor_old.h

46.345 tensor_old2 Class Reference

Rank 2 tensor_old.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old2:



46.345.1 Detailed Description

Definition at line 1064 of file tensor_old.h.

Public Member Functions

- tensor_old2 ()
  Create an empty tensor_old.
- tensor_old2 (size_t sz, size_t sz2)
  Create a rank 2 tensor_old of size (sz,sz2)
- double & get (size_t ix1, size_t ix2)
  Get the element indexed by (ix1,ix2)
- const double & get (size_t ix1, size_t ix2) const
  Get the element indexed by (ix1,ix2)
- void set (size_t ix1, size_t ix2, double val)
  Set the element indexed by (ix1,ix2) to value `val`.
- void set (size_t *index, double val)

Set the element indexed by `index` *to value* `val`.

- double & operator() (size_t ix, size_t iy)
  Get the element indexed by (ix1,ix2)

46.345.2 Member Function Documentation

46.345.2.1 void tensor_old2::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor_old.)

Definition at line 1104 of file tensor_old.h.

The documentation for this class was generated from the following file:

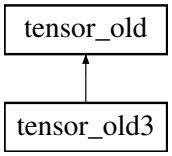
- tensor_old.h

46.346 tensor_old3 Class Reference

Rank 3 tensor_old.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old3:



46.346.1 Detailed Description

Definition at line 1177 of file tensor_old.h.

Public Member Functions

- tensor_old3 ()
  Create an empty tensor_old.
- tensor_old3 (size_t sz, size_t sz2, size_t sz3)
  Create a rank 3 tensor_old of size (sz,sz2,sz3)
- double & get (size_t ix1, size_t ix2, size_t ix3)
  Get the element indexed by (ix1,ix2,ix3)
- const double & get (size_t ix1, size_t ix2, size_t ix3) const
  Get the element indexed by (ix1,ix2,ix3)
- void set (size_t ix1, size_t ix2, size_t ix3, double val)
  Set the element indexed by (ix1,ix2,ix3) to value `val`.
- void set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`.

46.346.2 Member Function Documentation

46.346.2.1 void tensor_old3::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor_old.)

Definition at line 1218 of file tensor_old.h.

The documentation for this class was generated from the following file:

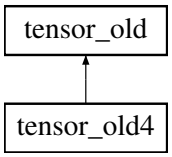
- tensor_old.h

46.347 tensor_old4 Class Reference

Rank 4 tensor_old.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old4:



46.347.1 Detailed Description

Definition at line 1289 of file tensor_old.h.

Public Member Functions

- tensor_old4 ()
  Create an empty tensor_old.
- tensor_old4 (size_t sz, size_t sz2, size_t sz3, size_t sz4)
  Create a rank 4 tensor_old of size (sz,sz2,sz3,sz4)
- double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4)
  Get the element indexed by (ix1,ix2,ix3,ix4)
- const double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4) const
  Get the element indexed by (ix1,ix2,ix3,ix4)
- void set (size_t ix1, size_t ix2, size_t ix3, size_t ix4, double val)
  Set the element indexed by (ix1,ix2,ix3,ix4) to value `val`.
- void set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`.

46.347.2 Member Function Documentation

46.347.2.1 void tensor_old4::set (size_t * *index*, double *val*) [inline]

(We have to explicitly provide this version since the set() function is overloaded in this child of tensor_old.)

Definition at line 1335 of file tensor_old.h.

The documentation for this class was generated from the following file:

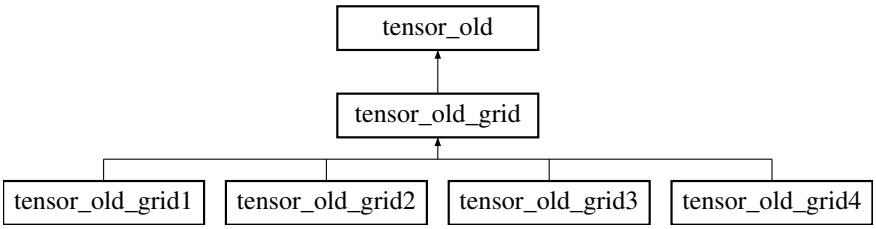
- tensor_old.h

46.348 tensor_old_grid Class Reference

Tensor_Old class with arbitrary dimensions with a grid.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old_grid:



46.348.1 Detailed Description

This tensor_old class allows one to assign the indexes to numerical scales, so that n-dimensional interpolation can be performed. To set the grid, use set_grid() and then interpolation can be done using interpolate().

By convention, member functions ending in the _val suffix return the closest grid-point to some user-specified values.

Idea for Future Only allocate space for grid if it is set

Idea for Future Could implement arithmetic operators + and - and some different products.

Idea for Future Consider creating a set_grid() function which takes grids from an object like hist_grid. Maybe make a constructor for a tensor_old_grid object which just takes as input a set of grids?

Definition at line 508 of file tensor_old.h.

Public Member Functions

- tensor_old_grid ()
  Create an empty tensor_old with zero rank.
- template<class uint_vec_t >
  tensor_old_grid (size_t rank, const uint_vec_t &dim)
  Create a tensor_old of rank `rank` with sizes given in `dim`.

Set functions

- template<class vec_t >
  int set_val (const vec_t &grdp, double val)
  Set the element closest to grid point `grdp` to value `val`.
- template<class vec_t , class vec2_t >
  int set_val (const vec_t &grdp, double val, vec2_t &closest)
  Set the element closest to grid point `grdp` to value `val`.

Get functions

- template<class vec_t >
  double get_val (const vec_t &grdp)
  Get the element closest to grid point `grdp`.
- template<class vec_t , class vec2_t >
  double get_val (const vec_t &grdp, const vec2_t &closest)
  Get the element closest to grid point `grdp` to value `val`.

Memory allocation

- template<class uint_vec_t >
  int tensor_old_allocate (size_t rank, const uint_vec_t &dim)
  Allocate space for a tensor_old of rank `rank` with sizes given in `dim`.
- int tensor_old_free ()
  Free allocated space (also sets rank to zero)

Grid manipulation

- bool is_grid_set () const
  Return true if the grid has been set.
- template<class vec_t >
  int set_grid (const vec_t &grid)
  Set the grid.
- int **set_grid_old** (double **vals)
- size_t lookup_grid (size_t i, double val)
  Lookup index for grid closest to `val`.
- double get_grid (size_t i, size_t j) const
  Lookup index for grid closest to `val`.
- template<class vec_t , class uint_vec_t >
  int lookup_grid_vec (const vec_t &vals, const uint_vec_t &indices) const
  Lookup indices for grid closest point to `vals`.
- size_t lookup_grid_val (size_t i, double &val, double &val2)
  Lookup index for grid closest to `val`, returning the grid point.

Protected Attributes

- double ** grd
  A rank-sized set of arrays for the grid points.
- bool grid_set
  If true, the grid has been set by the user.
- base_interp_mgr< double * > * bim1
  The interpolation manager.
- base_interp_mgr < array_const_subvector > * bim2
  The subvector interpolation manager.

Interpolation

- def_interp_mgr< double *, linear_interp > **dim1**
- def_interp_mgr < array_const_subvector, linear_interp > **dim2**
- int set_interp (base_interp_mgr< double * > &bi1, base_interp_mgr< array_const_subvector > &bi2)
  Set interpolation managers.
- double interpolate (double *vals)
  Interpolate values `vals` into the tensor_old, returning the result.

46.348.2 Constructor & Destructor Documentation

46.348.2.1 template<class uint_vec_t > tensor_old_grid::tensor_old_grid (size_t *rank*, const uint_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`. If the user requests any of the sizes to be zero, this constructor will call the error handler, create an empty tensor_old, and will allocate no memory.

Definition at line 546 of file tensor_old.h.

46.348.3 Member Function Documentation

46.348.3.1 template<class vec_t , class vec2_t > int tensor_old_grid::set_val (const vec_t & *grdp,* double *val,* vec2_t & *closest*) [inline]

The parameters `closest` and `grdp` may be identical, allowing one to update a vector `grdp` with the closest grid point.

Definition at line 604 of file tensor_old.h.

46.348.3.2 template<class uint_vec_t > int tensor_old_grid::tensor_old_allocate (size_t *rank,* const uint_vec_t & *dim*) [inline]

The parameter `dim` must be a pointer to an array of sizes with length `rank`.

If memory was previously allocated, it will be freed before the new allocation and previously specified grid data will be lost.

If the user requests any of the sizes to be zero, this function will call the error handler and will allocate no memory. If memory was previously allocated, the tensor_old is left unmodified and no deallocation is performed.

Reimplemented from tensor_old.

Definition at line 696 of file tensor_old.h.

46.348.3.3 template<class vec_t > int tensor_old_grid::set_grid (const vec_t & *grid*) [inline]

The grid must be specified for all of the dimensions at once. Denote $(\text{size})_0$ as the size of the first dimension, $(\text{size})_1$ as the size of the second dimesion, and so on. Then the first $(\text{size})_0$ entries in `grid` must be the grid for the first dimension, the next $(\text{size})_1$ entries must be the grid for the second dimension, and so on. Thus `grid` must be a vector of size

$$\sum_{i=0}^{\text{rank}} (\text{size})_i$$

Note that the grid is copied so the function argument may be destroyed by the user after calling set_grid() without affecting the tensor_old grid.

Idea for Future Define a more generic interface for matrix types

Definition at line 757 of file tensor_old.h.

46.348.3.4 template<class vec_t , class uint_vec_t > int tensor_old_grid::lookup_grid_vec (const vec_t & *vals,* const uint_vec_t & *indices*) const [inline]

The values in `vals` are not modified by this function.

Definition at line 821 of file tensor_old.h.

46.348.3.5 double tensor_old_grid::interpolate (double * *vals*) [inline]

This is a quick and dirty implementation of n-dimensional interpolation by recursive application of the 1-dimensional routine from smart_interp_vec, using the base interpolation object specified in the template parameter `base_interp_t`. This will be slow for sufficiently large data sets.

Idea for Future Maybe make this a template as well?

Idea for Future It should be straightforward to improve the scaling of this algorithm significantly by creating a "window" of local points around the point of interest. This could be done easily by constructing an initial subtensor_old. However, this should probably be superceded by a more generic alternative which avoids explicit use of the 1-d interpolation types.

Definition at line 886 of file tensor_old.h.

The documentation for this class was generated from the following file:

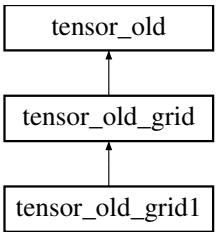
- tensor_old.h

46.349 tensor_old_grid1 Class Reference

Rank 2 tensor_old with a grid.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old_grid1:



46.349.1 Detailed Description

Definition at line 1007 of file tensor_old.h.

Public Member Functions

- tensor_old_grid1 ()
  Create an empty tensor_old.
- tensor_old_grid1 (size_t sz)
  Create a rank 2 tensor_old of size (sz,sz2,sz3)
- double & get (size_t ix1)
  Get the element indexed by (ix1)
- const double & get (size_t ix1) const
  Get the element indexed by (ix1)
- void set (size_t ix1, double val)
  Set the element indexed by (ix1) to value `val`.
- double interp (double x)
  Interpolate `x` and return the results.

The documentation for this class was generated from the following file:

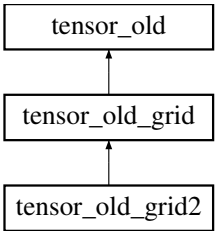
- tensor_old.h

46.350 tensor_old_grid2 Class Reference

Rank 2 tensor_old with a grid.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old_grid2:



46.350.1 Detailed Description

Definition at line 1116 of file tensor_old.h.

Public Member Functions

- tensor_old_grid2 ()
  Create an empty tensor_old.
- tensor_old_grid2 (size_t sz, size_t sz2)
  Create a rank 2 tensor_old of size (sz,sz2)
- double & get (size_t ix1, size_t ix2)
  Get the element indexed by (ix1,ix2)
- const double & get (size_t ix1, size_t ix2) const
  Get the element indexed by (ix1,ix2)
- void set (size_t ix1, size_t ix2, double val)
  Set the element indexed by (ix1,ix2) to value `val`.
- double interp (double x, double y)
  Interpolate (x,y) and return the results.

The documentation for this class was generated from the following file:

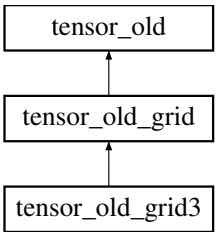
- tensor_old.h

46.351 tensor_old_grid3 Class Reference

Rank 3 tensor_old with a grid.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old_grid3:



46.351.1 Detailed Description

Definition at line 1226 of file tensor_old.h.

Public Member Functions

- tensor_old_grid3 ()
  Create an empty tensor_old.
- tensor_old_grid3 (size_t sz, size_t sz2, size_t sz3)
  Create a rank 3 tensor_old of size (sz,sz2,sz3)
- double & get (size_t ix1, size_t ix2, size_t ix3)
  Get the element indexed by (ix1,ix2,ix3)
- const double & get (size_t ix1, size_t ix2, size_t ix3) const
  Get the element indexed by (ix1,ix2,ix3)
- void set (size_t ix1, size_t ix2, size_t ix3, double val)
  Set the element indexed by (ix1,ix2,ix3) to value `val`.
- double interp (double x, double y, double z)
  Interpolate (x,y,z) and return the results.

The documentation for this class was generated from the following file:

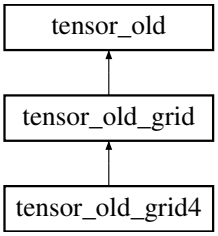
- tensor_old.h

46.352 tensor_old_grid4 Class Reference

Rank 4 tensor_old with a grid.

`#include <tensor_old.h>`

Inheritance diagram for tensor_old_grid4:



46.352.1 Detailed Description

Definition at line 1343 of file tensor_old.h.

Public Member Functions

- tensor_old_grid4 ()
  Create an empty tensor_old.
- tensor_old_grid4 (size_t sz, size_t sz2, size_t sz3, size_t sz4)
  Create a rank 4 tensor_old of size (sz,sz2,sz3,sz4)
- double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4)
  Get the element indexed by (ix1,ix2,ix3,ix4)
- const double & get (size_t ix1, size_t ix2, size_t ix3, size_t ix4) const
  Get the element indexed by (ix1,ix2,ix3,ix4)
- void set (size_t ix1, size_t ix2, size_t ix3, size_t ix4, double val)
  Set the element indexed by (ix1,ix2,ix3,ix4) to value `val`.
- double interp (double x, double y, double z, double a)
  Interpolate (x,y,z,a) and return the results.

The documentation for this class was generated from the following file:

- tensor_old.h

46.353 test_mgr Class Reference

A class to manage testing and record success and failure.

`#include <test_mgr.h>`

46.353.1 Detailed Description

Idea for Future test_mgr::success and test_mgr::last_fail should be protected, but that breaks the operator+() function. Can this be fixed?

Definition at line 37 of file test_mgr.h.

Public Member Functions

- bool report ()
  Provide a report of all tests so far.
- std::string get_last_fail ()
  Returns the description of the last test that failed.
- void set_output_level (int l)
  Set the output level.
- int get_ntests ()
  Return the number of tests performed so far.

The testing methods

- bool test_rel (double result, double expected, double rel_error, std::string description)
  Test for $|\text{result} - \text{expected}|/\text{expected} < \text{rel\_error}$.
- bool test_abs (double result, double expected, double abs_error, std::string description)
  Test for $|\text{result} - \text{expected}| < \text{abs\_error}$.
- bool test_fact (double result, double expected, double factor, std::string description)
  Test for $1/\text{factor} < \text{result}/\text{expected} < \text{factor}$.
- bool test_str (std::string result, std::string expected, std::string description)
  Test for result = expected.
- bool test_gen (bool value, std::string description)
  Test for result = expected.
- template<class vec_t , class vec2_t >
  bool test_rel_arr (int nv, const vec_t &result, const vec2_t &expected, double rel_error, std::string description)
  Test for $|\text{result} - \text{expected}|/\text{expected} < \text{rel\_error}$ *over each element of an array.*
- template<class mat_t , class mat2_t >
  bool test_rel_mat (int nr, int nc, const mat_t &result, const mat2_t &expected, double rel_error, std::string description)
  Test for $|\text{result} - \text{expected}|/\text{expected} < \text{rel\_error}$ *over each element of an array.*
- template<class vec_t , class vec2_t >
  bool test_abs_arr (int nv, const vec_t &result, const vec2_t &expected, double rel_error, std::string description)
  Test for $|\text{result} - \text{expected}| < \text{abs\_error}$ *over each element of an array.*
- template<class vec_t , class vec2_t >
  bool test_fact_arr (int nv, const vec_t &result, const vec2_t &expected, double factor, std::string description)
  Test for $1/factor < result/expected < factor$ *over each element of an array.*
- template<class vec_t >
  bool test_gen_arr (int nv, const vec_t &result, const vec_t &expected, std::string description)
  Test for equality of a generic array.

Data Fields

- bool success
  True if all tests have passed.
- std::string last_fail
  The description of the last failed test.

Protected Member Functions

- void process_test (bool ret, std::string d2, std::string description)
  A helper function for processing tests.

Protected Attributes

- int ntests
  The number of tests performed.
- int output_level
  The output level.

Friends

- const test_mgr operator+ (const test_mgr &left, const test_mgr &right)
  Add two test_mgr objects (if either failed, the sum fails)

46.353.2 Member Function Documentation

46.353.2.1 bool test_mgr::report ( )

Returns true if all tests have passed and false if at least one test failed.

46.353.2.2 void test_mgr::set_output_level (int /) [inline]

Possible values:

- 0 = No output
- 1 = Output only tests that fail
- 2 = Output all tests

Definition at line 79 of file test_mgr.h.

The documentation for this class was generated from the following file:

- test_mgr.h

46.354 twod_eqi_intp Class Reference

Two-dimensional interpolation for equally-spaced intervals.

`#include <twod_eqi_intp.h>`

46.354.1 Detailed Description

Note

This class is unfinished.

This implements the relations from Abramowitz and Stegun:

$$f(x_0+ph, y_0+qk) =$$

3-point

$$(1-p-q)f_{0,0} + pf_{1,0} + qf_{0,1}$$

4-point

$$(1-p)(1-q)f_{0,0} + p(1-q)f_{1,0} + q(1-p)f_{0,1} + pqf_{1,1}$$

6-point

$$\frac{q(q-1)}{2}f_{0,-1} + \frac{p(p-1)}{2}f_{-1,0} + (1+pq-p^2-q^2)f_{0,0} + \frac{p(p-2q+1)}{2}f_{1,0} + \frac{q(q-2p+1)}{2}f_{0,1} + pqf_{1,1}$$

Definition at line 56 of file twod_eqi_intp.h.

Public Member Functions

- double interp (double x, double y)
  Perform the 2-d interpolation.
- int set_type (int type)
  Set the interpolation type.

Data Fields

- double xoff
  Offset in x-direction.
- double yoff
  Offset in y-direction.

46.354.2 Member Function Documentation

46.354.2.1 int twod_eqi_intp::set_type (int *type*) [inline]

- 3: 3-point
- 4: 4-point
- 6: 6-point (default)

Definition at line 79 of file twod_eqi_intp.h.

The documentation for this class was generated from the following file:

- twod_eqi_intp.h

46.355 twod_intp Class Reference

Two-dimensional interpolation class.

`#include <twod_intp.h>`

46.355.1 Detailed Description

This class implements two-dimensional interpolation. Derivatives and integrals along both x- and y-directions can be computed.

The storage of the matrix to be specified in the function set_data() and this function is designed to follow the format:

$$\begin{array}{ccccc} & x_0 & x_1 & x_2 \\ y_0 & M_{00} & M_{01} & M_{02} \\ y_1 & M_{10} & M_{11} & M_{12} \\ y_2 & M_{20} & M_{21} & M_{22} \end{array}$$

thus the matrix should be `M`[i][j] where `i` is the y index and `j` is the x index. (See also the discussion in the User's guide in the section called Rows and columns vs. x and y.)

The function set_data(), does not copy the data, it stores pointers to the data. If the data is modified, then the function reset_interp() must be called to reset the interpolation information with the original pointer information. The storage for the data, including the arrays `x_grid` and `y_grid` are all managed by the user.

By default, cubic spline interpolation with natural boundary conditions is used. This can be changed with the set_interp() function.

There is an example for the usage of this class given in `examples/ex_twod_intp.cpp`.

Idea for Future Could also include mixed second/first derivatives: deriv_xxy and deriv_xyy.

Idea for Future Implement an improved caching system in case, for example `xfirst` is true and the last interpolation used the same value of `x`.

Definition at line 99 of file twod_intp.h.

Public Member Functions

- int set_data (size_t n_x, size_t n_y, ovector &x_grid, ovector &y_grid, omatrix &data, bool x_first=true)
  Initialize the data for the 2-dimensional interpolation.
- int reset_interp ()
  Reset the stored interpolation since the data has changed.
- double interp (double x, double y)
  Perform the 2-d interpolation.
- double deriv_x (double x, double y)
  Compute the partial derivative in the x-direction.
- double deriv2_x (double x, double y)
  Compute the partial second derivative in the x-direction.
- double integ_x (double x0, double x1, double y)
  Compute the integral in the x-direction between x=x0 and x=x1.
- double deriv_y (double x, double y)
  Compute the partial derivative in the y-direction.
- double deriv2_y (double x, double y)
  Compute the partial second derivative in the y-direction.
- double integ_y (double x, double y0, double y1)
  Compute the integral in the y-direction between y=y0 and y=y1.
- double deriv_xy (double x, double y)
  Compute the mixed partial derivative $\frac{\partial^2 f}{\partial x \partial y}$.
- int set_interp (base_interp_mgr< ovector_const_view > &b1, base_interp_mgr< ovector_const_subvector > &b2)
  Specify the base interpolation objects to use.
- int unset_data ()
  Inform the class the data has been modified or changed in a way that set_data() will need to be called again.

46.355.2 Member Function Documentation

46.355.2.1 int twod_intp::set_data (size_t *n_x*, size_t *n_y*, ovector & *x_grid*, ovector & *y_grid*, omatrix & *data*, bool *x_first* = true)

The interpolation type (passed directly to int_type) is specified in int_type and the data is specified in data. The data should be arranged so that the first array index is the y-value (the "row") and the second array index is the x-value (the "column"). The arrays xfun and yfun specify the two independent variables. xfun should be an array of length nx , and should be an array of length ny. The array data should be a two-dimensional array of size [ny][nx].

If x_first is true, then set_data() creates interpolation objects for each of the rows. Calls to interp() then uses these to create a column at the specified value of x. An interpolation object is created at this column to find the value of the function at the specified value y. If x_first is false, the opposite strategy is employed. These two options may give slightly different results.

46.355.2.2 int twod_intp::reset_interp ( )

This will return an error if the set_data() has not been called

46.355.2.3 int twod_intp::set_interp (base_interp_mgr< ovector_const_view > & *b1*, base_interp_mgr< ovector_const_subvector > & *b2*) [inline]

This allows the user to provide new interpolation objects for use in the two-dimensional interpolation. For example,

```
twod_intp ti;
ovector x(20), y(40);
omatrix d(40,20);

// Fill x, y, and d with the data, choose linear interpolation
// instead of the default cubic spline
def_interp_mgr<ovector_const_view,linear_interp> dim1;
def_interp_mgr<ovector_const_subvector,linear_interp> dim2;

ti.set_interp(dim1,dim2);
ti.set_data(20,40,x,y,d,true);
```

This function automatically calls reset_interp() and reset_data() if the data has already been set to reset the internal interpolation objects.

Definition at line 197 of file twod_intp.h.

The documentation for this class was generated from the following file:

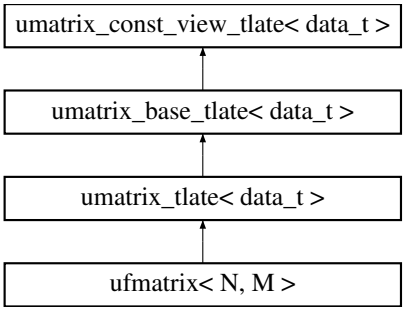
- twod_intp.h

46.356 ufmatrix< N, M > Class Template Reference

A matrix where the memory allocation is performed in the constructor.

`#include <umatrix_tlate.h>`

Inheritance diagram for ufmatrix< N, M >:



46.356.1 Detailed Description

template<size_t N, size_t M>class ufmatrix< N, M >

Definition at line 915 of file umatrix_tlate.h.

The documentation for this class was generated from the following file:

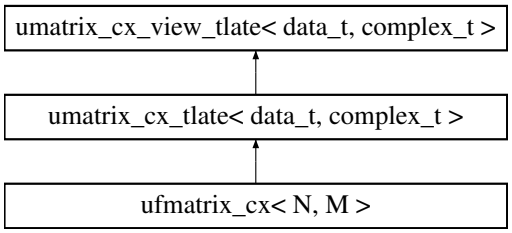
- umatrix_tlate.h

46.357 ufmatrix_cx< N, M > Class Template Reference

A matrix where the memory allocation is performed in the constructor.

`#include <umatrix_cx_tlate.h>`

Inheritance diagram for ufmatrix_cx< N, M >:



46.357.1 Detailed Description

template<size_t N, size_t M>class ufmatrix_cx< N, M >

Definition at line 708 of file umatrix_cx_tlate.h.

The documentation for this class was generated from the following file:

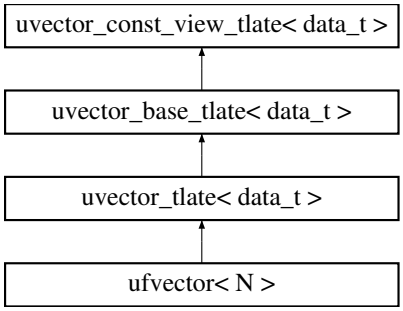
- umatrix_cx_tlate.h

46.358 ufvector< N > Class Template Reference

A vector with unit-stride where the memory allocation is performed in the constructor.

`#include <uvector_tlate.h>`

Inheritance diagram for ufvector< N >:



46.358.1 Detailed Description

template<size_t N = 0>class ufvector< N >

Definition at line 1054 of file uvector_tlate.h.

The documentation for this class was generated from the following file:

- uvector_tlate.h

46.359 umatrix_alloc Class Reference

A simple class to provide an `allocate()` function for umatrix.

`#include <umatrix_tlate.h>`

46.359.1 Detailed Description

Definition at line 903 of file umatrix_tlate.h.

Public Member Functions

- void allocate (umatrix &o, int i, int j)
  Allocate `v` for `i` elements.
- void free (umatrix &o)
  Free memory.

The documentation for this class was generated from the following file:

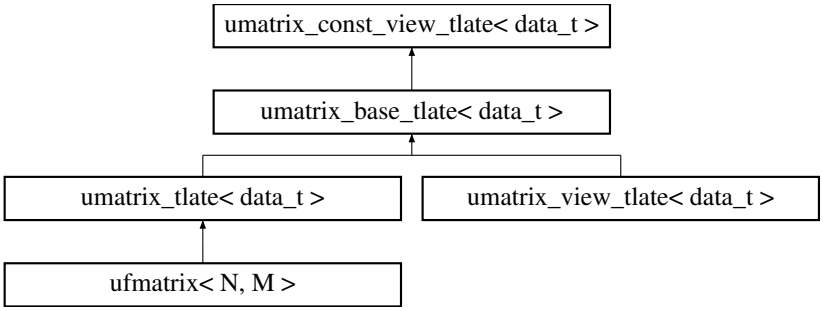
- umatrix_tlate.h

46.360 umatrix_base_tlate< data_t > Class Template Reference

A matrix view of double-precision numbers.

`#include <umatrix_tlate.h>`

Inheritance diagram for umatrix_base_tlate< data_t >:



46.360.1 Detailed Description

template<class data_t>class umatrix_base_tlate< data_t >

Definition at line 200 of file umatrix_tlate.h.

Public Member Functions

Copy constructors

- umatrix_base_tlate (umatrix_base_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- umatrix_base_tlate & operator= (umatrix_base_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- data_t * operator[ ] (size_t i)
  Array-like indexing.
- const data_t * operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i, size_t j)
  Array-like indexing.
- const data_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- data_t * get_ptr (size_t i, size_t j)
  Get pointer (with optional range-checking)
- int set (size_t i, size_t j, data_t val)
  Set (with optional range-checking)
- int set_all (data_t val)
  Set all of the value to be the value `val`.

Arithmetic

- umatrix_base_tlate< data_t > & operator+= (const umatrix_base_tlate< data_t > &x)
  operator+=
- umatrix_base_tlate< data_t > & operator-= (const umatrix_base_tlate< data_t > &x)
  operator-=
- umatrix_base_tlate< data_t > & operator+= (const data_t &y)
  operator+=
- umatrix_base_tlate< data_t > & operator-= (const data_t &y)
  operator-=
- umatrix_base_tlate< data_t > & operator*= (const data_t &y)
  operator=*

Protected Member Functions

- umatrix_base_tlate ()
  Empty constructor.

The documentation for this class was generated from the following file:

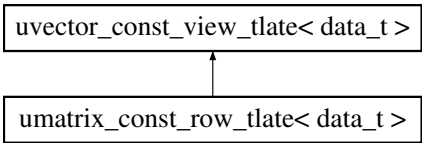
- umatrix_tlate.h

46.361 umatrix_const_row_tlate< data_t > Class Template Reference

Create a const vector from a row of a matrix.

`#include <umatrix_tlate.h>`

Inheritance diagram for umatrix_const_row_tlate< data_t >:



46.361.1 Detailed Description

template<class data_t>class umatrix_const_row_tlate< data_t >

Definition at line 818 of file umatrix_tlate.h.

Public Member Functions

- umatrix_const_row_tlate (const umatrix_base_tlate< data_t > &m, size_t i)
  Create a vector from row `i` of matrix `m`.

The documentation for this class was generated from the following file:

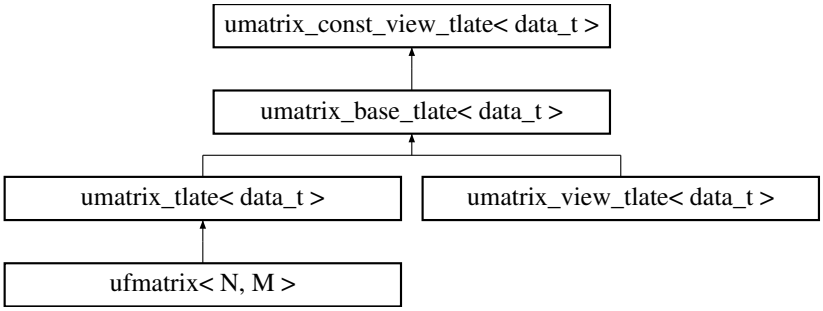
- umatrix_tlate.h

46.362 umatrix_const_view_tlate< data_t > Class Template Reference

A matrix view of double-precision numbers.

`#include <umatrix_tlate.h>`

Inheritance diagram for umatrix_const_view_tlate< data_t >:



46.362.1 Detailed Description

template<class data_t>class umatrix_const_view_tlate< data_t >

Definition at line 48 of file umatrix_tlate.h.

Public Member Functions

Copy constructors

- umatrix_const_view_tlate (const umatrix_const_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- umatrix_const_view_tlate & operator= (const umatrix_const_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- const data_t * operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- data_t get (size_t i, size_t j) const
  Get (with optional range-checking)
- const data_t * get_const_ptr (size_t i, size_t j) const
  Get pointer (with optional range-checking)
- size_t rows () const
  Method to return number of rows.
- size_t cols () const
  Method to return number of columns.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.

Protected Member Functions

- umatrix_const_view_tlate ()
  Empty constructor provided for use by umatrix_tlate(const umatrix_tlate &v)

Protected Attributes

- data_t * data
  The data.
- size_t size1

The number of rows.

- size_t size2
  The number of columns.
- int owner
  Zero if memory is owned elsewhere, 1 otherwise.

46.362.2 Member Function Documentation

46.362.2.1 template<class data_t> size_t umatrix_const_view_tlate< data_t >::rows ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 162 of file umatrix_tlate.h.

46.362.2.2 template<class data_t> size_t umatrix_const_view_tlate< data_t >::cols ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 171 of file umatrix_tlate.h.

The documentation for this class was generated from the following file:

- umatrix_tlate.h

46.363 umatrix_cx_alloc Class Reference

A simple class to provide an `allocate()` function for umatrix_cx.

`#include <umatrix_cx_tlate.h>`

46.363.1 Detailed Description

Definition at line 696 of file umatrix_cx_tlate.h.

Public Member Functions

- void allocate (umatrix_cx &o, int i, int j)
  Allocate `v` for `i` elements.
- void free (umatrix_cx &o)
  Free memory.

The documentation for this class was generated from the following file:

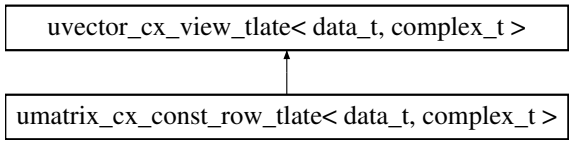
- umatrix_cx_tlate.h

46.364 umatrix_cx_const_row_tlate< data_t, complex_t > Class Template Reference

Create a const vector from a row of a matrix.

`#include <umatrix_cx_tlate.h>`

Inheritance diagram for umatrix_cx_const_row_tlate< data_t, complex_t >:



46.364.1 Detailed Description

template<class data_t, class complex_t>class umatrix_cx_const_row_tlate< data_t, complex_t >

Definition at line 624 of file umatrix_cx_tlate.h.

Public Member Functions

- umatrix_cx_const_row_tlate (const umatrix_cx_view_tlate< data_t, complex_t > &m, size_t i)
  Create a vector from row `i` of matrix `m`.

The documentation for this class was generated from the following file:

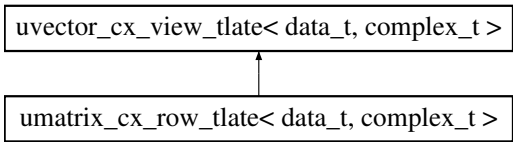
- umatrix_cx_tlate.h

46.365 umatrix_cx_row_tlate< data_t, complex_t > Class Template Reference

Create a vector from a row of a matrix.

`#include <umatrix_cx_tlate.h>`

Inheritance diagram for umatrix_cx_row_tlate< data_t, complex_t >:



46.365.1 Detailed Description

template<class data_t, class complex_t>class umatrix_cx_row_tlate< data_t, complex_t >

Definition at line 608 of file umatrix_cx_tlate.h.

Public Member Functions

- umatrix_cx_row_tlate (umatrix_cx_view_tlate< data_t, complex_t > &m, size_t i)
  Create a vector from row `i` of matrix `m`.

The documentation for this class was generated from the following file:

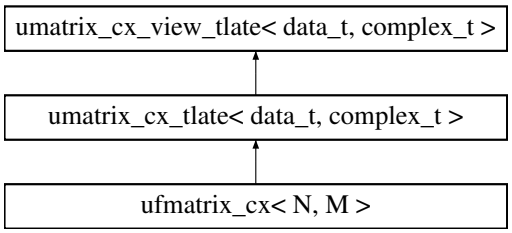
- umatrix_cx_tlate.h

46.366 umatrix_cx_tlate< data_t, complex_t > Class Template Reference

A matrix of double-precision numbers.

`#include <umatrix_cx_tlate.h>`

Inheritance diagram for umatrix_cx_tlate< data_t, complex_t >:



46.366.1 Detailed Description

template<class data_t, class complex_t>class umatrix_cx_tlate< data_t, complex_t >

Definition at line 366 of file umatrix_cx_tlate.h.

Public Member Functions

Standard constructor

- umatrix_cx_tlate (size_t r=0, size_t c=0)
  Create an umatrix of size n with owner as 'true'.

Copy constructors

- umatrix_cx_tlate (const umatrix_cx_tlate &v)
  Deep copy constructor, allocate new space and make a copy.
- umatrix_cx_tlate (const umatrix_cx_view_tlate< data_t, complex_t > &v)
  Deep copy constructor, allocate new space and make a copy.
- umatrix_cx_tlate & operator= (const umatrix_cx_tlate &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- umatrix_cx_tlate & operator= (const umatrix_cx_view_tlate< data_t, complex_t > &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- umatrix_cx_tlate (size_t n, uvector_cx_view_tlate< data_t, complex_t > uva[])
  Deep copy from an array of uvectors.
- umatrix_cx_tlate (size_t n, size_t n2, data_t **csa)
  Deep copy from a C-style 2-d array.

Memory allocation

- int allocate (size_t nrows, size_t ncols)
  Allocate memory after freeing any memory presently in use.
- int free ()
  Free the memory.

Other methods

- umatrix_cx_tlate< data_t, complex_t > transpose ()
  Compute the transpose (even if matrix is not square)

46.366.2 Member Function Documentation

46.366.2.1 template<class data_t, class complex_t> int umatrix_cx_tlate< data_t, complex_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 578 of file umatrix_cx_tlate.h.

The documentation for this class was generated from the following file:

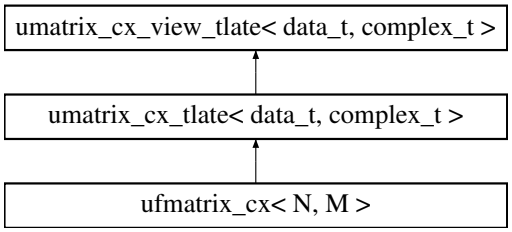
- umatrix_cx_tlate.h

46.367 umatrix_cx_view_tlate< data_t, complex_t > Class Template Reference

A matrix view of complex numbers.

`#include <umatrix_cx_tlate.h>`

Inheritance diagram for umatrix_cx_view_tlate< data_t, complex_t >:



46.367.1 Detailed Description

template<class data_t, class complex_t>class umatrix_cx_view_tlate< data_t, complex_t >

Definition at line 49 of file umatrix_cx_tlate.h.

Public Member Functions

Copy constructors

- umatrix_cx_view_tlate (const umatrix_cx_view_tlate &v)
  So2scllow copy constructor - create a new view of the same matrix.
- umatrix_cx_view_tlate & operator= (const umatrix_cx_view_tlate &v)
  So2scllow copy constructor - create a new view of the same matrix.

Get and set methods

- complex_t * operator[ ] (size_t i)
  Array-like indexing.
- const complex_t * operator[ ] (size_t i) const
  Array-like indexing.
- complex_t & operator() (size_t i, size_t j)
  Array-like indexing.
- const complex_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- complex_t get (size_t i, size_t j) const
  Get (with optional range-checking)
- complex_t * get_ptr (size_t i, size_t j)
  Get pointer (with optional range-checking)

- const complex_t * get_const_ptr (size_t i, size_t j) const
  Get pointer (with optional range-checking)
- int set (size_t i, size_t j, complex_t val)
  Set (with optional range-checking)
- int set (size_t i, size_t j, data_t re, data_t im)
  Set (with optional range-checking)
- int set_all (complex_t val)
  Set all of the value to be the value `val`.
- size_t rows () const
  Method to return number of rows.
- size_t cols () const
  Method to return number of columns.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.

Arithmetic

- umatrix_cx_view_tlate< data_t, complex_t > & operator+= (const umatrix_cx_view_tlate< data_t, complex_t > &x)
  operator+=
- umatrix_cx_view_tlate< data_t, complex_t > & operator-= (const umatrix_cx_view_tlate< data_t, complex_t > &x)
  operator-=
- umatrix_cx_view_tlate< data_t, complex_t > & operator+= (const data_t &y)
  operator+=
- umatrix_cx_view_tlate< data_t, complex_t > & operator-= (const data_t &y)
  operator-=
- umatrix_cx_view_tlate< data_t, complex_t > & operator*= (const data_t &y)
  operator=*

Protected Member Functions

- umatrix_cx_view_tlate ()
  Empty constructor provided for use by umatrix_cx_tlate(const umatrix_cx_tlate &v)

Protected Attributes

- data_t * data
  The data.
- size_t size1
  The number of rows.
- size_t size2
  The number of columns.
- int owner
  Zero if memory is owned elsewhere, 1 otherwise.

46.367.2 Member Function Documentation

46.367.2.1 template<class data_t, class complex_t> size_t umatrix_cx_view_tlate< data_t, complex_t >::rows ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 259 of file umatrix_cx_tlate.h.

46.367.2.2 template<class data_t, class complex_t> size_t umatrix_cx_view_tlate< data_t, complex_t >::cols ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 268 of file umatrix_cx_tlate.h.

The documentation for this class was generated from the following file:

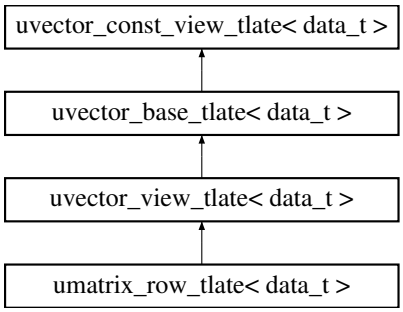
- umatrix_cx_tlate.h

46.368 umatrix_row_tlate< data_t > Class Template Reference

Create a vector from a row of a matrix.

`#include <umatrix_tlate.h>`

Inheritance diagram for umatrix_row_tlate< data_t >:



46.368.1 Detailed Description

template<class data_t>class umatrix_row_tlate< data_t >

Definition at line 800 of file umatrix_tlate.h.

Public Member Functions

- umatrix_row_tlate (umatrix_base_tlate< data_t > &m, size_t i)
  Create a vector from row `i` of matrix `m`.

The documentation for this class was generated from the following file:

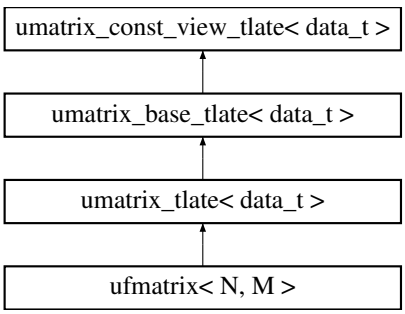
- umatrix_tlate.h

46.369 umatrix_tlate< data_t > Class Template Reference

A matrix of double-precision numbers.

`#include <umatrix_tlate.h>`

Inheritance diagram for umatrix_tlate< data_t >:



46.369.1 Detailed Description

template<class data_t>class umatrix_tlate< data_t >

Definition at line 561 of file umatrix_tlate.h.

Public Member Functions

Standard constructor

- umatrix_tlate (size_t r=0, size_t c=0)
  Create an umatrix of size n with owner as 'true'.

Copy constructors

- umatrix_tlate (const umatrix_tlate &v)
  Deep copy constructor, allocate new space and make a copy.
- umatrix_tlate (const umatrix_view_tlate< data_t > &v)
  Deep copy constructor, allocate new space and make a copy.
- umatrix_tlate & operator= (const umatrix_tlate &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- umatrix_tlate & operator= (const umatrix_view_tlate< data_t > &v)
  Deep copy constructor, if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- umatrix_tlate (size_t n, uvector_view_tlate< data_t > uva[])
  Deep copy from an array of uvectors.
- umatrix_tlate (size_t n, size_t n2, data_t **csa)
  Deep copy from a C-style 2-d array.

Memory allocation

- int allocate (size_t nrows, size_t ncols)
  Allocate memory after freeing any memory presently in use.
- int free ()
  Free the memory.

Other methods

- umatrix_tlate< data_t > transpose ()
  Compute the transpose (even if matrix is not square)

46.369.2 Member Function Documentation

46.369.2.1 template<class data_t> int umatrix_tlate< data_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 770 of file umatrix_tlate.h.

The documentation for this class was generated from the following file:

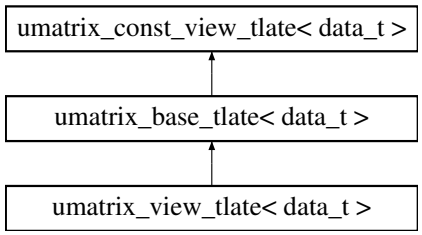
- umatrix_tlate.h

46.370 umatrix_view_tlate< data_t > Class Template Reference

A matrix view of double-precision numbers.

`#include <umatrix_tlate.h>`

Inheritance diagram for umatrix_view_tlate< data_t >:



46.370.1 Detailed Description

template<class data_t>class umatrix_view_tlate< data_t >

Definition at line 422 of file umatrix_tlate.h.

Public Member Functions

Copy constructors

- umatrix_view_tlate (const umatrix_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- umatrix_view_tlate & operator= (const umatrix_view_tlate &v)
  Shallow copy constructor - create a new view of the same matrix.
- umatrix_view_tlate (umatrix_base_tlate< data_t > &v)
  Shallow copy constructor - create a new view of the same matrix.
- umatrix_view_tlate & operator= (umatrix_base_tlate< data_t > &v)
  Shallow copy constructor - create a new view of the same matrix.

Get and set methods

- data_t * operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i, size_t j) const
  Array-like indexing.
- data_t * get_ptr (size_t i, size_t j) const
  Get pointer (with optional range-checking)
- int set (size_t i, size_t j, data_t val) const
  Set (with optional range-checking)
- int set_all (double val) const
  Set all of the value to be the value `val`.

Protected Member Functions

- umatrix_view_tlate ()
  Empty constructor.

The documentation for this class was generated from the following file:

- umatrix_tlate.h

46.371 uniform_grid< data_t > Class Template Reference

A class representing a uniform linear or logarithmic grid.

`#include <uniform_grid.h>`

Inheritance diagram for uniform_grid< data_t >:

46.371.1 Detailed Description

template<class data_t = double>class uniform_grid< data_t >

Note

This class has no public constructors and is to be instantiated through its children.

Definition at line 37 of file uniform_grid.h.

Public Member Functions

- size_t get_nbins ()
  Get the number of bins (regions in between grid points)
- size_t get_npoints ()
  Get the number of points in the grid (always get_nbins()+1)
- bool is_log ()
  Return true if the grid is logarithmic.
- template<class vec_t >
  void vector (vec_t &v)
  Fill a vector with the specified grid.
- const data_t operator[ ] (size_t i) const
  Get the grid point with index `i` ($i \in [0, \mathrm{n_bins}]$)

Protected Member Functions

- uniform_grid (data_t start, data_t end, data_t width, size_t n_bins, bool log=false)
  Construct a grid with specified values.

Protected Attributes

- data_t g_start
  The low-side of the first bin.
- data_t g_end
  The high-side of the last bin.
- data_t g_width
  The width of each bin.
- size_t g_n_bins
  The number of bins.
- bool g_log
  If true, use a logarithmic scale.

46.371.2 Constructor & Destructor Documentation

46.371.2.1 template<class data_t = double> uniform_grid< data_t >::uniform_grid (data_t *start*, data_t *end*, data_t *width*, size_t *n_bins*, bool *log* = `false`) [inline, protected]

Note

This function is not public because it might create grids that are non-sensical. We require users to create grid objects using one of the children which don't allow non-sensical grids.

Definition at line 64 of file uniform_grid.h.

The documentation for this class was generated from the following file:

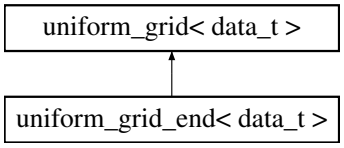
- uniform_grid.h

46.372 uniform_grid_end< data_t > Class Template Reference

Linear grid with fixed number of bins and fixed endpoint.

`#include <uniform_grid.h>`

Inheritance diagram for uniform_grid_end< data_t >:



46.372.1 Detailed Description

template<class data_t = double>class uniform_grid_end< data_t >

Definition at line 159 of file uniform_grid.h.

Public Member Functions

- **uniform_grid_end** (data_t start, data_t end, size_t n_bins)

The documentation for this class was generated from the following file:

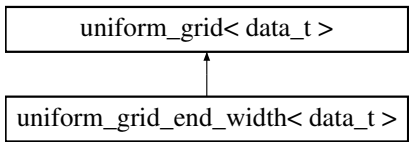
- uniform_grid.h

46.373 uniform_grid_end_width< data_t > Class Template Reference

Linear grid with fixed endpoint and fixed bin size.

`#include <uniform_grid.h>`

Inheritance diagram for uniform_grid_end_width< data_t >:



46.373.1 Detailed Description

template<class data_t = double>class uniform_grid_end_width< data_t >

Definition at line 180 of file uniform_grid.h.

Public Member Functions

- **uniform_grid_end_width** (data_t start, data_t end, data_t width)

The documentation for this class was generated from the following file:

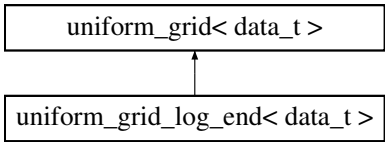
- uniform_grid.h

46.374 uniform_grid_log_end< data_t > Class Template Reference

Logarithmic grid with fixed number of bins and fixed endpoint.

`#include <uniform_grid.h>`

Inheritance diagram for uniform_grid_log_end< data_t >:



46.374.1 Detailed Description

template<class data_t = double>class uniform_grid_log_end< data_t >

Definition at line 191 of file uniform_grid.h.

Public Member Functions

- **uniform_grid_log_end** (data_t start, data_t end, size_t n_bins)

The documentation for this class was generated from the following file:

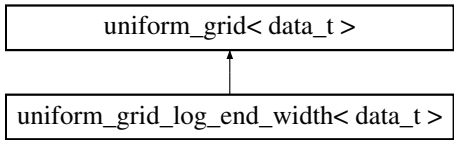
- uniform_grid.h

46.375 uniform_grid_log_end_width< data_t > Class Template Reference

Logarithmic grid with fixed endpoint and fixed bin size.

`#include <uniform_grid.h>`

Inheritance diagram for uniform_grid_log_end_width< data_t >:



46.375.1 Detailed Description

template<class data_t = double>class uniform_grid_log_end_width< data_t >

Definition at line 213 of file uniform_grid.h.

Public Member Functions

- **uniform_grid_log_end_width** (data_t start, data_t end, data_t width)

The documentation for this class was generated from the following file:

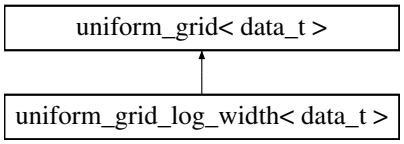
- uniform_grid.h

46.376 uniform_grid_log_width< data_t > Class Template Reference

Logarithmic grid with fixed number of bins and fixed bin size.

`#include <uniform_grid.h>`

Inheritance diagram for uniform_grid_log_width< data_t >:



46.376.1 Detailed Description

template<class data_t = double>class uniform_grid_log_width< data_t >

Definition at line 202 of file uniform_grid.h.

Public Member Functions

- **uniform_grid_log_width** (data_t start, data_t width, size_t n_bins)

The documentation for this class was generated from the following file:

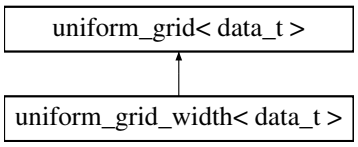
- uniform_grid.h

46.377 uniform_grid_width< data_t > Class Template Reference

Linear grid with fixed number of bins and fixed bin size.

`#include <uniform_grid.h>`

Inheritance diagram for uniform_grid_width< data_t >:



46.377.1 Detailed Description

template<class data_t = double>class uniform_grid_width< data_t >

Definition at line 170 of file uniform_grid.h.

Public Member Functions

- **uniform_grid_width** (data_t start, data_t width, size_t n_bins)

The documentation for this class was generated from the following file:

- uniform_grid.h

46.378 convert_units::unit_t Struct Reference

The type for caching unit conversions.

`#include <convert_units.h>`

46.378.1 Detailed Description

Definition at line 85 of file convert_units.h.

Data Fields

- std::string f
  The input unit.
- std::string t
  The output unit.
- double c
  The conversion factor.

The documentation for this struct was generated from the following file:

- convert_units.h

46.379 o2scl_linalg::uvector_2_mem Class Reference

Allocation object for 2 arrays of equal size.

`#include <tridiag_base.h>`

46.379.1 Detailed Description

This class is used in solve_tridiag_nonsym().

Definition at line 131 of file tridiag_base.h.

Public Member Functions

- void **allocate** (size_t n)
- void **free** ()

Data Fields

- o2scl::uvector **v1**
- o2scl::uvector **v2**

The documentation for this class was generated from the following file:

- tridiag_base.h

46.380 o2scl_linalg::uvector_4_mem Class Reference

Allocation object for 4 arrays of equal size.

`#include <tridiag_base.h>`

46.380.1 Detailed Description

This class is used in solve_tridiag_sym() and solve_cyc_tridiag_nonsym().

Definition at line 149 of file tridiag_base.h.

Public Member Functions

- void **allocate** (size_t n)
- void **free** ()

Data Fields

- o2scl::uvector **v1**
- o2scl::uvector **v2**
- o2scl::uvector **v3**
- o2scl::uvector **v4**

The documentation for this class was generated from the following file:

- tridiag_base.h

46.381 o2scl_linalg::uvector_5_mem Class Reference

Allocation object for 5 arrays of equal size.

`#include <tridiag_base.h>`

46.381.1 Detailed Description

This class is used in solve_cyc_tridiag_sym().

Definition at line 170 of file tridiag_base.h.

Public Member Functions

- void **allocate** (size_t n)
- void **free** ()

Data Fields

- o2scl::uvector **v1**
- o2scl::uvector **v2**
- o2scl::uvector **v3**
- o2scl::uvector **v4**
- o2scl::uvector **v5**

The documentation for this class was generated from the following file:

- tridiag_base.h

46.382 uvector_alloc Class Reference

A simple class to provide an `allocate()` function for uvector.

`#include <uvector_tlate.h>`

46.382.1 Detailed Description

Definition at line 1020 of file uvector_tlate.h.

Public Member Functions

- void allocate (uvector &o, size_t i)
  Allocate `v` for `i` elements.
- void free (uvector &o)
  Free memory.

The documentation for this class was generated from the following file:

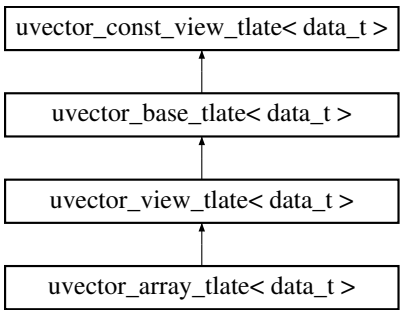
- uvector_tlate.h

46.383 uvector_array_tlate< data_t > Class Template Reference

Create a vector from an array.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_array_tlate< data_t >:



46.383.1 Detailed Description

template<class data_t>class uvector_array_tlate< data_t >

Definition at line 872 of file uvector_tlate.h.

Public Member Functions

- uvector_array_tlate (size_t n, data_t *dat)
  Create a vector from `dat` with size `n`.

The documentation for this class was generated from the following file:

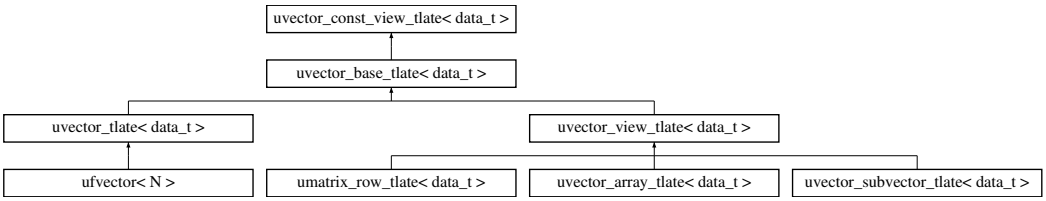
- uvector_tlate.h

46.384 uvector_base_tlate< data_t > Class Template Reference

A base class for uvector and uvector_view.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_base_tlate< data_t >:



46.384.1 Detailed Description

template<class data_t>class uvector_base_tlate< data_t >

This class pruvides a base class for uvector and uvector_view, mostly useful for creating function arguments which accept either uvector or uvector_view.

Definition at line 286 of file uvector_tlate.h.

Public Member Functions

Copy constructors

- uvector_base_tlate (uvector_base_tlate &v)
  Copy constructor - create a new view of the same vector.
- uvector_base_tlate & operator= (uvector_base_tlate &v)
  Copy constructor - create a new view of the same vector.

Get and set methods

- data_t & operator[ ] (size_t i)
  Array-like indexing.
- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.
- data_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- int set (size_t i, data_t val)
  Set (with optional range-checking)
- int set_all (data_t val)
  Set all of the value to be the value `val`.

Arithmetic

- uvector_base_tlate< data_t > & operator+= (const uvector_base_tlate< data_t > &x)
  operator+=
- uvector_base_tlate< data_t > & operator-= (const uvector_base_tlate< data_t > &x)
  operator-=
- uvector_base_tlate< data_t > & operator+= (const data_t &y)
  operator+=
- uvector_base_tlate< data_t > & operator-= (const data_t &y)
  operator-=
- uvector_base_tlate< data_t > & operator*= (const data_t &y)
  operator=*

Protected Member Functions

- uvector_base_tlate ()
  Empty constructor for use by children.

46.384.2 Constructor & Destructor Documentation

46.384.2.1 template<class data_t> uvector_base_tlate< data_t >::uvector_base_tlate ( ) [inline, protected]

Definition at line 473 of file uvector_tlate.h.

The documentation for this class was generated from the following file:

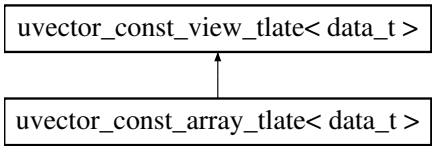
- uvector_tlate.h

46.385 uvector_const_array_tlate< data_t > Class Template Reference

Create a vector from an const array.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_const_array_tlate< data_t >:



46.385.1 Detailed Description

template<class data_t>class uvector_const_array_tlate< data_t >

Definition at line 906 of file uvector_tlate.h.

Public Member Functions

- uvector_const_array_tlate (size_t n, const data_t *dat)
  Create a vector from `dat` with size `n`.

The documentation for this class was generated from the following file:

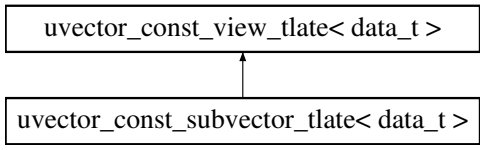
- uvector_tlate.h

46.386 uvector_const_subvector_tlate< data_t > Class Template Reference

Create a const vector from a subvector of another vector.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_const_subvector_tlate< data_t >:



46.386.1 Detailed Description

template<class data_t>class uvector_const_subvector_tlate< data_t >

Definition at line 926 of file uvector_tlate.h.

Public Member Functions

- uvector_const_subvector_tlate (const uvector_base_tlate< data_t > &orig, size_t offset, size_t n)

Create a vector from `orig`.

The documentation for this class was generated from the following file:

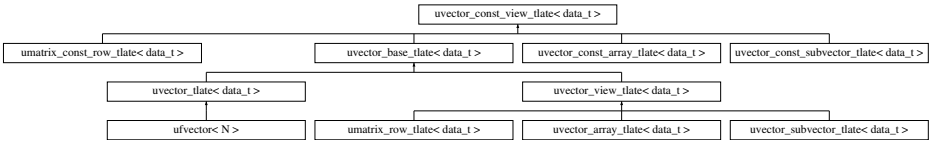
- uvector_tlate.h

46.387 uvector_const_view_tlate< data_t > Class Template Reference

A const vector view with unit stride.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_const_view_tlate< data_t >:



46.387.1 Detailed Description

template<class data_t>class uvector_const_view_tlate< data_t >

Idea for Future Could allow user-defined specification of restrict keyword

Definition at line 64 of file uvector_tlate.h.

Public Member Functions

- data_t norm () const
  Norm.

Copy constructors

- uvector_const_view_tlate (const uvector_const_view_tlate &v)
  Copy constructor - create a new view of the same vector.
- uvector_const_view_tlate & operator= (const uvector_const_view_tlate &v)
  Copy constructor - create a new view of the same vector.

Get and set methods

- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.
- data_t get (size_t i) const
  Get (with optional range-checking)
- const data_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)
- size_t size () const
  Method to return vector size.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.
- size_t lookup (const data_t x0) const
  Exhaustively look through the array for a particular value.
- data_t max () const
  Find the maximum element.
- data_t min () const
  Find the minimum element.

Protected Member Functions

- uvector_const_view_tlate ()
  Empty constructor provided for use by uvector_tlate(const uvector_tlate &v)

Protected Attributes

- data_t * data
  The data.
- size_t sz
  The vector sz.
- int owner
  Zero if memory is owned elsewhere, 1 otherwise.

Friends

- class **uvector_base_tlate< data_t >**
- class **uvector_view_tlate< data_t >**
- class **uvector_tlate< data_t >**
- class **uvector_subvector_tlate< data_t >**
- class **uvector_const_subvector_tlate< data_t >**

46.387.2 Member Function Documentation

46.387.2.1 template<class data_t> size_t uvector_const_view_tlate< data_t >::size ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 176 of file uvector_tlate.h.

46.387.2.2 template<class data_t> size_t uvector_const_view_tlate< data_t >::lookup (const data_t *x0*) const [inline]

This can only fail if *all* of the entries in the array are not finite, in which case it calls O2SCL_ERR() and returns 0.

If more than one entry is the same distance from `x0`, this function returns the entry with smallest index.

Definition at line 200 of file uvector_tlate.h.

46.387.2.3 template<class data_t> data_t uvector_const_view_tlate< data_t >::norm ( ) const [inline]

Todo Fix this so that norm() is computed as in ovector and so that integer norms are performed separately.

Definition at line 259 of file uvector_tlate.h.

The documentation for this class was generated from the following file:

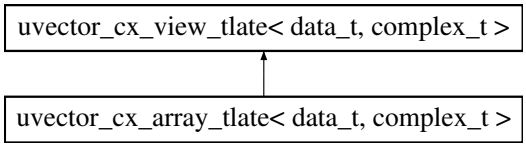
- uvector_tlate.h

46.388 uvector_cx_array_tlate< data_t, complex_t > Class Template Reference

Create a vector from an array.

`#include <uvector_cx_tlate.h>`

Inheritance diagram for uvector_cx_array_tlate< data_t, complex_t >:



46.388.1 Detailed Description

template<class data_t, class complex_t>class uvector_cx_array_tlate< data_t, complex_t >

Definition at line 488 of file uvector_cx_tlate.h.

Public Member Functions

- uvector_cx_array_tlate (size_t n, data_t *dat)
  Create a vector from `dat` with size `n`.

The documentation for this class was generated from the following file:

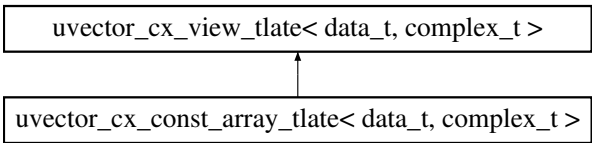
- uvector_cx_tlate.h

46.389 uvector_cx_const_array_tlate< data_t, complex_t > Class Template Reference

Create a vector from an array.

`#include <uvector_cx_tlate.h>`

Inheritance diagram for uvector_cx_const_array_tlate< data_t, complex_t >:



46.389.1 Detailed Description

template<class data_t, class complex_t>class uvector_cx_const_array_tlate< data_t, complex_t >

Definition at line 522 of file uvector_cx_tlate.h.

Public Member Functions

- uvector_cx_const_array_tlate (size_t n, const data_t *dat)
  Create a vector from `dat` with size `n`.

Protected Member Functions

These are inaccessible to ensure the vector is \c const.

- data_t & operator[ ] (size_t i)
  Array-like indexing.
- data_t & operator() (size_t i)
  Array-like indexing.
- data_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- int **set** (size_t i, data_t val)
- int **set_all** (double val)
- uvector_cx_view_tlate< data_t, complex_t > & operator+= (const uvector_cx_view_tlate< data_t, complex_t > &x)
  operator+=
- uvector_cx_view_tlate< data_t, complex_t > & operator-= (const uvector_cx_view_tlate< data_t, complex_t > &x)
  operator-=
- uvector_cx_view_tlate< data_t, complex_t > & operator*= (const data_t &y)
  operator=*

The documentation for this class was generated from the following file:

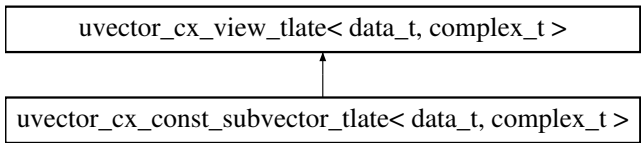
- uvector_cx_tlate.h

46.390 uvector_cx_const_subvector_tlate< data_t, complex_t > Class Template Reference

Create a vector from a subvector of another.

`#include <uvector_cx_tlate.h>`

Inheritance diagram for uvector_cx_const_subvector_tlate< data_t, complex_t >:



46.390.1 Detailed Description

template<class data_t, class complex_t>class uvector_cx_const_subvector_tlate< data_t, complex_t >

Definition at line 570 of file uvector_cx_tlate.h.

Public Member Functions

- uvector_cx_const_subvector_tlate (const uvector_cx_view_tlate< data_t, complex_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

The documentation for this class was generated from the following file:

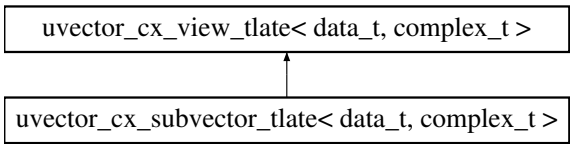
- uvector_cx_tlate.h

46.391 uvector_cx_subvector_tlate< data_t, complex_t > Class Template Reference

Create a vector from a subvector of another.

`#include <uvector_cx_tlate.h>`

Inheritance diagram for uvector_cx_subvector_tlate< data_t, complex_t >:



46.391.1 Detailed Description

template<class data_t, class complex_t>class uvector_cx_subvector_tlate< data_t, complex_t >

Definition at line 503 of file uvector_cx_tlate.h.

Public Member Functions

- uvector_cx_subvector_tlate (uvector_cx_view_tlate< data_t, complex_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

The documentation for this class was generated from the following file:

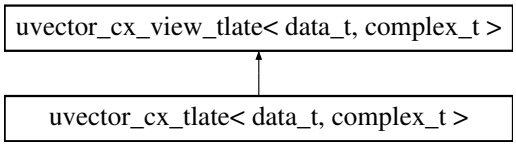
- uvector_cx_tlate.h

46.392 uvector_cx_tlate< data_t, complex_t > Class Template Reference

A vector of double-precision numbers with unit stride.

`#include <uvector_cx_tlate.h>`

Inheritance diagram for uvector_cx_tlate< data_t, complex_t >:



46.392.1 Detailed Description

template<class data_t, class complex_t>class uvector_cx_tlate< data_t, complex_t >

There is also an << operator for this class documented "Functions" section of uvector_tlate.h.

Definition at line 313 of file uvector_cx_tlate.h.

Public Member Functions

Standard constructor

- uvector_cx_tlate (size_t n=0)
  Create an uvector_cx of size n with owner as 'true'.

Copy constructors

- uvector_cx_tlate (const uvector_cx_tlate &v)
  Deep copy constructor - allocate new space and make a copy.
- uvector_cx_tlate (const uvector_cx_view_tlate< data_t, complex_t > &v)
  Deep copy constructor - allocate new space and make a copy.
- uvector_cx_tlate & operator= (const uvector_cx_tlate &v)
  Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- uvector_cx_tlate & operator= (const uvector_cx_view_tlate< data_t, complex_t > &v)
  Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.

Memory allocation

- int allocate (size_t nsize)
  Allocate memory for size n after freeing any memory presently in use.
- int free ()
  Free the memory.

46.392.2 Member Function Documentation

46.392.2.1 template<class data_t , class complex_t > int uvector_cx_tlate< data_t, complex_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 473 of file uvector_cx_tlate.h.

The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

46.393 uvector_cx_view_tlate< data_t, complex_t > Class Template Reference

A vector view of complex numbers with unit stride.

`#include <uvector_cx_tlate.h>`

Inheritance diagram for uvector_cx_view_tlate< data_t, complex_t >:

46.393.1 Detailed Description

template<class data_t, class complex_t>class uvector_cx_view_tlate< data_t, complex_t >

Idea for Future Write lookup() method, and possibly an erase() method.

Definition at line 47 of file uvector_cx_tlate.h.

Public Member Functions

Copy constructors

- uvector_cx_view_tlate (const uvector_cx_view_tlate &v)
  Copy constructor - create a new view of the same vector.
- uvector_cx_view_tlate & operator= (const uvector_cx_view_tlate &v)
  Copy constructor - create a new view of the same vector.

Get and set methods

- complex_t & operator[ ] (size_t i)
  Array-like indexing.
- const complex_t & operator[ ] (size_t i) const
  Array-like indexing.
- complex_t & operator() (size_t i)
  Array-like indexing.
- const complex_t & operator() (size_t i) const
  Array-like indexing.
- complex_t get (size_t i) const
  Get (with optional range-checking)
- complex_t * get_ptr (size_t i)
  Get pointer (with optional range-checking)
- const complex_t * get_const_ptr (size_t i) const
  Get pointer (with optional range-checking)
- int set (size_t i, const complex_t &val)
  Set (with optional range-checking)
- int set_all (const complex_t &val)
  Set all of the value to be the value `val`.
- size_t size () const
  Method to return vector size.

Other methods

- bool is_owner () const
  Return true if this object owns the data it refers to.

Arithmetic

- uvector_cx_view_tlate< data_t, complex_t > & operator+= (const uvector_cx_view_tlate< data_t, complex_t > &x)
  operator+=
- uvector_cx_view_tlate< data_t, complex_t > & operator-= (const uvector_cx_view_tlate< data_t, complex_t > &x)
  operator-=
- uvector_cx_view_tlate< data_t, complex_t > & operator+= (const data_t &y)
  operator+=
- uvector_cx_view_tlate< data_t, complex_t > & operator-= (const data_t &y)
  operator-=
- uvector_cx_view_tlate< data_t, complex_t > & operator*= (const data_t &y)
  operator=*
- data_t norm () const
  Norm.

Protected Member Functions

- uvector_cx_view_tlate ()
  Empty constructor provided for use by uvector_cx_tlate(const uvector_cx_tlate &v)

Protected Attributes

- data_t * data
  The data.
- size_t sz
  The vector sz.
- int owner
  Zero if memory is owned elsewhere, 1 otherwise.

46.393.2 Member Function Documentation

46.393.2.1 template<class data_t, class complex_t> size_t uvector_cx_view_tlate< data_t, complex_t >::size ( ) const [inline]

If no memory has been allocated, this will quietly return zero.

Definition at line 228 of file uvector_cx_tlate.h.

The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

46.394 uvector_int_alloc Class Reference

A simple class to provide an `allocate()` function for uvector_int.

`#include <uvector_tlate.h>`

46.394.1 Detailed Description

Definition at line 1031 of file uvector_tlate.h.

Public Member Functions

- void allocate (uvector_int &o, size_t i)
  Allocate `v` for `i` elements.
- void free (uvector_int &o)
  Free memory.

The documentation for this class was generated from the following file:

- uvector_tlate.h

46.395 uvector_size_t_alloc Class Reference

A simple class to provide an `allocate()` function for uvector_size_t.

`#include <uvector_tlate.h>`

46.395.1 Detailed Description

Definition at line 1042 of file uvector_tlate.h.

Public Member Functions

- void allocate (uvector_size_t &o, size_t i)
  Allocate `v` for `i` elements.
- void free (uvector_size_t &o)
  Free memory.

The documentation for this class was generated from the following file:

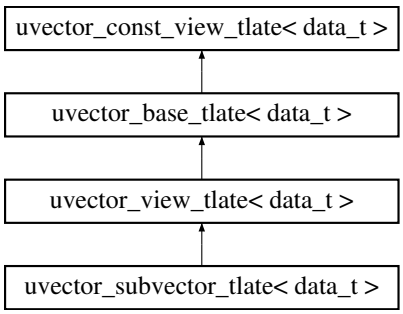
- uvector_tlate.h

46.396 uvector_subvector_tlate< data_t > Class Template Reference

Create a vector from a subvector of another.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_subvector_tlate< data_t >:



46.396.1 Detailed Description

template<class data_t>class uvector_subvector_tlate< data_t >

Definition at line 887 of file uvector_tlate.h.

Public Member Functions

- uvector_subvector_tlate (uvector_base_tlate< data_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

The documentation for this class was generated from the following file:

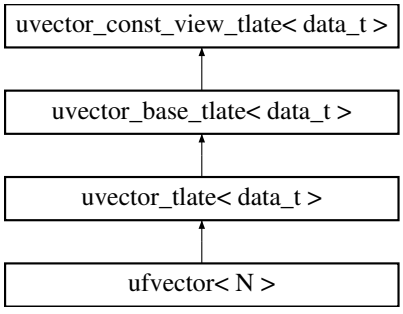
- uvector_tlate.h

46.397 uvector_tlate< data_t > Class Template Reference

A vector with unit stride.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_tlate< data_t >:



46.397.1 Detailed Description

template<class data_t>class uvector_tlate< data_t >

There are several useful methods which are defined in the parent class, uvector_view_tlate . There is also an << operator for this class documented "Functions" section of uvector_tlate.h.

Definition at line 621 of file uvector_tlate.h.

Public Member Functions

- int sort_unique ()
  Sort the vector and ensure all elements are unique by removing duplicates.

Standard constructor

- uvector_tlate (size_t n=0)
  Create an uvector of size `n` with owner as 'true'.

Copy constructors

- uvector_tlate (const uvector_tlate &v)
  Deep copy constructor - allocate new space and make a copy.
- uvector_tlate (const uvector_const_view_tlate< data_t > &v)
  Deep copy constructor - allocate new space and make a copy.
- uvector_tlate & operator= (const uvector_tlate &v)
  Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.
- uvector_tlate & operator= (const uvector_const_view_tlate< data_t > &v)
  Deep copy constructor - if owner is true, allocate space and make a new copy, otherwise, just copy into the view.

Memory allocation

- int allocate (size_t nsize)
  Allocate memory for size `n` after freeing any memory presently in use.
- int free ()
  Free the memory.

Other methods

- int erase (size_t ix)
  Erase an element from the array.

Friends

- class **uvector_view_tlate< data_t >**

46.397.2 Member Function Documentation

46.397.2.1 template<class data_t> int uvector_tlate< data_t >::allocate (size_t *nsize*) [inline]

If `nsize` is zero, this only frees the memory and allocates no additional memory.

Definition at line 784 of file uvector_tlate.h.

46.397.2.2 template<class data_t> int uvector_tlate< data_t >::free ( ) [inline]

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 805 of file uvector_tlate.h.

The documentation for this class was generated from the following file:

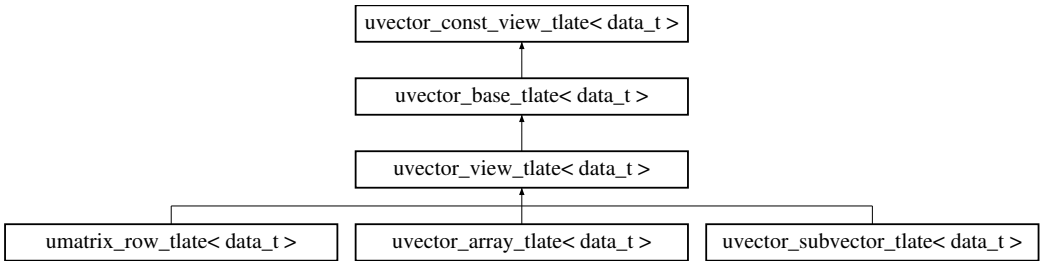
- uvector_tlate.h

46.398 uvector_view_tlate< data_t > Class Template Reference

A base class for uvectors.

`#include <uvector_tlate.h>`

Inheritance diagram for uvector_view_tlate< data_t >:



46.398.1 Detailed Description

template<class data_t>class uvector_view_tlate< data_t >

Definition at line 485 of file uvector_tlate.h.

Public Member Functions

Copy constructors

- uvector_view_tlate (const uvector_view_tlate &v)
  Copy constructor - create a new view of the same vector.
- uvector_view_tlate & operator= (const uvector_view_tlate &v)
  Copy constructor - create a new view of the same vector.
- uvector_view_tlate (uvector_base_tlate< data_t > &v)
  Copy constructor - create a new view of the same vector.
- uvector_view_tlate & operator= (uvector_base_tlate< data_t > &v)
  Copy constructor - create a new view of the same vector.

Get and set methods

- data_t & operator[ ] (size_t i) const
  Array-like indexing.
- data_t & operator() (size_t i) const
  Array-like indexing.
- data_t * get_ptr (size_t i) const
  Get pointer (with optional range-checking)
- int set (size_t i, data_t val) const
  Set (with optional range-checking)
- int set_all (data_t val) const
  Set all of the value to be the value `val`.

Protected Member Functions

- uvector_view_tlate ()
  Empty constructor provided for use by uvector_view_tlate(const uvector_view_tlate &v)

The documentation for this class was generated from the following file:

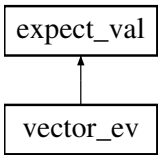
- uvector_tlate.h

46.399 vector_ev Class Reference

Vector expectation value.

`#include <expect_val.h>`

Inheritance diagram for vector_ev:



46.399.1 Detailed Description

See expect_val for some general notes on this and related classes.

This class is experimental.

This is a similar to scalar_ev, except that it allows updating and statistics for a set of scalars en masse. The data is stored internally in uvector and umatrix objects, but the public member functions operate with template types which are compatible with any vector class which provides `double &operator[]`.

Internally, data is stored in matrices designating the first (row) index as the user-specified vector index and designating the second (column) index as the block index.

Definition at line 325 of file expect_val.h.

Public Member Functions

- vector_ev (size_t n)
  Create for a vector of size `n` with no blocking.
- vector_ev (size_t n, size_t n_blocks, size_t n_per_block)
  Create for a vector of size `n` with `n_blocks` blocks and `n_per_block` points block.
- virtual void set_blocks (size_t n, size_t n_blocks, size_t n_per_block)
  Set for a vector of size `n` with `n_blocks` blocks and `n_per_block` points block.
- vector_ev (const vector_ev &ev)
  Copy constructor.
- vector_ev & operator= (const vector_ev &ev)
  Copy constructor.
- virtual void free ()
  Free allocated data.
- virtual void reset ()
  Clear all the data.
- template<class vec_t >
  void add (vec_t &v)
  Add measurement of value `val`.

Report statistics

- template<class vec_t , class vec2_t , class vec3_t >
  void current_avg_stats (vec_t &avg, vec2_t &std_dev, vec3_t &avg_err, size_t &m_block, size_t &m_per_block) const
  Report current average, standard deviation, and the error in the average and include block information.
- template<class vec_t , class vec2_t , class vec3_t >
  void current_avg (vec_t &avg, vec2_t &std_dev, vec3_t &avg_err) const
  Report current average, standard deviation, and the error in the average.
- template<class vec_t , class vec2_t , class vec3_t >
  void reblock_avg_stats (size_t new_blocks, vec_t &avg, vec2_t &std_dev, vec3_t &avg_err, size_t &m_per_block) const

Report average, standard deviation, and the error in the average assuming a new block size.

- template<class vec_t , class vec2_t , class vec3_t >
  void reblock_avg (size_t new_blocks, vec_t &avg, vec2_t &std_dev, vec3_t &avg_err) const
  Report average, standard deviation, and the error in the average assuming a new block size.

Protected Attributes

- umatrix last
  The most recent values for each block.
- umatrix vals
  The average for each block.
- uvector current
  The current rolling average.
- size_t nvec
  The size of the vector.

The documentation for this class was generated from the following file:

- expect_val.h

46.400 xmatrix Class Reference

A version of omatrix with better error checking.

`#include <omatrix_tlate.h>`

Inherits omatrix_tlate< double, gsl_matrix, gsl_vector_norm, gsl_block >.

46.400.1 Detailed Description

This demonstrates how operator[] could return an ovector_array object and thus provide more bounds-checking. This would demand including a new parameter in omatrix_view_tlate which contains the vector type.

Definition at line 1352 of file omatrix_tlate.h.

Public Member Functions

- **xmatrix** (size_t r=0, size_t c=0)
- ovector_array operator[ ] (size_t i)
  Array-like indexing.
- const ovector_const_array operator[ ] (size_t i) const
  Array-like indexing.

The documentation for this class was generated from the following file:

- omatrix_tlate.h

47 File Documentation

47.1 array.h File Reference

Various array classes.

`#include <iostream> #include <cmath> #include <string> #include <fstream> #include <sstream> × #include <vector> #include <map> #include <o2scl/err_hnd.h> #include <gsl/gsl_ieee_utils.- h> #include <gsl/gsl_sort.h>`

47.1.1 Detailed Description

For a more general discussion of vectors and matrices in O_2scl , see the Arrays, Vectors, Matrices and Tensors of the User's Guide.

This file contains classes and functions for operating with C-style 1- or 2-dimensional arrays and pointers to double. For an example of the usage of the array allocation classes, see the Multi-dimensional solver . For more generic operations on generic vector objects (including in some cases C-style arrays), see also the file vector.h .

This file contains the allocation classes

- array_alloc
- array_2d_alloc
- pointer_alloc
- pointer_2d_alloc

the classes for the manipulation of arrays in smart_interp

- array_reverse
- array_subvector
- array_subvector_reverse
- array_const_reverse
- array_const_subvector
- array_const_subvector_reverse

the array equivalent of omatrix_row and omatrix_col (see usage in `src/ode/ode_it_solve_ts.cpp`)

- array_2d_row
- array_2d_col

Note

The classes

- array_reverse
- array_subvector
- array_subvector_reverse
- array_const_reverse
- array_const_subvector
- array_const_subvector_reverse

can be used with pointers or arrays, but array_alloc and pointer_alloc are *not* interchangable.

Idea for Future Create a class which views a C-style array as a matrix and offers an `operator(,)`

Definition in file array.h.

Data Structures

- class array_alloc< vec_t >
  A simple class to provide an `allocate()` function for arrays.
- class array_2d_alloc< mat_t >
  A simple class to provide an `allocate()` function for 2-dimensional arrays.
- class pointer_alloc< base_t >
  A simple class to provide an `allocate()` function for pointers.
- class pointer_2d_alloc< base_t >
  A simple class to provide an `allocate()` function for pointers.
- struct pointer_2d_alloc< base_t >::pointer_comp
- class array_reverse< sz >
  A simple class which reverses the order of an array.
- class array_const_reverse< sz >
  A simple class which reverses the order of an array.
- class array_subvector
  A simple subvector class for an array (without error checking)
- class array_2d_col< R, C, data_t >
  Column of a 2d array.
- class array_2d_row< array_2d_t, data_t >
  Row of a 2d array.
- class array_const_subvector
  A simple subvector class for a const array (without error checking)
- class array_subvector_reverse
  Reverse a subvector of an array.
- class array_const_subvector_reverse
  Reverse a subvector of a const array.

47.2 cblas_base.h File Reference

O2scl basic linear algebra function templates.

47.2.1 Detailed Description

See o2scl_cblas for more documentation on these functions.

Idea for Future Add float and complex versions.

Idea for Future There are some Level-1 BLAS functions which are already present in vector.h. Should we move all of them to one place or the other? The ones in vector.h are generic in the sense that they can use doubles or floats, but the ones here can use either () or [].

Definition in file cblas_base.h.

Namespaces

- namespace o2scl_cblas
  Namespace for O2scl CBLAS function templates with operator[].

Enumerations

- enum o2scl_cblas::o2cblas_order { **o2cblas_RowMajor** = 101, **o2cblas_ColMajor** = 102 }
  Matrix order, either column-major or row-major.
- enum o2scl_cblas::o2cblas_transpose { **o2cblas_NoTrans** = 111, **o2cblas_Trans** = 112, **o2cblas_ConjTrans** = 113 }
  Transpose operations.

- enum o2scl_cblas::o2cblas_uplo { **o2cblas_Upper** = 121, **o2cblas_Lower** = 122 }
  Upper- or lower-triangular.
- enum o2scl_cblas::o2cblas_diag { **o2cblas_NonUnit** = 131, **o2cblas_Unit** = 132 }
  Unit or generic diagonal.
- enum o2scl_cblas::o2cblas_side { **o2cblas_Left** = 141, **o2cblas_Right** = 142 }
  Left or right sided operation.

Functions

Level-1 BLAS functions

- template<class vec_t >
  double o2scl_cblas::dasum (const size_t N, const vec_t &X)
  Compute the absolute sum of vector elements.
- template<class vec_t , class vec2_t >
  void o2scl_cblas::daxpy (const double alpha, const size_t N, const vec_t &X, vec2_t &Y)
  Compute $y = \alpha x + y$.
- template<class vec_t , class vec2_t >
  double o2scl_cblas::ddot (const size_t N, const vec_t &X, const vec2_t &Y)
  Compute $r = x \cdot y$.
- template<class vec_t >
  double o2scl_cblas::dnrm2 (const size_t N, const vec_t &X)
  Compute the norm of the vector `X`.
- template<class vec_t >
  void o2scl_cblas::dscal (const double alpha, const size_t N, vec_t &X)
  Compute $x = \alpha x$.

Level-2 BLAS functions

- template<class mat_t , class vec_t >
  void o2scl_cblas::dgemv (const enum o2cblas_order order, const enum o2cblas_transpose TransA, const size_t M, const size_t N, const double alpha, const mat_t &A, const vec_t &X, const double beta, vec_t &Y)
  Compute $y = \alpha [\mathrm{op}(A)] x + \beta y$.
- template<class mat_t , class vec_t >
  void o2scl_cblas::dtrsv (const enum o2cblas_order order, const enum o2cblas_uplo Uplo, const enum o2cblas_transpose TransA, const enum o2cblas_diag Diag, const size_t M, const size_t N, const mat_t &A, vec_t &X)
  Compute $x = \mathrm{op}(A)^{-1} x$.
- template<class mat_t , class vec_t >
  void o2scl_cblas::dtrmv (const enum o2cblas_order Order, const enum o2cblas_uplo Uplo, const enum o2cblas_transpose TransA, const enum o2cblas_diag Diag, const size_t N, const mat_t &A, vec_t &x)
  Compute $x = op(A)x$ *for the triangular matrix* `A`.

Level-3 BLAS functions

- template<class mat_t >
  void o2scl_cblas::dgemm (const enum o2cblas_order Order, const enum o2cblas_transpose TransA, const enum o2cblas_transpose TransB, const size_t M, const size_t N, const size_t K, const double alpha, const mat_t &A, const mat_t &B, const double beta, mat_t &C)
  Compute $y = \alpha \mathrm{op}(A) \mathrm{op}(B) + \beta C$.

Helper BLAS functions - Subvectors

- template<class vec_t , class vec2_t >
  void o2scl_cblas::daxpy_subvec (const double alpha, const size_t N, const vec_t &X, vec2_t &Y, const size_t ie)
  Compute $y = \alpha x + y$ *beginning with index* `ie` *and ending with index* `N-1`.
- template<class vec_t , class vec2_t >
  double o2scl_cblas::ddot_subvec (const size_t N, const vec_t &X, const vec2_t &Y, const size_t ie)
  Compute $r = x \cdot y$ *beginning with index* `ie` *and ending with index* `N-1`.
- template<class vec_t >
  double o2scl_cblas::dnrm2_subvec (const size_t N, const vec_t &X, const size_t ie)

Compute the norm of the vector X beginning with index ie and ending with index N-1.

- template<class vec_t >
  void o2scl_cblas::dscal_subvec (const double alpha, const size_t N, vec_t &X, const size_t ie)
  Compute $x = \alpha x$ beginning with index ie and ending with index N-1.

Helper BLAS functions - Subcolums of a matrix

- template<class mat_t , class vec_t >
  void o2scl_cblas::daxpy_subcol (const double alpha, const size_t M, const mat_t &X, const size_t ir, const size_t ic, vec_t &y)
  Compute $y = \alpha x + y$ for a subcolumn of a matrix.
- template<class mat_t , class vec_t >
  double o2scl_cblas::ddot_subcol (const size_t M, const mat_t &X, const size_t ir, const size_t ic, const vec_t &y)
  Compute $r = x \cdot y$ for a subcolumn of a matrix.
- template<class mat_t >
  double o2scl_cblas::dnrm2_subcol (const mat_t &A, const size_t ir, const size_t ic, const size_t M)
  Compute the norm of a subcolumn of a matrix.
- template<class mat_t >
  void o2scl_cblas::dscal_subcol (mat_t &A, const size_t ir, const size_t ic, const size_t M, const double alpha)
  Compute $x = \alpha x$ for a subcolumn of a matrix.

Helper BLAS functions - Subrows of a matrix

- template<class mat_t , class vec_t >
  void o2scl_cblas::daxpy_subrow (const double alpha, const size_t N, const mat_t &X, const size_t ir, const size_t ic, vec_t &Y)
  Compute $y = \alpha x + y$ for a subrow of a matrix.
- template<class mat_t , class vec_t >
  double o2scl_cblas::ddot_subrow (const size_t N, const mat_t &X, const size_t ir, const size_t ic, const vec_t &Y)
  Compute $r = x \cdot y$ for a subrow of a matrix.
- template<class mat_t >
  double o2scl_cblas::dnrm2_subrow (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
  Compute the norm of a subrow of a matrix.
- template<class mat_t >
  void o2scl_cblas::dscal_subrow (mat_t &A, const size_t ir, const size_t ic, const size_t N, const double alpha)
  Compute $x = \alpha x$ for a subrow of a matrix.

47.3 columnify.h File Reference

Functions to create output in columns.

`#include <iostream> #include <string> #include <vector> #include <o2scl/misc.h> #include <o2scl/array.h>`

47.3.1 Detailed Description

Definition in file columnify.h.

Data Structures

- class columnify
  Create nicely formatted columns from a table of strings.

Functions

- template<class mat_t >
  int matrix_out_paren (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)
  A operator for simple matrix output using operator()

- template<class mat_t >
  int matrix_cx_out_paren (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)
  A operator for simple complex matrix output using `operator()`
- template<class mat_t >
  int matrix_out (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)
  A operator for simple matrix output using `operator[]`.

47.3.2 Function Documentation

47.3.2.1 template<class mat_t > int matrix_out_paren (std::ostream & *os*, mat_t & *A*, size_t *nrows*, size_t *ncols*)

The type `mat_t` can be any matrix type which allows individual element access using `operator()(size_t,size_t)`.

This outputs all of the matrix elements using output settings specified by `os`. The alignment performed by columnify using columnify::align_dp, i.e. the numbers are aligned by their decimal points. If the numbers have no decimal points, then the decimal point is assumed to be to the right of the last character in the string represetation of the number.

Definition at line 245 of file columnify.h.

47.3.2.2 template<class mat_t > int matrix_cx_out_paren (std::ostream & *os*, mat_t & *A*, size_t *nrows*, size_t *ncols*)

Todo Doesn't this only work for GSL matrices? Compare this to the corresponding vector functions

Definition at line 278 of file columnify.h.

47.3.2.3 template<class mat_t > int matrix_out (std::ostream & *os*, mat_t & *A*, size_t *nrows*, size_t *ncols*)

The type `mat_t` can be any 2d-array type which allows individual element access using [size_t][size_t]

This outputs all of the matrix elements using output settings specified by `os`. The alignment performed by columnify using columnify::align_dp, i.e. the numbers are aligned by their decimal points. If the numbers have no decimal points, then the decimal point is assumed to be to the right of the last character in the string represetation of the number.

Idea for Future If all of the matrix elements are positive integers and scientific mode is not set, then we can avoid printing the extra spaces.

Definition at line 323 of file columnify.h.

47.4 cx_arith.h File Reference

Complex arithmetic.

```
#include <iostream> #include <complex> #include <cmath> #include <gsl/gsl_math.h> #include
<gsl/gsl_complex.h> #include <gsl/gsl_complex_math.h>
```

47.4.1 Detailed Description

Note

One should be careful about including this header file, especially in other header files as it overloads some of the standard mathematical functions, i.e. `sqrt()`. If you need access to both these functions and the standard functions for `double` objects, the easiest way is probably to include this file in a source (not header file) and use `using namespace std`.
This used to be in a separate namespace, called `o2scl_arith`, but this causes problems with Koenig lookup in template classes for operator*() when defined for vector addition (for example).

Idea for Future Define operators with assignment for complex + double?

Definition in file cx_arith.h.

Functions

- gsl_complex complex_to_gsl (std::complex< double > &d)
  Convert a complex number to GSL form.
- std::complex< double > gsl_to_complex (gsl_complex &g)
  Convert a complex number to STL form.

Binary operators for two complex numbers

- gsl_complex operator+ (gsl_complex x, gsl_complex y)
  Add two complex numbers.
- gsl_complex operator- (gsl_complex x, gsl_complex y)
  Subtract two complex numbers.
- gsl_complex operator* (gsl_complex x, gsl_complex y)
  Multiply two complex numbers.
- gsl_complex operator/ (gsl_complex x, gsl_complex y)
  Divide two complex numbers.

Binary operators with assignment for two complex numbers

- gsl_complex operator+= (gsl_complex &x, gsl_complex y)
  Add a complex number.
- gsl_complex operator-= (gsl_complex &x, gsl_complex y)
  Subtract a complex number.
- gsl_complex operator*= (gsl_complex &x, gsl_complex y)
  Multiply a complex number.
- gsl_complex operator/= (gsl_complex &x, gsl_complex y)
  Divide a complex number.

Binary operators with assignment for a complex and real

- gsl_complex operator+ (gsl_complex x, double y)
  Add a complex and real number.
- gsl_complex operator+ (double y, gsl_complex x)
  Add a complex and real number.
- gsl_complex operator- (gsl_complex x, double y)
  Subtract a complex and real number.
- gsl_complex operator- (double y, gsl_complex x)
  Subtract a complex and real number.
- gsl_complex operator* (gsl_complex x, double y)
  Multiply a complex and real number.
- gsl_complex operator* (double y, gsl_complex x)
  Multiply a complex and real number.
- gsl_complex operator/ (gsl_complex x, double y)
  Divide a complex and real number.

Miscellaneous functions

- double **arg** (gsl_complex x)
- double **abs** (gsl_complex x)
- double **abs2** (gsl_complex z)
- gsl_complex **conjugate** (gsl_complex a)

Square root and exponent functions

- gsl_complex **sqrt** (gsl_complex a)
- gsl_complex **sqrt_real** (double x)
- gsl_complex **pow** (gsl_complex a, gsl_complex b)
- gsl_complex **pow_real** (gsl_complex a, double b)

Logarithmic and exponential functions

- double **logabs** (gsl_complex z)
- gsl_complex **exp** (gsl_complex a)
- gsl_complex **log** (gsl_complex a)
- gsl_complex **log10** (gsl_complex a)
- gsl_complex **log_b** (gsl_complex a, gsl_complex b)

Trigonometric functions

- gsl_complex **sin** (gsl_complex a)
- gsl_complex **cos** (gsl_complex a)
- gsl_complex **tan** (gsl_complex a)
- gsl_complex **sec** (gsl_complex a)
- gsl_complex **csc** (gsl_complex a)
- gsl_complex **cot** (gsl_complex a)
- gsl_complex **asin** (gsl_complex a)
- gsl_complex **asin_real** (double a)
- gsl_complex **acos** (gsl_complex a)
- gsl_complex **acos_real** (double a)
- gsl_complex **atan** (gsl_complex a)
- gsl_complex **asec** (gsl_complex a)
- gsl_complex **asec_real** (double a)
- gsl_complex **acsc** (gsl_complex a)
- gsl_complex **acsc_real** (double a)
- gsl_complex **acot** (gsl_complex a)

Hyperbolic trigonometric functions

- gsl_complex **sinh** (gsl_complex a)
- gsl_complex **cosh** (gsl_complex a)
- gsl_complex **tanh** (gsl_complex a)
- gsl_complex **sech** (gsl_complex a)
- gsl_complex **csch** (gsl_complex a)
- gsl_complex **coth** (gsl_complex a)
- gsl_complex **asinh** (gsl_complex a)
- gsl_complex **acosh** (gsl_complex a)
- gsl_complex **acosh_real** (double a)
- gsl_complex **atanh** (gsl_complex a)
- gsl_complex **atanh_real** (double a)
- gsl_complex **asech** (gsl_complex a)
- gsl_complex **acsch** (gsl_complex a)
- gsl_complex **acoth** (gsl_complex a)

47.5 err_hnd.h File Reference

File for definitions for err_hnd_type and err_hnd_gsl.

`#include <iostream>` `#include <string>` `#include <gsl/gsl_errno.h>`

47.5.1 Detailed Description

See also exception.h .

Definition in file err_hnd.h.

Data Structures

- class err_hnd_type
  Class defining an error handler [abstract base].
- class err_hnd_gsl
  The error handler.

Defines

- #define O2SCL_ERR(d, n) o2scl::set_err_fn(d,__FILE__,__LINE__,n);
  Set an error with message `d` *and code* `n`.
- #define O2SCL_CONV(d, n, b) {if (b) o2scl::set_err_fn(d,__FILE__,__LINE__,n);}
  Set a "convergence" error.
- #define O2SCL_ERR2(d, d2, n)
  Set an error, two-string version.

- #define O2SCL_CONV2(d, d2, n, b)
  Set a "convergence" error, two-string version.
- #define O2SCL_ERR_RET(d, n) do { o2scl::set_err_fn(d,__FILE__,__LINE__,n); return n; } while (0)
  Set an error and return the error value.
- #define O2SCL_CONV_RET(d, n, b)
  Set a "convergence" error and return the error value.
- #define O2SCL_ERR2_RET(d, d2, n)
  Set an error and return the error value, two-string version.
- #define O2SCL_CONV2_RET(d, d2, n, b)
  Set an error and return the error value, two-string version.
- #define O2SCL_ASSERT(ev)
  A version of `assert`, i.e. exit if the error value is non-zero and do nothing otherwise.
- #define O2SCL_BOOL_ASSERT(ev, str)
  A version of `assert` for bool types. Exit if the argument is false.

Enumerations

- enum { gsl_success = 0, gsl_failure = -1, gsl_continue = -2, gsl_edom = 1, gsl_erange = 2, gsl_efault = 3, gsl_einval = 4, gsl_efailed = 5, gsl_efactor = 6, gsl_esanity = 7, gsl_enomem = 8, gsl_ebadfunc = 9, gsl_erunaway = 10, gsl_emaxiter = 11, gsl_ezerodiv = 12, gsl_ebadtol = 13, gsl_etol = 14, gsl_eundrflw = 15, gsl_eovrflw = 16, gsl_eloss = 17, gsl_eround = 18, gsl_ebadlen = 19, gsl_enotsqr = 20, gsl_esing = 21, gsl_ediverge = 22, gsl_eunsup = 23, gsl_eunimpl = 24, gsl_ecache = 25, gsl_etable = 26, gsl_enoprog = 27, gsl_enoprogj = 28, gsl_etolf = 29, gsl_etolx = 30, gsl_etolg = 31, gsl_eof = 32, gsl_enotfound = 33, gsl_ememtype = 34, gsl_efilenotfound = 35, gsl_eindex = 36, gsl_outsidecons = 37 }
  The integer error definitions.

Functions

- void set_err_fn (const char *desc, const char *file, int line, int errnum)
  Set an error.
- void error_update (int &ret, int err)
  Update an error value `err` with the value in `ret`.

Variables

- err_hnd_type * err_hnd
  The global error handler pointer.
- err_hnd_gsl alt_err_hnd
  An alternate GSL-like error handler.

47.5.2 Define Documentation

47.5.2.1 #define O2SCL_ERR2(d, d2, n)

Value:

```
o2scl::set_err_fn((std::string(d)+d2).c_str(), \
                                          __FILE__,__LINE__,n);
```

Definition at line 306 of file err_hnd.h.

47.5.2.2 #define O2SCL_CONV2(d, d2, n, b)

Value:

```
{if (b)                                  \
      o2scl::set_err_fn((std::string(d)+d2).c_str(),                    \
                        __FILE__,__LINE__,n);}
```

Definition at line 311 of file err_hnd.h.

47.5.2.3 #define O2SCL_CONV_RET(*d, n, b*)

Value:

```
do { if (!b) { return 0; } else {                                       \
      o2scl::set_err_fn(d,__FILE__,__LINE__,n); return n; } } while (0)
```

Definition at line 322 of file err_hnd.h.

47.5.2.4 #define O2SCL_ERR2_RET(*d, d2, n*)

Value:

```
do { o2scl::set_err_fn((std::string(d)+d2).c_str(),                     \
                       __FILE__,__LINE__,n); return n; } while (0)
```

Definition at line 328 of file err_hnd.h.

47.5.2.5 #define O2SCL_CONV2_RET(*d, d2, n, b*)

Value:

```
do { if (!b) {return 0; } else {                                        \
      o2scl::set_err_fn((std::string(d)+d2).c_str(),                    \
                        __FILE__,__LINE__,n); return n; } } while (0)
```

Definition at line 334 of file err_hnd.h.

47.5.2.6 #define O2SCL_ASSERT(*ev*)

Value:

```
do { if (ev!=0) { std::cout << "O2scl: Macro err_assert() causing exit" \
                            << " from error " << ev << " at "           \
                            << __LINE__ << " in file:\n "               \
                            << __FILE__ << std::endl;                   \
      std::cout << "Error handler string:\n " << err_hnd->get_str()     \
                << std::endl; exit(ev); } } while (0)
```

Idea for Future Make this consistent with assert() using NDEBUG?

Definition at line 359 of file err_hnd.h.

47.5.2.7 #define O2SCL_BOOL_ASSERT(*ev, str*)

Value:

```
do { if (ev==false) {                                                   \
      std::cout << "O2scl: Macro bool_assert() causing exit at line "   \
                << __LINE__ << " in file:\n "                           \
                << __FILE__ << std::endl;                               \
      std::cout << "Given string: " << str                            \
                << std::endl; exit(-1); } } while (0)
```

Definition at line 370 of file err_hnd.h.

47.5.3 Enumeration Type Documentation

47.5.3.1 anonymous enum

The errors associated with the integers between -2 and 32 are from GSL, the rest are specific to O_2scl .

Enumerator:

gsl_success
: Success (GSL)

gsl_failure
: Failure (GSL)

gsl_continue
: iteration has not converged (GSL)

gsl_edom
: input domain error, e.g sqrt(-1) (GSL)

gsl_erange
: output range error, e.g. exp(1e100) (GSL)

gsl_efault
: invalid pointer (GSL)

gsl_einval
: invalid argument supplied by user (GSL)

gsl_efailed
: generic failure (GSL)

gsl_efactor
: factorization failed (GSL)

gsl_esanity
: sanity check failed - shouldn't happen (GSL)

gsl_enomem
: malloc failed (GSL)

gsl_ebadfunc
: problem with user-supplied function (GSL)

gsl_erunaway
: iterative process is out of control (GSL)

gsl_emaxiter
: exceeded max number of iterations (GSL)

gsl_ezerodiv
: tried to divide by zero (GSL)

gsl_ebadtol
: user specified an invalid tolerance (GSL)

gsl_etol
: failed to reach the specified tolerance (GSL)

gsl_eundrflw
: underflow (GSL)

gsl_eovrflw
: overflow (GSL)

gsl_eloss
: loss of accuracy (GSL)

gsl_eround
: failed because of roundoff error (GSL)

gsl_ebadlen
: matrix, vector lengths are not conformant (GSL)

gsl_enotsqr
: matrix not square (GSL)

gsl_esing
: apparent singularity detected (GSL)

gsl_ediverge
: integral or series is divergent (GSL)

gsl_eunsup
: requested feature is not supported by the hardware (GSL)

gsl_eunimpl
requested feature not (yet) implemented (GSL)

gsl_ecache
cache limit exceeded (GSL)

gsl_etable
table limit exceeded (GSL)

gsl_enoprog
iteration is not making progress toward solution (GSL)

gsl_enoprogj
jacobian evaluations are not improving the solution (GSL)

gsl_etolf
cannot reach the specified tolerance in f (GSL)

gsl_etolx
cannot reach the specified tolerance in x (GSL)

gsl_etolg
cannot reach the specified tolerance in gradient (GSL)

gsl_eof
end of file (GSL)

gsl_enotfound
Generic "not found" result.

gsl_ememtype
Incorrect type for memory object.

gsl_efilenotfound
File not found.

gsl_eindex
Invalid index for array or matrix.

gsl_outsidecons
Outside constraint region.

Definition at line 45 of file err_hnd.h.

47.5.4 Function Documentation

47.5.4.1 void error_update (int & *ret*, int *err*) [inline]

If `ret` is zero, this sets `ret` to the value `err`, and if `ret` is nonzero this function does nothing.

Definition at line 352 of file err_hnd.h.

47.6 exception.h File Reference

File for definitions for err_hnd_cpp and the exceptions.

```
#include <stdexcept> #include <iostream> #include <o2scl/err_hnd.h>
```

47.6.1 Detailed Description

See also err_hnd.h .

Definition in file exception.h.

Data Structures

- class exc_exception
  Generic exception.
- class exc_logic_error
  Logic error exception.
- class exc_invalid_argument
  Invalid argument exception.
- class exc_runtime_error
  Generic runtime error exception.
- class exc_range_error
  Range error runtime exception.
- class exc_overflow_error
  Overflow error runtime exception.
- class exc_ios_failure
  I/O failure error exception.
- class err_hnd_cpp
  Error handler to throw C++ exceptions.

Variables

- err_hnd_cpp def_err_hnd
  The default error handler.

47.7 givens.h File Reference

File for Givens rotations.

`#include <gsl/gsl_math.h>` `#include <o2scl/err_hnd.h>` `#include <o2scl/permutation.h>` `#include <o2scl/cblas.h>` `#include <o2scl/vec_arith.h>` `#include <o2scl/givens_base.h>`

47.7.1 Detailed Description

Definition in file givens.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Defines

- #define **O2SCL_IX**(V, i) V[i]
- #define **O2SCL_IX2**(M, i, j) M[i][j]

Functions

- void o2scl_linalg::create_givens (const double a, const double b, double &c, double &s)
  Desc.

47.8 givens_base.h File Reference

File for Givens rotations.

47.8.1 Detailed Description

Todo Make sure create_givens() in givens.h is documented.

Definition in file givens_base.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Functions

- template<class mat1_t , class mat2_t >
  void o2scl_linalg::apply_givens_qr (size_t M, size_t N, mat1_t &Q, mat2_t &R, size_t i, size_t j, double c, double s)
  Apply a rotation to matrices from the QR decomposition.
- template<class mat1_t , class mat2_t >
  void o2scl_linalg::apply_givens_lq (size_t M, size_t N, mat1_t &Q, mat2_t &L, size_t i, size_t j, double c, double s)
  Apply a rotation to matrices from the LQ decomposition.
- template<class vec_t >
  void o2scl_linalg::apply_givens_vec (vec_t &v, size_t i, size_t j, double c, double s)
  Apply a rotation to a vector, $v \to G^T v$.

47.9 graph.h File Reference

Experimental functions for use with Root.

`#include <string> #include <Gtypes.h> #include <TApplication.h> #include <TArrow.h> #include <TAxis.h> #include <TBox.h> #include <TCanvas.h> #include <TColor.h> #include <TF1.h> × #include <TGraph.h> #include <TGraphErrors.h> #include <TGaxis.h> #include <TH1.h> #include <TH2.h> #include <TLatex.h> #include <TLegend.h> #include <TLine.h> #include <TList.h> × #include <TMarker.h> #include <TObjArray.h> #include <TROOT.h> #include <TStyle.h> #include <TPolyLine.h> #include <o2scl/constants.h> #include <o2scl/table_units.h>`

47.9.1 Detailed Description

All of the functions in graph.h are documented in the o2scl_graph namespace.

Definition in file graph.h.

Namespaces

- namespace o2scl_graph
  Namespace for experimental functions for use with Root.

Functions

- void o2scl_graph::arrow (double x1, double y1, double x2, double y2, TLine ∗&line, TPolyLine ∗&poly, double size=0.1, double size2=0.8, double alpha1=0.35)
  Draw a pretty arrow from (x1,y1) to (x2,y2)
- int o2scl_graph::new_graph (TCanvas ∗&c1, TPad ∗&p1, TH1 ∗&th1, std::string CanvasName, std::string WindowName, std::string PadName, double lleft, double lbottom, double lright, double ltop, int left=0, int top=0, int right=700, int bottom=700, bool logx=false, bool logy=false)

Make a canvas and pad.

- int o2scl_graph::new_graph_ticks (TCanvas ∗&c1, TPad ∗&p1, TH1 ∗&th1, std::string CanvasName, std::string WindowName, std::string PadName, double lleft, double lbottom, double lright, double ltop, TGaxis ∗&a1, TGaxis ∗&a2, int left=0, int top=0, int right=700, int bottom=700, bool logx=false, bool logy=false)
  Make a canvas and pad with more tick marks.
- TGraph ∗ o2scl_graph::table_graph (o2scl::table_units &at, std::string scolx, std::string scoly, int style=1, int color=1)
  Graph two columns from a data table.
- TGraphErrors ∗ o2scl_graph::table_graph_errors (o2scl::table_units &at, std::string scolx, std::string scoly, std::string xerr, std::string yerr, int style=1, int color=1)
  Plot colums from a data table with error bars.
- int o2scl_graph::two_up_graph (TCanvas ∗&c1, TPad ∗&p1, TPad ∗&p2, TH1 ∗&th1, TH1 ∗&th2, std::string CanvasName, std::string WindowName, std::string Pad1Name, std::string Pad2Name, double lowx1, double highx1, double lowx2, double highx2, double lowy, double highy, int left=0, int top=0, int right=1000, int bottom=700, bool logx1=false, bool logx2=false, bool logy=false, double alpha=1.3, double margin=0.1)
  Make a canvas with a two-up graph, side-by-side.
- int o2scl_graph::two_up_graphy (TCanvas ∗&c1, TPad ∗&p1, TPad ∗&p2, TH1 ∗&th1, TH1 ∗&th2, std::string CanvasName, std::string WindowName, std::string Pad1Name, std::string Pad2Name, double lowx, double highx, double lowy1, double highy1, double lowy2, double highy2, int left=0, int top=0, int right=1000, int bottom=700, bool logx=false, bool logy1=false, bool logy2=false, double alpha=1.3, double margin=0.1)
  Make a canvas with two plots, one on top of the other.

Variables

Some useful colors

- const int **o2scl_graph::kGray20** = kGray
- const int **o2scl_graph::kGray40** = kGray+1
- const int **o2scl_graph::kGray60** = kGray+2
- const int **o2scl_graph::kGray80** = kGray+3

Markers

- const int **o2scl_graph::m_small_dot** = 1
- const int **o2scl_graph::m_plus** = 2
- const int **o2scl_graph::m_asterisk** = 3
- const int **o2scl_graph::m_circle** = 4
- const int **o2scl_graph::m_times** = 5
- const int **o2scl_graph::m_med_dot** = 6
- const int **o2scl_graph::m_large_dot** = 7
- const int **o2scl_graph::m_fill_circle** = 8
- const int **o2scl_graph::m_fill_square** = 21
- const int **o2scl_graph::m_fill_up_triangle** = 22
- const int **o2scl_graph::m_fill_dn_triangle** = 23
- const int **o2scl_graph::m_open_circle** = 24
- const int **o2scl_graph::m_open_square** = 25
- const int **o2scl_graph::m_open_up_triangle** = 26
- const int **o2scl_graph::m_open_diamond** = 27
- const int **o2scl_graph::m_open_plus** = 28
- const int **o2scl_graph::m_fill_star** = 29
- const int **o2scl_graph::m_open_star** = 30
- const int **o2scl_graph::m_asterisk2** = 31
- const int **o2scl_graph::m_open_dn_triangle** = 32
- const int **o2scl_graph::m_fill_diamond** = 33
- const int **o2scl_graph::m_fill_plus** = 34

47.10 hdf_io.h File Reference

File for HDF I/O for table and table3d.

```
#include <hdf5.h> #include <o2scl/table.h> #include <o2scl/table_units.h> #include <o2scl/table3d.
h> #include <o2scl/constants.h> #include <o2scl/hist.h> #include <o2scl/hist_2d.h> #include
<o2scl/tensor.h> #include <o2scl/hdf_file.h>
```

47.10.1 Detailed Description

Definition in file hdf_io.h.

Data Structures

- struct iterate_parms
  A structure to pass information to and from iterate_func()

Functions

- int hdf_output (hdf_file &hf, const o2scl::table &t, std::string name)
  Output a table to a HDF file.
- int hdf_input (hdf_file &hf, o2scl::table &t, std::string name="")
  Input a table from a HDF file.
- int hdf_output (hdf_file &hf, const o2scl::table_units &t, std::string name)
  Output a table_units to a HDF file.
- int hdf_input (hdf_file &hf, o2scl::table_units &t, std::string name="")
  Input a table_units from a HDF file.
- int hdf_output (hdf_file &hf, const o2scl::table3d &t, std::string name)
  Output a table3d to a HDF file.
- int hdf_input (hdf_file &hf, o2scl::table3d &t, std::string name="")
  Input a table3d from a HDF file.
- int hdf_output (hdf_file &hf, const o2scl::hist &t, std::string name)
  Output a hist to a HDF file.
- int hdf_input (hdf_file &hf, o2scl::hist &t, std::string name="")
  Input a hist from a HDF file.
- int hdf_output (hdf_file &hf, const o2scl::hist_2d &t, std::string name)
  Output a hist_2d to a HDF file.
- int hdf_input (hdf_file &hf, o2scl::hist_2d &t, std::string name="")
  Input a hist_2d from a HDF file.
- int hdf_output (hdf_file &hf, const o2scl::tensor &t, std::string name)
  Output a tensor to a HDF file.
- int hdf_input (hdf_file &hf, o2scl::tensor &t, std::string name="")
  Input a tensor from a HDF file.
- int hdf_output (hdf_file &hf, const o2scl::tensor_grid &t, std::string name)
  Output a tensor to a HDF file.
- int hdf_input (hdf_file &hf, o2scl::tensor_grid &t, std::string name="")
  Input a tensor from a HDF file.

Helper struct and functions

- int iterate_func (hid_t loc, const char *name, const H5L_info_t *inf, void *op_data)
  Look at location `loc` in an HDF file for an O2scl object.
- int find_group_by_type (hdf_file &hf, std::string type, std::string &group_name, int verbose=0)
  Look in hdf_file `hf` for an O2scl object of type `type` and if found, set `group_name` to the associated object name.

47.10.2 Function Documentation

47.10.2.1 int hdf_output (hdf_file & *hf*, const o2scl::table_units & *t*, std::string *name*)

The unit conversion object, as specified in table_units::set_convert() is not written to the HDF file. Thus when a table_units object is read from a file, it always uses the default unit conversion object.

47.10.2.2 int hdf_input (hdf_file & *hf*, o2scl::table_units & *t*, std::string *name* = " ")

The unit conversion object, as specified in table_units::set_convert() is not written to the HDF file. Thus when a table_units object is read from a file, it always uses the default unit conversion object.

47.10.2.3 int iterate_func (hid_t *loc*, const char * *name*, const H5L_info_t * *inf*, void * *op_data*)

This is used by find_group_by_type() where `op_data` is a pointer to an object of type iterate_parms to look for O_2scl objects of a specified type without knowing the group name.

47.10.2.4 int find_group_by_type (hdf_file & *hf*, std::string *type*, std::string & *group_name*, int *verbose* = `0`)

This function returns 1 if an object of type `type` is found and 0 if it fails.

47.11 hh_base.h File Reference

File for householder solver.

`#include <o2scl/err_hnd.h> #include <o2scl/cblas.h> #include <o2scl/permutation.h> #include <o2scl/vec_arith.h>`

47.11.1 Detailed Description

Definition in file hh_base.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Functions

- template<class mat_t , class vec_t , class vec2_t >
  int o2scl_linalg::HH_solve (size_t n, mat_t &A, const vec_t &b, vec2_t &x)
  Solve linear system after Householder decomposition.
- template<class mat_t , class vec_t >
  int o2scl_linalg::HH_svx (size_t N, size_t M, mat_t &A, vec_t &x)
  Solve a linear system after Householder decomposition in place.

47.12 householder_base.h File Reference

File for Householder transformations.

`#include <gsl/gsl_machine.h> #include <o2scl/err_hnd.h> #include <o2scl/cblas.h> #include <o2scl/permutation.h> #include <o2scl/vec_arith.h>`

47.12.1 Detailed Description

Todo Better documentation for the Householder functions.

Definition in file householder_base.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Functions

- template<class vec_t >
  double o2scl_linalg::householder_transform (const size_t n, vec_t &v)
  Replace the vector `v` with a Householder vector and a coefficient tau that annihilates `v[1]` through `v[n-1]` (inclusive)
- template<class mat_t >
  double o2scl_linalg::householder_transform_subcol (mat_t &A, const size_t ir, const size_t ic, const size_t M)
  Compute the Householder transform of a vector formed with `n` rows of a column of a matrix.
- template<class mat_t >
  double o2scl_linalg::householder_transform_subrow (mat_t &A, const size_t ir, const size_t ic, const size_t N)
  Compute the Householder transform of a vector formed with the last `n` columns of a row of a matrix.
- template<class vec_t , class mat_t >
  void o2scl_linalg::householder_hm (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
  Apply a Householder transformation (v,τ) to matrix A of size M by N.
- template<class mat_t >
  void o2scl_linalg::householder_hm_sub (mat_t &M, const size_t ir, const size_t ic, const size_t nr, const size_t nc, const mat_t &M2, const size_t ir2, const size_t ic2, double tau)
  Apply a Householder transformation to submatrix of a larger matrix.
- template<class mat1_t , class mat2_t >
  void o2scl_linalg::householder_hm_sub2 (const size_t M, const size_t ic, double tau, const mat1_t &mv, mat2_t &A)
  Special version of Householder transformation for QR_unpack()
- template<class vec_t , class vec2_t >
  void o2scl_linalg::householder_hv (const size_t N, double tau, const vec_t &v, vec2_t &w)
  Apply a Householder transformation `v` to vector `w`.
- template<class mat_t , class vec_t >
  void o2scl_linalg::householder_hv_subcol (const mat_t &A, vec_t &w, double tau, const size_t ie, const size_t N)
  Apply a Householder transformation `v` to vector `w` where `v` is stored as a column in a matrix `A`.
- template<class mat_t >
  void o2scl_linalg::householder_hm1 (const size_t M, const size_t N, double tau, mat_t &A)
  Apply a Householder transformation (v,τ) to a matrix being build up from the identity matrix, using the first column of A as a - Householder vector.
- template<class vec_t , class mat_t >
  void o2scl_linalg::householder_mh (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
  Apply the Householder transformation `(v,tau)` to the right-hand side of the matrix `A`.
- template<class mat_t , class mat2_t >
  void o2scl_linalg::householder_mh_sub (mat_t &M, const size_t ir, const size_t ic, const size_t nr, const size_t nc, const mat2_t &M2, const size_t ir2, const size_t ic2, double tau)
  Apply the Householder transformation `(v,tau)` to the right-hand side of the matrix `A`.

47.13 lib_settings.h File Reference

File for definitions for lib_settings_class.

```
#include <iostream> #include <string>
```

47.13.1 Detailed Description

Definition in file lib_settings.h.

Data Structures

- class lib_settings_class
  A class to manage global library settings.

Namespaces

- namespace o2scl
  The main O_2scl namespace.
- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.
- namespace o2scl_linalg_paren
  The namespace for linear algebra classes and functions with operator()

Variables

- lib_settings_class lib_settings
  The global library settings object.

47.13.2 Variable Documentation

47.13.2.1 lib_settings_class lib_settings

This global object is used by polylog and some of the O_2scl_eos classes to find data files. It may also be used by the end-user to probe details of the O_2scl installation.

47.14 lu.h File Reference

File for LU decomposition and associated solver.

`#include <o2scl/err_hnd.h> #include <o2scl/permutation.h> #include <o2scl/cblas.h> #include <o2scl/vec_arith.h> #include <o2scl/lu_base.h>`

47.14.1 Detailed Description

Definition in file lu.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.
- namespace o2scl_linalg_paren
  The namespace for linear algebra classes and functions with operator()

Defines

- #define **O2SCL_IX**(V, i) V[i]
- #define **O2SCL_IX2**(M, i, j) M[i][j]
- #define **O2SCL_IX**(V, i) V(i)
- #define **O2SCL_IX2**(M, i, j) M(i,j)

Functions

- template<size_t N>
  int o2scl_linalg::LU_decomp_array_2d (const size_t n, double A[ ][N], o2scl::permutation &p, int &signum)
  Specialized version of LU_decomp for C-style 2D arrays.

47.15 lu_base.h File Reference

File for LU decomposition and associated solver.

47.15.1 Detailed Description

Definition in file lu_base.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Functions

- template<class mat_t >
  int o2scl_linalg::diagonal_has_zero (const size_t N, mat_t &A)
  Return 1 if at least one of the elements in the diagonal is zero.
- template<class mat_t >
  int o2scl_linalg::LU_decomp (const size_t N, mat_t &A, o2scl::permutation &p, int &signum)
  Compute the LU decomposition of the matrix `A`.
- template<class mat_t , class mat_row_t >
  int **o2scl_linalg::LU_decomp_alt** (const size_t N, mat_t &A, o2scl::permutation &p, int &signum)
- template<class mat_t , class vec_t , class vec2_t >
  int o2scl_linalg::LU_solve (const size_t N, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec2_t &x)
  Solve a linear system after LU decomposition.
- template<class mat_t , class vec_t >
  int o2scl_linalg::LU_svx (const size_t N, const mat_t &LU, const o2scl::permutation &p, vec_t &x)
  Solve a linear system after LU decomposition in place.
- template<class mat_t , class mat2_t , class vec_t , class vec2_t , class vec3_t >
  int o2scl_linalg::LU_refine (const size_t N, const mat_t &A, const mat2_t &LU, const o2scl::permutation &p, const vec_t &b, vec2_t &x, vec3_t &residual)
  Refine the solution of a linear system.
- template<class mat_t , class mat2_t , class mat_col_t >
  int o2scl_linalg::LU_invert (const size_t N, const mat_t &LU, const o2scl::permutation &p, mat2_t &inverse)
  Compute the inverse of a matrix from its LU decomposition.
- template<class mat_t >
  double o2scl_linalg::LU_det (const size_t N, const mat_t &LU, int signum)
  Compute the determinant of a matrix from its LU decomposition.
- template<class mat_t >
  double o2scl_linalg::LU_lndet (const size_t N, const mat_t &LU)
  Compute the logarithm of the absolute value of the determinant of a matrix from its LU decomposition.
- template<class mat_t >
  int o2scl_linalg::LU_sgndet (const size_t N, const mat_t &LU, int signum)
  Compute the sign of the determinant of a matrix from its LU decomposition.

47.16 minimize.h File Reference

One-dimensional minimization base class and associated functions.

```
#include <cmath> #include <o2scl/err_hnd.h> #include <o2scl/funct.h>
```

47.16.1 Detailed Description

Definition in file minimize.h.

Data Structures

- class minimize< func_t, dfunc_t >
  One-dimensional minimization [abstract base].
- class minimize_bkt< func_t, dfunc_t >
  One-dimensional bracketing minimization [abstract base].
- class minimize_de< func_t, dfunc_t >
  One-dimensional minimization using derivatives [abstract base].

Functions

- double constraint (double x, double center, double width, double height)
  Constrain x to be within `width` *of the value given by* `center`.
- double cont_constraint (double x, double center, double width, double height, double tightness=40.0, double exp_arg_limit=50.0)
  Constrain x to be within `width` *of the value given by* `center`.
- double lower_bound (double x, double center, double width, double height)
  Constrain x to be greater than the value given by `center`.
- double cont_lower_bound (double x, double center, double width, double height, double tightness=40.0, double exp_arg_limit=50.0)
  Constrain x to be greater than the value given by `center`.

47.16.2 Function Documentation

47.16.2.1 double constraint (double *x*, double *center*, double *width*, double *height*) [inline]

Defining $c =$ `center`, $w =$ `width`, $h =$ `height`, this returns the value $h(1+|x-c-w|/w)$ if $x > c+w$ and $h(1+|x-c+w|/w)$ if $x < c-w$. The value near $x = c-w$ or $x = c+w$ is h (the value of the function exactly at these points is zero) and the value at $x = c-2w$ or $x = c+2w$ is $2h$.

It is important to note that, for large distances of `x` from `center`, this only scales linearly. If you are trying to constrain a function which decreases more than linearly by making `x` far from `center`, then a minimizer may ignore this constraint.

Definition at line 370 of file minimize.h.

47.16.2.2 double cont_constraint (double *x*, double *center*, double *width*, double *height*, double *tightness* = `40.0`, double *exp_arg_limit* = `50.0`) [inline]

Defining $c =$ `center`, $w =$ `width`, $h =$ `height`, $t =$ `tightness`, and $\ell =$ `exp_arg_limit`, this returns the value

$$h\left(\frac{x-c}{w}\right)^2\left[1+e^{t(x-c+w)(c+w-x)/w^2}\right]^{-1}$$

This function is continuous and differentiable. Note that if $x = c$, then the function returns zero.

The exponential is handled gracefully by assuming that anything smaller than $\exp(-\ell)$ is zero. This creates a small discontinuity which can be removed with the sufficiently large value of ℓ.

It is important to note that, for large distances of `x` from `center`, this scales quadratically. If you are trying to constrain a function which decreases faster than quadratically by making `x` far from `center`, then a minimizer may ignore this constraint.

In the limit $t \to \infty$, this function converges towards the squared value of constraint(), except exactly at the points $x = c-w$ and $x = c+w$.

Definition at line 411 of file minimize.h.

47.16.2.3 double lower_bound (double *x*, double *center*, double *width*, double *height*) [inline]

Defining $c =$ `center`, $w =$ `width`, $h =$ `height`, this returns $h(1+|x-c|/w)$ if $x < c$ and zero otherwise. The value at $x = c$ is h , while the value at $x = c-w$ is $2h$.

It is important to note that, for large distances of x from center, this only scales linearly. If you are trying to constrain a function which decreases more than linearly by making x far from center, then a minimizer may ignore this constraint.

Definition at line 438 of file minimize.h.

47.16.2.4 double cont_lower_bound (double *x*, double *center*, double *width*, double *height*, double *tightness* = 40.0, double *exp_arg_limit* = 50.0) [inline]

Defining $c=$ center, $w=$ width, $h=$ height, $t=$ tightness, and $\ell=$ exp_arg_limit, this returns $h(c-x+w)/(w+w\exp(t(x-c)/w))$ and has the advantage of being a continuous and differentiable function. The value of the function exactly at $x=c$ is $h/2$, but for x just below c the function is h and just above c the function is quite small.

The exponential is handled gracefully by assuming that anything smaller than $\exp(-\ell)$ is zero. This creates a small discontinuity which can be removed with the sufficiently large value of ℓ.

It is important to note that, for large distances of x from center, this only scales linearly. If you are trying to constrain a function which decreases more than linearly by making x far from center, then a minimizer may ingore this constraint.

In the limit $t\to\infty$, this function converges towards lower_bound(), except exactly at the point $x=c$.

Definition at line 472 of file minimize.h.

47.17 misc.h File Reference

Miscellaneous functions.

```
#include <cstdlib> #include <iostream> #include <cmath> #include <string> #include <fstream> ×
#include <sstream> #include <vector> #include <gsl/gsl_vector.h> #include <gsl/gsl_ieee-
_utils.h> #include <o2scl/err_hnd.h> #include <o2scl/lib_settings.h>
```

47.17.1 Detailed Description

Definition in file misc.h.

Data Structures

- struct string_comp
  Simple string comparison.
- class gen_test_number< tot >
  Generate number sequence for testing.

Functions

- double fermi_function (double E, double mu, double T, double limit=40.0)
  Calculate a Fermi-Dirac distribution function safely.
- template<class string_arr_t >
  int screenify (size_t nin, const string_arr_t &in_cols, std::vector< std::string > &out_cols, size_t max_size=80)
  Reformat the columns for output of width size.
- int count_words (std::string str)
  Count the number of words in the string str.
- int remove_whitespace (std::string &s)
  Remove all whitespace from the string s.
- std::string binary_to_hex (std::string s)
  Take a string of binary quads and compress them to hexadecimal digits.
- template<class type_t >
  int gsl_alloc_arrays (size_t nv, size_t msize, const char *names[], type_t *ptrs[], std::string func_name)
  A convenient function to allocate several arrays of the same size.

- template<class type_t >
  int gsl_calloc_arrays (size_t nv, size_t msize, const char *names[ ], type_t *ptrs[ ], std::string func_name, type_t val)
  A convenient function to allocate and initialize several arrays of the same size.
- int gsl_calloc_gvecs (size_t nv, size_t msize, const char *names[ ], gsl_vector ***ptrs, std::string func_name)
  A convenient function to allocate and initialize several arrays of the same size.
- double quadratic_extremum_x (double x1, double x2, double x3, double y1, double y2, double y3)
  Return the x value of the extremum of a quadratic defined by three (x,y) pairs.
- double quadratic_extremum_y (double x1, double x2, double x3, double y1, double y2, double y3)
  Return the y value of the extremum of a quadratic defined by three (x,y) pairs.

47.17.2 Function Documentation

47.17.2.1 double fermi_function (double *E*, double *mu*, double *T*, double *limit* = 40.0)

$$[1+\exp(E/T-\mu/T)]^{-1}$$

This calculates a Fermi-Dirac distribution function guaranteeing that numbers larger than exp(limit) and smaller than exp(−limit) will be avoided. The default value of limit=40 ensures accuracy to within 1 part in 10^{17} compared to the maximum of the distribution (which is unity).

Note that this function may return Inf or NAN if limit is too large, depending on the machine precision.

47.17.2.2 template<class string_arr_t > int screenify (size_t *nin*, const string_arr_t & *in_cols*, std::vector< std::string > & *out_cols*, size_t *max_size* = 80)

Given a string array in_cols of size nin, screenify() reformats the array into columns creating a new string array out_cols.

For example, for an array of 10 strings

```
test1
test_of_string2
test_of_string3
test_of_string4
test5
test_of_string6
test_of_string7
test_of_string8
test_of_string9
test_of_string10
```

screenify() will create an array of 3 new strings:

```
test1           test_of_string4  test_of_string7  test_of_string10
test_of_string2 test5            test_of_string8
test_of_string3 test_of_string6  test_of_string9
```

If the value of max_size is less than the length of the longest input string (plus one for a space character), then the output strings may have a larger length than max_size.

Definition at line 94 of file misc.h.

47.17.2.3 int count_words (std::string *str*)

Words are defined as groups of characters separated by whitespace, where whitespace is any combination of adjacent spaces, tabs, carriage returns, etc. On most systems, whitespace is usually defined as any character corresponding to the integers 9 (horizontal tab), 10 (line feed), 11 (vertical tab), 12 (form feed), 13 (carriage return), and 32 (space bar). The test program misc_ts enumerates the characters between 0 and 255 (inclusive) that count as whitespace for this purpose.

Note that this function is used in text_in_file::string_in to perform string input.

47.17.2.4 int remove_whitespace (std::string & *s*)

This function removes all characters in `s` which correspond to the integer values 9, 10, 11, 12, 13, or 32.

47.17.2.5 std::string binary_to_hex (std::string *s*)

This function proceeds from left to right, ignoring parts of the string that do not consist of squences of four '1's or '0's.

47.17.2.6 template<class type_t > int gsl_alloc_arrays (size_t *nv*, size_t *msize*, const char * *names[]*, type_t * *ptrs[]*, std::string *func_name*)

Allocate memory for `nv` vectors of size `msize` with names specified in `names` and function name `func_name` to produce pointers in `ptrs`. This function is unsed internally to allocate several vectors in succession, taking care to free memory and call the error handler when an allocation fails.

The caller must make sure that previously allocated memory will be freed and an error is thrown if this function returns a non-zero value.

This function is used, for example, in gsl_miser.

Definition at line 290 of file misc.h.

47.17.2.7 template<class type_t > int gsl_calloc_arrays (size_t *nv*, size_t *msize*, const char * *names[]*, type_t * *ptrs[]*, std::string *func_name*, type_t *val*)

The caller must make sure that previously allocated memory will be freed and an error is thrown if this function returns a non-zero value.

Definition at line 320 of file misc.h.

47.17.2.8 int gsl_calloc_gvecs (size_t *nv*, size_t *msize*, const char * *names[]*, gsl_vector *** *ptrs*, std::string *func_name*)

The caller must make sure that previously allocated memory will be freed and an error is thrown if this function returns a non-zero value.

47.17.2.9 double quadratic_extremum_x (double *x1*, double *x2*, double *x3*, double *y1*, double *y2*, double *y3*)

Idea for Future Make a function quadratic_points(double x1, double x2, double x3, double y1, double y2, double y3, double &a, double &b, double &c)?

Idea for Future Make a quadratic_extremum_xy()?

47.18 omatrix_cx_tlate.h File Reference

File for definitions of complex matrices.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> x
#include <o2scl/err_hnd.h> #include <gsl/gsl_matrix.h> #include <gsl/gsl_complex.h> #include
<o2scl/ovector_tlate.h> #include <o2scl/ovector_cx_tlate.h>
```

47.18.1 Detailed Description

Definition in file omatrix_cx_tlate.h.

Data Structures

- class omatrix_cx_view_tlate< data_t, parent_t, block_t, complex_t >
  A matrix view of double-precision numbers.
- class omatrix_cx_tlate< data_t, parent_t, block_t, complex_t >

A matrix of double-precision numbers.
- class omatrix_cx_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >
  Create a vector from a row of a matrix.
- class omatrix_cx_const_row_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >
  Create a vector from a row of a matrix.
- class omatrix_cx_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >
  Create a vector from a column of a matrix.
- class omatrix_cx_const_col_tlate< data_t, mparent_t, vparent_t, block_t, complex_t >
  Create a vector from a column of a matrix.

Typedefs

- typedef omatrix_cx_tlate < double, gsl_matrix_complex, gsl_block_complex, gsl_complex > omatrix_cx
  omatrix_cx typedef
- typedef omatrix_cx_view_tlate < double, gsl_matrix_complex, gsl_block_complex, gsl_complex > omatrix_cx_view
  omatrix_cx_view typedef
- typedef omatrix_cx_row_tlate < double, gsl_matrix_complex, gsl_vector_complex, gsl_block_complex, gsl_complex > omatrix_cx_row
  omatrix_cx_row typedef
- typedef omatrix_cx_col_tlate < double, gsl_matrix_complex, gsl_vector_complex, gsl_block_complex, gsl_complex > omatrix_cx_col
  omatrix_cx_col typedef
- typedef omatrix_cx_const_row_tlate < double, gsl_matrix_complex, gsl_vector_complex, gsl_block_complex, gsl_complex > omatrix_cx_const_row
  omatrix_cx_const_row typedef
- typedef omatrix_cx_const_col_tlate < double, gsl_matrix_complex, gsl_vector_complex, gsl_block_complex, gsl_complex > omatrix_cx_const_col
  omatrix_cx_const_col typedef

47.19 omatrix_tlate.h File Reference

File for definitions of matrices.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <gsl/gsl_matrix.h> #include <gsl/gsl_ieee_utils.h> #include <o2scl/err_hnd.h> ×
#include <o2scl/ovector_tlate.h> #include <o2scl/array.h>
```

47.19.1 Detailed Description

Idea for Future The `xmatrix` class demonstrates how operator[] could return an ovector_array object and thus provide more bounds-checking. This would demand including a new parameter in omatrix_view_tlate which contains the vector type.

Definition in file omatrix_tlate.h.

Data Structures

- class omatrix_const_view_tlate< data_t, mparent_t, block_t >
  A const matrix view of omatrix objects.
- class omatrix_base_tlate< data_t, mparent_t, block_t >
  A base class for omatrix and omatrix_view.
- class omatrix_view_tlate< data_t, mparent_t, block_t >
  A matrix view of double-precision numbers.
- class omatrix_tlate< data_t, mparent_t, vparent_t, block_t >
  A matrix of double-precision numbers.
- class omatrix_array_tlate< data_t, mparent_t, block_t >

Create a matrix from an array.
- class omatrix_row_tlate< data_t, mparent_t, vparent_t, block_t >
  Create a vector from a row of a matrix.
- class omatrix_const_row_tlate< data_t, mparent_t, vparent_t, block_t >
  Create a const vector from a row of a matrix.
- class omatrix_col_tlate< data_t, mparent_t, vparent_t, block_t >
  Create a vector from a column of a matrix.
- class omatrix_const_col_tlate< data_t, mparent_t, vparent_t, block_t >
  Create a const vector from a column of a matrix.
- class omatrix_diag_tlate< data_t, mparent_t, vparent_t, block_t >
  Create a vector from the main diagonal.
- class omatrix_const_diag_tlate< data_t, mparent_t, vparent_t, block_t >
  Create a vector from the main diagonal.
- class omatrix_alloc
  A simple class to provide an `allocate()` function for omatrix.
- class ofmatrix< N, M >
- class xmatrix
  A version of omatrix with better error checking.

Typedefs

- typedef omatrix_tlate< double, gsl_matrix, gsl_vector_norm, gsl_block > omatrix
  omatrix typedef
- typedef omatrix_view_tlate < double, gsl_matrix, gsl_block > omatrix_view
  omatrix_view typedef
- typedef omatrix_const_view_tlate < double, gsl_matrix, gsl_block > omatrix_const_view
  omatrix_const_view typedef
- typedef omatrix_base_tlate < double, gsl_matrix, gsl_block > omatrix_base
  omatrix_base typedef
- typedef omatrix_row_tlate < double, gsl_matrix, gsl_vector_norm, gsl_block > omatrix_row
  omatrix_row typedef
- typedef omatrix_col_tlate < double, gsl_matrix, gsl_vector_norm, gsl_block > omatrix_col
  omatrix_col typedef
- typedef omatrix_const_row_tlate < double, gsl_matrix, gsl_vector_norm, gsl_block > omatrix_const_row
  omatrix_const_row typedef
- typedef omatrix_const_col_tlate < double, gsl_matrix, gsl_vector_norm, gsl_block > omatrix_const_col
  omatrix_const_col typedef
- typedef omatrix_diag_tlate < double, gsl_matrix, gsl_vector_norm, gsl_block > omatrix_diag
  omatrix_diag typedef
- typedef omatrix_const_diag_tlate < double, gsl_matrix, gsl_vector_norm, gsl_block > omatrix_const_diag
  omatrix_const_diag typedef
- typedef omatrix_array_tlate < double, gsl_matrix, gsl_block > omatrix_array
  omatrix_array typedef
- typedef omatrix_tlate< int, gsl_matrix_int, gsl_vector_int, gsl_block_int > omatrix_int
  omatrix_int typedef
- typedef omatrix_view_tlate < int, gsl_matrix_int, gsl_block_int > omatrix_int_view
  omatrix_int_view typedef
- typedef omatrix_base_tlate < int, gsl_matrix_int, gsl_block_int > omatrix_int_base
  omatrix_int_base typedef
- typedef omatrix_const_view_tlate< int, gsl_matrix_int, gsl_block_int > omatrix_int_const_view
  omatrix_int_const_view typedef
- typedef omatrix_row_tlate< int, gsl_matrix_int, gsl_vector_int, gsl_block_int > omatrix_int_row
  omatrix_int_row typedef
- typedef omatrix_col_tlate< int, gsl_matrix_int, gsl_vector_int, gsl_block_int > omatrix_int_col
  omatrix_int_col typedef
- typedef omatrix_const_row_tlate< int, gsl_matrix_int, gsl_vector_int, gsl_block_int > omatrix_int_const_row
  omatrix_int_const_row typedef
- typedef omatrix_const_col_tlate< int, gsl_matrix_int, gsl_vector_int, gsl_block_int > omatrix_int_const_col
  omatrix_int_const_col typedef
- typedef omatrix_diag_tlate < int, gsl_matrix_int, gsl_vector_int, gsl_block_int > omatrix_int_diag

omatrix_int_diag typedef
- typedef omatrix_const_diag_tlate< int, gsl_matrix_int, gsl_vector_int, gsl_block_int > omatrix_int_const_diag
  omatrix_int_const_diag typedef
- typedef omatrix_array_tlate < int, gsl_matrix_int, gsl_block_int > omatrix_int_array
  omatrix_int_array typedef

Functions

- template<class data_t , class parent_t , class block_t >
  std::ostream & operator<< (std::ostream &os, const omatrix_const_view_tlate< data_t, parent_t, block_t > &v)
  A operator for output of omatrix objects.

47.19.2 Function Documentation

47.19.2.1 template<class data_t , class parent_t , class block_t > std::ostream& operator<< (std::ostream & *os*, const omatrix_const_view_tlate< data_t, parent_t, block_t > & *v*)

This outputs all of the matrix elements. Each row is output with an endline character at the end of each row. Positive values are preceeded by an extra space. A 2x2 example:

```
-3.751935e-05 -6.785864e-04
-6.785864e-04  1.631984e-02
```

The function `gsl_ieee_double_to_rep()` is used to determine the sign of a number, so that "-0.0" as distinct from "+0.0" is handled correctly.

Idea for Future This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Definition at line 1284 of file omatrix_tlate.h.

47.20 ovector_cx_tlate.h File Reference

File for definitions of complex vectors.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> x
#include <vector> #include <complex> #include <o2scl/err_hnd.h> #include <o2scl/ovector-
_tlate.h> #include <gsl/gsl_vector.h> #include <gsl/gsl_complex.h>
```

47.20.1 Detailed Description

Definition in file ovector_cx_tlate.h.

Data Structures

- class ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t >
  A vector view of double-precision numbers.
- class ovector_cx_tlate< data_t, vparent_t, block_t, complex_t >
  A vector of double-precision numbers.
- class ovector_cx_array_tlate< data_t, vparent_t, block_t, complex_t >
  Create a vector from an array.
- class ovector_cx_array_stride_tlate< data_t, vparent_t, block_t, complex_t >
  Create a vector from an array with a stride.
- class ovector_cx_subvector_tlate< data_t, vparent_t, block_t, complex_t >

Create a vector from a subvector of another.

- class ovector_cx_const_array_tlate< data_t, vparent_t, block_t, complex_t >
  Create a vector from an array.
- class ovector_cx_const_array_stride_tlate< data_t, vparent_t, block_t, complex_t >
  Create a vector from an array_stride.
- class ovector_cx_const_subvector_tlate< data_t, vparent_t, block_t, complex_t >
  Create a vector from a subvector of another.
- class ovector_cx_real_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t >
  Create a real vector from the real parts of a complex vector.
- class ovector_cx_imag_tlate< data_t, vparent_t, block_t, cvparent_t, cblock_t, complex_t >
  Create a imaginary vector from the imaginary parts of a complex vector.
- class ofvector_cx< N >
  A complex vector where the memory allocation is performed in the constructor.

Typedefs

- typedef ovector_cx_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx
  ovector_cx typedef
- typedef ovector_cx_view_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_view
  ovector_cx_view typedef
- typedef ovector_cx_array_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_array
  ovector_cx_array typedef
- typedef ovector_cx_array_stride_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_array_stride
  ovector_cx_array_stride typedef
- typedef ovector_cx_subvector_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_subvector
  ovector_cx_subvector typedef
- typedef ovector_cx_const_array_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_const_array
  ovector_cx_const_array typedef
- typedef ovector_cx_const_array_stride_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_const_array_stride
  ovector_cx_const_array_stride typedef
- typedef ovector_cx_const_subvector_tlate < double, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_const_subvector
  ovector_cx_const_subvector typedef
- typedef ovector_cx_real_tlate < double, gsl_vector, gsl_block, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_real
  ovector_cx_real typedef
- typedef ovector_cx_imag_tlate < double, gsl_vector, gsl_block, gsl_vector_complex, gsl_block_complex, gsl_complex > ovector_cx_imag
  ovector_cx_imag typedef

Functions

- template<class data_t , class vparent_t , class block_t , class complex_t >
  ovector_cx_tlate< data_t, vparent_t, block_t, complex_t > conjugate (ovector_cx_tlate< data_t, vparent_t, block_t, complex_t > &v)
  Conjugate a vector.
- template<class data_t , class vparent_t , class block_t , class complex_t >
  std::ostream & operator<< (std::ostream &os, const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &v)
  A operator for naive vector output.

47.20.2 Function Documentation

47.20.2.1 template<class data_t , class vparent_t , class block_t , class complex_t > std::ostream& operator<< (std::ostream & *os*, const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & *v*)

This outputs all of the vector elements in the form (r,i). All of these are separated by one space character, though no trailing space or `endl` is sent to the output.

Definition at line 1066 of file ovector_cx_tlate.h.

47.21 ovector_rev_tlate.h File Reference

File for definitions of reversed vectors.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <vector> #include <gsl/gsl_vector.h> #include <o2scl/err_hnd.h> #include <o2scl/string-
_conv.h> #include <o2scl/ovector_tlate.h> #include <o2scl/array.h> #include <o2scl/vector.-
h>
```

47.21.1 Detailed Description

Definition in file ovector_rev_tlate.h.

Data Structures

- class ovector_reverse_tlate< data_t, vparent_t, block_t >
  Reversed view of a vector.
- class ovector_const_reverse_tlate< data_t, vparent_t, block_t >
  Reversed view of a vector.
- class ovector_subvector_reverse_tlate< data_t, vparent_t, block_t >
  Reversed view of a subvector.
- class ovector_const_subvector_reverse_tlate< data_t, vparent_t, block_t >
  Reversed view of a const subvector.

Typedefs

- typedef ovector_reverse_tlate < double, gsl_vector_norm, gsl_block > ovector_reverse
  ovector_reverse typedef
- typedef ovector_const_reverse_tlate < double, gsl_vector_norm, gsl_block > ovector_const_reverse
  ovector_const_reverse typedef
- typedef ovector_subvector_reverse_tlate < double, gsl_vector_norm, gsl_block > ovector_subvector_reverse
  ovector_subvector_reverse typedef
- typedef ovector_const_subvector_reverse_tlate < double, gsl_vector_norm, gsl_block > ovector_const_subvector_reverse
  ovector_const_subvector_reverse typedef
- typedef ovector_reverse_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_reverse
  ovector_int_reverse typedef
- typedef ovector_const_reverse_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_const_reverse
  ovector_int_const_reverse typedef
- typedef ovector_subvector_reverse_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_subvector_reverse
  ovector_int_subvector_reverse typedef
- typedef ovector_const_subvector_reverse_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_const_subvector_reverse
  ovector_int_const_subvector_reverse typedef

47.22 ovector_tlate.h File Reference

File for definitions of vectors.

```
#include <iostream> #include <cstdlib> #include <cmath> #include <string> #include <fstream> ×
#include <sstream> #include <vector> #include <gsl/gsl_vector.h> #include <gsl/gsl_sys.-
h> #include <o2scl/err_hnd.h> #include <o2scl/string_conv.h> #include <o2scl/uvector_tlate.-
h> #include <o2scl/array.h> #include <o2scl/vector.h>
```

47.22.1 Detailed Description

Idea for Future Clean up maybe by moving, for example, ovector reverse classes to a different header file

Idea for Future Define ovector_uint classes?

Definition in file ovector_tlate.h.

Data Structures

- class gsl_vector_norm
  Norm object for gsl vectors.
- class ovector_const_view_tlate< data_t, vparent_t, block_t >
  A const vector view with finite stride.
- class ovector_const_view_tlate< data_t, vparent_t, block_t >::const_iterator
  A const iterator for ovectors.
- class ovector_const_view_tlate< data_t, vparent_t, block_t >::iterator
  An iterator for ovectors.
- class ovector_base_tlate< data_t, vparent_t, block_t >
  A base class for ovector and ovector_view.
- class ovector_view_tlate< data_t, vparent_t, block_t >
  A vector view with finite stride.
- class ovector_tlate< data_t, vparent_t, block_t >
  A vector with finite stride.
- class ovector_array_tlate< data_t, vparent_t, block_t >
  Create a vector from an array.
- class ovector_array_stride_tlate< data_t, vparent_t, block_t >
  Create a vector from an array with a stride.
- class ovector_subvector_tlate< data_t, vparent_t, block_t >
  Create a vector from a subvector of another.
- class ovector_const_array_tlate< data_t, vparent_t, block_t >
  Create a const vector from an array.
- class ovector_const_array_stride_tlate< data_t, vparent_t, block_t >
  Create a const vector from an array with a stride.
- class ovector_const_subvector_tlate< data_t, vparent_t, block_t >
  Create a const vector from a subvector of another vector.
- class ovector_alloc
  A simple class to provide an `allocate()` function for ovector.
- class ovector_int_alloc
  A simple class to provide an `allocate()` function for ovector_int.
- class ofvector< N >
  A vector where the memory allocation is performed in the constructor.

Typedefs

- typedef ovector_tlate< double, gsl_vector_norm, gsl_block > ovector
  ovector typedef
- typedef ovector_base_tlate < double, gsl_vector_norm, gsl_block > ovector_base

ovector_base typedef
- typedef ovector_view_tlate < double, gsl_vector_norm, gsl_block > ovector_view
  ovector_view typedef
- typedef ovector_const_view_tlate < double, gsl_vector_norm, gsl_block > ovector_const_view
  ovector_const_view typedef
- typedef ovector_array_tlate < double, gsl_vector_norm, gsl_block > ovector_array
  ovector_array typedef
- typedef ovector_array_stride_tlate < double, gsl_vector_norm, gsl_block > ovector_array_stride
  ovector_array_stride typedef
- typedef ovector_subvector_tlate < double, gsl_vector_norm, gsl_block > ovector_subvector
  ovector_subvector typedef
- typedef ovector_const_array_tlate < double, gsl_vector_norm, gsl_block > ovector_const_array
  ovector_const_array typedef
- typedef ovector_const_array_stride_tlate < double, gsl_vector_norm, gsl_block > ovector_const_array_stride
  ovector_const_array_stride typedef
- typedef ovector_const_subvector_tlate < double, gsl_vector_norm, gsl_block > ovector_const_subvector
  ovector_const_subvector typedef
- typedef ovector_tlate< int, gsl_vector_int, gsl_block_int > ovector_int
  ovector_int typedef
- typedef ovector_base_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_base
  ovector_int_base typedef
- typedef ovector_view_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_view
  ovector_int_view typedef
- typedef ovector_const_view_tlate< int, gsl_vector_int, gsl_block_int > ovector_int_const_view
  ovector_int_const_base typedef
- typedef ovector_array_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_array
  ovector_int_array typedef
- typedef ovector_array_stride_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_array_stride
  ovector_int_array_stride typedef
- typedef ovector_subvector_tlate< int, gsl_vector_int, gsl_block_int > ovector_int_subvector
  ovector_int_subvector typedef
- typedef ovector_const_array_tlate< int, gsl_vector_int, gsl_block_int > ovector_int_const_array
  ovector_int_const_array typedef
- typedef ovector_const_array_stride_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_const_array_stride
  ovector_int_const_array_stride typedef
- typedef ovector_const_subvector_tlate < int, gsl_vector_int, gsl_block_int > ovector_int_const_subvector
  ovector_int_const_subvector typedef

Functions

- template<class data_t , class vparent_t , class block_t >
  std::ostream & operator<< (std::ostream &os, const ovector_const_view_tlate< data_t, vparent_t, block_t > &v)
  A operator for naive vector output.

47.22.2 Function Documentation

47.22.2.1 template<class data_t , class vparent_t , class block_t > std::ostream& operator<< (std::ostream & *os*, const ovector_const_view_tlate< data_t, vparent_t, block_t > & *v*)

This outputs all of the vector elements. All of these are separated by one space character, though no trailing space or `endl` is sent to the output. If the vector is empty, nothing is done.

Definition at line 1845 of file ovector_tlate.h.

47.23 permutation.h File Reference

File containing permutation class and associated functions.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <vector> #include <gsl/gsl_vector.h> #include <gsl/gsl_permutation.h> #include
<o2scl/err_hnd.h> #include <o2scl/string_conv.h> #include <o2scl/uvector_tlate.h> #include
<o2scl/array.h> #include <o2scl/vector.h>
```

47.23.1 Detailed Description

Definition in file permutation.h.

Data Structures

- class permutation
  A class for representing permutations.

Functions

- std::ostream & operator<< (std::ostream &os, const permutation &p)
  Output operator for permutations.

47.23.2 Function Documentation

47.23.2.1 std::ostream& operator<< (std::ostream & *os*, const permutation & *p*)

A space is output between the permutation elements but no space or endline character is output after the last element.

If the size is zero, this function outputs nothing and does not call the error handler.

47.24 poly.h File Reference

Classes for solving polynomials.

```
#include <iostream> #include <complex> #include <gsl/gsl_math.h> #include <gsl/gsl_complex-
_math.h> #include <gsl/gsl_complex.h> #include <gsl/gsl_poly.h> #include <o2scl/constants.-
h> #include <o2scl/err_hnd.h>
```

47.24.1 Detailed Description

Warning

One must be careful about using pow() in functions using complex<double> since pow(((complex<double>)0.0),3.0) returns (nan,nan). Instead, we should use pow(((complex<double>)0.0),3) which takes an integer for the second argument. The sqrt() function, always succeeds i.e. sqrt(((complex<double>)0.0))=0.0

Idea for Future The quartics are tested only for a4=1, which should probably be generalized.

Definition in file poly.h.

Data Structures

- class quadratic_real
  Solve a quadratic polynomial with real coefficients and real roots [abstract base].
- class quadratic_real_coeff
  Solve a quadratic polynomial with real coefficients and complex roots [abstract base].
- class quadratic_complex
  Solve a quadratic polynomial with complex coefficients and complex roots [abstract base].
- class cubic_real
  Solve a cubic polynomial with real coefficients and real roots [abstract base].
- class cubic_real_coeff
  Solve a cubic polynomial with real coefficients and complex roots [abstract base].
- class cubic_complex
  Solve a cubic polynomial with complex coefficients and complex roots [abstract base].
- class quartic_real
  Solve a quartic polynomial with real coefficients and real roots [abstract base].
- class quartic_real_coeff
  Solve a quartic polynomial with real coefficients and complex roots [abstract base].
- class quartic_complex
  Solve a quartic polynomial with complex coefficients and complex roots [abstract base].
- class poly_real_coeff
  Solve a general polynomial with real coefficients and complex roots [abstract base].
- class poly_complex
  Solve a general polynomial with complex coefficients [abstract base].
- class cern_cubic_real_coeff
  Solve a cubic with real coefficients and complex roots (CERNLIB)
- class cern_quartic_real_coeff
  Solve a quartic with real coefficients and complex roots (CERNLIB)
- class gsl_quadratic_real_coeff
  Solve a quadratic with real coefficients and complex roots (GSL)
- class gsl_cubic_real_coeff
  Solve a cubic with real coefficients and complex roots (GSL)
- class gsl_quartic_real
  Solve a quartic with real coefficients and real roots (GSL)
- class gsl_quartic_real2
  Solve a quartic with real coefficients and real roots (GSL)
- class gsl_poly_real_coeff
  Solve a general polynomial with real coefficients (GSL)
- class quadratic_std_complex
  Solve a quadratic with complex coefficients and complex roots.
- class cubic_std_complex
  Solve a cubic with complex coefficients and complex roots.
- class simple_quartic_real
  Solve a quartic with real coefficients and real roots.
- class simple_quartic_complex
  Solve a quartic with complex coefficients and complex roots.

47.25 qr_base.h File Reference

File for QR decomposition and associated solver.

```
#include <o2scl/householder.h> #include <o2scl/givens.h>
```

47.25.1 Detailed Description

Definition in file qr_base.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Functions

- template<class mat_t , class vec_t >
  void o2scl_linalg::QR_decomp (size_t M, size_t N, mat_t &A, vec_t &tau)
  Compute the QR decomposition of matrix A.
- template<class mat_t , class vec_t , class vec2_t , class vec3_t >
  void o2scl_linalg::QR_solve (size_t N, const mat_t &QR, const vec_t &tau, const vec2_t &b, vec3_t &x)
  Solve the system A x = b using the QR factorization.
- template<class mat_t , class vec_t , class vec2_t >
  void o2scl_linalg::QR_svx (size_t M, size_t N, const mat_t &QR, const vec_t &tau, vec2_t &x)
  Solve the system A x = b in place using the QR factorization.
- template<class mat_t , class vec_t , class vec2_t >
  void o2scl_linalg::QR_QTvec (const size_t M, const size_t N, const mat_t &QR, const vec_t &tau, vec2_t &v)
  Form the product Q^T v from a QR factorized matrix.
- template<class mat1_t , class mat2_t , class mat3_t , class vec_t >
  void o2scl_linalg::QR_unpack (const size_t M, const size_t N, const mat1_t &QR, const vec_t &tau, mat2_t &Q, mat3_t &R)

  Unpack the QR matrix to the individual Q and R components.
- template<class mat1_t , class mat2_t , class vec1_t , class vec2_t >
  void o2scl_linalg::QR_update (size_t M, size_t N, mat1_t &Q, mat2_t &R, vec1_t &w, vec2_t &v)
  Update a QR factorisation for A= Q R, A' = A + u v^T,.

47.26 smart_interp.h File Reference

File for "smart" interpolation routines.

`#include <o2scl/interp.h> #include <o2scl/vector.h>`

47.26.1 Detailed Description

In addition to the smart interpolation routines, this file contains the template functions vector_find_level(), vector_integ_linear() and vector_invert_enclosed_sum().

Definition in file smart_interp.h.

Data Structures

- class smart_interp< vec_t, svec_t >
  Smart interpolation class.
- class smart_interp_vec< vec_t, svec_t, alloc_vec_t, alloc_t >
  Smart interpolation class with pre-specified vectors.
- class sma_interp< n >
  A specialization of smart_interp for C-style double arrays.
- class sma_interp_vec< arr_t >
  A specialization of smart_interp_vec for C-style double arrays.

Typedefs

- typedef smart_interp < ovector_const_view, ovector_const_subvector > **sm_interp**
- typedef smart_interp_vec < ovector_const_view, ovector_const_subvector, ovector, ovector_alloc > **sm_interp_vec**

Functions

- template<class vec_t , class vec2_t >
  int vector_find_level (double level, size_t n, vec_t &x, vec2_t &y, ovector &locs)
  Perform inverse linear interpolation.
- template<class vec_t >
  double vector_integ_linear (size_t n, vec_t &x, vec_t &y)
  Compute the integral over `y(x)` using linear interpolation.
- template<class vec_t >
  int vector_invert_enclosed_sum (double sum, size_t n, vec_t &x, vec_t &y, double &lev)
  Compute the endpoints which enclose the regions whose integral is equal to `sum`.

47.26.2 Function Documentation

47.26.2.1 template<class vec_t , class vec2_t > int vector_find_level (double *level*, size_t *n*, vec_t & *x*, vec2_t & *y*, ovector & *locs*)

This function performs inverse linear interpolation of the data defined by `x` and `y`, finding all points in `x` which have the property level $= y(x)$. All points for which this relation holds are put into the vector `locs`. The previous information contained in vector `locs` before the function call is destroyed.

This is the 1-dimensional analog of finding contour levels as the contour class does for 2 dimensions.

Definition at line 867 of file smart_interp.h.

47.26.2.2 template<class vec_t > int vector_invert_enclosed_sum (double *sum*, size_t *n*, vec_t & *x*, vec_t & *y*, double & *lev*)

Defining a new function, $g(y_0)$ which takes as input any y-value, y_0 from the function $y(x)$ (specified with the parameters `x` and `y`) and outputs the integral of the function $y(x)$ over all regions where $y(x) > y_0$. This function inverts $g(y)$, taking the value of an integral as input, and returns the corresponding y-value in the variable `lev`.

In order to make sure that the intepretation of the integral is unambiguous, this function requires that the first and last values of `y` are equal, i.e. `y[0]==y[n-1]`.

This function is particularly useful, for example, in computing the region which defines 68% around a peak of data, thus providing approximate 1 σ limits.

Linear interpolation is used to describe the function g, and the precision of this function is limited by this assumption. This function may also sometimes fail if `sum` is very close to the minimum or maximum value of the function g.

Definition at line 938 of file smart_interp.h.

47.27 string_conv.h File Reference

Various string conversion functions.

```
#include <iostream> #include <cmath> #include <string> #include <fstream> #include <sstream> ×
#include <o2scl/lib_settings.h> #include <gsl/gsl_ieee_utils.h>
```

47.27.1 Detailed Description

Definition in file string_conv.h.

Functions

- std::string ptos (void *p)
  Convert a pointer to a string.
- std::string itos (int x)
  Convert an integer to a string.

- std::string uitos (size_t x)
  Convert a size_t to a string.
- std::string btos (bool b)
  Convert a boolean value to a string.
- std::string dtos (double x, int prec=6, bool auto_prec=false)
  Convert a double to a string.
- size_t size_of_exponent (double x)
  Returns the number of characters required to display the exponent of x in scientific mode.
- std::string dtos (double x, std::ostream &format)
  Convert a double to a string using a specified format.
- int stoi (std::string s, bool err_on_fail=true)
  Convert a string to an integer.
- size_t stoui (std::string s, bool err_on_fail=true)
  Convert a string to a size_t.
- bool stob (std::string s, bool err_on_fail=true)
  Convert a string to a boolean value.
- double stod (std::string s, bool err_on_fail=true)
  Convert a string to a double.
- std::string double_to_ieee_string (const double *x)
  Convert a double to a string containing IEEE representation.
- bool has_minus_sign (double *x)
  Find out if the number pointed to by x has a minus sign.
- bool is_number (std::string s)
  Return true if the string s is likely a integral or floating point number.

47.27.2 Function Documentation

47.27.2.1 std::string ptos (void * *p*)

This uses an `ostringstream` to convert a pointer to a string and is architecture-dependent.

47.27.2.2 std::string btos (bool *b*)

This returns `"1"` for true and `"0"` for false.

47.27.2.3 std::string dtos (double *x*, int *prec* = `6`, bool *auto_prec* = `false`)

If `auto_prec` is false, then the number is converted to a string in the `ios::scientific` mode, otherwise, neither the scientific or fixed mode flags are set and the number is converted to a string in "automatic" mode.

47.27.2.4 size_t size_of_exponent (double *x*)

This returns 2 or 3, depending on whether or not the absolute magnitude of the exponent is greater than or equal to 100. It uses `stringstream` to convert the number to a string and counts the number of characters directly.

47.27.2.5 int stoi (std::string *s*, bool *err_on_fail* = `true`)

If `err_on_fail` is true and the conversion fails, this function calls the error handler, otherwise this function just returns zero.

47.27.2.6 size_t stoui (std::string *s*, bool *err_on_fail* = `true`)

If `err_on_fail` is true and the conversion fails, this function calls the error handler, otherwise this function just returns zero.

47.27.2.7 bool stob (std::string *s*, bool *err_on_fail* = `true`)

This returns true if only if the string has at least one character and the first non-whitespace character is either `t`, `T`, or one of the numbers 1 through 9.

If `err_on_fail` is true and the conversion fails, this function calls the error handler, otherwise this function just returns false.

47.27.2.8 double stod (std::string *s*, bool *err_on_fail* = `true`)

If `err_on_fail` is true and the conversion fails, this function calls the error handler, otherwise this function just returns 0.0.

47.27.2.9 std::string double_to_ieee_string (const double * *x*)

Modeled after the GSL function `gsl_ieee_fprintf_double()`, but converts to a `string` instead of a `FILE *`.

47.27.2.10 bool has_minus_sign (double * *x*)

This function returns true if the number pointed to by `x` has a minus sign using the GSL IEEE functions. It is useful, for example, in distinguishing "-0.0" from "+0.0".

47.27.2.11 bool is_number (std::string *s*)

Note

The test employed is not exhaustive and this function may return `true` for some numbers and may return `false` for some non-numbers.

47.28 svdstep_base.h File Reference

File for SVD decomposition.

47.28.1 Detailed Description

Definition in file svdstep_base.h.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Functions

- template<class vec_t , class vec2_t >
  int o2scl_linalg::chop_small_elements (size_t N, vec_t &d, vec2_t &f)
  Desc.
- template<class vec_t , class vec2_t >
  double o2scl_linalg::trailing_eigenvalue (size_t n, const vec_t &d, const vec_t &f)
  Desc.
- int o2scl_linalg::create_schur (double d0, double f0, double d1, double &c, double &s)
  Desc.
- template<class vec_t , class vec2_t , class mat_t , class mat2_t >
  int o2scl_linalg::svd2 (size_t M, size_t N, vec_t &d, vec2_t &f, mat_t &U, mat2_t &V)
  Desc.
- template<class vec_t , class vec2_t , class mat_t >
  int o2scl_linalg::chase_out_intermediate_zero (size_t M, size_t n, vec_t &d, vec2_t &f, mat_t &U, size_t k0)
  Desc.
- template<class vec_t , class vec2_t , class mat_t >
  int o2scl_linalg::chase_out_trailing_zero (size_t N, size_t n, vec_t &d, vec2_t &f, mat_t &V)
  Desc.
- template<class vec_t , class vec2_t , class mat_t , class mat2_t >
  int o2scl_linalg::qrstep (size_t M, size_t N, size_t n, vec_t &d, vec2_t &f, mat_t &U, mat2_t &V)
  Desc.

47.29 tensor.h File Reference

File for definitions of tensors.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <gsl/gsl_matrix.h> #include <gsl/gsl_ieee_utils.h> #include <o2scl/err_hnd.h> ×
#include <o2scl/uvector_tlate.h> #include <o2scl/umatrix_tlate.h> #include <o2scl/smart-
_interp.h>
```

47.29.1 Detailed Description

Definition in file tensor.h.

Data Structures

- class tensor
  Tensor class with arbitrary dimensions.
- class tensor_grid
  Tensor class with arbitrary dimensions with a grid.
- class tensor1
  Rank 1 tensor.
- class tensor_grid1
  Rank 1 tensor with a grid.
- class tensor2
  Rank 2 tensor.
- class tensor_grid2
  Rank 2 tensor with a grid.
- class tensor3
  Rank 3 tensor.
- class tensor_grid3
  Rank 3 tensor with a grid.
- class tensor4
  Rank 4 tensor.
- class tensor_grid4
  Rank 4 tensor with a grid.

47.30 tensor_old.h File Reference

File for definitions of tensor_olds.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <gsl/gsl_matrix.h> #include <gsl/gsl_ieee_utils.h> #include <o2scl/err_hnd.h> ×
#include <o2scl/uvector_tlate.h> #include <o2scl/umatrix_tlate.h> #include <o2scl/smart-
_interp.h>
```

47.30.1 Detailed Description

Definition in file tensor_old.h.

Data Structures

- class tensor_old
  Tensor_Old class with arbitrary dimensions.
- class tensor_old_grid
  Tensor_Old class with arbitrary dimensions with a grid.

- class tensor_old1
  Rank 1 tensor_old.
- class tensor_old_grid1
  Rank 2 tensor_old with a grid.
- class tensor_old2
  Rank 2 tensor_old.
- class tensor_old_grid2
  Rank 2 tensor_old with a grid.
- class tensor_old3
  Rank 3 tensor_old.
- class tensor_old_grid3
  Rank 3 tensor_old with a grid.
- class tensor_old4
  Rank 4 tensor_old.
- class tensor_old_grid4
  Rank 4 tensor_old with a grid.

47.31 tridiag_base.h File Reference

File for solving tridiagonal systems.

`#include <o2scl/array.h> #include <o2scl/uvector_tlate.h>`

47.31.1 Detailed Description

Definition in file tridiag_base.h.

Data Structures

- class o2scl_linalg::pointer_2_mem
  Allocation object for 2 C-style arrays of equal size.
- class o2scl_linalg::pointer_4_mem
  Allocation object for 4 C-style arrays of equal size.
- class o2scl_linalg::pointer_5_mem
  Allocation object for 5 C-style arrays of equal size.
- class o2scl_linalg::uvector_2_mem
  Allocation object for 2 arrays of equal size.
- class o2scl_linalg::uvector_4_mem
  Allocation object for 4 arrays of equal size.
- class o2scl_linalg::uvector_5_mem
  Allocation object for 5 arrays of equal size.

Namespaces

- namespace o2scl_linalg
  The namespace for linear algebra classes and functions.

Functions

- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class mem_t , class mem_vec_t >
  void o2scl_linalg::solve_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N, mem_t &m)
  Solve a symmetric tridiagonal linear system with user-specified memory.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t , class mem_t , class mem_vec_t >
  void o2scl_linalg::solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N, mem_t &m)

Solve an asymmetric tridiagonal linear system with user-specified memory.

- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class mem_t , class mem_vec_t >
  void o2scl_linalg::solve_cyc_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N, mem_t &m)
  Solve a symmetric cyclic tridiagonal linear system with user specified memory.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t , class mem_t , class mem_vec_t >
  void o2scl_linalg::solve_cyc_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N, mem_t &m)
  Solve an asymmetric cyclic tridiagonal linear system with user-specified memory.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t >
  void o2scl_linalg::solve_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N)
  Solve a symmetric tridiagonal linear system.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t >
  void o2scl_linalg::solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N)
  Solve an asymmetric tridiagonal linear system.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t >
  void o2scl_linalg::solve_cyc_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec3_t &b, vec4_t &x, size_t N)
  Solve a symmetric cyclic tridiagonal linear system.
- template<class vec_t , class vec2_t , class vec3_t , class vec4_t , class vec5_t >
  void o2scl_linalg::solve_cyc_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec3_t &belowdiag, const vec4_t &rhs, vec5_t &x, size_t N)
  Solve an asymmetric cyclic tridiagonal linear system.

47.32 umatrix_cx_tlate.h File Reference

File for definitions of matrices.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <gsl/gsl_matrix.h> #include <gsl/gsl_ieee_utils.h> #include <o2scl/err_hnd.h> ×
#include <o2scl/uvector_tlate.h> #include <o2scl/uvector_cx_tlate.h>
```

47.32.1 Detailed Description

Definition in file umatrix_cx_tlate.h.

Data Structures

- class umatrix_cx_view_tlate< data_t, complex_t >
  A matrix view of complex numbers.
- class umatrix_cx_tlate< data_t, complex_t >
  A matrix of double-precision numbers.
- class umatrix_cx_row_tlate< data_t, complex_t >
  Create a vector from a row of a matrix.
- class umatrix_cx_const_row_tlate< data_t, complex_t >
  Create a const vector from a row of a matrix.
- class umatrix_cx_alloc
  A simple class to provide an `allocate()` function for umatrix_cx.
- class ufmatrix_cx< N, M >
  A matrix where the memory allocation is performed in the constructor.

Typedefs

- typedef umatrix_cx_tlate < double, gsl_complex > umatrix_cx
  umatrix_cx typedef

- typedef umatrix_cx_view_tlate < double, gsl_complex > umatrix_cx_view
  umatrix_cx_view typedef
- typedef umatrix_cx_row_tlate < double, gsl_complex > umatrix_cx_row
  umatrix_cx_row typedef
- typedef umatrix_cx_const_row_tlate < double, gsl_complex > umatrix_cx_const_row
  umatrix_cx_const_row typedef

Functions

- template<class data_t , class complex_t >
  std::ostream & operator<< (std::ostream &os, const umatrix_cx_view_tlate< data_t, complex_t > &v)
  A operator for naive matrix output.

47.32.2 Function Documentation

47.32.2.1 template<class data_t , class complex_t > std::ostream& operator<< (std::ostream & *os*, const umatrix_cx_view_tlate< data_t, complex_t > & *v*)

This outputs all of the matrix elements. Each row is output with an endline character at the end of each row. Positive values are preceeded by an extra space. A 2x2 example:

```
-3.751935e-05 -6.785864e-04
-6.785864e-04  1.631984e-02
```

The function `gsl_ieee_double_to_rep()` is used to determine the sign of a number, so that "-0.0" as distinct from "+0.0" is handled correctly.

Note

At the moment, this function assumes that scientific mode is on and showpos is off.

Idea for Future Make this function work even when scientific mode is not on, either by converting to scientific mode and converting back, or by leaving scientific mode off and padding with spaces

Definition at line 671 of file umatrix_cx_tlate.h.

47.33 umatrix_tlate.h File Reference

File for definitions of matrices.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <gsl/gsl_matrix.h> #include <gsl/gsl_ieee_utils.h> #include <o2scl/err_hnd.h> ×
#include <o2scl/uvector_tlate.h>
```

47.33.1 Detailed Description

Definition in file umatrix_tlate.h.

Data Structures

- class umatrix_const_view_tlate< data_t >
  A matrix view of double-precision numbers.
- class umatrix_base_tlate< data_t >

A matrix view of double-precision numbers.
- class umatrix_view_tlate< data_t >
  A matrix view of double-precision numbers.
- class umatrix_tlate< data_t >
  A matrix of double-precision numbers.
- class umatrix_row_tlate< data_t >
  Create a vector from a row of a matrix.
- class umatrix_const_row_tlate< data_t >
  Create a const vector from a row of a matrix.
- class umatrix_alloc
  A simple class to provide an `allocate()` function for umatrix.
- class ufmatrix< N, M >
  A matrix where the memory allocation is performed in the constructor.

Typedefs

- typedef umatrix_tlate< double > umatrix
  umatrix typedef
- typedef umatrix_view_tlate < double > umatrix_view
  umatrix_view typedef
- typedef umatrix_base_tlate < double > umatrix_base
  umatrix_base typedef
- typedef umatrix_const_view_tlate < double > umatrix_const_view
  umatrix_const_view typedef
- typedef umatrix_row_tlate< double > umatrix_row
  umatrix_row typedef
- typedef umatrix_const_row_tlate < double > umatrix_const_row
  umatrix_const_row typedef
- typedef umatrix_tlate< int > umatrix_int
  umatrix_int typedef
- typedef umatrix_view_tlate< int > umatrix_int_view
  umatrix_int_view typedef
- typedef umatrix_const_view_tlate< int > umatrix_int_const_view
  umatrix_int_const_view typedef
- typedef umatrix_base_tlate< int > umatrix_int_base
  umatrix_int_base typedef
- typedef umatrix_row_tlate< int > umatrix_int_row
  umatrix_int_row typedef
- typedef umatrix_const_row_tlate< int > umatrix_int_const_row
  umatrix_int_const_row typedef

Functions

- template<class data_t >
  std::ostream & operator<< (std::ostream &os, const umatrix_const_view_tlate< data_t > &v)
  A operator for simple matrix output.

47.33.2 Function Documentation

47.33.2.1 template<class data_t > std::ostream& operator<< (std::ostream & *os*, const umatrix_const_view_tlate< data_t > & *v*)

This outputs all of the matrix elements. Each row is output with an endline character at the end of each row. Positive values are preceeded by an extra space. A 2x2 example:

```
-3.751935e-05 -6.785864e-04
-6.785864e-04  1.631984e-02
```

The function `gsl_ieee_double_to_rep()` is used to determine the sign of a number, so that "-0.0" as distinct from "+0.0" is handled correctly.

Idea for Future This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Definition at line 878 of file umatrix_tlate.h.

47.34 uvector_cx_tlate.h File Reference

File for definitions of complex unit-stride vectors.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <vector> #include <o2scl/err_hnd.h> #include <gsl/gsl_vector.h>
```

47.34.1 Detailed Description

Definition in file uvector_cx_tlate.h.

Data Structures

- class uvector_cx_view_tlate< data_t, complex_t >
  A vector view of complex numbers with unit stride.
- class uvector_cx_tlate< data_t, complex_t >
  A vector of double-precision numbers with unit stride.
- class uvector_cx_array_tlate< data_t, complex_t >
  Create a vector from an array.
- class uvector_cx_subvector_tlate< data_t, complex_t >
  Create a vector from a subvector of another.
- class uvector_cx_const_array_tlate< data_t, complex_t >
  Create a vector from an array.
- class uvector_cx_const_subvector_tlate< data_t, complex_t >
  Create a vector from a subvector of another.

Typedefs

- typedef uvector_cx_tlate < double, gsl_complex > uvector_cx
  uvector_cx typedef
- typedef uvector_cx_view_tlate < double, gsl_complex > uvector_cx_view
  uvector_cx_view typedef
- typedef uvector_cx_array_tlate < double, gsl_complex > uvector_cx_array
  uvector_cx_array typedef
- typedef uvector_cx_subvector_tlate < double, gsl_complex > uvector_cx_subvector
  uvector_cx_subvector typedef
- typedef uvector_cx_const_array_tlate < double, gsl_complex > uvector_cx_const_array
  uvector_cx_const_array typedef
- typedef uvector_cx_const_subvector_tlate < double, gsl_complex > uvector_cx_const_subvector
  uvector_cx_const_subvector typedef

Functions

- template<class data_t , class complex_t >
  std::ostream & operator<< (std::ostream &os, const uvector_cx_view_tlate< data_t, complex_t > &v)
  A operator for naive vector output.

47.34.2 Function Documentation

47.34.2.1 template<class data_t , class complex_t > std::ostream& operator<< (std::ostream & *os,* const uvector_cx_view_tlate< data_t, complex_t > & *v*)

This outputs all of the vector elements. All of these are separated by one space character, though no trailing space or `endl` is sent to the output.

Definition at line 640 of file uvector_cx_tlate.h.

47.35 uvector_tlate.h File Reference

File for definitions of unit-stride vectors.

```
#include <iostream> #include <cstdlib> #include <string> #include <fstream> #include <sstream> ×
#include <vector> #include <gsl/gsl_vector.h> #include <o2scl/err_hnd.h> #include <o2scl/string-
_conv.h> #include <o2scl/array.h> #include <o2scl/vector.h>
```

47.35.1 Detailed Description

Definition in file uvector_tlate.h.

Data Structures

- class uvector_const_view_tlate< data_t >
  A const vector view with unit stride.
- class uvector_base_tlate< data_t >
  A base class for uvector and uvector_view.
- class uvector_view_tlate< data_t >
  A base class for uvectors.
- class uvector_tlate< data_t >
  A vector with unit stride.
- class uvector_array_tlate< data_t >
  Create a vector from an array.
- class uvector_subvector_tlate< data_t >
  Create a vector from a subvector of another.
- class uvector_const_array_tlate< data_t >
  Create a vector from an const array.
- class uvector_const_subvector_tlate< data_t >
  Create a const vector from a subvector of another vector.
- class uvector_alloc
  A simple class to provide an `allocate()` function for uvector.
- class uvector_int_alloc
  A simple class to provide an `allocate()` function for uvector_int.
- class uvector_size_t_alloc
  A simple class to provide an `allocate()` function for uvector_size_t.
- class ufvector< N >
  A vector with unit-stride where the memory allocation is performed in the constructor.

Typedefs

- typedef uvector_tlate< double > uvector
  uvector typedef
- typedef uvector_view_tlate < double > uvector_view
  uvector_view typedef
- typedef uvector_base_tlate < double > uvector_base

uvector_base typedef
- typedef uvector_const_view_tlate < double > uvector_const_view
  uvector_const_view typedef
- typedef uvector_array_tlate < double > uvector_array
  uvector_array typedef
- typedef uvector_subvector_tlate < double > uvector_subvector
  uvector_subvector typedef
- typedef uvector_const_array_tlate < double > uvector_const_array
  uvector_const_array typedef
- typedef uvector_const_subvector_tlate < double > uvector_const_subvector
  uvector_const_subvector typedef
- typedef uvector_tlate< int > uvector_int
  uvector_int typedef
- typedef uvector_view_tlate< int > uvector_int_view
  uvector_int_view typedef
- typedef uvector_base_tlate< int > uvector_int_base
  uvector_int_base typedef
- typedef uvector_const_view_tlate< int > uvector_int_const_view
  uvector_int_const_view typedef
- typedef uvector_array_tlate< int > uvector_int_array
  uvector_int_array typedef
- typedef uvector_subvector_tlate< int > uvector_int_subvector
  uvector_int_subvector typedef
- typedef uvector_const_array_tlate< int > uvector_int_const_array
  uvector_int_const_array typedef
- typedef uvector_const_subvector_tlate < int > uvector_int_const_subvector
  uvector_int_const_subvector typedef
- typedef uvector_tlate< size_t > uvector_size_t
  uvector_size_t typedef
- typedef uvector_view_tlate < size_t > uvector_size_t_view
  uvector_size_t_view typedef
- typedef uvector_base_tlate < size_t > uvector_size_t_base
  uvector_size_t_base typedef
- typedef uvector_const_view_tlate < size_t > uvector_size_t_const_view
  uvector_size_t_const_view typedef
- typedef uvector_array_tlate < size_t > uvector_size_t_array
  uvector_size_t_array typedef
- typedef uvector_subvector_tlate < size_t > uvector_size_t_subvector
  uvector_size_t_subvector typedef
- typedef uvector_const_array_tlate < size_t > uvector_size_t_const_array
  uvector_size_t_const_array typedef
- typedef uvector_const_subvector_tlate < size_t > uvector_size_t_const_subvector
  uvector_size_t_const_subvector typedef

Functions

- template<class data_t >
  std::ostream & operator<< (std::ostream &os, const uvector_const_view_tlate< data_t > &v)
  A operator for simple vector output.

47.35.2 Function Documentation

47.35.2.1 template<class data_t > std::ostream& operator<< (std::ostream & *os*, const uvector_const_view_tlate< data_t > & *v*)

This outputs all of the vector elements. All of these are separated by one space character, though no trailing space or `endl` is sent to the output. If the vector is empty, nothing is done.

Definition at line 1006 of file uvector_tlate.h.

47.36 vec_arith.h File Reference

Vector and matrix arithmetic.

`#include <iostream> #include <complex> #include <o2scl/ovector_tlate.h> #include <o2scl/omatrix_tlate.h> #include <o2scl/uvector_tlate.h> #include <o2scl/umatrix_tlate.h> #include <o2scl/ovect_cx_tlate.h> #include <o2scl/omatrix_cx_tlate.h> #include <o2scl/uvector_cx_tlate.h> #include <o2scl/umatrix_cx_tlate.h>`

47.36.1 Detailed Description

By default, the following operators are defined:

```
ovector operator+(ovector_const_view &x, ovector_const_view &y);
ovector operator+(ovector_const_view &x, uvector_const_view &y);
ovector operator+(uvector_const_view &x, ovector_const_view &y);
uvector operator+(uvector_const_view &x, uvector_const_view &y);

ovector_cx operator+(ovector_cx_view &x, ovector_cx_view &y);
ovector_cx operator+(ovector_cx_view &x, uvector_cx_view &y);
ovector_cx operator+(uvector_cx_view &x, ovector_cx_view &y);
uvector_cx operator+(uvector_cx_view &x, uvector_cx_view &y);

ovector operator-(ovector_const_view &x, ovector_const_view &y);
ovector operator-(ovector_const_view &x, uvector_const_view &y);
ovector operator-(uvector_const_view &x, ovector_const_view &y);
uvector operator-(uvector_const_view &x, uvector_const_view &y);

ovector_cx operator-(ovector_cx_view &x, ovector_cx_view &y);
ovector_cx operator-(ovector_cx_view &x, uvector_cx_view &y);
ovector_cx operator-(uvector_cx_view &x, ovector_cx_view &y);
uvector_cx operator-(uvector_cx_view &x, uvector_cx_view &y);

ovector operator*(omatrix_const_view &x, ovector_const_view &y);
ovector operator*(omatrix_const_view &x, uvector_const_view &y);
ovector operator*(umatrix_const_view &x, ovector_const_view &y);
uvector operator*(umatrix_const_view &x, uvector_const_view &y);

ovector_cx operator*(omatrix_cx_view &x, ovector_cx_view &y);
ovector_cx operator*(omatrix_cx_view &x, uvector_cx_view &y);
ovector_cx operator*(umatrix_cx_view &x, ovector_cx_view &y);
uvector_cx operator*(umatrix_cx_view &x, uvector_cx_view &y);

ovector operator*(ovector_const_view &x, omatrix_const_view &y);
ovector operator*(ovector_const_view &x, umatrix_const_view &y);
ovector operator*(uvector_const_view &x, omatrix_const_view &y);
uvector operator*(ovector_const_view &x, umatrix_const_view &y);

ovector trans_mult(ovector_const_view &x, omatrix_const_view &y);
ovector trans_mult(ovector_const_view &x, umatrix_const_view &y);
ovector trans_mult(uvector_const_view &x, omatrix_const_view &y);
uvector trans_mult(ovector_const_view &x, umatrix_const_view &y);

double dot(ovector_const_view &x, ovector_const_view &y);
double dot(ovector_const_view &x, uvector_const_view &y);
double dot(uvector_const_view &x, ovector_const_view &y);
double dot(uvector_const_view &x, uvector_const_view &y);

double dot(ovector_cx_view &x, ovector_cx_view &y);
double dot(ovector_cx_view &x, uvector_cx_view &y);
double dot(uvector_cx_view &x, ovector_cx_view &y);
double dot(uvector_cx_view &x, uvector_cx_view &y);

ovector operator*(double x, ovector_const_view &y);
uvector operator*(double x, uvector_const_view &y);
ovector operator*(ovector_const_view &x, double y);
uvector operator*(uvector_const_view &x, double y);
```

```
ovector pair_prod(ovector_const_view &x, ovector_const_view &y);
ovector pair_prod(ovector_const_view &x, uvector_const_view &y);
ovector pair_prod(uvector_const_view &x, ovector_const_view &y);
uvector pair_prod(uvector_const_view &x, uvector_const_view &y);

ovector_cx pair_prod(ovector_cx_view &x, ovector_const_view &y);
ovector_cx pair_prod(ovector_cx_view &x, uvector_const_view &y);
ovector_cx pair_prod(uvector_cx_view &x, ovector_const_view &y);
uvector_cx pair_prod(uvector_cx_view &x, uvector_const_view &y);
ovector_cx pair_prod(ovector_const_view &x, ovector_cx_view &y);
ovector_cx pair_prod(ovector_const_view &x, uvector_cx_view &y);
ovector_cx pair_prod(uvector_const_view &x, ovector_cx_view &y);
uvector_cx pair_prod(uvector_const_view &x, uvector_cx_view &y);
ovector_cx pair_prod(ovector_cx_view &x, ovector_cx_view &y);
ovector_cx pair_prod(ovector_cx_view &x, uvector_cx_view &y);
ovector_cx pair_prod(uvector_cx_view &x, ovector_cx_view &y);
uvector_cx pair_prod(uvector_cx_view &x, uvector_cx_view &y);

bool operator==(ovector_const_view &x, ovector_const_view &y);
bool operator==(ovector_const_view &x, uvector_const_view &y);
bool operator==(uvector_const_view &x, ovector_const_view &y);
bool operator==(uvector_const_view &x, uvector_const_view &y);

bool operator!=(ovector_const_view &x, ovector_const_view &y);
bool operator!=(ovector_const_view &x, uvector_const_view &y);
bool operator!=(uvector_const_view &x, ovector_const_view &y);
bool operator!=(uvector_const_view &x, uvector_const_view &y);
```

Note

This used to be in a separate namespace, called `o2scl_arith`, but this causes problems with Koenig lookup in template classes for operator∗() when defined for vector addition (for example).

Idea for Future Define operators for complex vector ∗ real matrix

Idea for Future Define == and != for complex vectors

Idea for Future These should be replaced by the BLAS routines where possible?

Definition in file vec_arith.h.

Defines

- #define O2SCL_OP_VEC_VEC_ADD(vec1, vec2, vec3)
  The header macro for vector-vector addition.
- #define O2SCL_OP_VEC_VEC_SUB(vec1, vec2, vec3)
  The header macro for vector-vector subtraction.
- #define O2SCL_OP_MAT_VEC_MULT(vec1, vec2, mat)
  The header macro for matrix-vector (right) multiplication.
- #define O2SCL_OP_CMAT_CVEC_MULT(vec1, vec2, mat)
  The header macro for complex matrix-vector (right) multiplication.
- #define O2SCL_OP_VEC_MAT_MULT(vec1, vec2, mat)
  The header macro for vector-matrix (left) multiplication.
- #define O2SCL_OP_TRANS_MULT(vec1, vec2, mat)
  The header macro for the `trans_mult` form of vector ∗ matrix.
- #define O2SCL_OP_DOT_PROD(dtype, vec1, vec2)
  The header macro for vector scalar (dot) product.
- #define O2SCL_OP_CX_DOT_PROD(dtype, vec1, vec2)
  The header macro for complex vector scalar (dot) product.
- #define O2SCL_OP_SCA_VEC_MULT(dtype, vecv, vec)

The header macro for scalar-vector multiplication.
- #define O2SCL_OP_VEC_SCA_MULT(dtype, vecv, vec)
  The header macro for vector-scalar multiplication.
- #define O2SCL_OP_VEC_VEC_PRO(vec1, vec2, vec3)
  The header macro for pairwise vector ∗ vector (where either vector can be real or complex)
- #define O2SCL_OPSRC_VEC_VEC_ADD(vec1, vec2, vec3)
  The source code macro for vector-vector addition.
- #define O2SCL_OPSRC_VEC_VEC_SUB(vec1, vec2, vec3)
  The source code macro for vector-vector subtraction.
- #define O2SCL_OPSRC_MAT_VEC_MULT(vec1, vec2, mat)
  The source code macro for matrix ∗ vector.
- #define O2SCL_OPSRC_CMAT_CVEC_MULT(vec1, vec2, mat)
  The source code macro for complex matrix ∗ complex vector.
- #define O2SCL_OPSRC_VEC_MAT_MULT(vec1, vec2, mat)
  The source code macro for the operator form of vector ∗ matrix.
- #define O2SCL_OPSRC_TRANS_MULT(vec1, vec2, mat)
  The source code macro for the `trans_mult` form of vector ∗ matrix.
- #define O2SCL_OPSRC_DOT_PROD(dtype, vec1, vec2)
  The source code macro for a vector dot product.
- #define O2SCL_OPSRC_CX_DOT_PROD(dtype, vec1, vec2)
  The source code macro for a complex vector dot product.
- #define O2SCL_OPSRC_SCA_VEC_MULT(dtype, vecv, vec)
  The source code macro for vector=scalar∗vector.
- #define O2SCL_OPSRC_VEC_SCA_MULT(dtype, vecv, vec)
  The source code macro for vector=vector∗scalar.
- #define O2SCL_OPSRC_VEC_VEC_PRO(vec1, vec2, vec3)
  The source code macro for pairwise vector ∗ vector (where either vector can be real or complex)
- #define O2SCL_OP_VEC_VEC_EQUAL(vec1, vec2)
  The header macro for vector==vector.
- #define O2SCL_OPSRC_VEC_VEC_EQUAL(vec1, vec2)
  The source code macro vector==vector.
- #define O2SCL_OP_VEC_VEC_NEQUAL(vec1, vec2)
  The header macro for vector!=vector.
- #define O2SCL_OPSRC_VEC_VEC_NEQUAL(vec1, vec2)
  The source code macro vector!=vector.

47.36.2 Define Documentation

47.36.2.1 #define O2SCL_OP_VEC_VEC_ADD(*vec1, vec2, vec3*)

Value:

```
vec1 operator+  \
    (const vec2 &x, const vec3 &y);
```

Given types `vec1`, `vec2`, and `vec_3`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator+(const vec2 &x, const vec3 &y);
```

The corresponding definition is given in O2SCL_OPSRC_VEC_VEC_ADD.

Note

This used to be in a separate namespace, called `o2scl_arith`, but this causes problems with Koenig lookup in template classes for operator∗() when defined for vector addition (for example).

Definition at line 158 of file vec_arith.h.

47.36.2.2 #define O2SCL_OP_VEC_VEC_SUB(*vec1, vec2, vec3*)

Value:

```
vec1 operator-  \
    (const vec2 &x, const vec3 &y);
```

Given types `vec1`, `vec2`, and `vec_3`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator-(const vec2 &x, const vec3 &y);
```

The corresponding definition is given in O2SCL_OPSRC_VEC_VEC_SUB.

Definition at line 199 of file vec_arith.h.

47.36.2.3 #define O2SCL_OP_MAT_VEC_MULT(*vec1, vec2, mat*)

Value:

```
vec1 operator*  \
    (const mat &m, const vec2 &x);
```

Given types `vec1`, `vec2`, and `mat`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator*(const mat &m, const vec3 &x);
```

See also the blas version of this function dgemv().

The corresponding definition is given in O2SCL_OPSRC_MAT_VEC_MULT.

Definition at line 243 of file vec_arith.h.

47.36.2.4 #define O2SCL_OP_CMAT_CVEC_MULT(*vec1, vec2, mat*)

Value:

```
vec1 operator*  \
    (const mat &m, const vec2 &x);
```

Given types `vec1`, `vec2`, and `mat`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator*(const mat &m, const vec3 &x);
```

The corresponding definition is given in O2SCL_OPSRC_CMAT_CVEC_MULT.

Definition at line 278 of file vec_arith.h.

47.36.2.5 #define O2SCL_OP_VEC_MAT_MULT(*vec1, vec2, mat*)

Value:

```
vec1 operator*  \
    (const vec2 &x, const mat &m);
```

Given types `vec1`, `vec2`, and `mat`, this macro provides the function declaration for adding two vectors using the form

```
vec1 operator*(const vec3 &x, const mat &m);
```

The corresponding definition is given in O2SCL_OPSRC_VEC_MAT_MULT.

Definition at line 312 of file vec_arith.h.

47.36.2.6 #define O2SCL_OP_TRANS_MULT(*vec1, vec2, mat*)

Value:

```
vec1 trans_mult \
    (const vec2 &x, const mat &m);
```

Definition at line 334 of file vec_arith.h.

47.36.2.7 #define O2SCL_OP_DOT_PROD(*dtype, vec1, vec2*)

Value:

```
dtype dot       \
    (const vec1 &x, const vec2 &y);
```

Given types `vec1`, `vec2`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
dtype operator*(const vec1 &x, const vec2 &y);
```

The corresponding definition is given in O2SCL_OPSRC_DOT_PROD.

Definition at line 367 of file vec_arith.h.

47.36.2.8 #define O2SCL_OP_CX_DOT_PROD(*dtype, vec1, vec2*)

Value:

```
dtype dot       \
    (const vec1 &x, const vec2 &y);
```

Given types `vec1`, `vec2`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
dtype operator*(const vec1 &x, const vec2 &y);
```

The corresponding definition is given in O2SCL_OPSRC_CX_DOT_PROD.

Definition at line 401 of file vec_arith.h.

47.36.2.9 #define O2SCL_OP_SCA_VEC_MULT(*dtype, vecv, vec*)

Value:

```
vec operator*   \
    (const dtype &x, const vecv &y);
```

Given types `vecv`, `vec`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
vec operator*(const dtype &x, const vecv &y);
```

The corresponding definition is given in O2SCL_OPSRC_SCA_VEC_MULT.

Definition at line 434 of file vec_arith.h.

47.36.2.10 #define O2SCL_OP_VEC_SCA_MULT(*dtype, vecv, vec*)

Value:

```
vec operator*   \
    (const vecv &x, const dtype &y);
```

Given types `vecv`, `vec`, and `dtype`, this macro provides the function declaration for adding two vectors using the form

```
vec operator*(const vecv &x, const dtype &y);
```

The corresponding definition is given in O2SCL_OPSRC_VEC_SCA_MULT.

Definition at line 461 of file vec_arith.h.

47.36.2.11 #define O2SCL_OP_VEC_VEC_PRO(*vec1, vec2, vec3*)

Value:

```
vec1 pair_prod  \
    (const vec2 &x, const vec3 &y);
```

Given types `vec1`, `vec2`, and `vec3`, this macro provides the function declaration for adding two vectors using the form

```
vec1 pair_prod(const vec2 &x, const vec3 &y);
```

The corresponding definition is given in O2SCL_OPSRC_VEC_VEC_PRO.

Definition at line 489 of file vec_arith.h.

47.36.2.12 #define O2SCL_OPSRC_VEC_VEC_ADD(*vec1, vec2, vec3*)

Value:

```
vec1 o2scl::operator+ \
      (const vec2 &x, const vec3 &y) {                                  \
      size_t m=x.size();                                                \
      if (y.size()<m) m=y.size();                                       \
      vec1 r(m);                                                        \
      for(size_t i=0;i<m;i++) {                                         \
        r[i]=x[i]+y[i];                                                 \
      }                                                                 \
      return r;                                                         \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_ADD

Definition at line 534 of file vec_arith.h.

47.36.2.13 #define O2SCL_OPSRC_VEC_VEC_SUB(*vec1, vec2, vec3*)

Value:

```
vec1 o2scl::operator- \
      (const vec2 &x, const vec3 &y) {                                  \
      size_t m=x.size();                                                \
      if (y.size()<m) m=y.size();                                       \
      vec1 r(m);                                                        \
      for(size_t i=0;i<m;i++) {                                         \
        r[i]=x[i]-y[i];                                                 \
      }                                                                 \
      return r;                                                         \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_SUB

Definition at line 550 of file vec_arith.h.

47.36.2.14 #define O2SCL_OPSRC_MAT_VEC_MULT(*vec1, vec2, mat*)

Value:

```
vec1 o2scl::operator* \
```

```
      (const mat &m, const vec2 &x) {                                   \
      size_t nr=m.rows();                                               \
      size_t nc=m.cols();                                               \
      vec1 res(nr);                                                     \
      for(size_t i=0;i<nr;i++) {                                        \
        double r=0.0;                                                   \
        for(size_t j=0;j<nc;j++) {                                      \
          r+=m[i][j]*x[j];                                              \
        }                                                               \
        res[i]=r;                                                       \
      }                                                                 \
      return res;                                                       \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_MAT_VEC_MULT

Definition at line 566 of file vec_arith.h.

47.36.2.15 #define O2SCL_OPSRC_CMAT_CVEC_MULT(*vec1, vec2, mat*)

Value:

```
vec1 o2scl::operator* \
      (const mat &m, const vec2 &x) {                                   \
      size_t nr=m.rows();                                               \
      size_t nc=m.cols();                                               \
      vec1 res(nr);                                                     \
      for(size_t i=0;i<nr;i++) {                                        \
        double re=0.0;                                                  \
        double im=0.0;                                                  \
        for(size_t j=0;j<nc;j++) {                                      \
          gsl_complex g=m[i][j]*x[j];                                   \
          re+=g.dat[0];                                                 \
          im+=g.dat[1];                                                 \
        }                                                               \
        res[i].dat[0]=re;                                               \
        res[i].dat[1]=im;                                               \
      }                                                                 \
      return res;                                                       \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_CMAT_CVEC_MULT

Definition at line 586 of file vec_arith.h.

47.36.2.16 #define O2SCL_OPSRC_VEC_MAT_MULT(*vec1, vec2, mat*)

Value:

```
vec1 o2scl::operator* \
      (const vec2 &x, const mat &m) {                                   \
      size_t nr=m.rows();                                               \
      size_t nc=m.cols();                                               \
      vec1 res(nr);                                                     \
      for(size_t j=0;j<nc;j++) {                                        \
        double r=0.0;                                                   \
        for(size_t i=0;i<nr;i++) {                                      \
          r+=x[i]*m[i][j];                                              \
        }                                                               \
        res[j]=r;                                                       \
      }                                                                 \
      return res;                                                       \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_MAT_MULT

Definition at line 610 of file vec_arith.h.

47.36.2.17 #define O2SCL_OPSRC_TRANS_MULT(*vec1, vec2, mat*)

Value:

```
vec1 o2scl::trans_mult \
      (const vec2 &x, const mat &m) {                                  \
      size_t nr=m.rows();                                              \
      size_t nc=m.cols();                                              \
      vec1 res(nr);                                                    \
      for(size_t j=0;j<nc;j++) {                                       \
        double r=0.0;                                                  \
        for(size_t i=0;i<nr;i++) {                                     \
          r+=x[i]*m[i][j];                                             \
        }                                                              \
        res[j]=r;                                                      \
      }                                                                \
      return res;                                                      \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_TRANS_MULT

Definition at line 631 of file vec_arith.h.

47.36.2.18 #define O2SCL_OPSRC_DOT_PROD(*dtype, vec1, vec2*)

Value:

```
dtype o2scl::dot        \
      (const vec1 &x, const vec2 &y) {                                 \
      size_t m=x.size();                                               \
      if (y.size()<m) m=y.size();                                      \
      dtype r=0;                                                       \
      for(size_t i=0;i<m;i++) {                                        \
        r+=x[i]*y[i];                                                  \
      }                                                                \
      return r;                                                        \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_DOT_PROD

Definition at line 651 of file vec_arith.h.

47.36.2.19 #define O2SCL_OPSRC_CX_DOT_PROD(*dtype, vec1, vec2*)

Value:

```
dtype o2scl::dot        \
      (const vec1 &x, const vec2 &y) {                                 \
      size_t m=x.size();                                               \
      if (y.size()<m) m=y.size();                                      \
      dtype r={{0.0,0.0}};                                             \
      for(size_t i=0;i<m;i++) {                                        \
        r+=x[i]*y[i];                                                  \
      }                                                                \
      return r;                                                        \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_CX_DOT_PROD

Definition at line 667 of file vec_arith.h.

47.36.2.20 #define O2SCL_OPSRC_SCA_VEC_MULT(*dtype, vecv, vec*)

Value:

```
vec o2scl::operator* \
```

```
      (const dtype &x, const vecv &y) {                                 \
      size_t m=y.size();                                                \
      vec r(m);                                                         \
      for(size_t i=0;i<m;i++) {                                         \
        r[i]=x*y[i];                                                    \
      }                                                                 \
      return r;                                                         \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_SCA_VEC_MULT

Definition at line 683 of file vec_arith.h.

47.36.2.21 #define O2SCL_OPSRC_VEC_SCA_MULT(*dtype, vecv, vec*)

Value:

```
vec o2scl::operator* \
      (const vecv &x, const dtype &y) {                                 \
      size_t m=x.size();                                                \
      vec r(m);                                                         \
      for(size_t i=0;i<m;i++) {                                         \
        r[i]=x[i]*y;                                                    \
      }                                                                 \
      return r;                                                         \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_SCA_MULT

Definition at line 698 of file vec_arith.h.

47.36.2.22 #define O2SCL_OPSRC_VEC_VEC_PRO(*vec1, vec2, vec3*)

Value:

```
vec1 o2scl::pair_prod \
      (const vec2 &x, const vec3 &y) {                                  \
      size_t m=x.size();                                                \
      if (y.size()<m) m=y.size();                                       \
      vec1 r(m);                                                        \
      for(size_t i=0;i<m;i++) {                                         \
        r[i]=x[i]*y[i];                                                 \
      }                                                                 \
      return r;                                                         \
    }
```

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_PRO

Definition at line 714 of file vec_arith.h.

47.36.2.23 #define O2SCL_OP_VEC_VEC_EQUAL(*vec1, vec2*)

Value:

```
bool operator== \
      (const vec1 &x, const vec2 &y);
```

Given types `vec1` and `vec2`, this macro provides the function declaration for vector equality comparisons using

```
bool operator==(const vec1 &x, const vec2 &y);
```

Note

Two vectors with different sizes are defined to be not equal, no matter what their contents.

The corresponding definition is given in O2SCL_OPSRC_VEC_VEC_EQUAL.

Definition at line 739 of file vec_arith.h.

47.36.2.24 #define O2SCL_OPSRC_VEC_VEC_EQUAL(*vec1, vec2*)

Value:

```
bool o2scl::operator== \
    (const vec1 &x, const vec2 &y) {                                    \
    size_t m=x.size();                                                  \
    size_t n=y.size();                                                  \
    if (m!=n) return false;                                             \
    for(size_t i=0;i<m;i++) {                                           \
      if (x[i]!=y[i]) return false;                                     \
    }                                                                   \
    return true;                                                        \
  }
```

Note

Two vectors with different sizes are defined to be not equal, no matter what their contents.

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_EQUAL

Definition at line 794 of file vec_arith.h.

47.36.2.25 #define O2SCL_OP_VEC_VEC_NEQUAL(*vec1, vec2*)

Value:

```
bool operator!= \
    (const vec1 &x, const vec2 &y);
```

Given types `vec1` and `vec2`, this macro provides the function declaration for vector inequality comparisons using

```
  bool operator==(const vec1 &x, const vec2 &y);
```

Note

Two vectors with different sizes are defined to be not equal, no matter what their contents.

The corresponding definition is given in O2SCL_OPSRC_VEC_VEC_NEQUAL.

Definition at line 819 of file vec_arith.h.

47.36.2.26 #define O2SCL_OPSRC_VEC_VEC_NEQUAL(*vec1, vec2*)

Value:

```
bool o2scl::operator!= \
    (const vec1 &x, const vec2 &y) {                                    \
    size_t m=x.size();                                                  \
    size_t n=y.size();                                                  \
    if (m!=n) return true;                                              \
    for(size_t i=0;i<m;i++) {                                           \
      if (x[i]!=y[i]) return true;                                      \
    }                                                                   \
    return false;                                                       \
  }
```

Note

Two vectors with different sizes are defined to be not equal, no matter what their contents.

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_NEQUAL

Definition at line 874 of file vec_arith.h.

47.37 vec_stats.h File Reference

File containing statistics template functions.

`#include <o2scl/err_hnd.h> #include <gsl/gsl_sort.h>`

47.37.1 Detailed Description

This file contains several function templates for computing statistics of vectors of double-precision data. It includes mean, median, variance, standard deviation, covariance, correlation, and other functions.

No additional range checking is done on the vectors.

Idea for Future Consider generalizing to other data types.

Definition in file vec_stats.h.

Functions

Vector functions

- template<class vec_t >
  double vector_mean (size_t n, const vec_t &data)
  Compute the mean of the first `n` elements of a vector.
- template<class vec_t >
  double vector_variance_fmean (size_t n, const vec_t &data, double mean)
  Compute variance with specified mean known in advance.
- template<class vec_t >
  double vector_variance (size_t n, const vec_t &data, double mean)
  Compute the variance with specified mean.
- template<class vec_t >
  double vector_variance (size_t n, const vec_t &data)
  Compute the variance.
- template<class vec_t >
  double vector_stddev_fmean (size_t n, const vec_t &data, double mean)
  Standard deviation with specified mean known in advance.
- template<class vec_t >
  double vector_stddev (size_t n, const vec_t &data)
  Standard deviation with specified mean.
- template<class vec_t >
  double vector_stddev (size_t n, const vec_t &data, double mean)
  Standard deviation with specified mean.
- template<class vec_t >
  double vector_absdev (size_t n, const vec_t &data, double mean)
  Absolute deviation from the specified mean.
- template<class vec_t >
  double vector_absdev (size_t n, const vec_t &data)
  Absolute deviation from the computed mean.
- template<class vec_t >
  double vector_skew (size_t n, const vec_t &data, double mean, double stddev)

Skewness with specified mean and standard deviation.

- template<class vec_t >
  double vector_skew (size_t n, const vec_t &data)
  Skewness with computed mean and standard deviation.
- template<class vec_t >
  double vector_kurtosis (size_t n, const vec_t &data, double mean, double stddev)
  Kurtosis with specified mean and standard deviation.
- template<class vec_t >
  double vector_kurtosis (size_t n, const vec_t &data)
  Kurtosis with computed mean and standard deviation.
- template<class vec_t >
  double vector_lag1_autocorr (size_t n, const vec_t &data, double mean)
  Lag-1 autocorrelation.
- template<class vec_t >
  double vector_lag1_autocorr (size_t n, const vec_t &data)
  Lag-1 autocorrelation.
- template<class vec_t >
  double vector_covariance (size_t n, const vec_t &data1, const vec_t &data2, double mean1, double mean2)
  Compute the covariance of two vectors.
- template<class vec_t >
  double vector_covariance (size_t n, const vec_t &data1, const vec_t &data2)
  Compute the covariance of two vectors.
- template<class vec_t >
  double vector_correlation (size_t n, const vec_t &data1, const vec_t &data2)
  Pearson's correlation.
- template<class vec_t , class vec2_t >
  double vector_pvariance (size_t n1, const vec_t &data1, size_t n2, const vec2_t &data2)
  Pooled variance.
- template<class vec_t >
  double vector_quantile_sorted (size_t n, const vec_t &data, const double f)
  Quantile from sorted data (ascending only)
- template<class vec_t >
  double vector_median_sorted (size_t n, const vec_t &data)
  Return the median of sorted (ascending or descending) data.
- template<class vec_t , class vec2_t , class vec3_t >
  double vector_chi_squared (size_t n, const vec_t &obs, const vec2_t &exp, const vec3_t &err)
  Compute the chi-squared statistic.

Weighted vector functions

- template<class vec_t >
  double wvector_mean (size_t n, const vec_t &data, const vec_t &weights)
  Compute the mean of weighted data.
- template<class vec_t >
  double wvector_factor (size_t n, const vec_t &weights)
  Compute a normalization factor for weighted data.
- template<class vec_t , class vec2_t >
  double wvector_variance_fmean (size_t n, const vec_t &data, const vec2_t &weights, double wmean)
  Compute the variance of a weighted vector with a mean known in advance.
- template<class vec_t , class vec2_t >
  double wvector_variance (size_t n, const vec_t &data, const vec2_t &weights, double wmean)
  Compute the variance of a weighted vector with specified mean.
- template<class vec_t , class vec2_t >
  double wvector_variance (size_t n, const vec_t &data, const vec2_t &weights)
  Compute the variance of a weighted vector where mean is computed automatically.
- template<class vec_t , class vec2_t >
  double wvector_stddev_fmean (size_t n, const vec_t &data, const vec2_t &weights, double wmean)
  Compute the standard deviation of a weighted vector with a mean known in advance.
- template<class vec_t , class vec2_t >
  double wvector_stddev (size_t n, const vec_t &data, const vec2_t &weights)
  Compute the standard deviation of a weighted vector where mean is computed automatically.
- template<class vec_t , class vec2_t >
  double wvector_stddev (size_t n, const vec_t &data, const vec2_t &weights, double wmean)

Compute the standard deviation of a weighted vector with specified mean.

- template<class vec_t , class vec2_t >
  double wvector_sumsq (size_t n, const vec_t &data, const vec2_t &weights, double wmean)
  Compute the weighted sum of squares of data about the specified weighted mean.
- template<class vec_t , class vec2_t >
  double wvector_sumsq (size_t n, const vec_t &data, const vec2_t &weights)
  Compute the weighted sum of squares of data about the weighted mean.
- template<class vec_t , class vec2_t >
  double wvector_absdev (size_t n, const vec_t &data, const vec2_t &weights, double wmean)
  Compute the absolute deviation of data about a specified mean.
- template<class vec_t , class vec2_t >
  double wvector_absdev (size_t n, const vec_t &data, const vec2_t &weights)
  Compute the absolute deviation of data about a specified mean.
- template<class vec_t , class vec2_t >
  double wvector_skew (size_t n, const vec_t &data, const vec2_t &weights, double wmean, double wsd)
  Compute the skewness of data with specified mean and standard deviation.
- template<class vec_t , class vec2_t >
  double wvector_skew (size_t n, const vec_t &data, const vec2_t &weights)
  Compute the skewness of data with specified mean and standard deviation.
- template<class vec_t , class vec2_t >
  double wvector_kurtosis (size_t n, const vec_t &data, const vec2_t &weights, double wmean, double wsd)
  Compute the kurtosis of data with specified mean and standard deviation.
- template<class vec_t , class vec2_t >
  double wvector_kurtosis (size_t n, const vec_t &data, const vec2_t &weights)
  Compute the kurtosis of data with specified mean and standard deviation.

47.37.2 Function Documentation

47.37.2.1 template<class vec_t > double vector_mean (size_t *n*, const vec_t & *data*)

This function produces the same results as `gsl_stats_mean()`.

If `n` is zero, this will return zero.

Definition at line 55 of file vec_stats.h.

47.37.2.2 template<class vec_t > double vector_variance_fmean (size_t *n*, const vec_t & *data*, double *mean*)

This function computes

$$\frac{1}{N}\sum_i (x_i - \mu)^2$$

where the value of μ is given in `mean`.

This function produces the same results as `gsl_stats_variance_with_fixed_mean()`.

Definition at line 76 of file vec_stats.h.

47.37.2.3 template<class vec_t > double vector_variance (size_t *n*, const vec_t & *data*, double *mean*)

This function computes

$$\frac{1}{N-1}\sum_i (x_i - \mu)^2$$

where the value of μ is given in `mean`.

This function produces the same results as `gsl_stats_variance_m`.

If `n` is 0 or 1, this function will call the error handler.

Definition at line 100 of file vec_stats.h.

47.37.2.4 template<class vec_t> double vector_variance (size_t *n*, const vec_t & *data*)

This function computes

$$\frac{1}{N-1}\sum_i (x_i - \mu)\, n^2$$

where μ is the mean computed with vector_mean().

This function produces the same results as `gsl_stats_variance`.

If `n` is 0 or 1, this function will call the error handler.

Definition at line 124 of file vec_stats.h.

47.37.2.5 template<class vec_t> double vector_stddev_fmean (size_t *n*, const vec_t & *data*, double *mean*)

This function computes

$$\sqrt{\frac{1}{N}\sum_i (x_i - \mu)^2}$$

where the value of μ is given in `mean`.

This function produces the same results as `gsl_stats_sd_with_fixed_mean()`.

If `n` is zero, this function will return zero without calling the error handler.

Definition at line 151 of file vec_stats.h.

47.37.2.6 template<class vec_t> double vector_stddev (size_t *n*, const vec_t & *data*)

This function computes

$$\sqrt{\frac{1}{N-1}\sum_i (x_i - \mu)^2}$$

where μ is the mean computed with vector_mean().

This function produces the same results as `gsl_stats_sd()`.

If `n` is 0 or 1, this function will call the error handler.

Definition at line 170 of file vec_stats.h.

47.37.2.7 template<class vec_t> double vector_stddev (size_t *n*, const vec_t & *data*, double *mean*)

This function computes

$$\sqrt{\frac{1}{N-1}\sum_i (x_i - \mu)^2}$$

where the value of μ is given in `mean`.

This function produces the same results as `gsl_stats_sd_m()`.

If `n` is 0 or 1, this function will call the error handler.

Definition at line 196 of file vec_stats.h.

47.37.2.8 template<class vec_t> double vector_absdev (size_t *n*, const vec_t & *data*, double *mean*)

This function computes

$$\sum_i |x_i - \mu|$$

where the value of μ is given in `mean`.

This function produces the same results as `gsl_stats_absdev_m()`.

If n is zero, this function will return zero without calling the error handler.

Definition at line 222 of file vec_stats.h.

47.37.2.9 template<class vec_t > double vector_absdev (size_t *n*, const vec_t & *data*)

This function computes

$$\sum_i |x_i - \mu|$$

where the value of μ is mean as computed from vector_mean().

This function produces the same results as gsl_stats_absdev().

If n is zero, this function will return zero without calling the error handler.

Definition at line 250 of file vec_stats.h.

47.37.2.10 template<class vec_t > double vector_skew (size_t *n*, const vec_t & *data*, double *mean*, double *stddev*)

This function computes

$$\frac{1}{N}\sum_i \left[\frac{(x_i - \mu)}{\sigma}\right]^3$$

where the values of μ and σ are given in mean and stddev.

This function produces the same results as gsl_stats_skew_m_sd().

If n is zero, this function will return zero without calling the error handler.

Definition at line 271 of file vec_stats.h.

47.37.2.11 template<class vec_t > double vector_skew (size_t *n*, const vec_t & *data*)

This function computes

$$\frac{1}{N}\sum_i \left[\frac{(x_i - \mu)}{\sigma}\right]^3$$

where the values of μ and σ are computed using vector_mean() and vector_stddev().

This function produces the same results as gsl_stats_skew().

If n is zero, this function will return zero without calling the error handler.

Definition at line 297 of file vec_stats.h.

47.37.2.12 template<class vec_t > double vector_kurtosis (size_t *n*, const vec_t & *data*, double *mean*, double *stddev*)

This function computes

$$-3 + \frac{1}{N}\sum_i \left[\frac{(x_i - \mu)}{\sigma}\right]^4$$

where the values of μ and σ are given in mean and stddev.

This function produces the same results as gsl_stats_kurtosis_m_sd().

If n is zero, this function will return zero without calling the error handler.

Definition at line 320 of file vec_stats.h.

47.37.2.13 template<class vec_t > double vector_kurtosis (size_t *n*, const vec_t & *data*)

This function computes

$$-3 + \frac{1}{N}\sum_i \left[\frac{(x_i - \mu)}{\sigma}\right]^4$$

where the values of μ and σ are computed using vector_mean() and vector_stddev().

This function produces the same results as `gsl_stats_kurtosis()`.

If `n` is zero, this function will return zero without calling the error handler.

Definition at line 346 of file vec_stats.h.

47.37.2.14 template<class vec_t> double vector_lag1_autocorr (size_t *n*, const vec_t & *data*, double *mean*)

This function computes

$$\left[\sum_i (x_i - \mu)(x_{i-1} - \mu)\right]\left[\sum_i (x_i - \mu)^2\right]^{-1}$$

This function produces the same results as `gsl_stats_lag1_autocorrelation_m()`.

If `n` is zero, this function will call the error handler.

Definition at line 369 of file vec_stats.h.

47.37.2.15 template<class vec_t> double vector_lag1_autocorr (size_t *n*, const vec_t & *data*)

This function computes

$$\left[\sum_i (x_i - \mu)(x_{i-1} - \mu)\right]\left[\sum_i (x_i - \mu)^2\right]^{-1}$$

This function produces the same results as `gsl_stats_lag1_autocorrelation()`.

If `n` is zero, this function will call the error handler.

Definition at line 404 of file vec_stats.h.

47.37.2.16 template<class vec_t> double vector_covariance (size_t *n*, const vec_t & *data1*, const vec_t & *data2*, double *mean1*, double *mean2*)

This function computes

$$\frac{1}{n-1}\sum_i (x_i - \bar{x})(y_i - \bar{y})$$

where $\bar{x}$ and $\bar{y}$ are specified in `mean1` and `mean2`, respectively.

This function produces the same results as `gsl_stats_covariance_m()`.

If `n` is zero, this function will return zero without calling the error handler.

Definition at line 426 of file vec_stats.h.

47.37.2.17 template<class vec_t> double vector_covariance (size_t *n*, const vec_t & *data1*, const vec_t & *data2*)

This function computes

$$\frac{1}{n-1}\sum_i (x_i - \bar{x})(y_i - \bar{y})$$

where $\bar{x}$ and $\bar{y}$ are the averages of `data1` and `data2` and are computed automatically using vector_mean().

This function produces the same results as `gsl_stats_covariance()`.

If `n` is zero, this function will return zero without calling the error handler.

Definition at line 455 of file vec_stats.h.

47.37.2.18 template<class vec_t> double vector_correlation (size_t *n*, const vec_t & *data1*, const vec_t & *data2*)

This function computes the Pearson correlation coefficient between `data1` and `data2` .

This function produces the same results as `gsl_stats_correlation()`.

If n is zero, this function will call the error handler.

Definition at line 488 of file vec_stats.h.

47.37.2.19 template<class vec_t , class vec2_t > double vector_pvariance (size_t *n1*, const vec_t & *data1*, size_t *n2*, const vec2_t & *data2*)

Todo Document this

This function produces the same results as gsl_stats_pvariance().

If n is zero, this function will return zero without calling the error handler.

Definition at line 543 of file vec_stats.h.

47.37.2.20 template<class vec_t > double vector_quantile_sorted (size_t *n*, const vec_t & *data*, const double *f*)

This function returns the quantile f of data which has already been sorted in ascending order. The quantile, q , is found by interpolation using

$$q = (1-\delta)x_i \delta x_{i+1}$$

where $i = \text{floor}[(n-1)f]$ and $\delta = (n-1)f - i$.

This function produces the same results as gsl_stats_quantile_from_sorted_data().

No checks are made to ensure the data is sorted, or to ensure that $0 \leq 0 \leq 1$. If n is zero, this function will return zero without calling the error handler.

Definition at line 570 of file vec_stats.h.

47.37.2.21 template<class vec_t > double vector_median_sorted (size_t *n*, const vec_t & *data*)

This function returns the median of sorted data (either ascending or descending), assuming the data has already been sorted. When the data set has an odd number of elements, the median is the value of the element at index $(n-1)/2$, otherwise, the median is taken to be the average of the elements at indices $(n-1)/2$ and $n/2$.

This function produces the same results as gsl_stats_median_from_sorted_data().

No checks are made to ensure the data is sorted. If n is zero, this function will return zero without calling the error handler.

Definition at line 598 of file vec_stats.h.

47.37.2.22 template<class vec_t , class vec2_t , class vec3_t > double vector_chi_squared (size_t *n*, const vec_t & *obs*, const vec2_t & *exp*, const vec3_t & *err*)

This function computes

$$\sum_i \left(\frac{\text{obs}_i - \text{exp}_i}{\text{err}_i} \right)^2$$

where obs are the observed values, exp are the expected values, and err are the errors.

Definition at line 622 of file vec_stats.h.

47.37.2.23 template<class vec_t > double wvector_mean (size_t *n*, const vec_t & *data*, const vec_t & *weights*)

This function computes

$$\left(\sum_i w_i x_i \right) \left(\sum_i w_i \right)^{-1}$$

This function produces the same results as gsl_stats_wmean().

Definition at line 650 of file vec_stats.h.

47.37.2.24 template<class vec_t > double wvector_factor (size_t *n*, const vec_t & *weights*)

This function is used internally in wvector_variance(size_t n, vec_t &data, const vec2_t &weights, double wmean) and wvector_-stddev(size_t n, vec_t &data, const vec2_t &weights, double wmean) .

Definition at line 672 of file vec_stats.h.

47.37.2.25 template<class vec_t , class vec2_t > double wvector_variance_fmean (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*)

This function computes

$$\left[\sum_i w_i (x_i - \mu)^2\right] \left[\sum_i w_i\right]^{-1}$$

This function produces the same results as `gsl_stats_wvariance_with_fixed_mean()`.

Definition at line 702 of file vec_stats.h.

47.37.2.26 template<class vec_t , class vec2_t > double wvector_variance (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*)

This function produces the same results as `gsl_stats_wvariance_m()`.

Definition at line 725 of file vec_stats.h.

47.37.2.27 template<class vec_t , class vec2_t > double wvector_variance (size_t *n*, const vec_t & *data*, const vec2_t & *weights*)

This function produces the same results as `gsl_stats_wvariance()`.

Definition at line 742 of file vec_stats.h.

47.37.2.28 template<class vec_t , class vec2_t > double wvector_stddev_fmean (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*)

This function produces the same results as `gsl_stats_wsd_with_fixed_mean()`.

Definition at line 756 of file vec_stats.h.

47.37.2.29 template<class vec_t , class vec2_t > double wvector_stddev (size_t *n*, const vec_t & *data*, const vec2_t & *weights*)

This function produces the same results as `gsl_stats_wsd()`.

Definition at line 768 of file vec_stats.h.

47.37.2.30 template<class vec_t , class vec2_t > double wvector_stddev (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*)

This function produces the same results as `gsl_stats_wsd_m()`.

Definition at line 781 of file vec_stats.h.

47.37.2.31 template<class vec_t , class vec2_t > double wvector_sumsq (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*)

This function produces the same results as `gsl_stats_wtss_m()`.

Definition at line 797 of file vec_stats.h.

47.37.2.32 template<class vec_t , class vec2_t > double wvector_sumsq (size_t *n*, const vec_t & *data*, const vec2_t & *weights*)

This function produces the same results as `gsl_stats_wtss()`.

Definition at line 818 of file vec_stats.h.

47.37.2.33 template<class vec_t , class vec2_t > double wvector_absdev (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*)

This function produces the same results as `gsl_stats_wabsdev_m()`.

Definition at line 831 of file vec_stats.h.

47.37.2.34 template<class vec_t , class vec2_t > double wvector_absdev (size_t *n*, const vec_t & *data*, const vec2_t & *weights*)

This function produces the same results as `gsl_stats_wabsdev()`.

Definition at line 852 of file vec_stats.h.

47.37.2.35 template<class vec_t , class vec2_t > double wvector_skew (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*, double *wsd*)

This function produces the same results as `gsl_stats_wskew_m_sd()`.

Definition at line 866 of file vec_stats.h.

47.37.2.36 template<class vec_t , class vec2_t > double wvector_skew (size_t *n*, const vec_t & *data*, const vec2_t & *weights*)

This function produces the same results as `gsl_stats_wskew()`.

Definition at line 888 of file vec_stats.h.

47.37.2.37 template<class vec_t , class vec2_t > double wvector_kurtosis (size_t *n*, const vec_t & *data*, const vec2_t & *weights*, double *wmean*, double *wsd*)

This function produces the same results as `gsl_stats_wkurtosis_m_sd()`.

Definition at line 901 of file vec_stats.h.

47.37.2.38 template<class vec_t , class vec2_t > double wvector_kurtosis (size_t *n*, const vec_t & *data*, const vec2_t & *weights*)

This function produces the same results as `gsl_stats_wkurtosis()`.

Definition at line 923 of file vec_stats.h.

47.38 vector.h File Reference

File for generic vector functions.

```
#include <iostream> #include <cmath> #include <string> #include <fstream> #include <sstream> ×
#include <o2scl/err_hnd.h> #include <o2scl/uniform_grid.h> #include <gsl/gsl_ieee_utils.-
h> #include <gsl/gsl_sort.h>
```

47.38.1 Detailed Description

This file contains a set of template functions which can be applied to almost any vector or matrix type which allow element access through `operator[]`. Detailed requirements on the template parameters are given in the functions below.

For a general discussion of vectors and matrices in O_2scl , see the Arrays, Vectors, Matrices and Tensors of the User's Guide.

For overloaded arithmetic operators involving vectors and matrices, see vec_arith.h . For statistics operations not included here, see vec_stats.h in the directory `src/other` . For assorted functions and classes which are specific to C-style arrays, see array.h . Also related are the matrix output functions, matrix_out(), matrix_cx_out_paren(), and matrix_out_paren() which are defined in columnify.h because they utilize the class columnify to format the output.

For functions which search for a value in an ordered (either increasing or decreasing) vector, see the class search_vec .

Idea for Future Create a matrix transpose copy function?

Idea for Future Create matrix swap row and column functions

Definition in file vector.h.

Data Structures

- class matrix_row< mat_t >
  Create a vector-like class from a row of a matrix.

Functions

- template<class vec_t , class data_t >
  void vector_grid (uniform_grid< data_t > g, vec_t &v)
  Fill a vector with a specified grid.

Copying functions

- template<class vec_t , class vec2_t >
  void vector_copy (size_t N, vec_t &src, vec2_t &dest)
  Simple vector copy.
- template<class vec_t , class vec2_t , class base_t >
  void vector_swap (size_t N, vec_t &v1, vec2_t &v2)
  Generic vector swap.
- template<class mat_t , class mat2_t >
  void matrix_copy (size_t M, size_t N, mat_t &src, mat2_t &dest)
  Simple matrix copy.
- template<class vec_t , class vec2_t >
  void vector_cx_copy_gsl (size_t N, vec_t &src, vec2_t &dest)
  GSL complex vector copy.
- template<class mat_t , class mat2_t >
  void matrix_cx_copy_gsl (size_t M, size_t N, mat_t &src, mat2_t &dest)
  GSL complex matrix copy.

Sorting functions

- template<class vec_t , class data_t >
  void sort_downheap (vec_t &data, size_t n, size_t k)
  Provide a downheap() function for vector_sort()
- template<class vec_t , class data_t >
  int vector_sort (size_t n, vec_t &data)
  Sort a vector (in increasing order)
- template<class vec_t , class vec_size_t >
  int sort_index_downheap (size_t N, const vec_t &data, vec_size_t &order, size_t k)
  Provide a downheap() function for vector_sort_index()
- template<class vec_t , class vec_size_t >
  int vector_sort_index (size_t n, const vec_t &data, vec_size_t &order)
  Create a permutation which sorts a vector (in increasing order)
- template<class vec_t >
  int vector_sort_double (size_t n, vec_t &data)
  Sort a vector of doubles (in increasing order)

Smallest and largest functions

- template<class vec_t , class data_t >
  int vector_smallest (size_t n, vec_t &data, size_t k, vec_t &smallest)
  Find the k smallest entries of a vector.
- template<class vec_t , class data_t >
  int vector_largest (size_t n, vec_t &data, size_t k, vec_t &largest)
  Find the k largest entries of a vector.

Vector minimum and maximum functions

- template<class vec_t , class data_t >
  data_t vector_max_value (size_t n, const vec_t &data)
  Compute the maximum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  size_t vector_max_index (size_t n, const vec_t &data)
  Compute the index which holds the maximum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  void vector_max (size_t n, const vec_t &data, size_t &index, data_t &val)
  Compute the maximum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  data_t vector_min_value (size_t n, const vec_t &data)
  Compute the minimum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  size_t vector_min_index (size_t n, const vec_t &data)
  Compute the index which holds the minimum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  void vector_min (size_t n, const vec_t &data, size_t &index, data_t &val)
  Compute the minimum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  void vector_minmax_value (size_t n, vec_t &data, data_t &min, data_t &max)
  Compute the minimum and maximum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  void vector_minmax_index (size_t n, vec_t &data, size_t &ix_min, size_t &ix_max)
  Compute the minimum and maximum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  void vector_minmax (size_t n, vec_t &data, size_t &ix_min, data_t &min, size_t &ix_max, data_t &max)
  Compute the minimum and maximum of the first `n` elements of a vector.

Matrix minimum and maximum functions

- template<class mat_t , class data_t >
  data_t matrix_max (size_t n, const size_t m, const mat_t &data)
  Compute the maximum of a matrix.
- template<class mat_t , class data_t >
  data_t matrix_min (size_t n, const size_t m, const mat_t &data)
  Compute the minimum of a matrix.
- template<class mat_t , class data_t >
  int matrix_minmax (size_t n, const size_t m, const mat_t &data, data_t &min, data_t &max)
  Compute the minimum and maximum of a matrix.

Searching functions

- template<class vec_t >
  size_t vector_lookup (size_t n, const vec_t &x, double x0)
  Lookup element `x0` in vector `x` of length `n`.
- template<class vec_t , class data_t >
  size_t vector_bsearch_inc (const data_t x0, const vec_t &x, size_t lo, size_t hi)
  Binary search a part of an increasing vector for `x0`.
- template<class vec_t , class data_t >
  size_t vector_bsearch_dec (const data_t x0, const vec_t &x, size_t lo, size_t hi)
  Binary search a part of an decreasing vector for `x0`.
- template<class vec_t , class data_t >
  size_t vector_bsearch (const data_t x0, const vec_t &x, size_t lo, size_t hi)
  Binary search a part of a monotonic vector for `x0`.

Miscellaneous mathematical functions

- template<class vec_t , class data_t >
  data_t vector_sum (size_t n, vec_t &data)
  Compute the sum of the first `n` elements of a vector.
- template<class vec_t , class data_t >
  data_t vector_norm (size_t n, const vec_t &x)
  Compute the norm of a vector of floating-point (single or double precision) numbers.

Other functions

- template<class vec_t , class data_t >
  int vector_rotate (size_t n, vec_t &data, size_t k)
  "Rotate" a vector so that the kth element is now the beginning
- template<class vec_t , class data_t >
  int vector_reverse (size_t n, vec_t &data)
  Reverse a vector.
- template<class vec_t >
  int vector_out (std::ostream &os, size_t n, const vec_t &v, bool endline=false)
  Output a vector to a stream.

47.38.2 Function Documentation

47.38.2.1 template<class vec_t , class vec2_t > void vector_copy (size_t *N*, vec_t & *src*, vec2_t & *dest*)

Note

This ordering is reversed from the GSL function `gsl_vector_memcpy()`. This is to be used with

```
vector_copy(N,source,destination);
```

instead of

```
gsl_vector_memcpy(destination,source);
```

It is assumed that the memory allocation for `dest` has already been performed.

This function will work for any class `vec2_t` which has an operator[] which returns a reference to the corresponding element and class `vec_t` with an operator[] which returns either a reference or the value of the corresponding element.

Definition at line 90 of file vector.h.

47.38.2.2 template<class vec_t , class vec2_t , class base_t > void vector_swap (size_t *N*, vec_t & *v1*, vec2_t & *v2*)

This function swaps the elements of `v1` and `v2`, one element at a time.

Definition at line 110 of file vector.h.

47.38.2.3 template<class mat_t , class mat2_t > void matrix_copy (size_t *M*, size_t *N*, mat_t & *src*, mat2_t & *dest*)

Note

This ordering is reversed from the GSL function `gsl_matrix_memcpy()`. This is to be used with

```
matrix_copy(N,source,destination);
```

instead of

```
gsl_matrix_memcpy(destination,source);
```

It is assumed that the memory allocation for `dest` has already been performed.

This function will work for any class `vec2_t` which has an operator[][] which returns a reference to the corresponding element and class `vec_t` with an operator[][] which returns either a reference or the value of the corresponding element.

Definition at line 157 of file vector.h.

47.38.2.4 template<class vec_t , class vec2_t > void vector_cx_copy_gsl (size_t *N*, vec_t & *src*, vec2_t & *dest*)

Idea for Future At present this works only with complex types based directly on the GSL complex format. This could be improved.

Definition at line 171 of file vector.h.

47.38.2.5 template<class mat_t , class mat2_t > void matrix_cx_copy_gsl (size_t *M*, size_t *N*, mat_t & *src*, mat2_t & *dest*)

Idea for Future At present this works only with complex types based directly on the GSL complex format. This could be improved.

Definition at line 184 of file vector.h.

47.38.2.6 template<class vec_t , class data_t > int vector_sort (size_t *n*, vec_t & *data*)

This is a generic sorting template function using a heapsort algorithm. It will work for any types `data_t` and `vec_t` for which

- `data_t` has a non-const version of `operator=`
- `data_t` has a less than operator to compare elements
- `vec_t::operator[]` returns a non-const reference to an object of type `data_t`

In particular, it will work with ovector, uvector, ovector_int, uvector_int (and other related O_2scl vector classes), the STL template class `std::vector`, and arrays and pointers of numeric, character, and string objects.

For example,

```
std::string list[3]={"dog","cat","fox"};
vector_sort<std::string[3],std::string>(3,list);
```

Note

With this function template alone, the user cannot avoid explicitly specifying the template types for this function because there is no parameter of type `data_t`, and function templates cannot handle default template types. For this reason, the function template vector_sort_double() was also created which provides the convenience of not requiring the user to specify the vector template type.
This sorting routine is not stable, i.e. equal elements have arbtrary final ordering

This works similarly to the GSL function `gsl_sort_vector()`.

Definition at line 251 of file vector.h.

47.38.2.7 template<class vec_t , class vec_size_t > int vector_sort_index (size_t *n*, const vec_t & *data*, vec_size_t & *order*)

This function takes a vector `data` and arranges a list of indices in `order`, which give a sorted version of the vector. The value `order[i]` gives the index of entry in in `data` which corresponds to the `i`th value in the sorted vector. The vector `data` is unchanged by this function, and the initial values in `order` are ignored. Before calling this function, `order` must already be allocated as a vector of size `n`.

For example, after calling this function, a sorted version the vector can be output with

```
size_t n=5;
double data[5]={3.1,4.1,5.9,2.6,3.5};
permutation order(n);
vector_sort_index(n,data,order);
for(size_t i=0;i<n;i++) {
  cout << data[order[i]] << endl;
}
```

To create a permutation which stores as its `i`th element, the index of `data[i]` in the sorted vector, you can invert the permutation created by this function.

This is a generic sorting template function. It will work for any types `vec_t` and `vec_size_t` for which

- `vec_t` has an `operator[]`, and
- `vec_size_t` has an `operator[]` which returns a `size_t` . One possible type for `vec_size_t` is permutation.

This works similarly to the GSL function `gsl_sort_index()`.

Definition at line 344 of file vector.h.

47.38.2.8 template<class vec_t > int vector_sort_double (size_t *n*, vec_t & *data*)

This function is just a wrapper for

```
vector_sort<vec_t,double>(n,data);
```

See the documentation of vector_sort() for more details.

Definition at line 396 of file vector.h.

47.38.2.9 template<class vec_t , class data_t > int vector_smallest (size_t *n*, vec_t & *data*, size_t *k*, vec_t & *smallest*)

Given a vector `data` of size `n` this sets the first `k` entries of the vector `smallest` to the k smallest entries from vector `data` in ascending order. The vector `smallest` must be allocated beforehand to hold at least `k` elements.

This works similarly to the GSL function `gsl_sort_smallest()`.

Note

This $\mathcal{O}(kN)$ algorithm is useful only when $k << N$.

If `k` is zero, then this function does nothing and returns gsl_success .

Definition at line 419 of file vector.h.

47.38.2.10 template<class vec_t , class data_t > int vector_largest (size_t *n*, vec_t & *data*, size_t *k*, vec_t & *largest*)

Given a vector `data` of size `n` this sets the first `k` entries of the vector `largest` to the k largest entries from vector `data` in descending order. The vector `largest` must be allocated beforehand to hold at least `k` elements.

This works similarly to the GSL function `gsl_sort_largest()`.

Note

This $\mathcal{O}(kN)$ algorithm is useful only when $k << N$.

If `k` is zero, then this function does nothing and returns gsl_success .

Definition at line 466 of file vector.h.

47.38.2.11 template<class mat_t , class data_t > data_t matrix_max (size_t *n*, const size_t *m*, const mat_t & *data*)

Idea for Future Write matrix_max_index() and related functions similar to the way vector_max_index() works.

Definition at line 701 of file vector.h.

47.38.2.12 template<class vec_t > size_t vector_lookup (size_t *n*, const vec_t & *x*, double *x0*)

This function finds the element in vector `x` which is closest to `x0`. It ignores all elements in `x` which are not finite. If the vector is empty (i.e. `n` is zero), or if all of the elements in `x` are not finite, then the error handler will be called.

This function works for all classes `vec_t` where an operator[] is defined which returns a double (either as a value or a reference).

Idea for Future Write matrix_lookup().

Definition at line 820 of file vector.h.

47.38.2.13 template<class vec_t , class data_t > size_t vector_bsearch_inc (const data_t *x0*, const vec_t & *x*, size_t *lo*, size_t *hi*)

This function performs a binary search of between `x[lo]` and `x[hi]`. It returns

- lo if x0 < x[lo]
- i if x[i] <= x0 < x[i+2] for lo <= i < hi
- hi-1 if x0 >= x[hi-1]

The element at x[hi] is never referenced by this function. The parameter hi can be either the index of the last element (e.g. n-1 for a vector of size n with starting index 0), or the index of one element (e.g. n for a vector of size n and a starting index 0) for a depending on whether or not the user wants to allow the function to return the index of the last element.

This function operates in the same way as gsl_interp_bsearch().

The operation of this function is undefined if the data is not strictly monotonic, i.e. if some of the data elements are equal.

This function will call the error handler if lo is greater than hi.

Definition at line 874 of file vector.h.

47.38.2.14 template<class vec_t , class data_t > size_t vector_bsearch_dec (const data_t *x0*, const vec_t & *x*, size_t *lo*, size_t *hi*)

This function performs a binary search of between x[lo] and x[hi] (inclusive). It returns

- lo if x0 > x[lo]
- i if x[i] >= x0 > x[i+1] for lo <= i < hi
- hi-1 if x0 <= x[hi-1]

The element at x[hi] is never referenced by this function. The parameter hi can be either the index of the last element (e.g. n-1 for a vector of size n with starting index 0), or the index of one element (e.g. n for a vector of size n and a starting index 0) for a depending on whether or not the user wants to allow the function to return the index of the last element.

The operation of this function is undefined if the data is not strictly monotonic, i.e. if some of the data elements are equal.

This function will call the error handler if lo is greater than hi.

Definition at line 919 of file vector.h.

47.38.2.15 template<class vec_t , class data_t > size_t vector_bsearch (const data_t *x0*, const vec_t & *x*, size_t *lo*, size_t *hi*)

This wrapper just calls vector_bsearch_inc() or vector_bsearch_dec() depending on the ordering of x.

Definition at line 945 of file vector.h.

47.38.2.16 template<class vec_t , class data_t > data_t vector_sum (size_t *n*, vec_t & *data*)

If n is zero, this will set sum to zero and return gsl_success.

Definition at line 962 of file vector.h.

47.38.2.17 template<class vec_t , class data_t > data_t vector_norm (size_t *n*, const vec_t & *x*)

This function is a more generic version of o2scl_cblas::dnrm2 .

Definition at line 978 of file vector.h.

47.38.2.18 template<class vec_t , class data_t > int vector_rotate (size_t *n*, vec_t & *data*, size_t *k*)

This is a generic template function which will work for any types data_t and vec_t for which

- data_t has an operator=
- vec_t::operator[] returns a reference to an object of type data_t

This function is used, for example, in pinside.

Note

This function is not the same as a Givens rotation, which is typically referred to in BLAS routines as `drot()`.

Definition at line 1025 of file vector.h.

47.38.2.19 template<class vec_t , class data_t > int vector_reverse (size_t *n*, vec_t & *data*)

If `n` is zero, this function will silently do nothing.

Definition at line 1044 of file vector.h.

47.38.2.20 template<class vec_t > int vector_out (std::ostream & *os*, size_t *n*, const vec_t & *v*, bool *endline* = `false`)

No trailing space is output after the last element, and an endline is output only if `endline` is set to `true`. If the parameter `n` is zero, this function silently does nothing.

Note that the O_2scl vector classes also have their own `operator<<()` defined for them.

This works with any class `vec_t` which has an operator[] which returns either the value of or a reference to the ith element and the element type has its own output operator which has been defined.

Definition at line 1070 of file vector.h.

47.39 vector_derint.h File Reference

Derivatives of integrals of functions stored in vectors with implicit fixed-size grid.

`#include <o2scl/ovector_tlate.h> #include <o2scl/interp.h>`

47.39.1 Detailed Description

Integrate a function over a fixed-size grid specified in a vector. The integrals are always specified without the factor defining the grid size (i.e. Δx), so the user must always multiply the result by the grid size afterwards to get the true integral.

These integration rules often expect a minimum number of points, so for smaller vectors they fall back onto rules which use fewer points. For empty vectors they return zero, for vectors of length 1, they always return the sole element of the vector, and for vectors of length 2, they always return the average of the two elements.

More points does not always mean higher accuracy.

Definition in file vector_derint.h.

Functions

- template<class vec_t , class vec2_t >
  void vector_deriv_threept (size_t n, vec_t &v, vec2_t &v2)
  Derivative of a vector with a three-point formula.
- template<class vec_t , class vec2_t >
  void vector_deriv_threept_tap (size_t n, vec_t &v, vec2_t &v2)
  Derivative of a vector with a three-point formula using two-point at the edges.
- template<class vec_t , class vec2_t >
  void vector_deriv_fivept (size_t n, vec_t &v, vec2_t &v2)
  Derivative of a vector with a five-point formula.
- template<class vec_t , class vec2_t >
  void vector_deriv_fivept_tap (size_t n, vec_t &v, vec2_t &v2)
  Derivative of a vector with a five-point formula with four- and three-point formulas used at the edges.
- template<class ovec_t >
  void vector_deriv_interp (size_t n, ovec_t &v, ovec_t &v2, o2scl::base_interp_mgr< ovec_t > &bim)
  Derivative from interpolation object.

- template<class vec_t >
  double vector_integ_trap (size_t n, vec_t &v)
  Integrate with an extended trapezoidal rule.
- template<class vec_t >
  double vector_integ_threept (size_t n, vec_t &v)
  Integrate with an extended 3-point rule (extended Simpson's rule)
- template<class vec_t >
  double vector_integ_extended4 (size_t n, vec_t &v)
  Integrate with an extended rule for 4 or more points.
- template<class vec_t >
  double vector_integ_durand (size_t n, vec_t &v)
  Integrate with Durand's rule for 4 or more points.
- template<class vec_t >
  double vector_integ_extended8 (size_t n, vec_t &v)
  Integrate with an extended rule for 8 or more points.
- template<class ovec_t >
  double vector_integ_interp (size_t n, ovec_t &v, base_interp_mgr< ovec_t > &bim)
  Integral from interpolation object.

47.39.2 Function Documentation

47.39.2.1 template<class vec_t > double vector_integ_threept (size_t *n*, vec_t & *v*)

Note

This uses an untested hack I wrote for even n.

Definition at line 181 of file vector_derint.h.

47.39.2.2 template<class vec_t > double vector_integ_extended4 (size_t *n*, vec_t & *v*)

This function falls back to the equivalent of vector_integ_threept() for 3 points.

Definition at line 213 of file vector_derint.h.

47.39.2.3 template<class vec_t > double vector_integ_durand (size_t *n*, vec_t & *v*)

This function falls back to the equivalent of vector_integ_threept() for 3 points.

Definition at line 236 of file vector_derint.h.

47.39.2.4 template<class vec_t > double vector_integ_extended8 (size_t *n*, vec_t & *v*)

This function falls back to vector_integ_extended4() for less than 8 points.

Definition at line 259 of file vector_derint.h.