O_2scl - An Object-Oriented Scientific Computing Library

Version 0.805

Contents

1 O2scl User's Guide

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
- The CERNLIB-based classes are rewritten in C++ and are often faster than their GSL counterparts
- 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 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
- Two mini-libraries
  - Thermodynamics of ideal and nearly-ideal particles with quantum statistics
  - Equations of state for finite density relevant for neutron stars

See licensing information at License Information.

1.1 Quick Reference to User's Guide

- Installation
- General Usage
- Compiling examples
- Related projects
- Complex Numbers
- Arrays, Vectors, Matrices and Tensors

1.2 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 libraries. 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 automatically installed by `make install`.

After `make install`, you may test the library with `make o2scl-test`.

This library requires GSL and is designed to work with GSL versions 1.9 or 1.10. Some classes may work with older versions of GSL, but this cannot be guaranteed.

Range-checking for vectors and matrices is performed similar to the GSL approach, and is turned on by default. You can disable range-checking by defining -DO2SCL_NO_RANGE_CHECK

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

The separate libraries o2scl_eos and o2scl_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 o2scl_eos depends on o2scl_part so using `-disable-partlib` without `-disable-eoslib` may not work.

There are several warning flags that are useful when configuring and compiling with `g++`. (See the GSL documentation for an excellent discussion.) For running `configure`, I use something like

```
CFLAGS="" CXXFLAGS="" CPPFLAGS="-ansi -pedantic -Wall -W          \
-Wconversion -Wno-unused -Wshadow -Wpointer-arith -Wcast-align     \
-Wwrite-strings -fshort-enums -ggdb -O3 -DGSL_RANGE_CHECK=1        \
-DHAVE_INLINE -DO2SCL_ARRAY_ABORT                                  \
-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.

with the references to the directory `/home/asteiner/install` in order to install somewhere in my user directory, and because my copy of GSL is installed there, rather than in `/usr/local`.

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

Un-installation: While there is no explicit "uninstall" procedure, 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`.

1.3 General Usage

Namespaces

Most of the classes reside in the namespace `o2scl` (removed from the documentation). 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). The CBLAS functions are in the o2scl_cblas namespace and vector and complex number arithmetic is in o2scl_arith. There are also two namespaces which hold integration coefficients, o2scl_inte_qag_coeffs and o2scl_inte_qng_coeffs.

Documentation conventions

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

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.

Error handling

Error handling is GSL-like with functions returning 0 for success and calling a GSL-like error handler. The error handler, err_hnd is a global pointer to an object of type err_class.

The default behaviour for all errors is simply store the error code and information. You can require the default error handler to print out any error (or even exit) using err_class::set_mode(). Also, if O2SCL_ARRAY_ABORT is defined, then `exit()` will be called any time a gsl_index error is called, e.g. an out-of-bounds array access.

Errors can be set through the macros set_err, set_err_ret, add_err, and add_err_ret, which are defined in the file err_hnd.h.

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

Note that the library functions do not typically reset the error handler. For this reason the user may need to reset the error handler before calling functions which do not return an integer error value so that they can easily test to see if an error occured, e.g. using err_hnd::get_errno().

The default error handler can be replaced by simply assigning the address of a descendant of err_class to err_hnd.

Library dependencies

All of the libraries need `libo2scl_base`, `libgsl`, and either `libgslcblas` or some externally-specified `libcblas` to operate. Other additional dependencies are listed below:

- `libo2scl_eos` needs `libo2scl_nuclei`, `libo2scl_part`, `libo2scl_other`, `libo2scl_minimize`, `libo2scl_inte`, and `libo2scl_root`
- `libo2scl_nuclei` needs `libo2scl_part`, `libo2scl_other`, `libo2scl_inte`, and `libo2scl_root`.
- `libo2scl_part` needs `libo2scl_other`, `libo2scl_inte`, and `libo2scl_root`.

1.4 Compiling examples

A few 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 and summarize these tests in `examples-summary.txt`. 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`.

See Example source code for the documented source code of the individual examples

Also, the testing code for each class is sometimes 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.

1.5 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 . 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 .

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

- 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

  This is the library on which the GSL integration routines are based (prefix `gsl_inte_`). It is available at http://www.netlib.org/quadpack .

- 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.

- 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".

- 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.

- 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).

- 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.

- 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.

1.6 Complex Numbers

Some rudimentary arithmetic operators 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,

```
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 provided in ovector_cx_tlate.h.

1.7 Arrays, Vectors, Matrices and Tensors

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).

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 `(,)`.

O_2scl 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.

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.

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.

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.

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 an argument of `0.0` 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. You can either use the error handler or test to see if the allocation succeeded by examining the size argument, e.g.

```
const size_t n=10;
ovector x(10);
if (x.size()==0) cout << "Failed." << endl;
```

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().

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.

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 (assuming the memory allocation has already been taken care of).

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 += matrix
- matrix -= matrix
- matrix += scalar
- matrix -= scalar
- matrix *= scalar

Binary operators like addition, subtraction, and matrix multiplication are also defined for ovector, uvector, and related objects in the o2scl_arith namespace. 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. The macros are detailed in 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.

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;
```

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.

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.

Views

Views are slightly different than in GSL in that they are now implemented as parent classes. 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[0] << " " << a[1] << endl;
```

Passing ovector parameters

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

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

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

```
void function(ovector_view &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) {
      set_err("Unallocated vector not sent.",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) {
    a.allocate(1);
    // do something with a
    return 0;
  }
};
```

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.

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 unused. 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.

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.

Const-correctness with vectors

There are several classes named with "`_const`" to provide different kinds of const views of const vectors. The keyword `const` still ought to be included to ensure that the object is treated properly. For example,

```
ovector o(2);
o[0]=3.0;
o[1]=-1.0;
const ovector_const_subvector ocs(o,1,1);
```

At present, const-correctness in O_2scl can be improperly removed, if the `const` keyword is not properly included. For example, the following code will compile, violated the const-correctness of the `ocs` variable.

```
ovector o(2);
o[0]=3.0;
o[1]=-1.0;
ovector_const_subvector ocs(o,1,1);
ovector_view ov(ocs);
ov[0]=2.0;
```

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 n-dimensional interpolation of the data defined on the specified grid.

1.8 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 is deallocated in the object destructor.

1.9 Linear algebra

There is a small set of linear algebra routines. These are not intended to be a replacement for LAPACK, but are designed for use by O_2scl routines so that they work for generic matrix and vector 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)
- 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)

1.10 Interpolation

The classes o2scl_interp and o2scl_interp_vec allow basic interpolation, lookup, differentiation, and integration of data given in two ovectors or ovector views. In contrast to the GSL routines, data which is presented with a decreasing independent variable is handled automatically. For interpolation with arrays rather than ovectors, use array_interp or array_interp_vec.

For fast interpolation of arrays where the independent variable is strictly increasing and without error-checking, you can directly use the children of base_interp.

The two interpolation interfaces

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_o2sclf=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_o2sclf=oi.interp(0.5);
```

Lookup and binary search

The class search_vec contains a searching functions for objects of type ovector which are monotonic. Note that if you want to find the index of an ovector where a particular value is located without any assumptions with regard to the ordering, you can use ovector::lookup() which performs an exhaustive search.

"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.

Two and higher dimensional interpolation

Preliminary support for two-dimensional interpolation is given in twod_intp, and n-dimensional interpolation in tensor_grid.

1.11 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.

1.12 Function Objects

Functions are passed to numerical routines using template-based function classes. 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
- grad_funct : Gradient function for minimizers
- ool_hfunct : Hessian product for constrained minimization

For each of these classes (except funct), there is a version named `_vfunct` instead of `_funct` which is designed to be used with C-style arrays instead.

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
- _strings: functions specified using strings, e.g. "x^2-2"
- _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 specified by a `void *` and directly returns the function value.

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 specifying a new type for the template parameter in the user class.

Note that 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.

1.13 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.

1.14 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(), double_to_latex(), double_to_html, and double_to_ieee_string().

There is a class called columnify, which 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.

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.

1.15 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.

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 also incorrect for second and third derivatives.

1.16 Integration

Integration is performed by descendants of inte and is provided in the library `o2scl_inte`.

There are several routines for one-dimensional integration.

- General integration over a finite interval: cern_adapt, cern_gauss, cern_gauss56, gsl_inte_qag, and gsl_inte_qng.
- General integration from 0 to ∞ : gsl_inte_qagiu

- General integration from $-\infty$ to 0: gsl_inte_qagil
- General integration from $-\infty$ to ∞ : gsl_inte_qagi
- Integration for a finite interval over an function with equally spaced abscissas provided in an array: eqi_inte
- General integration over a finite interval for a function with singularities: gsl_inte_qags and gsl_inte_qagp
- 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.

For the GSL-based integration routines, the variables inte::tolx and inte::tolf 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

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

and returns an error to attempt to ensure that

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

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` .

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

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

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).

1.17 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 (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 do not separately handle cases where 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.

At present, the polynomial routines work with complex numbers as objects of type `std::complex<double>` and are located in library `o2scl_other`.

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.

1.18 Equation Solving

Equation solving classes are stored in the library `o2scl_root`.

One-dimensional solvers

Solution of one equation in one variable is accomplished by children of the class root. This base class provides the structure for three different solving methods:

- root::solve(double &x, void *pa, funct &func) which solves a function given an initial guess `x`
- root::solve_bkt(double &x1, double x2, void *pa, funct &func) which solves a function given a solution bracketed between `x1` and `x2`. The values of the function at `x1` and `x2` must have different signs.
- root::solve_de(double &x, void *pa, funct &func, funct &df) which solves a function given an initial guess `x` and the derivative of the function `df`.

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.

If not all of these three functions are overloaded, then the source code in root 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. Also, gsl_root_stef is a "root-polishing" routine given derivatives of the form root::solve_de(). If either root::solve() or root::solve_bkt() are called, then root::solve_de() will be called with finite-differencing used to estimate the derivative. Of course, it is frequently most efficient to use the solver in the way it was intended.

Todo

Double check this documentation above

Multi-dimensional solvers

Solution of more than one equation is accomplished by descendants of the class mroot. There are two basic functions

- mroot::msolve(size_t n, ovector &x, void *pa, mm_funct &func) which solves the `n` equations given in `func` with an initial guess `x`.

For multi-dimensional solving, you can use either cern_mroot or gsl_mroot_hybrids. While cern_mroot does not use user-supplied derivatives, gsl_mroot_hybrids can use user-supplied derivative information (as in the GSL hybridsj method).

1.19 Minimization

One-dimensional minimization is performed by descendants of minimize and provided in the library `o2scl_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_simp, gsl_mmin_conp, gsl_mmin_conf, and gsl_mmin_bfgs2. The class multi_min_fix is a convenient way to perform a minimization while fixing some of the original parameters. The class gsl_mmin_simp does not require or use any derivative information, but the other minimization classes are intended for use when derivatives are available.

Simulated annealing methods are also provided (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 found a true minimum.

A naive 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 below.

1.20 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 "GENCAN" method 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 or ool_hvfunct 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 OOL classes.

1.21 Monte Carlo Integration

Monte Carlo integration is performed by descendants of mcarlo_inte in the library `o2scl_mcarlo` (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.

1.22 Simulated Annealing

Minimization by simulated annealing is performed by descendants of sim_anneal (see gsl_anneal). The annealing schedule is given by a descendant of tptr_schedule (see tptr_geoseries).

1.23 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.

1.24 Solution of Ordinary Differential Equations

Classes for non-adaptive integration are provided as descendents of odestep 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 or a child of ode_vfunct when using arrays.

Solution of simple initial value problems is performed by ode_iv_solve.

Preliminary support for boundary value problems is given in children of ode_bv_solve.

1.25 Random Number Generation

Random number generators are descendants of rnga and are provided in the library `o2scl_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. The GSL random number generator code is reimplemented in the class gsl_rnga to avoid an additional performance penalty. This may not be a truly "object-oriented" interface in that it does not use virtual functions, but it avoids any possible performance penalty. Random number generators are implemented as templates in sim_anneal and mcarlo_inte. In these classes, the random number generator is a template type, rather than a member data pointer, in order to ensure fast execution.

1.26 Two-dimensional Interpolation

Successive use of smart_interp is implemented in twod_intp. Also, see planar_intp and quad_intp and the computation of contour lines in contour. These latter three classes are somewhat experimental at present.

1.27 Other Routines

(These are all experimental)

Fourier transforms - see gsl_fft

Series acceleration - see gsl_series

Chebyshev approximations - see gsl_chebapp

Timing execution - see timer_gettod and timer_clock

Polylogarithms - see polylog

1.28 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.

1.29 Object I/O

The I/O portion of the library is still experimental.

Collections of objects can be stored in a collection class, and these collections can be written to or read from text or binary files. User-defined classes may be added to the collections and may be read and written to files as long as a descendant of io_base is provided.

Every type has an associated I/O type which is a descendant of io_base. In order to perform any sort of input/output on any type, an object of the corresponding I/O type must be instantiated by the user. This is not done automatically by the library. (Since it doesn't know which objects are going to be used ahead of time, the library would have to instantiate *all* of the I/O objects, which is needlessly slow.) This makes the I/O slightly less user-friendly, but much more efficient. For convenience, each subsection of the library has a class (named with an `_ioc` suffix) which will automatically allocate all I/O types for that subsection.

Level 1 functions: Functions that input/output data from library-defined objects and internal types from files and combine these objects in collections. These are primarily member functions of the class collection.

Level 2 functions: Functions which are designed to allow the user to input or output data for user-generated objects. These are primarily member functions of classes cinput and coutput.

Level 3 functions: Functions which allow low-level modifications on how input and output is performed. Usage of level 3 functions is not immediately recommended for the casual user.

Level 1 usage:

For adding an object to a collection when you have a pointer to the I/O object for the associated type:

```
int collection::add(std::string name, io_base *tio, void *vec,
int sz=0, int sz2=0, bool overwrt=true, bool owner=false);
```

For adding an object to a collection otherwise:

```
int collection::add(std::string name, std::string stype,
void *vec, int sz=0, int sz2=0,
bool overwrt=true, bool owner=false);
```

To retrieve an object as a

```
void *
```

from a collection use one of:

```
int get(std::string tname, void *&vec);
int get(std::string tname, void *&vec, int &sz);
int get(std::string tname, void *&vec, int &sz, int &sz2);
int get(std::string tname, std::string &stype, void *&vec);
int get(std::string tname, std::string &stype, void *&vec, int &sz);
int get(std::string tname, std::string &stype, void *&vec, int &sz,
int &sz2);
```

When retrieving a scalar object without error- and type-checking you can use the shorthand version:

```
void *get(std::string name);
```

To output one object to a file:

```
int collection::out_one(out_file_format *outs, std::string stype,
std::string name, void *vp, int sz=0, int sz2=0);
```

To input one object from a file with a given type and name:

```
int collection::in_one_name(in_file_format *ins, std::string stype,
std::string name, void *&vp, int &sz, int &sz2);
```

To input the first object of a given type from a file:

```
int collection::in_one(in_file_format *ins, std::string stype,
std::string &name, void *&vp, int &sz, int &sz2);
```

Level 2 usage (string-based):

If you don't have a pointer to the io_base child object corresponding to the type of subobject that you are manipulating, then you can use the following functions, which take the type name as a string.

To input a sub-object in an io_base template for which memory has already been allocated use one of:

```
int collection::object_in(std::string type, in_file_format *ins, void *vp,
                          std::string &name);
int collection::object_in(std::string type, in_file_format *ins, void *vp,
                          int sz, std::string &name);
int collection::object_in(std::string type, in_file_format *ins, void *vp,
                          int sz, int sz2, std::string &name);
```

To automatically allocate memory and input a sub-object of a io_base template use one of:

```
int collection::object_in_mem(std::string type, in_file_format *ins,
                              void *vp, std::string &name);
int collection::object_in_mem(std::string type, in_file_format *ins,
                              void *vp, int sz, std::string &name);
int collection::object_in_mem(std::string type, in_file_format *ins,
                              void *vp, int sz, int sz2, std::string &name);
```

To output a subobject in an io_base template use:

```
int collection::object_out(std::string type, out_file_format *outs,
void *op, int sz=0, int sz2=0, std::string name="");
```

Level 2 usage (with io_base pointer):

To input a sub-object in an io_base template for which memory has already been allocated use one of:

```
virtual int object_in(cinput *cin, in_file_format *ins, object *op,
                      std::string &name);
virtual int object_in(cinput *cin, in_file_format *ins, object *op,
                      int sz, std::string &name);
virtual int object_in(cinput *cin, in_file_format *ins, object **op,
                      int sz, int sz2, std::string &name);
template<size_t N>
int object_in(cinput *co, in_file_format *ins,
              object op[][N], int sz, std::string &name);
```

To automatically allocate memory and input a sub-object of a io_base template use one of:

```
virtual int object_in_mem(cinput *cin, in_file_format *ins,
                          object *&op, std::string &name);
virtual int object_in_mem(cinput *cin, in_file_format *ins, object *&op,
                          int &sz, std::string &name);
virtual int object_in_mem(cinput *cin, in_file_format *ins,
                          object **&op, int &sz, int &sz2,
                          std::string &name);
template<size_t N>
int object_in_mem(cinput *co, in_file_format *ins,
                  object op[][N], int &sz, std::string &name);
```

To output a subobject in an io_base template use:

```
virtual int object_out(coutput *cout, out_file_format *outs, object *op,
                       int sz=0, std::string name="");
virtual int object_out(coutput *cout, out_file_format *outs, object **op,
                       int sz, int sz2, std::string name="");
template<size_t N>
int object_out(coutput *cout, out_file_format *outs,
               object op[][N], int sz, std::string name="");
```

To automatically allocate/deallocate memory for an object, use:

```
virtual int mem_alloc(object *&op);
virtual int mem_alloc_arr(object *&op, int sz);
virtual int mem_alloc_2darr(object **&op, int sz, int sz2);
virtual int mem_free(object *op);
virtual int mem_free_arr(object *op);
virtual int mem_free_2darr(object **op, int sz);
```

Usage of io_tlate

The functions io_tlate::input() and io_tlate::output() need to be implemented for every class has information for I/O. For subobjects of the class, cinput::object_in() and cinput::object_out() can be called to input or output the information associated with the subobject. For input, cinput::object_in_name(), cinput::object_in_mem(), and cinput::object_in_mem_name() allow the freedom to input an object with a name or with memory allocation. The function coutput::object_out_name() allows one to output an object with a name. If the class contains a pointer to the subobject, then io_base::pointer_in() or io_base::pointer_out() can be used.

1.30 Example source code

Example list

- Function and solver example shows how member functions and external parameters are supplied to O_2scl numerical routines. In this case, a member function representing an equation with one unknown is solved using a one-dimesional solver.
- Multidimensional solver example demonstrates the multidimensional function solver and several different methods of specifying the function to be solved.
- Multidimensional minimizer example
- Numerical differentiation example
- Numerical integration example
- Ordinary differential equations example
- Simulated annealing example demonstrates multidimensional minimization by simulated annealing
- Multidimensional integration example demonstrates several ways to compute a multidimensional integral including Monte Carlo and direct methods
- Two-dimensional interpolation example

1.30.1 Function and solver example

```
/* 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);
  }

};

// A simple function to make the plot
int plot(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<double,funct<double> > solver;

  my_class c;
  c.set_parameter();

  // This is the "magic" that allows specification of class member
  // functions as functions to solve. This object-oriented 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_noerr<my_class,double> function(&c,&my_class::function_to_solve);

  double x1=-1;
  double x2=2;
  double p=1.1;

  // The value verbose=1 prints out iteration information
  // and verbose=2 requires a keypress between iterations.
  // The parameter p=0.1 is used.
```

```
  solver.verbose=1;
  solver.solve_bkt(x1,x2,p,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)
  plot(x1);

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

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

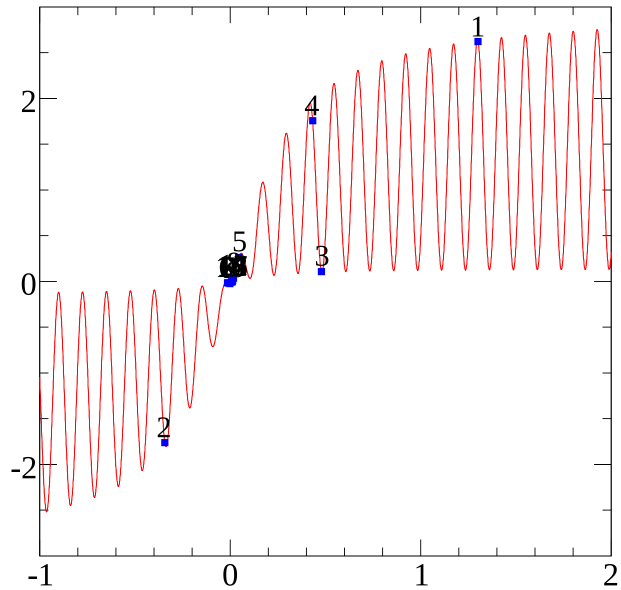


Figure 1: ex_fptr.eps

1.30.2 Multidimensional solver example

```
/* 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_view &x,
        ovector_view &y, void *&pa) {
```

```
  y[0]=sin(x[1]-0.2);
  y[1]=sin(x[0]-0.25);
  return 0;
}

class cl {

public:

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

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

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

  int mfnd(size_t nv, ovector_view &x, ovector_view &y,
           omatrix_view &j, void *&pa) {
    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 double x[2], double y[2], void *&pa) {
    y[0]=sin(x[1]-0.2);
    y[1]=sin(x[0]-0.25);
    return 0;
  }

  int mfnad(size_t nv, double x[], double y[], double j[2][2], void *&pa) {
    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;
  void *vp=NULL;
  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,void *,ovector_view> f1(&acl,&cl::mfn);
gsl_mroot_hybrids<void *> cr1;

x[0]=0.5;
x[1]=0.5;
acl.nf=0;
int ret1=cr1.msolve(2,x,vp,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<void *> cr2;

x[0]=0.5;
x[1]=0.5;
acl.nf=0;
int ret2=cr2.msolve(2,x,vp,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<void *> cr3;

x[0]=0.5;
x[1]=0.5;
cr3.allocate(2);
cr3.set(2,x,f1,vp);
done=false;
do {
  r3=cr3.iterate();
  double resid=fabs(cr3.f[0])+fabs(cr3.f[1]);
  if (resid<cr3.tolf || 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,void *,ovector_view,omatrix_view> 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,vp,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<void *> cr5;

x[0]=0.5;
x[1]=0.5;
cr5.allocate(2);
cr5.set_de(2,x,f1,j4,vp);
done=false;
do {
  r3=cr5.iterate();
  double resid=fabs(cr5.f[0])+fabs(cr5.f[1]);
  if (resid<cr5.tolf || 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_vfunct_mfptr<cl,void *,2> f6(&acl,&cl::mfna);
gsl_mroot_hybrids<void *,mm_vfunct_mfptr<cl,void *,2>,double[2],
  double[2],array_alloc<double[2]> > cr6;

xa[0]=0.5;
xa[1]=0.5;
cr6.msolve(2,xa,vp,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<void *,mm_vfunct_mfptr<cl,void *,2>,double[2],
  double[2],array_alloc<double[2]> > cr7;

xa[0]=0.5;
xa[1]=0.5;
cr7.msolve(2,xa,vp,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_vfunct_mfptr<cl,void *,2> j8(&acl,&cl::mfnad);
gsl_mroot_hybrids<void *,mm_vfunct_mfptr<cl,void *,2>,double[2],
  double[2],array_alloc<double[2]>,double[2][2],double[2][2],
  array_2d_alloc<double[2][2]>,jac_vfunct<void *,2> > cr8;

xa[0]=0.5;
xa[1]=0.5;
cr8.msolve_de(2,xa,vp,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<void *,cl,ovector_view> cr9;

x[0]=0.5;
x[1]=0.5;
cr9.msolve(2,x,vp,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_view &, ovector_view &,
                      void *&);
  gsl_mroot_hybrids<void *,gfnt,ovector_view> cr10;
  gfnt f10=&gfn;

  x[0]=0.5;
  x[1]=0.5;
  cr10.msolve(2,x,vp,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
```

1.30.3 Multidimensional minimizer example

```
/* Example: ex_mmin.cpp
   -------------------------------------------------------------------
   Example usage of the multidimensional minimizers
*/

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/multi_funct.h>
#include <o2scl/gsl_mmin_simp.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:

  int mfn(size_t nv, const ovector_view &x, double &y, void *&pa) {
    y=(x[0]-2.0)*(x[0]-2.0)+(x[1]-1.0)*(x[1]-1.0);
    return 0;
  }

};

int main(void) {
  cl acl;
  ovector x(2);
  void *vp=NULL;
  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,void *,ovector_view> f1(&acl,&cl::mfn);
  gsl_mmin_simp<void *> gm1;
  gsl_mmin_conf<void *> gm2;
  gsl_mmin_conp<void *> gm3;
  gsl_mmin_bfgs2<void *> gm4;
```

```
  x[0]=0.5;
  x[1]=0.5;
  gm1.mmin(2,x,fmin,vp,f1);
  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");

  x[0]=0.5;
  x[1]=0.5;
  gm2.mmin(2,x,fmin,vp,f1);
  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");

  x[0]=0.5;
  x[1]=0.5;
  gm3.mmin(2,x,fmin,vp,f1);
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],2.0,1.0e-4,"3a");
  t.test_rel(x[1],1.0,1.0e-4,"3b");

  x[0]=0.5;
  x[1]=0.5;
  gm4.mmin(2,x,fmin,vp,f1);
  cout << "Found minimum at: " << x << endl;
  t.test_rel(x[0],2.0,1.0e-4,"4a");
  t.test_rel(x[1],1.0,1.0e-4,"4b");

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

1.30.4 Numerical differentiation example

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

#include <cmath>
#include <o2scl/test_mgr.h>
#include <o2scl/funct.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) {
  cl acl;
  ovector x(2);
  double xa[2];
  int i;
  void *vp=0;
  size_t tmp;
  int r1, r2, r3;
```

```
  bool done;

  test_mgr t;
  t.set_output_level(2);

  funct_mfptr_nopar<cl,void *> f1(&acl,&cl::function);

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

  // Compute the first derivative using the gsl_deriv class and
  // verify that the answer is correct
  double d1=gd.calc(1.0,vp,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,vp,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,vp,f1);
  t.test_rel(d3,-4.0*sin(2.0),1.0e-8,"gsl_deriv");

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

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

1.30.5 Numerical integration example

```
/* 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;
  void *vp=0;
  test_mgr t;

  t.set_output_level(1);

  funct_mfptr_nopar<cl,void *> f1(&acl,&cl::integrand);
  funct_mfptr_nopar<cl,void *> 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<void *> g;
  gsl_inte_qagi<void *> gi;
  gsl_inte_qagiu<void *> gu;
  gsl_inte_qagil<void *> gl;
  cern_adapt<void *> 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,vp,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,vp,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,vp,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
  acl.nf=0;
  g.integ_err(f2,0.0,1.0,vp,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");

  // An integral from 0 to 1
  acl.nf=0;
  ca.integ_err(f2,0.0,1.0,vp,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
```

1.30.6 Ordinary differential equations example

```
/* 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>

using namespace std;
using namespace o2scl;

// 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_view &y,
           ovector_view &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 Ai(x) function.
int derivs2(double x, size_t nv, const ovector_view &y,
            ovector_view &dydx, double &alpha) {
  dydx[0]=y[1];
  dydx[1]=y[0]*x;
  return 0;
}

int main(void) {
  int i;
  double x, dx=1.0e-1;
  ovector y(2), dydx(2), yout(2), yerr(2), dydx_out(2);
  void *vp=NULL;
  test_mgr t;
  t.set_output_level(1);
```

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

ode_funct_fptr<double,ovector_view> od(derivs);
ode_funct_fptr<double,ovector_view> od2(derivs2);
gsl_rkck<double> ode;
gsl_rk8pd<double> ode2;
double alpha=1.0;

// Solve using the non-adaptive Cash-Karp stepper.

cout << "Bessel function, Cash-Karp: " << 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) {
  ode.step(x,dx,2,y,dydx,y,yerr,dydx,alpha,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-5,"rkck");
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

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

// 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,alpha,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");
  }
  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;

  // 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,alpha,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");
  }
  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;

  // Solve using the GSL adaptive stepper

  cout << "Adaptive stepper: " << endl;
  gsl_astep<double> 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,dx,10.0,2,y,dydx,yerr,alpha,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");
    k++;
  }
  cout << "Accuracy at end: "
       << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
  cout << endl;

  // 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,dx,10.0,2,y,dydx,yerr,alpha,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");
  k++;
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// 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,dx,10.0,2,y,dydx,yerr,alpha,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");
  k++;
}
cout << "Accuracy at end: "
     << fabs(y[0]-gsl_sf_bessel_J1(x))/gsl_sf_bessel_J1(x) << endl;
cout << endl;

// Solve using the O2scl initial value solver

cout << "Initial value solver: " << endl;
ode_iv_solve<double> ode4;
const int ngrid=101;
ovector xg(ngrid), yinit(2);
omatrix yg(ngrid,2);
for(i=0;i<ngrid;i++) xg[i]=((double)i)/10.0;
yinit[0]=0.0;
```

```
  yinit[1]=0.5;
  ode4.solve_grid(0.0,10.0,0.1,2,yinit,ngrid,xg,yg,alpha,od);

  cout << " x             J1(calc)      J1(exact)     rel. diff." << endl;
  for(i=1;i<ngrid;i+=10) {
    cout << xg[i] << " " << yg[i][0] << " "
         << gsl_sf_bessel_J1(xg[i]) << " ";
    cout << fabs((yg[i][0]-gsl_sf_bessel_J1(xg[i]))/
                 gsl_sf_bessel_J1(xg[i])) << 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");
  }
  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;

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

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

1.30.7 Simulated annealing example

```
/* 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
double funx(size_t nvar, const ovector_view &x, void *&vp) {
  double ret, a, b;
  a=(x[0]-2.0);
  return -gsl_sf_bessel_J0(a);
}

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

  cout.setf(ios::scientific);

  gsl_anneal<void *,multi_funct<void *> > ga;
  tptr_geoseries<ovector_view> tsch;
  double result;
  ovector init(1);

  multi_funct_fptr_noerr<void *> fx(funx);

  // Set the temperature schedule
  tsch.set_series(2.0,2.0e-6,1.5);
  ga.set_tptr_schedule(tsch);
  ga.ntrial=1000;
```

```
  void *vpx=0;

  // Choose an initial point at a local minimum away from
  // the global minimum
  init[0]=9.0;
  ga.mmin(1,init,result,vpx,fx);
  cout << "x: " << init[0]
       << ", minimum function value: " << result << endl;
  cout << endl;

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

  t.report();

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

1.30.8 Multidimensional integration example

```
/* 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_view &x, void *&vp) {
  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;

  void *vpx=0;
  gsl_vegas<void *,multi_funct<void *> > gm;
  ovector a(3), b(3);
  a.set_all(0.0);
  b.set_all(M_PI);

  multi_funct_fptr_noerr<void *> tf(test_fun);

  gm.n_points=100000;
  gm.minteg_err(tf,3,a,b,vpx,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
```

1.30.9 Two-dimensional interpolation example

```
/* 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;

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);
  x[0]=0.0;
  x[1]=1.0;
  x[2]=2.0;
  y[0]=3.0;
  y[1]=2.0;
  y[2]=1.0;
  cout << 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
```

1.31 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 functions
- Const-correctness and type-safety should be used wherever possible, and
- The design should be somewhat compatible with GSL. Also, the library provides higher-level routines for situations which do not require lower-level access.

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)
sring_conv.h : (none)
lib_settings.h : (none)
array.h: err_hnd.h
uvector_tlate.h: err_hnd.h
ovector_tlate.h: uvector_tlate.h array.h err_hnd.h
misc.h : ovector_tlate.h uvector_tlate.h array.h lib_settings.h err_hnd.h
test_mgr.h : misc.h ovector_tlate.h uvector_tlate.h
.              array.h lib_settings.h err_hnd.h
```

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.

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 ; } whil

    }
  {
    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.cpp`.

Global objects

There are three global objects that are created in libo2scl_base:

- def_err_hnd is the default 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.

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 access the same instance of a class. In this respect, O_2scl is no different from GSL.

Documentation design

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

1.32 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`.

```
  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),
```

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.
```

1.33 Acknowledgements

I would like to thank the creators of GSL for their excellent work!

1.34 Bibliography

Some of the references which contain links should direct you to the work referred to directly through dx.doi.org.

Bus75: J.C.P. Bus and T.J. Dekker, ACM Trans. Math. Software 1 (1975) 330-345.

Fletcher87: R. Fletcher, Practical methods of optimization (John Wiley & Sons, Chichester 1987) 39.

Krabs83: W. Krabs, Einfuhrung in die lieare und nictlineare Optimierung fur Ingenieure (BSB B.G. Teubner, Leipzig 1983) 84.

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).

Rutishauser63: H. Rutishauser, Ausdehnung des Rombergschen Prinzips (Extension of Romberg's Principle), Numer. Math. 5 (1963) 48-54.

2 Todo List

page O2scl User's Guide Double check this documentation above

Class bin_size Not working yet.

Class collection
- If pointer_in gets a null pointer it does nothing. Should we replace this behaviour by two pointer_in() functions. One which does nothing if it gets a null pointer, and one which will go ahead and set the pointer to null. This is useful for output object which have default values to be used if they are given a null pointer.
- More testing on rewrite() function.
- Think more about adding arrays of pointers? pointers to arrays?
- Modify static data output so that if no objects of a type are included, then no static data is output for that type? (No, it's too hard to go through all objects looking for an object of a particular type).

Class columnify Move the screenify() functionality from misc.h into this class?

Class composite_inte Convert to using std::vector<inte> for the 1-d integration pointers?

Class contour
- Some contours which should be closed are not properly closed yet. See the tests for examples which fail.
- Use twod_intp for regrid_data
- Include the functionality to provide regions to be shaded instead of just lines. This can be done by providing a method, i.e. regions() which converts the curves into regions.
- Rework documentation to refer to rows and columns, not x and y

Class eqi_deriv The uncertainties in the derivatives are not yet computed and the second and third derivative formulas are not yet finished.

Global eqi_deriv::deriv_array(size_t nv, double dx, const vec_t &y, vec_t &dydx) generalize to other values of npoints.

Class gaussian_2d Double check that sigma is implemented correctly

Class gsl_fit Properly generalize other vector types than ovector_view

Class gsl_fit Allow the user to specify the derivatives

Class gsl_fit 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_inte_qag Verbose output has been setup for this class, but this needs to be done for some of the other GSL-like integrators

Class gsl_inte_qag Document workspace size here somehow

Class gsl_inte_qag Document use of last_iter

Class gsl_inte_qagiu I had to add extra code to check for non-finite values for some integrations. This should be checked.

Class gsl_inte_qawf_cos Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Class gsl_inte_qawf_sin Improve documentation a little

Class gsl_inte_qawo_cos Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Class gsl_inte_qawo_sin Improve documentation

Class gsl_inte_qaws Finish this!

Class gsl_inte_singular::extrapolation_table Improve the documentation

Class gsl_miser Document the fact that min_calls and min_calls_per_bisection need to be set beforehand

Class gsl_mmin_bfgs2 Works with generic vector objects, but doesn't allow specification of jacobian yet.

Class gsl_mmin_conf A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed, probably by rewriting take_step to produce a vec_t &x1 rather than a gsl_vector *x1;

Global gsl_mmin_conf::it_info Document this better

Class gsl_mmin_conp A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed.

Class gsl_mmin_conp Document stopping conditions

Global gsl_mmin_linmin::minimize(gsl_mmin_wrap_base &wrap, double rho, double sigma, double tau1, double tau2, double tau3, int
Properly reference Fletcher here.

Class gsl_quartic_real2 Document the distinction between this class and gsl_quartic_real

Class gsl_root_brent There is some duplication in the variables `x_lower`, `x_upper`, `a`, and `b`, which could be removed.

Class gsl_series Covert to use a more general vector

Class gsl_vegas Need to double check that the verbose output is good for all settings of verbose.

Class gsl_vegas BINS_MAX and bins_max are somehow duplicates. Fix this.

Class gsl_vegas Document the member data more carefully

Class hybrids_base Document the individual functions for this class

Class io_base Should the remove() functions be moved to class collection?

Class multi_min_fix Generalize to all vector types

Class o2scl_interp_vec Need to fix constructor to behave properly if init() fails. It should free the memory and set `ln` to zero.

Class ode_it_solve Max and average tolerance?

Class ode_it_solve partial correction option?

Global ode_iv_solve::solve_grid_derivs(double x0, double x1, double h, size_t n, vec_t &ystart, size_t nsol, vec_t &xsol, mat_t &ysol, ma
Add error information

Global ode_iv_solve::solve_final_value_derivs(double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_t &dydx_start, v
Add error information

Class ool_constr_mmin Implement automatic computations of gradient and Hessian

Class ool_constr_mmin Construct a non-trivial example for the "examples" directory

Class ool_constr_mmin Finish mmin() interface

Global ool_constr_mmin::mmin(size_t nvar, vec_t &xx, double &fmin, param_t &pa, func_t &ff) Need to finish this function somehow since it's pure virtual in multi_min.

Class ool_mmin_pgrad Complete the mmin() interface with automatic gradient

Class ool_mmin_pgrad Replace the explicit norm computation below with the more accurate dnrm2 from linalg

Class other_todos_and_bugs
- The o2scl-test and o2scl-examples targets require grep, awk, tail, cat, and wc. It would be good to reduce this list to ensure better compatibility.
- More examples and benchmarks
- Document a list of all global functions and operators
- Make sure we have a uvector_alloc, 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

Class ovector_cx_tlate There is a slight difference between how this works in comparison to MV++. The function allocate() operates a little differently than newsize(), as it will feel free to allocate new memory when owner is false. It's not clear if this is an issue, however, since it doesn't appear possible to create an ovector_cx_tlate with a value of `owner` equal to zero. This situation ought to be clarified further.

Class ovector_cx_tlate Add subvector_stride, const_subvector_stride

Class ovector_cx_view_tlate Move conversion b/w complex<double> and gsl_complex to cx_arith.h

Class ovector_view_tlate Check about self assignment as noted in `http://www.cs.caltech.edu/courses/cs11/material/cpp`

Class ovector_view_tlate Need to double-check and ensure operator[] and operator() are properly available on all of the various ovector_view children.

Class polylog
- Give error estimate?

- Improve accuracy?
- Use more sophisticated interpolation?
- Add the series $Li(n,x) = x + 2^{-n}x^2 + 3^{-n}x^3 + ...$ for $x \to 0$?
- Implement for positive arguments < 1.0
- Make another polylog class which implements series acceleration?

Global rnga::clock_seed() Ensure this function is ANSI compatible

Class search_vec The documentation here is still kind of unclear.

Class simple_jacobian Double check that this class works with arrays

Class simple_quartic_real 3/8/07 - Compilation at the NSCL produced non-finite values in solve_r() for some values of the coefficients. This should be checked.

Class simple_quartic_real It looks like this code is tested only for a4=1, and if so, the tests should be generalized.

Class simple_quartic_real Also, there is a hard-coded number in here (10^{-6}), which might be moved to a data member?

Global smart_interp::find_subset(const double a, const double b, size_t sz, const vec_t &x, const vec_t &y, size_t &nsz, bool &increasing After row and row2 are set, check to make sure the entire inside region is monotonic before expanding

Class table Better testing of automatic resizing with user- and class-owned columns

Global table::set_nlines_auto(size_t il) Resolve whether set() should really use this approach. Also, resolve whether this should replace set_nlines() (It could be that the answer is no, because as the documentation in the other version states, the other version is useful if you have columns not owned by the table.)

Global table::new_column(std::string name, ovector_view *ldat) We've got to figure out what to do if ldat is too small. If it's smaller than nlines, obviously we should just fail, but what if it's size is between nlines and maxlines?

Class tensor More complete testing.

Class tensor Add const get functions for const references

Global text_out_file::text_out_file(std::ostream *out_file, int width=80) Ensure streams are not opened in binary mode for safety.

Class timer_gettod Better testing which doesn't use a fixed number of mathematical operations, but automatically selects enough operations.

Class uvector_tlate Create a sort_unique() method as in ovector.

File array.h Ensure that array_row works, either here or in src/ode/ode_it_solve_ts.cpp

File cblas_base.h Finish dgemm()

Global matrix_out If all of the matrix elements are positive integers and scientific mode is not set, then we can avoid printing the extra spaces.

File cx_arith.h Define operators with assignment for complex + double

File cx_arith.h Ensure all the trig functions are tested

Global screenify Convert to the new version "screenify2"

Global double_to_html Add a pad_zeros parameter as in double_to_latex().

Global double_to_latex Consider converting to a class so the user can modify the strings used in giving the exponents, etc.

Global operator<< This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Global operator<< This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

File vec_arith.h Properly document the operators defined as macros

3 Download O2scl

The present version is 0.805. The source distribution can be obtained from

- `http://sourceforge.net/project/showfiles.php?group_id=206918`

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
```

4 Ideas for future development

Class akima_interp It appears that the interp() function below searches for indices slightly differently than the original GSL eval() function. This should be checked, as it might be slightly non-optimal in terms of speed (shouldn't matter for the accuracy).

Class base_interp These might work for decreasing functions by just replacing calls to search_vec::bsearch_inc() with search_vec::bsearch_dec(). If this is the case, then this should be rewritten accordingly. (I think I might have removed the acceleration)

Class cern_adapt Allow user to set the initial segments?

Class cern_gauss Allow user to change `cst`?

Class cern_mroot Modify this so it handles functions which return non-zero values.

Class cern_mroot Move some of the memory allocation out of msolve()

Class cern_mroot Give the user access to the number of function calls

Class cern_mroot_root Double-check this class to make sure it cannot fail while returning 0 for success.

Class contour
- Work on how memory is allocated
- Create a new separate struct for contour curves

Class deriv Improve the methods for second and third derivatives

Class deriv This does not have pure virtual functions, but I'd still like to prevent the user from directly instantiating a deriv object.

Class file_detect Allow the user to specify the compression commands in configure, or at least specify the path to gzip, bzip2, etc.

Class gsl_anneal Implement a more general simulated annealing routine which would allow the solution of discrete problems like the Traveling Salesman problem.

Class gsl_anneal Implement a method which automatically minimizes within some specified tolerance?

Class gsl_astep Fix so that memory allocation/deallocation is performed only when necessary

Class gsl_astep Allow user to find out how many steps were taken, etc.

Class gsl_chebapp Implement eval_err(), eval_n() and eval_n_err() methods.

Class gsl_deriv Include the forward and backward GSL derivatives

Class gsl_inte_cheb 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_kronrod::gsl_integration_qk_o2scl(func_t &func, const int n, const double xgk[], const double wg[], const double wgk[],
This function, in principle, could be replaced with a generic integration pointer.

Class gsl_inte_qng Compare directly with GSL as is done in gsl_inte_qag_ts.

Global gsl_inte_singular::qags(func_t &func, const int qn, const double xgk[], const double wg[], const double wgk[], double fv1[], doub
Remove goto statements?

Class gsl_inte_singular::extrapolation_table Move this to a new class, with qelg() as a method

Class gsl_inte_table Move gsl_integration_workspace to a separate class and remove this class, making all children direct descendants of gsl_inte instead. We'll have to figure out what to do with the data member `wkspace` though. Some work on this front is already in gsl_inte_qag_b.h.

Class gsl_mmin_wrapper There's a bit of extra vector copying here which could potentially be avoided.

Class gsl_root_stef There's some extra copying here which can probably be removed.

Class gsl_root_stef Compare directly to GSL.

Class gsl_vegas Could convert bins and boxes to a more useful structure

Class lanczos The function eigen_tdiag() automatically sorts the eigenvalues, which may not be necessary.

Class minimize This does not have pure virtual functions, but I'd still like to prevent the user from directly instantiating a minimize object.

Global minimize::bracket(double &ax, double &bx, double &cx, double &fa, double &fb, double &fc, param_t &pa, func_t &func)
Improve this algorithm with the standard golden ratio method?

Class nonadapt_step Modify so that memory allocation/deallocation is only performed when necessary

Class ode_bv_shoot Implement shooting from an internal point, either using a different class or using this one.

Global ode_iv_solve::solve_table(double x0, double x1, double h, size_t n, vec_t &ystart, size_t &nsol, vec_t &xsol, mat_t &ysol, param_
Consider modifying so that this can handle tables which are too small by removing half the rows and doubling the stepsize.

Class omatrix_view_tlate This class isn't sufficiently general for some applications, such as sub-matrices of higher-dimensional structures. It might be nice to create a more general class with a "stride" and a "tda".

Class omatrix_view_tlate The xmatrix class demonstrates how operator[] could return an ovector_array object and thus provide better bounds-checking. This would demand including a new parameter in omatrix_view_tlate which contains the vector type.

Class other_todos_and_bugs There may be a problem with const-correctness in vectors. I'm not sure how it's best solved. It could be best to create two kinds of ovector_view's: one const and one not. 10/19/07: I think it's the case that neither ovector_-const_subvector, or const ovector_subvector are truly const, but it's only const ovector_const_subvector that would be truly const. I'm not sure if this is related to the issue of constness in ovector_view discussed above.

Class other_todos_and_bugs Consider breaking documentation up into sections?

Class ovector_view_tlate Consider an operator== ?

Class pinside The inside() functions actually copy the points twice. This can be made more efficient.

Class planar_intp 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.

Class root This does not have pure virtual functions, but I'd still like to prevent the user from directly instantiating a root object.

Class simple_jacobian GSL-1.10 updated fdjac.c and this update could be implemented below.

Class table Be more restrictive about allowable column names

Class table Add interp() and related functions which avoid caching and can thus be const (This has been started with interp_const())

Class table The nlines vs. maxlines and automatic resizing of table-owned vs. user-owned vectors could be reconsidered, especially now that ovectors can automatically resize on their own. 10/16/07: This issue may be unimportant, as it might be better to just move to a template based approach with a user-specified vector type. The interpolation is now flexible enough to handle different types. Might check to ensure sorting works with other types.

Class table 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 tensor Could implement arithmetic operators + and - and some different products.

Class tensor Add slicing to get ovector or omatrix objects

Class tensor_grid Only allocate space for grid if it is set

Class tensor_grid Could implement arithmetic operators + and - and some different products.

Global tensor_grid::set_grid(double ∗∗val) Define a more generic interface for matrix types

Global tensor_grid::interpolate(double ∗vals) 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.

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.

Class twod_intp Implement an improved caching system in case, for example `xfirst` is true and the last interpolation used the same value of `x`.

Global twod_intp::set_interp(size_t ni, base_interp< ovector_view > ∗it, base_interp< ovector_const_subvector > ∗it_sub, base_interp<
Use std::vector for the first two base_interp arguments?

Class uvector_cx_view_tlate Write lookup() method, and possibly an erase() method.

Class uvector_view_tlate Could allow user-defined specification of restrict keyword

File cblas_base.h Convert to size_t and add float and complex versions

Global dscal_subcol Implement explicit loop unrolling

Global daxpy_hv_sub Implement explicit loop unrolling

Global ddot_hv_sub Implement explicit loop unrolling

Global err_assert Make this consistent with assert() using NDEBUG?

Global LU_invert could rewrite to avoid mat_col_t

Global operator<< This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Global solve_tridiag_sym Convert into class for memory managment and combine with other functions below

File vec_arith.h Define operators for complex vector $*$ real matrix

File vec_arith.h These should be replaced by the BLAS routines where possible?

Global matrix_cx_copy_gsl At present this works only with complex types based directly on the GSL complex format. This could be improved.

Global vector_cx_copy_gsl At present this works only with complex types based directly on the GSL complex format. This could be improved.

5 Bug List

Class collection
- Ensure that the user cannot add a object with a name of ptrXXX.
- Test_type does not test handle static data or pointers.
- Check strings and words for characters that we can't handle
- The present version of a text-file requires strings to contain at least one printable character.
- Ensure that all matching is done by both type and name if possible.

Class other_todos_and_bugs
- BLAS libraries not named `libblas` or `libgslblas` are not properly detected in ./configure and will have to be added manually.
- The -lm flag may not be added properly by ./configure

Class quad_intp This class doesn't seem to work at present.

6 Namespace Documentation

6.1 gsl_cgs Namespace Reference

6.1.1 Detailed Description

GSL constants in CGS units.

The CGS units are given below each constant

Variables

- const double schwarzchild_radius = 2.95325008e5
  cm
- const double speed_of_light = 2.99792458e10
  cm / s
- const double gravitational_constant = 6.673e-8
  cm^3 / g s^2
- const double plancks_constant_h = 6.62606876e-27
  g cm^2 / s
- const double plancks_constant_hbar = 1.05457159642e-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 = 1.602176462e-12
  g cm^2 / s^2
- const double mass_electron = 9.10938188e-28
  g
- const double mass_muon = 1.88353109e-25
  g
- const double mass_proton = 1.67262158e-24
  g
- const double mass_neutron = 1.67492716e-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 bohr_magneton = 9.27400899e-20
  A cm^2.
- const double nuclear_magneton = 5.05078317e-23
  A cm^2.
- const double electron_magnetic_moment = 9.28476362e-20
  A cm^2.
- const double proton_magnetic_moment = 1.410606633e-22
  A 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 = 1.66053873e-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 = 9.6485341472e4
  A s / mol.
- const double electron_charge = 1.602176462e-19
  A s.
- const double gauss = 1e-1
  g / A s^2
- 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

6.2 gsl_cgsm Namespace Reference

6.2.1 Detailed Description

GSL constants in CGSM units.

The CGSM units are given below each constant

Variables

- const double schwarzchild_radius = 2.95325008e5
  cm
- const double speed_of_light = 2.99792458e10
  cm / s
- const double gravitational_constant = 6.673e-8
  cm^3 / g s^2
- const double plancks_constant_h = 6.62606876e-27
  g cm^2 / s
- const double plancks_constant_hbar = 1.05457159642e-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 = 1.602176462e-12
  g cm^2 / s^2

- const double mass_electron = 9.10938188e-28
  g
- const double mass_muon = 1.88353109e-25
  g
- const double mass_proton = 1.67262158e-24
  g
- const double mass_neutron = 1.67492716e-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 bohr_magneton = 9.27400899e-21
  abamp cm^2
- const double nuclear_magneton = 5.05078317e-24
  abamp cm^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 = 1.66053873e-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 = 9.6485341472e3
  abamp s / mol
- const double electron_charge = 1.602176462e-20
  abamp s
- const double gauss = 1e0
  g / abamp s^2
- 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

6.3 gsl_mks Namespace Reference

6.3.1 Detailed Description

GSL constants in MKS units.

The MKS units are given below each constant

Variables

- const double schwarzchild_radius = 2.95325008e3
  m
- const double speed_of_light = 2.99792458e8
  m / s
- const double gravitational_constant = 6.673e-11
  m^3 / kg s^2
- const double plancks_constant_h = 6.62606876e-34
  kg m^2 / s
- const double plancks_constant_hbar = 1.05457159642e-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 = 1.602176462e-19
  kg m^2 / s^2
- const double mass_electron = 9.10938188e-31
  kg
- const double mass_muon = 1.88353109e-28
  kg
- const double mass_proton = 1.67262158e-27
  kg
- const double mass_neutron = 1.67492716e-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 = 1.66053873e-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 = 9.6485341472e4
  A s / mol.
- const double electron_charge = 1.602176462e-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

6.4 gsl_mksa Namespace Reference

6.4.1 Detailed Description

GSL constants in MKSA units.

The MKSA units are given below each constant

Variables

- const double schwarzchild_radius = 2.95325008e3
  m
- const double speed_of_light = 2.99792458e8
  m / s
- const double gravitational_constant = 6.673e-11
  m^3 / kg s^2
- const double plancks_constant_h = 6.62606876e-34
  kg m^2 / s
- const double plancks_constant_hbar = 1.05457159642e-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 = 1.602176462e-19
  kg m^2 / s^2
- const double mass_electron = 9.10938188e-31
  kg
- const double mass_muon = 1.88353109e-28
  kg
- const double mass_proton = 1.67262158e-27
  kg
- const double mass_neutron = 1.67492716e-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 = 1.66053873e-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 = 9.6485341472e4
  A s / mol.
- const double electron_charge = 1.602176462e-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

6.5 gsl_num Namespace Reference

6.5.1 Detailed Description

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`).

6.6 o2scl Namespace Reference

6.6.1 Detailed Description

The main O_2scl namespace.

By default, all O_2scl classes and functions which are not listed as being in one of O_2scl 's smaller specialized namespaces are in this namespace. This namespace has been removed from the documentation to simplify the formatting.

Functions

- template<class vec_t, class vec2_t>
  int solve_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec_t &b, vec_t &x, size_t N)
  Solve a symmetric tridiagonal linear system.
- template<class vec_t, class vec2_t>
  int solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec2_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)
  Solve an asymmetric tridiagonal linear system.
- template<class vec_t>
  int solve_cyc_tridiag_sym (const vec_t &diag, const vec_t &offdiag, const vec_t &b, vec_t &x, size_t N)
  Solve a symmetric cyclic tridiagonal linear system.
- template<class vec_t>
  int solve_cyc_tridiag_nonsym (const vec_t &diag, const vec_t &abovediag, const vec_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)
  Solve an asymmetric cyclic tridiagonal linear system.

6.6.2 Function Documentation

6.6.2.1 int o2scl::solve_cyc_tridiag_nonsym (const vec_t & *diag*, const vec_t & *abovediag*, const vec_t & *belowdiag*, const vec_t & *rhs*, vec_t & *x*, size_t *N*) `[inline]`

Solve an asymmetric cyclic tridiagonal linear system.

solve following system w/o the corner elements and then use Sherman-Morrison formula to compensate for them

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] ...

Definition at line 289 of file tridiag_base.h.

6.6.2.2 int o2scl::solve_cyc_tridiag_sym (const vec_t & *diag*, const vec_t & *offdiag*, const vec_t & *b*, vec_t & *x*, size_t *N*) `[inline]`

Solve a symmetric cyclic tridiagonal linear system.

for description of method see [Engeln-Mullges + Uhlig, p. 96]

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] ...

Definition at line 189 of file tridiag_base.h.

6.6.2.3 int o2scl::solve_tridiag_nonsym (const vec_t & *diag*, const vec2_t & *abovediag*, const vec2_t & *belowdiag*, const vec_t & *rhs*, vec_t & *x*, size_t *N*) `[inline]`

Solve an asymmetric tridiagonal linear system.

plain gauss elimination, only not bothering with the zeroes

diag[0] abovediag[0] 0 ..... belowdiag[0] diag[1] abovediag[1] ..... 0 belowdiag[1] diag[2] 0 0 belowdiag[2] .....

Definition at line 122 of file tridiag_base.h.

6.6.2.4 int o2scl::solve_tridiag_sym (const vec_t & *diag*, const vec2_t & *offdiag*, const vec_t & *b*, vec_t & *x*, size_t *N*) [inline]

Solve a symmetric tridiagonal linear system.

Idea for future

Convert into class for memory managment and combine with other functions below

For description of method see [Engeln-Mullges + Uhlig, p. 92]

diag[0] offdiag[0] 0 ..... offdiag[0] diag[1] offdiag[1] ..... 0 offdiag[1] diag[2] 0 0 offdiag[2] .....

Definition at line 49 of file tridiag_base.h.

6.7 o2scl_arith Namespace Reference

6.7.1 Detailed Description

A namespace for arithmetic on complex numbers and vectors.

Functions

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)

6.8 o2scl_cblas Namespace Reference

6.8.1 Detailed Description

Namespace for O2scl CBLAS function templates with operator[].

Enumerations

- enum O2CBLAS_ORDER { **O2cblasRowMajor** = 101, **O2cblasColMajor** = 102 }
  Matrix order, either column-major or row-major.
- enum O2CBLAS_TRANSPOSE { **O2cblasNoTrans** = 111, **O2cblasTrans** = 112, **O2cblasConjTrans** = 113 }
  Transpose operations.
- enum O2CBLAS_UPLO { **O2cblasUpper** = 121, **O2cblasLower** = 122 }
  Upper- or lower-triangular.
- enum O2CBLAS_DIAG { **O2cblasNonUnit** = 131, **O2cblasUnit** = 132 }
  Unit or generic diagonal.
- enum O2CBLAS_SIDE { **O2cblasLeft** = 141, **O2cblasRight** = 142 }
  Left or right sided operation.

Functions

- template<class mat_t, class vec_t>
  int dgemm (const enum O2CBLAS_ORDER Order, const enum O2CBLAS_TRANSPOSE TransA, const enum O2CBLAS_-TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const mat_t &A, const mat_t &B, const double beta, mat_t &C)
  Compute $y = \alpha \mathrm{op}(A) x + \beta y$.

Standard BLAS functions

- template<class vec_t, class vec2_t>
  void daxpy (const int N, const double alpha, const vec_t &X, vec2_t &Y)
  Compute $y = \alpha x + y$.
- template<class vec_t, class vec2_t>
  double ddot (const int N, const vec_t &X, const vec2_t &Y)
  Compute $r = x \cdot y$.
- template<class vec_t>
  void dscal (const int N, const double alpha, vec_t &X)
  Compute $x = \alpha x$.
- template<class vec_t>
  double dnrm2 (const int N, const vec_t &X)
  Compute the squared norm of the vector `X`.
- template<class mat_t, class vec_t>
  int dgemv (const enum O2CBLAS_ORDER order, const enum O2CBLAS_TRANSPOSE TransA, const int M, const int 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>
  int dtrsv (const enum O2CBLAS_ORDER order, const enum O2CBLAS_UPLO Uplo, const enum O2CBLAS_-TRANSPOSE TransA, const enum O2CBLAS_DIAG Diag, const int M, const int N, const mat_t &A, vec_t &X)
  Compute $x = \mathrm{op}(A)^{-1} x$.

Helper BLAS functions

- template<class vec_t, class vec2_t>
  void daxpy_subvec (const int N, const double alpha, const vec_t &X, vec2_t &Y, const int ie)
  Compute $x = \alpha x$ beginning with index `ie` *and ending with index* `N-1`.

- template<class vec_t, class vec2_t>
  double ddot_subvec (const int N, const vec_t &X, const vec2_t &Y, const int ie)
  Compute $r = x \cdot y$ beginning with index `ie` *and ending with index* `N-1`.
- template<class vec_t>
  void dscal_subvec (const int N, const double alpha, vec_t &X, const int ie)
  Compute $x = \alpha x$ beginning with index `ie` *and ending with index* `N-1`.
- template<class vec_t>
  double dnrm2_subvec (const int N, const vec_t &X, const int ie)
  Compute the squared norm of the vector `X` *beginning with index* `ie` *and ending with index* `N-1`.
- template<class mat_t>
  double dnrm2_subcol (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
  Compute the squared norm of the last `N` *rows of a column of a matrix.*
- template<class mat_t>
  void dscal_subcol (mat_t &A, const size_t ir, const size_t ic, const size_t n, const double alpha)
  Compute $x = \alpha x$.
- template<class mat_t, class vec_t>
  void daxpy_hv_sub (const int N, const double alpha, const mat_t &X, vec_t &Y, const int ie)
  Compute $x = \alpha x$ for householder_hv_sub().
- template<class mat_t, class vec_t>
  double ddot_hv_sub (const int N, const mat_t &X, const vec_t &Y, const int ie)
  Compute $r = x \cdot y$ for householder_hv_sub().

6.8.2 Function Documentation

6.8.2.1 void o2scl_cblas::daxpy_hv_sub (const int *N*, const double *alpha*, const mat_t & *X*, vec_t & *Y*, const int *ie*) `[inline]`

Compute $x = \alpha x$ for householder_hv_sub().

Used in householder_hv_sub().

Idea for future

Implement explicit loop unrolling

Definition at line 560 of file cblas_base.h.

6.8.2.2 void o2scl_cblas::daxpy_subvec (const int *N*, const double *alpha*, const vec_t & *X*, vec2_t & *Y*, const int *ie*) `[inline]`

Compute $x = \alpha x$ beginning with index `ie` and ending with index `N-1`.

Used in householder_hv().

Definition at line 380 of file cblas_base.h.

6.8.2.3 double o2scl_cblas::ddot_hv_sub (const int *N*, const mat_t & *X*, const vec_t & *Y*, const int *ie*) `[inline]`

Compute $r = x \cdot y$ for householder_hv_sub().

Used in householder_hv_sub().

Idea for future

Implement explicit loop unrolling

Definition at line 580 of file cblas_base.h.

6.8.2.4 double o2scl_cblas::ddot_subvec (const int *N*, const vec_t & *X*, const vec2_t & *Y*, const int *ie*) [inline]

Compute $r = x \cdot y$ beginning with index `ie` and ending with index `N-1`.

Used in householder_hv().

Definition at line 410 of file cblas_base.h.

6.8.2.5 double o2scl_cblas::dnrm2_subcol (const mat_t & *M*, const size_t *ir*, const size_t *ic*, const size_t *N*) [inline]

Compute the squared norm of the last `N` rows of a column of a matrix.

Given matrix `M`, this computes the norm of the last `N` rows of the column with index `ic`, beginning with the element with index `ir`. If the matrix `M` has `r` rows, and `c` columns, then the parameter `N` should be `r-ir`.

Used in householder_transform_subcol().

Definition at line 505 of file cblas_base.h.

6.8.2.6 double o2scl_cblas::dnrm2_subvec (const int *N*, const vec_t & *X*, const int *ie*) [inline]

Compute the squared norm of the vector `X` beginning with index `ie` and ending with index `N-1`.

Used in householder_transform().

Definition at line 462 of file cblas_base.h.

6.8.2.7 void o2scl_cblas::dscal_subcol (mat_t & *A*, const size_t *ir*, const size_t *ic*, const size_t *n*, const double *alpha*) [inline]

Compute $x = \alpha x$.

Used in householder_transform_subcol().

Idea for future

Implement explicit loop unrolling

Definition at line 545 of file cblas_base.h.

6.8.2.8 void o2scl_cblas::dscal_subvec (const int *N*, const double *alpha*, vec_t & *X*, const int *ie*) [inline]

Compute $x = \alpha x$ beginning with index `ie` and ending with index `N-1`.

Used in householder_transform().

Definition at line 438 of file cblas_base.h.

6.9 o2scl_cblas_paren Namespace Reference

6.9.1 Detailed Description

Namespace for O2scl CBLAS function templates with operator().

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.

6.10 o2scl_const Namespace Reference

6.10.1 Detailed Description

O2scl constants.

Variables

- 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 ev_mks = 1.602176462e-19
  1 eV in $kg \cdot m^2/s^2$ (Joules)
- 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

- const double hc_mev_fm = 197.3269631
  $\hbar c$ in MeV fm
- const double gfermi_gev = 1.16637e-5
  Fermi coupling constant (G_F) in GeV^{-2}.
- const double hc_mev_cm = 1.973269631e-11
  $\hbar c$ 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.

6.10.2 Variable Documentation

6.10.2.1 const double e2_gaussian = o2scl_const::hc_mev_fm*gsl_num::fine_structure

Electron charge squared in Gaussian units.

In Gaussian Units:

$$\vec{\nabla}\cdot\vec{E} = 4\pi\rho\,, \quad \vec{E} = -\vec{\nabla}\Phi\,, \quad \nabla^2\Phi = -4\pi\rho\,,$$
$$F = \tfrac{q_1 q_2}{r^2}\,, \quad W = \tfrac{1}{2}\int \rho V d^3x = \tfrac{1}{8\pi}\int |\vec{E}|^2 d^3x\,, \quad \alpha = \tfrac{e^2}{\hbar c} = \tfrac{1}{137}$$

Definition at line 968 of file constants.h.

6.10.2.2 const double e2_hlorentz = gsl_num::fine_structure*4.0*pi

Electron charge sqaured in Heaviside-Lorentz units where $\hbar = c = 1$.

In Heaviside-Lorentz units:

$$\vec{\nabla}\cdot\vec{E} = \rho\,, \quad \vec{E} = -\vec{\nabla}\Phi\,, \quad \nabla^2\Phi = -\rho\,,$$
$$F = \tfrac{q_1 q_2}{4\pi r^2}\,, \quad W = \tfrac{1}{2}\int \rho V d^3x = \tfrac{1}{2}\int |\vec{E}|^2 d^3x\,, \quad \alpha = \tfrac{e^2}{4\pi} = \tfrac{1}{137}$$

Definition at line 988 of file constants.h.

6.10.2.3 const double e2_mksa = gsl_mksa::electron_charge

Electron charge squared in SI(MKSA) units.

In MKSA units:

$$\vec{\nabla}\cdot\vec{E} = \rho\,, \quad \vec{E} = -\vec{\nabla}\Phi\,, \quad \nabla^2\Phi = -\rho\,,$$
$$F = \tfrac{1}{4\pi\varepsilon_0}\tfrac{q_1 q_2}{r^2}\,, \quad W = \tfrac{1}{2}\int \rho V d^3x = \tfrac{\varepsilon_0}{2}\int |\vec{E}|^2 d^3x\,, \quad \alpha = \tfrac{e^2}{4\pi\varepsilon_0\hbar c} = \tfrac{1}{137}$$

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 1014 of file constants.h.

6.11 o2scl_fm Namespace Reference

6.11.1 Detailed Description

Constants in units of fm.

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}.

Variables

- const double mev = 1.0/o2scl_const::hc_mev_fm
  1 MeV *in* fm^{-1}

- const double kg = mev/1.782661731e-30
  1 kg *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 Kelvin = 8.617342e-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 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

Masses from Particle Physics Booklet

(see also D.E. Groom, et. al., Euro. Phys. J. C 15 (2000) 1.)

- const double mass_electron = 0.510998902/o2scl_const::hc_mev_fm
  Electron mass in fm^{-1}.
- const double mass_muon = 105.658357/o2scl_const::hc_mev_fm
  Muon mass in fm^{-1}.
- const double mass_amu = 931.494013/o2scl_const::hc_mev_fm
  Atomic mass unit in fm^{-1}.
- const double mass_neutron = 939.565/o2scl_const::hc_mev_fm
  Neutron mass in fm^{-1}.
- const double mass_proton = 938.272/o2scl_const::hc_mev_fm
  Proton mass in fm^{-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}

www.nist.gov

- const double mass_alpha = 3727.37905/o2scl_const::hc_mev_fm
  Alpha particle mass in fm^{-1}.

6.11.2 Variable Documentation

6.11.2.1 const double mass_alpha = 3727.37905/o2scl_const::hc_mev_fm

Alpha particle mass in fm^{-1}.

This does not include the mass of the additional two electrons which are present in a helium atom.

Definition at line 1079 of file constants.h.

6.12 o2scl_inte_qag_coeffs Namespace Reference

6.12.1 Detailed Description

A namespace for the GSL adaptive integration coefficients.

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.

6.13 o2scl_inte_qng_coeffs Namespace Reference

6.13.1 Detailed Description

A namespace for the quadrature coefficients for non-adaptive integration.

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]

6.13.2 Variable Documentation

6.13.2.1 const double w10[5] `[static]`

Weights of the 10-point formula

Definition at line 51 of file gsl_inte_qng.h.

6.13.2.2 const double w21a[5] `[static]`

Weights of the 21-point formula for abscissae x1

Definition at line 69 of file gsl_inte_qng.h.

6.13.2.3 const double w21b[6] `[static]`

Weights of the 21-point formula for abscissae x2

Definition at line 78 of file gsl_inte_qng.h.

6.13.2.4 const double w43a[10] `[static]`

Weights of the 43-point formula for abscissae x1, x3

Definition at line 103 of file gsl_inte_qng.h.

6.13.2.5 const double w43b[12] `[static]`

Weights of the 43-point formula for abscissae x3

Definition at line 117 of file gsl_inte_qng.h.

6.13.2.6 const double w87a[21] `[static]`

Weights of the 87-point formula for abscissae x1, x2, x3

Definition at line 159 of file gsl_inte_qng.h.

6.13.2.7 const double w87b[23] `[static]`

Weights of the 87-point formula for abscissae x4

Definition at line 184 of file gsl_inte_qng.h.

6.13.2.8 const double x1[5] `[static]`

Abscissae common to the 10-, 21-, 43- and 87-point rule

Definition at line 42 of file gsl_inte_qng.h.

6.13.2.9 const double x2[5] `[static]`

Abscissae common to the 21-, 43- and 87-point rule

Definition at line 60 of file gsl_inte_qng.h.

6.13.2.10 const double x3[11] `[static]`

Abscissae common to the 43- and 87-point rule

Definition at line 88 of file gsl_inte_qng.h.

6.13.2.11 const double x4[22] `[static]`

Abscissae of the 87-point rule

Definition at line 133 of file gsl_inte_qng.h.

6.14 o2scl_linalg Namespace Reference

6.14.1 Detailed Description

Namespace for O2scl linear algebra function templates with operator[].

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)
  Desc.
- 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)
  Desc.
- template<class vec_t>
  void apply_givens_vec (vec_t &v, size_t i, size_t j, double c, double s)
  Desc.
- template<class mat_t, class vec_t>
  int HH_solve (size_t n, mat_t &A, const vec_t &b, vec_t &x)
  Desc.
- template<class mat_t, class vec_t>
  int HH_svx (size_t N, size_t M, mat_t &A, vec_t &x)
  Desc.
- 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 the last* `n-1` *elements of* `v`*.*

- template<class mat_t>
  double householder_transform_subcol (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` *rows of a column of a matrix.*
- template<class vec_t, class mat_t>
  int householder_hm (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
  Apply a householder transformation v,tau to matrix m.
- template<class mat_t>
  int 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 `v`, `tau` *to submatrix of m.*
- template<class vec_t>
  int householder_hv (const size_t N, double tau, const vec_t &v, vec_t &w)
  Apply a householder transformation `v` *to vector* `w`*.*
- template<class mat_t, class vec_t>
  int householder_hv_sub (const mat_t &M, vec_t &w, double tau, const size_t ie, const size_t N)
  Apply a householder transformation `v` *to vector* `w`*.*
- template<class mat1_t, class mat2_t>
  int 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 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 vec_t>
  int LU_solve (const size_t N, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_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 vec_t>
  int LU_refine (const size_t N, const mat_t &A, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_t &x, vec_t &residual)
  Refine the solution of a linear system.
- template<class mat_t, class mat_col_t>
  int LU_invert (const size_t N, const mat_t &LU, const o2scl::permutation &p, mat_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>
  int 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>
  int QR_solve (size_t N, const mat_t &QR, const vec_t &tau, const vec_t &b, vec_t &x)
  Solve the system A x = b using the QR factorization.
- template<class mat_t, class vec_t>
  int QR_svx (size_t M, size_t N, const mat_t &QR, const vec_t &tau, vec_t &x)
  Solve the system A x = b in place using the QR factorization.
- template<class mat_t, class vec_t>
  int QR_QTvec (const size_t M, const size_t N, const mat_t &QR, const vec_t &tau, vec_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>
  int 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>
  int 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,.

6.14.2 Function Documentation

6.14.2.1 int 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*) [inline]

Apply a householder transformation `v`, `tau` to submatrix of m.

Used in QR_decomp().

Definition at line 143 of file householder_base.h.

6.14.2.2 int o2scl_linalg::householder_hv_sub (const mat_t & *M*, vec_t & *w*, double *tau*, const size_t *ie*, const size_t *N*) [inline]

Apply a householder transformation `v` to vector `w`.

Used in QR_QTvec().

Definition at line 211 of file householder_base.h.

6.14.2.3 double o2scl_linalg::householder_transform_subcol (mat_t & *A*, const size_t *ir*, const size_t *ic*, const size_t *n*) [inline]

Compute the householder transform of a vector formed with the last `n` rows of a column of a matrix.

Used in QR_decomp().

Definition at line 75 of file householder_base.h.

6.14.2.4 int o2scl_linalg::LU_decomp (const size_t *N*, mat_t & *A*, o2scl::permutation & *p*, int & *signum*) [inline]

Compute the LU decomposition of the matrix `A`.

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).

Definition at line 51 of file lu_base.h.

6.14.2.5 double o2scl_linalg::LU_det (const size_t *N*, const mat_t & *LU*, int *signum*) [inline]

Compute the determinant of a matrix from its LU decomposition.

These functions 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 230 of file lu_base.h.

6.14.2.6 int o2scl_linalg::LU_invert (const size_t *N*, const mat_t & *LU*, const o2scl::permutation & *p*, mat_t & *inverse*) [inline]

Compute the inverse of a matrix from its LU decomposition.

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 (consult any introductory textbook on numerical linear algebra for details).

Idea for future

could rewrite to avoid mat_col_t

Definition at line 195 of file lu_base.h.

6.14.2.7 double o2scl_linalg::LU_lndet (const size_t *N*, const mat_t & *LU*) [inline]

Compute the logarithm of the absolute value of the determinant of a matrix from its LU decomposition.

These functions 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 253 of file lu_base.h.

6.14.2.8 int o2scl_linalg::LU_refine (const size_t *N*, const mat_t & *A*, const mat_t & *LU*, const o2scl::permutation & *p*, const vec_t & *b*, vec_t & *x*, vec_t & *residual*) [inline]

Refine the solution of a linear system.

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 162 of file lu_base.h.

6.14.2.9 int o2scl_linalg::LU_sgndet (const size_t *N*, const mat_t & *LU*, int *signum*) [inline]

Compute the sign of the determinant of a matrix from its LU decomposition.

These functions 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 274 of file lu_base.h.

6.14.2.10 int o2scl_linalg::LU_solve (const size_t *N*, const mat_t & *LU*, const o2scl::permutation & *p*, const vec_t & *b*, vec_t & *x*) [inline]

Solve a linear system after LU decomposition.

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 113 of file lu_base.h.

6.14.2.11 int o2scl_linalg::LU_svx (const size_t *N*, const mat_t & *LU*, const o2scl::permutation & *p*, vec_t & *x*) [inline]

Solve a linear system after LU decomposition in place.

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 134 of file lu_base.h.

6.14.2.12 int o2scl_linalg::QR_update (size_t *M*, size_t *N*, mat1_t & *Q*, mat2_t & *R*, vec1_t & *w*, vec2_t & *v*) [inline]

Update a QR factorisation for A= Q R , A' = A + u v^T,.

`M` and `N` are the number of rows and columns of `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 147 of file qr_base.h.

6.15 o2scl_linalg_paren Namespace Reference

6.15.1 Detailed Description

Namespace for O2scl linear algebra function templates with operator().

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.

Functions

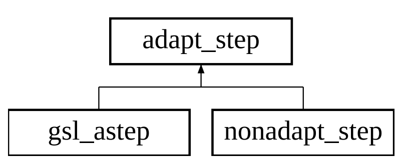
- void **create_givens** (const double a, const double b, double &c, double &s)

7 Data Structure Documentation

7.1 adapt_step Class Template Reference

`#include <adapt_step.h>`

Inheritance diagram for adapt_step::



7.1.1 Detailed Description

template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class adapt_step< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Adaptive stepper [abstract base].

The adaptive stepper routines are based on several applications of ordinary ODE steppers (implemented in odestep). 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 52 of file adapt_step.h.

Public Member Functions

- virtual int astep (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &dydx_out, vec_t &yerr, param_t &pa, func_t &derivs)=0
  Make an adaptive integration step of the system `derivs`.
- virtual int astep_derivs (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &dydx, vec_t &yerr, param_t &pa, func_t &derivs)=0
  Make an adaptive integration step of the system `derivs` *with derivatives.*
- int set_step (odestep< param_t, func_t, vec_t > &step)
  Set stepper.

Data Fields

- int verbose
  Set output level.
- gsl_rkck< param_t, func_t, vec_t, alloc_vec_t, alloc_t > def_step
  The default stepper.

Protected Attributes

- odestep< param_t, func_t, vec_t > * stepp
  Pointer to the stepper being used.

7.1.2 Member Function Documentation

7.1.2.1 virtual int astep (double & *x*, double & *h*, double *xmax*, size_t *n*, vec_t & *y*, vec_t & *dydx_out*, vec_t & *yerr*, param_t & *pa*, func_t & *derivs*) `[pure virtual]`

Make an adaptive integration step of the system `derivs`.

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, and nonadapt_step.

7.1.2.2 virtual int astep_derivs (double & *x*, double & *h*, double *xmax*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yerr*, param_t & *pa*, func_t & *derivs*) `[pure virtual]`

Make an adaptive integration step of the system `derivs` with derivatives.

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, and nonadapt_step.

7.1.2.3 int set_step (odestep< param_t, func_t, vec_t > & *step*) [inline]

Set stepper.

This sets the stepper for use in the adaptive step routine. If no stepper is specified, then the default (gsl_rkck) is used.

Definition at line 106 of file adapt_step.h.

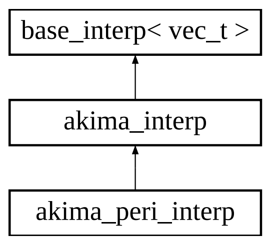
The documentation for this class was generated from the following file:

- adapt_step.h

7.2 akima_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for akima_interp::



7.2.1 Detailed Description

template<class vec_t> class akima_interp< vec_t >

Akima spline interpolation (GSL).

Idea for future

It appears that the interp() function below searches for indices slightly differently than the original GSL eval() function. This should be checked, as it might be slightly non-optimal in terms of speed (shouldn't matter for the accuracy).

Definition at line 663 of file interp.h.

Public Member Functions

- akima_interp (bool periodic=false)
  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 $\int_a^b y(x)\, dx$.

Protected Member Functions

- void akima_calc (const vec_t &x_array, size_t size, double m[ ])
  For initializing the interpolation.

Protected Attributes

- bool peri
  True for periodic boundary conditions.

Storage for Akima spline interpolation

- double ∗ **b**
- double ∗ **c**
- double ∗ **d**
- double ∗ **um**

7.2.2 Member Function Documentation

7.2.2.1 virtual int init (const vec_t & *xa*, const vec_t & *ya*, size_t *size*) `[inline, virtual]`

Initialize interpolation routine.

Periodic boundary conditions

Non-periodic boundary conditions

Reimplemented from base_interp.

Definition at line 764 of file interp.h.

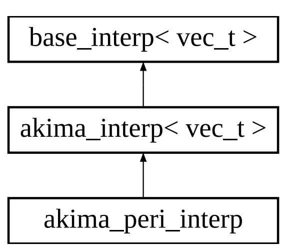
The documentation for this class was generated from the following file:

- interp.h

7.3 akima_peri_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for akima_peri_interp::



7.3.1 Detailed Description

template<class vec_t> class akima_peri_interp< vec_t >

Akima spline interpolation with periodic boundary conditions (GSL).

This is convenient to allow interpolation objects to be supplied as template parameters

Definition at line 921 of file interp.h.

The documentation for this class was generated from the following file:

- interp.h

7.4 array_2d_alloc Class Template Reference

```
#include <array.h>
```

7.4.1 Detailed Description

template<class mat_t> class array_2d_alloc< mat_t >

A simple class to provide an `allocate()` function for 2-dimensional arrays.

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

Definition at line 106 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

7.5 array_2d_column Class Template Reference

```
#include <array.h>
```

7.5.1 Detailed Description

template<size_t R, size_t C> class array_2d_column< R, C >

Column of a 2d array.

This works because two-dimensional arrays are always continguous (as indicated in appendix C of Soustroup's book)

Definition at line 266 of file array.h.

Public Member Functions

- array_2d_column (double mat[R][C], size_t i)
  Create an object as the `i`*th column of* `mat`.
- 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

7.6 array_2d_row Class Template Reference

`#include <array.h>`

7.6.1 Detailed Description

template<class array_2d_t> class array_2d_row< array_2d_t >

Row of a 2d array.

Definition at line 303 of file array.h.

Public Member Functions

- array_2d_row (array_2d_t &mat, size_t i)
  Create an object as the `i`*th row of* `mat`*.*
- 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

7.7 array_alloc Class Template Reference

`#include <array.h>`

7.7.1 Detailed Description

template<class vec_t> class array_alloc< vec_t >

A simple class to provide an `allocate()` function for arrays.

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.

Definition at line 89 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

7.8 array_const_reverse Class Template Reference

```
#include <array.h>
```

7.8.1 Detailed Description

template<size_t sz> class array_const_reverse< sz >

A simple class which reverses the order of an array.

Definition at line 188 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

7.9 array_const_subvector Class Reference

```
#include <array.h>
```

7.9.1 Detailed Description

A simple subvector class for a const array (without error checking).

Definition at line 340 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

7.10 array_const_subvector_reverse Class Reference

```
#include <array.h>
```

7.10.1 Detailed Description

Reverse a subvector of a const array.

Definition at line 423 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.

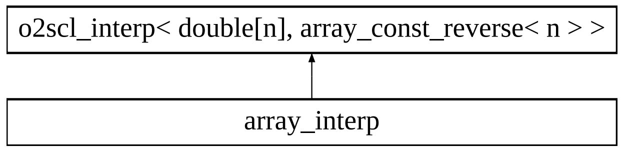
The documentation for this class was generated from the following file:

- array.h

7.11 array_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for array_interp::



7.11.1 Detailed Description

template<size_t n> class array_interp< n >

A specialization of o2scl_interp for C-style double arrays.

Definition at line 1384 of file interp.h.

Public Member Functions

- array_interp (base_interp< double[n]> &it, base_interp< array_const_reverse< n > > &rit)
  Create with base interpolation objects `it` *and* `rit`*.*
- array_interp (base_interp< double[n]> &it)
  Create with base interpolation object `it` *and use the default base interpolation object for reversed arrays.*
- array_interp ()
  Create an interpolator using the default base interpolation objects.

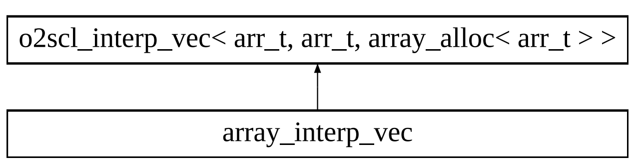
The documentation for this class was generated from the following file:

- interp.h

7.12 array_interp_vec Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for array_interp_vec::



7.12.1 Detailed Description

template<class arr_t> class array_interp_vec< arr_t >

A specialization of o2scl_interp_vec for C-style arrays.

Definition at line 1414 of file interp.h.

Public Member Functions

- array_interp_vec (base_interp< 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

7.13 array_reverse Class Template Reference

`#include <array.h>`

7.13.1 Detailed Description

template<size_t sz> class array_reverse< sz >

A simple class which reverses the order of an array.

Definition at line 151 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

7.14 array_row Class Template Reference

`#include <array.h>`

7.14.1 Detailed Description

template<class data_t, class array_2d_t> class array_row< data_t, array_2d_t >

Extract a row of a C-style 2d-array.

Definition at line 459 of file array.h.

Public Member Functions

- array_row (array_2d_t &a, size_t i)
  View the `i`*th row of the 2-d array* `a`*.*
- data_t & operator[ ] (size_t i)
  The element in the `i`*th column of the chosen row.*

Protected Attributes

- data_t ∗ p
  The pointer.

The documentation for this class was generated from the following file:

- array.h

7.15 array_subvector Class Reference

`#include <array.h>`

7.15.1 Detailed Description

A simple subvector class for an array (without error checking).

Definition at line 218 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

7.16 array_subvector_reverse Class Reference

`#include <array.h>`

7.16.1 Detailed Description

Reverse a subvector of an array.

Definition at line 378 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.

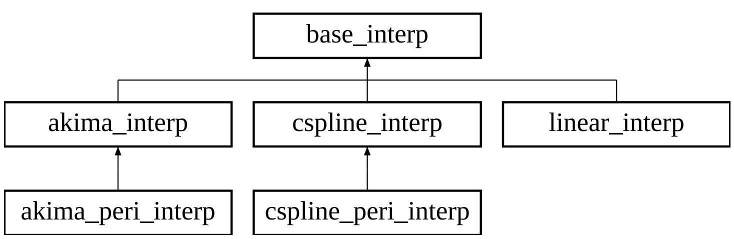
The documentation for this class was generated from the following file:

- array.h

7.17 base_interp Class Template Reference

`#include <interp.h>`

Inheritance diagram for base_interp::



7.17.1 Detailed Description

template<class vec_t> class base_interp< vec_t >

Base low-level interpolation class.

The descendants of this class are intended to be fast interpolation routines for increasing functions, leaving the some error handling, user-friendliness, and other more complicated improvements for other classes.

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. If the vectors do not change, then you may call the interpolation functions in succession.

All of the descendants are based on the GSL interpolation routines and give identical results.

Idea for future

These might work for decreasing functions by just replacing calls to search_vec::bsearch_inc() with search_vec::bsearch_dec(). If this is the case, then this should be rewritten accordingly. (I think I might have removed the acceleration)

Definition at line 66 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)
  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)
  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 $y''(x = x_0)$ *.*
- virtual int integ (const vec_t &x, const vec_t &y, size_t size, double a, double b, double &result)
  Give the value of the integral $\int_a^b y(x)\, dx$ *.*

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
  The binary search object.

7.17.2 Field Documentation

7.17.2.1 size_t min_size

The minimum size of the vectors to interpolate between.

This needs to be set in the constructor of the children for access by the class user

Definition at line 103 of file interp.h.

The documentation for this class was generated from the following file:

- interp.h

7.18 base_ioc Class Reference

```
#include <base_ioc.h>
```

7.18.1 Detailed Description

Setup I/O objects for base library classes.

Definition at line 41 of file base_ioc.h.

Data Fields

- bool_io_type ∗ **bool_io**
- char_io_type ∗ **char_io**
- double_io_type ∗ **double_io**
- int_io_type ∗ **int_io**
- long_io_type ∗ **long_io**
- string_io_type ∗ **string_io**
- word_io_type ∗ **word_io**
- table_io_type ∗ **table_io**

The documentation for this class was generated from the following file:

- base_ioc.h

7.19 bin_size Class Reference

```
#include <bin_size.h>
```

7.19.1 Detailed Description

Determine bin size (CERNLIB).

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.

Todo

Not working yet.

Definition at line 47 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)

7.19.2 Member Function Documentation

7.19.2.1 int calc_bin (double *al*, double *ah*, int *na*, double & *bl*, double & *bh*, int & *nb*, double & *bwid*)

Compute bin size.

- `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`.

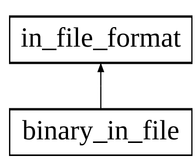
The documentation for this class was generated from the following file:

- bin_size.h

7.20 binary_in_file Class Reference

```
#include <binary_file.h>
```

Inheritance diagram for binary_in_file::



7.20.1 Detailed Description

Binary input file.

Definition at line 137 of file binary_file.h.

Public Member Functions

- binary_in_file (std::string file_name)
  Read an input file with name `file_name`.
- virtual int bool_in (bool &dat, std::string name="")

Input a bool variable.
- virtual int char_in (char &dat, std::string name="")
  Input a char variable.
- virtual int double_in (double &dat, std::string name="")
  Input a double variable.
- virtual int float_in (float &dat, std::string name="")
  Input a float variable.
- virtual int int_in (int &dat, std::string name="")
  Input an int variable.
- virtual int long_in (unsigned long int &dat, std::string name="")
  Input an long variable.
- virtual int string_in (std::string &dat, std::string name="")
  Input a string variable.
- virtual int word_in (std::string &dat, std::string name="")
  Input a word variable.
- virtual int init_file ()
  Read the initialization.
- virtual int clean_up ()
  Clean up the file.
- virtual int start_object (std::string &type, std::string &name)
  Begin reading an object.
- virtual int skip_object ()
  Skip the present object for the next call to read_type().
- virtual int end_object ()
  Finish reading an object.

Protected Attributes

- std::ifstream ins
  The input stream.

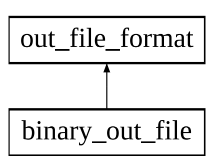
The documentation for this class was generated from the following file:

- binary_file.h

7.21 binary_out_file Class Reference

```
#include <binary_file.h>
```

Inheritance diagram for binary_out_file::



7.21.1 Detailed Description

Binary output file.

Definition at line 41 of file binary_file.h.

Public Member Functions

- binary_out_file (std::string file_name)
  Create a binary output file with name `file_name`.
- virtual int bool_out (bool dat, std::string name="")
  Output a bool variable.
- virtual int char_out (char dat, std::string name="")
  Output a char variable.
- virtual int double_out (double dat, std::string name="")
  Output a double variable.
- virtual int float_out (float dat, std::string name="")
  Output a float variable.
- virtual int int_out (int dat, std::string name="")
  Output an int variable.
- virtual int long_out (unsigned long int dat, std::string name="")
  Output an long variable.
- virtual int string_out (std::string dat, std::string name="")
  Output a string.
- virtual int word_out (std::string dat, std::string name="")
  Output a word.
- virtual int start_object (std::string type, std::string name="")
  Start an object.
- virtual int end_object ()
  End object output (does nothing for a binary file).
- virtual int end_line ()
  End a line of output (does nothing for a binary file).
- virtual int init_file ()
  Output initialization.
- virtual int clean_up ()
  Finish the file.

Protected Attributes

- bool compressed
  True if the file is to be compressed.
- bool gzip
  True if the compression is to be performed by gzip.
- std::ofstream outs
  The output stream.
- std::string user_filename
  The filename specified by the user.
- std::string temp_filename
  The temporary filename.

The output format

- int **fill**
- int **precision**

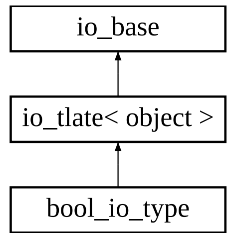
The documentation for this class was generated from the following file:

- binary_file.h

7.22 bool_io_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for bool_io_type::



7.22.1 Detailed Description

I/O object for bool variables.

Definition at line 1639 of file collection.h.

Public Member Functions

- bool_io_type (const char ∗t)
  Desc.
- int addb (collection &co, std::string name, bool x, bool overwrt=true)
  Add a bool to a collection.
- bool getb (collection &co, std::string tname)
  Get a bool from a collection.
- int get_def (collection &co, std::string tname, bool &op, bool def=false)
  Get a bool from a collection.

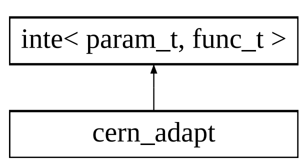
The documentation for this class was generated from the following file:

- collection.h

7.23 cern_adapt Class Template Reference

```
#include <cern_adapt.h>
```

Inheritance diagram for cern_adapt::



7.23.1 Detailed Description

template<class param_t, class func_t = funct<param_t>, size_t nsub = 100> class cern_adapt< param_t, func_t, nsub >

Adaptive integration (CERNLIB).

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 stops if the absolute tolerance is less than tolx, the relative tolerance is less than tolf, or the number of segments exceeds the template parameter `nsub` (in which case the error handler is called, since the integration may not have been successful). The number of segments used in the last integration can be obtained from get_nsegments().

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 segments?

Definition at line 60 of file cern_adapt.h.

Public Member Functions

- int set_inte (inte< param_t, func_t > &i)
  Set the base integration object to use.
- size_t get_nsegments ()
  Return the number of segments used in the last integration.
- int get_ith_segment (size_t i, double &xlow, double &xhigh, double &value, double &errsq)
  Return the ith segment.
- template<class vec_t>
  int get_segments (vec_t &xlow, vec_t &xhigh, vec_t &value, vec_t &errsq)
  Return all of the segments.
- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` from `a` to `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err)
  Integrate function `func` from `a` to `b` giving result `res` and error `err`.

Data Fields

- size_t nseg
  Number of subdivisions.

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 nter
  Previous number of subdivisions.
- cern_gauss56< param_t, func_t > cg56
  Default integration object.
- inte< param_t, func_t > ∗ it
  The base integration object.

7.23.2 Field Documentation

7.23.2.1 size_t nseg

Number of subdivisions.

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 segments, then adapt until tolerance is obtained.

Definition at line 113 of file cern_adapt.h.

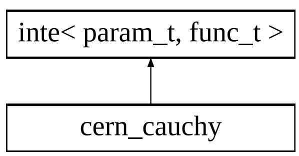
The documentation for this class was generated from the following file:

- cern_adapt.h

7.24 cern_cauchy Class Template Reference

```
#include <cern_cauchy.h>
```

Inheritance diagram for cern_cauchy::



7.24.1 Detailed Description

template<class param_t, class func_t> class cern_cauchy< param_t, func_t >

Cauchy principal value integration (CERNLIB).

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 cern_gauss.

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

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< param_t, func_t > &i)
  Set the base integration object to use.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, 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, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.

Data Fields

- double s
  The singularity (must be set before calling integ() or integ_err()).
- cern_gauss< param_t, func_t > def_inte
  Default integration object.

Protected Attributes

- inte< param_t, func_t > * it
  The base integration object.

Integration constants

- double **x** [12]
- double **w** [12]

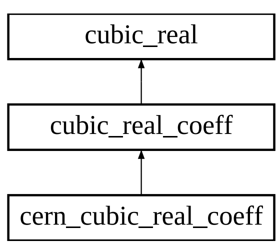
The documentation for this class was generated from the following file:

- cern_cauchy.h

7.25 cern_cubic_real_coeff Class Reference

`#include <poly.h>`

Inheritance diagram for cern_cubic_real_coeff::



7.25.1 Detailed Description

Solve a cubic with real coefficients and complex roots (CERNLIB).

Definition at line 389 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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *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 original CERNLIB interface.
- const char ∗ type ()
  Return a string denoting the type ("cern_cubic_real_coeff").

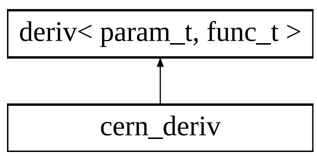
The documentation for this class was generated from the following file:

- poly.h

7.26 cern_deriv Class Template Reference

```
#include <cern_deriv.h>
```

Inheritance diagram for cern_deriv::



7.26.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class cern_deriv< param_t, func_t >

Numerical differentiation routine (CERNLIB).

This uses Romberg extrapolation to compute the derivative with the finite-differencing formula

$$f'(x) = [f(x+h) - f(x-h)]/(2h)$$

If root::verbose is greater than zero, then each iteration prints out the extrapolation table, and if root::verbose is greater than 1, then a keypress is required at the end of each iteration.

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

Note:

Second and third derivatives are computed by naive nested applications of the formula for the first derivative and the uncertainty for these will likely be underestimated.

Definition at line 62 of file cern_deriv.h.

Public Member Functions

- virtual int calc_err (double x, param_t &pa, 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

- virtual int calc_err_int (double x, typename deriv< param_t, func_t >::dpars &pa, funct< typename deriv< param_t, func_t >::dpars > &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]

7.26.2 Member Function Documentation

7.26.2.1 virtual int calc_err_int (double *x*, typename deriv< param_t, func_t >::dpars & *pa*, funct< typename deriv< param_t, func_t >::dpars > & *func*, double & *dfdx*, double & *err*) `[inline, protected, virtual]`

Calculate the first derivative of `func` w.r.t. x.

This is an internal version of calc() which is used in computing second and third derivatives

Definition at line 217 of file cern_deriv.h.

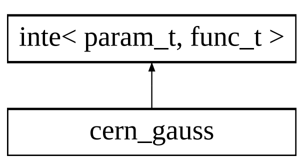
The documentation for this class was generated from the following file:

- cern_deriv.h

7.27 cern_gauss Class Template Reference

#include <cern_gauss.h>

Inheritance diagram for cern_gauss::



7.27.1 Detailed Description

template<class param_t, class func_t> class cern_gauss< param_t, func_t >

Gaussian quadrature (CERNLIB).

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 $1, \frac{1}{2}, \frac{1}{4}, ...$ 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::tolf 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

$$\frac{|G - I|}{I_{abs} + k} < \text{tolf}$$

will nearly always be true when no error is returned. For functions with no singularities in the interval, the accuracy will usually be higher than this.

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 89 of file cern_gauss.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, param_t &pa, 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, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.

Protected Attributes

Integration constants

- double **x** [12]
- double **w** [12]

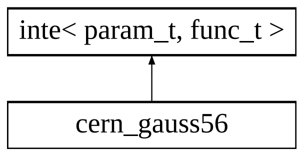
The documentation for this class was generated from the following file:

- cern_gauss.h

7.28 cern_gauss56 Class Template Reference

```
#include <cern_gauss56.h>
```

Inheritance diagram for cern_gauss56::



7.28.1 Detailed Description

template<class param_t, class func_t> class cern_gauss56< param_t, func_t >

5,6-point Gaussian quadrature (CERNLIB)

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 45 of file cern_gauss56.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, 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]

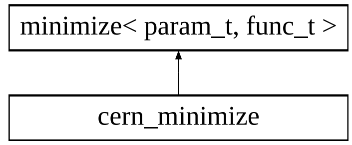
The documentation for this class was generated from the following file:

- cern_gauss56.h

7.29 cern_minimize Class Template Reference

```
#include <cern_minimize.h>
```

Inheritance diagram for cern_minimize::



7.29.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class cern_minimize< param_t, func_t >

One-dimensional minimization (CERNLIB).

The golden section search is applied in the interval (a, b) using a fixed number n of function evaluations where

$$n = \left[2.08 \ln(|a - b|/\mathrm{tolx}) + \frac{1}{2} \right] + 1$$

The accuracy depends on the function. A choice of $\mathrm{tolx} > 10^{-8}$ usually results in a relative error of x which is smaller than or of the order of tolx .

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:

The number of function evaluations can be quite large if multi_min::tolx is sufficiently small. If multi_min::tolx is exactly zero, then 10^{-8} will be used instead.

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 59 of file cern_minimize.h.

Public Member Functions

- virtual int min_bkt (double &x, double a, double b, double &y, param_t &pa, 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.

7.29.2 Member Function Documentation

7.29.2.1 virtual int min_bkt (double & *x*, double *a*, double *b*, double & *y*, param_t & *pa*, func_t & *func*) `[inline, virtual]`

Calculate the minimum `min` of `func` between `a` and `b`.

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< param_t, func_t >.

Definition at line 94 of file cern_minimize.h.

7.29.2.2 int set_delta (double *d*) `[inline]`

Set the value of δ.

If this is not called before min_bkt() is used, then the suggested value $\delta = 10\mathrm{tolx}$ is used.

Definition at line 170 of file cern_minimize.h.

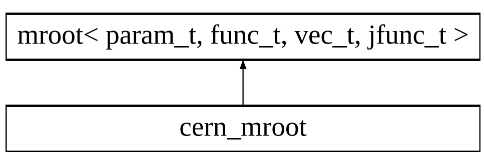
The documentation for this class was generated from the following file:

- cern_minimize.h

7.30 cern_mroot Class Template Reference

```
#include <cern_mroot.h>
```

Inheritance diagram for cern_mroot::



7.30.1 Detailed Description

template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class jfunc_t = jac_funct<param_t,vec_t,omatrix_view>> class cern_mroot< param_t, func_t, vec_t, alloc_vec_t, alloc_t, jfunc_t >

Multi-dimensional mroot-finding routine (CERNLIB).

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

$$1: \quad \max|f_i(x)| \leq \mathrm{tolf}$$

$$2: \quad \max|x_i - x'_i| \leq \text{tolx} \times \max|x_i|$$

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

$$\begin{aligned} 0 &= f_0(x_0, x_1, ..., x_{n-1}) \\ 0 &= f_1(x_0, x_1, ..., x_{n-1}) \\ &... \\ 0 &= f_{n-1}(x_0, x_1, ..., x_{n-1}) \end{aligned}$$

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 Multidimensional solver example .

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

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 86 of file cern_mroot.h.

Public Member Functions

- int get_info ()
  Get the value of `INFO` *from the last call to msolve().*
- virtual const char * type ()
  Return the type, `"cern_mroot"`*.*
- virtual int msolve (size_t nvar, vec_t &x, param_t &pa, 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`.
- int info
  Internal storage for the value of `info`.
- int mpt [289]
  Store the number of function evaluations.

7.30.2 Member Function Documentation

7.30.2.1 int get_info () `[inline]`

Get the value of `INFO` from the last call to msolve().

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::tolx is too small or the Jacobian is nearly singular or the variables are badly scaled.

9 - Either root::tolf or root::tolx is not greater than zero or the specified number of variables is ≤ 0.

Definition at line 143 of file cern_mroot.h.

7.30.3 Field Documentation

7.30.3.1 int maxf

Maximum number of function evaluations.

If $\mathrm{maxf} \leq 0$, 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 152 of file cern_mroot.h.

7.30.3.2 double eps

The smallest floating point number (default $\sim 1.49012 \times 10^{-8}$).

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 181 of file cern_mroot.h.

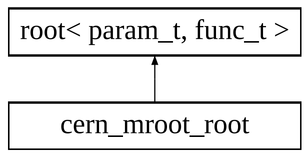
The documentation for this class was generated from the following file:

- cern_mroot.h

7.31 cern_mroot_root Class Template Reference

```
#include <cern_mroot_root.h>
```

Inheritance diagram for cern_mroot_root::



7.31.1 Detailed Description

template<class param_t = void ∗, class func_t = funct<param_t>> class cern_mroot_root< param_t, func_t >

One-dimensional version of cern_mroot.

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

$$1: \quad \max|f_i(x)| \le \mathrm{tolf}$$
$$2: \quad \max|x_i - x'_i| \le \mathrm{tolx} \times \max|x_i|$$

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 63 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, param_t &pa, 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`.

7.31.2 Member Function Documentation

7.31.2.1 int get_info () `[inline]`

Get the value of `INFO` from the last call to solve() (default 0).

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::tolx is too small or the Jacobian is nearly singular or the variables are badly scaled.

9 - Either root::tolf or root::tolx is not greater than zero.

Definition at line 96 of file cern_mroot_root.h.

7.31.3 Field Documentation

7.31.3.1 int maxf

Maximum number of function evaluations.

If $\mathrm{maxf} \leq 0$, 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 105 of file cern_mroot_root.h.

7.31.3.2 double eps

The smallest floating point number (default $\sim 1.49012 \times 10^{-8}$).

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 134 of file cern_mroot_root.h.

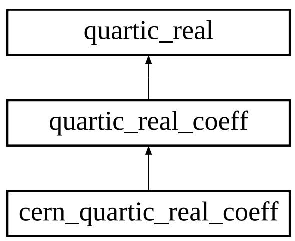
The documentation for this class was generated from the following file:

- cern_mroot_root.h

7.32 cern_quartic_real_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for cern_quartic_real_coeff::



7.32.1 Detailed Description

Solve a quartic with real coefficients and complex roots (CERNLIB).

Definition at line 409 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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 original CERNLIB 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.

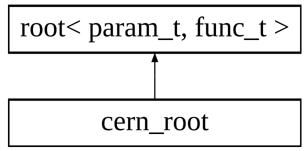
The documentation for this class was generated from the following file:

- poly.h

7.33 cern_root Class Template Reference

```
#include <cern_root.h>
```

Inheritance diagram for cern_root::



7.33.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class cern_root< param_t, func_t >

One-dimensional root-finding routine (CERNLIB).

This class attempts to find x_0 and x_1 in $[a, b]$ such that $f(x_0)f(x_1) \leq 0$, $|f(x_0)| \leq |f(x_1)|$, and $|x_0 - x_1| \leq 2\,\mathrm{tolx}\,(1 + |x_0|)$.

The variable cern_root::tolx defaults to 10^{-8} and cern_root::ntrial defaults to 200.

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, param_t &pa, 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.

7.33.2 Member Function Documentation

7.33.2.1 int set_mode (int *m*) [inline]

Set mode of solution (1 or 2).

- 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, 1 is assumed.

Definition at line 108 of file cern_root.h.

7.33.3 Field Documentation

7.33.3.1 int mode [protected]

Internal storage for the mode.

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 73 of file cern_root.h.

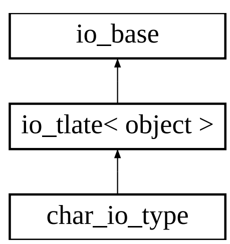
The documentation for this class was generated from the following file:

- cern_root.h

7.34 char_io_type Class Reference

`#include <collection.h>`

Inheritance diagram for char_io_type::



7.34.1 Detailed Description

I/O object for char variables.

Definition at line 1671 of file collection.h.

Public Member Functions

- char_io_type (const char ∗t)
  Desc.
- int addc (collection &co, std::string name, char x, bool overwrt=true)
  Add a char to a collection.
- char getcc (collection &co, std::string tname)
  Get a char from a collection.
- int get_def (collection &co, std::string tname, char &op, char def='x')
  Get a char from a collection.

7.34.2 Member Function Documentation

7.34.2.1 char getcc (collection & *co*, std::string *tname*)

Get a char from a collection.

Some older systems have trouble with functions named `getc`, so this is named `getcc` instead.

The documentation for this class was generated from the following file:

- collection.h

7.35 cinput Class Reference

`#include <collection.h>`

7.35.1 Detailed Description

Class to control object input.

Definition at line 854 of file collection.h.

Public Member Functions

- int object_in (std::string type, in_file_format ∗ins, void ∗vp, std::string &name)
  Input an object.
- int object_in (std::string type, in_file_format ∗ins, void ∗vp, int sz, std::string &name)
  Input an array of objects.
- int object_in (std::string type, in_file_format ∗ins, void ∗vp, int sz, int sz2, std::string &name)
  Input a 2-d array of objects.
- int object_in_mem (std::string type, in_file_format ∗ins, void ∗&vp, std::string &name)
  Input an object, allocating memory first.
- int object_in_mem (std::string type, in_file_format ∗ins, void ∗&vp, int &sz, std::string &name)
  Input an array of objects, allocating memory first.
- int object_in_mem (std::string type, in_file_format ∗ins, void ∗&vp, int &sz, int &sz2, std::string &name)
  Input a 2-d array of objects, allocating memory first.

Protected Types

- typedef std::vector< pointer_input >::iterator ipiter
  An iterator for the input pointers.

Protected Member Functions

- cinput (collection ∗co)
  Create a new input object for a collection.
- int assign_pointers (collection ∗co)
  Assign all of the pointers read with the appropriate objects.

Protected Attributes

- std::vector< pointer_input > input_ptrs
  The pointers that need to be set.
- collection ∗ cop

The pointer to the collection stored in the constructor.

The documentation for this class was generated from the following file:

- collection.h

7.36 cli Class Reference

`#include <cli.h>`

7.36.1 Detailed Description

Configurable command-line interface.

Somewhat 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.

Definition at line 207 of file cli.h.

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)
  Desc.
- int call_args (std::vector< cmd_line_arg > &ca)
  Call functions corresponding to command-line args.
- int process_args (int argv, const char ∗argc[ ], std::vector< cmd_line_arg > &ca, int debug=0)
  Process command-line arguments.
- int process_args (std::string s, std::vector< cmd_line_arg > &ca, int debug=0)
  Process command-line arguments.
- int set_verbose (int v)
  Set verbosity.
- int run_interactive ()
  Run the interactive mode.
- int set_comm_option (comm_option &ic)
  Add a new command.
- int set_parameters (collection &co)
  Create a new command.
- int set_param_help (std::string param, std::string help)
  Set one-line help text for a parameter named `param`.
- int set_alias (std::string alias, std::string str)
  Set an alias `alias` for the string `str`.

Data Fields

- bool gnu_intro
  If true, output the usual GNU intro when run_interactive() is called.
- bool sync_verbose
  If true, then sync 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 hard-coded command objects

- comm_option **c_help**
- comm_option **c_quit**
- comm_option **c_exit**
- comm_option **c_license**
- comm_option **c_warranty**
- comm_option **c_set**
- comm_option **c_get**
- comm_option **c_run**
- comm_option **c_no_intro**
- comm_option **c_alias**

Protected Member Functions

- int apply_alias (std::string &s, std::string sold, std::string snew)
  Replace all occurences of `sold` *with* `snew` *in* `s`*.*
- int separate (std::string str, std::vector< std::string > &sv)
  Separate a string into words.
- bool string_equal (std::string s1, std::string s2)
  Compare two strings, treating dashes as underscores.

The hard-coded command functions

- int **comm_option_run** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_get** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_set** (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_warranty** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_no_intro** (std::vector< std::string > &sv, bool itive_com)
- int **comm_option_alias** (std::vector< std::string > &sv, bool itive_com)

Protected Attributes

- int verbose
  Control screen output.
- collection ∗ cop
  Pointer to collection for parameters.
- 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 ∗ > clist
  List of commands.

Help for parameters

- std::vector< std::string > **ph_name**
- std::vector< std::string > **ph_desc**

Aliases

- std::vector< std::string > **al1**
- std::vector< std::string > **al2**

7.36.2 Member Function Documentation

7.36.2.1 int set_verbose (int *v*)

Set verbosity.

Most errors are output to the screen even if verbose is zero.

7.36.2.2 int set_comm_option (comm_option & *ic*)

Add a new command.

Each command/option must have either a short form in comm_option::shrt or a long from in comm_option::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.

7.36.2.3 int set_parameters (collection & *co*)

Create a new command.

Set the parameters with a collection

7.36.2.4 int set_alias (std::string *alias*, std::string *str*) `[inline]`

Set an alias `alias` for the string `str`.

Aliases can also be set using the command `'alias'`, but that version allows only one-word aliases.

Definition at line 383 of file cli.h.

7.36.3 Field Documentation

7.36.3.1 bool gnu_intro

If true, output the usual GNU intro when run_interactive() is called.

In order to conform to GNU standards, this ought not be set to false by default.

Definition at line 273 of file cli.h.

The documentation for this class was generated from the following file:

- cli.h

7.37 cmd_line_arg Struct Reference

```
#include <cli.h>
```

7.37.1 Detailed Description

A command-line argument.

Definition at line 180 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 * cop
  A pointer to the appropriate (0, unless it's an option).

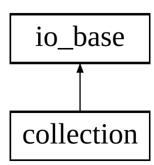
The documentation for this struct was generated from the following file:

- cli.h

7.38 collection Class Reference

```
#include <collection.h>
```

Inheritance diagram for collection::



7.38.1 Detailed Description

Collection of objects.

By default, the fout() functions alphabetize the objects by name, but this is not a requirement for files read using fin().

Important issues: 1. Pointers are not set until after an entire file is read so that objects that are pointed to may occur anywhere in a file. This means that the information that is pointed to cannot be used in the io_tlate_d::input() function.

Todo

- If pointer_in gets a null pointer it does nothing. Should we replace this behaviour by two pointer_in() functions. One which does nothing if it gets a null pointer, and one which will go ahead and set the pointer to null. This is useful for output object which have default values to be used if they are given a null pointer.
- More testing on rewrite() function.
- Think more about adding arrays of pointers? pointers to arrays?
- Modify static data output so that if no objects of a type are included, then no static data is output for that type? (No, it's too hard to go through all objects looking for an object of a particular type).

Bug

- Ensure that the user cannot add a object with a name of ptrXXX.
- Test_type does not test handle static data or pointers.
- Check strings and words for characters that we can't handle
- The present version of a text-file requires strings to contain at least one printable character.
- Ensure that all matching is done by both type and name if possible.

Structure: collection::fout() does the following:

- create an object of type coutput
- Add all objects in the list to the pointer map `ptr_map` (with `output=true`) so that they can be referred to by pointers later
- Output all static data using io_type_info::static_fout()
- Output all of the items in the list `plist` (see below). Any pointers which are not already in `ptr_map` are added at this point (with `output=false`
- Call coutput::pointer_map_fout() to output all objects that were referred to but not in the list

To output individual items, collection::fout() does the following:

- Call either io_base::out_wrapper() or io_base::out_hc_wrapper()
- In turn, these functions call io_base::output(), which the user has overloaded
- If the function io_base::output() calls io_tlate::object_out() then the io_base::output() function appropriate for that object is called. No type or name information is included, but size integers are included if the object is a 1- or 2-d array.
- If the function io_base::output() calls io_base::pointer_out(), then the object is searched for in the `ptr_map` . If it is not there, then the object is added and assigned a name. The type and name are then output. If the pointer is NULL, then both the type and the name are set to `null` .

Definition at line 457 of file collection.h.

Public Member Functions

- int get_type (text_out_file &tof, std::string stype, std::string name)
  Output object of type `stype` *and name* `name` *to output* `tof`*.*
- int get (text_out_file &tof, std::string &stype, std::string name)
  Output object with name `name` *to output* `tof`*.*
- int set (std::string name, text_in_file &tif)
  Set object named `name` *with input from* `tif`*.*
- int set (std::string name, std::string val)
  Set object named `name` *with input from* `val`*.*

Output to file methods

- int fout (out_file_format *outs)
  Output entire list to `outs`*.*
- int fout (std::string filename)
  Output entire list to a textfile named `filename`*.*

Input from file methods

If `overwrt` *is true, then any objects which already exist with the same name are overwritten with the objects in the file. The collection owns all the objects read. (Since it created them, the collection assumes it ought to be responsible to destroy them.)*

- int fin (std::string file_name, bool overwrt=false, int verbose=0)
  Read a collection from text file named `file_name`.
- int fin (in_file_format *ins, bool overwrt=false, int verbose=0)
  Read a collection from `ins`.

Miscellaneous methods

- int test_type (o2scl::test_mgr &t, std::string stype, void *obj, void *&newobj, bool scrout=false)
  Test the output for type `stype`.
- int rewrite (std::string in_name, std::string out_name)
  Update a file containing a collection.
- int disown (std::string name)
  Force the collection to assume that the ownership of `name` *is external.*
- int summary (std::ostream *out, bool show_addresses=false)
  Summarize contents of collection.
- int remove (std::string name)
  Remove an object for the collection.
- void clear ()
  Remove all objects from the list.
- int npa ()
  Count number of objects.

Generic add methods

If `overwrt` *is true, then any objects which already exist with the same name as* `name` *are overwritten. If owner=true, then the collection will own the memory allocated for the object and will free that memory with delete when the object is removed or the collection is deleted.*

- int **add** (std::string name, io_base *tio, void *vec, int sz=0, int sz2=0, bool overwrt=true, bool owner=false)
- int **add** (std::string name, std::string stype, void *vec, int sz=0, int sz2=0, bool overwrt=true, bool owner=false)

Generic get methods

- int get (std::string tname, void *&vec)
  Get an object.
- int get (std::string tname, void *&vec, int &sz)
  Get an array of objects.
- int get (std::string tname, void *&vec, int &sz, int &sz2)
  Get a 2-d array of objects.
- int get (std::string tname, std::string &stype, void *&vec)
  Get an object and its type.
- int get (std::string tname, std::string &stype, void *&vec, int &sz)
  Get an array of objects and their type.
- int get (std::string tname, std::string &stype, void *&vec, int &sz, int &sz2)
  Get a 2-d array of objects and their type.
- void * get (std::string name)
  Get an object (alternative form).

Input and output of individual objects

- int out_one (out_file_format *outs, std::string stype, std::string name, void *vp, int sz=0, int sz2=0)
  Output one object to a file.
- int out_one (std::string fname, std::string stype, std::string name, void *vp, int sz=0, int sz2=0)
  Output one object to a file.
- int in_one_name (in_file_format *ins, std::string stype, std::string name, void *&vp, int &sz, int &sz2)
  Input one object from a file with name `name`.
- int in_one (in_file_format *ins, std::string stype, std::string &name, void *&vp, int &sz, int &sz2)
  Input one object from a file.
- int in_one (std::string fname, std::string stype, std::string &name, void *&vp, int &sz, int &sz2)
  Input one object from a file.

Iterator functions

- iterator begin ()
  Return an iterator to the start of the collection.
- iterator end ()
  Return an iterator to the end of the collection.
- type_iterator begin (std::string utype)
  Return an iterator to the first element of type `utype` *in the collection.*
- type_iterator end (std::string utype)
  Return an iterator to the end of the collection.

Protected Types

- typedef std::map< std::string, collection_entry, string_comp >::iterator piter
  A convenient iterator definition for the collection.

Protected Attributes

- std::map< std::string, collection_entry, string_comp > plist
  The actual collection.

Data Structures

- class iterator
  An iterator for stepping through a collection.
- class type_iterator
  An iterator for stepping through the entries in a collection of a particular type.

7.38.2 Member Function Documentation

7.38.2.1 int rewrite (std::string *in_name*, std::string *out_name*)

Update a file containing a collection.

This method loads the file from "fin" and produces a file at "fout" containing all of the objects from "fin", updated by their new values in the present list if possible. Then, it adds to the end of "fout" any objects in the present list that were not originally contained in "fin".

7.38.2.2 int disown (std::string *name*)

Force the collection to assume that the ownership of `name` is external.

This allows the user to take over ownership of the object named `name`. This is particularly useful if the object is read from a file (since then object is owned initially by the collection), and you want to delete the collection, but retain the object.

7.38.2.3 int remove (std::string *name*)

Remove an object for the collection.

Free the memory `name` if it is owned by the collection and then remove it from the collection.

7.38.2.4 int out_one (out_file_format * *outs*, std::string *stype*, std::string *name*, void * *vp*, int *sz* = `0`, int *sz2* = `0`)

Output one object to a file.

This does not disturb any objects in the collection. The pointer specified does not need to be in the collection and is not added to the collection.

7.38.2.5 int out_one (std::string *fname*, std::string *stype*, std::string *name*, void ∗ *vp*, int *sz* = 0, int *sz2* = 0)

Output one object to a file.

This does not disturb any objects in the collection. The pointer specified does not need to be in the collection and is not added to the collection.

7.38.2.6 int in_one_name (in_file_format ∗ *ins*, std::string *stype*, std::string *name*, void ∗& *vp*, int & *sz*, int & *sz2*)

Input one object from a file with name `name`.

This does not disturb any objects in the collection. The pointer specified does not need to be in the collection and is not added to the collection.

7.38.2.7 int in_one (in_file_format ∗ *ins*, std::string *stype*, std::string & *name*, void ∗& *vp*, int & *sz*, int & *sz2*)

Input one object from a file.

This does not disturb any objects in the collection. The pointer specified does not need to be in the collection and is not added to the collection.

7.38.2.8 int in_one (std::string *fname*, std::string *stype*, std::string & *name*, void ∗& *vp*, int & *sz*, int & *sz2*)

Input one object from a file.

This does not disturb any objects in the collection. The pointer specified does not need to be in the collection and is not added to the collection.

The documentation for this class was generated from the following file:

- collection.h

7.39 collection::iterator Class Reference

```
#include <collection.h>
```

7.39.1 Detailed Description

An iterator for stepping through a collection.

Definition at line 684 of file collection.h.

Public Member Functions

- iterator operator++ ()
  Prefix increment.
- iterator operator++ (int unused)
  Postfix increment.
- iterator operator– ()
  Prefix decrement.
- collection_entry ∗ operator → () const
  Dereference.
- std::string name ()
  Return the name of the collection entry.

Protected Member Functions

- iterator (piter p)
  Create an iterator from the STL iterator.

Protected Attributes

- piter pit
  Local storage for the STL iterator.

Friends

- int operator== (const iterator &i1, const iterator &i2)
  Equality comparison for two iterators.
- int operator!= (const iterator &i1, const iterator &i2)
  Inequality comparison for two iterators.

The documentation for this class was generated from the following file:

- collection.h

7.40 collection::type_iterator Class Reference

`#include <collection.h>`

7.40.1 Detailed Description

An iterator for stepping through the entries in a collection of a particular type.

Definition at line 740 of file collection.h.

Public Member Functions

- type_iterator operator++ ()
  Prefix increment.
- type_iterator operator++ (int unused)
  Postfix increment.
- collection_entry * operator → () const
  Dereference.
- std::string name ()
  Return the name of the collection entry.

Protected Member Functions

- type_iterator (piter p, std::string type, collection *cop)
  Constructor.

Protected Attributes

- std::string ltype
  Local storage for the type.
- collection * lcop

Store a pointer to the collection.
- piter pit
  The STL iterator.

Friends

- int operator== (const type_iterator &i1, const type_iterator &i2)
  Equality comparison for two iterators.
- int operator!= (const type_iterator &i1, const type_iterator &i2)
  Inequality comparison for two iterators.

The documentation for this class was generated from the following file:

- collection.h

7.41 collection_entry Struct Reference

```
#include <collection.h>
```

7.41.1 Detailed Description

An entry in a collection.

Definition at line 52 of file collection.h.

Data Fields

- void ∗ data
  The pointer to the object.
- int size
  The first size parameter.
- int size2
  The second size parameter.
- bool owner
  True if the collection owns this object.
- class io_base ∗ iop
  A pointer to the corresponding io_base object.

The documentation for this struct was generated from the following file:

- collection.h

7.42 columnify Class Reference

```
#include <columnify.h>
```

7.42.1 Detailed Description

Create nicely formatted columns from a table of strings.

This is a brute-force approach of order $\mathrm{ncols} \times \mathrm{nrows}$.

Todo

Move the screenify() functionality from misc.h into this class?

Definition at line 48 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.

7.42.2 Member Function Documentation

7.42.2.1 int align (const mat_string_t & *table*, size_t *ncols*, size_t *nrows*, vec_string_t & *ctable*, vec_int_t & *align_spec*) `[inline]`

Take `table` and create a new object `ctable` with appropriately formatted columns.

The table of strings should be stored in `table` in "column-major" order, 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.

Definition at line 82 of file columnify.h.

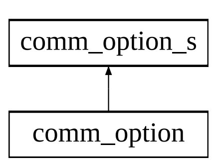
The documentation for this class was generated from the following file:

- columnify.h

7.43 comm_option Class Reference

```
#include <cli.h>
```

Inheritance diagram for comm_option::



7.43.1 Detailed Description

Command for interactive mode in cli.

See the cli class for more details.

Definition at line 140 of file cli.h.

Public Member Functions

- comm_option (comm_option_s c)
  Desc.

Static Public Attributes

Possible values of ::type

- static const int **command** = 0
- static const int **cl_param** = 1
- static const int **both** = 2

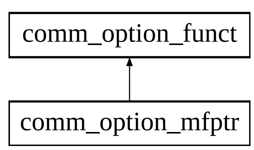
The documentation for this class was generated from the following file:

- cli.h

7.44 comm_option_funct Class Reference

```
#include <cli.h>
```

Inheritance diagram for comm_option_funct::



7.44.1 Detailed Description

Base for cli command function.

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)
  The basic function called by cli.

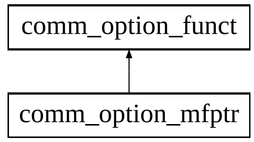
The documentation for this class was generated from the following file:

- cli.h

7.45 comm_option_mfptr Class Template Reference

`#include <cli.h>`

Inheritance diagram for comm_option_mfptr::



7.45.1 Detailed Description

template<class tclass> class comm_option_mfptr< tclass >

Member function pointer for cli command function.

Definition at line 67 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 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.

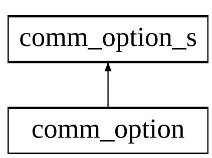
The documentation for this class was generated from the following file:

- cli.h

7.46 comm_option_s Struct Reference

`#include <cli.h>`

Inheritance diagram for comm_option_s::



7.46.1 Detailed Description

Command for interactive mode in cli.

See the cli class for more details.

Definition at line 112 of file cli.h.

Data Fields

- char shrt
  Short option (`'\0'` for none).
- std::string lng
  Long option (must be specified).
- std::string desc
  Description for help (default is empty string).
- 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 (default is empty string).
- std::string help
  The help description (default is empty string).
- 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 (default command).

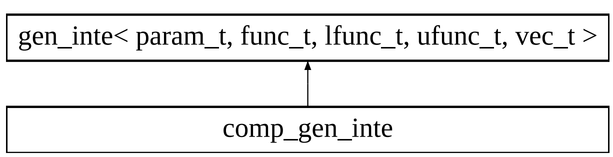
The documentation for this struct was generated from the following file:

- cli.h

7.47 comp_gen_inte Class Template Reference

```
#include <comp_gen_inte.h>
```

Inheritance diagram for comp_gen_inte::



7.47.1 Detailed Description

template<class param_t, class func_t, class lfunc_t = func_t, class ufunc_t = func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class comp_gen_inte< param_t, func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t >

Naive generalized multi-dimensional integration.

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 their specification in the list `inte **ip`. For the example of a two-dimensional integration `ip[0]` is called first with limits `a[0]` and `b[0]` and performs the integral of the integral given by `ip[1]` of the function from `a[1]` to `b[1]`, both of which may depend explicitly on `x[0]`. The integral performed is:

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

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).

Definition at line 69 of file comp_gen_inte.h.

Public Member Functions

- int set_ptrs (inte< od_parms, funct< od_parms > > **ip, int n)
  Designate the pointers to the integration routines.
- virtual double ginteg (func_t &func, size_t n, func_t &lower, func_t &upper, param_t &pa)
  Integrate function `func` *from* $\ell_i = f_i(x_i)$ *to* $u_i = g_i(x_i)$ *for* $0 < i < \mathrm{n} - 1$.
- virtual const char * type ()
  Return string denoting type ("comp_gen_inte").

Protected Member Functions

- double odfunc (double x, od_parms &od)
  The one-dimensional integration function.

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.
- funct_mfptr_noerr< comp_gen_inte< param_t, func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t >, od_parms > * fmn
  The function to send to the integrators.
- size_t ndim
  The number of dimensions.
- inte< od_parms, funct< od_parms > > ** ptrs
  Pointers to the integration objects.
- bool ptrs_set
  True if the integration objects have been specified.

Data Structures

- struct od_parms
  Parameters to send to the 1-d integration functions.

7.47.2 Member Function Documentation

7.47.2.1 int set_ptrs (inte< od_parms, funct< od_parms > > ** *ip*, int *n*) `[inline]`

Designate the pointers to the integration routines.

The user can, in principle, specify the one instance of an inte object for several of the pointers, but this is discouraged as most inte objects cannot be used this way. This function will not warn you if some of the pointers specified in `ip` refer to the same object.

If more 1-d integration routines than necessary are given, the extras will be unused.

Definition at line 125 of file comp_gen_inte.h.

The documentation for this class was generated from the following file:

- comp_gen_inte.h

7.48 comp_gen_inte::od_parms Struct Reference

```
#include <comp_gen_inte.h>
```

7.48.1 Detailed Description

template<class param_t, class func_t, class lfunc_t = func_t, class ufunc_t = func_t, class vec_t = ovector_view, class alloc_-vec_t = ovector, class alloc_t = ovector_alloc> struct comp_gen_inte< param_t, func_t, lfunc_t, ufunc_t, vec_t, alloc_vec_t, alloc_t >::od_parms

Parameters to send to the 1-d integration functions.

For basic usage, the class-user needs this type to specify the parameter type for 1-d integration objects.

Definition at line 96 of file comp_gen_inte.h.

Data Fields

- alloc_vec_t * cx
  The independent variable vector.
- lfunc_t * lower
  The function specifying the lower limits.
- ufunc_t * upper
  The function specifying the upper limits.
- func_t * func
  The function to be integrated.
- int ndim
  The number of dimensions.
- int idim
  The present recursion level.
- param_t * vp
  The user-specified parameter.

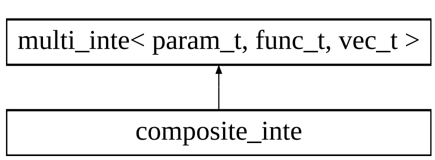
The documentation for this struct was generated from the following file:

- comp_gen_inte.h

7.49 composite_inte Class Template Reference

```
#include <composite_inte.h>
```

Inheritance diagram for composite_inte::



7.49.1 Detailed Description

template<class param_t, class func_t, class vec_t, class alloc_vec_t, class alloc_t> class composite_inte< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Naive multi-dimensional integration over a hypercube.

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 integration routines are specified in the function set_ptrs().

The integration routines are called in order of their specification in the list `inte **ip`. For the example of a two-dimensional integration `ip[0]` is called first with limits `a[0]` and `b[0]` and performs the integral of the integral given by `ip[1]` of the function from `a[1]` to `b[1]`. In other words, the integral performed is:

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

No error estimate is performed. Error estimation for multiple dimension integrals is provided by the Monte Carlo integration classes (see mcarlo_inte).

Todo

Convert to using std::vector<inte> for the 1-d integration pointers?

Definition at line 68 of file composite_inte.h.

Public Member Functions

- int set_ptrs (inte< od_parms, funct< od_parms > > **ip, int n)
  Designate the pointers to the integration routines.
- virtual double minteg (func_t &func, size_t n, const vec_t &a, const vec_t &b, param_t &pa)
  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").

Protected Member Functions

- double odfunc (double x, od_parms &od)
  The one-dimensional integration function.

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`*.*
- funct_mfptr_noerr< composite_inte< param_t, func_t, vec_t, alloc_vec_t, alloc_t >, od_parms > * fmn
  This function to send to the integrators.
- size_t ndim
  The number of dimensions.
- inte< od_parms, funct< od_parms > > ** iptrs
  Pointers to the integration objects.
- bool ptrs_set
  True if the integration objects have been specified.

Data Structures

- struct od_parms
  Parameters to send to the 1-d integration functions.

7.49.2 Member Function Documentation

7.49.2.1 int set_ptrs (inte< od_parms, funct< od_parms > > ** *ip*, int *n*) [inline]

Designate the pointers to the integration routines.

This function allows duplicate objects in this list in order to allow the user to use only one object for more than one of the integrations.

If more 1-d integration routines than necessary are given, the extras will be unused.

Definition at line 120 of file composite_inte.h.

The documentation for this class was generated from the following file:

- composite_inte.h

7.50 composite_inte::od_parms Struct Reference

```
#include <composite_inte.h>
```

7.50.1 Detailed Description

template<class param_t, class func_t, class vec_t, class alloc_vec_t, class alloc_t> struct composite_inte< param_t, func_t, vec_t, alloc_vec_t, alloc_t >::od_parms

Parameters to send to the 1-d integration functions.

This structure is not intended to be frequently used directly by the class-user, but must be public so that the 1-d integration objects can be specified.

Definition at line 80 of file composite_inte.h.

Data Fields

- 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.
- int ndim
  The user-specified number of dimensions.
- int idim
  The present recursion level.
- param_t * vp
  The user-specified parameter.

The documentation for this struct was generated from the following file:

- composite_inte.h

7.51 contour Class Reference

`#include <contour.h>`

7.51.1 Detailed Description

Calculate contour lines from a two-dimensional data set.

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() which returns the number of contours (which is often larger than the number of contour levels, since one level can result in multiple contours).
- Retrieve the contours individually using calls to get_contour().

The data should be stored so that the y-index is first, i.e. data[iy][ix]. One can always switch `x_fun` and `y_fun` if this is not the case. 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.

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 uses cubic spline interpolation to refine the data and thus make the curves more continuous.

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, so if this is the case, then either a more refined data set should be specified or regrid_data() should be used. For contours which do not close inside the region of interest, the results will always end at either the minimum or maximum values of `x_fun` or `y_fun` (no extrapolation is ever done).

As an example, for the function

$$15e^{-(x-20)^2/400-(y-5)^2/25} + 40e^{-(x-70)^2/4900-(y-2)^2/4}$$

a 10x10 grid gives the contours:

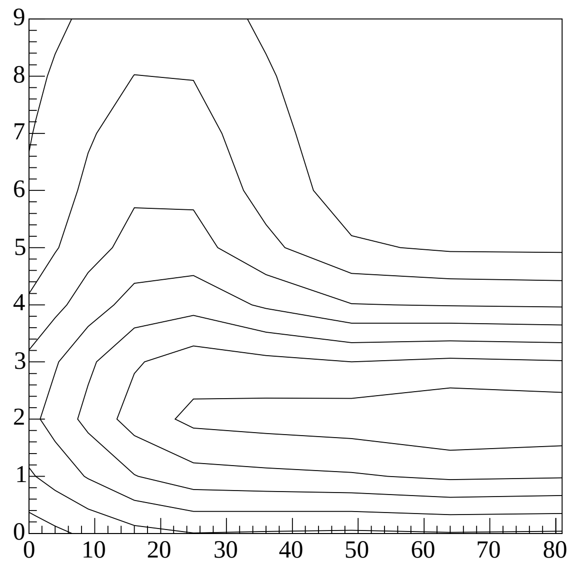


Figure 2: contourg.eps

While after a call to regrid_data(3,3), the contours are a little smoother:

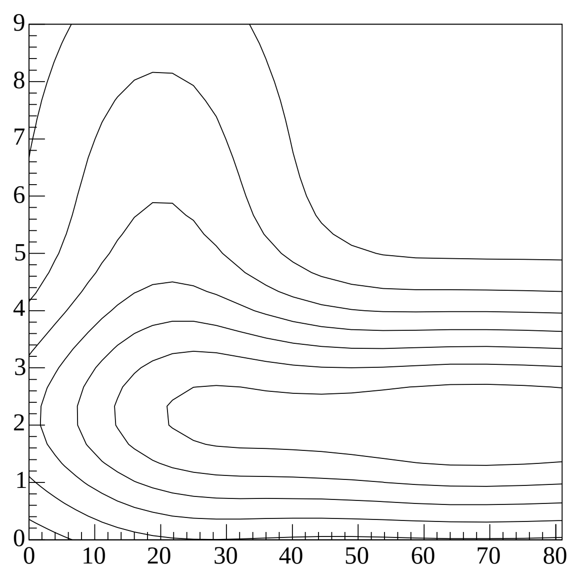


Figure 3: contourg2.eps

Mathematica gives a similar result:

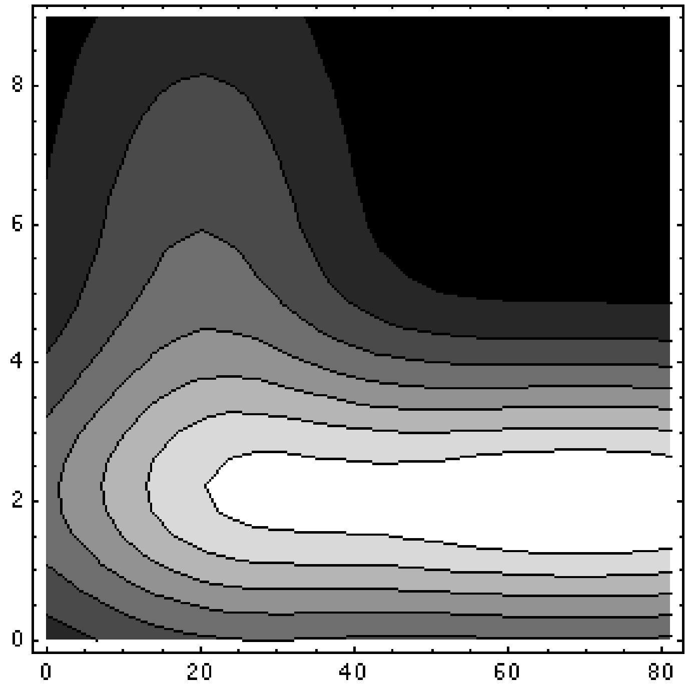


Figure 4: contourg3.eps

The Algorithm:

This works by viewing the data as defining a square two-dimensional grid. The function calc() exhaustively enumerates every line segment in the grid which involves a level crossing and then organizing the points defined by the intersection of a line segment with a level curve into a full contour.

Representing Contours by Fill Regions

If the user wants to "shade" the contours to provide a shaded or colored contour plot, then it is useful to provide a set of closed contours to be shaded instead of the (possibly) open contours returned by calc_contours(). After a call to calc_contours(), the function regions() can be used to organize the closed contours into regions to be shaded. Open contours are closed by adding points defining lines along the edges of the data and closed contours are inverted (if necessary) by adding an external line emanating from the closed contour and properly including the boundary edges.

Todo

- Some contours which should be closed are not properly closed yet. See the tests for examples which fail.
- Use twod_intp for regrid_data
- Include the functionality to provide regions to be shaded instead of just lines. This can be done by providing a method, i.e. regions() which converts the curves into regions.
- Rework documentation to refer to rows and columns, not x and y

Idea for future

- Work on how memory is allocated
- Create a new separate struct for contour curves

Definition at line 130 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.
- template<class vec_t>
  int calc_contours (vec_t &new_levels, bool debug=false)
  Calculate the contours.
- int get_contour_size (int index)
  Return the number of points in the specified contour.
- template<class vec_t>
  int get_contour (int index, double &val, int &csize, vec_t &x, vec_t &y)
  Get a contour.
- int regions (bool larger)
  Compute closed contour regions (unfinished).

Regrid function

- int regrid_data (size_t xfact, size_t yfact)
  Regrid the data.

Obtain internal data

- int get_data (size_t &sizex, size_t &sizey, const ovector ∗&x_fun, const ovector ∗&y_fun, const ovector ∗∗&udata)
  Get the data.
- int get_edges (const std::vector< omatrix_int ∗ > ∗rints, const std::vector< omatrix_int ∗ > ∗bints, const std::vector< omatrix ∗ > ∗rpoints, const std::vector< omatrix ∗ > ∗bpoints)
  Return the edges used to compute the contours.
- int get_edges_for_level (size_t nl, omatrix_int &rints, omatrix_int &bints, omatrix &rpoints, omatrix &bpoints)
  Return the edges used to compute the contours.

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=1)
  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=1)
  Find next point starting from a point on a bottom edge.
- int find_intersections (size_t ilev, double &level)
  Find all of the intersections of the edges with the contour level.
- int right_edges (double level, o2scl::sm_interp ∗si)
  Interpolate all right edge crossings.
- int bottom_edges (double level, o2scl::sm_interp ∗si)
  Interpolate all bottom edge crossings.
- int process_line (int j, int k, int dir, std::vector< double > &x, std::vector< double > &y, bool first=true)
  Create a contour line from a starting edge.
- bool is_point_inside_old (double x1, double y1, const ovector_view &x, const ovector_view &y, double xscale=0.01, double yscale=0.01)
  Test if a point is inside a closed contour (unfinished).
- int free_memory ()
  Free memory.

- bool lines_cross_old (double x1, double y1, double x2, double y2, double x3, double y3, double x4, double y4)
  Check if lines cross.
- int check_data ()
  Check to ensure the x- and y-arrays are monotonic.
- int smooth_contours (size_t nfact)
  Smooth the contours by adding internal points using cubic interpolation (this doesn't work).

Protected Attributes

- int **new_debug**
- pinside pi
  Object to find if a point is inside a polygon.

User-specified data

- int **nx**
- int **ny**
- ovector $*$ **xfun**
- ovector $*$ **yfun**
- ovector $**$ **data**

User-specified contour levels

- int **nlev**
- ovector **levels**
- bool **levels_set**

Generated curves

- int **ncurves**
- std::vector< int > **csizes**
- std::vector< double > **vals**
- std::vector< std::vector< double > > **xc**
- std::vector< std::vector< double > > **yc**

Storage for edges

- std::vector< omatrix $*$ > **redges**
- std::vector< omatrix $*$ > **bedges**
- std::vector< omatrix_int $*$ > **re**
- std::vector< omatrix_int $*$ > **be**
- omatrix_int $*$ **rep**
- omatrix_int $*$ **bep**
- omatrix $*$ **redgesp**
- omatrix $*$ **bedgesp**

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

7.51.2 Member Function Documentation

7.51.2.1 int set_data (size_t *sizex*, size_t *sizey*, const vec_t & *x_fun*, const vec_t & *y_fun*, const mat_t & *udata*) [inline]

Set the data.

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 152 of file contour.h.

7.51.2.2 int set_levels (size_t *nlevels*, vec_t & *ulevels*) [inline]

Set the contour levels.

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 188 of file contour.h.

7.51.2.3 int calc_contours (vec_t & *new_levels*, bool *debug* = `false`) [inline]

Calculate the contours.

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.

Definition at line 210 of file contour.h.

7.51.2.4 int get_contour (int *index*, double & *val*, int & *csize*, vec_t & *x*, vec_t & *y*) [inline]

Get a contour.

Given the `index`, which is between 0 and the number of contours returned by calc_contours() minus 1 (inclusive), this returns the level for this contour with the x and y-values in `x` and `y` which are both of length `csize`. The vectors `x` and `y` must have been previously allocated. The user can obtain the necessary size for `x` and `y` by calling get_contour_size().

Definition at line 389 of file contour.h.

7.51.2.5 int regrid_data (size_t *xfact*, size_t *yfact*)

Regrid the data.

The uses cubic spline interpolation to refine the data set, ideally used before attempting to calculate the contour levels. If the original number of data points is $(\mathrm{nx}, \mathrm{ny})$, then the new number of data points is

$$(\mathrm{xfact}\,(\mathrm{nx}-1)+1, \mathrm{yfact}\,(\mathrm{ny}-1)+1)$$

7.51.2.6 int get_data (size_t & *sizex*, size_t & *sizey*, const ovector $*$& *x_fun*, const ovector $*$& *y_fun*, const ovector $**$& *udata*) [inline]

Get the data.

This is useful to see how the data has changed after a call to regrid_data().

Definition at line 439 of file contour.h.

7.51.2.7 int get_edges_for_level (size_t *nl*, omatrix_int & *rints*, omatrix_int & *bints*, omatrix & *rpoints*, omatrix & *bpoints*)

Return the edges used to compute the contours.

This function allocates memory for `rints`, `bints`, `rpoints`, and `bpoints` and fills them with a copy of the data that the class is using. As such, there is no need for them to be const.

7.51.2.8 bool is_point_inside_old (double *x1*, double *y1*, const ovector_view & *x*, const ovector_view & *y*, double *xscale* = `0.01`, double *yscale* = `0.01`) `[protected]`

Test if a point is inside a closed contour (unfinished).

This returns true if the point (x1,y1) is "inside" the contour (i.e. a collection of line segments) given in `x` and `y`. The arrays `x` and `y` must be "ordered" so that adjacent points are placed at adjacent entries. The result is undefined if this is not the case or if the contours are not properly closed. The first and last points in `x` and `y` should be the same to indicate a closed contour. The values xscale and yscale should be an approximate scale for the contours `x` and `y`.

Note:

This function is deprecated and has been replaced by pinside

7.51.2.9 bool lines_cross_old (double *x1*, double *y1*, double *x2*, double *y2*, double *x3*, double *y3*, double *x4*, double *y4*) `[protected]`

Check if lines cross.

Returns true if the line segment defined by (x1,y1) and (x2,y2) and the line segment defined by (x3,y3) and (x4,y4) have an intersection point that is between the endpoints of both segments. The function handles vertical and horizontal lines appropriately. This function will fail if x1==y1 and x2==y2 or if x3==y3 and x4==y4, i.e. if the points do not really define a line. If the function fails, it returns false.

Note:

This function is deprecated and has been replaced by pinside

7.51.2.10 int smooth_contours (size_t *nfact*) `[protected]`

Smooth the contours by adding internal points using cubic interpolation (this doesn't work).

This makes the contours smoother by adding internal points between each contour line segment determined by cubic spline interpolation.

For more accurate contours, it is better to provide the original data on a finer grid, or use regrid_data().

The documentation for this class was generated from the following file:

- contour.h

7.52 coutput Class Reference

```
#include <collection.h>
```

7.52.1 Detailed Description

Class to control object output.

Definition at line 915 of file collection.h.

Public Member Functions

- int object_out (std::string type, out_file_format ∗outs, void ∗op, int sz=0, int sz2=0, std::string name="")
  Output an object.

Protected Types

- typedef std::map< void ∗, pointer_output, ltptr >::iterator pmiter
  A convenient iterator for the pointer list.

Protected Member Functions

- coutput (class collection ∗co)
  Create a new object from a pointer to a collection.
- int pointer_lookup (void ∗vp, std::string &name, collection_entry ∗&ep)
  Look for an object in the collection given a pointer.
- int pointer_map_fout (out_file_format ∗out)
  Output all of the remaining pointers to 'out'.

Protected Attributes

- std::map< void ∗, pointer_output, ltptr > ptr_map
  The list pointers to object to be written to the file.
- collection ∗ cop
  The pointer to the collection stored in the constructor.
- int npointers
  Keep track of the number of pointers added to ptr_map.

Data Structures

- struct ltptr
  Order the pointers by numeric value.

7.52.2 Member Function Documentation

7.52.2.1 int pointer_lookup (void ∗ *vp*, std::string & *name*, collection_entry ∗& *ep*) `[protected]`

Look for an object in the collection given a pointer.

Lookup the pointer `vp` in the collection, and return its name and collection_entry

7.52.3 Field Documentation

7.52.3.1 int npointers `[protected]`

Keep track of the number of pointers added to ptr_map.

These are counted for the purposes of making a unique name. This is initialized in fout() and incremented in io_base::pointer_out

Definition at line 971 of file collection.h.

The documentation for this class was generated from the following file:

- collection.h

7.53 coutput::ltptr Struct Reference

`#include <collection.h>`

7.53.1 Detailed Description

Order the pointers by numeric value.

Definition at line 937 of file collection.h.

Public Member Functions

- bool operator() (const void $*$p1, const void $*$p2) const
  Returns $p_1 < p_2$.

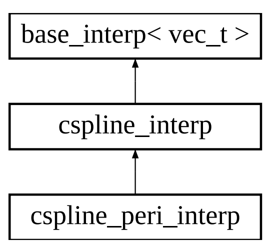
The documentation for this struct was generated from the following file:

- collection.h

7.54 cspline_interp Class Template Reference

`#include <interp.h>`

Inheritance diagram for cspline_interp::



7.54.1 Detailed Description

template<class vec_t> class cspline_interp< vec_t >

Cubic spline interpolation (GSL).

Definition at line 270 of file interp.h.

Public Member Functions

- cspline_interp (bool periodic=false)
  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 $\int_a^b y(x)\, dx$ *.*

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

- bool peri
  True for periodic boundary conditions.

Storage for cubic spline interpolation

- double ∗ **c**
- double ∗ **g**
- double ∗ **diag**
- double ∗ **offdiag**

7.54.2 Member Function Documentation

7.54.2.1 virtual int init (const vec_t & *xa*, const vec_t & *ya*, size_t *size*) `[inline, virtual]`

Initialize interpolation routine.

Periodic boundary conditions

Natural boundary conditions

Reimplemented from base_interp.

Definition at line 351 of file interp.h.

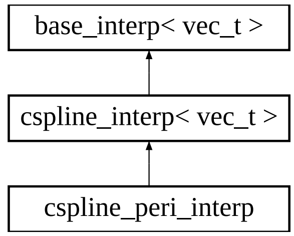
The documentation for this class was generated from the following file:

- interp.h

7.55 cspline_peri_interp Class Template Reference

`#include <interp.h>`

Inheritance diagram for cspline_peri_interp::



7.55.1 Detailed Description

template<class vec_t> class cspline_peri_interp< vec_t >

Cubic spline interpolation with periodic boundary conditions (GSL).

This is convenient to allow interpolation objects to be supplied as template parameters

Definition at line 645 of file interp.h.

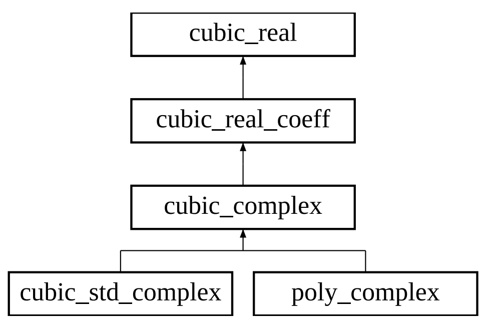
The documentation for this class was generated from the following file:

- interp.h

7.56 cubic_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for cubic_complex::



7.56.1 Detailed Description

Solve a cubic polynomial with complex coefficients and complex roots [abstract base].

Definition at line 190 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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *giving the real solution* $x = x_1$ *and two complex solutions* $x = x_1$, $x = x_2$, *and* $x = x_3$.

- virtual int 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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *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").

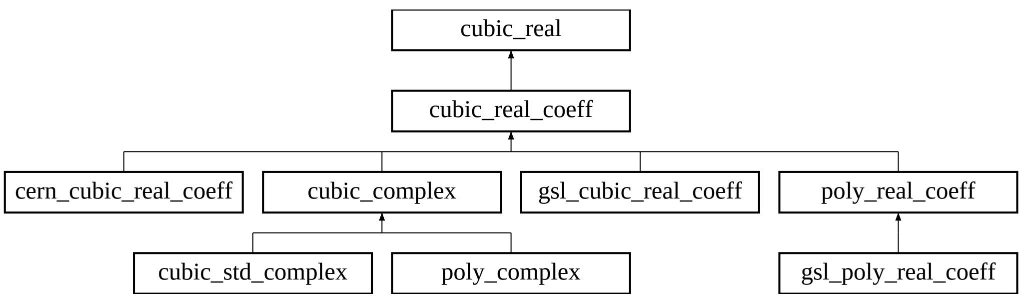
The documentation for this class was generated from the following file:

- poly.h

7.57 cubic_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for cubic_real::



7.57.1 Detailed Description

Solve a cubic polynomial with real coefficients and real roots [abstract base].

Definition at line 139 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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *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").

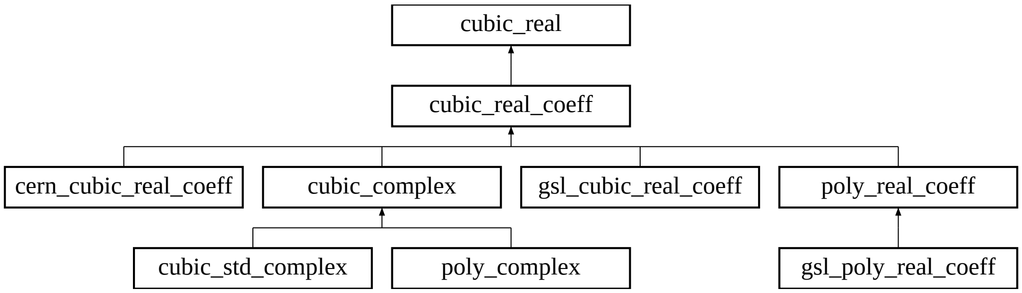
The documentation for this class was generated from the following file:

- poly.h

7.58 cubic_real_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for cubic_real_coeff::



7.58.1 Detailed Description

Solve a cubic polynomial with real coefficients and complex roots [abstract base].

Definition at line 160 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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *giving the real solution* $x = x_1$ *and two complex solutions* $x = x_1$ *,* $x = x_2$ *, and* $x = x_3$ *.*
- const char $*$ type ()
  Return a string denoting the type ("cubic_real_coeff").

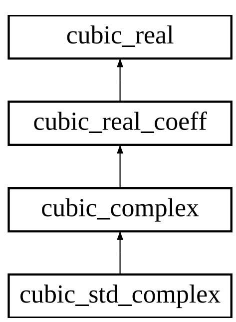
The documentation for this class was generated from the following file:

- poly.h

7.59 cubic_std_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for cubic_std_complex::



7.59.1 Detailed Description

Solve a cubic with complex coefficients and complex roots.

Definition at line 602 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 $a_3 x^3 + b_3 x^2 + c_3 x + d_3 = 0$ *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").

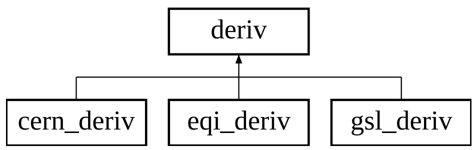
The documentation for this class was generated from the following file:

- poly.h

7.60 deriv Class Template Reference

```
#include <deriv.h>
```

Inheritance diagram for deriv::



7.60.1 Detailed Description

template<class param_t, class func_t> class deriv< param_t, func_t >

Numerical differentiation base.

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

Idea for future

This does not have pure virtual functions, but I'd still like to prevent the user from directly instantiating a deriv object.

Definition at line 57 of file deriv.h.

Public Member Functions

- virtual double calc (double x, param_t &pa, func_t &func)
  Calculate the first derivative of `func` *w.r.t. x.*

- virtual double calc2 (double x, param_t &pa, func_t &func)
  Calculate the second derivative of `func` *w.r.t. x.*
- virtual double calc3 (double x, param_t &pa, 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, param_t &pa, func_t &func, double &dfdx, double &err)
  Calculate the first derivative of `func` *w.r.t. x and the uncertainty.*
- virtual int calc2_err (double x, param_t &pa, 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, param_t &pa, 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

- int verbose
  Output control.

Protected Member Functions

- virtual double calc_int (double x, dpars &pa, o2scl::funct< dpars > &func)
  Calculate the first derivative of `func` *w.r.t. x.*
- virtual int calc_err_int (double x, dpars &pa, o2scl::funct< dpars > &func, double &dfdx, double &err)
  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.

Data Structures

- struct dpars
  A structure for passing the function to second and third derivatives.

7.60.2 Member Function Documentation

7.60.2.1 virtual double calc (double *x*, param_t & *pa*, func_t & *func*) `[inline, virtual]`

Calculate the first derivative of `func` w.r.t. x.

After calling calc(), the error may be obtained from get_err().

Definition at line 92 of file deriv.h.

7.60.2.2 virtual double calc_int (double *x*, dpars & *pa*, o2scl::funct< dpars > & *func*) [inline, protected, virtual]

Calculate the first derivative of `func` w.r.t. x.

This is an internal version of calc() which is used in computing second and third derivatives

Definition at line 180 of file deriv.h.

7.60.2.3 virtual int calc_err_int (double *x*, dpars & *pa*, o2scl::funct< dpars > & *func*, double & *dfdx*, double & *err*) [inline, protected, virtual]

Calculate the first derivative of `func` w.r.t. x and the uncertainty.

This is an internal version of calc_err() which is used in computing second and third derivatives

Definition at line 194 of file deriv.h.

The documentation for this class was generated from the following file:

- deriv.h

7.61 deriv::dpars Struct Reference

```
#include <deriv.h>
```

7.61.1 Detailed Description

template<class param_t, class func_t> struct deriv< param_t, func_t >::dpars

A structure for passing the function to second and third derivatives.

Definition at line 64 of file deriv.h.

Data Fields

- func_t * func
  The pointer to the function.
- param_t * up
  The pointer to the user-specified parameters.

The documentation for this struct was generated from the following file:

- deriv.h

7.62 deriv_ioc Class Reference

```
#include <deriv_ioc.h>
```

7.62.1 Detailed Description

Setup I/O objects for numerical differentiation classes.

Definition at line 37 of file deriv_ioc.h.

Data Fields

- deriv_io_type * **deriv_io**
- eqi_deriv_io_type * **eqi_deriv_io**
- cern_deriv_io_type * **cern_deriv_io**
- gsl_deriv_io_type * **gsl_deriv_io**

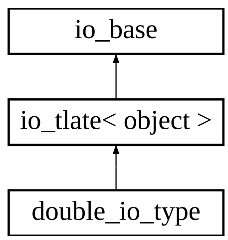
The documentation for this class was generated from the following file:

- deriv_ioc.h

7.63 double_io_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for double_io_type::



7.63.1 Detailed Description

I/O object for double variables.

Definition at line 1707 of file collection.h.

Public Member Functions

- double_io_type (const char *t)
  Desc.
- int addd (collection &co, std::string name, double x, bool overwrt=true)
  Add a double to a collection.
- double getd (collection &co, std::string tname)
  Get a double from a collection.
- int get_def (collection &co, std::string tname, double &op, double def=0.0)
  Get a double from a collection.

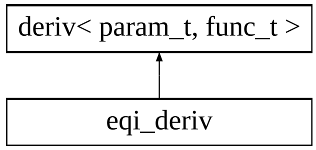
The documentation for this class was generated from the following file:

- collection.h

7.64 eqi_deriv Class Template Reference

```
#include <eqi_deriv.h>
```

Inheritance diagram for eqi_deriv::



7.64.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view> class eqi_deriv< param_t, func_t, vec_t >

Derivatives for equally-spaced abscissas.

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:

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.

Todo

The uncertainties in the derivatives are not yet computed and the second and third derivative formulas are not yet finished.

Two-point formula (note that this is independent of p).

$$f'(x_0 + ph) = \frac{1}{h}\left[f_1 - f_0\right]$$

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

$$\begin{aligned} f'(x_0 + ph) &= \frac{1}{h}\left[\frac{2p^3-3p^2-p+1}{12}f_{-2} - \frac{4p^3-3p^2-8p+4}{6}f_{-1}\right. \\ &\quad + \frac{2p^3-5p}{2}f_0 - \frac{4p^3+3p^2-8p-4}{6}f_1 \\ &\quad \left. + \frac{2p^3+3p^2-p-1}{12}f_2\right] \end{aligned}$$

The relations above can be confined to give formulas for second derivative formulas: Three-point formula

$$f'(x_0 + ph) = \frac{1}{h^2}\left[f_{-1} - 2f_0 + f_1\right]$$

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 135 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 double calc (double x, void ∗pa, func_t &func)
  Calculate the first derivative of `func` *w.r.t. x.*
- virtual double calc2 (double x, void ∗pa, func_t &func)
  Calculate the second derivative of `func` *w.r.t. x.*
- virtual double calc3 (double x, void ∗pa, func_t &func)
  Calculate the third derivative of `func` *w.r.t. x.*
- double calc_array (double x, double x0, double dx, size_t nx, const vec_t &y)
  Calculate the derivative at `x` *given an array.*
- double calc2_array (double x, double x0, double dx, size_t nx, const vec_t &y)
  Calculate the second derivative at `x` *given an array.*
- double calc3_array (double x, double x0, double dx, size_t nx, const vec_t &y)
  Calculate the third derivative at `x` *given an array.*
- int deriv_array (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).

7.64.2 Member Function Documentation

7.64.2.1 int set_npoints (int *npoints*) `[inline]`

Set the number of points to use for first derivatives (default 5).

Acceptable values are 2-5 (see above).

Definition at line 157 of file eqi_deriv.h.

7.64.2.2 int set_npoints2 (int *npoints*) [inline]

Set the number of points to use for second derivatives (default 5).

Acceptable values are 3-5 (see above).

Definition at line 183 of file eqi_deriv.h.

7.64.2.3 double calc_array (double *x*, double *x0*, double *dx*, size_t *nx*, const vec_t & *y*) [inline]

Calculate the derivative at `x` given an array.

This calculates the derivative at `x` given a func_tion 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 234 of file eqi_deriv.h.

7.64.2.4 double calc2_array (double *x*, double *x0*, double *dx*, size_t *nx*, const vec_t & *y*) [inline]

Calculate the second derivative at `x` given an array.

This calculates the second derivative at `x` given a func_tion 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 251 of file eqi_deriv.h.

7.64.2.5 double calc3_array (double *x*, double *x0*, double *dx*, size_t *nx*, const vec_t & *y*) [inline]

Calculate the third derivative at `x` given an array.

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 269 of file eqi_deriv.h.

7.64.2.6 int deriv_array (size_t *nv*, double *dx*, const vec_t & *y*, vec_t & *dydx*) [inline]

Calculate the derivative of an entire array.

Right now this uses np=5.

Todo

generalize to other values of npoints.

Definition at line 283 of file eqi_deriv.h.

The documentation for this class was generated from the following file:

- eqi_deriv.h

7.65 err_class Class Reference

`#include <err_hnd.h>`

7.65.1 Detailed Description

The error handler.

An error handler for use in O2scl 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 135 of file err_hnd.h.

Public Member Functions

- void set (const char ∗reason, const char ∗file, int line, int lerrno)
  Set an error.
- void add (const char ∗reason, const char ∗file, int line, int lerrno)
  Add information to previous error.
- void get (const char ∗&reason, const char ∗&file, int &line, int &lerrno)
  Get the last error.
- int get_errno ()
  Return the last error number.
- int get_line ()
  Return the line number of the last error.
- const char ∗ get_reason ()
  Return the reason for the last error.
- const char ∗ get_file ()
  Return the file name of the last error.
- const char ∗ get_str ()
  Return a string summarizing the last error.
- void reset ()
  Remove last error information.
- void set_mode (int m)
  Force a hard exit if an error occurs.

Static Public Member Functions

- static void gsl_hnd (const char ∗reason, const char ∗file, int line, int lerrno)
  Set an error.

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 35).

Protected Attributes

- int a_errno
  The error number.
- int a_line
  The line number.
- int mode
  The mode of error handling.
- 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 err_class * ptr
  A pointer to the default error handler.
- 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.

7.65.2 Member Function Documentation

7.65.2.1 static void gsl_hnd (const char * *reason*, const char * *file*, int *line*, int *lerrno*) [static]

Set an error.

This is separate from set(), since the gsl error handler needs to be a static function.

7.65.3 Field Documentation

7.65.3.1 bool array_abort

If true, call exit() when an array index error is set (default true).

This is ignored if `O2SCL_ARRAY_ABORT` is not defined.

Definition at line 192 of file err_hnd.h.

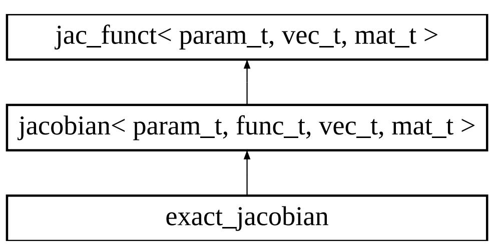
The documentation for this class was generated from the following file:

- err_hnd.h

7.66 exact_jacobian Class Template Reference

`#include <jacobian.h>`

Inheritance diagram for exact_jacobian::



7.66.1 Detailed Description

template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_view, class mat_t = omatrix_view> class exact_jacobian< param_t, func_t, vec_t, mat_t >

A direct calculation of the jacobian using a deriv object.

Note that it is sometimes 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 443 of file jacobian.h.

Public Member Functions

- int set_deriv (deriv< ej_parms, funct< ej_parms > > &de)
  Set the derivative object.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &jac, param_t &pa)
  The operator().

Data Fields

- gsl_deriv< ej_parms, funct< ej_parms > > def_deriv
  The default derivative object.

Protected Member Functions

- int dfn (double x, double &y, ej_parms &ejp)
  Function for the derivative object.

Protected Attributes

- deriv< ej_parms, funct< ej_parms > > * dptr
  Pointer to the derivative object.

Data Structures

- struct ej_parms
  Parameter structure for passing information.

The documentation for this class was generated from the following file:

- jacobian.h

7.67 exact_jacobian::ej_parms Struct Reference

```
#include <jacobian.h>
```

7.67.1 Detailed Description

template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_view, class mat_t = omatrix_view> struct exact_jacobian< param_t, func_t, vec_t, mat_t >::ej_parms

Parameter structure for passing information.

This class is primarily useful for specifying derivatives for using the jacobian::set_deriv() function.

Definition at line 465 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.
- param_t ∗ pa
  The parameters.

The documentation for this struct was generated from the following file:

- jacobian.h

7.68 file_detect Class Reference

`#include <file_detect.h>`

7.68.1 Detailed Description

Read a (possibly compressed) file and automatically detect the file format.

Really nasty hack. This works by copying the file to a temporary file in /tmp and then uncompressing it using a call to `system`("gunzip /tmp/filename") . When the file is closed, the temporary file is removed using 'rm -f'.

If the filename ends with ".gz" or ".bz2", then input_detect will try to uncompress it (using gunzip or bunzip2), otherwise, the file will be treated as normal.

Note that there must be enough disk space in the temporary directory for the uncompressed file or the read will fail.

Idea for future

Allow the user to specify the compression commands in configure, or at least specify the path to gzip, bzip2, etc.

Definition at line 57 of file file_detect.h.

Public Member Functions

- in_file_format ∗ open (const char ∗s)
  Open an input file with the given name.
- virtual int close ()
  Close an input file.
- virtual bool is_compressed ()
  Return `true` if the opened file was originally compressed.
- virtual bool is_binary ()
  Return `true` if the opened file was a binary file.

Protected Attributes

- std::string temp_filename
  The temporary filename.
- std::string user_filename
  The user-supplied filename.
- in_file_format * iffp
  The input file.
- bool compressed
  True if the file was compressed.
- bool binary
  True if the file was a binary file.

7.68.2 Member Function Documentation

7.68.2.1 in_file_format* open (const char * *s*)

Open an input file with the given name.

If the filename ends with ".gz" or ".bz2", then the file is assumed to be compressed.

It is important to note that the file is not closed until file_detect::close() method is called.

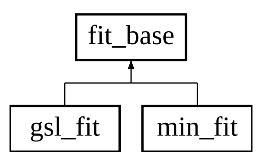
The documentation for this class was generated from the following file:

- file_detect.h

7.69 fit_base Class Template Reference

`#include <fit_base.h>`

Inheritance diagram for fit_base::



7.69.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view> class fit_base< param_t, func_t, vec_t, mat_t >

Non-linear least-squares fitting [abstract base].

Definition at line 268 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, param_t &pa, 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.

7.69.2 Member Function Documentation

7.69.2.1 virtual int print_iter (size_t *nv*, vec_t & *x*, double *y*, int *iter*, double *value* = `0.0`, double *limit* = `0.0`) `[inline, virtual]`

Print out iteration information.

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 287 of file fit_base.h.

7.69.2.2 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*, param_t & *pa*, func_t & *fitfun*) `[pure virtual]`

Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Implemented in fit_fix_pars, gsl_fit, and min_fit.

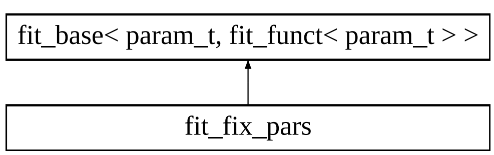
The documentation for this class was generated from the following file:

- fit_base.h

7.70 fit_fix_pars Class Template Reference

```
#include <fit_fix.h>
```

Inheritance diagram for fit_fix_pars::



7.70.1 Detailed Description

template<class param_t, class bool_vec_t> class fit_fix_pars< param_t, bool_vec_t >

Multidimensional fitting fixing some parameters and varying others.

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_view &xdat, ovector_view &ydat, ovector_view &yerr, size_t npar, ovector_view &par, omatrix_view &covar, double &chi2, param_t &pa, fit_funct< param_t > &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_view &xdat, ovector_view &ydat, ovector_view &yerr, size_t npar, ovector_view &par, bool_vec_t &fix, omatrix_view &covar, double &chi2, param_t &pa, fit_funct< param_t > &fitfun)
  Fit function `func` *while fixing some parameters as specified in* `fix`.
- int set_fit (fit_base< param_t, fit_funct_mfptr< fit_fix_pars, param_t > > &fitter)
  Change the base minimizer.

Data Fields

- gsl_fit< param_t, fit_funct_mfptr< fit_fix_pars, param_t > > def_fit
  The default base minimizer.

Protected Member Functions

- virtual int fit_func (size_t nv, ovector_view &x, double xx, double &y, param_t &pa)
  The new function to send to the minimizer.

Protected Attributes

- fit_base< param_t, fit_funct_mfptr< fit_fix_pars, param_t > > ∗ fitp
  The minimizer.
- fit_funct< param_t > ∗ 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_view ∗ xp
  The user-specified initial vector.

Private Member Functions

- **fit_fix_pars** (const fit_fix_pars &)
- fit_fix_pars & **operator=** (const fit_fix_pars &)

7.70.2 Member Function Documentation

7.70.2.1 virtual int fit (size_t *ndat*, ovector_view & *xdat*, ovector_view & *ydat*, ovector_view & *yerr*, size_t *npar*, ovector_view & *par*, omatrix_view & *covar*, double & *chi2*, param_t & *pa*, fit_funct< param_t > & *fitfun*) `[inline, virtual]`

Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Implements fit_base< param_t, fit_funct< param_t > >.

Definition at line 56 of file fit_fix.h.

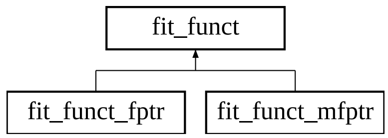
The documentation for this class was generated from the following file:

- fit_fix.h

7.71 fit_funct Class Template Reference

`#include <fit_base.h>`

Inheritance diagram for fit_funct::



7.71.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class fit_funct< param_t, vec_t >

Fitting function [abstract base].

Definition at line 38 of file fit_base.h.

Public Member Functions

- virtual int operator() (size_t np, vec_t &p, double x, double &y, param_t &pa)=0
    Using parameters in `p`*, predict* `y` *given* `x`*.*

Private Member Functions

- **fit_funct** (const fit_funct &)
- fit_funct & **operator=** (const fit_funct &)

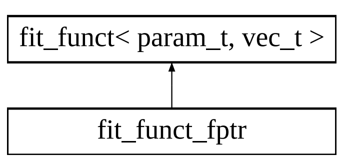
The documentation for this class was generated from the following file:

- fit_base.h

7.72 fit_funct_fptr Class Template Reference

`#include <fit_base.h>`

Inheritance diagram for fit_funct_fptr::



7.72.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class fit_funct_fptr< param_t, vec_t >

Function pointer fitting function.

Definition at line 63 of file fit_base.h.

Public Member Functions

- fit_funct_fptr (int(∗fp)(size_t np, vec_t &p, double x, double &y, param_t &pa))
  Specify a fitting function by a function pointer.
- virtual int operator() (size_t np, vec_t &p, double x, double &y, param_t &pa)
  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

- int(∗ fptr)(size_t np, vec_t &p, double x, double &y, param_t &pa)
  Storage for the user-specified function pointer.

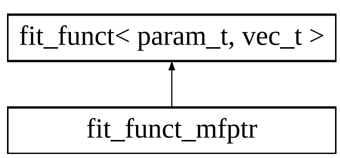
The documentation for this class was generated from the following file:

- fit_base.h

7.73 fit_funct_mfptr Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit_funct_mfptr::



7.73.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class fit_funct_mfptr< tclass, param_t, vec_t >

Member function pointer fitting function.

Definition at line 104 of file fit_base.h.

Public Member Functions

- fit_funct_mfptr (tclass ∗tp, int(tclass::∗fp)(size_t np, vec_t &p, double x, double &y, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t np, vec_t &p, double x, double &y, param_t &pa)
  Using parameters in `p`*, predict* `y` *given* `x`*.*

Protected Attributes

- int(tclass::∗ fptr)(size_t np, vec_t &p, double x, double &y, param_t &pa)
  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 &)

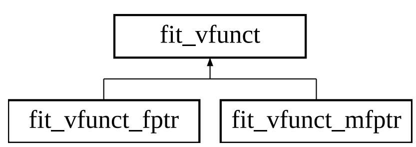
The documentation for this class was generated from the following file:

- fit_base.h

7.74 fit_vfunct Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit_vfunct::



7.74.1 Detailed Description

template<class param_t, size_t nvar> class fit_vfunct< param_t, nvar >

Fitting function [abstract base].

Definition at line 151 of file fit_base.h.

Public Member Functions

- virtual int operator() (size_t np, double p[ ], double x, double &y, param_t &pa)=0
  Using parameters in `p`*, predict* `y` *given* `x`*.*

Private Member Functions

- **fit_vfunct** (const fit_vfunct &)
- fit_vfunct & **operator=** (const fit_vfunct &)

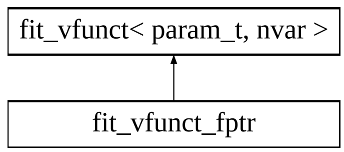
The documentation for this class was generated from the following file:

- fit_base.h

7.75 fit_vfunct_fptr Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit_vfunct_fptr::



7.75.1 Detailed Description

template<class param_t, size_t nvar> class fit_vfunct_fptr< param_t, nvar >

Function pointer fitting function.

Definition at line 175 of file fit_base.h.

Public Member Functions

- fit_vfunct_fptr (int(∗fp)(size_t np, double p[ ], double x, double &y, param_t &pa))
  Specify a fitting function by a function pointer.
- virtual int operator() (size_t np, double p[ ], double x, double &y, param_t &pa)
  Using parameters in `p`*, predict* `y` *given* `x`.

Protected Attributes

- int(∗ fptr)(size_t np, double p[ ], double x, double &y, param_t &pa)
  Storage for the user-specified function pointer.

Private Member Functions

- **fit_vfunct_fptr** (const fit_vfunct_fptr &)
- fit_vfunct_fptr & **operator=** (const fit_vfunct_fptr &)

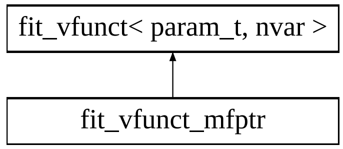
The documentation for this class was generated from the following file:

- fit_base.h

7.76 fit_vfunct_mfptr Class Template Reference

```
#include <fit_base.h>
```

Inheritance diagram for fit_vfunct_mfptr::



7.76.1 Detailed Description

template<class tclass, class param_t, size_t nvar> class fit_vfunct_mfptr< tclass, param_t, nvar >

Member function pointer fitting function.

Definition at line 218 of file fit_base.h.

Public Member Functions

- fit_vfunct_mfptr (tclass ∗tp, int(tclass::∗fp)(size_t np, double p[ ], double x, double &y, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t np, double p[ ], double x, double &y, param_t &pa)
  Using parameters in `p`*, predict* `y` *given* `x`.

Protected Attributes

- int(tclass::∗ fptr)(size_t np, double p[ ], double x, double &y, param_t &pa)
  Storage for the user-specified function pointer.
- tclass ∗ tptr
  Storage for the class pointer.

Private Member Functions

- **fit_vfunct_mfptr** (const fit_vfunct_mfptr &)
- fit_vfunct_mfptr & **operator=** (const fit_vfunct_mfptr &)

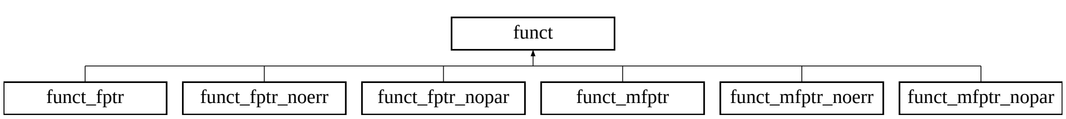
The documentation for this class was generated from the following file:

- fit_base.h

7.77 funct Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for funct::



7.77.1 Detailed Description

template<class param_t> class funct< param_t >

One-dimensional function [abstract base].

This class generalizes a function y(x).

Definition at line 40 of file funct.h.

Public Member Functions

- virtual int operator() (double x, double &y, param_t &pa)=0
  The overloaded operator().
- virtual double operator() (double x, param_t &pa)
  The overloaded operator().

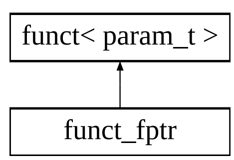
The documentation for this class was generated from the following file:

- funct.h

7.78 funct_fptr Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for funct_fptr::



7.78.1 Detailed Description

template<class param_t> class funct_fptr< param_t >

Function pointer to a function.

Definition at line 75 of file funct.h.

Public Member Functions

- funct_fptr (int(∗fp)(double x, double &y, param_t &pa))
  Specify the function pointer.
- virtual int operator() (double x, double &y, param_t &pa)
  The overloaded operator().
- virtual double operator() (double x, param_t &pa)
  The overloaded operator().

Protected Attributes

- int(∗ fptr)(double x, double &y, param_t &pa)

Storage for the function pointer.

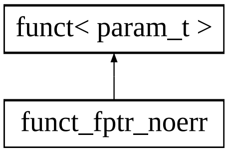
The documentation for this class was generated from the following file:

- funct.h

7.79 funct_fptr_noerr Class Template Reference

`#include <funct.h>`

Inheritance diagram for funct_fptr_noerr::



7.79.1 Detailed Description

template<class param_t> class funct_fptr_noerr< param_t >

Function pointer to a function.

Definition at line 124 of file funct.h.

Public Member Functions

- funct_fptr_noerr (double(∗fp)(double x, param_t &pa))
  Specify the function pointer.
- virtual int operator() (double x, double &y, param_t &pa)
  The overloaded operator().
- virtual double operator() (double x, param_t &pa)
  The overloaded operator().

Protected Attributes

- double(∗ fptr)(double x, param_t &pa)
  Storage for the function pointer.

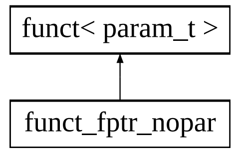
The documentation for this class was generated from the following file:

- funct.h

7.80 funct_fptr_nopar Class Template Reference

`#include <funct.h>`

Inheritance diagram for funct_fptr_nopar::



7.80.1 Detailed Description

template<class param_t> class funct_fptr_nopar< param_t >

Function pointer to a function.

Definition at line 170 of file funct.h.

Public Member Functions

- funct_fptr_nopar (double(∗fp)(double x))
  Specify the function pointer.
- virtual int operator() (double x, double &y, param_t &pa)
  The overloaded operator().
- virtual double operator() (double x, param_t &pa)
  The overloaded operator().

Protected Attributes

- double(∗ fptr)(double x)
  Storage for the function pointer.

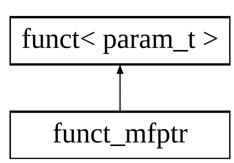
The documentation for this class was generated from the following file:

- funct.h

7.81 funct_mfptr Class Template Reference

`#include <funct.h>`

Inheritance diagram for funct_mfptr::



7.81.1 Detailed Description

template<class tclass, class param_t> class funct_mfptr< tclass, param_t >

Member function pointer to a one-dimensional function.

Definition at line 216 of file funct.h.

Public Member Functions

- funct_mfptr (tclass ∗tp, int(tclass::∗fp)(double x, double &y, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (double x, double &y, param_t &pa)
  The overloaded operator().
- virtual double operator() (double x, param_t &pa)
  The overloaded operator().

Protected Attributes

- int(tclass::∗ fptr)(double x, double &y, param_t &pa)
  Storage for the member function pointer.
- tclass ∗ tptr
  Store the pointer to the class instance.

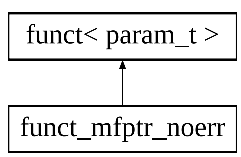
The documentation for this class was generated from the following file:

- funct.h

7.82 funct_mfptr_noerr Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for funct_mfptr_noerr::



7.82.1 Detailed Description

template<class tclass, class param_t> class funct_mfptr_noerr< tclass, param_t >

Member function pointer to a one-dimensional function.

Definition at line 266 of file funct.h.

Public Member Functions

- funct_mfptr_noerr (tclass ∗tp, double(tclass::∗fp)(double x, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (double x, double &y, param_t &pa)
  The overloaded operator().
- virtual double operator() (double x, param_t &pa)
  The overloaded operator().

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.

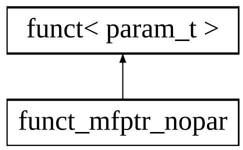
The documentation for this class was generated from the following file:

- funct.h

7.83 funct_mfptr_nopar Class Template Reference

```
#include <funct.h>
```

Inheritance diagram for funct_mfptr_nopar::



7.83.1 Detailed Description

template<class tclass, class param_t> class funct_mfptr_nopar< tclass, param_t >

Member function pointer to a one-dimensional function.

Definition at line 317 of file funct.h.

Public Member Functions

- funct_mfptr_nopar (tclass ∗tp, double(tclass::∗fp)(double x))
  Specify the member function pointer.
- virtual int operator() (double x, double &y, param_t &pa)
  The overloaded operator().
- virtual double operator() (double x, param_t &pa)
  The overloaded operator().

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

7.84 gaussian_2d Class Template Reference

```
#include <gaussian_2d.h>
```

7.84.1 Detailed Description

template<class rng_t> class gaussian_2d< rng_t >

Generate two random numbers from a normal distribution.

Todo

Double check that sigma is implemented correctly

Definition at line 37 of file gaussian_2d.h.

Public Member Functions

- void random (double sigma, double &x, double &y)
  Generate two numbers from a distribution with zero mean and standard deviation sigma.

Data Fields

- rng_t r
  Desc.

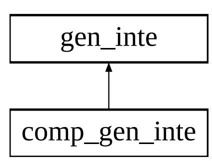
The documentation for this class was generated from the following file:

- gaussian_2d.h

7.85 gen_inte Class Template Reference

```
#include <gen_inte.h>
```

Inheritance diagram for gen_inte::



7.85.1 Detailed Description

template<class param_t, class func_t, class lfunc_t, class ufunc_t, class vec_t = ovector_view> class gen_inte< param_t, func_t, lfunc_t, ufunc_t, vec_t >

Generalized multi-dimensional integration [abstract base].

In order to allow the user to specify only three functions (for the integrand, the lower limits, and the upper limits) the first integer variable is used to distinguish among the variable limits. So the function $a_0()$ is just lower(0,NULL,vp) where vp is a void pointer, the function a_1 is lower(1,x,vp) where x is a 1-dimensional vector, and the function a_i is lower(i,x,vp) where x is an i-dimensional vector. Similarly, the function b_i is upper(i,x,vp) where x is an i-dimensional vector.

Definition at line 44 of file gen_inte.h.

Public Member Functions

- virtual double ginteg (func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, param_t &pa)=0
  Integrate function `func` *from* $x_i = f_i(x_i)$ *to* $x_i = g_i(x_i)$ *for* $0 < i < \mathrm{ndim} - 1$.
- virtual int ginteg_err (func_t &func, size_t ndim, lfunc_t &a, ufunc_t &b, param_t &pa, double &res, double &err)
  Integrate function `func` *from* $x_i = f_i(x_i)$ *to* $x_i = g_i(x_i)$ *for* $0 < i < \mathrm{ndim} - 1$.
- 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 tolf
  The maximum "uncertainty" in the value of the integral.

Protected Attributes

- double interror
  The uncertainty for the last integration computation.

7.85.2 Member Function Documentation

7.85.2.1 virtual double ginteg (func_t & *func*, size_t *ndim*, lfunc_t & *a*, ufunc_t & *b*, param_t & *pa*) `[pure virtual]`

Integrate function `func` from $x_i = f_i(x_i)$ to $x_i = g_i(x_i)$ for $0 < i < \mathrm{ndim} - 1$.

7.85.2.2 virtual int ginteg_err (func_t & *func*, size_t *ndim*, lfunc_t & *a*, ufunc_t & *b*, param_t & *pa*, double & *res*, double & *err*) `[inline, virtual]`

Integrate function `func` from $x_i = f_i(x_i)$ to $x_i = g_i(x_i)$ for $0 < i < \mathrm{ndim} - 1$.

Definition at line 77 of file gen_inte.h.

7.85.2.3 double get_error () `[inline]`

Return the error in the result from the last call to ginteg() or ginteg_err().

This will quietly return zero if no integrations have been performed.

Definition at line 90 of file gen_inte.h.

The documentation for this class was generated from the following file:

- gen_inte.h

7.86 gen_test_number Class Template Reference

```
#include <misc.h>
```

7.86.1 Detailed Description

template<size_t tot> class gen_test_number< tot >

Generate number sequence for testing.

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. The template argument `tot` should probably be greater than or equal to three.

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
```

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()`.

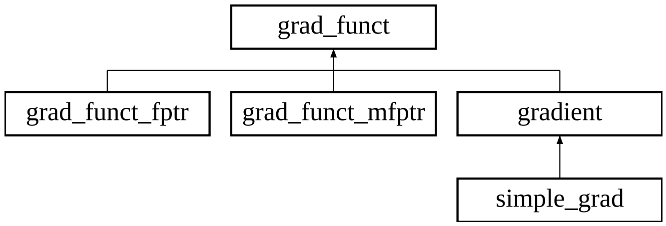
The documentation for this class was generated from the following file:

- misc.h

7.87 grad_funct Class Template Reference

`#include <multi_min.h>`

Inheritance diagram for grad_funct::



7.87.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class grad_funct< param_t, vec_t >

Gradient function [abstract base].

Definition at line 36 of file multi_min.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &g, param_t &pa)=0
  Compute the gradient `g` *at the point* `x`.

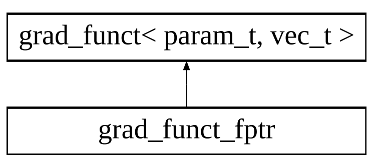
The documentation for this class was generated from the following file:

- multi_min.h

7.88 grad_funct_fptr Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad_funct_fptr::



7.88.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class grad_funct_fptr< param_t, vec_t >

Function pointer to a gradient.

Definition at line 50 of file multi_min.h.

Public Member Functions

- grad_funct_fptr (int(∗fp)(size_t nv, vec_t &x, vec_t &g, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &g, param_t &pa)
  Compute the gradient `g` *at the point* `x`.

Protected Attributes

- int(* fptr)(size_t nv, vec_t &x, vec_t &g, param_t &pa)
  The function pointer.

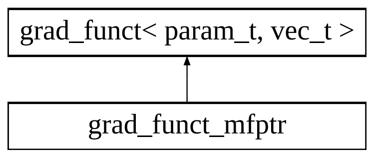
The documentation for this class was generated from the following file:

- multi_min.h

7.89 grad_funct_mfptr Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad_funct_mfptr::



7.89.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class grad_funct_mfptr< tclass, param_t, vec_t >

Member function pointer to a gradient.

Definition at line 91 of file multi_min.h.

Public Member Functions

- grad_funct_mfptr (tclass *tp, int(tclass::*fp)(size_t nv, vec_t &x, vec_t &g, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &g, param_t &pa)
  Compute the gradient `g` *at the point* `x`.

Protected Attributes

- int(tclass::* fptr)(size_t nv, vec_t &x, vec_t &g, param_t &pa)
  Member function pointer.
- tclass * tptr
  Class pointer.

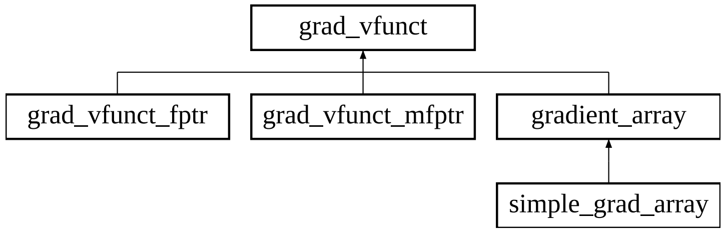
The documentation for this class was generated from the following file:

- multi_min.h

7.90 grad_vfunct Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for grad_vfunct::



7.90.1 Detailed Description

template<class param_t, size_t nv> class grad_vfunct< param_t, nv >

Base class for a gradient function using arrays [abstract base].

Definition at line 208 of file multi_min.h.

Public Member Functions

- virtual int operator() (size_t nvar, double x[nv], double g[nv], param_t &pa)=0
  Compute the gradient `g` *at the point* `x`.

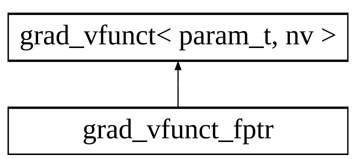
The documentation for this class was generated from the following file:

- multi_min.h

7.91 grad_vfunct_fptr Class Template Reference

`#include <multi_min.h>`

Inheritance diagram for grad_vfunct_fptr::



7.91.1 Detailed Description

template<class param_t, size_t nv> class grad_vfunct_fptr< param_t, nv >

Function pointer to a gradient.

Definition at line 224 of file multi_min.h.

Public Member Functions

- grad_vfunct_fptr (int(*fp)(size_t nv, double x[nv], double g[nv], param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nvar, double x[nv], double g[nv], param_t &pa)
  Compute the gradient `g` *at the point* `x`.

Protected Attributes

- int(∗ fptr)(size_t nvar, double x[nv], double g[nv], param_t &pa)
  Function pointer.

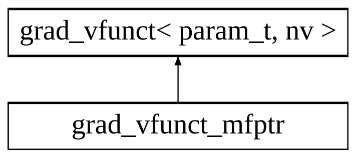
The documentation for this class was generated from the following file:

- multi_min.h

7.92 grad_vfunct_mfptr Class Template Reference

`#include <multi_min.h>`

Inheritance diagram for grad_vfunct_mfptr::



7.92.1 Detailed Description

template<class tclass, class param_t, size_t nv> class grad_vfunct_mfptr< tclass, param_t, nv >

Member function pointer to a gradient.

Definition at line 267 of file multi_min.h.

Public Member Functions

- grad_vfunct_mfptr (tclass ∗tp, int(tclass::∗fp)(size_t nvar, double x[nv], double g[nv], param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nvar, double x[nv], double g[nv], param_t &pa)
  Compute the gradient `g` *at the point* `x`.

Protected Attributes

- int(tclass::∗ fptr)(size_t nvar, double x[nv], double g[nv], param_t &pa)
  Member function pointer.
- tclass ∗ tptr
  Class pointer.

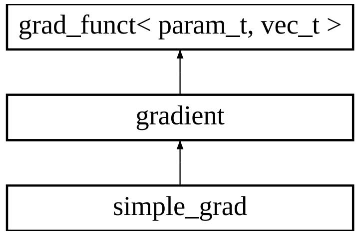
The documentation for this class was generated from the following file:

- multi_min.h

7.93 gradient Class Template Reference

`#include <multi_min.h>`

Inheritance diagram for gradient::



7.93.1 Detailed Description

template< class param_t, class func_t, class vec_t = ovector_view > class gradient< param_t, func_t, vec_t >

Class for automatically computing gradients [abstract base].

Definition at line 136 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, param_t &pa)=0
  Compute the gradient `g` *at the point* `x`.

Protected Attributes

- func_t ∗ func
  A pointer to the user-specified function.

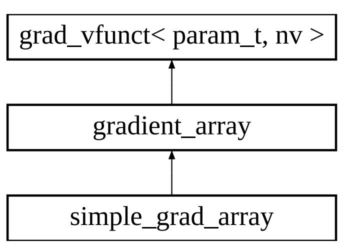
The documentation for this class was generated from the following file:

- multi_min.h

7.94 gradient_array Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for gradient_array::



7.94.1 Detailed Description

template< class param_t, class func_t, size_t nv > class gradient_array< param_t, func_t, nv >

Base class for automatically computing gradients with arrays [abstract base].

Definition at line 316 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 nvar, double x[nv], double g[nv], param_t &pa)=0
  Compute the gradient `g` *at the point* `x`.

Protected Attributes

- func_t ∗ func
  A pointer to the user-specified function.

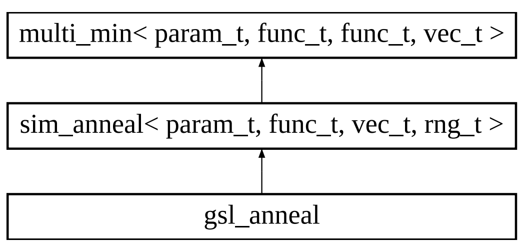
The documentation for this class was generated from the following file:

- multi_min.h

7.95 gsl_anneal Class Template Reference

```
#include <gsl_anneal.h>
```

Inheritance diagram for gsl_anneal::



7.95.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class rng_t = gsl_rnga> class gsl_anneal< param_t, func_t, vec_t, alloc_vec_t, alloc_t, rng_t >

Multidimensional minimization by simulated annealing (GSL).

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.

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 and temperature schedule are set in the parent, sim_anneal. The variables multi_min::tolx and multi_min::tolf are not used.

The step size for each dimension is specified in set_stepsize(). The number of stepsizes specified need not be the same as the number of dimensions. If `nstep` is the number of stepsizes, then the stepsize for dimension `i` is

```
step_size[i % nstep]
```

Idea for future

Implement a more general simulated annealing routine which would allow the solution of discrete problems like the Traveling Salesman problem.

Idea for future

Implement a method which automatically minimizes within some specified tolerance?

Definition at line 77 of file gsl_anneal.h.

Public Member Functions

- virtual int mmin (size_t nvar, vec_t &x0, double &fmin, param_t &pa, 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_stepsize (size_t n, vec2_t &ss)
  Set the step.

Data Fields

- double boltz
  Boltzmann factor (default 1.0).

Protected Member Functions

- 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.
- size_t nstep
  Number of step sizes.
- double ∗ step_sizes
  Step sizes.

Storage for present, next, and best vectors

- alloc_vec_t **x**
- alloc_vec_t **new_x**
- alloc_vec_t **best_x**

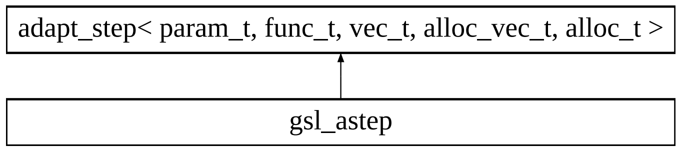
The documentation for this class was generated from the following file:

- gsl_anneal.h

7.96 gsl_astep Class Template Reference

```
#include <gsl_astep.h>
```

Inheritance diagram for gsl_astep::



7.96.1 Detailed Description

template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_astep< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Adaptive ODE stepper (GSL).

To modify the ODE stepper which is used, use the adapt_step::set_step().

Idea for future

Fix so that memory allocation/deallocation is performed only when necessary

Idea for future

Allow user to find out how many steps were taken, etc.

Definition at line 223 of file gsl_astep.h.

Public Member Functions

- virtual int astep_derivs (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &dydx, vec_t &yerr, param_t &pa, func_t &derivs)
  Make an adaptive integration step of the system `derivs` *with derivatives.*
- virtual int astep (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &u_dydx_out, vec_t &yerr, param_t &pa, 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 &t, double &h, double t1, size_t nvar, vec_t &y, vec_t &dydx, vec_t &yout2, vec_t &yerr, vec_t &dydx_out2, param_t &pa, 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_out
  Temporary storage for dydx_out.
- 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.

7.96.2 Member Function Documentation

7.96.2.1 virtual int astep_derivs (double & *x*, double & *h*, double *xmax*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yerr*, param_t & *pa*, func_t & *derivs*) `[inline, virtual]`

Make an adaptive integration step of the system `derivs` with derivatives.

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.

Definition at line 338 of file gsl_astep.h.

7.96.2.2 virtual int astep (double & *x*, double & *h*, double *xmax*, size_t *n*, vec_t & *y*, vec_t & *u_dydx_out*, vec_t & *yerr*, param_t & *pa*, func_t & *derivs*) `[inline, virtual]`

Make an adaptive integration step of the system `derivs`.

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.

Definition at line 380 of file gsl_astep.h.

The documentation for this class was generated from the following file:

- gsl_astep.h

7.97 gsl_chebapp Class Template Reference

`#include <gsl_chebapp.h>`

7.97.1 Detailed Description

template<class param_t, class func_t> class gsl_chebapp< param_t, func_t >

Chebyshev approximation (GSL).

Approximate a function using a Chebyshev series:

$$f(x) = \sum_n c_n T_n(x) \qquad \text{where} \qquad T_n(x) = \cos(n \arccos x)$$

Idea for future

Implement eval_err(), eval_n() and eval_n_err() methods.

Definition at line 46 of file gsl_chebapp.h.

Public Member Functions

- int init (func_t &func, double a, double b, param_t &vp)
  Initialize a Chebyshev approximation of the function `func` *over the interval from* `a` *to* `b`*.*
- int set_order (size_t od)
  Set the order (default 5).
- double eval (double x)
  Evaluate the approximation.
- gsl_chebapp ∗ deriv ()
  Return a pointer to an approximation to the derivative.
- gsl_chebapp ∗ inte ()
  Return a pointer to an approximation to the integral.
- double get_coefficient (size_t ix)
  Get the coefficient.

7.97.2 Member Function Documentation

7.97.2.1 int init (func_t & *func*, double *a*, double *b*, param_t & *vp*) `[inline]`

Initialize a Chebyshev approximation of the function `func` over the interval from `a` to `b`.

The interval must be specified so that $a < b$.

Definition at line 60 of file gsl_chebapp.h.

7.97.2.2 int set_order (size_t *od*) `[inline]`

Set the order (default 5).

The function init() must be called after calling set_order() to reinitialize the series for the new order.

Definition at line 97 of file gsl_chebapp.h.

7.97.2.3 gsl_chebapp∗ deriv () `[inline]`

Return a pointer to an approximation to the derivative.

The new gsl_chebapp object is allocated by new, and the memory should be deallocated using delete by the user.

Definition at line 122 of file gsl_chebapp.h.

7.97.2.4 gsl_chebapp∗ inte () `[inline]`

Return a pointer to an approximation to the integral.

The new gsl_chebapp object is allocated by new, and the memory should be deallocated using delete by the user.

Definition at line 141 of file gsl_chebapp.h.

7.97.2.5 double get_coefficient (size_t *ix*) [inline]

Get the coefficient.

Legal values of the argument are 0 to `order+1`

Definition at line 159 of file gsl_chebapp.h.

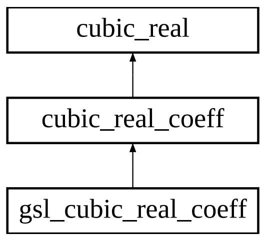
The documentation for this class was generated from the following file:

- gsl_chebapp.h

7.98 gsl_cubic_real_coeff Class Reference

`#include <poly.h>`

Inheritance diagram for gsl_cubic_real_coeff::



7.98.1 Detailed Description

Solve a cubic with real coefficients and complex roots (GSL).

Definition at line 459 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* $x = x_3$ *.*
- 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()`.

7.98.2 Member Function Documentation

7.98.2.1 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()`.

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.

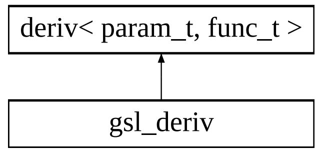
The documentation for this class was generated from the following file:

- poly.h

7.99 gsl_deriv Class Template Reference

```
#include <gsl_deriv.h>
```

Inheritance diagram for gsl_deriv::



7.99.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_deriv< param_t, func_t >

Numerical differentiation (GSL).

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.

Note:

Second and third derivatives are computed by naive nested applications of the formula for the first derivative and the uncertainty for these will likely be underestimated.

Idea for future

Include the forward and backward GSL derivatives

Definition at line 54 of file gsl_deriv.h.

Public Member Functions

- virtual int calc_err (double x, param_t &pa, 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

- virtual int calc_err_int (double x, typename deriv< param_t, func_t >::dpars &pa, funct< typename deriv< param_t, func_t >::dpars > &func, double &dfdx, double &err)
  Internal version of calc_err() for second and third derivatives.
- template<class func2_t, class param2_t>
  int central_deriv (double x, double hh, double &result, double &abserr_round, double &abserr_trunc, func2_t &func, param2_t &pa)
  Compute derivative using 5-point rule.

7.99.2 Member Function Documentation

7.99.2.1 int central_deriv (double *x*, double *hh*, double & *result*, double & *abserr_round*, double & *abserr_trunc*, func2_t & *func*, param2_t & *pa*) [inline, protected]

Compute derivative using 5-point rule.

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 200 of file gsl_deriv.h.

7.99.3 Field Documentation

7.99.3.1 double h

Initial stepsize.

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 71 of file gsl_deriv.h.

7.99.3.2 double h_opt

The last value of the optimized stepsize.

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 80 of file gsl_deriv.h.

The documentation for this class was generated from the following file:

- gsl_deriv.h

7.100 gsl_fft Class Reference

```
#include <gsl_fft.h>
```

7.100.1 Detailed Description

Real mixed-radix fast Fourier transform.

This is a simple wrapper for the GSL FFT functions which automatically allocates the necessary memory.

Definition at line 42 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.

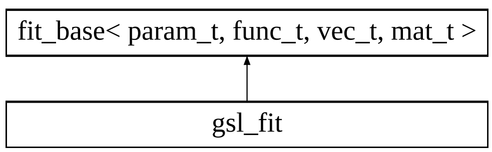
The documentation for this class was generated from the following file:

- gsl_fft.h

7.101 gsl_fit Class Template Reference

```
#include <gsl_fit.h>
```

Inheritance diagram for gsl_fit::



7.101.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view, class bool_vec_t = bool *> class gsl_fit< param_t, func_t, vec_t, mat_t, bool_vec_t >

Non-linear least-squares fitting class (GSL).

The GSL-based fitting class using a Levenberg-Marquardt type algorithm. The algorithm stops when

$$|dx_i| < \mathrm{epsabs} + \mathrm{epsrel} \times |x_i|$$

where dx is the last step and x is the current position. If test_gradient is true, then additionally fit() requires that

$$\sum_i |g_i| < \mathrm{epsabs}$$

where g_i is the i-th component of the gradient of the function $\Phi(x)$ where

$$\Phi(x) = ||F(x)||^2$$

Todo

Properly generalize other vector types than ovector_view

Todo

Allow the user to specify the derivatives

Todo

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 66 of file gsl_fit.h.

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, param_t &pa, 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 (int nv, gsl_vector ∗x, gsl_vector ∗dx, int iter, double l_epsabs, double l_epsrel)
  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.

Data Structures

- struct func_par
  A structure for passing to the functions func(), dfunc(), and fdfunc().

7.101.2 Member Function Documentation

7.101.2.1 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*, param_t & *pa*, func_t & *fitfun*) `[inline, virtual]`

Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Implements fit_base.

Definition at line 88 of file gsl_fit.h.

The documentation for this class was generated from the following file:

- gsl_fit.h

7.102 gsl_fit::func_par Struct Reference

```
#include <gsl_fit.h>
```

7.102.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view, class bool_vec_t = bool *> struct gsl_fit< param_t, func_t, vec_t, mat_t, bool_vec_t >::func_par

A structure for passing to the functions func(), dfunc(), and fdfunc().

Definition at line 219 of file gsl_fit.h.

Data Fields

- func_t & f
  The function object.
- param_t * vp
  The user-specified parameter.
- 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.

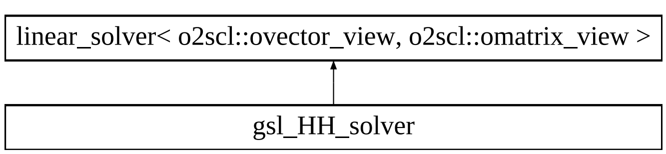
The documentation for this struct was generated from the following file:

- gsl_fit.h

7.103 gsl_HH_solver Class Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for gsl_HH_solver::



7.103.1 Detailed Description

GSL Householder solver.

Definition at line 176 of file ode_it_solve.h.

Public Member Functions

- virtual int solve (size_t n, o2scl::omatrix_view &A, o2scl::ovector_view &b, o2scl::ovector_view &x)
  Solve square linear system $Ax = b$ *of size* `n`.

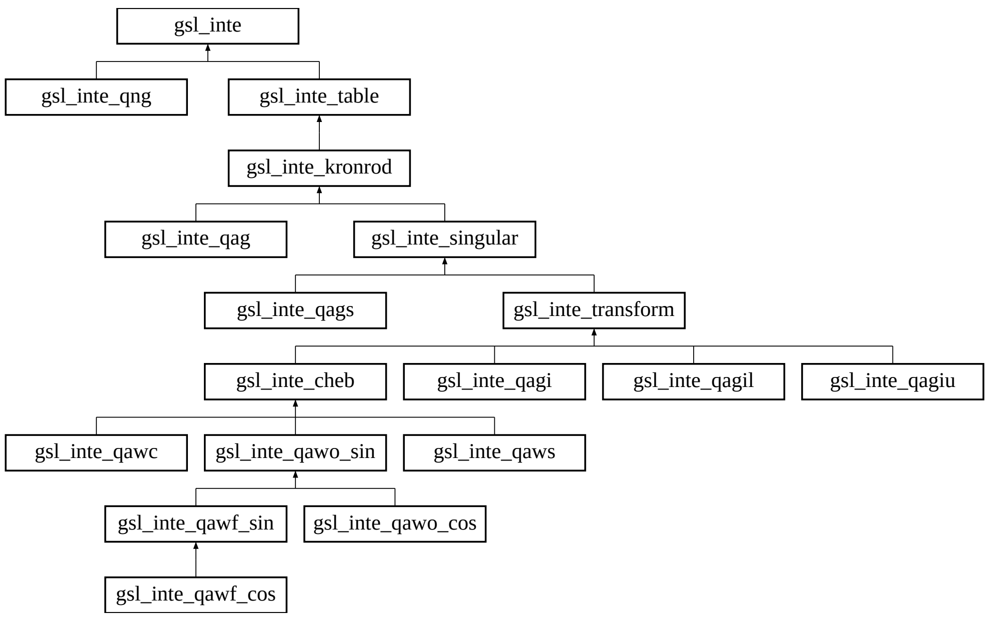
The documentation for this class was generated from the following file:

- ode_it_solve.h

7.104 gsl_inte Class Reference

`#include <gsl_inte.h>`

Inheritance diagram for gsl_inte::



7.104.1 Detailed Description

GSL integration base.

This base class does not perform any actual integration.

Definition at line 37 of file gsl_inte.h.

Protected Member Functions

- double rescale_error (double err, const double result_abs, const double result_asc)
  Rescale errors appropriately.

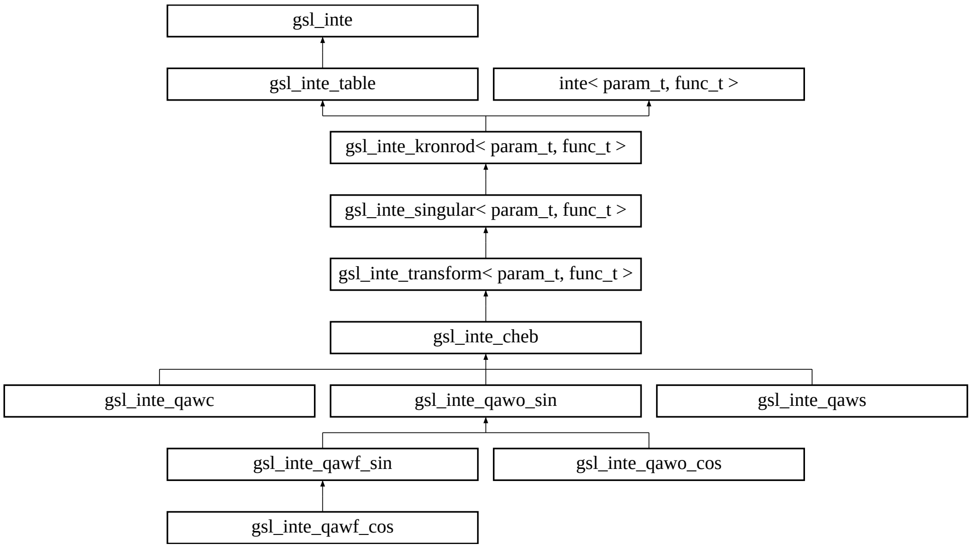
The documentation for this class was generated from the following file:

- gsl_inte.h

7.105 gsl_inte_cheb Class Template Reference

`#include <gsl_inte_qawc.h>`

Inheritance diagram for gsl_inte_cheb::



7.105.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_cheb< param_t, func_t >

Chebyshev integration (GSL).

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 not be specified in the argument `func` to integ(). This is different from how the cern_cauchy operates.

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 46 of file gsl_inte_qawc.h.

Public Member Functions

- void compute_moments (double cc, double *moment)
  Compute the Chebyshev moments.
- void gsl_integration_qcheb (func_t &f, double a, double b, double *cheb12, double *cheb24, param_t &pa)
  Perform the integration.

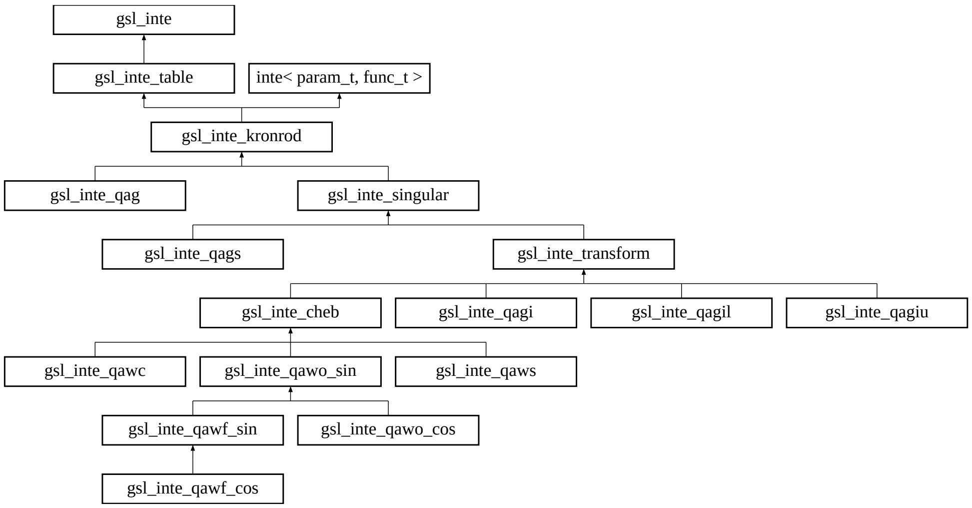
The documentation for this class was generated from the following file:

- gsl_inte_qawc.h

7.106 gsl_inte_kronrod Class Template Reference

`#include <gsl_inte_qag_b.h>`

Inheritance diagram for gsl_inte_kronrod::



7.106.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_kronrod< param_t, func_t >

Basic Gauss-Kronrod integration class (GSL).

Definition at line 548 of file gsl_inte_qag_b.h.

Public Member Functions

- virtual void gsl_integration_qk_o2scl (func_t &func, const int n, const double xgk[ ], const double wg[ ], const double wgk[ ], double fv1[ ], double fv2[ ], double a, double b, double ∗result, double ∗abserr, double ∗resabs, double ∗resasc, param_t &pa)

  The GSL Gauss-Kronrod integration function.

7.106.2 Member Function Documentation

7.106.2.1 virtual void gsl_integration_qk_o2scl (func_t & *func*, const int *n*, const double *xgk*[ ], const double *wg*[ ], const double *wgk*[ ], double *fv1*[ ], double *fv2*[ ], double *a*, double *b*, double ∗ *result*, double ∗ *abserr*, double ∗ *resabs*, double ∗ *resasc*, param_t & *pa*) `[inline, virtual]`

The GSL Gauss-Kronrod integration function.

Given abcissas and weights, this performs the integration of `func` between `a` and `b`, providing a result with uncertainties.

This function is designed for use with the values given in the o2scl_inte_qag_coeffs namespace.

Idea for future

This function, in principle, could be replaced with a generic integration pointer.

Reimplemented in gsl_inte_transform.

Definition at line 567 of file gsl_inte_qag_b.h.

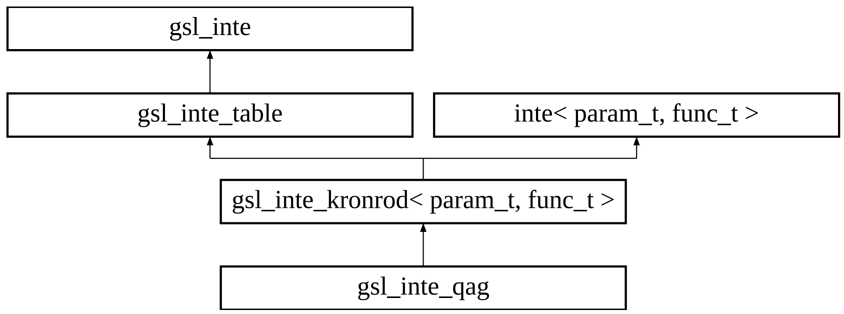
The documentation for this class was generated from the following file:

- gsl_inte_qag_b.h

7.107 gsl_inte_qag Class Template Reference

```
#include <gsl_inte_qag.h>
```

Inheritance diagram for gsl_inte_qag::



7.107.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_inte_qag< param_t, func_t >

Adaptive integration a function with finite limits of integration (GSL).

Todo

Verbose output has been setup for this class, but this needs to be done for some of the other GSL-like integrators

Todo

Document workspace size here somehow

Todo

Document use of last_iter

Definition at line 44 of file gsl_inte_qag.h.

Public Member Functions

- gsl_inte_qag (int key=1)
  Create an integrator with the specified key.
- int set_key (int key)
  Set the number of integration points.
- int get_key ()
  Return the key used (1-6).
- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  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 int qn, const double xgk[ ], const double wg[ ], const double wgk[ ], double fv1[ ], double fv2[ ], const double a, const double b, const double l_epsabs, const double l_epsrel, const size_t limit, double ∗result, double ∗abserr, param_t &pa)
  Perform an adaptive integration given the coefficients, and returning `result`.

Protected Attributes

- int lkey
  Select the number of integration points.

7.107.2 Member Function Documentation

7.107.2.1 int set_key (int *key*) `[inline]`

Set the number of integration points.

The possible values for `key` are:

- 1: GSL_INTEG_GAUSS15 (default)
- 2: GSL_INTEG_GAUSS21
- 3: GSL_INTEG_GAUSS31
- 4: GSL_INTEG_GAUSS41
- 5: GSL_INTEG_GAUSS51
- 6: GSL_INTEG_GAUSS61

If an integer other than 1-6 is given, the default (GSL_INTEG_GAUSS15) is assumed, and the error handler is called.

Definition at line 82 of file gsl_inte_qag.h.

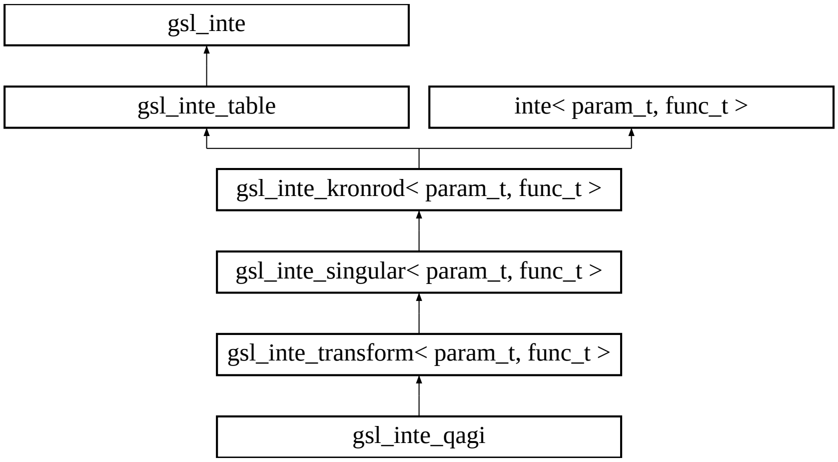
The documentation for this class was generated from the following file:

- gsl_inte_qag.h

7.108 gsl_inte_qagi Class Template Reference

`#include <gsl_inte_qagi.h>`

Inheritance diagram for gsl_inte_qagi::



7.108.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_inte_qagi< param_t, func_t >

Integrate a function from $-\infty$ to ∞ (GSL).

Definition at line 36 of file gsl_inte_qagi.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* $-\infty$ *to* ∞.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  Integrate function `func` *from* ∞ *to* ∞ *giving result* `res` *and error* `err`.

Protected Member Functions

- virtual double transform (func_t &func, double t, param_t &pa)
  Tranformation to $t \in (0, 1]$.

7.108.2 Member Function Documentation

7.108.2.1 virtual double integ (func_t & *func*, double *a*, double *b*, param_t & *pa*) `[inline, virtual]`

Integrate function `func` from $-\infty$ to ∞.

The values given in `a` and `b` are ignored

Reimplemented from inte.

Definition at line 46 of file gsl_inte_qagi.h.

7.108.2.2 virtual int integ_err (func_t & *func*, double *a*, double *b*, param_t & *pa*, double & *res*, double & *err2*) `[inline, virtual]`

Integrate function `func` from ∞ to ∞ giving result `res` and error `err`.

The values `a` and `b` are ignored

Reimplemented from inte.

Definition at line 58 of file gsl_inte_qagi.h.

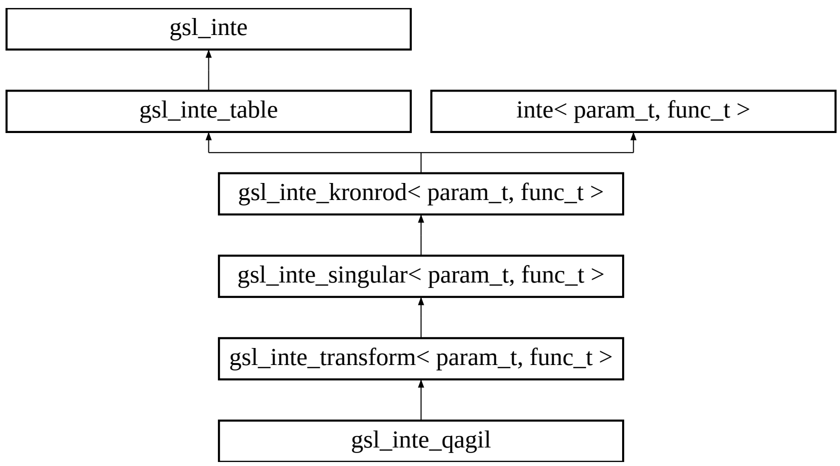
The documentation for this class was generated from the following file:

- gsl_inte_qagi.h

7.109 gsl_inte_qagil Class Template Reference

```
#include <gsl_inte_qagil.h>
```

Inheritance diagram for gsl_inte_qagil::



7.109.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_inte_qagil< param_t, func_t >

Integrate a function from $-\infty$ to b (GSL).

Definition at line 36 of file gsl_inte_qagil.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* $-\infty$ *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  Integrate function `func` *from* $-\infty$ *to* `b` *and place the result in* `res` *and the error in* `err2`.

Protected Member Functions

- virtual double transform (func_t &func, double t, param_t &pa)
  Transform to $t \in (0, 1]$.

Protected Attributes

- double lb
  Store the upper limit.

7.109.2 Member Function Documentation

7.109.2.1 virtual double integ (func_t & *func*, double *a*, double *b*, param_t & *pa*) `[inline, virtual]`

Integrate function `func` from $-\infty$ to `b`.

The value given in `a` is ignored.

Reimplemented from inte.

Definition at line 55 of file gsl_inte_qagil.h.

7.109.2.2 virtual int integ_err (func_t & *func*, double *a*, double *b*, param_t & *pa*, double & *res*, double & *err2*) `[inline, virtual]`

Integrate function `func` from $-\infty$ to `b` and place the result in `res` and the error in `err2`.

The value given in `a` is ignored.

Reimplemented from inte.

Definition at line 68 of file gsl_inte_qagil.h.

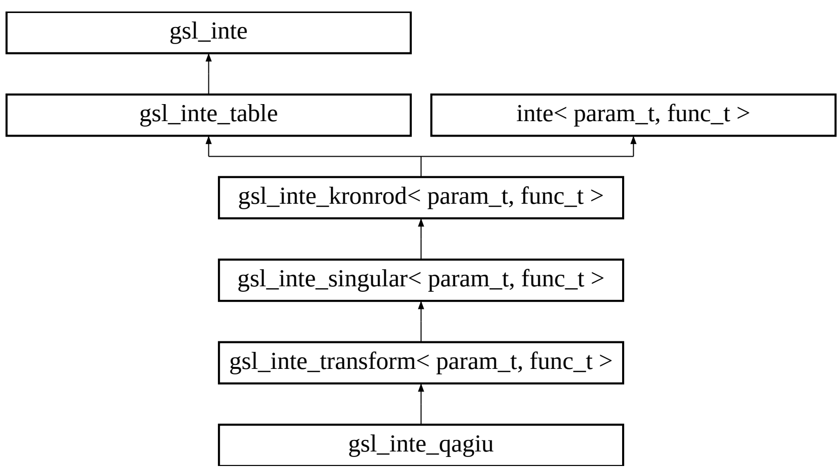
The documentation for this class was generated from the following file:

- gsl_inte_qagil.h

7.110 gsl_inte_qagiu Class Template Reference

`#include <gsl_inte_qagiu.h>`

Inheritance diagram for gsl_inte_qagiu::



7.110.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_inte_qagiu< param_t, func_t >

Integrate a function from a to ∞ (GSL).

Todo

I had to add extra code to check for non-finite values for some integrations. This should be checked.

The extra line was of the form:

```
if (!finite(area1)) area1=0.0;
```

Definition at line 44 of file gsl_inte_qagiu.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* ∞.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  Integrate function `func` *from* `a` *to* ∞ *giving result* `res` *and error* `err`.

Protected Member Functions

- virtual double transform (func_t &func, double t, param_t &pa)
  Transform to $t \in (0, 1]$.

Protected Attributes

- double la
  Store the lower limit.

7.110.2 Member Function Documentation

7.110.2.1 virtual double integ (func_t & *func*, double *a*, double *b*, param_t & *pa*) `[inline, virtual]`

Integrate function `func` from `a` to ∞.

The value `b` is ignored.

Reimplemented from inte.

Definition at line 64 of file gsl_inte_qagiu.h.

7.110.2.2 virtual int integ_err (func_t & *func*, double *a*, double *b*, param_t & *pa*, double & *res*, double & *err2*) `[inline, virtual]`

Integrate function `func` from `a` to ∞ giving result `res` and error `err`.

The value `b` is ignored.

Reimplemented from inte.

Definition at line 77 of file gsl_inte_qagiu.h.

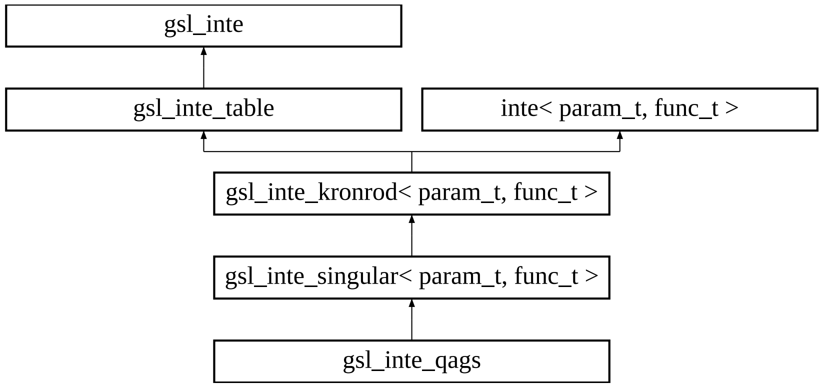
The documentation for this class was generated from the following file:

- gsl_inte_qagiu.h

7.111 gsl_inte_qags Class Template Reference

`#include <gsl_inte_qags.h>`

Inheritance diagram for gsl_inte_qags::



7.111.1 Detailed Description

template<class param_t, class func_t = funct<void ∗>> class gsl_inte_qags< param_t, func_t >

Integrate a function with a singularity (GSL).

Definition at line 36 of file gsl_inte_qags.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  Integrate function `func` *from* `a` *to* `b` *and place the result in* `res` *and the error in* `err`.

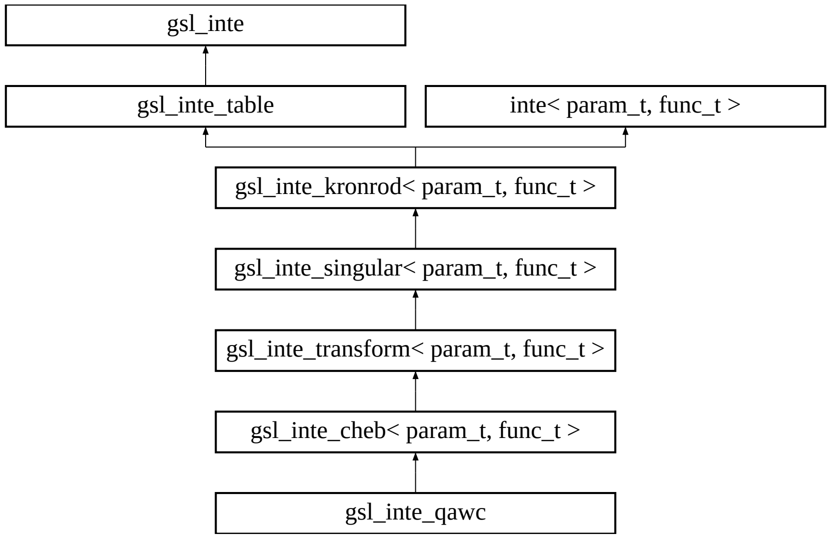
The documentation for this class was generated from the following file:

- gsl_inte_qags.h

7.112 gsl_inte_qawc Class Template Reference

`#include <gsl_inte_qawc.h>`

Inheritance diagram for gsl_inte_qawc::



7.112.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qawc< param_t, func_t >

Adaptive Cauchy principal value integration (GSL).

Definition at line 287 of file gsl_inte_qawc.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` from `a` to `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  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, const size_t limit, double *result, double *abserr, param_t &pa)
  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, param_t &pa)
  25-point quadrature for Cauchy principal values
- virtual double transform (func_t &func, double x, param_t &pa)
  Add the singularity to the function.
- const char * type ()
  Return string denoting type ("gsl_inte_qawc").

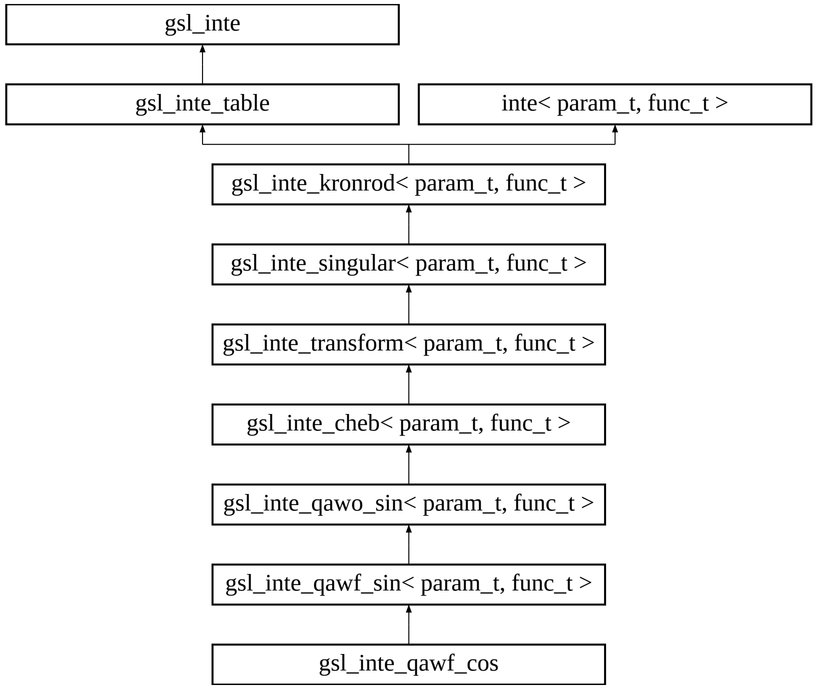
The documentation for this class was generated from the following file:

- gsl_inte_qawc.h

7.113 gsl_inte_qawf_cos Class Template Reference

`#include <gsl_inte_qawf.h>`

Inheritance diagram for gsl_inte_qawf_cos::



7.113.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qawf_cos< param_t, func_t >

Adaptive integration a function with finite limits of integration (GSL).

Todo

Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Definition at line 327 of file gsl_inte_qawf.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  Integrate function `func` *from* `a` *to* `b` *and place the result in* `res` *and the error in* `err`.

Protected Member Functions

- virtual double transform (func_t &func, double x, param_t &pa)
  Add the oscillating part to the integrand.
- const char * type ()
  Return string denoting type ("gsl_inte_qawf_cos").

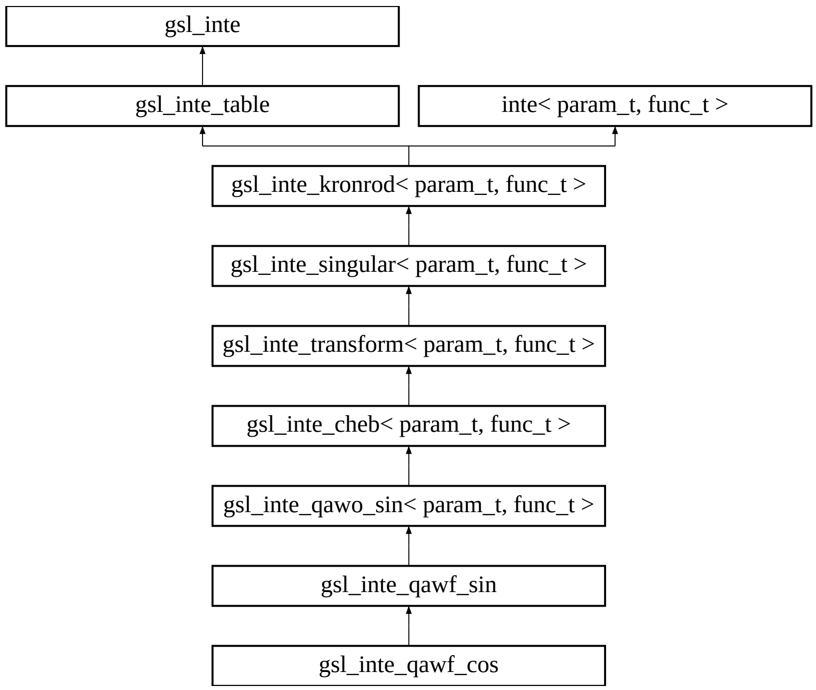
The documentation for this class was generated from the following file:

- gsl_inte_qawf.h

7.114 gsl_inte_qawf_sin Class Template Reference

```
#include <gsl_inte_qawf.h>
```

Inheritance diagram for gsl_inte_qawf_sin::



7.114.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qawf_sin< param_t, func_t >

Adaptive integration for oscillatory integrals (GSL).

Todo

Improve documentation a little

Definition at line 39 of file gsl_inte_qawf.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  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, const size_t limit, double ∗result, double ∗abserr, param_t &pa)

The full GSL integration routine called by integ_err().

- virtual double transform (func_t &func, double x, param_t &pa)
  Add the oscillating part to the integrand.
- const char ∗ type ()
  Return string denoting type ("gsl_inte_qawf_sin").

Protected Attributes

- gsl_integration_workspace ∗ cyclew
  The integration workspace.

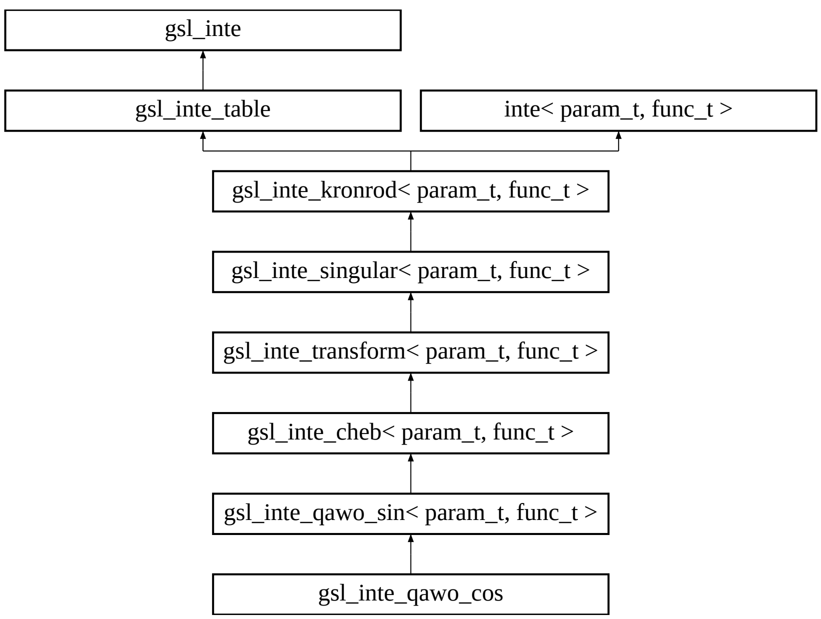
The documentation for this class was generated from the following file:

- gsl_inte_qawf.h

7.115 gsl_inte_qawo_cos Class Template Reference

```
#include <gsl_inte_qawo.h>
```

Inheritance diagram for gsl_inte_qawo_cos::



7.115.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qawo_cos< param_t, func_t >

Adaptive integration a function with finite limits of integration (GSL).

Todo

Verbose output has been setup for this class, but this needs to be done for the other GSL-like integrators

Definition at line 649 of file gsl_inte_qawo.h.

Public Member Functions

- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  Integrate function `func` *from* `a` *to* `b` *and place the result in* `res` *and the error in* `err`.

Protected Member Functions

- virtual double transform (func_t &func, double x, param_t &pa)
  Add the oscillating part to the integrand.
- const char ∗ type ()
  Return string denoting type ("gsl_inte_qawo_cos").

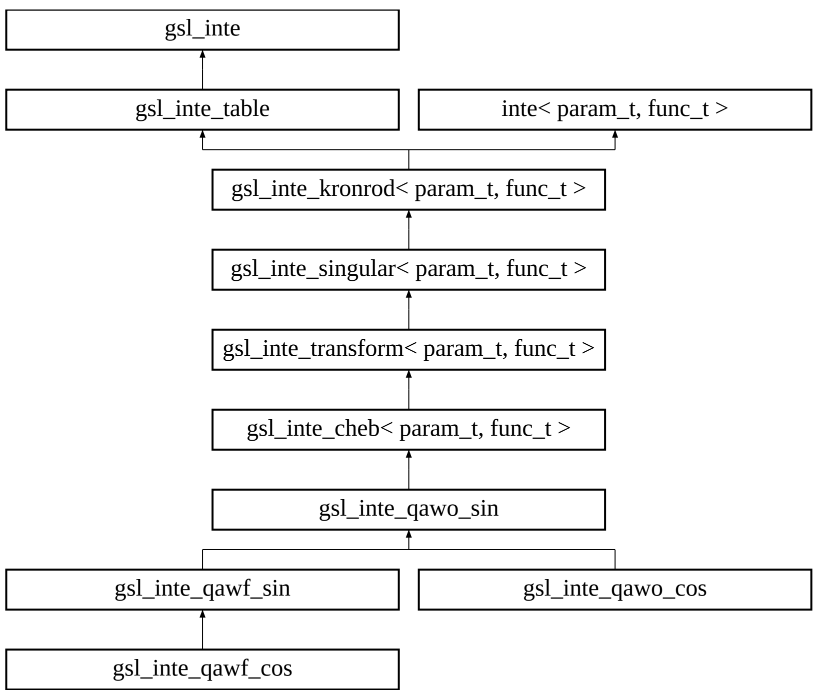
The documentation for this class was generated from the following file:

- gsl_inte_qawo.h

7.116 gsl_inte_qawo_sin Class Template Reference

`#include <gsl_inte_qawo.h>`

Inheritance diagram for gsl_inte_qawo_sin::



7.116.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qawo_sin< param_t, func_t >

Adaptive integration for oscillatory integrals (GSL).

Todo

Improve documentation

Definition at line 38 of file gsl_inte_qawo.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`*.*
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  Integrate function `func` *from* `a` *to* `b` *and place the result in* `res` *and the error in* `err`*.*

Data Fields

- double omega
  Desc.
- size_t tab_size
  Desc.

Protected Member Functions

- int qawo (func_t &func, const double a, const double epsabs, const double epsrel, const size_t limit, gsl_integration_workspace *loc_w, gsl_integration_qawo_table *wf, double *result, double *abserr, param_t &pa)
  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, param_t &pa)
  25-point quadrature for oscillating functions
- virtual double transform (func_t &func, double x, param_t &pa)
  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.

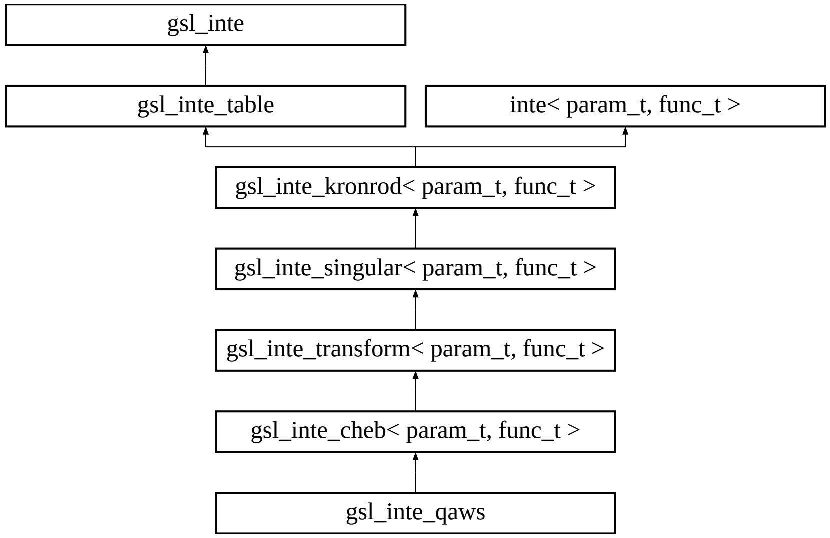
The documentation for this class was generated from the following file:

- gsl_inte_qawo.h

7.117 gsl_inte_qaws Class Template Reference

```
#include <gsl_inte_qaws.h>
```

Inheritance diagram for gsl_inte_qaws::



7.117.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qaws< param_t, func_t >

QAWS integration (GSL).

Note:

This is unfinished.

Todo

Finish this!

Definition at line 40 of file gsl_inte_qaws.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err2)
  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 qaws (func_t &func, const double a, const double b, const double c, const double epsabs, const double epsrel, const size_t limit, double ∗result, double ∗abserr, param_t &pa)
  Desc.
- double **fn_qaws** (double t, void ∗params)
- double **fn_qaws_L** (double x, void ∗params)

- double **fn_qaws_R** (double x, void *params)
- void **compute_result** (const double *r, const double *cheb12, const double *cheb24, double *result12, double *result24)
- void **qc25s** (gsl_function *f, double a, double b, double a1, double b1, gsl_integration_qaws_table *t, double *result, double *abserr, int *err_reliable)
- void **qc25s** (gsl_function *f, double a, double b, double a1, double b1, gsl_integration_qaws_table *t, double *result, double *abserr, int *err_reliable)
- double **fn_qaws** (double x, void *params)
- double **fn_qaws_L** (double x, void *params)
- double **fn_qaws_R** (double x, void *params)
- void **compute_result** (const double *r, const double *cheb12, const double *cheb24, double *result12, double *result24)
- virtual double transform (func_t &func, double x, param_t &pa)
  Desc.
- const char * type ()
  Return string denoting type ("gsl_inte_qaws").

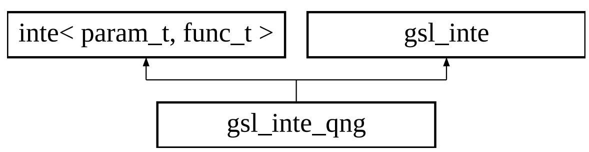
The documentation for this class was generated from the following file:

- gsl_inte_qaws.h

7.118 gsl_inte_qng Class Template Reference

```
#include <gsl_inte_qng.h>
```

Inheritance diagram for gsl_inte_qng::



7.118.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_qng< param_t, func_t >

Non-adaptive integration from a to b (GSL).

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 tolx and tolf.

Idea for future

Compare directly with GSL as is done in gsl_inte_qag_ts.

Definition at line 226 of file gsl_inte_qng.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, 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.

7.118.2 Field Documentation

7.118.2.1 size_t feval

The number of function evalutions for the last integration.

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 set to 88.

Definition at line 241 of file gsl_inte_qng.h.

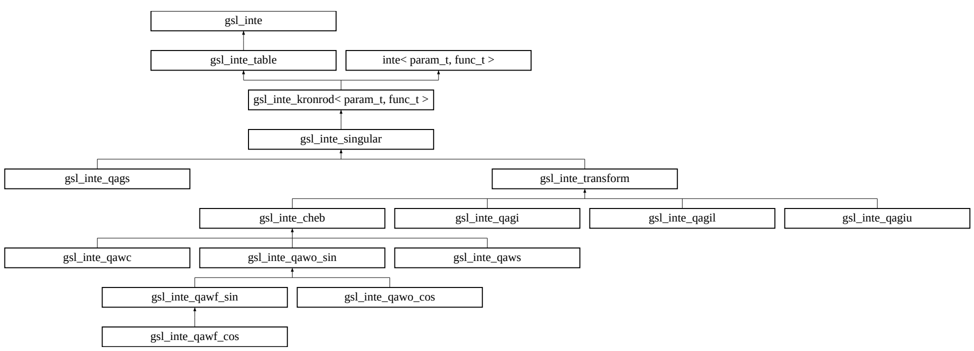
The documentation for this class was generated from the following file:

- gsl_inte_qng.h

7.119 gsl_inte_singular Class Template Reference

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for gsl_inte_singular::



7.119.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_singular< param_t, func_t >

Base class for integrating a function with a singularity (GSL).

This class contains the extrapolation table mechanics and the base integration function for singular integrals from GSL. The casual end-user should use gsl_inte_qags, gsl_inte_qagil, and gsl_inte_qagiu for the actual integration.

Definition at line 660 of file gsl_inte_qag_b.h.

Protected Member Functions

- void initialise_table (struct extrapolation_table *table)
  Desc.
- void append_table (struct extrapolation_table *table, double y)
  Desc.

- int test_positivity (double result, double resabs)
  Desc.
- void qelg (struct extrapolation_table *table, double *result, double *abserr)
  Desc.
- int large_interval (gsl_integration_workspace *workspace)
  Desc.
- void reset_nrmax (gsl_integration_workspace *workspace)
  Desc.
- int increase_nrmax (gsl_integration_workspace *workspace)
  Desc.
- int qags (func_t &func, const int qn, const double xgk[ ], const double wg[ ], const double wgk[ ], double fv1[ ], double fv2[ ], const double a, const double b, const double l_epsabs, const double l_epsrel, const size_t limit, double *result, double *abserr, param_t &pa)
  Integration function.

Data Structures

- struct extrapolation_table
  A structure for extrapolation for gsl_inte_qags.

7.119.2 Member Function Documentation

7.119.2.1 int qags (func_t & *func*, const int *qn*, const double *xgk*[ ], const double *wg*[ ], const double *wgk*[ ], double *fv1*[ ], double *fv2*[ ], const double *a*, const double *b*, const double *l_epsabs*, const double *l_epsrel*, const size_t *limit*, double * *result*, double * *abserr*, param_t & *pa*) `[inline, protected]`

Integration function.

Idea for future

Remove goto statements?

Output iteration information

Definition at line 915 of file gsl_inte_qag_b.h.

The documentation for this class was generated from the following file:

- gsl_inte_qag_b.h

7.120 gsl_inte_singular::extrapolation_table Struct Reference

```
#include <gsl_inte_qag_b.h>
```

7.120.1 Detailed Description

template<class param_t, class func_t> struct gsl_inte_singular< param_t, func_t >::extrapolation_table

A structure for extrapolation for gsl_inte_qags.

Todo

Improve the documentation

Idea for future

Move this to a new class, with qelg() as a method

Definition at line 672 of file gsl_inte_qag_b.h.

Data Fields

- size_t n
  Desc.
- double rlist2 [52]
  Desc.
- size_t nres
  Desc.
- double res3la [3]
  Desc.

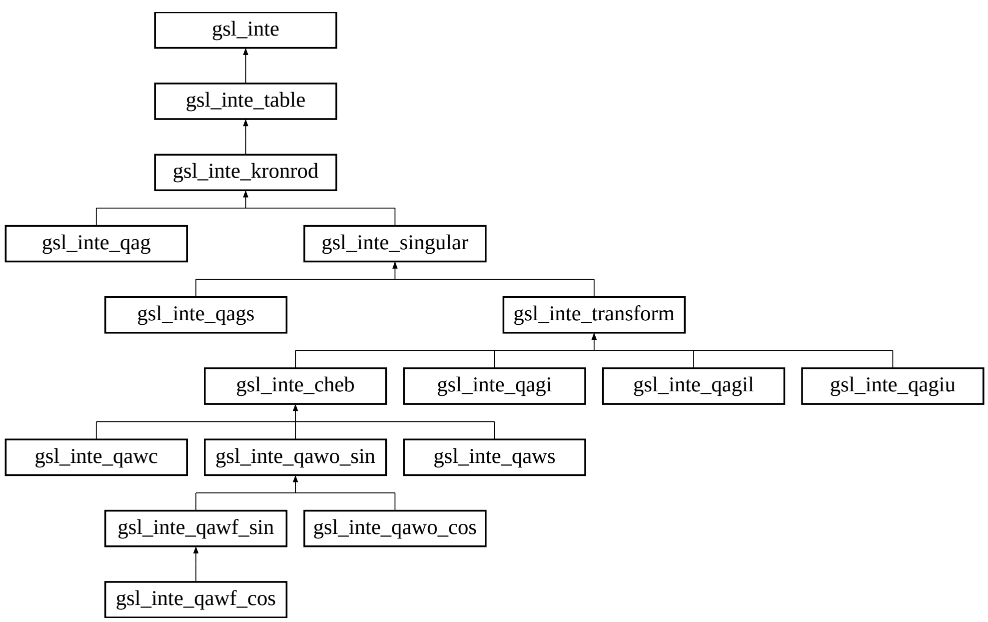
The documentation for this struct was generated from the following file:

- gsl_inte_qag_b.h

7.121 gsl_inte_table Class Reference

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for gsl_inte_table::



7.121.1 Detailed Description

Base routines for the GSL adaptive integration classes.

This class contains several functions for manipulating the GSL integration workspace.

Idea for future

Move gsl_integration_workspace to a separate class and remove this class, making all children direct descendants of gsl_inte instead. We'll have to figure out what to do with the data member `wkspace` though. Some work on this front is already in gsl_inte_qag_b.h.

Definition at line 489 of file gsl_inte_qag_b.h.

Public Member Functions

- int set_wkspace (size_t size)
  Set the integration workspace size.
- void initialise (gsl_integration_workspace ∗workspace, double a, double b)
  Initialize the workspace for an integration with limits `a` *and* `b`*.*
- void set_initial_result (gsl_integration_workspace ∗workspace, double result, double error)
  Set the result at position zero.
- void retrieve (const gsl_integration_workspace ∗workspace, double ∗a, double ∗b, double ∗r, double ∗e)
  Retrieve the ith result from the workspace.
- void qpsrt (gsl_integration_workspace ∗workspace)
  Sort the workspace.
- void update (gsl_integration_workspace ∗workspace, double a1, double b1, double area1, double error1, double a2, double b2, double area2, double error2)
  Update workspace with new results and resort.
- double sum_results (const gsl_integration_workspace ∗workspace)
  Add up all of the contributions to construct the final result.
- int subinterval_too_small (double a1, double a2, double b2)
  Find out if the present subinterval is too small.
- void append_interval (gsl_integration_workspace ∗workspace, double a1, double b1, double area1, double error1)
  Append new results to workspace.

Data Fields

- gsl_integration_workspace ∗ w
  The integration workspace.
- int wkspace
  The size of the integration workspace (default 1000).

7.121.2 Member Function Documentation

7.121.2.1 void retrieve (const gsl_integration_workspace ∗ *workspace*, double ∗ *a*, double ∗ *b*, double ∗ *r*, double ∗ *e*)

Retrieve the ith result from the workspace.

The workspace variable `i` is used to specify which interval is requested.

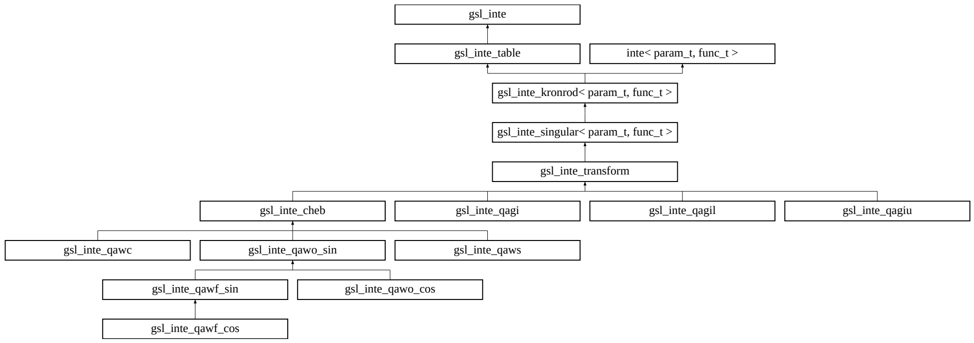
The documentation for this class was generated from the following file:

- gsl_inte_qag_b.h

7.122 gsl_inte_transform Class Template Reference

```
#include <gsl_inte_qag_b.h>
```

Inheritance diagram for gsl_inte_transform::



7.122.1 Detailed Description

template<class param_t, class func_t> class gsl_inte_transform< param_t, func_t >

Integrate a function with a singularity (GSL) [abstract base].

Definition at line 1296 of file gsl_inte_qag_b.h.

Public Member Functions

- virtual double transform (func_t &func, double t, param_t &pa)=0
  The transformation to apply to the user-supplied function.
- virtual void gsl_integration_qk_o2scl (func_t &func, const int n, const double xgk[ ], const double wg[ ], const double wgk[ ], double fv1[ ], double fv2[ ], double a, double b, double ∗result, double ∗abserr, double ∗resabs, double ∗resasc, param_t &pa)
  The basic Gauss-Kronrod integration function.

7.122.2 Member Function Documentation

7.122.2.1 virtual void gsl_integration_qk_o2scl (func_t & *func*, const int *n*, const double *xgk*[ ], const double *wg*[ ], const double *wgk*[ ], double *fv1*[ ], double *fv2*[ ], double *a*, double *b*, double ∗ *result*, double ∗ *abserr*, double ∗ *resabs*, double ∗ *resasc*, param_t & *pa*) `[inline, virtual]`

The basic Gauss-Kronrod integration function.

This is basically just a copy of gsl_inte_qag::gsl_integration_qk_o2scl() which is rewritten to call the internal transformed function rather than directly calling the user-specified function.

Reimplemented from gsl_inte_kronrod.

Definition at line 1313 of file gsl_inte_qag_b.h.

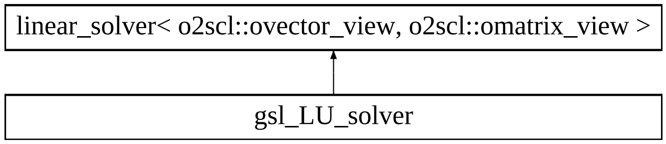
The documentation for this class was generated from the following file:

- gsl_inte_qag_b.h

7.123 gsl_LU_solver Class Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for gsl_LU_solver::



7.123.1 Detailed Description

GSL solver by LU decomposition.

Definition at line 133 of file ode_it_solve.h.

Public Member Functions

- virtual int solve (size_t n, o2scl::omatrix_view &A, o2scl::ovector_view &b, o2scl::ovector_view &x)
  Solve square linear system $Ax = b$ of size $\mathtt{n}$.

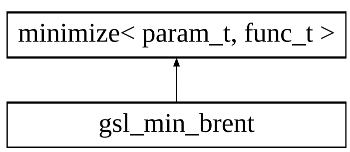
The documentation for this class was generated from the following file:

- ode_it_solve.h

7.124 gsl_min_brent Class Template Reference

```
#include <gsl_min_brent.h>
```

Inheritance diagram for gsl_min_brent::



7.124.1 Detailed Description

template<class param_t, class func_t = funct<param_t>> class gsl_min_brent< param_t, func_t >

One-dimensional minimization using Brent's method (GSL).

The minimization in the function min_bkt() is complete when the bracketed interval is smaller than $\mathrm{tol} = \mathrm{tolx} + \mathrm{tolf} \cdot \mathrm{min}$, where $\mathrm{min} = \min(|\mathrm{lower}|, |\mathrm{upper}|)$.

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.

Definition at line 50 of file gsl_min_brent.h.

Public Member Functions

- int set (func_t &func, double xmin, double lower, double upper, param_t &pa)
  Set the function and the initial brackeing interval.

- int set_with_values (func_t &func, double xmin, double fmin, double lower, double fl, double upper, double fu, param_t &pa)

  Set the function, the initial brackeing interval, and the corresponding function values.
- int iterate ()
  Perform an iteration.
- virtual int min_bkt (double &x2, double x1, double x3, double &fmin, param_t &pa, 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 mound.
- 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, param_t &pa)
  Compute the function values at the various points.

Protected Attributes

- func_t * uf
  The function.
- param_t * up
  The parameters.

Temporary storage

- double **d**
- double **e**
- double **v**
- double **w**
- double **f_v**
- double **f_w**

7.124.2 Member Function Documentation

7.124.2.1 virtual int min_bkt (double & *x2*, double *x1*, double *x3*, double & *fmin*, param_t & *pa*, func_t & *func*) `[inline, virtual]`

Calculate the minimum `fmin` of `func` with `x2` bracketed between `x1` and `x3`.

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< param_t, func_t >.

Definition at line 324 of file gsl_min_brent.h.

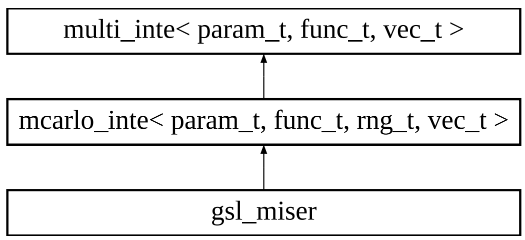
The documentation for this class was generated from the following file:

- gsl_min_brent.h

7.125 gsl_miser Class Template Reference

```
#include <gsl_miser.h>
```

Inheritance diagram for gsl_miser::



7.125.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class rng_t = gsl_rnga, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_miser< param_t, func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

Multidimensional integration using Miser Miser Carlo (GSL).

Todo

Document the fact that min_calls and min_calls_per_bisection need to be set beforehand

Definition at line 47 of file gsl_miser.h.

Public Member Functions

- 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, param_t &pa, 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, param_t &pa, 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, param_t &pa)
  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

- double dither
  Introduce random variation into bisection (default 0.0).
- double estimate_frac
  Specify fraction of function calls for estimating variance.
- double alpha
  How estimated variances for two sub-regions are combined.
- size_t min_calls
  Minimum number of calls to estimate the variance (default 100).
- size_t min_calls_per_bisection
  Minimum number of calls required to proceed with bisection (default 4000).

Protected Member Functions

- virtual int estimate_corrmc (func_t &func, size_t ndim, const vec_t &xl, const vec_t &xu, param_t &pa, size_t calls, double &res, double &err, const double lxmid[ ], double lsigma_l[ ], double lsigma_r[ ])
  Desc.

Protected Attributes

- size_t dim
  Desc.
- double ∗ xmid
  Desc.
- double ∗ sigma_l
  Desc.
- double ∗ sigma_r
  Desc.
- double ∗ fmax_l
  Desc.
- double ∗ fmax_r
  Desc.
- double ∗ fmin_l
  Desc.
- double ∗ fmin_r
  Desc.
- double ∗ fsum_l
  Desc.
- double ∗ fsum_r
  Desc.
- double ∗ fsum2_l
  Desc.
- double ∗ fsum2_r
  Desc.
- size_t ∗ hits_l
  Desc.
- size_t ∗ hits_r
  Desc.
- alloc_t ao
  Memory allocator.
- alloc_vec_t x
  The most recent integration point.

7.125.2 Field Documentation

7.125.2.1 double dither

Introduce random variation into bisection (default 0.0).

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 65 of file gsl_miser.h.

7.125.2.2 double estimate_frac

Specify fraction of function calls for estimating variance.

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 77 of file gsl_miser.h.

7.125.2.3 double alpha

How estimated variances for two sub-regions are combined.

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 100 of file gsl_miser.h.

7.125.2.4 size_t min_calls

Minimum number of calls to estimate the variance (default 100).

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.  The
default value of MIN_CALLS is ‘16 * dim’.
```

Definition at line 116 of file gsl_miser.h.

7.125.2.5 size_t min_calls_per_bisection

Minimum number of calls required to proceed with bisection (default 4000).

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.  The default value of this
parameter is '32 * min_calls'.
```

Definition at line 132 of file gsl_miser.h.

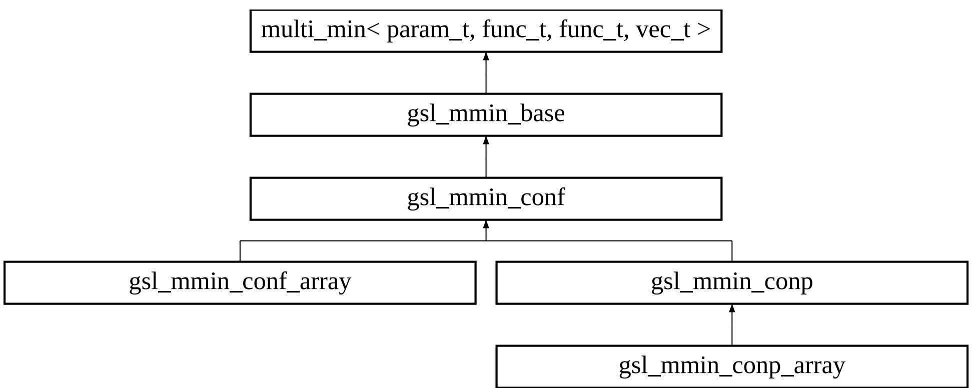
The documentation for this class was generated from the following file:

- gsl_miser.h

7.126 gsl_mmin_base Class Template Reference

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for gsl_mmin_base::



7.126.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,ovector_view>, class auto_grad_t = gradient<param_t,func_-t,ovector_view>, class def_auto_grad_t = simple_grad<param_t,func_t,ovector_view>> class gsl_mmin_base< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

Base minimization routines for gsl_mmin_conf and gsl_mmin_conp.

Default template arguments

- `param_t` - multi_funct
- `vec_t` - ovector_view
- `alloc_vec_t` - ovector

- alloc_t - ovector_alloc
- dfunc_t - grad_funct
- auto_grad_t - gradient
- def_auto_grad_t - simple_grad

Definition at line 53 of file gsl_mmin_conf.h.

Public Member Functions

- int base_set (func_t &ufunc, param_t &pa)
  Set the function.
- int base_set_de (func_t &ufunc, dfunc_t &udfunc, param_t &pa)
  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 gsl_vector ∗x, const gsl_vector ∗px, double stepx, double lambda, gsl_vector ∗x1x, gsl_vector ∗dx)
  Take a step.
- void intermediate_point (const gsl_vector ∗x, const gsl_vector ∗px, double lambda, double pg, double stepa, double stepc, double fa, double fc, gsl_vector ∗x1x, gsl_vector ∗dx, gsl_vector ∗gradient, double ∗stepx, double ∗f)
  Line minimization.
- void minimize (const gsl_vector ∗x, const gsl_vector ∗xp, double lambda, double stepa, double stepb, double stepc, double fa, double fb, double fc, double xtol, gsl_vector ∗x1x, gsl_vector ∗dx1x, gsl_vector ∗x2x, gsl_vector ∗dx2x, gsl_vector ∗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.
- param_t ∗ params
  User-specified parameter.
- size_t dim

Memory size.
- alloc_t ao
  Memory allocation.
- alloc_vec_t avt
  Temporary vector.
- alloc_vec_t avt2
  Temporary vector.

7.126.2 Member Function Documentation

7.126.2.1 void intermediate_point (const gsl_vector ∗ *x*, const gsl_vector ∗ *px*, double *lambda*, double *pg*, double *stepa*, double *stepc*, double *fa*, double *fc*, gsl_vector ∗ *x1x*, gsl_vector ∗ *dx*, gsl_vector ∗ *gradient*, double ∗ *stepx*, double ∗ *f*) `[inline, protected]`

Line minimization.

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 102 of file gsl_mmin_conf.h.

7.126.2.2 void minimize (const gsl_vector ∗ *x*, const gsl_vector ∗ *xp*, double *lambda*, double *stepa*, double *stepb*, double *stepc*, double *fa*, double *fb*, double *fc*, double *xtol*, gsl_vector ∗ *x1x*, gsl_vector ∗ *dx1x*, gsl_vector ∗ *x2x*, gsl_vector ∗ *dx2x*, gsl_vector ∗ *gradient*, double ∗ *xstep*, double ∗ *f*, double ∗ *gnorm_u*) `[inline, protected]`

Perform the minimization.

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 153 of file gsl_mmin_conf.h.

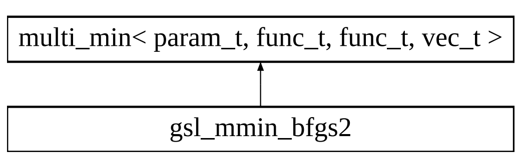
The documentation for this class was generated from the following file:

- gsl_mmin_conf.h

7.127 gsl_mmin_bfgs2 Class Template Reference

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for gsl_mmin_bfgs2::



7.127.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,ovector_view>, class auto_grad_t = gradient<param_t,func_t,ovector_view>, class def_auto_grad_t = simple_grad<param_t,func_t,ovector_view>> class gsl_mmin_bfgs2< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

Multidimensional minimization by the BFGS algorithm (GSL).

This class includes the optimizations from the GSL minimizer `vector_bfgs2`.

Todo

Works with generic vector objects, but doesn't allow specification of jacobian yet.

Definition at line 391 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, param_t &upa)
  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, param_t &upa)
  Set the function, the gradient, and the initial guess.
- virtual int mmin (size_t nn, vec_t &xx, double &fmin, param_t &pa, 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, param_t &pa, 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.
- double lmin_tol
  The tolerance for the 1-dimensional minimizer.

Protected Attributes

- gsl_mmin_linmin lm
  The line minimizer.
- size_t dim
  Memory size.
- alloc_t ao
  Memory allocation.

The original variables from the GSL state structure

- int **iter**
- double **step**
- double **g0norm**
- double **pnorm**
- double **delta_f**
- double **fp0**
- gsl_vector ∗ **x0**
- gsl_vector ∗ **g0**
- gsl_vector ∗ **p**
- gsl_vector ∗ **dx0**
- gsl_vector ∗ **dg0**

- gsl_mmin_wrapper< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_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**
- gsl_vector ∗ **st_dx**
- gsl_vector ∗ **st_grad**
- double **st_f**

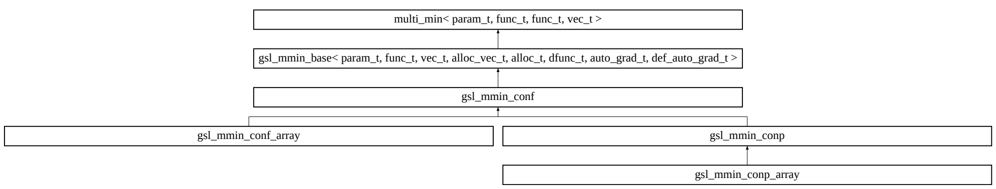
The documentation for this class was generated from the following file:

- gsl_mmin_bfgs2.h

7.128 gsl_mmin_conf Class Template Reference

```
#include <gsl_mmin_conf.h>
```

Inheritance diagram for gsl_mmin_conf::



7.128.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,ovector_view>, class auto_grad_t = gradient<param_t,func_-t,ovector_view>, class def_auto_grad_t = simple_grad<param_t,func_t,ovector_view>> class gsl_mmin_conf< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

Multidimensional minimization by the Fletcher-Reeves conjugate gradient algorithm (GSL).

The variable multi_min::tolf is used as the maximum value of the gradient and is 10^{-4} be default.

The gsl iterate() function for this minimizer chooses to return `GSL_ENOPROG` if the iteration fails to make progress without calling the error handler. This is presumably because the iterate() function can fail to make progress when the algorithm has succeeded in finding the minimum. I prefer to return a non-zero value from a function only in cases where the error handler will also be called, so the user is clear on what an "error" means in the local context. Thus if iterate() is failing to make progress, instead of returning a non-zero value, it sets the value of it_info to a non-zero value.

Todo

A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed, probably by rewriting take_step to produce a vec_t &x1 rather than a gsl_vector ∗x1;

Those who look in detail at the code will 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.

Default template arguments

- `param_t` - multi_funct
- `vec_t` - ovector_view
- `alloc_vec_t` - ovector
- `alloc_t` - ovector_alloc
- `dfunc_t` - grad_funct
- `auto_grad_t` - gradient
- `def_auto_grad_t` - simple_grad

Definition at line 384 of file gsl_mmin_conf.h.

Public Member Functions

- virtual int iterate ()
  Perform an iteration.
- virtual const char * type ()
  Return string denoting type("gsl_mmin_conf").
- 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, param_t &pa)
  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, param_t &pa)
  Set the function and initial guess.
- virtual int mmin (size_t nn, vec_t &xx, double &fmin, param_t &pa, 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, param_t &pa, 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).
- int it_info
  Information from the last call to iterate().

Protected Attributes

- alloc_vec_t avt5
  Temporary vector.
- alloc_vec_t avt6
  Temporary vector.
- alloc_vec_t avt7
  Temporary vector.
- alloc_vec_t avt8
  Temporary vector.

The original variables from the GSL state structure

- int iter
  Desc.
- double step
  Desc.
- double tol
  Desc.
- gsl_vector ∗ x1
  Desc.
- gsl_vector ∗ dx1
  Desc.
- gsl_vector ∗ x2
  Desc.
- double pnorm
  Desc.
- gsl_vector ∗ p
  Desc.
- double g0norm
  Desc.
- gsl_vector ∗ g0
  Desc.

Store the arguments to set() so we can use them for iterate()

- gsl_vector ∗ ugx
  Desc.
- gsl_vector ∗ ugg
  Desc.
- gsl_vector ∗ udx
  Desc.
- double it_min
  Desc.

7.128.2 Member Function Documentation

7.128.2.1 virtual int set (vec_t & *x*, double *u_step_size*, double *tol_u*, func_t & *ufunc*, param_t & *pa*) `[inline, virtual]`

Set the function and initial guess.

Evaluate the function and its gradient

Definition at line 688 of file gsl_mmin_conf.h.

7.128.2.2 virtual int set_de (vec_t & *x*, double *u_step_size*, double *tol_u*, func_t & *ufunc*, dfunc_t & *udfunc*, param_t & *pa*) `[inline, virtual]`

Set the function and initial guess.

Evaluate the function and its gradient

Definition at line 724 of file gsl_mmin_conf.h.

7.128.3 Field Documentation

7.128.3.1 int it_info

Information from the last call to iterate().

This is 1 if `pnorm` or `gnorm` are 0 and 2 if `stepb` is zero.

Todo

Document this better

Definition at line 464 of file gsl_mmin_conf.h.

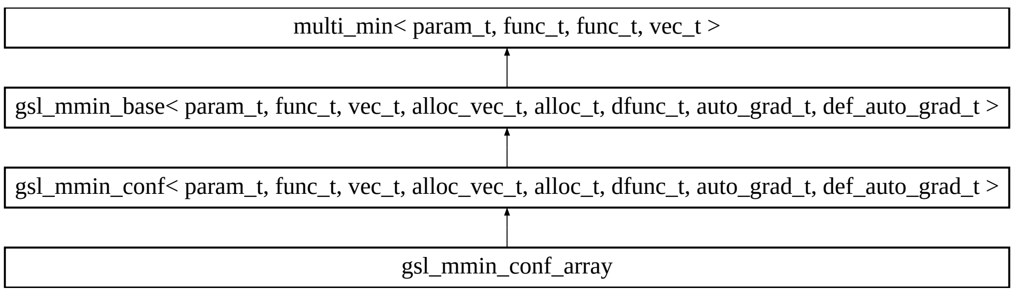
The documentation for this class was generated from the following file:

- gsl_mmin_conf.h

7.129 gsl_mmin_conf_array Class Template Reference

#include <gsl_mmin_conf.h>

Inheritance diagram for gsl_mmin_conf_array::



7.129.1 Detailed Description

template<class param_t, size_t nv> class gsl_mmin_conf_array< param_t, nv >

An array version of gsl_mmin_conf.

Definition at line 871 of file gsl_mmin_conf.h.

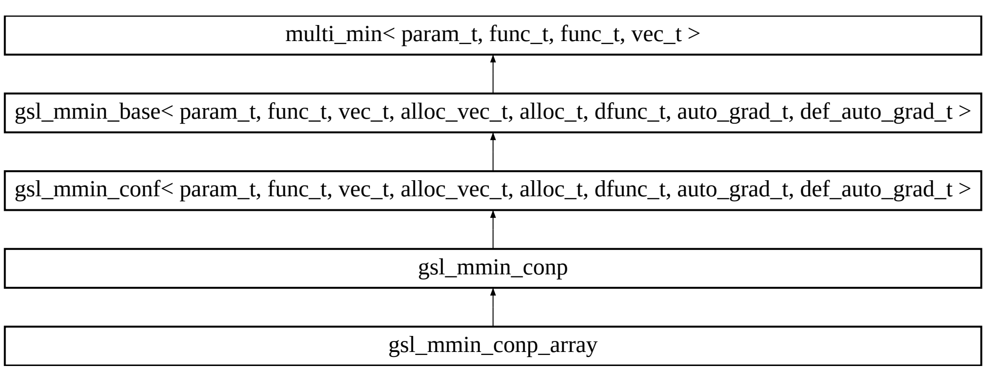
The documentation for this class was generated from the following file:

- gsl_mmin_conf.h

7.130 gsl_mmin_conp Class Template Reference

#include <gsl_mmin_conp.h>

Inheritance diagram for gsl_mmin_conp::



7.130.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,ovector_view>, class auto_grad_t = gradient<param_t,func_t,ovector_view>, class def_auto_grad_t = simple_grad<param_t,func_t,ovector_view>> class gsl_mmin_conp< param_t, func_t, vec_t, alloc_vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

Multidimensional minimization by the Polak-Ribiere conjugate gradient algorithm (GSL).

The variable multi_min::tolf is used as the maximum value of the gradient and is 10^{-4} be default.

The gsl iterate() function for this minimizer chooses to return `GSL_ENOPROG` if the iteration fails to make progress without calling the error handler. This is presumably because the iterate() function can fail to make progress when the algorithm has succeeded in finding the minimum. I prefer to return a non-zero value from a function only in cases where the error handler will also be called, so the user is clear on what an "error" means in the local context. Thus if iterate() is failing to make progress, instead of returning a non-zero value, it sets the value of it_info to a non-zero value.

Todo

A bit of needless copying is required in the function wrapper to convert from `gsl_vector` to the templated vector type. This can be fixed.

Todo

Document stopping conditions

Definition at line 63 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").

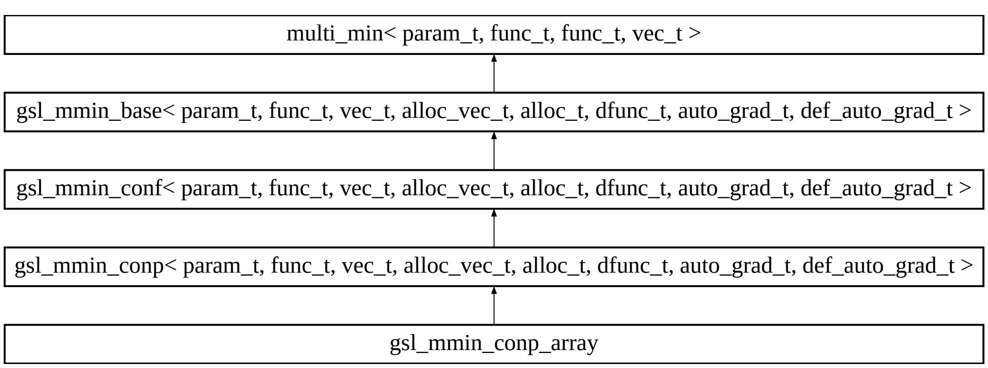
The documentation for this class was generated from the following file:

- gsl_mmin_conp.h

7.131 gsl_mmin_conp_array Class Template Reference

```
#include <gsl_mmin_conp.h>
```

Inheritance diagram for gsl_mmin_conp_array::



7.131.1 Detailed Description

template<class param_t, size_t nv> class gsl_mmin_conp_array< param_t, nv >

An array version of gsl_mmin_conp.

Definition at line 189 of file gsl_mmin_conp.h.

The documentation for this class was generated from the following file:

- gsl_mmin_conp.h

7.132 gsl_mmin_linmin Class Reference

```
#include <gsl_mmin_bfgs2.h>
```

7.132.1 Detailed Description

The line minimizer for gsl_mmin_bfgs2.

Definition at line 317 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.

7.132.2 Member Function Documentation

7.132.2.1 double interp_quad (double *f0*, double *fp0*, double *f1*, double *zl*, double *zh*) `[protected]`

Minimize the interpolating quadratic.

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

7.132.2.2 double cubic (double *c0*, double *c1*, double *c2*, double *c3*, double *z*) `[protected]`

Minimize the interpolating cubic.

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).

7.132.2.3 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.

Recommended values from Fletcher are rho = 0.01, sigma = 0.1, tau1 = 9, tau2 = 0.05, tau3 = 0.5

Todo

Properly reference Fletcher here.

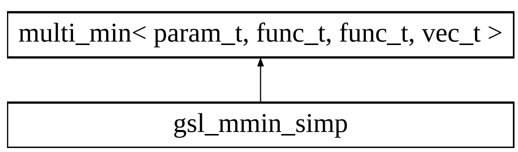
The documentation for this class was generated from the following file:

- gsl_mmin_bfgs2.h

7.133 gsl_mmin_simp Class Template Reference

```
#include <gsl_mmin_simp.h>
```

Inheritance diagram for gsl_mmin_simp::



7.133.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_mmin_simp< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Multidimensional minimization by the Simplex method (GSL).

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.

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.

Definition at line 68 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, param_t &pa, 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, param_t &pa, 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, param_t &pa, 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, param_t &pa, 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, param_t &pa, 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, param_t &pa, 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, param_t &pa)
  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.
- param_t * params
  Parameters.
- 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).

7.133.2 Member Function Documentation

7.133.2.1 double nmsimplex_size () `[inline, protected]`

Compute the size of the simplex.

Calculates simplex size as average sum of length of vectors from simplex center to corner points:

$(1/n) \sum ||y - y_{\mathrm{middlepoint}}||$

Definition at line 118 of file gsl_mmin_simp.h.

7.133.2.2 virtual int move_corner_err (const double *coeff*, size_t *corner*, vec_t & *xc*, func_t & *f*, size_t *nvar*, param_t & *pa*, double & *newval*) `[inline, protected, virtual]`

Move a corner of a simplex.

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 140 of file gsl_mmin_simp.h.

7.133.2.3 virtual int contract_by_best (size_t *best*, vec_t & *xc*, func_t & *f*, size_t *nvar*, param_t & *pa*) `[inline, protected, virtual]`

Contract the simplex towards the best point.

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 169 of file gsl_mmin_simp.h.

7.133.2.4 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*) `[inline, virtual]`

Print out iteration information.

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 687 of file gsl_mmin_simp.h.

7.133.3 Field Documentation

7.133.3.1 bool avoid_nonzero `[protected]`

If true, try to automatically avoid regions where the function returns a non-zero value (default false).

Note:

This option doesn't work yet, so I've made the variable protected to prevent the user from changing it.

Definition at line 241 of file gsl_mmin_simp.h.

7.133.3.2 int print_simplex

Print simplex information in print_iter() (default 0).

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 288 of file gsl_mmin_simp.h.

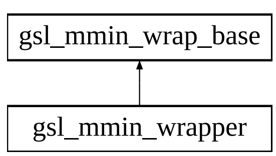
The documentation for this class was generated from the following file:

- gsl_mmin_simp.h

7.134 gsl_mmin_wrap_base Class Reference

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for gsl_mmin_wrap_base::



7.134.1 Detailed Description

Virtual base for the gsl_mmin_bfgs2 wrapper.

This 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, void ∗params)=0
  Function.
- virtual double wrap_df (double alpha, void ∗params)=0
  Derivative.
- virtual void wrap_fdf (double alpha, void ∗params, double ∗f, double ∗df)=0
  Function and derivative.

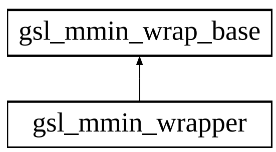
The documentation for this class was generated from the following file:

- gsl_mmin_bfgs2.h

7.135 gsl_mmin_wrapper Class Template Reference

```
#include <gsl_mmin_bfgs2.h>
```

Inheritance diagram for gsl_mmin_wrapper::



7.135.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class dfunc_t = grad_funct<param_t,ovector_view>, class auto_grad_t = gradient<param_t,func_t,ovector_view>, class def_auto_grad_t = simple_grad<param_t,func_t,ovector_view>> class gsl_mmin_wrapper< param_t, func_t, vec_t, alloc_-vec_t, alloc_t, dfunc_t, auto_grad_t, def_auto_grad_t >

Wrapper class for the gsl_mmin_bfgs2 minimizer.

This is a reimplmentation of the internal GSL wrapper for function calls in the BFGS minimizer.

Idea for future

There's a bit of extra vector copying here which could potentially be avoided.

Definition at line 69 of file gsl_mmin_bfgs2.h.

Public Member Functions

- void prepare_wrapper (func_t &ufunc, param_t &upa, gsl_vector ∗t_x, double f, gsl_vector ∗t_g, gsl_vector ∗t_p)
  Initialize wrapper.
- void update_position (double alpha, vec_t &t_x, double ∗t_f, gsl_vector ∗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, void ∗params)
  Evaluate the function.
- virtual double wrap_df (double alpha, void ∗params)
  Evaluate the derivative.
- int simple_df (vec_t &x2, vec_t &g2)
  A simple derivative.
- virtual void wrap_fdf (double alpha, void ∗params, double ∗f, double ∗df)
  Evaluate the function and the derivative.

Protected Attributes

- func_t ∗ func
  Function.
- dfunc_t ∗ dfunc
  Derivative.
- auto_grad_t ∗ agrad
  Test.
- param_t ∗ pa
  Parameters.

fixed values

- gsl_vector ∗ **x**
- gsl_vector ∗ **g**
- gsl_vector ∗ **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**

7.135.2 Member Function Documentation

7.135.2.1 void change_direction () `[inline]`

Convert cache values to the new minimizer direction.

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 290 of file gsl_mmin_bfgs2.h.

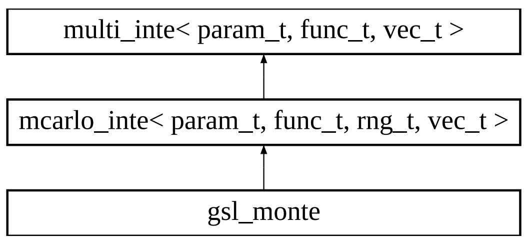
The documentation for this class was generated from the following file:

- gsl_mmin_bfgs2.h

7.136 gsl_monte Class Template Reference

```
#include <gsl_monte.h>
```

Inheritance diagram for gsl_monte::



7.136.1 Detailed Description

template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_monte< param_t, func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

Multidimensional integration using plain Monte Carlo (GSL).

Definition at line 43 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, param_t &pa, 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, param_t &pa)
  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").

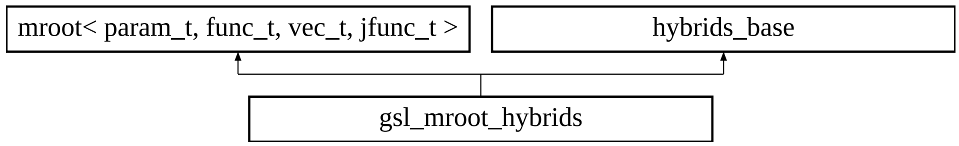
The documentation for this class was generated from the following file:

- gsl_monte.h

7.137 gsl_mroot_hybrids Class Template Reference

```
#include <gsl_mroot_hybrids.h>
```

Inheritance diagram for gsl_mroot_hybrids::



7.137.1 Detailed Description

template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_view, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc, class jfunc_t = jac_funct<param_t,vec_t,mat_t>> class gsl_mroot_hybrids< param_t, func_t, vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t, jfunc_t >

Multidimensional root-finding algorithm using Powell's Hybrid method (GSL).

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. 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::tolf 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 Multidimensional solver example .

Definition at line 322 of file gsl_mroot_hybrids.h.

Public Member Functions

- virtual int set_jacobian (jacobian< param_t, 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, param_t &pa, 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, param_t &pa, func_t &ufunc)
  Solve `ufunc` *using* `xx` *as an initial guess, returning* `xx`.
- int set (size_t nn, vec_t &ax, func_t &ufunc, param_t &pa)
  Set the function, the parameters, and the initial guess.
- int set_de (size_t nn, vec_t &ax, func_t &ufunc, jfunc_t &dfunc, param_t &pa)
  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< param_t, 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

- void compute_Rg (size_t N, const gsl_matrix ∗r, const gsl_vector ∗gradient, vec_t &Rg)
  Desc.
- void compute_wv (size_t n, const gsl_vector ∗qtdf, const gsl_vector ∗rdx, const vec_t &dxx, const gsl_vector ∗diag, double pnorm, gsl_vector ∗w, gsl_vector ∗v)
  Desc.
- void compute_rdx (size_t N, const gsl_matrix ∗r, const vec_t &dxx, gsl_vector ∗rdx)
  Desc.
- double scaled_enorm_tvec (size_t n, const gsl_vector ∗d, const vec_t &ff)
  Desc.
- double compute_delta (size_t n, gsl_vector ∗diag, vec_t &xx)
  Desc.
- double enorm_tvec (const vec_t &ff)
  Desc.
- int compute_trial_step_tvec (size_t N, vec_t &xl, vec_t &dxl, vec_t &xx_trial)
  Desc.
- int compute_df_tvec (size_t n, const vec_t &ff_trial, const vec_t &fl, gsl_vector ∗dfl)
  Desc.
- void compute_diag_tvec (size_t n, const mat_t &J, gsl_vector ∗diag)
  Desc.
- void compute_qtf_tvec (size_t N, const gsl_matrix ∗q, const vec_t &ff, gsl_vector ∗qtf)
  Desc.
- void update_diag_tvec (size_t n, const mat_t &J, gsl_vector ∗diag)
  Desc.
- void scaled_addition_tvec (size_t N, double alpha, gsl_vector ∗newton, double beta, gsl_vector ∗gradient, vec_t &pp)
  Desc.
- int dogleg (size_t n, const gsl_matrix ∗r, const gsl_vector ∗qtf, const gsl_vector ∗diag, double delta, gsl_vector ∗newton, gsl_vector ∗gradient, vec_t &p)
  Take a dogleg step.
- int solve_set (size_t nn, vec_t &xx, param_t &pa, func_t &ufunc)
  Finish the solution after set() or set_de() has been called.

Protected Attributes

- jfunc_t ∗ jac
  Pointer to the user-specified Jacobian object.
- jacobian< param_t, func_t, vec_t, mat_t > ∗ ajac
  Pointer to the automatic Jacobian object.
- alloc_t ao
  Memory allocator for objects of type `alloc_vec_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.
- o2scl_hybrid_state_t< vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t > state
  The solver state.
- param_t * params
  The function parameters.
- size_t dim
  The number of equations and unknowns.
- bool jac_given
  True if the jacobian has been given.
- func_t * fnewp
  Pointer to the user-specified function.
- bool set_called
  True if "set" has been called.

7.137.2 Member Function Documentation

7.137.2.1 int iterate () `[inline]`

Perform an iteration.

At the end of the iteration, the current value of the solution is stored in x.

Definition at line 714 of file gsl_mroot_hybrids.h.

7.137.2.2 virtual int msolve_de (size_t *nn*, vec_t & *xx*, param_t & *pa*, func_t & *ufunc*, jfunc_t & *dfunc*) `[inline, virtual]`

Solve `func` with derivatives `dfunc` using `x` as an initial guess, returning `x`.

Reimplemented from mroot.

Definition at line 975 of file gsl_mroot_hybrids.h.

7.137.3 Field Documentation

7.137.3.1 bool shrink_step

If true, iterate() will shrink the step-size automatically if the function returns a non-zero value (default true).

The original GSL behavior can be obtained by setting this to `false`.

Definition at line 679 of file gsl_mroot_hybrids.h.

The documentation for this class was generated from the following file:

- gsl_mroot_hybrids.h

7.138 gsl_ode_control Class Template Reference

`#include <gsl_astep.h>`

7.138.1 Detailed Description

template<class vec_t> class gsl_ode_control< vec_t >

Control structure for gsl_astep.

This class implements both the "standard" and "scaled" step control methods from GSL. The standard control method is the default. To use the scaled controle, 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,

$$D_i = \mathrm{eps\_abs} + \mathrm{eps\_rel} + (\mathrm{a\_y}|y_i| + \mathrm{a\_dydt}h|y'_i|)$$

while in the scaled version the user specifies the scale for each component, s_i,

$$D_i = \mathrm{eps\_abs}s_i + \mathrm{eps\_rel} + (\mathrm{a\_y}|y_i| + \mathrm{a\_dydt}h|y'_i|)$$

The desired error level D_i is compared to then observed error $E_i = |\mathrm{yerr}_i|$. 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,

$$h_{\mathrm{new}} = S h_{\mathrm{old}} \left(\frac{E}{D}\right)^{-1/q}$$

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,

$$h_{\mathrm{new}} = S h_{\mathrm{old}} \left(\frac{E}{D}\right)^{-1/(q+1)}$$

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.

Definition at line 92 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).

Protected Attributes

- size_t sdim
  Number of scalings.
- double ∗ scale_abs
  Scalings.

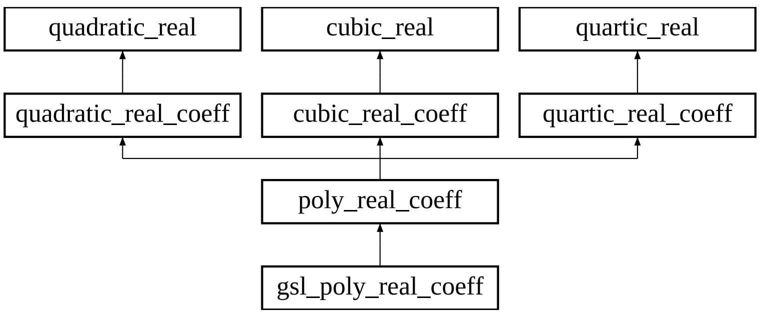
The documentation for this class was generated from the following file:

- gsl_astep.h

7.139 gsl_poly_real_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for gsl_poly_real_coeff::



7.139.1 Detailed Description

Solve a general polynomial with real coefficients (GSL).

Definition at line 530 of file poly.h.

Public Member Functions

- virtual int solve_rc (int n, const double co[ ], std::complex< double > ro[ ])
  Solve the n-th order polynomial.
- 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 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_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* $x = x_4$.
- 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.

7.139.2 Member Function Documentation

7.139.2.1 virtual int solve_rc (int *n*, const double *co*[ ], std::complex< double > *ro*[ ]) `[virtual]`

Solve the n-th order polynomial.

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].

Implements poly_real_coeff.

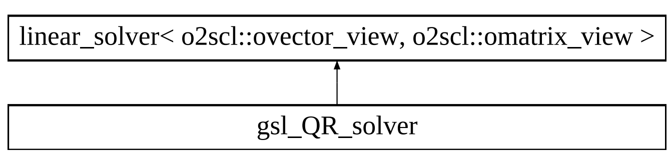
The documentation for this class was generated from the following file:

- poly.h

7.140 gsl_QR_solver Class Reference

`#include <ode_it_solve.h>`

Inheritance diagram for gsl_QR_solver::



7.140.1 Detailed Description

GSL solver by QR decomposition.

Definition at line 156 of file ode_it_solve.h.

Public Member Functions

- virtual int solve (size_t n, o2scl::omatrix_view &A, o2scl::ovector_view &b, o2scl::ovector_view &x)
  Solve square linear system $Ax = b$ of size n.

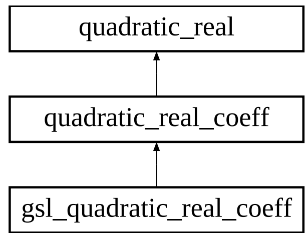
The documentation for this class was generated from the following file:

- ode_it_solve.h

7.141 gsl_quadratic_real_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for gsl_quadratic_real_coeff::



7.141.1 Detailed Description

Solve a quadratic with real coefficients and complex roots (GSL).

Definition at line 442 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 $a_2 x^2 + b_2 x + c_2 = 0$ 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").

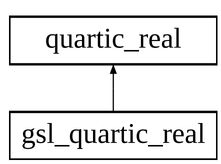
The documentation for this class was generated from the following file:

- poly.h

7.142 gsl_quartic_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for gsl_quartic_real::



7.142.1 Detailed Description

Solve a quartic with real coefficients and real roots (GSL).

Definition at line 490 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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 ("gsl_quartic_real").

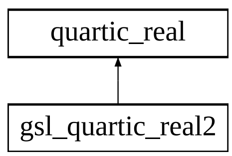
The documentation for this class was generated from the following file:

- poly.h

7.143 gsl_quartic_real2 Class Reference

```
#include <poly.h>
```

Inheritance diagram for gsl_quartic_real2::



7.143.1 Detailed Description

Solve a quartic with real coefficients and real roots (GSL).

Todo

Document the distinction between this class and gsl_quartic_real

Definition at line 512 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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 ("gsl_quartic_real2").

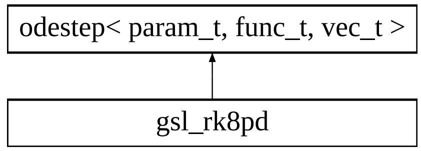
The documentation for this class was generated from the following file:

- poly.h

7.144 gsl_rk8pd Class Template Reference

```
#include <gsl_rk8pd.h>
```

Inheritance diagram for gsl_rk8pd::



7.144.1 Detailed Description

template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rk8pd< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Embedded Runge-Kutta Prince-Dormand ODE stepper (GSL).

Definition at line 39 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, param_t &pa, 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]

7.144.2 Member Function Documentation

7.144.2.1 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*, param_t & *pa*, func_t & *derivs*) `[inline, virtual]`

Perform an integration step.

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` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

Implements odestep.

Definition at line 229 of file gsl_rk8pd.h.

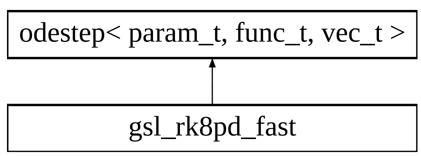
The documentation for this class was generated from the following file:

- gsl_rk8pd.h

7.145 gsl_rk8pd_fast Class Template Reference

```
#include <gsl_rk8pd.h>
```

Inheritance diagram for gsl_rk8pd_fast::



7.145.1 Detailed Description

template<size_t N, class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rk8pd_fast< N, param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Faster embedded Runge-Kutta Prince-Dormand ODE stepper (GSL).

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.

Definition at line 399 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, param_t &pa, 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]

7.145.2 Member Function Documentation

7.145.2.1 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*, param_t & *pa*, func_t & *derivs*) `[inline, virtual]`

Perform an integration step.

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` 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 odestep.

Definition at line 601 of file gsl_rk8pd.h.

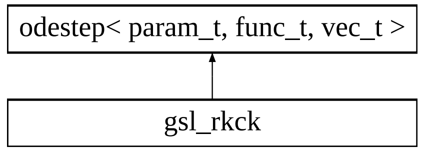
The documentation for this class was generated from the following file:

- gsl_rk8pd.h

7.146 gsl_rkck Class Template Reference

`#include <gsl_rkck.h>`

Inheritance diagram for gsl_rkck::



7.146.1 Detailed Description

template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rkck< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Cash-Karp embedded Runge-Kutta ODE stepper (GSL).

Definition at line 39 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, param_t &pa, 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**

7.146.2 Member Function Documentation

7.146.2.1 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*, param_t & *pa*, func_t & *derivs*) `[inline, virtual]`

Perform an integration step.

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` using function `derivs` to calculate derivatives. The parameters `yout` and `y` and the parameters `dydx_out` and `dydx` may refer to the same object.

Implements odestep.

Definition at line 126 of file gsl_rkck.h.

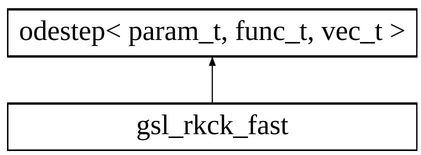
The documentation for this class was generated from the following file:

- gsl_rkck.h

7.147 gsl_rkck_fast Class Template Reference

```
#include <gsl_rkck.h>
```

Inheritance diagram for gsl_rkck_fast::



7.147.1 Detailed Description

template<size_t N, class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_rkck_fast< N, param_t, func_t, vec_t, alloc_vec_t, alloc_t >

Faster Cash-Karp embedded Runge-Kutta ODE stepper (GSL).

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.

Definition at line 217 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, param_t &pa, 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**

7.147.2 Member Function Documentation

7.147.2.1 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*, param_t & *pa*, func_t & *derivs*) `[inline, virtual]`

Perform an integration step.

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` 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 odestep.

Definition at line 309 of file gsl_rkck.h.

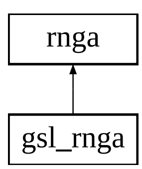
The documentation for this class was generated from the following file:

- gsl_rkck.h

7.148 gsl_rnga Class Reference

`#include <gsl_rnga.h>`

Inheritance diagram for gsl_rnga::



7.148.1 Detailed Description

Random number generator (GSL).

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.

Definition at line 42 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 rng 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 $[0, \max - 1]$.
- gsl_rng * get_gsl_rng ()
  Return a pointer to the gsl_rng object (deprecated).
- 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.

7.148.2 Member Function Documentation

7.148.2.1 gsl_rng* get_gsl_rng ()

Return a pointer to the gsl_rng object (deprecated).

Used in gsl_miser and gsl_anneal.

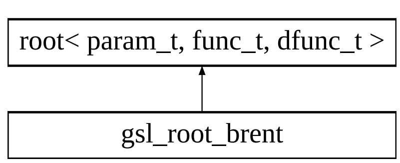
The documentation for this class was generated from the following file:

- gsl_rnga.h

7.149 gsl_root_brent Class Template Reference

```
#include <gsl_root_brent.h>
```

Inheritance diagram for gsl_root_brent::



7.149.1 Detailed Description

template<class param_t, class func_t = funct<void *>, class dfunc_t = func_t> class gsl_root_brent< param_t, func_t, dfunc_t >

One-dimensional root-finding (GSL).

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::tolx. If test_form is 1, then the function stops when the residual is less than root::tolf. If test_form is 2, then both tests are applied.

Todo

There is some duplication in the variables `x_lower`, `x_upper`, `a`, and `b`, which could be removed.

Definition at line 54 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, param_t &pa, 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, param_t &pa)
  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.
- param_t * params
  The function parameters.

Storage for solver state

- double **a**
- double **b**
- double **c**
- double **d**
- double **e**
- double **fa**
- double **fb**
- double **fc**

7.149.2 Member Function Documentation

7.149.2.1 int iterate (func_t & *f*) [inline]

Perform an iteration.

This function always returns gsl_success.

Definition at line 74 of file gsl_root_brent.h.

7.149.2.2 virtual int solve_bkt (double & *x1*, double *x2*, param_t & *pa*, func_t & *f*) [inline, virtual]

Solve `func` in region $x_1 < x < x_2$ returning x_1.

Test the bracket size

Test the residual

Test the bracket size and the residual

Reimplemented from root.

Definition at line 186 of file gsl_root_brent.h.

7.149.2.3 int set (func_t & *ff*, double *lower*, double *upper*, param_t & *pa*) [inline]

Set the information for the solver.

This function always returns gsl_success.

Definition at line 298 of file gsl_root_brent.h.

The documentation for this class was generated from the following file:

- gsl_root_brent.h

7.150 gsl_root_stef Class Template Reference

```
#include <gsl_root_stef.h>
```

7.150.1 Detailed Description

template<class param_t, class func_t, class dfunc_t> class gsl_root_stef< param_t, func_t, dfunc_t >

Steffenson equation solver (GSL).

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 $|root - x| < tol$, where $tol = tolx + tolf2root$.

If test_residual is set to true, then the solver additionally requires that the absolute value of the function is less than root::tolf.

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.

Definition at line 61 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, param_t &pa, 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, param_t &pa)
  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.
- param_t ∗ params
  The function parameters.

7.150.2 Member Function Documentation

7.150.2.1 int iterate () `[inline]`

Perform an iteration.

After a successful iteration, root contains the most recent value of the root.

Definition at line 115 of file gsl_root_stef.h.

7.150.2.2 int set (func_t & *fun*, dfunc_t & *dfun*, double *guess*, param_t & *pa*) [inline]

Set the information for the solver.

Set the function, the derivative, the initial guess and the parameters.

Definition at line 229 of file gsl_root_stef.h.

The documentation for this class was generated from the following file:

- gsl_root_stef.h

7.151 gsl_series Class Reference

```
#include <gsl_series.h>
```

7.151.1 Detailed Description

Series acceleration by Levin u-transform (GSL).

Given an array of terms in a sum, this attempts to evaluate the entire sum with an estimate of the error.

Todo

Covert to use a more general vector

Definition at line 43 of file gsl_series.h.

Public Member Functions

- gsl_series (int size=1)
  `size` *is the number of terms in the series*
- double series_accel (double ∗x, double &err)
  Return the accelerated sum of the series with a simple error estimate.
- double series_accel_err (double ∗x, double &err)
  Return the accelerated sum of the series with an accurate error estimate.
- int set_size (int new_size)
  Set the number of terms.

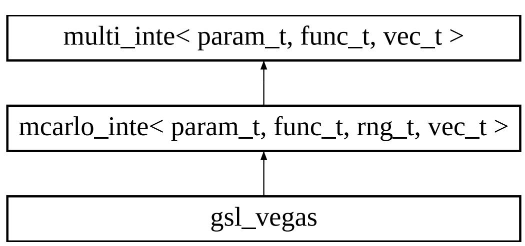
The documentation for this class was generated from the following file:

- gsl_series.h

7.152 gsl_vegas Class Template Reference

```
#include <gsl_vegas.h>
```

Inheritance diagram for gsl_vegas::



7.152.1 Detailed Description

template<class param_t, class func_t = multi_funct<param_t>, class rng_t = gsl_rnga, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class gsl_vegas< param_t, func_t, rng_t, vec_t, alloc_vec_t, alloc_t >

Multidimensional integration using Vegas Monte Carlo (GSL).

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.

Original documentation from GSL:

```
This is an implementation of the adaptive Monte-Carlo algorithm
of G. P. Lepage, originally described in J. Comp. Phys. 27, 192(1978).
The current version of the algorithm was described in the Cornell
preprint CLNS-80/447 of March, 1980.

This code follows most closely the c version by D.R.Yennie, coded
in 1984.

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

Need to double check that the verbose output is good for all settings of verbose.

Todo

BINS_MAX and bins_max are somehow duplicates. Fix this.

Todo

Document the member data more carefully

Idea for future

Could convert bins and boxes to a more useful structure

Definition at line 91 of file gsl_vegas.h.

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

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, param_t &pa, 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, param_t &pa, 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, param_t &pa)
  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 (int boxt[ ])
  Initialize box coordinates.
- int change_box_coord (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 (int lbin[ ], double y)
  Add the most recently generated result to the distribution.
- void random_point (vec_t &lx, int lbin[ ], double *bin_vol, const 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**
- size_t **bins_max**
- unsigned int **bins**
- unsigned int **boxes**
- double ∗ **xi**
- double ∗ **xin**
- double ∗ **delx**
- double ∗ **weight**
- double **vol**
- int ∗ **bin**
- int ∗ **box**
- double ∗ **d**
- double **jac**
- double **wtd_int_sum**
- double **sum_wgts**
- double **chi_sum**
- unsigned int **it_start**
- unsigned int **it_num**
- unsigned int **samples**
- unsigned int **calls_per_box**
- alloc_t ao
  Memory allocation object.
- alloc_vec_t x
  Point for function evaluation.

Static Protected Attributes

- static const int BINS_MAX = 50
  Desc.

7.152.2 Member Function Documentation

7.152.2.1 int change_box_coord (int *boxt*[ ]) `[inline, protected]`

Change box coordinates.

Steps through the box coord like {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 192 of file gsl_vegas.h.

7.152.2.2 void accumulate_distribution (int *lbin*[ ], double *y*) `[inline, protected]`

Add the most recently generated result to the distribution.

This is among the member functions that is not virtual because it is part of the innermost loop.

Definition at line 245 of file gsl_vegas.h.

7.152.2.3 void random_point (vec_t & *lx*, int *lbin*[ ], double ∗ *bin_vol*, const int *lbox*[ ], const vec_t & *xl*, const vec_t & *xu*) `[inline, protected]`

Generate a random position in a given box.

Use the random number generator r 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 265 of file gsl_vegas.h.

7.152.2.4 virtual int vegas_minteg_err (int *stage*, func_t & *func*, size_t *ndim*, const vec_t & *xl*, const vec_t & *xu*, param_t & *pa*, double & *res*, double & *err*) `[inline, virtual]`

Integrate function `func` from x=a to x=b.

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.

Definition at line 633 of file gsl_vegas.h.

7.152.3 Field Documentation

7.152.3.1 double alpha

The stiffness of the rebinning algorithm (default 1.5).

This usual range is between 1 and 2.

Definition at line 115 of file gsl_vegas.h.

7.152.3.2 double chisq

The chi-squared per degree of freedom for the weighted estimate of the integral.

From GSL documentation:

```
The value of CHISQ should be close to 1.  A value of CHISQ
which differs significantly from 1 indicates that the values
from different iterations are inconsistent.  In this case the
weighted error will be under-estimated, and further iterations
of the algorithm are needed to obtain reliable results.
```

Definition at line 133 of file gsl_vegas.h.

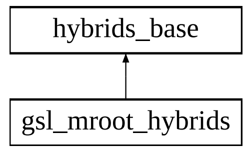
The documentation for this class was generated from the following file:

- gsl_vegas.h

7.153 hybrids_base Class Reference

```
#include <gsl_mroot_hybrids_b.h>
```

Inheritance diagram for hybrids_base::



7.153.1 Detailed Description

Base functions for gsl_mroot_hybrids.

This is a trivial recasting of the functions that were in file scope in the GSL version of the hybrids solver.

Todo

Document the individual functions for this class

Definition at line 46 of file gsl_mroot_hybrids_b.h.

Protected Member Functions

- double enorm (const gsl_vector *f)
  Compute the norm of `f`.
- double scaled_enorm (const gsl_vector *d, const gsl_vector *f)
  Compute the norm of $\vec{f} \cdot \vec{d}$.
- double enorm_sum (const gsl_vector *a, const gsl_vector *b)
  Compute the norm of $\vec{a} + \vec{b}$.
- double compute_actual_reduction (double fnorm, double fnorm1)
  Desc.
- double compute_predicted_reduction (double fnorm, double fnorm1)
  Desc.
- void compute_qtf (const gsl_matrix *q, const gsl_vector *f, gsl_vector *qtf)
  Compute $Q^T f$.
- int newton_direction (const gsl_matrix *r, const gsl_vector *qtf, gsl_vector *p)
  Desc.
- void gradient_direction (const gsl_matrix *r, const gsl_vector *qtf, const gsl_vector *diag, gsl_vector *g)
  Desc.
- void minimum_step (double gnorm, const gsl_vector *diag, gsl_vector *g)
  Desc.

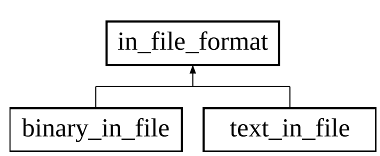
The documentation for this class was generated from the following file:

- gsl_mroot_hybrids_b.h

7.154 in_file_format Class Reference

```
#include <file_format.h>
```

Inheritance diagram for in_file_format::



7.154.1 Detailed Description

Class for input file formats [abstract base].

Definition at line 104 of file file_format.h.

Public Member Functions

- virtual int bool_in (bool &dat, std::string name="")=0
  Input a bool variable.
- virtual int char_in (char &dat, std::string name="")=0
  Input a char variable.
- virtual int double_in (double &dat, std::string name="")=0
  Input a double variable.
- virtual int float_in (float &dat, std::string name="")=0
  Input a float variable.
- virtual int int_in (int &dat, std::string name="")=0
  Input an int variable.
- virtual int long_in (unsigned long int &dat, std::string name="")=0
  Input an long variable.
- virtual int string_in (std::string &dat, std::string name="")=0
  Input a string variable.
- virtual int word_in (std::string &dat, std::string name="")=0
  Input a word variable.
- virtual int start_object (std::string &type, std::string &name)=0
  Start object input.
- virtual int skip_object ()=0
  Skip the present object for the next call to read_type().
- virtual int end_object ()=0
  End object input.
- virtual int init_file ()=0
  Read initialization.
- virtual int clean_up ()=0
  Finish file input.

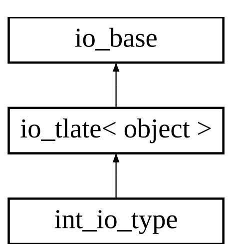
The documentation for this class was generated from the following file:

- file_format.h

7.155 int_io_type Class Reference

`#include <collection.h>`

Inheritance diagram for int_io_type::



7.155.1 Detailed Description

I/O object for int variables.

Definition at line 1739 of file collection.h.

Public Member Functions

- int_io_type (const char *t)
  Desc.
- int addi (collection &co, std::string name, int x, bool overwrt=true)
  Add a int to a collection.
- int geti (collection &co, std::string tname)
  Get a int from a collection.
- int get_def (collection &co, std::string tname, int &op, int def=0)
  Get a int from a collection.

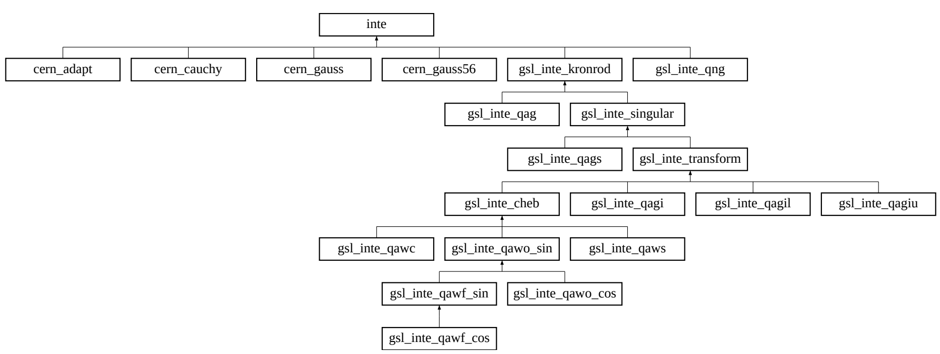
The documentation for this class was generated from the following file:

- collection.h

7.156 inte Class Template Reference

`#include <inte.h>`

Inheritance diagram for inte::



7.156.1 Detailed Description

template<class param_t, class func_t> class inte< param_t, func_t >

Base integration class.

Definition at line 35 of file inte.h.

Public Member Functions

- virtual double integ (func_t &func, double a, double b, param_t &pa)
  Integrate function `func` *from* `a` *to* `b`.
- virtual int integ_err (func_t &func, double a, double b, param_t &pa, double &res, double &err)
  Integrate function `func` *from* `a` *to* `b` *and place the result in* `res` *and the error in* `err`.
- double get_error ()

Return the error in the result from the last call to integ().

- virtual const char * type ()
  Return string denoting type ("inte").

Data Fields

- int verbose
  Verbosity.
- int last_iter
  The most recent number of iterations taken.
- double tolf
  The maximum relative uncertainty in the value of the integral (default 10^{-8}).
- double tolx
  The maximum absolute uncertainty in the value of the integral (default 10^{-8}).

Protected Attributes

- double interror
  The uncertainty for the last integration computation.

7.156.2 Member Function Documentation

7.156.2.1 virtual int integ_err (func_t & *func*, double *a*, double *b*, param_t & *pa*, double & *res*, double & *err*) `[inline, virtual]`

Integrate function `func` from `a` to `b` and place the result in `res` and the error in `err`.

Ideally, if this function succeeds, then `err` should be less than or close to tolf.

Reimplemented in cern_adapt, cern_cauchy, cern_gauss, cern_gauss56, gsl_inte_qag, gsl_inte_qagi, gsl_inte_qagil, gsl_inte_qagiu, gsl_inte_qags, gsl_inte_qawc, gsl_inte_qawf_sin, gsl_inte_qawf_cos, gsl_inte_qawo_sin, gsl_inte_qawo_cos, gsl_inte_qaws, and gsl_inte_qng.

Definition at line 77 of file inte.h.

7.156.2.2 double get_error () `[inline]`

Return the error in the result from the last call to integ().

This will quietly return zero if no integrations have been performed.

Definition at line 88 of file inte.h.

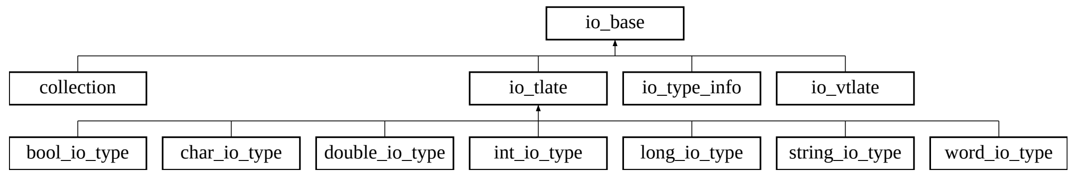
The documentation for this class was generated from the following file:

- inte.h

7.157 io_base Class Reference

`#include <collection.h>`

Inheritance diagram for io_base::



7.157.1 Detailed Description

I/O base class.

This class is necessary so that the collection method source code and the io_base method source code doesn't have to go in header files.

Todo

Should the remove() functions be moved to class collection?

Definition at line 98 of file collection.h.

Public Member Functions

- io_base (int sw=0)
  Create a new I/O object.
- io_base (const char *t)
  Create a new object only if an I/O object for type `t` *is not yet present.*

Functions to be overloaded in descendants of io_base

- virtual const char * type ()
  Return the type of an object.
- virtual bool has_static_data ()
  If true, then the object contains static data.

Functions useful for in in() and out()

- virtual int pointer_in (cinput *co, in_file_format *ins, void **pp, std::string &stype)
  Input a pointer.
- virtual int pointer_out (coutput *co, out_file_format *outs, void *ptr, std::string stype)
  Output an object to `outs` *of type* `stype`.

Protected Member Functions

- virtual int stat_in_noobj (cinput *co, in_file_format *ins)
  Automatically create an object for stat_in.
- virtual int stat_out_noobj (coutput *co, out_file_format *outs)
  Automatically create an object for stat_out.
- virtual int in_wrapper (cinput *co, in_file_format *ins, void *&vp)
  Allocate memory and input an object.
- virtual int in_wrapper (cinput *co, in_file_format *ins, void *&vp, int &sz)
  Allocate memory and input an array of objects.
- virtual int in_wrapper (cinput *co, in_file_format *ins, void *&vp, int &sz, int &sz2)
  Allocate memory and input a 2-d array of objects.
- virtual int out_wrapper (coutput *co, out_file_format *outs, void *vp, int sz, int sz2)
  Internal function to output an object (or an array or 2-d array).
- virtual int object_in_void (cinput *cin, in_file_format *ins, void *op, std::string &name)

Input an object (no memory allocation).
- virtual int object_in_void (cinput ∗cin, in_file_format ∗ins, void ∗op, int sz, std::string &name)
  Input an array of objects (no memory allocation).
- virtual int object_in_void (cinput ∗cin, in_file_format ∗ins, void ∗op, int sz, int sz2, std::string &name)
  Input a 2-d array of objects (no memory allocation).
- virtual int object_in_mem_void (cinput ∗cin, in_file_format ∗ins, void ∗&op, std::string &name)
  Input an object (no memory allocation).
- virtual int object_in_mem_void (cinput ∗cin, in_file_format ∗ins, void ∗&op, int &sz, std::string &name)
  Input an array of objects (no memory allocation).
- virtual int object_in_mem_void (cinput ∗cin, in_file_format ∗ins, void ∗&op, int &sz, int &sz2, std::string &name)
  Input a 2-d array of objects (no memory allocation).
- virtual int object_out_void (coutput ∗cout, out_file_format ∗outs, void ∗op, int sz, int sz2, std::string name="")
  Output an object, an array of objects, or a 2-d array of objects.

Functions to remove the memory that was allocated for an object

- virtual int remove (void ∗vp)
  Remove the memory for an object.
- virtual int remove_arr (void ∗vp)
  Remove the memory for an array of objects.
- virtual int remove_2darr (void ∗vp, int sz)
  Remove the memory for a 2-dimensional array of objects.

Protected Attributes

- int sw_store
  Store the value of `sw` given in the constructor so that we know if we need to remove the type in the destructor.

Static Protected Attributes

- static class io_manager ∗ iom
  A pointer to the type manager.
- static int objs_count
  A count of the number of objects.

7.157.2 Constructor & Destructor Documentation

7.157.2.1 io_base (int *sw* = `0`)

Create a new I/O object.

If `sw` is different from zero, then the type will not be added to the io_manager. This is useful if you want an object to be its own I/O class, in which case you may want to make sure that the io_manager only tries to add the type once. There is no need to have an I/O object for every instance of a particular type.

7.157.3 Member Function Documentation

7.157.3.1 virtual int pointer_out (coutput ∗ *co*, out_file_format ∗ *outs*, void ∗ *ptr*, std::string *stype*) `[virtual]`

Output an object to `outs` of type `stype`.

This is useful for to output a pointer to an object in the out() or stat_out() functions for a class. The data for the object which is pointed to is separate from the object and is only referred to once if more than one objects point to it.

The documentation for this class was generated from the following file:

- collection.h

7.158 io_manager Class Reference

`#include <collection.h>`

7.158.1 Detailed Description

Manage I/O type information.

This class is automatically created, utilized, and destroyed by io_base.

Definition at line 257 of file collection.h.

Public Member Functions

- int add_type (io_base ∗iop)
  Add a type to the list.
- int is_type (io_base ∗iop)
  Return 0 if `iop` *points to a valid type.*
- int add_type (io_base ∗iop, const char ∗t)
  Add type `iop` *to the manager assuming the type name* `t`*.*
- int remove_type (io_base ∗iop)
  Remove a type from the list.

Protected Types

- typedef std::vector< io_base ∗ >::iterator titer
  A useful definition for iterating through types.

Protected Member Functions

- io_base ∗ get_ptr (std::string stype)
  Get a pointer to type `stype`*.*
- io_manager ()
  Empty constructor.

Protected Attributes

- std::vector< io_base ∗ > tlist
  The list of types in the form of io_base pointers.

7.158.2 Member Function Documentation

7.158.2.1 int add_type (io_base ∗ *iop*)

Add a type to the list.

Unfortunately, add_type() cannot ensure that no type is added more than once, since the type is not specified until the entire constructor hierarchy has been executed and add_type() is called at the top of this hierarchy.

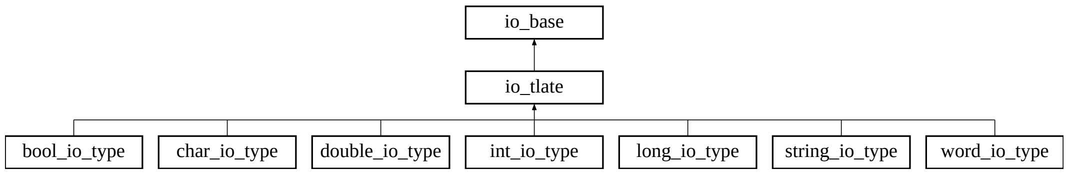
The documentation for this class was generated from the following file:

- collection.h

7.159 io_tlate Class Template Reference

```
#include <collection.h>
```

Inheritance diagram for io_tlate::



7.159.1 Detailed Description

template<class object> class io_tlate< object >

A template for adding I/O classes (documents template io_tlate).

Note that the generic interface here only works with pointers, not with the actual objects themselves. This is important, because it avoids the problem of I/O for an object with private copy and assigment operators. For basic types (bool, char, double, int, etc.), some additional add() and get() functions are defined.

Definition at line 1059 of file collection.h.

Public Member Functions

- io_tlate ()
  Create an I/O class for type `object`*.*
- io_tlate (const char *t)
  Create an I/O class for type `object` *only if another object of type* `t` *is not yet present.*
- template<>
  int **input** (cinput *co, in_file_format *ins, bool *dp)
- template<>
  int **output** (coutput *co, out_file_format *outs, bool *dp)
- template<>
  const char * type ()
  Return the type of an object.
- template<>
  int **input** (cinput *co, in_file_format *ins, char *dp)
- template<>
  int **output** (coutput *co, out_file_format *outs, char *dp)
- template<>
  const char * type ()
  Return the type of an object.
- template<>
  int **input** (cinput *co, in_file_format *ins, double *dp)
- template<>
  int **output** (coutput *co, out_file_format *outs, double *dp)
- template<>
  const char * type ()
  Return the type of an object.
- template<>
  int **input** (cinput *co, in_file_format *ins, int *dp)
- template<>
  int **output** (coutput *co, out_file_format *outs, int *dp)

- template<>
  const char * type ()
  Return the type of an object.
- template<>
  int **input** (cinput *co, in_file_format *ins, unsigned long int *dp)
- template<>
  int **output** (coutput *co, out_file_format *outs, unsigned long int *dp)
- template<>
  const char * type ()
  Return the type of an object.
- template<>
  int **input** (cinput *co, in_file_format *ins, std::string *dp)
- template<>
  int **output** (coutput *co, out_file_format *outs, std::string *dp)
- template<>
  const char * type ()
  Return the type of an object.
- template<>
  int **input** (cinput *co, o2scl::in_file_format *ins, table *ta)
- template<>
  int **output** (coutput *co, o2scl::out_file_format *outs, table *at)
- template<>
  const char * type ()
  Return the type of an object.
- template<>
  int **input** (cinput *co, in_file_format *ins, gsl_series *gs)
- template<>
  int **output** (coutput *co, out_file_format *outs, gsl_series *gs)
- template<>
  const char * type ()
  Return the type of an object.

Functions to be overloaded

These functions should be overloaded in all descendants of io_tlate.

- virtual const char * type ()
  The name of the type to be processed.
- virtual int input (cinput *cin, in_file_format *ins, object *op)
  Method for reading an object from `ins`.
- virtual int output (coutput *cout, out_file_format *outs, object *op)
  Method for writing an object to `outs`.

Functions to be overloaded for static data

These functions should be overloaded in all descendants of io_tlate which control I/O for classes which contain static data.

- virtual bool has_static_data ()
  `true` *if the object contains static I/O data*
- virtual int stat_input (cinput *cin, in_file_format *ins, object *op)
  Method for reading static data for an object from `ins`.
- virtual int stat_output (coutput *cout, out_file_format *outs, object *op)
  Method for writing static data for an object to `outs`.

Input functions

- virtual int object_in (cinput *cin, in_file_format *ins, object *op, std::string &name)
  Read an object from `ins`.
- virtual int object_in (cinput *cin, in_file_format *ins, object *op, int sz, std::string &name)
  Read an array of objects from `ins`.

- virtual int object_in (cinput ∗cin, in_file_format ∗ins, object ∗∗op, int sz, int sz2, std::string &name)
    Read a 2-d array of objects from `ins`.
- template<size_t N>
  int object_in (cinput ∗co, in_file_format ∗ins, object op[ ][N], int sz, std::string &name)
    Create memory for a 2-d array of objects and read it from `ins`.
- virtual int object_in_mem (cinput ∗cin, in_file_format ∗ins, object ∗&op, std::string &name)
    Create memory for an object and read it from `ins`.
- virtual int object_in_mem (cinput ∗cin, in_file_format ∗ins, object ∗&op, int &sz, std::string &name)
    Create memory for an object and read it from `ins`.
- virtual int object_in_mem (cinput ∗cin, in_file_format ∗ins, object ∗∗&op, int &sz, int &sz2, std::string &name)
    Create memory for an object and read it from `ins`.
- template<size_t N>
  int object_in_mem (cinput ∗co, in_file_format ∗ins, object op[ ][N], int &sz, std::string &name)
    Create memory for a 2-d array of objects and read it from `ins`.

Output functions

- virtual int object_out (coutput ∗cout, out_file_format ∗outs, object ∗op, int sz=0, std::string name="")
    Output an object (or an array of objects) to `outs`.
- virtual int object_out (coutput ∗cout, out_file_format ∗outs, object ∗∗op, int sz, int sz2, std::string name="")
    Output an object (or an array of objects) to `outs`.
- template<size_t N>
  int object_out (coutput ∗cout, out_file_format ∗outs, object op[ ][N], int sz, std::string name="")
    Output a 2-d array of objects to `outs`.

Memory allocation

- virtual int mem_alloc (object ∗&op)
    Create memory for an object.
- virtual int mem_alloc_arr (object ∗&op, int sz)
    Create memory for an object.
- virtual int mem_alloc_2darr (object ∗∗&op, int sz, int sz2)
    Create memory for an object.

Add and get objects from a collection

- int add (collection &coll, std::string name, object ∗op, int sz=0, bool overwrt=true, bool owner=false)
    Add an object(s) to a collection.
- int add_2darray (collection &coll, std::string name, object ∗∗op, int sz, int sz2, bool overwrt=true, bool owner=false)
    Add an object(s) to a collection.
- int get (collection &coll, std::string tname, object ∗&op)
    Get an object(s) from a collection.
- int get (collection &co, std::string tname, object ∗&op, int &sz)
    Get an object(s) from a collection.
- int get (collection &co, std::string tname, object ∗∗&op, int &sz, int &sz2)
    Get an object(s) from a collection.

Other functions

- virtual int mem_free (object ∗op)
    Free the memory associated with an object.
- virtual int mem_free_arr (object ∗op)
    Free the memory associated with an array of objects.
- virtual int mem_free_2darr (object ∗∗op, int sz)
    Free the memory associated with a 2-d array of objects.

Protected Member Functions

- virtual int object_in_void (cinput ∗cin, in_file_format ∗ins, void ∗op, std::string &name)
  Input an object (no memory allocation).
- virtual int object_in_void (cinput ∗cin, in_file_format ∗ins, void ∗op, int sz, std::string &name)
  Input an array of objects (no memory allocation).
- virtual int object_in_void (cinput ∗cin, in_file_format ∗ins, void ∗op, int sz, int sz2, std::string &name)
  Input a 2-d array of objects (no memory allocation).
- virtual int object_in_mem_void (cinput ∗cin, in_file_format ∗ins, void ∗&vp, std::string &name)
  Input an object (no memory allocation).
- virtual int object_in_mem_void (cinput ∗cin, in_file_format ∗ins, void ∗&vp, int &sz, std::string &name)
  Input an array of objects (no memory allocation).
- virtual int object_in_mem_void (cinput ∗cin, in_file_format ∗ins, void ∗&vp, int &sz, int &sz2, std::string &name)
  Input a 2-d array of objects (no memory allocation).
- virtual int object_out_void (coutput ∗cout, out_file_format ∗outs, void ∗op, int sz=0, int sz2=0, std::string name="")
  Output an object, an array of objects, or a 2-d array of objects.
- virtual int stat_in_noobj (cinput ∗cin, in_file_format ∗ins)
  Automatically create an object for stat_in.
- virtual int stat_out_noobj (coutput ∗cout, out_file_format ∗outs)
  Automatically create an object for stat_out.
- int in_wrapper (cinput ∗cin, in_file_format ∗ins, void ∗&vp)
  Allocate memory and input an object.
- int in_wrapper (cinput ∗cin, in_file_format ∗ins, void ∗&vp, int &sz)
  Allocate memory and input an array of objects.
- int in_wrapper (cinput ∗cin, in_file_format ∗ins, void ∗&vp, int &sz, int &sz2)
  Allocate memory and input a 2-d array of objects.
- int out_wrapper (coutput ∗cout, out_file_format ∗outs, void ∗vp, int sz, int sz2)
  Internal function to output an object (or an array or 2-d array).
- virtual int remove (void ∗vp)
  Remove the memory for an object.
- virtual int remove_arr (void ∗vp)
  Remove the memory for an array of objects.
- virtual int remove_2darr (void ∗vp, int sz)
  Remove the memory for a 2-dimensional array of objects.
- virtual int stat_in_wrapper (cinput ∗cin, in_file_format ∗ins, void ∗vp)
  Static input for an object.
- virtual int stat_out_wrapper (coutput ∗cout, out_file_format ∗outs, void ∗vp)
  Static output for an object.

7.159.2 Member Function Documentation

7.159.2.1 virtual int stat_input (cinput ∗ *cin*, in_file_format ∗ *ins*, object ∗ *op*) `[inline, virtual]`

Method for reading static data for an object from `ins`.

One must be careful about objects which set the static data in their constructors. An object is automatically created in order to read its static data. This means that if the static data is set in the constructor, then possibly useful information will be overwritten through the creation of this temporary object.

If one needs to set static data in the constructor of a singleton object, then the create() and remove() functions should be empty and a separate pointer to the singleton should be provided instead of void ∗vp.

This is only used if has_static_data() returns true;

Definition at line 1119 of file collection.h.

7.159.2.2 int object_in_mem (cinput * *co*, in_file_format * *ins*, object *op*[ ][N], int & *sz*, std::string & *name*) [inline]

Create memory for a 2-d array of objects and read it from `ins`.

Note that you must specify in advance the size `N`.

Definition at line 1442 of file collection.h.

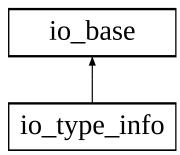
The documentation for this class was generated from the following file:

- collection.h

7.160 io_type_info Class Reference

`#include <collection.h>`

Inheritance diagram for io_type_info::



7.160.1 Detailed Description

User interface to provide I/O type information.

Definition at line 324 of file collection.h.

Public Member Functions

Type manipulation

- int is_type (std::string stype)
  Return 0 if `stype` is a valid I/O type.
- int remove_type (std::string stype)
  Remove `stype` from the list of valid I/O types.
- virtual int clear_types ()
  Remove all types in the list of valid I/O types.
- void type_summary (std::ostream *outs, bool pointers=false)
  Print a summary of valid types to the `outs` stream.
- int add_type (io_base *iop)
  Add an I/O type to the list.

Protected Types

- typedef std::vector< io_base * >::iterator titer
  A useful definition for iterating through types.

Protected Member Functions

- int static_fout (coutput *co, out_file_format *out)
  Output the static information for the I/O types.
- int static_fout_restricted (coutput *co, out_file_format *out, std::set< std::string, string_comp > list)
  Output the static information for the I/O types not in the list.

7.160.2 Member Function Documentation

7.160.2.1 int remove_type (std::string *stype*)

Remove `stype` from the list of valid I/O types.

This method is dangerous, as it can't check to ensure that no collection has remaining objects of the type to be removed.

7.160.2.2 virtual int clear_types () `[virtual]`

Remove all types in the list of valid I/O types.

This method is dangerous as it doesn't ensure that all collections are empty.

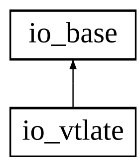
The documentation for this class was generated from the following file:

- collection.h

7.161 io_vtlate Class Template Reference

#include <collection.h>

Inheritance diagram for io_vtlate::



7.161.1 Detailed Description

template<class object> class io_vtlate< object >

A template for adding I/O classes.

Definition at line 983 of file collection.h.

Public Member Functions

Functions to be overloaded

These functions should be overloaded in all descendants of io_tlate.

- virtual const char ∗ type ()
  The name of the type to be processed.
- virtual int input (cinput ∗cin, in_file_format ∗ins, object ∗op)
  Method for reading an object from `ins`.
- virtual int output (coutput ∗cout, out_file_format ∗outs, object ∗op)
  Method for writing an object to `outs`.

Functions to be overloaded for static data

These functions should be overloaded in all descendants of io_tlate which control I/O for classes which contain static data.

- virtual bool has_static_data ()
  `true` *if the object contains static I/O data*
- virtual int stat_input (cinput ∗cin, in_file_format ∗ins, object ∗op)

Method for reading static data for an object from `ins`.

- virtual int stat_output (coutput *cout, out_file_format *outs, object *op)

    Method for writing static data for an object to `outs`.

7.161.2 Member Function Documentation

7.161.2.1 virtual int stat_input (cinput * *cin*, in_file_format * *ins*, object * *op*) [inline, virtual]

Method for reading static data for an object from `ins`.

One must be careful about objects which set the static data in their constructors. An object is automatically created in order to read its static data. This means that if the static data is set in the constructor, then possibly useful information will be overwritten through the creation of this temporary object.

If one needs to set static data in the constructor of a singleton object, then the create() and remove() functions should be empty and a separate pointer to the singleton should be provided instead of void *vp.

This is only used if has_static_data() returns true;

Definition at line 1034 of file collection.h.

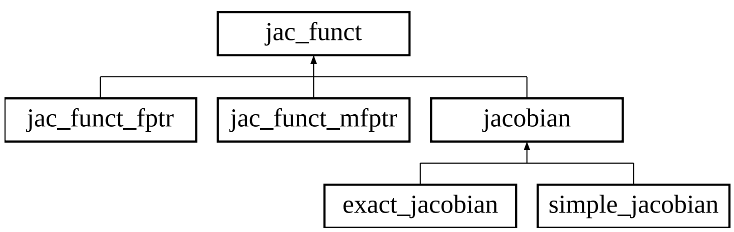
The documentation for this class was generated from the following file:

- collection.h

7.162 jac_funct Class Template Reference

`#include <jacobian.h>`

Inheritance diagram for jac_funct::



7.162.1 Detailed Description

template<class param_t, class vec_t = ovector_view, class mat_t = omatrix_view> class jac_funct< param_t, vec_t, mat_t >

Base for a square Jacobian where J is computed at x given y=f(x) [abstract base].

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.

For Jacobian functions with C-style arrays and matrices, use the corresponding children of jac_vfunct .

Definition at line 56 of file jacobian.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa)=0
  The operator().

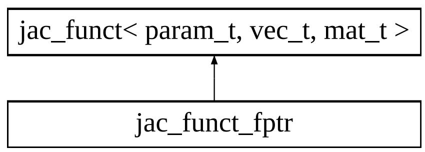
The documentation for this class was generated from the following file:

- jacobian.h

7.163 jac_funct_fptr Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jac_funct_fptr::



7.163.1 Detailed Description

template<class param_t, class vec_t = ovector_view, class mat_t = omatrix_view> class jac_funct_fptr< param_t, vec_t, mat_t >

Function pointer to jacobian.

Definition at line 83 of file jacobian.h.

Public Member Functions

- jac_funct_fptr (int(*fp)(size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa)
  The operator().

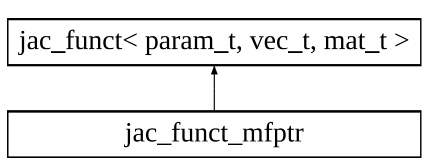
The documentation for this class was generated from the following file:

- jacobian.h

7.164 jac_funct_mfptr Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jac_funct_mfptr::



7.164.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view, class mat_t = omatrix_view> class jac_funct_mfptr< tclass, param_t, vec_t, mat_t >

Member function pointer to a Jacobian.

Definition at line 126 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, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa)
  The operator().

Protected Attributes

- int(tclass::∗ fptr)(size_t nv, vec_t &x, vec_t &y, mat_t &j, param_t &pa)
  Member function pointer.
- tclass ∗ tptr
  Class pointer.

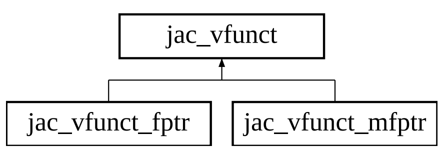
The documentation for this class was generated from the following file:

- jacobian.h

7.165 jac_vfunct Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jac_vfunct::



7.165.1 Detailed Description

template<class param_t, size_t nv> class jac_vfunct< param_t, nv >

Base for a square Jacobian where J is computed at x given y=f(x) [abstract base].

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 restored to their original values before operator() exits.

Definition at line 188 of file jacobian.h.

Public Member Functions

- virtual int operator() (size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)=0
  The operator().

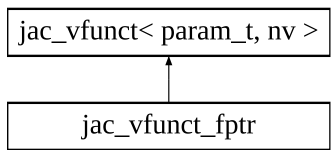
The documentation for this class was generated from the following file:

- jacobian.h

7.166 jac_vfunct_fptr Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jac_vfunct_fptr::



7.166.1 Detailed Description

template<class param_t, size_t nv> class jac_vfunct_fptr< param_t, nv >

Function pointer to jacobian.

Definition at line 214 of file jacobian.h.

Public Member Functions

- jac_vfunct_fptr (int(*fp)(size_t nv, double x[nv], double y[nv], double j[nv][nv], param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)
  The operator().

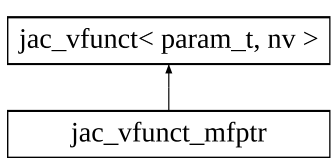
The documentation for this class was generated from the following file:

- jacobian.h

7.167 jac_vfunct_mfptr Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jac_vfunct_mfptr::



7.167.1 Detailed Description

template<class tclass, class param_t, size_t nv> class jac_vfunct_mfptr< tclass, param_t, nv >

Member function pointer to a Jacobian.

Definition at line 256 of file jacobian.h.

Public Member Functions

- jac_vfunct_mfptr (tclass *tp, int(tclass::*fp)(size_t nv, double x[nv], double y[nv], double j[nv][nv], param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv2, double x[nv], double y[nv], double j[nv][nv], param_t &pa)
  The operator().

Protected Attributes

- int(tclass::* fptr)(size_t nv, double x[nv], double y[nv], double j[nv][nv], param_t &pa)
  Member function pointer.
- tclass * tptr
  Class pointer.

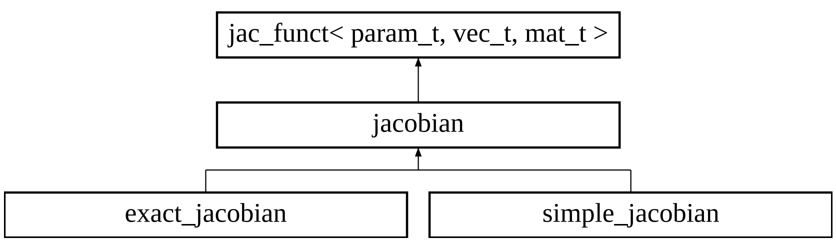
The documentation for this class was generated from the following file:

- jacobian.h

7.168 jacobian Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for jacobian::



7.168.1 Detailed Description

template<class param_t, class func_t = mm_funct<param_t>, class vec_t = ovector_view, class mat_t = omatrix_view> class jacobian< param_t, func_t, vec_t, mat_t >

Base for providing a numerical jacobian [abstract base].

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 312 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, param_t &pa)=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

7.169 lanczos Class Template Reference

`#include <lanczos_base.h>`

7.169.1 Detailed Description

template<class vec_t, class mat_t, class alloc_vec_t, class alloc_t> class lanczos< vec_t, mat_t, alloc_vec_t, alloc_t >

Lanczos diagonalization.

This is useful for approximating the largest eigenvalues of a symmetric matrix.

The vector and matrix types can be any type which provides access via `operator`[], given suitably constructed allocation types.

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.

Definition at line 41 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 tri-diagonal 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.

7.169.2 Member Function Documentation

7.169.2.1 int eigenvalues (size_t *size*, mat_t & *mat*, size_t *n_iter*, vec_t & *eigen*, vec_t & *diag*, vec_t & *off_diag*) `[inline]`

Approximate the largest eigenvalues of a symmetric matrix `mat` using the Lanczos method.

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 77 of file lanczos_base.h.

7.169.2.2 int eigen_tdiag (size_t *n*, vec_t & *diag*, vec_t & *off_diag*) `[inline]`

In-place diagonalization of a tri-diagonal matrix.

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 162 of file lanczos_base.h.

7.169.2.3 void product (size_t *n*, mat_t & *a*, vec_t & *w*, vec_t & *prod*) `[inline, protected]`

Naive matrix-vector product.

It is assumed that memory is already allocated for `prod`.

Definition at line 299 of file lanczos_base.h.

The documentation for this class was generated from the following file:

- lanczos_base.h

7.170 lib_settings_class Class Reference

```
#include <lib_settings.h>
```

7.170.1 Detailed Description

A class to manage global library settings.

Definition at line 64 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 range_check ()
  Return true if range checking was turned on during installation.
- 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.

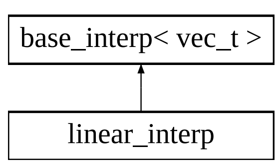
The documentation for this class was generated from the following file:

- lib_settings.h

7.171 linear_interp Class Template Reference

```
#include <interp.h>
```

Inheritance diagram for linear_interp::



7.171.1 Detailed Description

template<class vec_t> class linear_interp< vec_t >

Linear interpolation (GSL).

Linear interpolation requires no calls to allocate(), free() or init(), as there is no internal storage required.

Definition at line 149 of file interp.h.

Public Member Functions

- 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 $y''(x = x_0)$.
- 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 $\int_a^b y(x)\,dx$.

The documentation for this class was generated from the following file:

- interp.h

7.172 linear_solver Class Template Reference

```
#include <ode_it_solve.h>
```

7.172.1 Detailed Description

template<class vec_t, class mat_t> class linear_solver< vec_t, mat_t >

A generic solver for the linear system $Ax = b$.

Definition at line 124 of file ode_it_solve.h.

Public Member Functions

- virtual int **solve** (size_t n, mat_t &a, vec_t &b, vec_t &x)

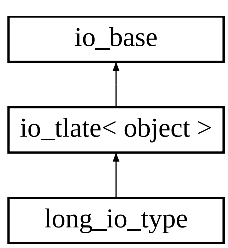
The documentation for this class was generated from the following file:

- ode_it_solve.h

7.173 long_io_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for long_io_type::



7.173.1 Detailed Description

I/O object for long variables.

Definition at line 1771 of file collection.h.

Public Member Functions

- long_io_type (const char *t)
  Desc.
- int addl (collection &co, std::string name, unsigned long int x, bool overwrt=true)
  Add a long to a collection.
- int getl (collection &co, std::string tname)
  Get a long from a collection.
- int get_def (collection &co, std::string tname, unsigned long int &op, unsigned long int def=0)
  Get a long from a collection.

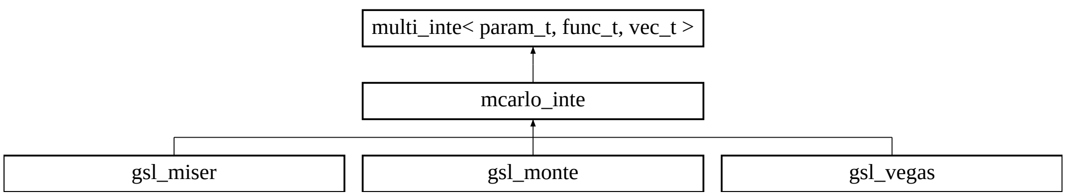
The documentation for this class was generated from the following file:

- collection.h

7.174 mcarlo_inte Class Template Reference

```
#include <mcarlo_inte.h>
```

Inheritance diagram for mcarlo_inte::



7.174.1 Detailed Description

template<class param_t, class func_t, class rng_t = gsl_rnga, class vec_t = ovector_view> class mcarlo_inte< param_t, func_t, rng_t, vec_t >

Monte-Carlo integration [abstract base].

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 44 of file mcarlo_inte.h.

Public Member Functions

- virtual const char * type ()
  Return string denoting type ("mcarlo_inte").

Data Fields

- int n_points
  Number of integration points (default 1000).
- rng_t def_rng
  The random number generator.

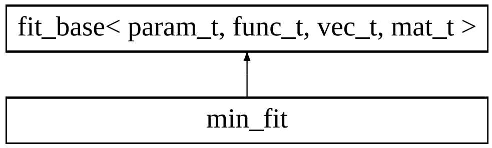
The documentation for this class was generated from the following file:

- mcarlo_inte.h

7.175 min_fit Class Template Reference

```
#include <min_fit.h>
```

Inheritance diagram for min_fit::



7.175.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view> class min_fit< param_t, func_t, vec_t, mat_t >

Non-linear least-squares fitting class with generic minimizer.

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.

Definition at line 53 of file min_fit.h.

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, param_t &pa, 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< func_par ∗, multi_funct< func_par ∗, 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< func_par ∗, multi_funct< func_par ∗, 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< func_par ∗, multi_funct< func_par ∗, vec_t > > ∗ mmp
  The minimizer.
- bool min_set
  True if the minimizer has been set by the user.

Data Structures

- struct func_par
  A structure for passing information to the GSL functions.

7.175.2 Member Function Documentation

7.175.2.1 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*, param_t & *pa*, func_t & *fitfun*) `[inline, virtual]`

Fit the data specified in (xdat,ydat) to the function `fitfun` with the parameters in `par`.

The covariance matrix for the parameters is returned in `covar` and the value of χ^2 is returned in `chi2`.

Implements fit_base.

Definition at line 93 of file min_fit.h.

The documentation for this class was generated from the following file:

- min_fit.h

7.176 min_fit::func_par Struct Reference

`#include <min_fit.h>`

7.176.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class mat_t = omatrix_view> struct min_fit< param_t, func_t, vec_t, mat_t >::func_par

A structure for passing information to the GSL functions.

This structure is given so that the user can specify the minimizer to use.

Definition at line 70 of file min_fit.h.

Data Fields

- func_t & f
  The fitting function.
- param_t & vp
  The user-specified parameter.
- 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

7.177 minimize Class Template Reference

```
#include <minimize.h>
```

7.177.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t> class minimize< param_t, func_t, dfunc_t >

One-dimensional minimization [abstract base].

Note:

This base class does not actually perform any minimization. Use either gsl_min_brent or cern_minimize.

Idea for future

This does not have pure virtual functions, but I'd still like to prevent the user from directly instantiating a minimize object.

Definition at line 48 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, param_t &pa, 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, param_t &pa, 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, param_t &pa, func_t &func, dfunc_t &df)
  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, param_t &pa, 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 tolf
  The tolerance for the minimum function value.
- double tolx
  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).

Protected Attributes

- bool over_bkt
  Should be true if min_bkt() is overloaded.
- bool over_de
  Should be true if min_de() is overloaded.

7.177.2 Member Function Documentation

7.177.2.1 virtual int print_iter (double *x*, double *y*, int *iter*, double *value* = `0.0`, double *limit* = `0.0`, std::string *comment* = `""`) `[inline, virtual]`

Print out iteration information.

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 108 of file minimize.h.

7.177.2.2 virtual int min (double & *x*, double & *fmin*, param_t & *pa*, func_t & *func*) `[inline, virtual]`

Calculate the minimum `min` of `func` w.r.t 'x'.

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt() with the newly bracketed minimum.

Definition at line 139 of file minimize.h.

7.177.2.3 virtual int min_bkt (double & *x2*, double *x1*, double *x3*, double & *fmin*, param_t & *pa*, func_t & *func*) `[pure virtual]`

Calculate the minimum `min` of `func` with x2 bracketed between x1 and x3.

If this is not overloaded, it ignores the bracket and calls min().

Implemented in cern_minimize, and gsl_min_brent.

7.177.2.4 virtual int min_de (double & *x*, double & *fmin*, param_t & *pa*, func_t & *func*, dfunc_t & *df*) `[inline, virtual]`

Calculate the minimum `min` of `func` with derivative `dfunc` w.r.t 'x'.

If this is not overloaded, it attempts to bracket the minimum using bracket() and then calls min_bkt_de() with the newly bracketed minimum.

Definition at line 166 of file minimize.h.

7.177.2.5 virtual int bracket (double & *ax*, double & *bx*, double & *cx*, double & *fa*, double & *fb*, double & *fc*, param_t & *pa*, func_t & *func*) `[inline, virtual]`

Given interval `(ax,bx)`, attempt to bracket a minimum for function `func`.

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 routine will fail if the function has the same value at `ax`, `bx`, and the midpoint `(ax+bx)/2`.

Idea for future

Improve this algorithm with the standard golden ratio method?

Definition at line 196 of file minimize.h.

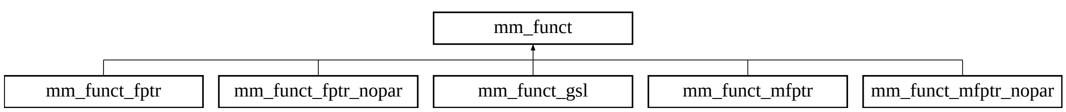
The documentation for this class was generated from the following file:

- minimize.h

7.178 mm_funct Class Template Reference

`#include <mm_funct.h>`

Inheritance diagram for mm_funct::



7.178.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class mm_funct< param_t, vec_t >

Array of multi-dimensional functions [abstract base].

This class generalizes `nv` functions of `nv` variables, i.e. $y_j(x_0, x_1, ..., x_{nv-1})$ for $0 \leq j \leq nv - 1$.

For functions with C-style arrays, use the corresponding children of mm_vfunct .

Definition at line 45 of file mm_funct.h.

Public Member Functions

- virtual int operator() (size_t nv, const vec_t &x, vec_t &y, param_t &pa)=0
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`.

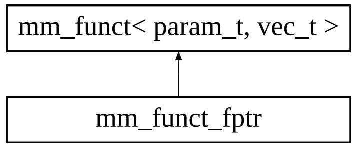
The documentation for this class was generated from the following file:

- mm_funct.h

7.179 mm_funct_fptr Class Template Reference

#include <mm_funct.h>

Inheritance diagram for mm_funct_fptr::



7.179.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class mm_funct_fptr< param_t, vec_t >

Function pointer to array of multi-dimensional functions.

Definition at line 72 of file mm_funct.h.

Public Member Functions

- mm_funct_fptr (int(∗fp)(size_t nv, const vec_t &x, vec_t &y, param_t &pa))
  Specify the function pointer.
- int set_function (int(∗fp)(size_t nv, const vec_t &x, vec_t &y, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y, param_t &pa)
  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 &pa)
  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 &)

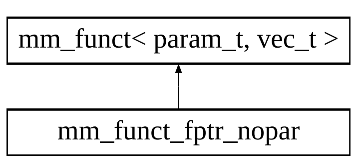
The documentation for this class was generated from the following file:

- mm_funct.h

7.180 mm_funct_fptr_nopar Class Template Reference

#include <mm_funct.h>

Inheritance diagram for mm_funct_fptr_nopar::



7.180.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class mm_funct_fptr_nopar< param_t, vec_t >

Function pointer to array of multi-dimensional functions with no parameters.

Definition at line 125 of file mm_funct.h.

Public Member Functions

- mm_funct_fptr_nopar (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, param_t &pa)
  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_nopar** (const mm_funct_fptr_nopar &)
- mm_funct_fptr_nopar & **operator=** (const mm_funct_fptr_nopar &)

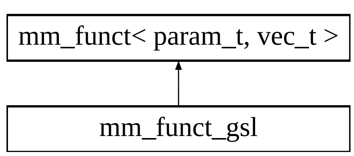
The documentation for this class was generated from the following file:

- mm_funct.h

7.181 mm_funct_gsl Class Template Reference

`#include <mm_funct.h>`

Inheritance diagram for mm_funct_gsl::



7.181.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class mm_funct_gsl< param_t, vec_t >

Function pointer to a gsl_multiroot_function.

This works because with the template parameter `vec_t` as an ovector_view class because ovector_view is inherited from `gsl_vector`.

Definition at line 180 of file mm_funct.h.

Public Member Functions

- mm_funct_gsl (int(∗fp)(const gsl_vector ∗x, param_t &pa, gsl_vector ∗f))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y, param_t &pa)
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(∗ fptr)(const gsl_vector ∗x, param_t &pa, gsl_vector ∗f)
  The function pointer to the user-supplied function.

Private Member Functions

- **mm_funct_gsl** (const mm_funct_gsl &)
- mm_funct_gsl & **operator=** (const mm_funct_gsl &)

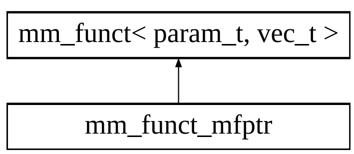
The documentation for this class was generated from the following file:

- mm_funct.h

7.182 mm_funct_mfptr Class Template Reference

`#include <mm_funct.h>`

Inheritance diagram for mm_funct_mfptr::



7.182.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class mm_funct_mfptr< tclass, param_t, vec_t >

Member function pointer to an array of multi-dimensional functions.

Definition at line 220 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, 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 &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, const vec_t &x, vec_t &y, param_t &pa)
  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.

Private Member Functions

- **mm_funct_mfptr** (const mm_funct_mfptr &)
- mm_funct_mfptr & **operator=** (const mm_funct_mfptr &)

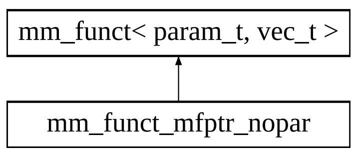
The documentation for this class was generated from the following file:

- mm_funct.h

7.183 mm_funct_mfptr_nopar Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm_funct_mfptr_nopar::



7.183.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class mm_funct_mfptr_nopar< tclass, param_t, vec_t >

Member function pointer to an array of multi-dimensional functions.

Definition at line 278 of file mm_funct.h.

Public Member Functions

- mm_funct_mfptr_nopar ()
  Empty constructor.
- mm_funct_mfptr_nopar (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, param_t &pa)
  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_nopar** (const mm_funct_mfptr_nopar &)
- mm_funct_mfptr_nopar & **operator=** (const mm_funct_mfptr_nopar &)

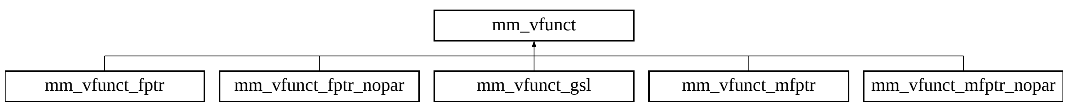
The documentation for this class was generated from the following file:

- mm_funct.h

7.184 mm_vfunct Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct::



7.184.1 Detailed Description

template<class param_t, size_t nv> class mm_vfunct< param_t, nv >

Array of multi-dimensional functions with arrays [abstract base].

This class generalizes `nv` functions of `nv` variables, i.e. $y_j(x_0, x_1, ..., x_{nv-1})$ for $0 \leq j \leq nv - 1$.

To use ovector_view objects instead of C-style arrays, use mm_funct.

Definition at line 345 of file mm_funct.h.

Public Member Functions

- virtual int operator() (size_t nvar, const double x[nv], double y[nv], param_t &pa)=0
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`*.*

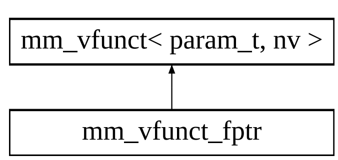
The documentation for this class was generated from the following file:

- mm_funct.h

7.185 mm_vfunct_fptr Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_fptr::



7.185.1 Detailed Description

template<class param_t, size_t nv> class mm_vfunct_fptr< param_t, nv >

Function pointer to array of multi-dimensional functions with arrays.

Definition at line 372 of file mm_funct.h.

Public Member Functions

- mm_vfunct_fptr (int(∗fp)(size_t nvar, const double x[nv], double y[nv], param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nvar, const double x[nv], double y[nv], param_t &pa)
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`*.*

Protected Attributes

- int(∗ fptr)(size_t nvar, const double x[nv], double y[nv], param_t &pa)
  The function pointer.

Private Member Functions

- **mm_vfunct_fptr** (const mm_vfunct_fptr &)
- mm_vfunct_fptr & **operator=** (const mm_vfunct_fptr &)

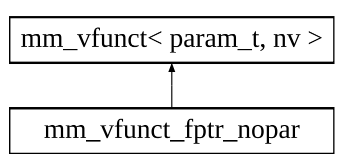
The documentation for this class was generated from the following file:

- mm_funct.h

7.186 mm_vfunct_fptr_nopar Class Template Reference

#include <mm_funct.h>

Inheritance diagram for mm_vfunct_fptr_nopar::



7.186.1 Detailed Description

template<class param_t, size_t nv> class mm_vfunct_fptr_nopar< param_t, nv >

Function pointer to array of multi-dimensional functions with arrays and no parameters.

Definition at line 418 of file mm_funct.h.

Public Member Functions

- mm_vfunct_fptr_nopar (int(∗fp)(size_t nvar, const double x[nv], double y[nv]))
  Specify the function pointer.
- virtual int operator() (size_t nvar, const double x[nv], double y[nv], param_t &pa)
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(∗ fptr)(size_t nvar, const double x[nv], double y[nv])
  The function pointer.

Private Member Functions

- **mm_vfunct_fptr_nopar** (const mm_vfunct_fptr_nopar &)
- mm_vfunct_fptr_nopar & **operator=** (const mm_vfunct_fptr_nopar &)

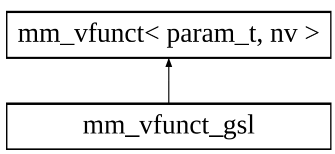
The documentation for this class was generated from the following file:

- mm_funct.h

7.187 mm_vfunct_gsl Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_gsl::



7.187.1 Detailed Description

template<class param_t, size_t nv> class mm_vfunct_gsl< param_t, nv >

Function pointer to a gsl_multiroot_function with arrays.

Definition at line 465 of file mm_funct.h.

Public Member Functions

- mm_vfunct_gsl (int(∗fp)(const gsl_vector ∗x, param_t &pa, gsl_vector ∗f))
  Specify the function pointer.
- virtual int operator() (size_t nvar, const double x[nv], double y[nv], param_t &pa)
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(∗ fptr)(const gsl_vector ∗x, param_t &pa, gsl_vector ∗f)
  The function pointer.

Private Member Functions

- **mm_vfunct_gsl** (const mm_vfunct_gsl &)
- mm_vfunct_gsl & **operator=** (const mm_vfunct_gsl &)

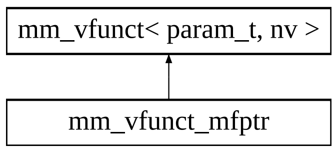
The documentation for this class was generated from the following file:

- mm_funct.h

7.188 mm_vfunct_mfptr Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_mfptr::



7.188.1 Detailed Description

template<class tclass, class param_t, size_t nv> class mm_vfunct_mfptr< tclass, param_t, nv >

Member function pointer to an array of multi-dimensional functions with arrays.

Definition at line 506 of file mm_funct.h.

Public Member Functions

- mm_vfunct_mfptr (tclass *tp, int(tclass::*fp)(size_t nvar, const double x[nv], double y[nv], param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nvar, const double x[nv], double y[nv], param_t &pa)
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nvar, const double x[nv], double y[nv], param_t &pa)
  The member function pointer.
- tclass * tptr
  The class pointer.

Private Member Functions

- **mm_vfunct_mfptr** (const mm_vfunct_mfptr &)
- mm_vfunct_mfptr & **operator=** (const mm_vfunct_mfptr &)

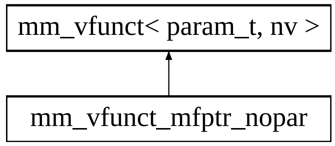
The documentation for this class was generated from the following file:

- mm_funct.h

7.189 mm_vfunct_mfptr_nopar Class Template Reference

```
#include <mm_funct.h>
```

Inheritance diagram for mm_vfunct_mfptr_nopar::



7.189.1 Detailed Description

template<class tclass, class param_t, size_t nv> class mm_vfunct_mfptr_nopar< tclass, param_t, nv >

Member function pointer to an array of multi-dimensional functions with arrays.

Definition at line 552 of file mm_funct.h.

Public Member Functions

- mm_vfunct_mfptr_nopar (tclass *tp, int(tclass::*fp)(size_t nvar, const double x[nv], double y[nv]))
  Specify the member function pointer.
- virtual int operator() (size_t nvar, const double x[nv], double y[nv], param_t &pa)
  Compute `nv` *functions,* `y`*, of* `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(tclass::* fptr)(size_t nvar, const double x[nv], double y[nv])
  The member function pointer.
- tclass * tptr
  The class pointer.

Private Member Functions

- **mm_vfunct_mfptr_nopar** (const mm_vfunct_mfptr_nopar &)
- mm_vfunct_mfptr_nopar & **operator=** (const mm_vfunct_mfptr_nopar &)

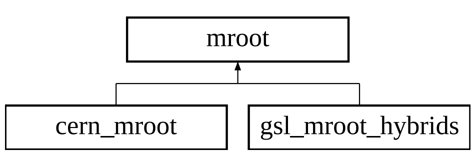
The documentation for this class was generated from the following file:

- mm_funct.h

7.190 mroot Class Template Reference

```
#include <mroot.h>
```

Inheritance diagram for mroot::



7.190.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class jfunc_t = jac_funct<param_t,vec_t,omatrix_view>> class mroot< param_t, func_t, vec_t, jfunc_t >

Multidimensional root-finding [abstract base].

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, param_t &pa);
```

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, param_t &pa);
```

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 Multidimensional solver example .

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, param_t &pa, func_t &func)=0
  Solve `func` *using* `x` *as an initial guess, returning* `x`*.*
- virtual int msolve_de (size_t n, vec_t &x, param_t &pa, 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 tolf
  The maximum value of the functions for success (default 1.0e-8).
- double tolx
  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.

7.190.2 Member Function Documentation

7.190.2.1 virtual int msolve_de (size_t *n*, vec_t & *x*, param_t & *pa*, func_t & *func*, jfunc_t & *dfunc*) `[inline, virtual]`

Solve `func` with derivatives `dfunc` using `x` as an initial guess, returning `x`.

By default, this function just calls msolve() and ignores the last argument.

Reimplemented in gsl_mroot_hybrids.

Definition at line 106 of file mroot.h.

7.190.2.2 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* = `""`) `[inline]`

Print out iteration information.

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 126 of file mroot.h.

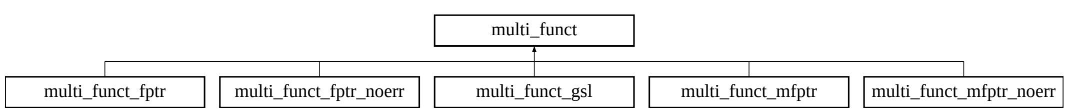
The documentation for this class was generated from the following file:

- mroot.h

7.191 multi_funct Class Template Reference

#include <multi_funct.h>

Inheritance diagram for multi_funct::



7.191.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class multi_funct< param_t, vec_t >

Multi-dimensional function [abstract base].

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.

For functions with C-style arrays, use the corresponding children of multi_vfunct .

Definition at line 44 of file multi_funct.h.

Public Member Functions

- virtual int operator() (size_t nv, const vec_t &x, double &y, param_t &pa)=0
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.

- virtual double operator() (size_t nv, const vec_t &x, param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

7.191.2 Member Function Documentation

7.191.2.1 virtual double operator() (size_t *nv*, const vec_t & *x*, param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented in multi_funct_fptr, multi_funct_gsl, multi_funct_fptr_noerr, multi_funct_mfptr, and multi_funct_mfptr_noerr.

Definition at line 65 of file multi_funct.h.

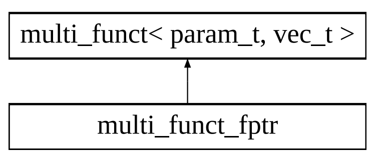
The documentation for this class was generated from the following file:

- multi_funct.h

7.192 multi_funct_fptr Class Template Reference

#include <multi_funct.h>

Inheritance diagram for multi_funct_fptr::



7.192.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class multi_funct_fptr< param_t, vec_t >

Function pointer to a multi-dimensional function.

Definition at line 85 of file multi_funct.h.

Public Member Functions

- multi_funct_fptr (int(*fp)(size_t nv, const vec_t &x, double &y, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const vec_t &x, param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(* fptr)(size_t nv, const vec_t &x, double &y, param_t &pa)
  Store the function pointer.

Private Member Functions

- **multi_funct_fptr** (const multi_funct_fptr &)
- multi_funct_fptr & **operator=** (const multi_funct_fptr &)

7.192.2 Member Function Documentation

7.192.2.1 virtual double operator() (size_t *nv*, const vec_t & *x*, param_t & *pa*) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_funct.

Definition at line 112 of file multi_funct.h.

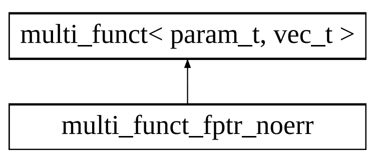
The documentation for this class was generated from the following file:

- multi_funct.h

7.193 multi_funct_fptr_noerr Class Template Reference

#include <multi_funct.h>

Inheritance diagram for multi_funct_fptr_noerr::



7.193.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class multi_funct_fptr_noerr< param_t, vec_t >

Function pointer to a multi-dimensional function without error control.

Definition at line 203 of file multi_funct.h.

Public Member Functions

- multi_funct_fptr_noerr (double(∗fp)(size_t nv, const vec_t &x, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const vec_t &x, param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- double(∗ fptr)(size_t nv, const vec_t &x, param_t &pa)
  Store the function pointer.

Private Member Functions

- **multi_funct_fptr_noerr** (const multi_funct_fptr_noerr &)
- multi_funct_fptr_noerr & **operator=** (const multi_funct_fptr_noerr &)

7.193.2 Member Function Documentation

7.193.2.1 virtual double operator() (size_t *nv*, const vec_t & *x*, param_t & *pa*) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_funct.

Definition at line 229 of file multi_funct.h.

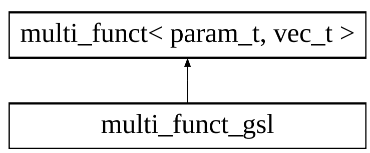
The documentation for this class was generated from the following file:

- multi_funct.h

7.194 multi_funct_gsl Class Template Reference

`#include <multi_funct.h>`

Inheritance diagram for multi_funct_gsl::



7.194.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class multi_funct_gsl< param_t, vec_t >

Function pointer to a gsl_multimin_function.

Definition at line 144 of file multi_funct.h.

Public Member Functions

- multi_funct_gsl (double(*fp)(const gsl_vector *x, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const vec_t &x, param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- double(* fptr)(const gsl_vector *x, param_t &pa)
  Store the function pointer.

Private Member Functions

- **multi_funct_gsl** (const multi_funct_gsl &)
- multi_funct_gsl & **operator=** (const multi_funct_gsl &)

7.194.2 Member Function Documentation

7.194.2.1 virtual double operator() (size_t *nv*, const vec_t & *x*, param_t & *pa*) [inline, virtual]

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_funct.

Definition at line 170 of file multi_funct.h.

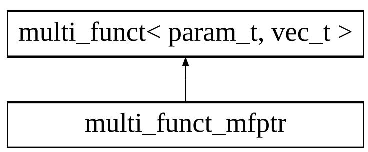
The documentation for this class was generated from the following file:

- multi_funct.h

7.195 multi_funct_mfptr Class Template Reference

#include <multi_funct.h>

Inheritance diagram for multi_funct_mfptr::



7.195.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class multi_funct_mfptr< tclass, param_t, vec_t >

Member function pointer to a multi-dimensional function.

Definition at line 261 of file multi_funct.h.

Public Member Functions

- multi_funct_mfptr (tclass ∗tp, int(tclass::∗fp)(size_t nv, const vec_t &x, double &y, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, const vec_t &x, double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const vec_t &x, param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(tclass::∗ fptr)(size_t nv, const vec_t &x, double &y, param_t &pa)
  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 &)

7.195.2 Member Function Documentation

7.195.2.1 virtual double operator() (size_t *nv*, const vec_t & *x*, param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_funct.

Definition at line 287 of file multi_funct.h.

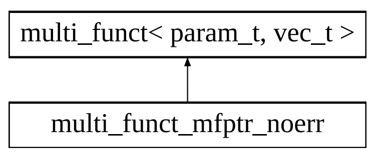
The documentation for this class was generated from the following file:

- multi_funct.h

7.196 multi_funct_mfptr_noerr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi_funct_mfptr_noerr::



7.196.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class multi_funct_mfptr_noerr< tclass, param_t, vec_t >

Member function pointer to a multi-dimensional function.

Definition at line 317 of file multi_funct.h.

Public Member Functions

- multi_funct_mfptr_noerr (tclass ∗tp, double(tclass::∗fp)(size_t nv, const vec_t &x, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, const vec_t &x, double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const vec_t &x, param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- double(tclass::∗ fptr)(size_t nv, const vec_t &x, param_t &pa)
  Store the function pointer.
- tclass ∗ tptr
  Store a pointer to the class instance.

Private Member Functions

- **multi_funct_mfptr_noerr** (const multi_funct_mfptr_noerr &)
- multi_funct_mfptr_noerr & **operator=** (const multi_funct_mfptr_noerr &)

7.196.2 Member Function Documentation

7.196.2.1 virtual double operator() (size_t *nv*, const vec_t & *x*, param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_funct.

Definition at line 344 of file multi_funct.h.

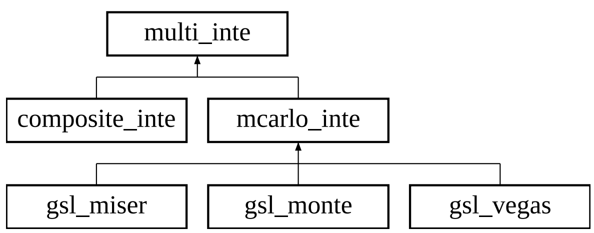
The documentation for this class was generated from the following file:

- multi_funct.h

7.197 multi_inte Class Template Reference

`#include <multi_inte.h>`

Inheritance diagram for multi_inte::



7.197.1 Detailed Description

template$<$class param_t, class func_t, class vec_t = ovector_view$>$ class multi_inte$<$ param_t, func_t, vec_t $>$

Multi-dimensional integration over a hypercube [abstract base].

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 42 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, param_t &pa)=0
  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, param_t &pa, 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.*
- 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

- int verbose
  Verbosity.
- double tolf
  The maximum "uncertainty" in the value of the integral (default 10^{-8}).

Protected Attributes

- double interror
  The uncertainty for the last integration computation.

7.197.2 Member Function Documentation

7.197.2.1 double get_error () `[inline]`

Return the error in the result from the last call to minteg() or minteg_err().

This will quietly return zero if no integrations have been performed.

Definition at line 92 of file multi_inte.h.

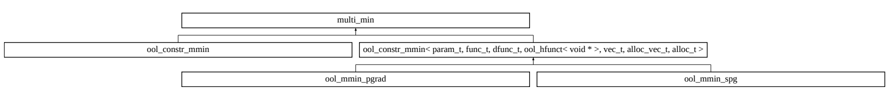
The documentation for this class was generated from the following file:

- multi_inte.h

7.198 multi_min Class Template Reference

`#include <multi_min.h>`

Inheritance diagram for multi_min::



7.198.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t, class vec_t = ovector_view> class multi_min< param_t, func_t, dfunc_t, vec_t >

Multidimensional minimization [abstract base].

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, param_t &pa);
```

where `f` is the value of the function at `x` with parameter `pa` 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, const vec_t &x, vec_t &g, param_t &pa);
```

where `g` is the gradient of the function at `x`.

Definition at line 409 of file multi_min.h.

Public Member Functions

- virtual int mmin (size_t nvar, vec_t &x, double &fmin, param_t &pa, 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, param_t &pa, 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 tolf

    Tolerance.
- double tolx

    The minimum allowable stepsize.
- int last_ntrial

    The number of iterations for in the most recent minimization.

7.198.2 Member Function Documentation

7.198.2.1 int print_iter (size_t *nv*, vec2_t & *x*, double *y*, int *iter*, double *value*, double *limit*, std::string *comment*) `[inline]`

Print out iteration information.

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 465 of file multi_min.h.

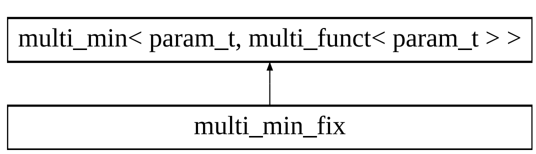
The documentation for this class was generated from the following file:

- multi_min.h

7.199 multi_min_fix Class Template Reference

```
#include <multi_min_fix.h>
```

Inheritance diagram for multi_min_fix::



7.199.1 Detailed Description

template<class param_t, class bool_vec_t> class multi_min_fix< param_t, bool_vec_t >

Multidimensional minimizer fixing some variables and varying others.

Todo

Generalize to all vector types

Definition at line 39 of file multi_min_fix.h.

Public Member Functions

- multi_min_fix ()
  Specify the member function pointer.
- virtual int mmin (size_t nvar, ovector_view &x, double &fmin, param_t &pa, multi_funct< param_t > &func)
  Calculate the minimum `min` *of* `func` *w.r.t. the array* `x` *of size* `nvar`.
- virtual int mmin_fix (size_t nvar, ovector_view &x, double &fmin, bool_vec_t &fix, param_t &pa, multi_funct< param_t > &func)
  Calculate the minimum of `func` *while fixing some parameters as specified in* `fix`.
- int set_mmin (multi_min< param_t, multi_funct_mfptr< multi_min_fix, param_t > > &min)
  Change the base minimizer.

Data Fields

- gsl_mmin_simp< param_t, multi_funct_mfptr< multi_min_fix, param_t > > def_mmin
  The default base minimizer.

Protected Member Functions

- virtual int min_func (size_t nv, const ovector_view &x, double &y, param_t &pa)
  The new function to send to the minimizer.

Protected Attributes

- multi_min< param_t, multi_funct_mfptr< multi_min_fix, param_t > > ∗ mmp
  The minimizer.
- multi_funct< param_t > ∗ 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_view ∗ xp
  The user-specified initial vector.

Private Member Functions

- **multi_min_fix** (const multi_min_fix &)
- multi_min_fix & **operator=** (const multi_min_fix &)

7.199.2 Member Function Documentation

7.199.2.1 virtual int mmin_fix (size_t *nvar*, ovector_view & *x*, double & *fmin*, bool_vec_t & *fix*, param_t & *pa*, multi_funct< param_t > & *func*) `[inline, virtual]`

Calculate the minimum of `func` while fixing some parameters as specified in `fix`.

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 94 of file multi_min_fix.h.

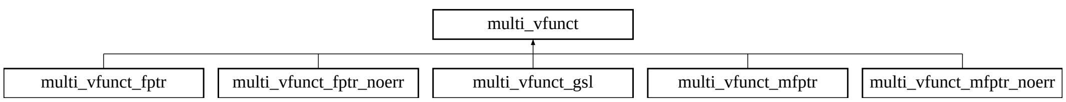
The documentation for this class was generated from the following file:

- multi_min_fix.h

7.200 multi_vfunct Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi_vfunct::



7.200.1 Detailed Description

template<class param_t, size_t nvar> class multi_vfunct< param_t, nvar >

Multi-dimensional function base with arrays [abstract base].

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.

Definition at line 381 of file multi_funct.h.

Public Member Functions

- virtual int operator() (size_t nv, const double x[nvar], double &y, param_t &pa)=0
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const double x[nvar], param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

7.200.2 Member Function Documentation

7.200.2.1 virtual double operator() (size_t *nv*, const double *x*[nvar], param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented in multi_vfunct_fptr, multi_vfunct_gsl, multi_vfunct_fptr_noerr, multi_vfunct_mfptr, and multi_vfunct_mfptr_noerr.

Definition at line 401 of file multi_funct.h.

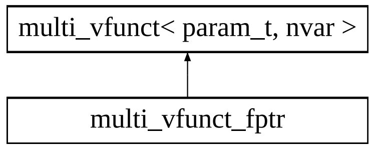
The documentation for this class was generated from the following file:

- multi_funct.h

7.201 multi_vfunct_fptr Class Template Reference

#include <multi_funct.h>

Inheritance diagram for multi_vfunct_fptr::



7.201.1 Detailed Description

template<class param_t, size_t nvar> class multi_vfunct_fptr< param_t, nvar >

Function pointer to a multi-dimensional function with arrays.

Definition at line 421 of file multi_funct.h.

Public Member Functions

- multi_vfunct_fptr (int(*fp)(size_t nv, const double x[nvar], double &y, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const double x[nvar], double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const double x[nvar], param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(* fptr)(size_t nv, const double x[nvar], double &y, param_t &pa)
  Store the function pointer.

Private Member Functions

- **multi_vfunct_fptr** (const multi_vfunct_fptr &)
- multi_vfunct_fptr & **operator=** (const multi_vfunct_fptr &)

7.201.2 Member Function Documentation

7.201.2.1 virtual double operator() (size_t *nv*, const double *x*[nvar], param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_vfunct.

Definition at line 449 of file multi_funct.h.

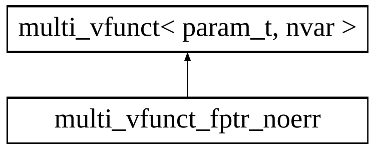
The documentation for this class was generated from the following file:

- multi_funct.h

7.202 multi_vfunct_fptr_noerr Class Template Reference

`#include <multi_funct.h>`

Inheritance diagram for multi_vfunct_fptr_noerr::



7.202.1 Detailed Description

template<class param_t, size_t nvar> class multi_vfunct_fptr_noerr< param_t, nvar >

Function pointer to a multi-dimensional function with arrays and without error control.

Definition at line 541 of file multi_funct.h.

Public Member Functions

- multi_vfunct_fptr_noerr (double(∗fp)(size_t nv, const double x[nvar], param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const double x[nvar], double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const double x[nvar], param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- double(∗ fptr)(size_t nv, const double x[nvar], param_t &pa)
  Store the function pointer.

Private Member Functions

- **multi_vfunct_fptr_noerr** (const multi_vfunct_fptr_noerr &)
- multi_vfunct_fptr_noerr & **operator=** (const multi_vfunct_fptr_noerr &)

7.202.2 Member Function Documentation

7.202.2.1 virtual double operator() (size_t *nv*, const double *x*[nvar], param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_vfunct.

Definition at line 568 of file multi_funct.h.

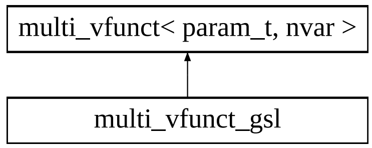
The documentation for this class was generated from the following file:

- multi_funct.h

7.203 multi_vfunct_gsl Class Template Reference

`#include <multi_funct.h>`

Inheritance diagram for multi_vfunct_gsl::



7.203.1 Detailed Description

template< class param_t, size_t nvar > class multi_vfunct_gsl< param_t, nvar >

Function pointer to a gsl_multimin_function with arrays.

Definition at line 481 of file multi_funct.h.

Public Member Functions

- multi_vfunct_gsl (double(*fp)(const gsl_vector *x, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const double x[nvar], double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const double x[nvar], param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- double(* fptr)(const gsl_vector *x, param_t &pa)
  Store the function pointer.

Private Member Functions

- **multi_vfunct_gsl** (const multi_vfunct_gsl &)
- multi_vfunct_gsl & **operator=** (const multi_vfunct_gsl &)

7.203.2 Member Function Documentation

7.203.2.1 virtual double operator() (size_t *nv*, const double *x*[nvar], param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_vfunct.

Definition at line 508 of file multi_funct.h.

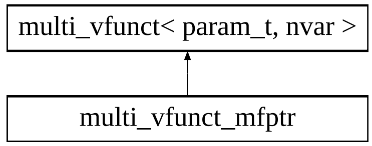
The documentation for this class was generated from the following file:

- multi_funct.h

7.204 multi_vfunct_mfptr Class Template Reference

```
#include <multi_funct.h>
```

Inheritance diagram for multi_vfunct_mfptr::



7.204.1 Detailed Description

template<class tclass, class param_t, size_t nvar> class multi_vfunct_mfptr< tclass, param_t, nvar >

Member function pointer to a multi-dimensional function with arrays.

Definition at line 601 of file multi_funct.h.

Public Member Functions

- multi_vfunct_mfptr (tclass ∗tp, int(tclass::∗fp)(size_t nv, const double x[nvar], double &y, param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, const double x[nvar], double &y, param_t &pa)
  Compute a function `y` *of* `nv` *variables stored in* `x` *with parameter* `pa`.
- virtual double operator() (size_t nv, const double x[nvar], param_t &pa)
  Return the value of a function of `nv` *variables stored in* `x` *with parameter* `pa`.

Protected Attributes

- int(tclass::∗ fptr)(size_t nv, const double x[nvar], double &y, param_t &pa)
  Store the function pointer.
- tclass ∗ tptr
  Store a pointer to the class instance.

Private Member Functions

- **multi_vfunct_mfptr** (const multi_vfunct_mfptr &)
- multi_vfunct_mfptr & **operator=** (const multi_vfunct_mfptr &)

7.204.2 Member Function Documentation

7.204.2.1 virtual double operator() (size_t *nv*, const double *x*[nvar], param_t & *pa*) `[inline, virtual]`

Return the value of a function of `nv` variables stored in `x` with parameter `pa`.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_vfunct.

Definition at line 629 of file multi_funct.h.

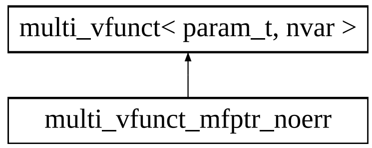
The documentation for this class was generated from the following file:

- multi_funct.h

7.205 multi_vfunct_mfptr_noerr Class Template Reference

#include <multi_funct.h>

Inheritance diagram for multi_vfunct_mfptr_noerr::



7.205.1 Detailed Description

template< class tclass, class param_t, size_t nvar > class multi_vfunct_mfptr_noerr< tclass, param_t, nvar >

Member function pointer to a multi-dimensional function with arrays.

Definition at line 662 of file multi_funct.h.

Public Member Functions

- multi_vfunct_mfptr_noerr (tclass *tp, double(tclass::*fp)(size_t nv, const double x[nvar], param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, const double x[nvar], double &y, param_t &pa)
  Compute a function y *of* nv *variables stored in* x *with parameter* pa.
- virtual double operator() (size_t nv, const double x[nvar], param_t &pa)
  Return the value of a function of nv *variables stored in* x *with parameter* pa.

Protected Attributes

- double(tclass::* fptr)(size_t nv, const double x[nvar], param_t &pa)
  Store the function pointer.
- tclass * tptr
  Store a pointer to the class instance.

Private Member Functions

- **multi_vfunct_mfptr_noerr** (const multi_vfunct_mfptr_noerr &)
- multi_vfunct_mfptr_noerr & **operator=** (const multi_vfunct_mfptr_noerr &)

7.205.2 Member Function Documentation

7.205.2.1 virtual double operator() (size_t *nv*, const double *x*[nvar], param_t & *pa*) [inline, virtual]

Return the value of a function of nv variables stored in x with parameter pa.

Note that this is reimplemented in all children because if one member function operator() is reimplemented, all must be.

Reimplemented from multi_vfunct.

Definition at line 690 of file multi_funct.h.

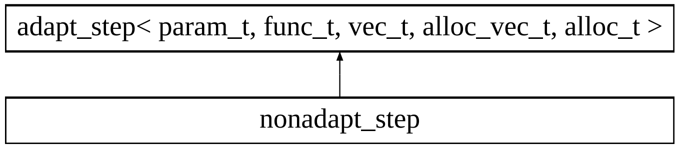
The documentation for this class was generated from the following file:

- multi_funct.h

7.206 nonadapt_step Class Template Reference

`#include <nonadapt_step.h>`

Inheritance diagram for nonadapt_step::



7.206.1 Detailed Description

template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class nonadapt_step< param_t, func_t, vec_t, alloc_vec_t, alloc_t >

An non-adaptive stepper implementation of adapt_step.

This class simply calls the specified ODE stepper without any attempt to modify the size of the step, and is primarily useful to allow for simple comparisons between adaptive and non-adaptive solution. To modify the ODE stepper which is used, use the adapt_step::set_step().

Idea for future

Modify so that memory allocation/deallocation is only performed when necessary

Definition at line 49 of file nonadapt_step.h.

Public Member Functions

- virtual int astep (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &u_dydx_out, vec_t &yerr, param_t &pa, func_t &derivs)
  Make an adaptive integration step of the system `derivs`.
- virtual int astep_derivs (double &x, double &h, double xmax, size_t n, vec_t &y, vec_t &dydx, vec_t &yerr, param_t &pa, func_t &derivs)
  Make an adaptive integration step of the system `derivs` *with derivatives.*

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`.

7.206.2 Member Function Documentation

7.206.2.1 virtual int astep (double & *x*, double & *h*, double *xmax*, size_t *n*, vec_t & *y*, vec_t & *u_dydx_out*, vec_t & *yerr*, param_t & *pa*, func_t & *derivs*) `[inline, virtual]`

Make an adaptive integration step of the system `derivs`.

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.

Definition at line 70 of file nonadapt_step.h.

7.206.2.2 virtual int astep_derivs (double & *x*, double & *h*, double *xmax*, size_t *n*, vec_t & *y*, vec_t & *dydx*, vec_t & *yerr*, param_t & *pa*, func_t & *derivs*) [inline, virtual]

Make an adaptive integration step of the system `derivs` with derivatives.

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.

Definition at line 100 of file nonadapt_step.h.

The documentation for this class was generated from the following file:

- nonadapt_step.h

7.207 o2scl_hybrid_state_t Class Template Reference

`#include <gsl_mroot_hybrids.h>`

7.207.1 Detailed Description

template<class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_t = omatrix_view, class alloc_mat_t = omatrix, class mat_alloc_t = omatrix_alloc> class o2scl_hybrid_state_t< vec_t, alloc_vec_t, alloc_t, mat_t, alloc_mat_t, mat_alloc_t >

State class for gsl_mroot_hybrids.

Definition at line 43 of file gsl_mroot_hybrids.h.

Public Member Functions

- int allocate (size_t n)
  Allocate memory for a solver with `n` *variables.*
- int free ()
  Free allocated memory.

Data Fields

- alloc_t va
  Vector allocator.
- mat_alloc_t ma
  Matrix allocator.
- size_t iter
  Number of iterations.
- size_t ncfail
  Desc.
- size_t ncsuc
  Desc.
- size_t nslow1
  Desc.
- size_t nslow2
  Desc.
- double fnorm
  Desc.
- double delta

Desc.
- alloc_mat_t J
  Jacobian.
- gsl_matrix ∗ q
  Q matrix from QR decomposition.
- gsl_matrix ∗ r
  R matrix from QR decomposition.
- gsl_vector ∗ tau
  tau vector from QR decomposition
- gsl_vector ∗ diag
  Desc.
- gsl_vector ∗ qtf
  Desc.
- gsl_vector ∗ newton
  Desc.
- gsl_vector ∗ gradient
  Desc.
- gsl_vector ∗ df
  Desc.
- gsl_vector ∗ qtdf
  Desc.
- gsl_vector ∗ rdx
  Desc.
- gsl_vector ∗ w
  Desc.
- gsl_vector ∗ v
  Desc.
- size_t dim2
  Number of variables.

The documentation for this class was generated from the following file:

- gsl_mroot_hybrids.h

7.208 o2scl_interp Class Template Reference

`#include <interp.h>`

7.208.1 Detailed Description

template<class vec_t = ovector_view, class rvec_t = ovector_const_reverse> class o2scl_interp< vec_t, rvec_t >

Interpolation class.

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 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.

The class automatically handles decreasing arrays by converting from an object of type `vec_t` to an object ov type `rvec_t`.

While `vec_t` may be any vector type which allows indexing via [], `rvec_t` must be a vector type which allows indexing and has a constructor with one of the two forms

```
rvec_t::rvec_t(vec_t &v);
rvec_t::rvec_t(vec_t v);
```

so that o2scl_interp can automatically "reverse" a vector if necessary.

It is important that different instances of o2scl_interp_vec and o2scl_interp not be given the same interpolation objects, as they will clash.

Definition at line 962 of file interp.h.

Public Member Functions

- o2scl_interp (base_interp< vec_t > &it, base_interp< rvec_t > &rit)
  Create with base interpolation objects `it` *and* `rit`*.*
- o2scl_interp (base_interp< vec_t > &it)
  Create with base interpolation object `it` *and use def_ritp for reverse interpolation if necessary.*
- o2scl_interp ()
  Create an interpolator using def_itp and def_ritp.
- 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< vec_t > &it, base_interp< rvec_t > &rit)
  Set base interpolation object.

Data Fields

- cspline_interp< vec_t > def_itp
  Default base interpolation object (cubic spline with natural boundary conditions).
- cspline_interp< rvec_t > def_ritp
  Default base interpolation object for reversed vectors (cubic spline with natural boundary conditions).

Protected Attributes

- base_interp< vec_t > * itp
  Pointer to base interpolation object.
- base_interp< rvec_t > * ritp
  Pointer to base interpolation object for reversed vectors.

The documentation for this class was generated from the following file:

- interp.h

7.209 o2scl_interp_vec Class Template Reference

```
#include <interp.h>
```

7.209.1 Detailed Description

template<class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class o2scl_interp_vec< vec_t, alloc_vec_t, alloc_t >

Interpolation class for pre-specified vector.

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.

The class automatically handles decreasing arrays by copying the old array to a reversed version. For several interpolations on the same data, copying the initial array can be faster than overloading operator[].

The type of interpolation to be performed can be specified using the set_type() function. The default is cubic splines with natural boundary conditions.

It is important that different instances of o2scl_interp_vec and o2scl_interp not be given the same interpolation objects, as they will clash.

Todo

Need to fix constructor to behave properly if init() fails. It should free the memory and set `ln` to zero.

Definition at line 1238 of file interp.h.

Public Member Functions

- o2scl_interp_vec (base_interp< 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< vec_t > &it)
  Set base interpolation object.

Data Fields

- cspline_interp< vec_t > def_itp
  Default base interpolation object (cubic spline with natural boundary conditions).

Protected Attributes

- alloc_t ao
  Memory allocator for objects of type `alloc_vec_t`*.*
- base_interp< vec_t > * itp
  Pointer to base interpolation object.
- 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.
- alloc_vec_t lrx
  Reversed version of x.
- alloc_vec_t lry
  Reversed version of y.
- size_t ln
  Size of user-specified vectors.

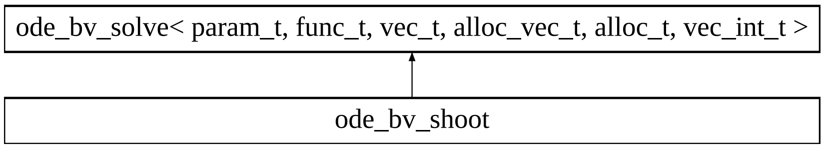
The documentation for this class was generated from the following file:

- interp.h

7.210 ode_bv_shoot Class Template Reference

`#include <ode_bv_solve.h>`

Inheritance diagram for ode_bv_shoot::



7.210.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_view> class ode_bv_shoot< param_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >

Solve boundary-value ODE problems by shooting.

Idea for future

Implement shooting from an internal point, either using a different class or using this one.

Definition at line 145 of file ode_bv_solve.h.

Public Member Functions

- virtual int solve (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_int_t &index, param_t &pa, func_t &derivs)
  Solve the boundary-value problem.

Protected Member Functions

- int solve_fun (size_t nv, const vec_t &sx, vec_t &sy, param_t &pa)
  Desc.

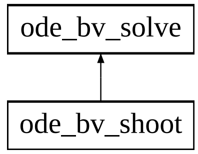
The documentation for this class was generated from the following file:

- ode_bv_solve.h

7.211 ode_bv_solve Class Template Reference

`#include <ode_bv_solve.h>`

Inheritance diagram for ode_bv_solve::



7.211.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class vec_int_t = ovector_int_view> class ode_bv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t, vec_int_t >

Solve boundary-value ODE problems.

Definition at line 42 of file ode_bv_solve.h.

Public Member Functions

- virtual int solve (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_int_t &index, param_t &pa, func_t &derivs)=0
  Solve the boundary-value problem.
- int set_iv (ode_iv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t > &ois)
  Set initial value solver.
- int set_mroot (mroot< param_t, mm_funct< param_t > > &root)
  Set the equation solver.

Data Fields

- ode_iv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t > def_ois
  The default initial value solver.
- gsl_mroot_hybrids< param_t, mm_funct< param_t > > def_mroot
  The default equation solver.
- int verbose
  Set output level.

Static Public Attributes

Values for the index array

- 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.

Protected Attributes

- ode_iv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t > ∗ oisp
  The solver for the initial value problem.
- mroot< param_t, mm_funct< param_t > > ∗ 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.
- 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.

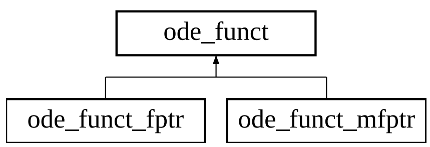
The documentation for this class was generated from the following file:

- ode_bv_solve.h

7.212 ode_funct Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for ode_funct::



7.212.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class ode_funct< param_t, vec_t >

Ordinary differential equation function [abstract base].

This base class provides the basic format for specifying ordinary differential equations to integrate with the O2scl ODE solvers. Select the appropriate child of this class according to the kind of functions which are to be given to the solver.

For functions with C-style arrays, use the corresponding children of ode_vfunct .

Definition at line 44 of file ode_funct.h.

Public Member Functions

- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &pa)=0
  The overloaded operator().

Private Member Functions

- **ode_funct** (const ode_funct &)
- ode_funct & **operator=** (const ode_funct &)

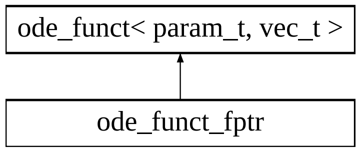
The documentation for this class was generated from the following file:

- ode_funct.h

7.213 ode_funct_fptr Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for ode_funct_fptr::



7.213.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class ode_funct_fptr< param_t, vec_t >

Provide ODE functions in the form of function pointers.

Definition at line 67 of file ode_funct.h.

Public Member Functions

- ode_funct_fptr (int(∗fp)(double, size_t, const vec_t &, vec_t &, param_t &))
  Create an object given a function pointer.
- virtual int operator() (double x, size_t nv, const vec_t &y, vec_t &dydx, param_t &pa)
  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.

Private Member Functions

- **ode_funct_fptr** (const ode_funct_fptr &)
- ode_funct_fptr & **operator=** (const ode_funct_fptr &)

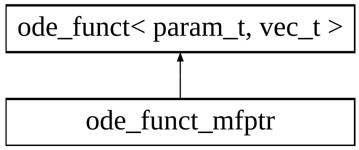
The documentation for this class was generated from the following file:

- ode_funct.h

7.214 ode_funct_mfptr Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for ode_funct_mfptr::



7.214.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class ode_funct_mfptr< tclass, param_t, vec_t >

Provide ODE functions in the form of member function pointers.

Definition at line 112 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, param_t &))
  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, param_t &pa)
  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.

Private Member Functions

- **ode_funct_mfptr** (const ode_funct_mfptr &)
- ode_funct_mfptr & **operator=** (const ode_funct_mfptr &)

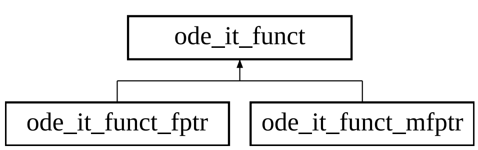
The documentation for this class was generated from the following file:

- ode_funct.h

7.215 ode_it_funct Class Template Reference

```
#include <ode_it_solve.h>
```

Inheritance diagram for ode_it_funct::



7.215.1 Detailed Description

template<class vec_t = o2scl::ovector_view> class ode_it_funct< vec_t >

Function class for ode_it_solve.

Definition at line 49 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`.

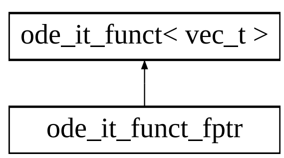
The documentation for this class was generated from the following file:

- ode_it_solve.h

7.216 ode_it_funct_fptr Class Template Reference

`#include <ode_it_solve.h>`

Inheritance diagram for ode_it_funct_fptr::



7.216.1 Detailed Description

template<class vec_t = o2scl::ovector_view> class ode_it_funct_fptr< vec_t >

Function pointer for ode_it_solve.

Definition at line 66 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.

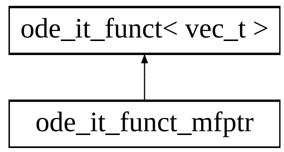
The documentation for this class was generated from the following file:

- ode_it_solve.h

7.217 ode_it_funct_mfptr Class Template Reference

#include <ode_it_solve.h>

Inheritance diagram for ode_it_funct_mfptr::



7.217.1 Detailed Description

template<class tclass, class vec_t = o2scl::ovector_view> class ode_it_funct_mfptr< tclass, vec_t >

Member function pointer for ode_it_solve.

Definition at line 92 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

7.218 ode_it_make_Coord Class Reference

#include <ode_it_solve.h>

7.218.1 Detailed Description

Make a coordinate matrix for ode_it_solve.

Definition at line 197 of file ode_it_solve.h.

Public Member Functions

- o2scl::Coord_Mat ∗ make (size_t ngrid, size_t neq, size_t nbleft)
  Create a compressed-column format matrix for ode_it_solve.

Data Fields

- o2scl::uvector_int * r
  The row index.
- o2scl::uvector_int * c
  The column pointer.
- o2scl::uvector * vals
  The matrix entries.

The documentation for this class was generated from the following file:

- ode_it_solve.h

7.219 ode_it_solve Class Template Reference

```
#include <ode_it_solve.h>
```

7.219.1 Detailed Description

template<class func_t, class vec_t, class mat_t, class matrix_row_t, class solver_vec_t, class solver_mat_t> class ode_it_-solve< func_t, vec_t, mat_t, matrix_row_t, solver_vec_t, solver_mat_t >

ODE solver using a generic linear solver to solve finite-difference equations.

Todo

Max and average tolerance?

Todo

partial correction option?

Definition at line 270 of file ode_it_solve.h.

Public Member Functions

- int set_solver (linear_solver< solver_vec_t, solver_mat_t > &ls)
  Set the linear solver.
- int solve (size_t ngrid, size_t neq, size_t nbleft, 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

- int verbose
  Set level of output (default 0).
- double h
  Stepsize for finite differencing (default 10^{-4}).
- double tolf
  Tolerance (default 10^{-8}).
- size_t niter
  Maximum number of iterations (default 30).

Protected Member Functions

- double fd_left (size_t ieq, size_t ivar, double x, vec_t &y)
  Compute the derivatives of the LHS boundary conditions.
- double fd_right (size_t ieq, size_t ivar, double x, vec_t &y)
  Compute the derivatives of the RHS boundary conditions.
- 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

- 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**

The documentation for this class was generated from the following file:

- ode_it_solve.h

7.220 ode_iv_solve Class Template Reference

```
#include <ode_iv_solve.h>
```

7.220.1 Detailed Description

template<class param_t, class func_t = ode_funct<param_t>, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc, class mat_row_t = omatrix_row> class ode_iv_solve< param_t, func_t, vec_t, alloc_vec_t, alloc_t, mat_row_t >

Solve an initial-value ODE problems given an adaptive ODE stepper.

Definition at line 42 of file ode_iv_solve.h.

Public Member Functions

- int set_adapt_step (adapt_step< param_t, func_t, vec_t, alloc_vec_t, alloc_t > &as)
  Set the adaptive stepper to use.
- template<class mat_t>
  int solve_table (double x0, double x1, double h, size_t n, vec_t &ystart, size_t &nsol, vec_t &xsol, mat_t &ysol, param_t &pa, func_t &derivs)
  Solve the initial-value problem and output a table.
- template<class mat_t>
  int solve_grid (double x0, double x1, double h, size_t n, vec_t &ystart, size_t nsol, vec_t &xsol, mat_t &ysol, param_t &pa, func_t &derivs)
  Solve the initial-value problem from `x0` to `x1` over a grid.
- template<class mat_t>
  int solve_grid_derivs (double x0, double x1, double h, size_t n, vec_t &ystart, size_t nsol, vec_t &xsol, mat_t &ysol, mat_t &dydx_sol, param_t &pa, func_t &derivs)
  Solve the initial-value problem from `x0` to `x1` over a grid storing derivatives.

- int solve_final_value (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, param_t &pa, func_t &derivs)
  Solve the initial-value problem to get the final value.
- int solve_final_value_derivs (double x0, double x1, double h, size_t n, vec_t &ystart, vec_t &yend, vec_t &dydx_start, vec_t &dydx_end, param_t &pa, func_t &derivs)
  Solve the initial-value problem to get the final value and derivative.
- virtual const char * type ()
  Return the type, `"ode_iv_solve"`.

Data Fields

- int verbose
  Set output level.
- size_t nsteps_out
  Number of output points if verbose is greater than zero (default 10).
- int ntrial
  Maximum number of steps for solve_final_value() (default 1000).
- bool exit_on_fail
  If true, stop the solution if the adaptive stepper fails.
- gsl_astep< param_t, func_t, vec_t, alloc_vec_t, alloc_t > gsl_astp
  The default adaptive stepper.

Protected Member Functions

- virtual int print_iter (double x, size_t nv, vec_t &y)
  Print out iteration information.

Protected Attributes

- alloc_vec_t dydx
  Derivative.
- alloc_t ao
  Memory allocator.
- adapt_step< param_t, func_t, vec_t, alloc_vec_t, alloc_t > * astp
  The adaptive stepper.

7.220.2 Member Function Documentation

7.220.2.1 int solve_table (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, size_t & *nsol*, vec_t & *xsol*, mat_t & *ysol*, param_t & *pa*, func_t & *derivs*) `[inline]`

Solve the initial-value problem and output a table.

Initially, `xsol` should be a vector of size `nsol`, and `ysol` should be a two-dimensional array (i.e. omatrix_view) of size [nsol][n]. On exit, `nsol` will will be the size of the solution table, less than or equal to the original value of `nsol`.

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.

If the given value of `h` is small enough, the solution may generate more points than the space initially allocated and the full solution will not be generated.

Idea for future

Consider modifying so that this can handle tables which are too small by removing half the rows and doubling the stepsize.

Definition at line 114 of file ode_iv_solve.h.

7.220.2.2 int solve_grid (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, size_t *nsol*, vec_t & *xsol*, mat_t & *ysol*, param_t & *pa*, func_t & *derivs*) `[inline]`

Solve the initial-value problem from `x0` to `x1` over a grid.

Initially, `xsol` should be an array of size `nsol`, and `ysol` should be a `matrix` 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.

Definition at line 190 of file ode_iv_solve.h.

7.220.2.3 int solve_grid_derivs (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, size_t *nsol*, vec_t & *xsol*, mat_t & *ysol*, mat_t & *dydx_sol*, param_t & *pa*, func_t & *derivs*) `[inline]`

Solve the initial-value problem from `x0` to `x1` over a grid storing derivatives.

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.

Todo

Add error information

Definition at line 286 of file ode_iv_solve.h.

7.220.2.4 int solve_final_value (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, vec_t & *yend*, param_t & *pa*, func_t & *derivs*) `[inline]`

Solve the initial-value problem to get the final value.

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.

The solution fails if more than ntrial steps are required.

Definition at line 391 of file ode_iv_solve.h.

7.220.2.5 int solve_final_value_derivs (double *x0*, double *x1*, double *h*, size_t *n*, vec_t & *ystart*, vec_t & *yend*, vec_t & *dydx_start*, vec_t & *dydx_end*, param_t & *pa*, func_t & *derivs*) `[inline]`

Solve the initial-value problem to get the final value and derivative.

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.

The solution fails if more than ntrial steps are required.

Todo

Add error information

Definition at line 415 of file ode_iv_solve.h.

7.220.3 Field Documentation

7.220.3.1 bool exit_on_fail

If true, stop the solution if the adaptive stepper fails.

If this is false, then failures in the adaptive stepper are ignored.

Definition at line 546 of file ode_iv_solve.h.

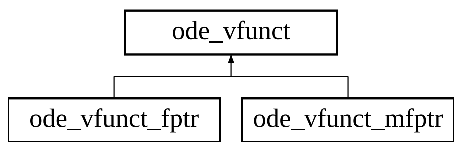
The documentation for this class was generated from the following file:

- ode_iv_solve.h

7.221 ode_vfunct Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for ode_vfunct::



7.221.1 Detailed Description

template<class param_t, size_t nv> class ode_vfunct< param_t, nv >

Ordinary differential equation function with arrays [abstract base].

This base class provides the basic format for specifying ordinary differential equations to integrate with the O2scl 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 167 of file ode_funct.h.

Public Member Functions

- virtual int operator() (double x, size_t nvar, const double y[nv], double dydx[nv], param_t &pa)=0
  Compute the `nv` *derivatives as a function of the* `nv` *functions specified in* `y` *at the point* `x`*.*

Private Member Functions

- **ode_vfunct** (const ode_vfunct &)
- ode_vfunct & **operator=** (const ode_vfunct &)

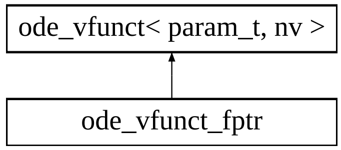
The documentation for this class was generated from the following file:

- ode_funct.h

7.222 ode_vfunct_fptr Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for ode_vfunct_fptr::



7.222.1 Detailed Description

template<class param_t, size_t nv> class ode_vfunct_fptr< param_t, nv >

Function pointer to a function.

Definition at line 192 of file ode_funct.h.

Public Member Functions

- ode_vfunct_fptr (int(∗fp)(double, size_t, const double y[nv], double dydx[nv], param_t &))
  Specify the function pointer.
- virtual int operator() (double x, size_t nvar, const double y[nv], double dydx[nv], param_t &pa)
  The overloaded operator().

Protected Attributes

- int(∗ fptr)(double x, size_t nvar, const double y[nv], double dydx[nv], param_t &)
  The function pointer.

Private Member Functions

- **ode_vfunct_fptr** (const ode_vfunct_fptr &)
- ode_vfunct_fptr & **operator=** (const ode_vfunct_fptr &)

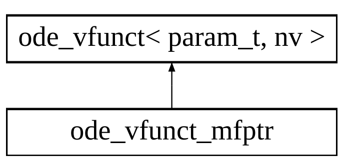
The documentation for this class was generated from the following file:

- ode_funct.h

7.223 ode_vfunct_mfptr Class Template Reference

```
#include <ode_funct.h>
```

Inheritance diagram for ode_vfunct_mfptr::



7.223.1 Detailed Description

template<class tclass, class param_t, size_t nv> class ode_vfunct_mfptr< tclass, param_t, nv >

Provide ODE functions in the form of member function pointers.

Definition at line 236 of file ode_funct.h.

Public Member Functions

- ode_vfunct_mfptr (tclass *tp, int(tclass::*fp)(double x, size_t nvar, const double y[nv], double dydx[nv], param_t &))
  Specify the member function pointer.
- virtual int operator() (double x, size_t nvar, const double y[nv], double dydx[nv], param_t &pa)
  The overloaded operator().

Protected Attributes

- int(tclass::* fptr)(double x, size_t nvar, const double y[nv], double dydx[nv], param_t &)
  Pointer to the member function.
- tclass * tptr
  Pointer to the class.

Private Member Functions

- **ode_vfunct_mfptr** (const ode_vfunct_mfptr &)
- ode_vfunct_mfptr & **operator=** (const ode_vfunct_mfptr &)

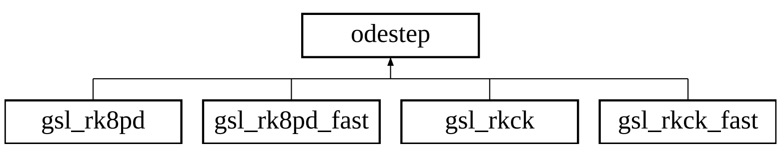
The documentation for this class was generated from the following file:

- ode_funct.h

7.224 odestep Class Template Reference

```
#include <odestep.h>
```

Inheritance diagram for odestep::



7.224.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view> class odestep< param_t, func_t, vec_t >

ODE stepper base [abstract base].

Note:

This base class does not actually perform any ODE solving use gsl_rkck or gsl_rk8pd.

Definition at line 40 of file odestep.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, param_t &pa, func_t &derivs)=0
  Perform an integration step.

Protected Attributes

- int order
  The order of the ODE stepper.

7.224.2 Member Function Documentation

7.224.2.1 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*, param_t & *pa*, func_t & *derivs*) `[pure virtual]`

Perform an integration step.

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.

Implemented in gsl_rk8pd, gsl_rk8pd_fast, gsl_rkck, and gsl_rkck_fast.

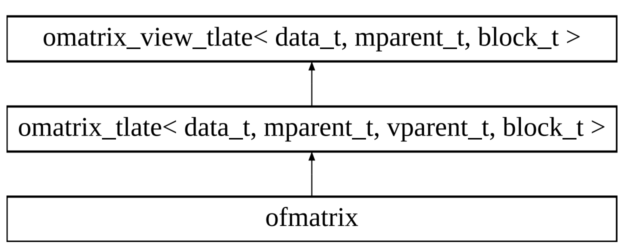
The documentation for this class was generated from the following file:

- odestep.h

7.225 ofmatrix Class Template Reference

`#include <omatrix_tlate.h>`

Inheritance diagram for ofmatrix::



7.225.1 Detailed Description

template<size_t N, size_t M> class ofmatrix< N, M >

A matrix where the memory allocation is performed in the constructor.

Definition at line 958 of file omatrix_tlate.h.

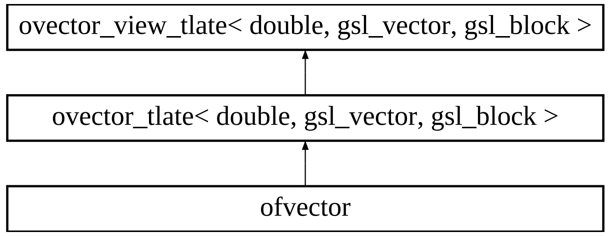
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.226 ofvector Class Template Reference

#include <ovector_tlate.h>

Inheritance diagram for ofvector::



7.226.1 Detailed Description

template<size_t N = 0> class ofvector< N >

A vector where the memory allocation is performed in the constructor.

Definition at line 2025 of file ovector_tlate.h.

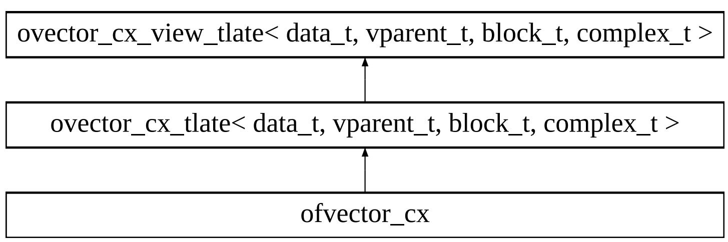
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.227 ofvector_cx Class Template Reference

#include <ovector_cx_tlate.h>

Inheritance diagram for ofvector_cx::



7.227.1 Detailed Description

template<size_t N = 0> class ofvector_cx< N >

A vector where the memory allocation is performed in the constructor.

Definition at line 1023 of file ovector_cx_tlate.h.

The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.228 omatrix_alloc Class Reference

`#include <omatrix_tlate.h>`

7.228.1 Detailed Description

A simple class to provide an `allocate()` function for omatrix.

Definition at line 946 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.

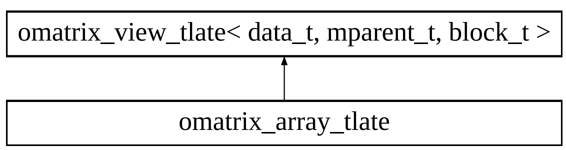
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.229 omatrix_array_tlate Class Template Reference

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_array_tlate::



7.229.1 Detailed Description

template<class data_t, class mparent_t, class block_t> class omatrix_array_tlate< data_t, mparent_t, block_t >

Create a matrix from an array.

Definition at line 697 of file omatrix_tlate.h.

Public Member Functions

- omatrix_array_tlate (size_t tot, data_t *dat, size_t start, size_t ltda, size_t sz1, size_t sz2)
  Create a vector from `dat` *with size* `n`.

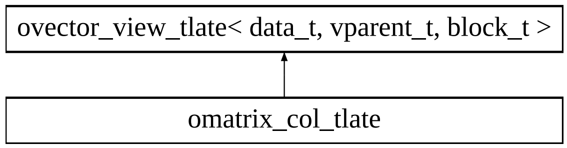
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.230 omatrix_col_tlate Class Template Reference

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_col_tlate::



7.230.1 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 >

Create a vector from a column of a matrix.

Definition at line 779 of file omatrix_tlate.h.

Public Member Functions

- omatrix_col_tlate (omatrix_view_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from col `i` *of matrix* `m`.

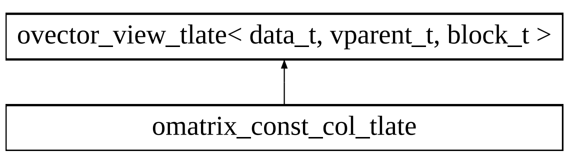
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.231 omatrix_const_col_tlate Class Template Reference

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_const_col_tlate::



7.231.1 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 >

Create a const vector from a column of a matrix.

Definition at line 804 of file omatrix_tlate.h.

Public Member Functions

- omatrix_const_col_tlate (omatrix_view_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from col `i` *of matrix* `m`.

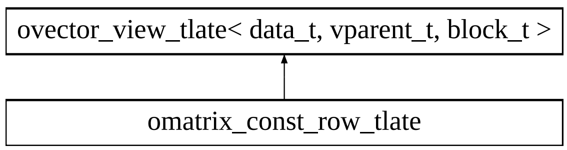
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.232 omatrix_const_row_tlate Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for omatrix_const_row_tlate::



7.232.1 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 >

Create a const vector from a row of a matrix.

Definition at line 753 of file omatrix_tlate.h.

Public Member Functions

- omatrix_const_row_tlate (const omatrix_view_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from row `i` *of matrix* `m`.

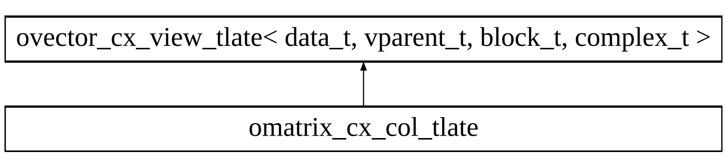
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.233 omatrix_cx_col_tlate Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for omatrix_cx_col_tlate::



7.233.1 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 >

Create a vector from a column of a matrix.

Definition at line 670 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`.

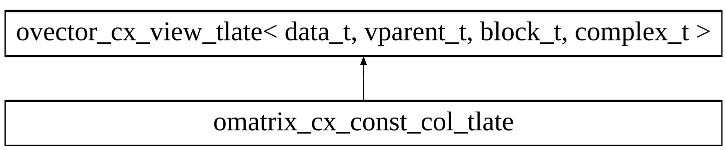
The documentation for this class was generated from the following file:

- omatrix_cx_tlate.h

7.234 omatrix_cx_const_col_tlate Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for omatrix_cx_const_col_tlate::



7.234.1 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 >

Create a vector from a column of a matrix.

Definition at line 690 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`.

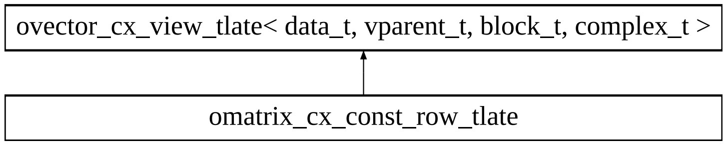
The documentation for this class was generated from the following file:

- omatrix_cx_tlate.h

7.235 omatrix_cx_const_row_tlate Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for omatrix_cx_const_row_tlate::



7.235.1 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 >

Create a vector from a row of a matrix.

Definition at line 650 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`*.*

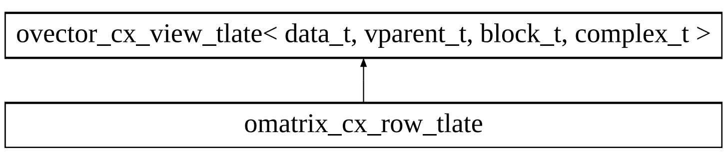
The documentation for this class was generated from the following file:

- omatrix_cx_tlate.h

7.236 omatrix_cx_row_tlate Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for omatrix_cx_row_tlate::



7.236.1 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 >

Create a vector from a row of a matrix.

Definition at line 630 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`*.*

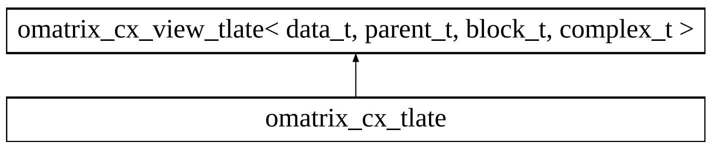
The documentation for this class was generated from the following file:

- omatrix_cx_tlate.h

7.237 omatrix_cx_tlate Class Template Reference

```
#include <omatrix_cx_tlate.h>
```

Inheritance diagram for omatrix_cx_tlate::



7.237.1 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 >

A matrix of double-precision numbers.

Definition at line 398 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.

7.237.2 Member Function Documentation

7.237.2.1 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 584 of file omatrix_cx_tlate.h.

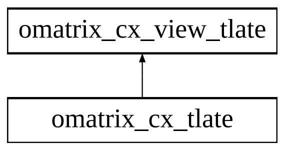
The documentation for this class was generated from the following file:

- omatrix_cx_tlate.h

7.238 omatrix_cx_view_tlate Class Template Reference

#include <omatrix_cx_tlate.h>

Inheritance diagram for omatrix_cx_view_tlate::



7.238.1 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 >

A matrix view of double-precision numbers.

Definition at line 49 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.

7.238.2 Member Function Documentation

7.238.2.1 size_t rows () const `[inline]`

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

Definition at line 346 of file omatrix_cx_tlate.h.

7.238.2.2 size_t cols () const `[inline]`

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 356 of file omatrix_cx_tlate.h.

7.238.2.3 size_t tda () const `[inline]`

Method to return matrix tda.

If no memory has been allocated, this will quietly return zero.

Definition at line 366 of file omatrix_cx_tlate.h.

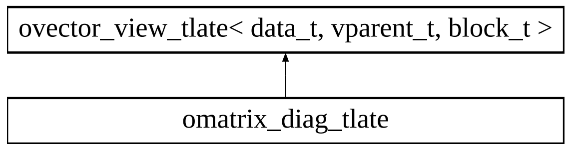
The documentation for this class was generated from the following file:

- omatrix_cx_tlate.h

7.239 omatrix_diag_tlate Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for omatrix_diag_tlate::



7.239.1 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 >

Create a vector from the main diagonal.

Definition at line 829 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`.

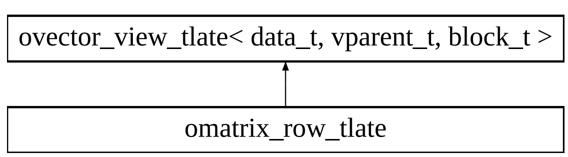
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.240 omatrix_row_tlate Class Template Reference

```
#include <omatrix_tlate.h>
```

Inheritance diagram for omatrix_row_tlate::



7.240.1 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 >

Create a vector from a row of a matrix.

Definition at line 728 of file omatrix_tlate.h.

Public Member Functions

- omatrix_row_tlate (omatrix_view_tlate< data_t, mparent_t, block_t > &m, size_t i)
  Create a vector from row `i` *of matrix* `m`.

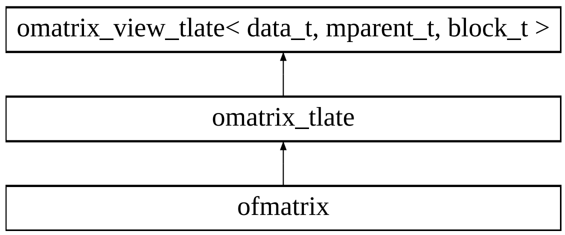
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.241 omatrix_tlate Class Template Reference

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_tlate::



7.241.1 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 >

A matrix of double-precision numbers.

Definition at line 381 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_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_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_view_tlate< data_t, vparent_t, block_t > ova[ ])
  Deep copy from an array of ovectors.
- omatrix_tlate (size_t n, uvector_view_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).

7.241.2 Member Function Documentation

7.241.2.1 int free () `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 656 of file omatrix_tlate.h.

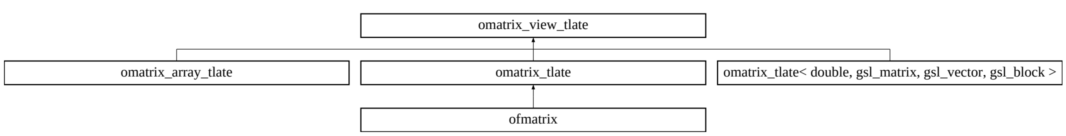
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.242 omatrix_view_tlate Class Template Reference

`#include <omatrix_tlate.h>`

Inheritance diagram for omatrix_view_tlate::



7.242.1 Detailed Description

template<class data_t, class mparent_t, class block_t> class omatrix_view_tlate< data_t, mparent_t, block_t >

A matrix view of double-precision numbers.

Idea for future

This class isn't sufficiently general for some applications, such as sub-matrices of higher-dimensional structures. It might be nice to create a more general class with a "stride" and a "tda".

Idea for future

The `xmatrix` class demonstrates how operator[] could return an ovector_array object and thus provide better bounds-checking. This would demand including a new parameter in omatrix_view_tlate which contains the vector type.

Definition at line 62 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.

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 (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).
- 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`*.*
- 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.
- mparent_t ∗ get_gsl_matrix ()
  Return a gsl matrix.
- const mparent_t ∗ get_gsl_matrix_const () const
  Return a `const` *gsl matrix.*

Arithmetic

- omatrix_view_tlate< data_t, mparent_t, block_t > & operator+= (const omatrix_view_tlate< data_t, mparent_t, block_t > &x)
  operator+=
- omatrix_view_tlate< data_t, mparent_t, block_t > & operator-= (const omatrix_view_tlate< data_t, mparent_t, block_t > &x)
  operator-=
- omatrix_view_tlate< data_t, mparent_t, block_t > & operator+= (const data_t &y)
  operator+=
- omatrix_view_tlate< data_t, mparent_t, block_t > & operator-= (const data_t &y)
  operator-=
- omatrix_view_tlate< data_t, mparent_t, block_t > & operator∗= (const data_t &y)
  operator∗=

Protected Member Functions

- omatrix_view_tlate ()
  Empty constructor provided for use by omatrix_tlate(const omatrix_tlate &v).

7.242.2 Member Function Documentation

7.242.2.1 size_t rows () const [inline]

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

Definition at line 252 of file omatrix_tlate.h.

7.242.2.2 size_t cols () const [inline]

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 262 of file omatrix_tlate.h.

7.242.2.3 size_t tda () const [inline]

Method to return matrix tda.

If no memory has been allocated, this will quietly return zero.

Definition at line 272 of file omatrix_tlate.h.

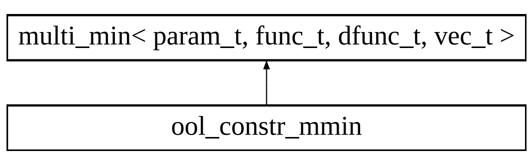
The documentation for this class was generated from the following file:

- omatrix_tlate.h

7.243 ool_constr_mmin Class Template Reference

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_constr_mmin::



7.243.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_view, class alloc_-vec_t = ovector, class alloc_t = ovector_alloc> class ool_constr_mmin< param_t, func_t, dfunc_t, hfunc_t, vec_t, alloc_vec_t, alloc_t >

Constrained multidimensional minimization base (OOL).

Todo

Implement automatic computations of gradient and Hessian

Todo

Construct a non-trivial example for the "examples" directory

Todo

Finish mmin() interface

Definition at line 304 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, param_t &par)
  Set the function, the initial guess, and the parameters.
- virtual int set_hess (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 set_constraints (size_t nc, vec_t &lower, vec_t &upper)
  Set the constraints.
- virtual int iterate ()
  Perform an iteration.
- virtual int is_optimal ()
  See if we're finished.
- virtual int mmin (size_t nvar, vec_t &xx, double &fmin, param_t &pa, 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, param_t &pa, 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, param_t &pa, 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").

Static Public Attributes

OOL-specific error codes

- static const int OOL_UNBOUNDEDF = 1101
  Lower unbounded function.
- static const int OOL_INFEASIBLE = 1102
  Infeasible point.
- static const int OOL_FINNERIT = 1103
  Too many inner iterations.
- static const int OOL_FLSEARCH = 1104
  Line search failed.
- static const int OOL_FDDIR = 1105
  Unable to find a descent direction.

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.
- param_t * param
  User-specified parameters.
- 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.

7.243.2 Member Function Documentation

7.243.2.1 int calc_Hv (const size_t *nind*, gsl_vector_uint * *Ind*, vec_t & *X*, vec_t & *Xc*, vec_t & *V*, vec_t & *Hv*) `[inline, protected]`

Evaluate a hessian times a vector from the reduced space.

Expand to full space

Definition at line 459 of file ool_constr_mmin.h.

7.243.2.2 virtual int mmin (size_t *nvar*, vec_t & *xx*, double & *fmin*, param_t & *pa*, func_t & *ff*) [inline, virtual]

Calculate the minimum `min` of `func` w.r.t. the array `x` of size `nvar`.

Todo

Need to finish this function somehow since it's pure virtual in multi_min.

Implements multi_min.

Definition at line 614 of file ool_constr_mmin.h.

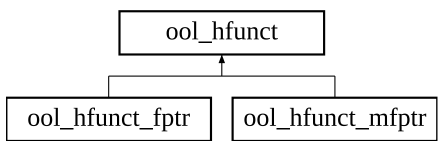
The documentation for this class was generated from the following file:

- ool_constr_mmin.h

7.244 ool_hfunct Class Template Reference

`#include <ool_constr_mmin.h>`

Inheritance diagram for ool_hfunct::



7.244.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class ool_hfunct< param_t, vec_t >

Hessian product function for ool_constr_mmin [abstract base].

Definition at line 43 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, param_t &pa)=0
  Evaluate $H(x) \cdot v$.

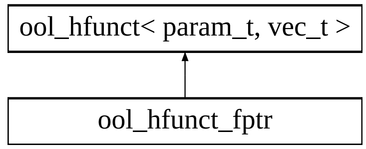
The documentation for this class was generated from the following file:

- ool_constr_mmin.h

7.245 ool_hfunct_fptr Class Template Reference

`#include <ool_constr_mmin.h>`

Inheritance diagram for ool_hfunct_fptr::



7.245.1 Detailed Description

template<class param_t, class vec_t = ovector_view> class ool_hfunct_fptr< param_t, vec_t >

A hessian product supplied by a function pointer.

Definition at line 70 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, param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)
  Evaluate $H(x) \cdot v$.

Protected Attributes

- int(∗ fptr)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)
  Store the function pointer.

Private Member Functions

- **ool_hfunct_fptr** (const ool_hfunct_fptr &)
- ool_hfunct_fptr & **operator=** (const ool_hfunct_fptr &)

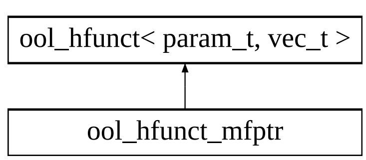
The documentation for this class was generated from the following file:

- ool_constr_mmin.h

7.246 ool_hfunct_mfptr Class Template Reference

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_hfunct_mfptr::



7.246.1 Detailed Description

template<class tclass, class param_t, class vec_t = ovector_view> class ool_hfunct_mfptr< tclass, param_t, vec_t >

A hessian product supplied by a member function pointer.

Definition at line 118 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, param_t &pa))
  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, param_t &pa)
  Evaluate $H(x) \cdot v$.

Protected Attributes

- int(tclass::∗ fptr)(size_t nv, const vec_t &x, const vec_t &v, vec_t &hv, param_t &pa)
  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 &)

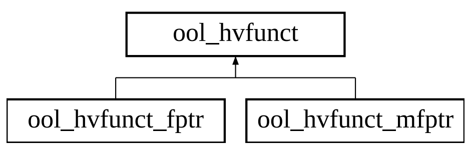
The documentation for this class was generated from the following file:

- ool_constr_mmin.h

7.247 ool_hvfunct Class Template Reference

#include <ool_constr_mmin.h>

Inheritance diagram for ool_hvfunct::



7.247.1 Detailed Description

template<class param_t, size_t nvar> class ool_hvfunct< param_t, nvar >

Hessian product function base for ool_constr_mmin using arrays.

Definition at line 168 of file ool_constr_mmin.h.

Public Member Functions

- virtual int operator() (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)=0
  Evaluate $H(x) \cdot v$.

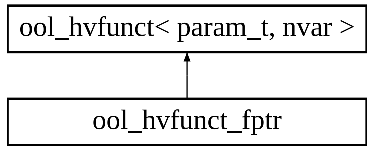
The documentation for this class was generated from the following file:

- ool_constr_mmin.h

7.248 ool_hvfunct_fptr Class Template Reference

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_hvfunct_fptr::



7.248.1 Detailed Description

template<class param_t, size_t nvar> class ool_hvfunct_fptr< param_t, nvar >

A hessian product supplied by a function pointer using arrays.

Definition at line 196 of file ool_constr_mmin.h.

Public Member Functions

- ool_hvfunct_fptr (int(∗fp)(size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa))
  Specify the function pointer.
- virtual int operator() (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
  Evaluate $H(x) \cdot v$.

Protected Attributes

- int(∗ fptr)(size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
  Store the function pointer.

Private Member Functions

- **ool_hvfunct_fptr** (const ool_hvfunct_fptr &)
- ool_hvfunct_fptr & **operator=** (const ool_hvfunct_fptr &)

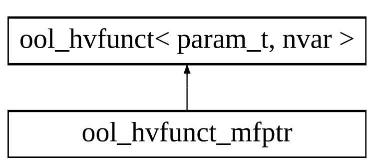
The documentation for this class was generated from the following file:

- ool_constr_mmin.h

7.249 ool_hvfunct_mfptr Class Template Reference

```
#include <ool_constr_mmin.h>
```

Inheritance diagram for ool_hvfunct_mfptr::



7.249.1 Detailed Description

template<class tclass, class param_t, size_t nvar> class ool_hvfunct_mfptr< tclass, param_t, nvar >

A hessian product supplied by a member function pointer using arrays.

Definition at line 247 of file ool_constr_mmin.h.

Public Member Functions

- ool_hvfunct_mfptr (tclass ∗tp, int(tclass::∗fp)(size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa))
  Specify the member function pointer.
- virtual int operator() (size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
  Evaluate $H(x) \cdot v$.

Protected Attributes

- int(tclass::∗ fptr)(size_t nv, const double x[nvar], const double v[nvar], double hv[nvar], param_t &pa)
  Store the function pointer.
- tclass ∗ tptr
  Store a pointer to the class instance.

Private Member Functions

- **ool_hvfunct_mfptr** (const ool_hvfunct_mfptr &)
- ool_hvfunct_mfptr & **operator=** (const ool_hvfunct_mfptr &)

The documentation for this class was generated from the following file:

- ool_constr_mmin.h

7.250 ool_mmin_gencan Class Template Reference

```
#include <ool_mmin_gencan.h>
```

7.250.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t, class hfunc_t = func_t, class vec_t = ovector_view, 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 >

Constrained minimization by the "GENCAN" method (OOL).

Note:

Not working yet

Definition at line 47 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.

7.250.2 Field Documentation

7.250.2.1 double fmin

Minimum function value (default 10^{-99}).

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 685 of file ool_mmin_gencan.h.

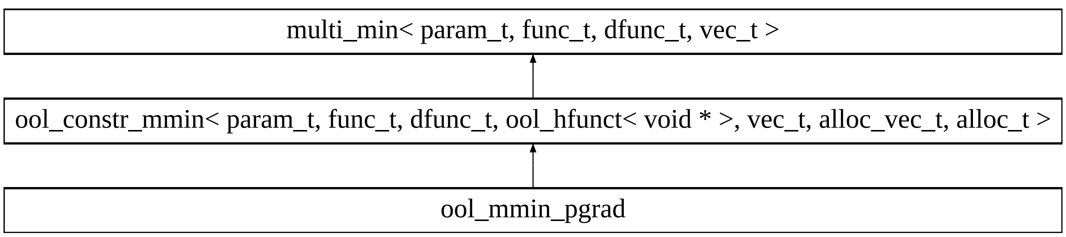
The documentation for this class was generated from the following file:

- ool_mmin_gencan.h

7.251 ool_mmin_pgrad Class Template Reference

```
#include <ool_mmin_pgrad.h>
```

Inheritance diagram for ool_mmin_pgrad::



7.251.1 Detailed Description

template<class param_t, class func_t, class dfunc_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class ool_mmin_pgrad< param_t, func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >

Constrained minimization by the projected gradient method (OOL).

Todo

Complete the mmin() interface with automatic gradient

Todo

Replace the explicit norm computation below with the more accurate dnrm2 from linalg

Definition at line 50 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, 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_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< void * > 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.

7.251.2 Field Documentation

7.251.2.1 double fmin

Minimum function value (default 10^{-99}).

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 140 of file ool_mmin_pgrad.h.

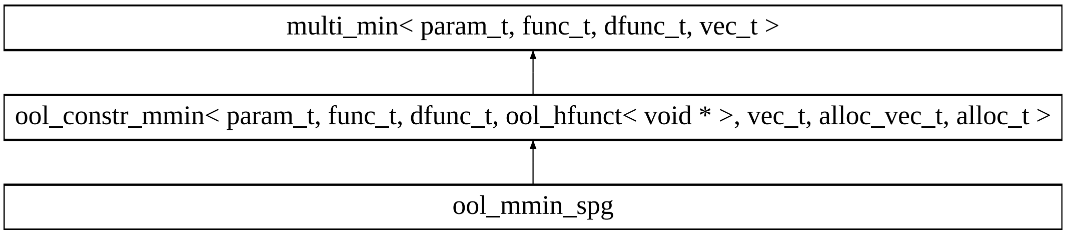
The documentation for this class was generated from the following file:

- ool_mmin_pgrad.h

7.252 ool_mmin_spg Class Template Reference

`#include <ool_mmin_spg.h>`

Inheritance diagram for ool_mmin_spg::



7.252.1 Detailed Description

template<class param_t, class func_t, class dfunc_t, class vec_t = ovector_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class ool_mmin_spg< param_t, func_t, dfunc_t, vec_t, alloc_vec_t, alloc_t >

Constrained minimization by the spectral projected gradient method (OOL).

Definition at line 46 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, 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_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< void * > 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.

7.252.2 Field Documentation

7.252.2.1 double fmin

Minimum function value (default 10^{-99}).

If the function value is below this value, then the algorithm assumes that the function is not bounded and exits.

Definition at line 194 of file ool_mmin_spg.h.

The documentation for this class was generated from the following file:

- ool_mmin_spg.h

7.253 other_ioc Class Reference

```
#include <other_ioc.h>
```

7.253.1 Detailed Description

Setup I/O for series acceleration.

Definition at line 36 of file other_ioc.h.

Data Fields

- gsl_series_io_type ∗ **gsl_series_io**

The documentation for this class was generated from the following file:

- other_ioc.h

7.254 other_todos_and_bugs Class Reference

`#include <main.h>`

7.254.1 Detailed Description

An empty class to add some items to the todo and bug lists.

Todo

- The o2scl-test and o2scl-examples targets require grep, awk, tail, cat, and wc. It would be good to reduce this list to ensure better compatibility.
- More examples and benchmarks
- Document a list of all global functions and operators
- Make sure we have a uvector_alloc, 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

There may be a problem with const-correctness in vectors. I'm not sure how it's best solved. It could be best to create two kinds of ovector_view's: one const and one not. 10/19/07: I think it's the case that neither ovector_const_subvector, or const ovector_subvector are truly const, but it's only const ovector_const_subvector that would be truly const. I'm not sure if this is related to the issue of constness in ovector_view discussed above.

Idea for future

Consider breaking documentation up into sections?

Bug

- BLAS libraries not named `libblas` or `libgslblas` are not properly detected in ./configure and will have to be added manually.

- The -lm flag may not be added properly by ./configure

Definition at line 1844 of file main.h.

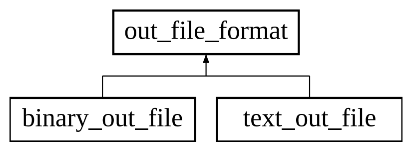
The documentation for this class was generated from the following file:

- main.h

7.255 out_file_format Class Reference

```
#include <file_format.h>
```

Inheritance diagram for out_file_format::



7.255.1 Detailed Description

Class for output file formats [abstract base].

Definition at line 44 of file file_format.h.

Public Member Functions

- virtual int bool_out (bool dat, std::string name="")=0
  Output a bool variable.
- virtual int char_out (char dat, std::string name="")=0
  Output a char variable.
- virtual int float_out (float dat, std::string name="")=0
  Output a float variable.
- virtual int double_out (double dat, std::string name="")=0
  Output a double variable.
- virtual int int_out (int dat, std::string name="")=0
  Output an int variable.
- virtual int long_out (unsigned long int dat, std::string name="")=0
  Output an long variable.
- virtual int string_out (std::string dat, std::string name="")=0
  Output a string.
- virtual int word_out (std::string dat, std::string name="")=0
  Output a word.
- virtual int start_object (std::string type, std::string name="")=0
  Start object output.
- virtual int end_object ()=0
  End object output.
- virtual int end_line ()=0
  End a line of output.
- virtual int init_file ()=0
  Output initialization.
- virtual int clean_up ()=0
  Finish the file.

The documentation for this class was generated from the following file:

- file_format.h

7.256 ovector_alloc Class Reference

```
#include <ovector_tlate.h>
```

7.256.1 Detailed Description

A simple class to provide an `allocate()` function for ovector.

Definition at line 2001 of file ovector_tlate.h.

Public Member Functions

- void allocate (ovector &o, int i)
  Allocate `v` *for* `i` *elements.*
- void free (ovector &o)
  Free memory.

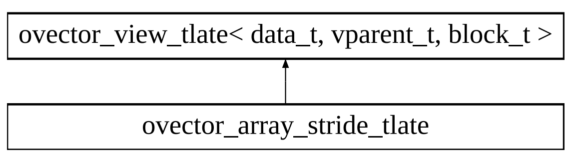
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.257 ovector_array_stride_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_array_stride_tlate::



7.257.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_array_stride_tlate< data_t, vparent_t, block_t >

Create a vector from an array with a stride.

Definition at line 1050 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`*.*

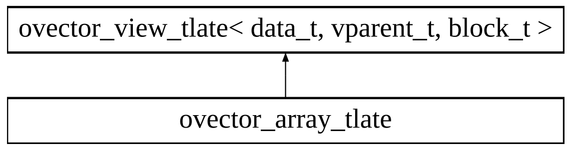
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.258 ovector_array_tlate Class Template Reference

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_array_tlate::



7.258.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_array_tlate< data_t, vparent_t, block_t >

Create a vector from an array.

Definition at line 1031 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`.

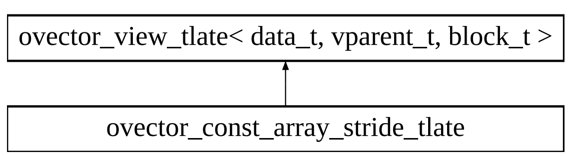
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.259 ovector_const_array_stride_tlate Class Template Reference

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_const_array_stride_tlate::



7.259.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_const_array_stride_tlate< data_t, vparent_t, block_t >

Create a const vector from an array with a stride.

Definition at line 1197 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`.

- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.

Protected Member Functions

Ensure \c const by hiding non-const members

- 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)
  Set (with optional range-checking).
- int swap (ovector_view_tlate< data_t, vparent_t, block_t > &x)
  Swap vectors.
- int set_all (double val)
  Set all of the value to be the value `val`.
- vparent_t ∗ get_gsl_vector ()
  Return a gsl vector.
- ovector_view_tlate< data_t, vparent_t, block_t > & operator+= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator+=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator-= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator-=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator∗= (const data_t &y)
  operator∗=

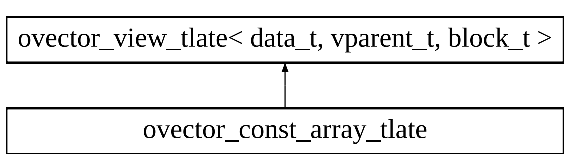
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.260 ovector_const_array_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_const_array_tlate::



7.260.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_const_array_tlate< data_t, vparent_t, block_t >

Create a const vector from an array.

Definition at line 1097 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`*.*
- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.

Protected Member Functions

Ensure \c const by hiding non-const members

- 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)
  Set (with optional range-checking).
- int swap (ovector_view_tlate< data_t, vparent_t, block_t > &x)
  Swap vectors.
- int set_all (double val)
  Set all of the value to be the value `val`*.*
- vparent_t ∗ get_gsl_vector ()
  Return a gsl vector.
- ovector_view_tlate< data_t, vparent_t, block_t > & operator+= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator+=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator-= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator-=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator∗= (const data_t &y)
  operator∗=

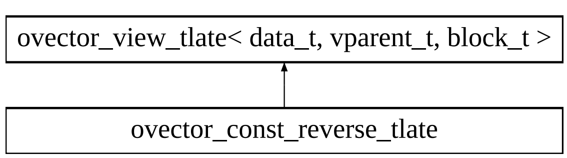
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.261 ovector_const_reverse_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_const_reverse_tlate::



7.261.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_const_reverse_tlate< data_t, vparent_t, block_t >

Reversed view of a vector.

Definition at line 1562 of file ovector_tlate.h.

Public Member Functions

- ovector_const_reverse_tlate (const 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

- 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).

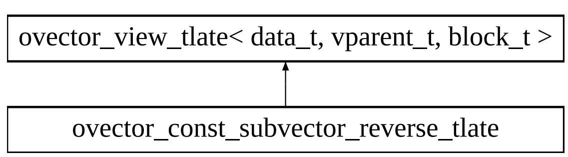
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.262 ovector_const_subvector_reverse_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_const_subvector_reverse_tlate::



7.262.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_const_subvector_reverse_tlate< data_t, vparent_t, block_t >

Reversed view of a const subvector.

Definition at line 1819 of file ovector_tlate.h.

Public Member Functions

- ovector_const_subvector_reverse_tlate (const ovector_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).

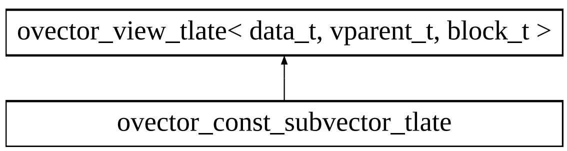
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.263 ovector_const_subvector_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_const_subvector_tlate::



7.263.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_const_subvector_tlate< data_t, vparent_t, block_t >

Create a const vector from a subvector of another vector.

Definition at line 1294 of file ovector_tlate.h.

Public Member Functions

- ovector_const_subvector_tlate (const ovector_view_tlate< data_t, vparent_t, block_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`*.*
- const data_t & operator[ ] (size_t i) const
  Array-like indexing.
- const data_t & operator() (size_t i) const
  Array-like indexing.

Protected Member Functions

Ensure \c const by hiding non-const members

- 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)

Set (with optional range-checking).
- int swap (ovector_view_tlate< data_t, vparent_t, block_t > &x)
  Swap vectors.
- int set_all (double val)
  Set all of the value to be the value `val`.
- vparent_t ∗ get_gsl_vector ()
  Return a gsl vector.
- ovector_view_tlate< data_t, vparent_t, block_t > & operator+= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator+=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator-= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator-=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator∗= (const data_t &y)
  operator∗=

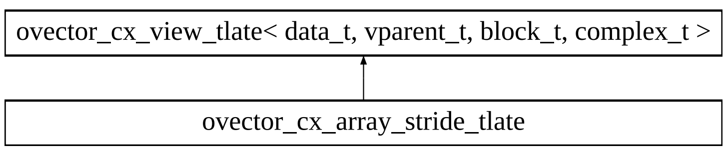
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.264 ovector_cx_array_stride_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_array_stride_tlate::



7.264.1 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 >

Create a vector from an array with a stride.

Definition at line 758 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`.

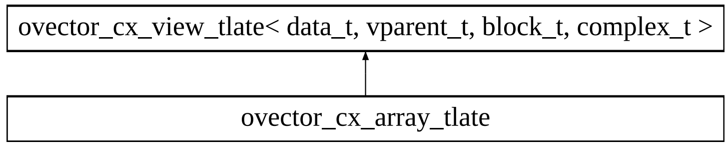
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.265 ovector_cx_array_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_array_tlate::



7.265.1 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 >

Create a vector from an array.

Definition at line 738 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`.

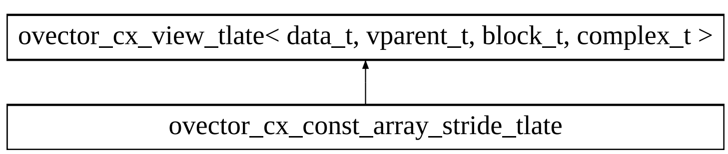
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.266 ovector_cx_const_array_stride_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_const_array_stride_tlate::



7.266.1 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 >

Create a vector from an array_stride.

Definition at line 863 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 **swap** (ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
- 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∗=

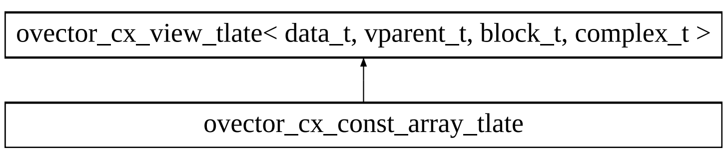
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.267 ovector_cx_const_array_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_const_array_tlate::



7.267.1 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 >

Create a vector from an array.

Definition at line 808 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 **swap** (ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
- 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=*

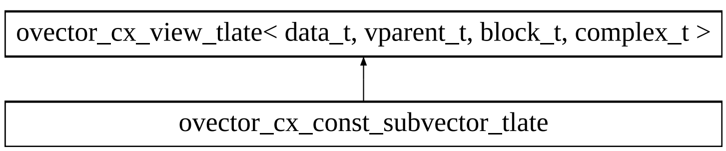
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.268 ovector_cx_const_subvector_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_const_subvector_tlate::



7.268.1 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 >

Create a vector from a subvector of another.

Definition at line 919 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 **swap** (ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > &x)
- 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=*

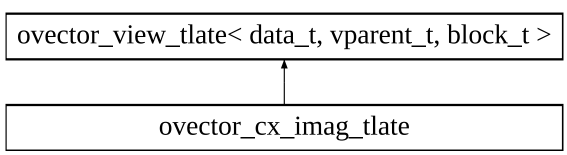
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.269 ovector_cx_imag_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_imag_tlate::



7.269.1 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 >

Create a imaginary vector from the imaginary parts of a complex vector.

Definition at line 1001 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.

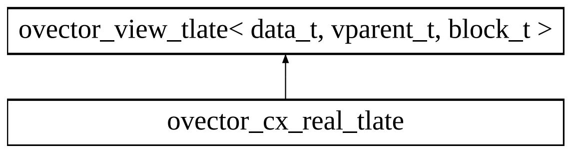
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.270 ovector_cx_real_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_real_tlate::



7.270.1 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 >

Create a real vector from the real parts of a complex vector.

Definition at line 979 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.

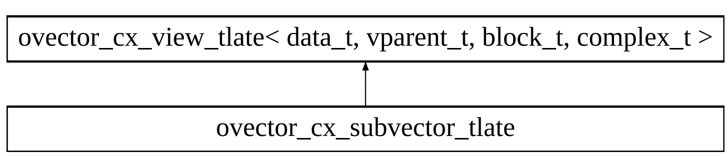
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.271 ovector_cx_subvector_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_subvector_tlate::



7.271.1 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 >

Create a vector from a subvector of another.

Definition at line 779 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`.

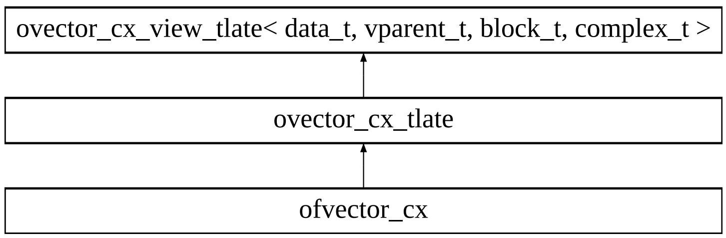
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.272 ovector_cx_tlate Class Template Reference

```
#include <ovector_cx_tlate.h>
```

Inheritance diagram for ovector_cx_tlate::



7.272.1 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 >

A vector of double-precision numbers.

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

There is a slight difference between how this works in comparison to MV++. The function allocate() operates a little differently than newsize(), as it will feel free to allocate new memory when owner is false. It's not clear if this is an issue, however, since it doesn't appear possible to create an ovector_cx_tlate with a value of `owner` equal to zero. This situation ought to be clarified further.

Todo

Add subvector_stride, const_subvector_stride

Definition at line 503 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.

7.272.2 Member Function Documentation

7.272.2.1 int free () `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 708 of file ovector_cx_tlate.h.

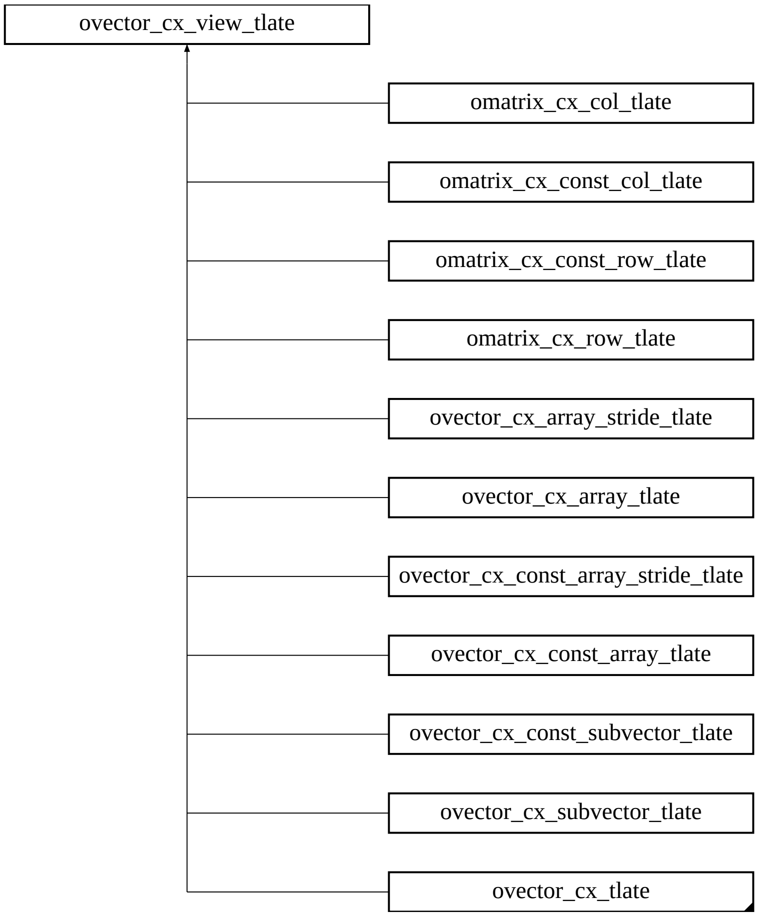
The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.273 ovector_cx_view_tlate Class Template Reference

`#include <ovector_cx_tlate.h>`

Inheritance diagram for ovector_cx_view_tlate::



7.273.1 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 >

A vector view of double-precision numbers.

Todo

Move conversion b/w complex<double> and gsl_complex to cx_arith.h

Definition at line 59 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].

7.273.2 Member Function Documentation

7.273.2.1 size_t size () const `[inline]`

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 316 of file ovector_cx_tlate.h.

7.273.2.2 size_t stride () const `[inline]`

Method to return vector stride.

If no memory has been allocated, this will quietly return zero.

Definition at line 326 of file ovector_cx_tlate.h.

The documentation for this class was generated from the following file:

- ovector_cx_tlate.h

7.274 ovector_int_alloc Class Reference

`#include <ovector_tlate.h>`

7.274.1 Detailed Description

A simple class to provide an `allocate()` function for ovector_int.

Definition at line 2013 of file ovector_tlate.h.

Public Member Functions

- void allocate (ovector_int &o, int i)
  Allocate `v` *for* `i` *elements.*
- void free (ovector_int &o)
  Free memory.

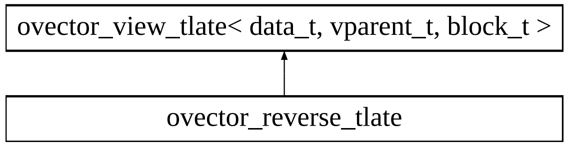
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.275 ovector_reverse_tlate Class Template Reference

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_reverse_tlate::



7.275.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_reverse_tlate< data_t, vparent_t, block_t >

Reversed view of a vector.

Note that you cannot reverse a reversed vector and expect to get the original vector back.

Definition at line 1402 of file ovector_tlate.h.

Public Member Functions

- 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`.

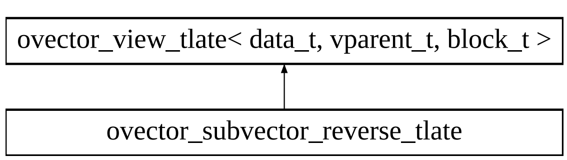
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.276 ovector_subvector_reverse_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_subvector_reverse_tlate::



7.276.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_subvector_reverse_tlate< data_t, vparent_t, block_t >

Reversed view of a subvector.

Note that you cannot reverse a reversed vector and expect to get the original vector back.

Definition at line 1652 of file ovector_tlate.h.

Public Member Functions

- ovector_subvector_reverse_tlate (ovector_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

- 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`.

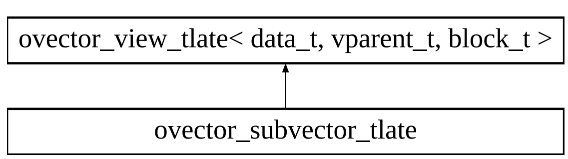
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.277 ovector_subvector_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_subvector_tlate::



7.277.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_subvector_tlate< data_t, vparent_t, block_t >

Create a vector from a subvector of another.

Definition at line 1072 of file ovector_tlate.h.

Public Member Functions

- ovector_subvector_tlate (ovector_view_tlate< data_t, vparent_t, block_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

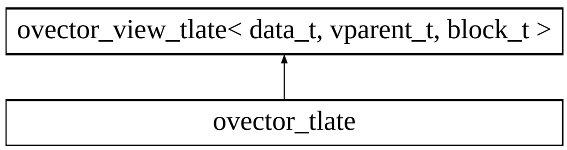
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.278 ovector_tlate Class Template Reference

```
#include <ovector_tlate.h>
```

Inheritance diagram for ovector_tlate::



7.278.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_tlate< data_t, vparent_t, block_t >

A vector with finite stride.

There are several global binary operators associated with objects of type uvector_tlate. The are documented in the "Functions" section of ovector_tlate.h.

Definition at line 545 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, allocate new space and make a copy.
- ovector_tlate (const ovector_view_tlate< data_t, vparent_t, block_t > &v)
  Deep copy constructor, allocate new space and make a copy.
- ovector_tlate (const uvector_view_tlate< data_t > &v)
  Deep copy constructor, allocate new space and make a copy.
- 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_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_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.

Stack-like operations (very experimental)

- 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 ()
  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.

7.278.2 Member Function Documentation

7.278.2.1 int free () `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 848 of file ovector_tlate.h.

7.278.2.2 int reserve (size_t *cap*) `[inline]`

Reserve memory by increasing capacity.

Increase the maximum capacity of the vector so that calls to push_back() do not need to automatically increase the capacity.

This function quietly does nothing if `cap` is smaller than the present vector size given by size().

Definition at line 930 of file ovector_tlate.h.

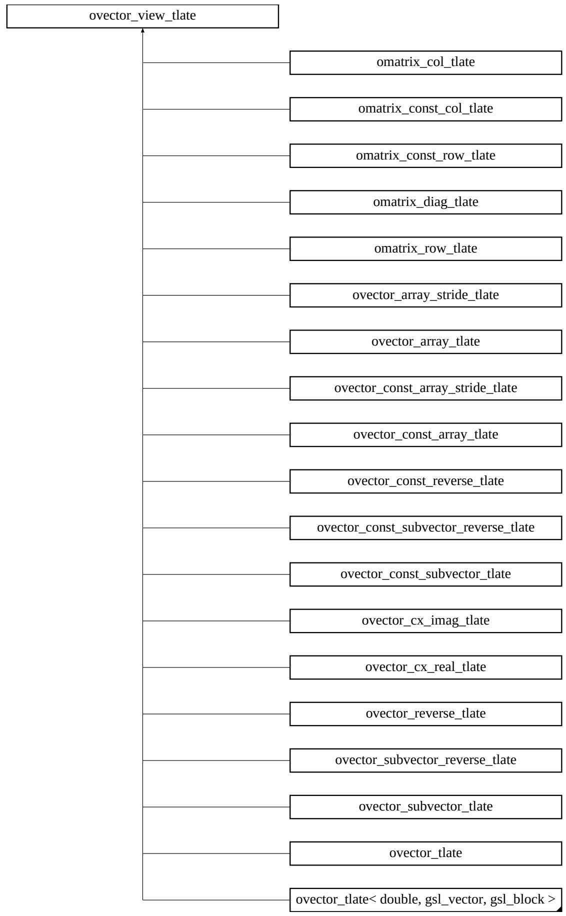
The documentation for this class was generated from the following file:

- ovector_tlate.h

7.279 ovector_view_tlate Class Template Reference

`#include <ovector_tlate.h>`

Inheritance diagram for ovector_view_tlate::



7.279.1 Detailed Description

template<class data_t, class vparent_t, class block_t> class ovector_view_tlate< data_t, vparent_t, block_t >

A vector view with finite stride.

Todo

Check about self assignment as noted in `http://www.cs.caltech.edu/courses/cs11/material/cpp/donnie/cpp-ops.`

Todo

Need to double-check and ensure operator[] and operator() are properly available on all of the various ovector_view children.

Idea for future

Consider an `operator==` ?

Definition at line 59 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 (const uvector_view_tlate< data_t > &v)
  Shallow copy constructor - view a unit-stride vector.
- ovector_view_tlate & operator= (const uvector_view_tlate< data_t > &v)
  Shallow copy constructor - view a unit-stride 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 (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`.
- 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.

Arithmetic

- ovector_view_tlate< data_t, vparent_t, block_t > & operator+= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator+=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator-= (const ovector_view_tlate< data_t, vparent_t, block_t > &x)
  operator-=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator+= (data_t &x)
  operator+=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator-= (data_t &x)
  operator-=
- ovector_view_tlate< data_t, vparent_t, block_t > & operator*= (const data_t &y)
  operator=*
- data_t norm () const
  Norm.

Other methods

- int swap (ovector_view_tlate< data_t, vparent_t, block_t > &x)
  Swap vectors.
- 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.
- 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.
- 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_view_tlate ()
  Empty constructor provided for use by ovector_tlate(const ovector_tlate &v).

7.279.2 Member Function Documentation

7.279.2.1 size_t size () const `[inline]`

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 253 of file ovector_tlate.h.

7.279.2.2 size_t capacity () const `[inline]`

Method to return capacity.

Analogous to `std::vector<>.capacity()`

Definition at line 262 of file ovector_tlate.h.

7.279.2.3 size_t stride () const `[inline]`

Method to return vector stride.

If no memory has been allocated, this will quietly return zero.

Definition at line 273 of file ovector_tlate.h.

7.279.2.4 bool is_owner () const `[inline]`

Return true if this object owns the data it refers to.

This can be used to determine if an object is a "vector_view", or a legitmate "vector". If is_owner() is true, then it is an ovector_tlate object.

If any O2scl class creates a ovector_tlate object in which is_owner() returns false, then it is a bug and should be reported.

Definition at line 370 of file ovector_tlate.h.

7.279.2.5 size_t lookup (const data_t *x0*) const `[inline]`

Exhaustively look through the array for a particular value.

This can only fail if *all* of the entries in the array are not finite, in which case it calls set_err() and returns 0.

Definition at line 381 of file ovector_tlate.h.

The documentation for this class was generated from the following file:

- ovector_tlate.h

7.280 permutation Class Reference

```
#include &lt;permutation.h&gt;
```

7.280.1 Detailed Description

A permutation.

This permutation is completely compatible with the GSL permutation object.

Definition at line 54 of file permutation.h.

Public Member Functions

- **permutation** (size_t dim=0)
- 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)

7.280.2 Member Function Documentation

7.280.2.1 size_t size () const [inline]

Return permutation size.

If no memory has been allocated, this will quietly return zero.

Definition at line 204 of file permutation.h.

7.280.2.2 int free () [inline]

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 226 of file permutation.h.

7.280.2.3 int apply (vec_t & *v*) const [inline]

Apply the permutation to a vector.

Now have k==i, i.e. the least in its cycle

Definition at line 277 of file permutation.h.

7.280.2.4 int apply_inverse (vec_t & *v*) const [inline]

Apply the inverse permutation to a vector.

Now have k==i, i.e. the least in its cycle

Definition at line 302 of file permutation.h.

The documentation for this class was generated from the following file:

- permutation.h

7.281 pinside Class Reference

```
#include <pinside.h>
```

7.281.1 Detailed Description

Test line intersection and point inside polygon.

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 48 of file pinside.h.

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_view &xa, const o2scl::ovector_view &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`.

Data Structures

- struct line
  Internal line definition for pinside.
- struct point
  Internal point definition for pinside.

7.281.2 Member Function Documentation

7.281.2.1 int intersect (double *x1*, double *y1*, double *x2*, double *y2*, double *x3*, double *y3*, double *x4*, double *y4*) `[inline]`

Determine if two line segments intersect.

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 81 of file pinside.h.

7.281.2.2 int inside (double *x*, double *y*, const o2scl::ovector_view & *xa*, const o2scl::ovector_view & *ya*)

Determine if point (x,y) is inside a polygon.

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.

7.281.2.3 int inside (double *x*, double *y*, size_t *n*, const vec_t & *xa*, const vec_t & *ya*) `[inline]`

Determine if point (x,y) is inside a polygon.

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 115 of file pinside.h.

The documentation for this class was generated from the following file:

- pinside.h

7.282 pinside::line Struct Reference

`#include <pinside.h>`

7.282.1 Detailed Description

Internal line definition for pinside.

Definition at line 59 of file pinside.h.

Data Fields

- point **p1**
- point **p2**

The documentation for this struct was generated from the following file:

- pinside.h

7.283 pinside::point Struct Reference

`#include <pinside.h>`

7.283.1 Detailed Description

Internal point definition for pinside.

Definition at line 53 of file pinside.h.

Data Fields

- double **x**
- double **y**

The documentation for this struct was generated from the following file:

- pinside.h

7.284 planar_intp Class Template Reference

`#include <planar_intp.h>`

7.284.1 Detailed Description

template<class vec_t, class mat_t> class planar_intp< vec_t, mat_t >

Interpolate among two independent variables with planes.

This is an analog of 1-d linear interpolation for two dimensions. For a set of data $x_i, y_i, f_{j,i}$, values for 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 is done by finding the plane that goes through three closest points in the data set.

This procedure will fail if the three closest points are co-linear, and interp() will then call set_err() and return zero.

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`[].

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.

Definition at line 60 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, double &x1, double &y1, double &x2, double &y2, double &x3, double &y3)
  Planar interpolation returning the closest points.

Protected Member Functions

- int swap (int &i1, double &c1, int &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.

7.284.2 Member Function Documentation

7.284.2.1 int interp (double *x*, double *y*, vec_t & *ip*) `[inline]`

Perform the planar interpolation.

It is assumed that `ip` is properly allocated beforehand.

Definition at line 94 of file planar_intp.h.

7.284.2.2 int interp (double *x*, double *y*, vec_t & *ip*, double & *x1*, double & *y1*, double & *x2*, double & *y2*, double & *x3*, double & *y3*) `[inline]`

Planar interpolation returning the closest points.

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 108 of file planar_intp.h.

The documentation for this class was generated from the following file:

- planar_intp.h

7.285 pointer_2d_alloc Class Template Reference

`#include <array.h>`

7.285.1 Detailed Description

template<class base_t> class pointer_2d_alloc< base_t >

A simple class to provide an `allocate()` function for pointers.

Uses `new` and `delete`.

Definition at line 134 of file array.h.

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.

The documentation for this class was generated from the following file:

- array.h

7.286 pointer_alloc Class Template Reference

`#include <array.h>`

7.286.1 Detailed Description

template<class base_t> class pointer_alloc< base_t >

A simple class to provide an `allocate()` function for pointers.

Uses `new` and `delete`.

Definition at line 120 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.

The documentation for this class was generated from the following file:

- array.h

7.287 pointer_input Struct Reference

`#include <collection.h>`

7.287.1 Detailed Description

A pointer input structure.

Definition at line 76 of file collection.h.

Data Fields

- std::string name
  The name of the pointer.
- void ∗∗ ptr
  The pointer.
- std::string stype
  The type of the object pointed to.

The documentation for this struct was generated from the following file:

- collection.h

7.288 pointer_output Struct Reference

`#include <collection.h>`

7.288.1 Detailed Description

A pointer output structure.

Definition at line 66 of file collection.h.

Data Fields

- std::string name
  The name of the pointer.
- collection_entry ∗ ep
  Pointer to the collection entry.
- bool output
  True if the pointer has been written to the file.

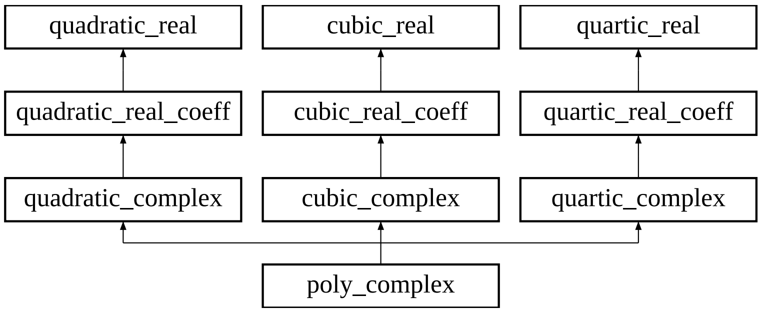
The documentation for this struct was generated from the following file:

- collection.h

7.289 poly_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for poly_complex::



7.289.1 Detailed Description

Solve a general polynomial with complex coefficients [abstract base].

Definition at line 361 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").

7.289.2 Member Function Documentation

7.289.2.1 virtual int solve_c (int *n*, const std::complex< double > *co*[ ], std::complex< double > *ro*[ ]) `[pure virtual]`

Solve the n-th order polynomial.

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].

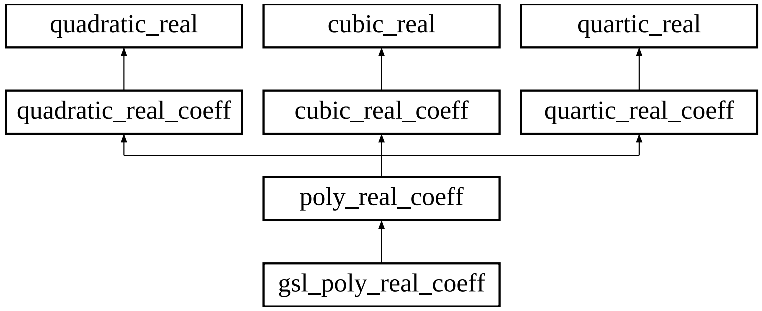
The documentation for this class was generated from the following file:

- poly.h

7.290 poly_real_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for poly_real_coeff::



7.290.1 Detailed Description

Solve a general polynomial with real coefficients and complex roots [abstract base].

Definition at line 337 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").

7.290.2 Member Function Documentation

7.290.2.1 virtual int solve_rc (int *n*, const double *co*[ ], std::complex< double > *ro*[ ]) `[pure virtual]`

Solve the n-th order polynomial.

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

7.291 polylog Class Reference

```
#include <polylog.h>
```

7.291.1 Detailed Description

Polylogarithms (approximate) $Li_n(x)$.

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 $x < -1$

Todo

- 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?

For reference, there are exact relations

$$\mathrm{Li}_2\left(\frac{1}{2}\right) = \frac{1}{12}\left[\pi^2 - 6\left(\ln 2\right)^2\right]$$

$$\mathrm{Li}_3\left(\frac{1}{2}\right) = \frac{1}{24}\left[4\left(\ln 2\right)^3 - 2\pi^2\ln 2 + 21\zeta(3)\right]$$

$$\mathrm{Li}_{-1}(x) = \frac{x}{\left(1-x\right)^2}$$

$$\mathrm{Li}_{-2}(x) = \frac{x\left(x+1\right)}{\left(1-x\right)^3}$$

Definition at line 77 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

7.292 quad_intp Class Template Reference

```
#include <quad_intp.h>
```

7.292.1 Detailed Description

template<class vec_t, class mat_t> class quad_intp< vec_t, mat_t >

Interpolate a function of two independent variables with a quadratic polynomial.

This is a "conic-section" interpolation for two dimensions, using the function

$$z(x,y) = a_1 x^2 + a_2 xy + a_3 y^2 + a_4 x + a_5 y + a_6$$

For a set of data x_i, y_i, z_i, a value of z is predicted given a value of x and y. This is done by finding the conic section obeying the above relation that goes through six closest points in the data set.

This procedure does not always succeed. It fails when the 6 closest points are somehow degenerate, for example, if they are all colinear.

The vector and matrix types can be any types which have suitably defined functions `operator`[].

There is no caching so the numeric values of the data may be freely changed between calls to interp().

Bug

This class doesn't seem to work at present.

Definition at line 64 of file quad_intp.h.

Public Member Functions

- int **compare** (const void ∗x, const void ∗y)
- 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 quad interpolation.
- int interp (double x, double y, vec_t &ip)
  Perform the quadratic interpolation.

Protected Member Functions

- int **swap** (int &i1, double &c1, int &i2, double &c2)

Protected Attributes

- int np
  The number of grid points.
- int 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 given by the user.

7.292.2 Member Function Documentation

7.292.2.1 int interp (double *x*, double *y*, vec_t & *ip*) `[inline]`

Perform the quadratic interpolation.

It is assumed that `ip` is properly allocated beforehand.

Definition at line 105 of file quad_intp.h.

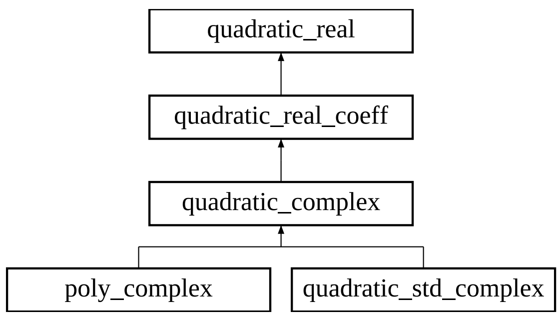
The documentation for this class was generated from the following file:

- quad_intp.h

7.293 quadratic_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for quadratic_complex::



7.293.1 Detailed Description

Solve a quadratic polynomial with complex coefficients and complex roots [abstract base].

Definition at line 105 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 $a_2 x^2 + b_2 x + c_2 = 0$ *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 $a_2 x^2 + b_2 x + c_2 = 0$ *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 $a_2 x^2 + b_2 x + c_2 = 0$ *giving the two complex solutions* $x = x_1$ *and* $x = x_2$*.*
- const char $*$ type ()
  Return a string denoting the type ("quadratic_complex").

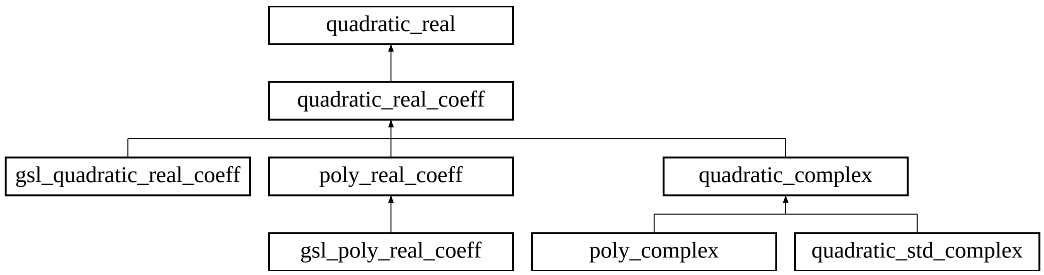
The documentation for this class was generated from the following file:

- poly.h

7.294 quadratic_real Class Reference

`#include <poly.h>`

Inheritance diagram for quadratic_real::



7.294.1 Detailed Description

Solve a quadratic polynomial with real coefficients and real roots [abstract base].

Definition at line 59 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 $a_2 x^2 + b_2 x + c_2 = 0$ *giving the two solutions* $x = x_1$ *and* $x = x_2$.
- const char $*$ type ()
  Return a string denoting the type ("quadratic_real").

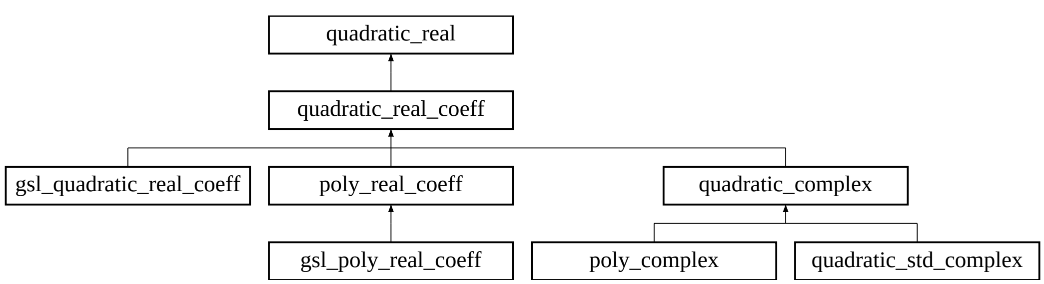
The documentation for this class was generated from the following file:

- poly.h

7.295 quadratic_real_coeff Class Reference

`#include <poly.h>`

Inheritance diagram for quadratic_real_coeff::



7.295.1 Detailed Description

Solve a quadratic polynomial with real coefficients and complex roots [abstract base].

Definition at line 78 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").

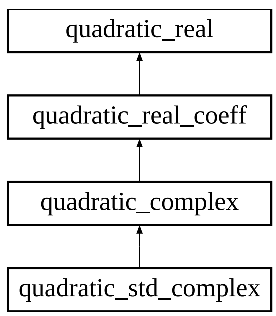
The documentation for this class was generated from the following file:

- poly.h

7.296 quadratic_std_complex Class Reference

`#include <poly.h>`

Inheritance diagram for quadratic_std_complex::



7.296.1 Detailed Description

Solve a quadratic with complex coefficients and complex roots.

Definition at line 584 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").

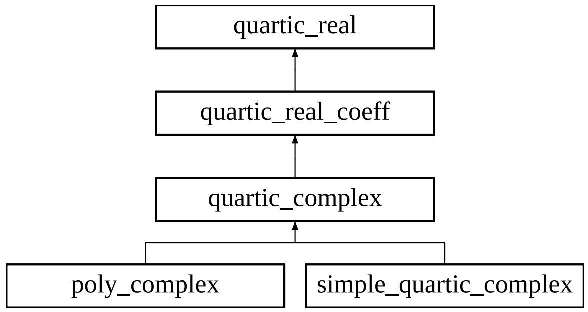
The documentation for this class was generated from the following file:

- poly.h

7.297 quartic_complex Class Reference

`#include <poly.h>`

Inheritance diagram for quartic_complex::



7.297.1 Detailed Description

Solve a quartic polynomial with complex coefficients and complex roots [abstract base].

Definition at line 290 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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").

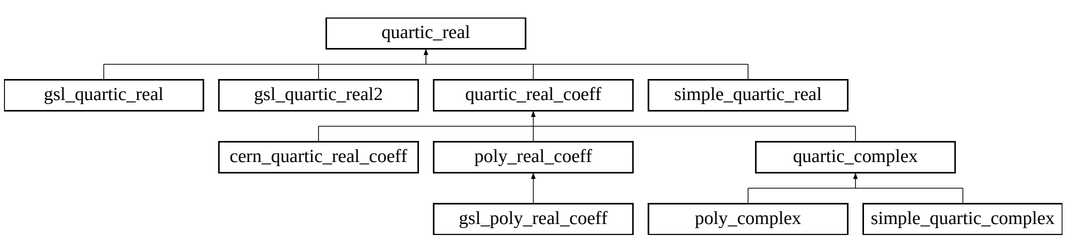
The documentation for this class was generated from the following file:

- poly.h

7.298 quartic_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for quartic_real::



7.298.1 Detailed Description

Solve a quartic polynomial with real coefficients and real roots [abstract base].

Definition at line 232 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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").

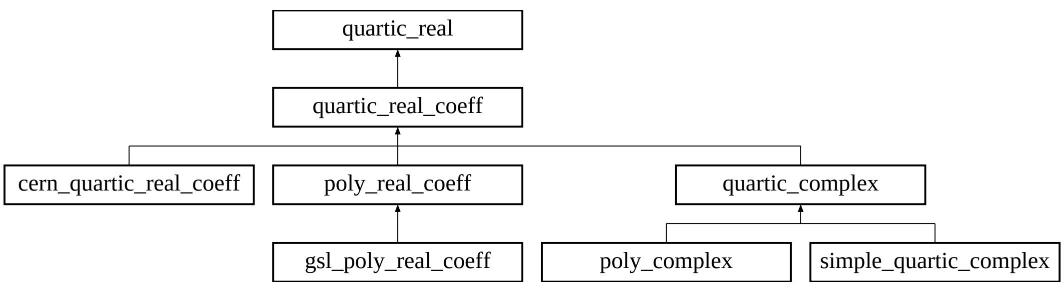
The documentation for this class was generated from the following file:

- poly.h

7.299 quartic_real_coeff Class Reference

```
#include <poly.h>
```

Inheritance diagram for quartic_real_coeff::



7.299.1 Detailed Description

Solve a quartic polynomial with real coefficients and complex roots [abstract base].

Definition at line 256 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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").

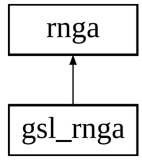
The documentation for this class was generated from the following file:

- poly.h

7.300 rnga Class Reference

```
#include <rnga.h>
```

Inheritance diagram for rnga::



7.300.1 Detailed Description

Random number generator base.

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().

Definition at line 50 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.

7.300.2 Member Function Documentation

7.300.2.1 void clock_seed ()

Initialize the seed with a value taken from the computer clock.

This is a naive seed generator which uses `seed=time(NULL)` to generate a seed.

Todo

Ensure this function is ANSI compatible

Reimplemented in gsl_rnga.

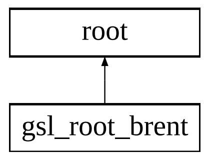
The documentation for this class was generated from the following file:

- rnga.h

7.301 root Class Template Reference

```
#include <root.h>
```

Inheritance diagram for root::



7.301.1 Detailed Description

template<class param_t, class func_t, class dfunc_t = func_t> class root< param_t, func_t, dfunc_t >

1-dimensional solver base class

Note:

This class does not actually do any solving, it is present to provide member data and various functions common to all the 1 dimensional solvers.

Idea for future

This does not have pure virtual functions, but I'd still like to prevent the user from directly instantiating a root object.

Definition at line 48 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, param_t &pa, func_t &func)
  Solve `func` *using* `x` *as an initial guess.*
- virtual int solve_bkt (double &x1, double x2, param_t &pa, func_t &func)
  Solve `func` *in region* $x_1 < x < x_2$ *returning* x_1 .
- virtual int solve_de (double &x, param_t &pa, func_t &func, dfunc_t &df)
  Solve `func` *using* `x` *as an initial guess using derivatives* `df`.

Data Fields

- double tolf
  The maximum value of the functions for success (default 10^{-8}).
- double tolx
  The minimum allowable stepsize (default 10^{-12}).
- int verbose
  Output control (default 0).
- int ntrial
  Maximum number of iterations (default 100).
- bool over_bkt
  Should be true if root_bkt() is overloaded.
- bool over_de
  Should be true if root_de() is overloaded.
- double bracket_step
  The stepsize for automatic bracketing (default 10^{-4}).
- int last_ntrial
  The number of iterations for in the most recent minimization.

7.301.2 Member Function Documentation

7.301.2.1 virtual int print_iter (double *x*, double *y*, int *iter*, double *value* = `0.0`, double *limit* = `0.0`, std::string *comment* = `""`) `[inline, virtual]`

Print out iteration information.

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 112 of file root.h.

7.301.3 Field Documentation

7.301.3.1 double bracket_step

The stepsize for automatic bracketing (default 10^{-4}).

If this is exactly zero, it will be reset to 10^{-4} by solve().

Definition at line 92 of file root.h.

The documentation for this class was generated from the following file:

- root.h

7.302 search_vec Class Template Reference

`#include <search_vec.h>`

7.302.1 Detailed Description

template<class vec_t> class search_vec< vec_t >

Searching class for monotonic data.

A searching class for monotonic vectors. A caching system similar to `gsl_interp_accel` is used.

For normal usage, just 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.

Todo

The documentation here is still kind of unclear.

Definition at line 52 of file search_vec.h.

Public Member Functions

- size_t find (const double x0, size_t n, const vec_t &x)
  Search an increasing or decreasing vector.
- size_t find_inc (const double x0, size_t n, const vec_t &x)
  Search part of a increasing vector.
- size_t find_dec (const double x0, size_t n, const vec_t &x)
  Search part of a decreasing vector.
- size_t ordered_lookup (const double x0, size_t n, const vec_t &x)
  Find the index of x0 in the ordered array `x`.
- size_t ordered_interval (const double x0, size_t n, const vec_t &x)
  Find the interval containing x0 in the ordered array `x`.
- size_t bsearch_inc (const double x0, const vec_t &x, size_t lo, size_t hi) const
  Binary search a part of an increasing vector.
- size_t bsearch_dec (const double x0, const vec_t &x, size_t lo, size_t hi) const
  Binary search a part of an decreasing vector.

Protected Attributes

- size_t cache
  Storage for the most recent index.

7.302.2 Member Function Documentation

7.302.2.1 size_t ordered_lookup (const double *x0*, size_t *n*, const vec_t & *x*) `[inline]`

Find the index of x0 in the ordered array `x`.

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 some of the values in the ovector are not finite, then the output of this function is not defined.

If x[i] is non-monotonic, consider using ovector_view_tlate::lookup() or uvector_view_tlate::lookup() instead of this function.

Definition at line 120 of file search_vec.h.

7.302.2.2 size_t ordered_interval (const double *x0*, size_t *n*, const vec_t & *x*) `[inline]`

Find the interval containing x0 in the ordered array `x`.

This returns the index i for which x[i]<=x0<x[i+1].

If the array is increasing and x0<x[0], then 0 is returned. If the array is increasing and x0>x[n-1], then nvar-1 is returned (this behavior is slightly different from GSL). The decreasing case is handled analogously.

If some of the values in the vector are not finite, then the output of this function is not defined.

If x[i] is non-monotonic, consider using ovector_view_tlate::lookup() or uvector_view_tlate::lookup() instead of this function.

Definition at line 167 of file search_vec.h.

7.302.2.3 size_t bsearch_inc (const double *x0*, const vec_t & *x*, size_t *lo*, size_t *hi*) const [inline]

Binary search a part of an increasing vector.

This function performs a binary search of between `x[lo]` and `x[hi-1]`. It returns

- `lo` if `x0` < `x[lo+1]`
- `i` if `x[i]` <= `x0` < `x[i+1]` for `lo` <= `i` < `hi`
- `hi-1` if `x0` >= `x[hi-1]`

The cache is not used for this function.

Definition at line 200 of file search_vec.h.

7.302.2.4 size_t bsearch_dec (const double *x0*, const vec_t & *x*, size_t *lo*, size_t *hi*) const [inline]

Binary search a part of an decreasing vector.

This function performs a binary search of between `x[lo]` and `x[hi-1]`. It returns

- `lo` if `x0` > `x[lo+1]`
- `i` if `x[i]` >= `x0` > `x[i+1]` for `lo` <= `i` < `hi`
- `hi-1` if `x0` <= `x[hi-1]`

The cache is not used for this function.

Definition at line 226 of file search_vec.h.

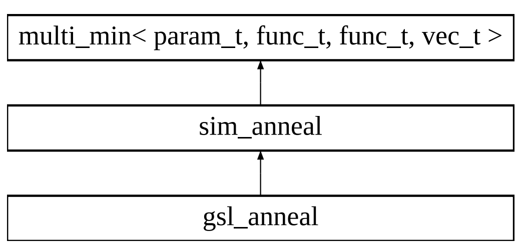
The documentation for this class was generated from the following file:

- search_vec.h

7.303 sim_anneal Class Template Reference

`#include <sim_anneal.h>`

Inheritance diagram for sim_anneal::



7.303.1 Detailed Description

template<class param_t, class func_t, class vec_t = ovector_view, class rng_t = gsl_rnga> class sim_anneal< param_t, func_t, vec_t, rng_t >

Simulated annealing base.

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 minimize::ntrial which defaults to 100.

Definition at line 57 of file sim_anneal.h.

Public Member Functions

- virtual int mmin (size_t nvar, vec_t &x, double &fmin, param_t &pa, func_t &func)=0
  Calculate the minimum `fmin` *of* `func` *w.r.t the array* `x` *of size* `nvar`.
- int set_tptr_schedule (tptr_schedule< vec_t > &tr)
  Specify the temperature schedule.
- 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.
- tptr_geoseries< vec_t > def_schedule
  The default temperature schedule.

Protected Attributes

- tptr_schedule< vec_t > * tp
  Pointer to the temperature annealing schedule.

7.303.2 Member Function Documentation

7.303.2.1 virtual int print_iter (size_t *nv*, vec_t & *x*, double *y*, int *iter*, double *tptr*, std::string *comment*) `[inline, virtual]`

Print out iteration information.

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 92 of file sim_anneal.h.

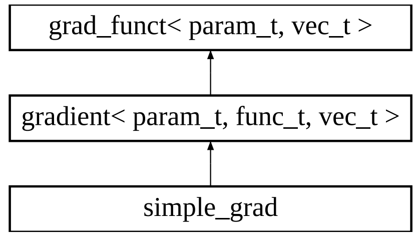
The documentation for this class was generated from the following file:

- sim_anneal.h

7.304 simple_grad Class Template Reference

`#include <multi_min.h>`

Inheritance diagram for simple_grad::



7.304.1 Detailed Description

template<class param_t, class func_t, class vec_t> class simple_grad< param_t, func_t, vec_t >

Simple automatic computation of gradient by finite differencing.

Definition at line 164 of file multi_min.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &g, param_t &pa)
  Compute the gradient `g` *at the point* `x`.

Data Fields

- double epsrel
  The relative stepsize for finite-differencing (default 10^{-4} *).*
- double epsmin
  The minimum stepsize (default 10^{-15}*).*

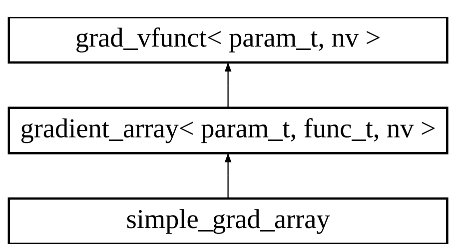
The documentation for this class was generated from the following file:

- multi_min.h

7.305 simple_grad_array Class Template Reference

```
#include <multi_min.h>
```

Inheritance diagram for simple_grad_array::



7.305.1 Detailed Description

template<class param_t, class func_t, size_t nv> class simple_grad_array< param_t, func_t, nv >

Simple automatic computation of gradient by finite differencing with arrays.

Definition at line 347 of file multi_min.h.

Public Member Functions

- virtual int operator() (size_t nvar, double x[nv], double g[nv], param_t &pa)
  Compute the gradient `g` *at the point* `x`.

Data Fields

- double epsrel
  The relative stepsize for finite-differencing (default 10^{-4} *).*
- double epsmin
  The minimum stepsize (default 10^{-15}*).*

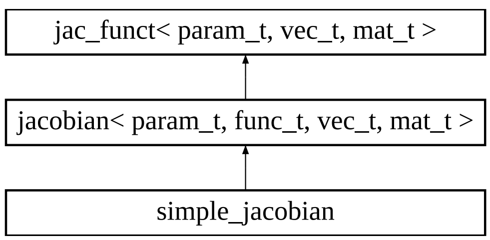
The documentation for this class was generated from the following file:

- multi_min.h

7.306 simple_jacobian Class Template Reference

```
#include <jacobian.h>
```

Inheritance diagram for simple_jacobian::



7.306.1 Detailed Description

template<class param_t, class func_t = mm_funct<void *>, class vec_t = ovector_view, class mat_t = omatrix_view, class alloc_vec_t = ovector, class alloc_t = ovector_alloc> class simple_jacobian< param_t, func_t, vec_t, mat_t, alloc_vec_t, alloc_t >

Simple automatic Jacobian.

This class computes a numerical Jacobian by finite differencing. The stepsize is chosen to be $h_j = \mathrm{epsrel} x_j$ or $h_j = \mathrm{epsmin}$ if $\mathrm{epsrel} x_j < \mathrm{epsmin}$.

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.

Todo

Double check that this class works with arrays

Idea for future

GSL-1.10 updated fdjac.c and this update could be implemented below.

Definition at line 373 of file jacobian.h.

Public Member Functions

- virtual int operator() (size_t nv, vec_t &x, vec_t &y, mat_t &jac, param_t &pa)
  The operator().

Data Fields

- double epsrel
  The relative stepsize for finite-differencing (default 10^{-4} *).*
- double epsmin
  The minimum stepsize (default 10^{-15}*).*
- alloc_t ao
  For memory allocation.

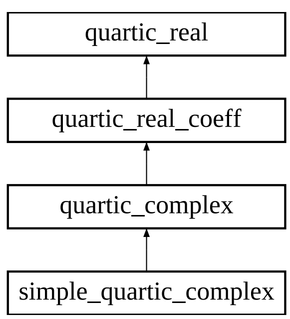
The documentation for this class was generated from the following file:

- jacobian.h

7.307 simple_quartic_complex Class Reference

```
#include <poly.h>
```

Inheritance diagram for simple_quartic_complex::



7.307.1 Detailed Description

Solve a quartic with complex coefficients and complex roots.

Definition at line 647 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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").

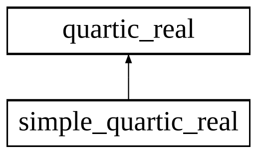
The documentation for this class was generated from the following file:

- poly.h

7.308 simple_quartic_real Class Reference

```
#include <poly.h>
```

Inheritance diagram for simple_quartic_real::



7.308.1 Detailed Description

Solve a quartic with real coefficients and real roots.

Todo

3/8/07 - Compilation at the NSCL produced non-finite values in solve_r() for some values of the coefficients. This should be checked.

Todo

It looks like this code is tested only for a4=1, and if so, the tests should be generalized.

Todo

Also, there is a hard-coded number in here (10^{-6}), which might be moved to a data member?

Definition at line 630 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 $a_4 x^4 + b_4 x^3 + c_4 x^2 + d_4 x + e_4 = 0$ *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").

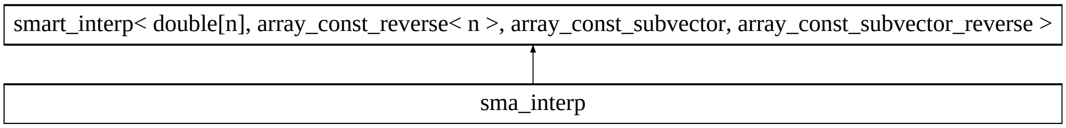
The documentation for this class was generated from the following file:

- poly.h

7.309 sma_interp Class Template Reference

```
#include <smart_interp.h>
```

Inheritance diagram for sma_interp::



7.309.1 Detailed Description

template<size_t n> class sma_interp< n >

A specialization of smart_interp for C-style double arrays.

Definition at line 952 of file smart_interp.h.

Public Member Functions

- sma_interp (base_interp< double[n]> &it1, base_interp< array_const_reverse< n > > &it2, base_interp< array_const_subvector > &it3, base_interp< array_const_subvector_reverse > &it4)
    Create with base interpolation objects.
- sma_interp ()
    Create with default interpolation objects.

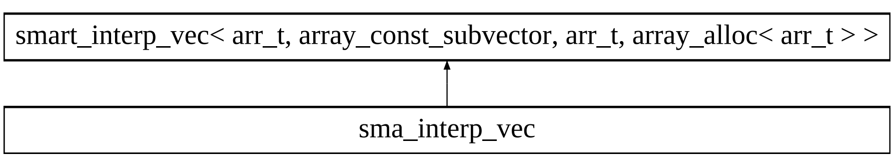
The documentation for this class was generated from the following file:

- smart_interp.h

7.310 sma_interp_vec Class Template Reference

```
#include <smart_interp.h>
```

Inheritance diagram for sma_interp_vec::



7.310.1 Detailed Description

template<class arr_t> class sma_interp_vec< arr_t >

A specialization of smart_interp_vec for C-style arrays.

Definition at line 977 of file smart_interp.h.

Public Member Functions

- sma_interp_vec (base_interp< arr_t > &it, base_interp< array_const_subvector > &it2, size_t n, const arr_t &x, const arr_t &y)
  Create with base interpolation object `it` *and* `it2`.
- sma_interp_vec (size_t n, const arr_t &x, const arr_t &y)
  Create with default interpolation objects.

The documentation for this class was generated from the following file:

- smart_interp.h

7.311 smart_interp Class Template Reference

```
#include <smart_interp.h>
```

7.311.1 Detailed Description

template<class vec_t = ovector_view, class rvec_t = ovector_const_reverse, class svec_t = ovector_const_subvector, class srvec_t = ovector_const_subvector_reverse> class smart_interp< vec_t, rvec_t, svec_t, srvec_t >

Smart interpolation class.

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 46 of file smart_interp.h.

Public Member Functions

- **smart_interp** (base_interp< vec_t > &it1, base_interp< rvec_t > &it2, base_interp< svec_t > &it3, base_interp< srvec_t > &it4)
- 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 $y^{prime\prime}(x=x_0)$ *.*
- 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 objects

- cspline_interp< vec_t > **cit1**
- cspline_interp< rvec_t > **cit2**
- cspline_interp< svec_t > **cit3**
- cspline_interp< srvec_t > **cit4**

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, bool &increasing)
  Try to find the largest monotonic and finite region around the desired location.

Protected Attributes

Storage internally created subvectors

- const svec_t ∗ **sx**
- const svec_t ∗ **sy**
- const srvec_t ∗ **srx**
- const srvec_t ∗ **sry**

Pointers to interpolation objects

- base_interp< vec_t > ∗ **rit1**
- base_interp< rvec_t > ∗ **rit2**
- base_interp< svec_t > ∗ **rit3**
- base_interp< srvec_t > ∗ **rit4**

7.311.2 Member Function Documentation

7.311.2.1 int find_subset (const double *a*, const double *b*, size_t *sz*, const vec_t & *x*, const vec_t & *y*, size_t & *nsz*, bool & *increasing*) `[inline, protected]`

Try to find the largest monotonic and finite region around the desired location.

Todo

After row and row2 are set, check to make sure the entire inside region is monotonic before expanding

Definition at line 563 of file smart_interp.h.

The documentation for this class was generated from the following file:

- smart_interp.h

7.312 smart_interp_vec Class Template Reference

```
#include <smart_interp.h>
```

7.312.1 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 >

Smart interpolation class with pre-specified vectors.

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 654 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< vec_t > &it1, base_interp< svec_t > &it2, size_t n, const vec_t &x, const vec_t &y)
  Create with base interpolation objects `it` *and* `rit`.
- 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

- cspline_interp< vec_t > cit1
  Default base interpolation object.
- cspline_interp< svec_t > cit2
  Default base interpolation object.

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 monotonic and finite region around the desired location.

Protected Attributes

- 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.
- 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.
- alloc_vec_t lrx
  Reversed version of vector.
- alloc_vec_t lry
  Reversed version of vector.
- size_t ln
  Size of user-specifed vector.

The documentation for this class was generated from the following file:

- smart_interp.h

7.313 string_comp Struct Reference

```
#include <misc.h>
```

7.313.1 Detailed Description

Naive string comparison.

This is used internally for the STL routines which require a way to compare strings in the class table and in the I/O classes.

Definition at line 171 of file misc.h.

Public Member Functions

- bool operator() (const std::string s1, const std::string s2) const
  Return `s1<s2`.

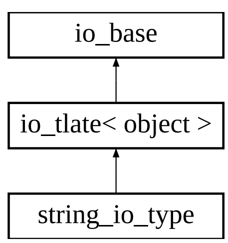
The documentation for this struct was generated from the following file:

- misc.h

7.314 string_io_type Class Reference

```
#include <collection.h>
```

Inheritance diagram for string_io_type::



7.314.1 Detailed Description

I/O object for string variables.

Definition at line 1804 of file collection.h.

Public Member Functions

- string_io_type (const char ∗t)
  Desc.
- int adds (collection &co, std::string name, std::string s, bool overwrt=true)
  Add a string to a collection.
- std::string gets (collection &co, std::string tname)
  Get a string from a collection.
- int get_def (collection &co, std::string tname, std::string &op, std::string def="")
  Get a string from a collection.

The documentation for this class was generated from the following file:

- collection.h

7.315 table Class Reference

```
#include <table.h>
```

7.315.1 Detailed Description

Data table.

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.

Data representation

Each individual column is just an ovector_view (or any descendant of an ovector_view) 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(size_t c, size_t r) 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 (string table::get_column_name(int index), and double table::get_column(int index)) takes only constant time, and finding a column by its name (int lookup_column() and double ∗ table::get_column()) 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 columns must be shifted accordingly.

Ownership of any column may be changed at any time, but care must be taken to ensure that memory allocation errors do not occur. These errors should not occur when no columns are owned by the user.

Because of the structure, this class is not suitable for the matrix manipulation. The classes omatrix and umatrix are better used for that purpose.

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 O(R∗C). Columns which are not owned by the table are not modified, so the table will not allow an increase in the number of lines beyond the size of the smallest user-owned column. 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_-vecp. 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, or the entire table can be sorted by one column.

Allowable column names

In general, column names may be of any form as long as they don't contain whitespace, e.g. `123"`.#$xy∼ is a legitmate column name. The column name should be restricted to contain only letters, numbers, and underscores and may not begin with a digit.

Thread-safety

Generally, the member functions are thread-safe in the sense that one would expect. Simple get() and set() functions are thread-safe, while insertion and deletion operations are not. It makes little sense to try to make insertion and deletion thread-safe. The interpolation routines are not thread-safe.

I/O and command-line manipulation

When data from an object of type table is output to a file through the collection class, 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

Better testing of automatic resizing with user- and class-owned columns

Idea for future

Be more restrictive about allowable column names

Idea for future

Add interp() and related functions which avoid caching and can thus be const (This has been started with interp_const())

Idea for future

The nlines vs. maxlines and automatic resizing of table-owned vs. user-owned vectors could be reconsidered, especially now that ovectors can automatically resize on their own. 10/16/07: This issue may be unimportant, as it might be better to just move to a template based approach with a user-specified vector type. The interpolation is now flexible enough to handle different types. Might check to ensure sorting works with other types.

Idea for future

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 197 of file table.h.

Public Member Functions

- table (int cmaxlines=0)
  Create a new table with space for nlines<=cmaxlines.
- int set_interp (base_interp< ovector_view > &bi1, base_interp< ovector_const_subvector > &bi2)
  Set the base interpolation objects.
- int read_generic (std::istream &fin)
  Read a generic data file.

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` *- O(1).*
- 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)
  Get value from row `row` *of column number* `icol` *- O(1).*
- int 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.
- int get_maxlines ()
  Return the maximum number of lines.
- ovector_view * get_column (std::string col)
  Returns a pointer to the column named `col` *- O(log(C)).*
- const ovector_view * get_column_const (std::string col) const
  Returns a pointer to the column named `col` *- O(log(C)).*
- ovector_view * get_column (size_t icol)
  Returns a pointer to the column of index `icol` *- O(1).*
- const ovector_view * get_column (size_t icol) const
  Returns a pointer to the column of index `icol` *- O(1).*
- const ovector_view & operator[ ] (size_t icol) const
  Returns the column of index `icol` *- O(1) (const version).*
- ovector_view & operator[ ] (size_t icol)
  Returns the column of index `icol` *- O(1).*
- const ovector_view & operator[ ] (std::string scol) const
  Returns the column named `scol` *- O(log(C)) (const version).*
- ovector_view & 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).*

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)).
- int new_column (std::string name, ovector_view *ldat)
  Add a new column owned by the user - O(log(C)).
- int lookup_column (std::string name, int &ix)

Find the index for column named `name` *- O(log(C)).*
- int rename_column (std::string olds, std::string news)
  Rename column named `olds` *to* `news` *- O(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).*
- int ch_owner (std::string name, bool ow)
  Modify ownership - O(log(C)).
- bool get_owner (std::string name) const
  Get ownership - O(log(C)).
- const gsl_vector * get_gsl_vector (std::string name) const
  Get a gsl_vector from column `name` *- O(log(C)).*
- int check_synchro () const
  Return 0 if the tree and list are properly synchronized.
- 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).*
- int lookup_form (std::string formula, double &maxval)
  Search for row with maximum value of formula.

Interpolation, differentiation, and integration, max, and min

- 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`*.*
- 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 (size_t ix, double x0, size_t iy)
  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 (size_t ix, double x0, size_t iy)
  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 (size_t ix, double x1, double x2, size_t iy)
  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, bool linked=true)
  Make a subtable.

Add space

- int inc_maxlines (size_t llines)
  Manually increase the maximum number of lines.

Delete methods

- 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 value `val` *in column* `scol`*.*
- int delete_row (size_t irow)
  Delete the row of index `irow`*.*

Clear methods

- void zero_table ()
  Zero the data entries but keep the column names and nlines fixed.
- void clear_table ()
  Clear the table and the column names.
- void clear_data ()
  Remove all of the data by setting the number of lines to zero.

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

- 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.
- virtual int set_constant (std::string name, double val)
  Add a constant.
- virtual double get_constant (std::string name)
  Get a constant.
- virtual int remove_constant (std::string name)
  Remove a constant.

Protected Types

- typedef struct table::col_s **col**
- typedef struct table::sortd_s **sortd**

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.

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< ovector_view > * intp1
  A pointer to the interpolation object.
- base_interp< ovector_const_subvector > * intp2
  A pointer to the subvector interpolation object.
- cspline_interp< ovector_view > cintp1
  The default interpolation object.
- cspline_interp< ovector_const_subvector > cintp2
  The default subvector interpolation object.
- search_vec< ovector > se
  The vector-searching object.
- bool intp_set
  True if the interpolation type has been set.
- std::string intp_colx
  The last x-column interpolated.
- std::string intp_coly
  The last y-column interpolated.

Data Structures

- struct col_s
  Column structure for table [protected].
- struct sortd_s
  A structure for sorting in table [protected].

7.315.2 Member Function Documentation

7.315.2.1 int set (std::string *col*, size_t *row*, double *val*)

Set row `row` of column named `col` to value `val` - O(log(C)).

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.

7.315.2.2 int set_nlines (size_t *il*)

Set the number of lines.

This function is stingy about increasing the table memory space and will only increase it enough to fit `il` lines, which is useful if you have columns not owned by the table.

7.315.2.3 int set_nlines_auto (size_t *il*)

Set the number of lines.

Todo

Resolve whether set() should really use this approach. Also, resolve whether this should replace set_nlines() (It could be that the answer is no, because as the documentation in the other version states, the other version is useful if you have columns not owned by the table.)

7.315.2.4 ovector_view* get_column (size_t *icol*) [inline]

Returns a pointer to the column of index `icol` - O(1).

Note that several of the methods require reallocation of memory and pointers previously returned by this function will be incorrect.

Definition at line 278 of file table.h.

7.315.2.5 const ovector_view* get_column (size_t *icol*) const [inline]

Returns a pointer to the column of index `icol` - O(1).

Note that several of the methods require reallocation of memory and pointers previously returned by this function will be incorrect.

Definition at line 289 of file table.h.

7.315.2.6 const ovector_view& operator[ ] (size_t *icol*) const [inline]

Returns the column of index `icol` - O(1) (const version).

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.

Definition at line 303 of file table.h.

7.315.2.7 ovector_view& operator[ ] (size_t *icol*) [inline]

Returns the column of index `icol` - O(1).

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.

Definition at line 317 of file table.h.

7.315.2.8 const ovector_view& operator[ ] (std::string *scol*) const [inline]

Returns the column named `scol` - O(log(C)) (const version).

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.

Definition at line 330 of file table.h.

7.315.2.9 ovector_view& operator[ ] (std::string *scol*) [inline]

Returns the column named `scol` - O(log(C)).

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.

Definition at line 344 of file table.h.

7.315.2.10 int new_column (std::string *name*, ovector_view * *ldat*)

Add a new column owned by the user - O(log(C)).

This function does not modify the number of lines of data in the table.

Todo

We've got to figure out what to do if ldat is too small. If it's smaller than nlines, obviously we should just fail, but what if it's size is between nlines and maxlines?

7.315.2.11 int lookup_column (std::string *name*, int & *ix*)

Find the index for column named `name` - O(log(C)).

If the column is not present, this does not call the error handler, but quietly sets `ix` to zero and returns gsl_notfound.

7.315.2.12 int rename_column (std::string *olds*, std::string *news*)

Rename column named `olds` to `news` - O(C).

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.

7.315.2.13 int init_column (std::string *scol*, double *val*)

Initialize all values of column named `scol` to `val` - O(log(C)∗R).

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.

7.315.2.14 int ch_owner (std::string *name*, bool *ow*)

Modify ownership - O(log(C)).

Warning:

columns allocated using malloc() should never be owned by the table object since it uses delete instead of free().

7.315.2.15 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.

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.

7.315.2.16 size_t ordered_lookup (std::string *col*, double *val*)

Look for a value in an ordered column.

O(log(C)∗log(R))

7.315.2.17 int lookup_form (std::string *formula*, double & *maxval*)

Search for row with maximum value of formula.

This searches the table for the maximum value of the specified formula. For example, to find the row for which the column `mu` is 2 and `T` is 3, you can use

```
table::lookup_form("-abs(mu-2)-abs(T-3)");
```

7.315.2.18 double interp (std::string *sx*, double *x0*, std::string *sy*)

Interpolate `x0` from `sx` into `sy`.

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

7.315.2.19 double interp_const (std::string *sx*, double *x0*, std::string *sy*) const

Interpolate `x0` from `sx` into `sy`.

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

7.315.2.20 double interp (size_t *ix*, double *x0*, size_t *iy*)

Interpolate `x0` from `ix` into `iy`.

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

7.315.2.21 double deriv (std::string *sx*, double *x0*, std::string *sy*)

The first derivative of the function sy(sx) at sx=x0.

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

7.315.2.22 double deriv (size_t *ix*, double *x0*, size_t *iy*)

The first derivative of the function iy(ix) at ix=x0.

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

7.315.2.23 double deriv2 (std::string *sx*, double *x0*, std::string *sy*)

The second derivative of the function sy(sx) at sx=x0.

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

7.315.2.24 double deriv2 (size_t *ix*, double *x0*, size_t *iy*)

The second derivative of the function iy(ix) at ix=x0.

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

7.315.2.25 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.

O(log(C)∗log(R)) but can be as bad as O(log(C)∗R) if the relevant columns are not well ordered.

7.315.2.26 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.

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

7.315.2.27 int integ (std::string *x*, std::string *y*, std::string *ynew*)

The integral of the function iy(ix).

O(log(R)) but can be as bad as O(R) if the relevant columns are not well ordered.

7.315.2.28 table* subtable (std::string *list*, size_t *top*, size_t *bottom*, bool *linked* = `true`)

Make a subtable.

Uses the columns specified in `list` from the row `top` to the row of index `bottom`. If `linked` is false the the data will be independent from the original table.

7.315.2.29 int delete_column (std::string *scol*)

Delete column named `scol` - O(C).

This is slow because the iterators in alist are mangled and we have to call reset_list to get them back.

7.315.2.30 void clear_data () `[inline]`

Remove all of the data by setting the number of lines to zero.

This leaves the column names intact and does not remove the constants.

Definition at line 724 of file table.h.

7.315.2.31 int summary (std::ostream * *out*, int *ncol* = `79`) const

Output a summary of the information stored.

Outputs the number of constants, the number of columns, a list of the column names, and the number of lines of data.

7.315.2.32 int reset_list () `[protected]`

Set the elements of alist with the appropriate iterators from atree - O(C).

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

7.316 table::col_s Struct Reference

```
#include <table.h>
```

7.316.1 Detailed Description

Column structure for table [protected].

Definition at line 833 of file table.h.

Data Fields

- ovector_view * 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

7.317 table::sortd_s Struct Reference

`#include <table.h>`

7.317.1 Detailed Description

A structure for sorting in table [protected].

Definition at line 877 of file table.h.

Data Fields

- double val
  Value to sort.
- int indx
  Sorted index.

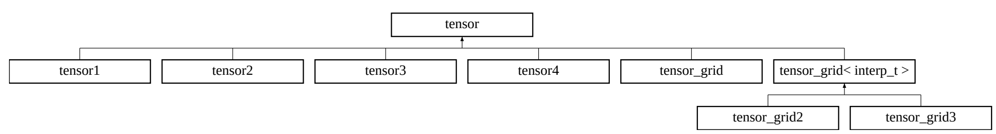
The documentation for this struct was generated from the following file:

- table.h

7.318 tensor Class Reference

`#include <tensor.h>`

Inheritance diagram for tensor::



7.318.1 Detailed Description

Tensor class with arbitrary dimensions.

Todo

More complete testing.

Todo

Add const get functions for const references

Idea for future

Could implement arithmetic operators + and - and some different products.

Idea for future

Add slicing to get ovector or omatrix objects

Definition at line 58 of file tensor.h.

Public Member Functions

- tensor ()
  Create an empty tensor with zero rank.
- tensor (size_t rank, size_t *dim)
  Create a tensor of rank `rank` *with sizes given in* `dim`*.*
- virtual int set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`*.*
- virtual double get (size_t *index)
  Get the element indexed by `index`*.*
- ovector_view vector_slice (size_t ix, size_t *index)
  Fix all but one index to create a vector.
- omatrix_view matrix_slice (size_t *index, size_t ix)
  Fix all but two indices to create a matrix.
- virtual int get_rank ()
  Return the rank of the tensor.
- virtual int tensor_allocate (size_t rank, size_t *dim)
  Allocate space for a tensor of rank `rank` *with sizes given in* `dim`*.*
- virtual int tensor_free ()
  Free allocated space (also sets rank to zero).
- virtual size_t get_size (size_t i)
  Returns the size of the ith index.
- virtual size_t total_size ()
  Returns the size of the tensor.
- size_t pack_indices (size_t *index)
  Pack the indices into a single array index.
- int unpack_indices (size_t ix, size_t *index)
  Unpack the single array index into indices.

Protected Attributes

- double * **data**
- size_t * size
  A rank-sized array of the sizes of each dimension.
- size_t rk
  Rank.

7.318.2 Constructor & Destructor Documentation

7.318.2.1 tensor (size_t *rank*, size_t * *dim*) [inline]

Create a tensor of rank `rank` with sizes given in `dim`.

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 92 of file tensor.h.

7.318.3 Member Function Documentation

7.318.3.1 ovector_view vector_slice (size_t *ix*, size_t * *index*) [inline]

Fix all but one index to create a vector.

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.tensor_allocate(3,dim);
```

the following code

```
size_t index[3]={1,0,3};
ovector_view v=t.vector_slice(index,1);
```

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 198 of file tensor.h.

7.318.3.2 omatrix_view matrix_slice (size_t * *index*, size_t *ix*) `[inline]`

Fix all but two indices to create a matrix.

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.

Definition at line 218 of file tensor.h.

7.318.3.3 virtual int tensor_allocate (size_t *rank*, size_t * *dim*) `[inline, virtual]`

Allocate space for a tensor of rank `rank` with sizes given in `dim`.

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 in tensor_grid, and tensor_grid< interp_t >.

Definition at line 254 of file tensor.h.

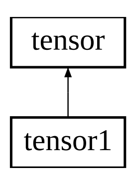
The documentation for this class was generated from the following file:

- tensor.h

7.319 tensor1 Class Reference

`#include <tensor.h>`

Inheritance diagram for tensor1::



7.319.1 Detailed Description

Rank 1 tensor.

Definition at line 746 of file tensor.h.

Public Member Functions

- tensor1 ()
  Create an empty tensor.
- tensor1 (size_t sz)
  Create a rank 1 tensory of size `sz`.
- virtual double get (size_t ∗index)
  Get the element indexed by `index`.
- virtual int set (size_t ∗index, double val)
  Set the element indexed by `index` *to value* `val`.
- virtual double get (size_t ix)
  Get the element indexed by `ix`.
- virtual int set (size_t index, double val)
  Set the element indexed by `index` *to value* `val`.
- virtual double & operator[ ] (size_t ix)
  Get an element using array-like indexing.
- virtual double & operator() (size_t ix)
  Get an element using operator().

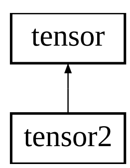
The documentation for this class was generated from the following file:

- tensor.h

7.320 tensor2 Class Reference

```
#include <tensor.h>
```

Inheritance diagram for tensor2::



7.320.1 Detailed Description

Rank 2 tensor.

Definition at line 781 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).
- virtual double get (size_t ∗index)
  Get the element indexed by `index`.
- virtual int set (size_t ∗index, double val)
  Set the element indexed by `index` *to value* `val`.
- virtual double get (size_t ix1, size_t ix2)
  Get the element indexed by (ix1,ix2).
- virtual int set (size_t ix1, size_t ix2, double val)
  Set the element indexed by (ix1,ix2) to value `val`.

- virtual double & operator() (size_t ix, size_t iy)
  Get the element indexed by (ix1,ix2).

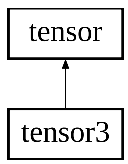
The documentation for this class was generated from the following file:

- tensor.h

7.321 tensor3 Class Reference

```
#include <tensor.h>
```

Inheritance diagram for tensor3::



7.321.1 Detailed Description

Rank 3 tensor.

Definition at line 886 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).
- virtual double get (size_t *index)
  Get the element indexed by `index`.
- virtual int set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`.
- virtual double get (size_t ix1, size_t ix2, size_t ix3)
  Get the element indexed by (ix1,ix2,ix3).
- virtual int set (size_t ix1, size_t ix2, size_t ix3, double val)
  Set the element indexed by (ix1,ix2,ix3) to value `val`.

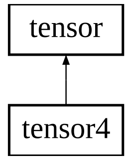
The documentation for this class was generated from the following file:

- tensor.h

7.322 tensor4 Class Reference

```
#include <tensor.h>
```

Inheritance diagram for tensor4::



7.322.1 Detailed Description

Rank 4 tensor.

Definition at line 989 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).
- virtual double get (size_t ∗index)
  Get the element indexed by `index`.
- virtual int set (size_t ∗index, double val)
  Set the element indexed by `index` *to value* `val`.
- virtual double get (size_t ix1, size_t ix2, size_t ix3, size_t ix4)
  Get the element indexed by (ix1,ix2,ix3,ix4).
- virtual int 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`.

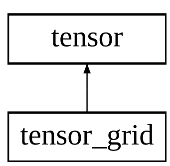
The documentation for this class was generated from the following file:

- tensor.h

7.323 tensor_grid Class Template Reference

`#include <tensor.h>`

Inheritance diagram for tensor_grid::



7.323.1 Detailed Description

template<template< class c_t > class base_interp_t = cspline_interp> class tensor_grid< base_interp_t >

Tensor class with arbitrary dimensions.

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().

Idea for future

Only allocate space for grid if it is set

Idea for future

Could implement arithmetic operators + and - and some different products.

Definition at line 360 of file tensor.h.

Public Member Functions

- tensor_grid ()
  Create an empty tensor with zero rank.
- tensor_grid (size_t rank, size_t *dim)
  Create a tensor of rank `rank` *with sizes given in* `dim`.
- virtual int set_val (double *grdp, double val)
  Set the element closest to grid point `grdp` *to value* `val`.
- virtual int set_val (double *grdp, double val, double *closest)
  Set the element closest to grid point `grdp` *to value* `val`.
- virtual double get_val (double *grdp)
  Get the element closest to grid point `grdp`.
- virtual double get_val (double *grdp, double *closest)
  Get the element closest to grid point `grdp` *to value* `val`.
- virtual int set_grid (double **val)
  Set the grid.
- virtual int tensor_allocate (size_t rank, size_t *dim)
  Allocate space for a tensor of rank `rank` *with sizes given in* `dim`.
- virtual int tensor_free ()
  Free allocated space (also sets rank to zero).
- virtual size_t lookup_grid (size_t i, double val)
  Lookup index for grid closest to `val`.
- virtual double get_grid (size_t i, size_t j)
  Lookup index for grid closest to `val`.
- virtual int lookup_grid (double *vals, size_t *indices)
  Lookup indices for grid closest to `val`.
- virtual size_t lookup_grid_val (size_t i, double val, double &val2)
  Lookup index for grid closest to `val`*, returning the grid point.*
- virtual double interpolate (double *vals)
  Interpolate values `vals` *into the tensor, returning the result.*

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.

7.323.2 Constructor & Destructor Documentation

7.323.2.1 tensor_grid (size_t *rank*, size_t * *dim*) [inline]

Create a tensor of rank `rank` with sizes given in `dim`.

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 390 of file tensor.h.

7.323.3 Member Function Documentation

7.323.3.1 virtual int set_grid (double ** *val*) [inline, virtual]

Set the grid.

The parameter `grid` must define the grid, so that `val`[i][j] is the jth grid point for the ith index. The size of array `grid`[i] should be given by `dim`[i] where `dim` was the argument given in the constructor or to the function tensor_allocate().

Note that the grid is copied so the function argument may be destroyed by the user after calling set_grid().

Idea for future

Define a more generic interface for matrix types

Definition at line 521 of file tensor.h.

7.323.3.2 virtual int tensor_allocate (size_t *rank*, size_t * *dim*) [inline, virtual]

Allocate space for a tensor of rank `rank` with sizes given in `dim`.

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 547 of file tensor.h.

7.323.3.3 virtual double interpolate (double * *vals*) [inline, virtual]

Interpolate values `vals` into the tensor, returning the result.

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

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.

Definition at line 654 of file tensor.h.

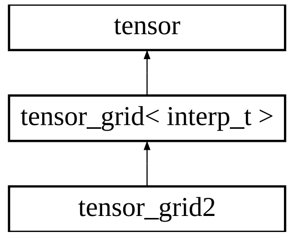
The documentation for this class was generated from the following file:

- tensor.h

7.324 tensor_grid2 Class Template Reference

`#include <tensor.h>`

Inheritance diagram for tensor_grid2::



7.324.1 Detailed Description

template<template< class c_t > class interp_t = cspline_interp> class tensor_grid2< interp_t >

Rank 2 tensor with a grid.

Definition at line 828 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,sz3).
- virtual double get (size_t ∗index)
  Get the element indexed by `index`.
- virtual int set (size_t ∗index, double val)
  Set the element indexed by `index` *to value* `val`.
- virtual double get (size_t ix1, size_t ix2)
  Get the element indexed by (ix1,ix2,ix3).
- virtual int set (size_t ix1, size_t ix2, double val)
  Set the element indexed by (ix1,ix2,ix3) to value `val`.

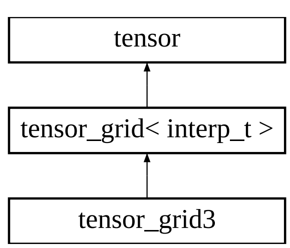
The documentation for this class was generated from the following file:

- tensor.h

7.325 tensor_grid3 Class Template Reference

`#include <tensor.h>`

Inheritance diagram for tensor_grid3::



7.325.1 Detailed Description

template<template< class c_t > class interp_t = cspline_interp> class tensor_grid3< interp_t >

Rank 3 tensor with a grid.

Definition at line 929 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).
- virtual double get (size_t *index)
  Get the element indexed by `index`.
- virtual int set (size_t *index, double val)
  Set the element indexed by `index` *to value* `val`.
- virtual double get (size_t ix1, size_t ix2, size_t ix3)
  Get the element indexed by (ix1,ix2,ix3).
- virtual int set (size_t ix1, size_t ix2, size_t ix3, double val)
  Set the element indexed by (ix1,ix2,ix3) to value `val`.

The documentation for this class was generated from the following file:

- tensor.h

7.326 test_mgr Class Reference

`#include <test_mgr.h>`

7.326.1 Detailed Description

A class to manage testing and record success and failure.

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 38 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 $|\mathrm{result}-\mathrm{expected}|/\mathrm{expected} < \mathrm{rel\_error}$.
- bool test_abs (double result, double expected, double abs_error, std::string description)
  Test for $|\mathrm{result}-\mathrm{expected}|/ < \mathrm{abs\_error}$.
- bool test_fact (double result, double expected, double factor, std::string description)
  Test for $1/factor < result/expected < factor$ *??*
- bool test_str (std::string result, std::string expected, std::string description)
  Test for $\mathrm{result} = \mathrm{expected}$.
- bool test_gen (bool value, std::string description)
  Test for $\mathrm{result} = \mathrm{expected}$.
- template<class vec_t, class vec2_t>
  bool test_rel_arr (int nv, vec_t &result, vec2_t &expected, double rel_error, std::string description)
  Test for $|\mathrm{result}-\mathrm{expected}|/\mathrm{expected} < \mathrm{rel\_error}$.
- template<class mat_t, class mat2_t>
  bool test_rel_mat (int nr, int nc, mat_t &result, mat2_t &expected, double rel_error, std::string description)
  Test for $|\mathrm{result}-\mathrm{expected}|/\mathrm{expected} < \mathrm{rel\_error}$.
- template<class vec_t, class vec2_t>
  bool test_abs_arr (int nv, vec_t &result, vec2_t &expected, double rel_error, std::string description)
  Test for $|\mathrm{result}-\mathrm{expected}|/ < \mathrm{abs\_error}$.
- template<class vec_t, class vec2_t>
  bool test_fact_arr (int nv, vec_t &result, vec2_t &expected, double factor, std::string description)
  Test for $1/factor < result/expected < factor$ *??*
- template<class vec_t>
  bool test_gen_arr (int nv, vec_t &result, 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).

7.326.2 Member Function Documentation

7.326.2.1 bool report ()

Provide a report of all tests so far.

Returns true if all tests have passed and false if at least one test failed.

7.326.2.2 void set_output_level (int *l*) `[inline]`

Set the output level.

Possible values:

- 0 = No output
- 1 = Output only tests that fail
- 2 = Output all tests

Definition at line 78 of file test_mgr.h.

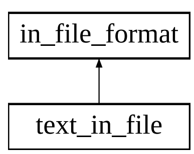
The documentation for this class was generated from the following file:

- test_mgr.h

7.327 text_in_file Class Reference

```
#include <text_file.h>
```

Inheritance diagram for text_in_file::



7.327.1 Detailed Description

An input text file.

Note: Text files are not entirely architecture-independent, For example, a larger integer will not be read correctly on small integer systems.

Definition at line 242 of file text_file.h.

Public Member Functions

- text_in_file (std::istream ∗in_file)
  Use input stream `in_file` *for text input.*
- text_in_file (std::string file_name)
  Read an input file with name `file_name`*.*
- virtual int bool_in (bool &dat, std::string name="")
  Input a bool variable.
- virtual int char_in (char &dat, std::string name="")
  Input a char variable.
- virtual int double_in (double &dat, std::string name="")
  Input a double variable.
- virtual int float_in (float &dat, std::string name="")
  Input a float variable.
- virtual int int_in (int &dat, std::string name="")
  Input an int variable.
- virtual int long_in (unsigned long int &dat, std::string name="")
  Input an long variable.

- virtual int string_in (std::string &dat, std::string name="")
  Input a string variable.
- virtual int word_in (std::string &dat, std::string name="")
  Input a word variable.
- virtual int start_object (std::string &type, std::string &name)
  Start object input.
- virtual int skip_object ()
  Skip the present object for the next call to read_type().
- virtual int end_object ()
  End object input.
- virtual int init_file ()
  Initialize file input.
- virtual int clean_up ()
  Finish file input.
- std::string reformat_string (std::string in)
  Add brackets and replace carriage returns with spaces.

Protected Member Functions

- bool is_hc_type (std::string type)
  If true, then `type` is a "hard-coded" type.
- virtual int word_in_noerr (std::string &dat, std::string name="")
  A version of word_in() which doesn't call the error handler.

Protected Attributes

- std::stack< bool > hcs
  A list to indicate if the current object and subobjects are "hard-coded".
- std::istream * ins
  The input stream.
- bool from_string
  True if the string version of the constructor was called.

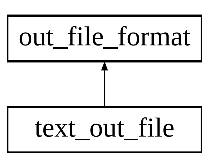
The documentation for this class was generated from the following file:

- text_file.h

7.328 text_out_file Class Reference

`#include <text_file.h>`

Inheritance diagram for text_out_file::



7.328.1 Detailed Description

An output text file.

A collection file is simply a text file containing a list of objects specially formatted for input and output. Each entry in the text file is of the form:

object_type object_name object_version word_number word1 word2 word3 ...

The type, name, and version are all strings that contain no whitespace. "a_name" is a valid name but, "a name" is not.

Parts of the object definition may be separated by any amount of whitespace, with the exception of 'strings'.

The collection file may contain comments, which are lines that begin with the '#' character. Comments may last more than one line (so long as every line begins with '#'), but they may not occur in the middle of an object definition.

```
# comment 1
double a 4.0
double[] c 4
4.5 3.4 2.3 4.2
# comment 2
double b 5.0
```

is acceptable, but

```
# comment 1
double a 4.0
double[] c 4
4.5 3.4 2.3 4.2
double b
# comment 2
5.0
```

is not, since the second comment occurs in the middle of the definition for the object named 'b'.

Normal variable: type name version word data1 ... data2

Object containing pointer: type name version word data1 ... ptr_type ptr_name ... data2

where ptr_type is the type of the object being pointed to and ptr_name is the name of the object. If it's not in the list, then the object is assigned a unique name of the form ptrX where 'X' is an integer >= 0.

Static objects are the same, except they are preceeded by the keyword `static` and do not have a name associated with them.

This is useful for output to text files and to std::cout, i.e. text_out_file tf(&cout);

Note: Text files are not entirely architecture-independent, For example, a larger integer will not be read correctly on small integer systems.

Definition at line 97 of file text_file.h.

Public Member Functions

- text_out_file (std::ostream ∗out_file, int width=80)
  Use output stream `out_file` *for text output.*
- text_out_file (std::string file_name, std::ostream ∗prop=NULL, bool append=false, int width=80)
  Create an output file with name `file_name`*.*
- virtual int bool_out (bool dat, std::string name="")
  Output a bool variable.
- virtual int char_out (char dat, std::string name="")
  Output a char variable.
- virtual int char_out_internal (char dat, std::string name="")
  Output a char variable.
- virtual int double_out (double dat, std::string name="")
  Output a double variable.

- virtual int float_out (float dat, std::string name="")
  Output a float variable.
- virtual int int_out (int dat, std::string name="")
  Output an int variable.
- virtual int long_out (unsigned long int dat, std::string name="")
  Output an long variable.
- virtual int string_out (std::string dat, std::string name="")
  Output a string.
- virtual int word_out (std::string dat, std::string name="")
  Output a word.
- virtual int start_object (std::string type, std::string name)
  Start object output.
- virtual int end_object ()
  End object output.
- virtual int end_line ()
  End line.
- virtual int init_file ()
  Output initialization.
- virtual int clean_up ()
  Finish the file.
- int comment_out (std::string comment)
  Output a comment (only for text files).
- std::string reformat_string (std::string in)
  Add brackets and replace carriage returns with spaces.

Protected Member Functions

- virtual int flush ()
  Flush the string buffer.
- bool is_hc_type (std::string type)
  If true, then `type` is a "hard-coded" type.

Protected Attributes

- std::stack< bool > hcs
  A list to indicate if the current object and subobjects are "hard-coded".
- bool from_string
  True if the constructor was called with a string, false otherwise.
- bool compressed
  True if the file is to be compressed.
- bool gzip
  True if the file is to be compressed with gzip.
- int file_width
  The width of the file.
- std::ostream * outs
  The output stream.
- std::ostream * props
  A pointer to an output stream to define output properties.
- std::ostringstream * strout
  The temporary buffer as a stringstream.
- std::string user_filename
  The user-specified filename.
- std::string temp_filename
  The temporary filename used.

7.328.2 Constructor & Destructor Documentation

7.328.2.1 text_out_file (std::ostream * *out_file*, int *width* = 80)

Use output stream `out_file` for text output.

This constructor assumes that the I/O properties of `out_file` have already been set.

Note that the stream `out_file` should not have been opened in binary mode, and errors will likely occur if this is the case.

Todo

Ensure streams are not opened in binary mode for safety.

7.328.2.2 text_out_file (std::string *file_name*, std::ostream * *prop* = NULL, bool *append* = false, int *width* = 80)

Create an output file with name `file_name`.

If `prop` is not null, then the I/O properties (precision, fill, flags, etc) for the newly created file are taken to be the same as `prop` .

The documentation for this class was generated from the following file:

- text_file.h

7.329 timer_clock Class Reference

```
#include <timer.h>
```

7.329.1 Detailed Description

Provide an interface for timing execution using `clock()`.

Note:

Note that the time return by clock() is reset on some regular interval (sometimes 72 minutes) and this class does not yet account for this.

Definition at line 116 of file timer.h.

Public Member Functions

- double time_since ()
  Number of seconds elapsed.
- void time_since (int &d, int &h, int &m, int &s, double &f)
  Time elapsed in days, hours, minutes, seconds, and fractions of seconds.
- void time_remaining (int n, int tot, int &d, int &h, int &m, int &s, double &f)
  Time remaining if `n` out of `tot` tasks have been completed.
- std::string interval_to_string (int d, int h, int m, int s, double f=0.0)
  Convert a time interval to a string.

Protected Attributes

- clock_t time
  Desc.

The documentation for this class was generated from the following file:

- timer.h

7.330 timer_gettod Class Reference

```
#include <timer.h>
```

7.330.1 Detailed Description

Provide an interface for timing execution using `gettimeofday()`.

Todo

Better testing which doesn't use a fixed number of mathematical operations, but automatically selects enough operations.

Definition at line 44 of file timer.h.

Public Member Functions

- int reset ()
  Set time 'zero'.
- int set ()
  Store the present time.
- double seconds_elapsed ()
  Return the number of seconds between set() and reset().
- int time_elapsed (int &d, int &h, int &m, int &s, int &usec)
  Return the time between set() and reset().
- int time_remaining (int n, int ntot, int &d, int &h, int &m, int &s, int &usec)
  Time remaining if `n` out of `tot` tasks have been completed.
- std::string time_remaining (int n, int ntot)
  Time remaining if `n` out of `tot` tasks have been completed.
- std::string interval_to_string (int d, int h, int m, int s, int usec)
  Convert a time interval to a string.

Protected Attributes

- struct timeval zero
  The last time the clock was reset.
- struct timeval mark
  The most resent time from set().
- struct timezone tz
  The timezone.

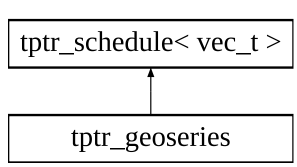
The documentation for this class was generated from the following file:

- timer.h

7.331 tptr_geoseries Class Template Reference

```
#include <tptr_geoseries.h>
```

Inheritance diagram for tptr_geoseries::



7.331.1 Detailed Description

template<class vec_t = ovector_view> class tptr_geoseries< vec_t >

Temperature schedule for a geometric series.

The temperature begins at `start`, and is divided by `ratio`, until it is smaller than `end`. The ending value is divided by `sqrt(ratio)` to avoid finite precision problems for series when `start/end` is an integral power of `ratio`.

The default schedule is $T = 1/(1.01)^n$ for $n = 0, 1, 2, 3, ..., 463$ (until $T < 0.01$) given by `ustart=1`, `uend=0.01`, `uratio=1.01`.

Definition at line 48 of file tptr_geoseries.h.

Public Member Functions

- int set_series (double udstart, double uend, double uratio)
  Set the limits for the geometric series.
- int get_npoints ()
  Get the number of temperatures in the series.
- virtual double start (double min, int nv, const vec_t &best, void ∗vp)
  Return the first temperature.
- virtual double next (double min, int nv, const vec_t &best, void ∗vp)
  Return the next temperature.
- virtual bool done (double min, int nv, const vec_t &best, void ∗vp)
  Return true if the last step made the temperature too small.
- virtual const char ∗ type ()
  Return string denoting type ("tptr_geoseries").

Protected Attributes

- double last
  The last temperature returned.

parameters for the schedule

- double **dstart**
- double **end**
- double **ratio**

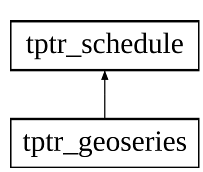
The documentation for this class was generated from the following file:

- tptr_geoseries.h

7.332 tptr_schedule Class Template Reference

```
#include <tptr_schedule.h>
```

Inheritance diagram for tptr_schedule::



7.332.1 Detailed Description

template<class vec_t = ovector_view> class tptr_schedule< vec_t >

Simulated annealing temperature schedule base.

The schedules are designed to be used in the following way

```
for(temper=tp->start(fmin,nvar,pb,NULL);
tp->done(fmin,nvar,pb,NULL)==false;
temper=tp->next(fmin,nvar,pb,NULL)) {
}
```

Definition at line 47 of file tptr_schedule.h.

Public Member Functions

- virtual double start (double min, int nv, const vec_t &best, void ∗vp)
  Return the first temperature.
- virtual double next (double min, int nv, const vec_t &best, void ∗vp)
  Return the next temperature.
- virtual bool done (double min, int nv, const vec_t &best, void ∗vp)
  Return true if the last step made the temperature too small.
- virtual const char ∗ type ()
  Return string denoting type ("tptr_schedule").

The documentation for this class was generated from the following file:

- tptr_schedule.h

7.333 twod_eqi_intp Class Reference

`#include <twod_eqi_intp.h>`

7.333.1 Detailed Description

Two-dimensional interpolation for equally-spaced intervals.

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 55 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.

7.333.2 Member Function Documentation

7.333.2.1 int set_type (int *type*) [inline]

Set the interpolation type.

- 3: 3-point
- 4: 4-point
- 6: 6-point (default)

Definition at line 78 of file twod_eqi_intp.h.

The documentation for this class was generated from the following file:

- twod_eqi_intp.h

7.334 twod_intp Class Reference

```
#include <twod_intp.h>
```

7.334.1 Detailed Description

Two-dimensional interpolation class.

This class implements two-dimensional interpolation. Derivatives and integrals along both x- and y-directions can be computed. The function set_data(), does not copy the data but rather stores pointers to the data. If the data is modified, then the function reset_interp() can be called to reset the interpolation information with the original pointer information.

The storage for the data, including the arrays `x_fun` and `y_fun` are all managed by the user. If the data is changed without calling reset_interp(), then interp() will return incorrect results.

By default, cubic spline interpolation with natural boundary conditions is used. This can be changed with the set_interp() function.

Warning:

> This class assumes that the data specified through set_data() is not deallocated or modified by the user until unset_data() has been called.

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 68 of file twod_intp.h.

Public Member Functions

- int set_data (int n_x, int n_y, ovector &x_fun, ovector &y_fun, omatrix &u_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 (size_t ni, base_interp< ovector_view > *it, base_interp< ovector_const_subvector > *it_sub, base_interp< ovector_view > &it2, base_interp< ovector_const_subvector > &it2_sub)
  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.

7.334.2 Member Function Documentation

7.334.2.1 int set_data (int *n_x*, int *n_y*, ovector & *x_fun*, ovector & *y_fun*, omatrix & *u_data*, bool *x_first* = `true`)

Initialize the data for the 2-dimensional interpolation.

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. In general, if the data is "more accurate" in the x direction than in the y direction, it is probably better to choose `x_first=true`.

7.334.2.2 int reset_interp ()

Reset the stored interpolation since the data has changed.

This will return an error if the set_data() has not been called

7.334.2.3 int set_interp (size_t *ni*, base_interp< ovector_view > * *it*, base_interp< ovector_const_subvector > * *it_sub*, base_interp< ovector_view > & *it2*, base_interp< ovector_const_subvector > & *it2_sub*) `[inline]`

Specify the base interpolation objects to use.

This allows the user to provide new interpolation objects for use in the two-dimensional interpolation. This class requires an array of interpolation objects for the first two arguments because one interpolation object is required for each row (or each column). The argument `ni` specifies the size of these arrays. In the case where the user intends the x interpolation first (i.e. `x_first` = `true` in set_data()), the parameter `ni` should be equal to `ny`. For `x_first=false`, `ni` should be equal to `nx`. If the class does not find enough interpolation objects, i.e. if `ni` is smaller than the values suggested above, the class will switch back to the default internal interpolation objects when it runs out of user-specified interpolation objects. 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
linear_interp<ovector_view> li[41];
linear_interp<ovector_const_subvector> li2[41];

ti.set_interp(40,li,li2,li[40],li2[40]);
ti.set_data(20,40,x,y,d,true);
```

This function automatically calls reset_interp() if the data has already been set to reset the internal interpolation objects.

Idea for future

Use std::vector for the first two base_interp arguments?

Definition at line 183 of file twod_intp.h.

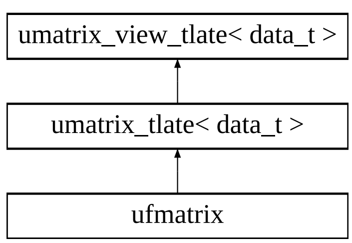
The documentation for this class was generated from the following file:

- twod_intp.h

7.335 ufmatrix Class Template Reference

`#include <umatrix_tlate.h>`

Inheritance diagram for ufmatrix::



7.335.1 Detailed Description

template<size_t N, size_t M> class ufmatrix< N, M >

A matrix where the memory allocation is performed in the constructor.

Definition at line 701 of file umatrix_tlate.h.

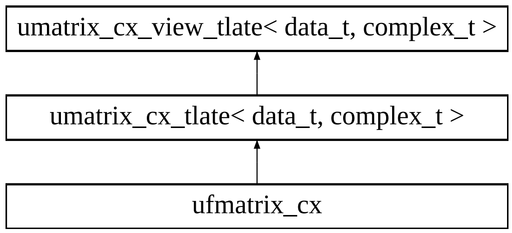
The documentation for this class was generated from the following file:

- umatrix_tlate.h

7.336 ufmatrix_cx Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for ufmatrix_cx::



7.336.1 Detailed Description

template<size_t N, size_t M> class ufmatrix_cx< N, M >

A matrix where the memory allocation is performed in the constructor.

Definition at line 705 of file umatrix_cx_tlate.h.

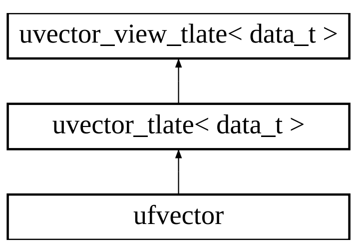
The documentation for this class was generated from the following file:

- umatrix_cx_tlate.h

7.337 ufvector Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for ufvector::



7.337.1 Detailed Description

template<size_t N = 0> class ufvector< N >

A vector with unit-stride where the memory allocation is performed in the constructor.

Definition at line 851 of file uvector_tlate.h.

The documentation for this class was generated from the following file:

- uvector_tlate.h

7.338 umatrix_alloc Class Reference

```
#include <umatrix_tlate.h>
```

7.338.1 Detailed Description

A simple class to provide an `allocate()` function for umatrix.

Definition at line 689 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.

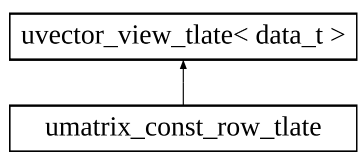
The documentation for this class was generated from the following file:

- umatrix_tlate.h

7.339 umatrix_const_row_tlate Class Template Reference

```
#include <umatrix_tlate.h>
```

Inheritance diagram for umatrix_const_row_tlate::



7.339.1 Detailed Description

template<class data_t> class umatrix_const_row_tlate< data_t >

Create a const vector from a row of a matrix.

Definition at line 609 of file umatrix_tlate.h.

Public Member Functions

- umatrix_const_row_tlate (const umatrix_view_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

7.340 umatrix_cx_alloc Class Reference

```
#include <umatrix_cx_tlate.h>
```

7.340.1 Detailed Description

A simple class to provide an `allocate()` function for umatrix_cx.

Definition at line 693 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.

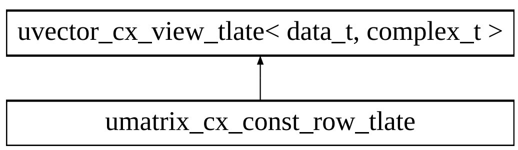
The documentation for this class was generated from the following file:

- umatrix_cx_tlate.h

7.341 umatrix_cx_const_row_tlate Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for umatrix_cx_const_row_tlate::



7.341.1 Detailed Description

template<class data_t, class complex_t> class umatrix_cx_const_row_tlate< data_t, complex_t >

Create a const vector from a row of a matrix.

Definition at line 622 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`*.*

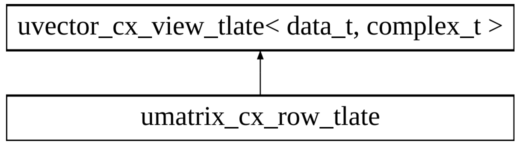
The documentation for this class was generated from the following file:

- umatrix_cx_tlate.h

7.342 umatrix_cx_row_tlate Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for umatrix_cx_row_tlate::



7.342.1 Detailed Description

template<class data_t, class complex_t> class umatrix_cx_row_tlate< data_t, complex_t >

Create a vector from a row of a matrix.

Definition at line 606 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`*.*

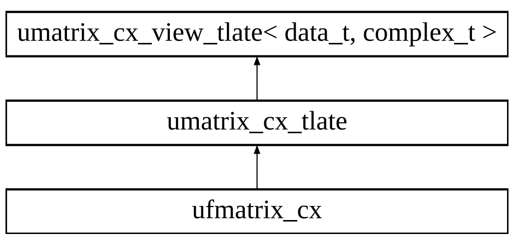
The documentation for this class was generated from the following file:

- umatrix_cx_tlate.h

7.343 umatrix_cx_tlate Class Template Reference

```
#include <umatrix_cx_tlate.h>
```

Inheritance diagram for umatrix_cx_tlate::



7.343.1 Detailed Description

template<class data_t, class complex_t> class umatrix_cx_tlate< data_t, complex_t >

A matrix of double-precision numbers.

Definition at line 372 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).

7.343.2 Member Function Documentation

7.343.2.1 int free () `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 577 of file umatrix_cx_tlate.h.

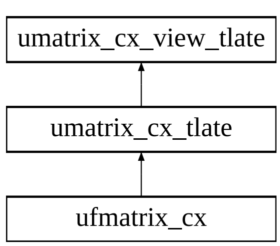
The documentation for this class was generated from the following file:

- umatrix_cx_tlate.h

7.344 umatrix_cx_view_tlate Class Template Reference

`#include <umatrix_cx_tlate.h>`

Inheritance diagram for umatrix_cx_view_tlate::



7.344.1 Detailed Description

template<class data_t, class complex_t> class umatrix_cx_view_tlate< data_t, complex_t >

A matrix view of complex numbers.

Definition at line 50 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.

7.344.2 Member Function Documentation

7.344.2.1 size_t rows () const `[inline]`

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

Definition at line 263 of file umatrix_cx_tlate.h.

7.344.2.2 size_t cols () const `[inline]`

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 273 of file umatrix_cx_tlate.h.

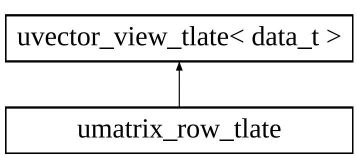
The documentation for this class was generated from the following file:

- umatrix_cx_tlate.h

7.345 umatrix_row_tlate Class Template Reference

`#include <umatrix_tlate.h>`

Inheritance diagram for umatrix_row_tlate::



7.345.1 Detailed Description

template<class data_t> class umatrix_row_tlate< data_t >

Create a vector from a row of a matrix.

Definition at line 591 of file umatrix_tlate.h.

Public Member Functions

- umatrix_row_tlate (umatrix_view_tlate< data_t > &m, size_t i)
  Create a vector from row `i` *of matrix* `m`*.*

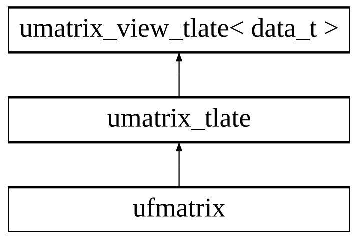
The documentation for this class was generated from the following file:

- umatrix_tlate.h

7.346 umatrix_tlate Class Template Reference

```
#include <umatrix_tlate.h>
```

Inheritance diagram for umatrix_tlate::



7.346.1 Detailed Description

template<class data_t> class umatrix_tlate< data_t >

A matrix of double-precision numbers.

Definition at line 358 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).

7.346.2 Member Function Documentation

7.346.2.1 int free () `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 562 of file umatrix_tlate.h.

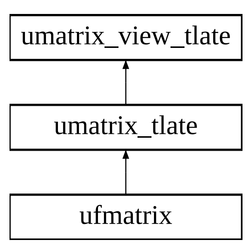
The documentation for this class was generated from the following file:

- umatrix_tlate.h

7.347 umatrix_view_tlate Class Template Reference

```
#include <umatrix_tlate.h>
```

Inheritance diagram for umatrix_view_tlate::



7.347.1 Detailed Description

template<class data_t> class umatrix_view_tlate< data_t >

A matrix view of double-precision numbers.

Definition at line 52 of file umatrix_tlate.h.

Public Member Functions

Copy constructors

- umatrix_view_tlate (const umatrix_view_tlate &v)
  So2scllow copy constructor - create a new view of the same matrix.
- umatrix_view_tlate & operator= (const umatrix_view_tlate &v)
  So2scllow 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 (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).
- 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`.
- 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_view_tlate< data_t > & operator+= (const umatrix_view_tlate< data_t > &x)
  operator+=
- umatrix_view_tlate< data_t > & operator-= (const umatrix_view_tlate< data_t > &x)
  operator-=
- umatrix_view_tlate< data_t > & operator+= (const data_t &y)
  operator+=
- umatrix_view_tlate< data_t > & operator-= (const data_t &y)
  operator-=
- umatrix_view_tlate< data_t > & operator*= (const data_t &y)
  operator=*

Protected Member Functions

- umatrix_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.

7.347.2 Member Function Documentation

7.347.2.1 size_t rows () const [inline]

Method to return number of rows.

If no memory has been allocated, this will quietly return zero.

Definition at line 249 of file umatrix_tlate.h.

7.347.2.2 size_t cols () const [inline]

Method to return number of columns.

If no memory has been allocated, this will quietly return zero.

Definition at line 259 of file umatrix_tlate.h.

The documentation for this class was generated from the following file:

- umatrix_tlate.h

7.348 uvector_alloc Class Reference

```
#include <uvector_tlate.h>
```

7.348.1 Detailed Description

A simple class to provide an `allocate()` function for uvector.

Definition at line 828 of file uvector_tlate.h.

Public Member Functions

- void allocate (uvector &o, int i)
  Allocate `v` *for* `i` *elements.*
- void free (uvector &o)
  Free memory.

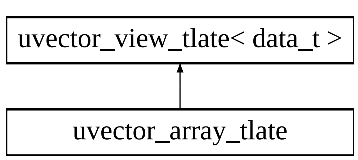
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.349 uvector_array_tlate Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for uvector_array_tlate::



7.349.1 Detailed Description

template<class data_t> class uvector_array_tlate< data_t >

Create a vector from an array.

Definition at line 628 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`.

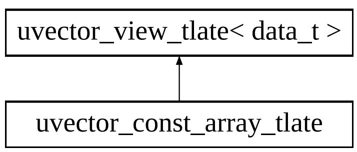
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.350 uvector_const_array_tlate Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for uvector_const_array_tlate::



7.350.1 Detailed Description

template<class data_t> class uvector_const_array_tlate< data_t >

Create a vector from an const array.

Definition at line 662 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`.

Protected Member Functions

Ensure \c const by hiding non-const members

- 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)

Set (with optional range-checking).
- int swap (uvector_view_tlate< data_t > &x)
  Swap vectors.
- int **set_all** (double val)
- uvector_view_tlate< data_t > & operator+= (const uvector_view_tlate< data_t > &x)
  operator+=
- uvector_view_tlate< data_t > & operator-= (const uvector_view_tlate< data_t > &x)
  operator-=
- uvector_view_tlate< data_t > & operator*= (const data_t &y)
  operator=*

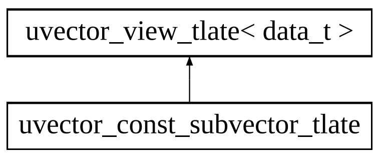
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.351 uvector_const_subvector_tlate Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for uvector_const_subvector_tlate::



7.351.1 Detailed Description

template<class data_t> class uvector_const_subvector_tlate< data_t >

Create a const vector from a subvector of another vector.

Definition at line 712 of file uvector_tlate.h.

Public Member Functions

- uvector_const_subvector_tlate (const uvector_view_tlate< data_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`.

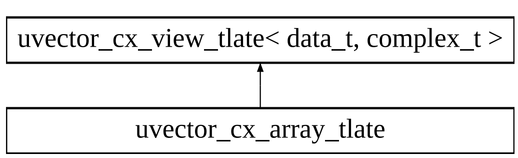
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.352 uvector_cx_array_tlate Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for uvector_cx_array_tlate::



7.352.1 Detailed Description

template<class data_t, class complex_t> class uvector_cx_array_tlate< data_t, complex_t >

Create a vector from an array.

Definition at line 511 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`*.*

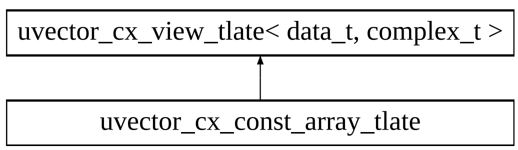
The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

7.353 uvector_cx_const_array_tlate Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for uvector_cx_const_array_tlate::



7.353.1 Detailed Description

template<class data_t, class complex_t> class uvector_cx_const_array_tlate< data_t, complex_t >

Create a vector from an array.

Definition at line 545 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 swap (uvector_cx_view_tlate< data_t, complex_t > &x)
  Swap vectors.
- 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=*

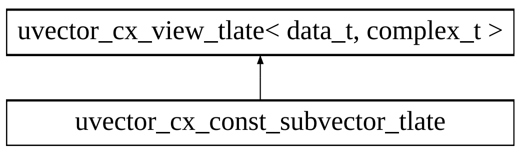
The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

7.354 uvector_cx_const_subvector_tlate Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for uvector_cx_const_subvector_tlate::



7.354.1 Detailed Description

template<class data_t, class complex_t> class uvector_cx_const_subvector_tlate< data_t, complex_t >

Create a vector from a subvector of another.

Definition at line 596 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`.

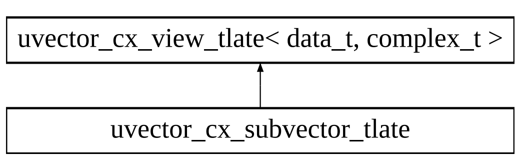
The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

7.355 uvector_cx_subvector_tlate Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for uvector_cx_subvector_tlate::



7.355.1 Detailed Description

template<class data_t, class complex_t> class uvector_cx_subvector_tlate< data_t, complex_t >

Create a vector from a subvector of another.

Definition at line 526 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`.

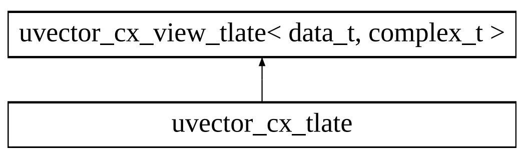
The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

7.356 uvector_cx_tlate Class Template Reference

```
#include <uvector_cx_tlate.h>
```

Inheritance diagram for uvector_cx_tlate::



7.356.1 Detailed Description

template<class data_t, class complex_t> class uvector_cx_tlate< data_t, complex_t >

A vector of double-precision numbers with unit stride.

There are several global binary operators associated with objects of type uvector_cx_tlate. The are documented in the "Functions" section of uvector_cx_tlate.h.

Definition at line 342 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.

7.356.2 Member Function Documentation

7.356.2.1 int free () `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 496 of file uvector_cx_tlate.h.

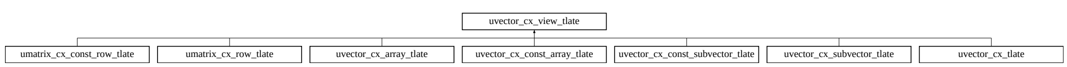
The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

7.357 uvector_cx_view_tlate Class Template Reference

`#include <uvector_cx_tlate.h>`

Inheritance diagram for uvector_cx_view_tlate::



7.357.1 Detailed Description

template<class data_t, class complex_t> class uvector_cx_view_tlate< data_t, complex_t >

A vector view of complex numbers with unit stride.

Idea for future

Write lookup() method, and possibly an erase() method.

Definition at line 48 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

- int swap (uvector_cx_view_tlate< data_t, complex_t > &x)
  Swap vectors.
- 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.

7.357.2 Member Function Documentation

7.357.2.1 size_t size () const `[inline]`

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 234 of file uvector_cx_tlate.h.

The documentation for this class was generated from the following file:

- uvector_cx_tlate.h

7.358 uvector_int_alloc Class Reference

```
#include <uvector_tlate.h>
```

7.358.1 Detailed Description

A simple class to provide an `allocate()` function for uvector_int.

Definition at line 839 of file uvector_tlate.h.

Public Member Functions

- void allocate (uvector_int &o, int i)
  Allocate `v` *for* `i` *elements.*
- void free (uvector_int &o)
  Free memory.

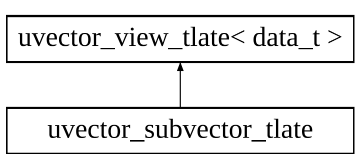
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.359 uvector_subvector_tlate Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for uvector_subvector_tlate::



7.359.1 Detailed Description

template<class data_t> class uvector_subvector_tlate< data_t >

Create a vector from a subvector of another.

Definition at line 643 of file uvector_tlate.h.

Public Member Functions

- uvector_subvector_tlate (uvector_view_tlate< data_t > &orig, size_t offset, size_t n)
  Create a vector from `orig`*.*

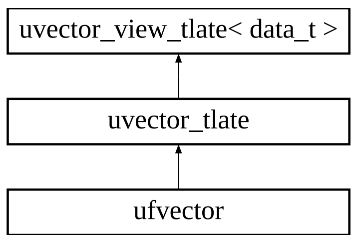
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.360 uvector_tlate Class Template Reference

```
#include <uvector_tlate.h>
```

Inheritance diagram for uvector_tlate::



7.360.1 Detailed Description

template<class data_t> class uvector_tlate< data_t >

A vector with unit stride.

There are several global binary operators associated with objects of type uvector_tlate. The are documented in the "Functions" section of uvector_tlate.h.

Todo

Create a sort_unique() method as in ovector.

Definition at line 401 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_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_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.

7.360.2 Member Function Documentation

7.360.2.1 int free () `[inline]`

Free the memory.

This function will safely do nothing if used without first allocating memory or if called multiple times in succession.

Definition at line 561 of file uvector_tlate.h.

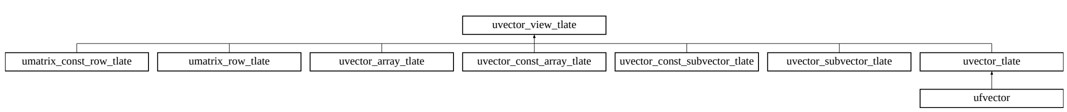
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.361 uvector_view_tlate Class Template Reference

#include <uvector_tlate.h>

Inheritance diagram for uvector_view_tlate::



7.361.1 Detailed Description

template<class data_t> class uvector_view_tlate< data_t >

A vector view with unit stride.

Idea for future

Could allow user-defined specification of restrict keyword

Definition at line 52 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.

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 (data_t val)
  Set all of the value to be the value `val`.
- size_t size () const
  Method to return vector size.

Other methods

- int swap (uvector_view_tlate< data_t > &x)
  Swap vectors.
- 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.

Arithmetic

- uvector_view_tlate< data_t > & operator+= (const uvector_view_tlate< data_t > &x)
  operator+=
- uvector_view_tlate< data_t > & operator-= (const uvector_view_tlate< data_t > &x)
  operator-=
- uvector_view_tlate< data_t > & operator+= (const data_t &y)
  operator+=
- uvector_view_tlate< data_t > & operator-= (const data_t &y)
  operator-=
- uvector_view_tlate< data_t > & operator*= (const data_t &y)
  operator=*
- data_t norm () const
  Norm.

Protected Member Functions

- uvector_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.

7.361.2 Member Function Documentation

7.361.2.1 size_t size () const [inline]

Method to return vector size.

If no memory has been allocated, this will quietly return zero.

Definition at line 231 of file uvector_tlate.h.

7.361.2.2 size_t lookup (const data_t *x0*) const [inline]

Exhaustively look through the array for a particular value.

This can only fail if *all* of the entries in the array are not finite, in which case it calls set_err() and returns 0. The error handler is reset at the beginning of lookup().

Definition at line 273 of file uvector_tlate.h.

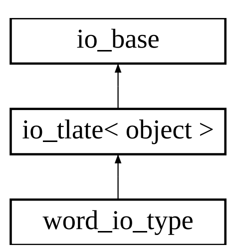
The documentation for this class was generated from the following file:

- uvector_tlate.h

7.362 word_io_type Class Reference

#include <collection.h>

Inheritance diagram for word_io_type::



7.362.1 Detailed Description

I/O object for words.

Definition at line 1831 of file collection.h.

Public Member Functions

- word_io_type (const char ∗t)
  Desc.
- int input (cinput ∗co, in_file_format ∗ins, std::string ∗dp)
  Desc.
- int output (coutput ∗co, out_file_format ∗outs, std::string ∗dp)
  Desc.
- int addw (collection &co, std::string name, std::string w, bool overwrt=true)
  Add a string to a collection.
- std::string getw (collection &co, std::string tname)
  Get a word from a collection.
- int get_def (collection &co, std::string tname, std::string &op, std::string def="")
  Get a word from a collection.
- const char ∗ type ()
  Desc.

The documentation for this class was generated from the following file:

- collection.h

8 File Documentation

8.1 array.h File Reference

8.1.1 Detailed Description

Various array classes.

This file contains classes and functions for operating with C-style 1- or 2-dimensional arrays and pointers to double. 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 or umatrix_row (see usage proposed in src/ode/ode_it_-solve_ts.cpp)
- array_row

For an example of the usage of the array allocation classes, see the Multidimensional solver example .

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.

Todo

Ensure that array_row works, either here or in src/ode/ode_it_solve_ts.cpp

Definition in file array.h.

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_ieee_utils.h>
#include <gsl/gsl_sort.h>
```

Data Structures

- class array_alloc
  A simple class to provide an `allocate()` *function for arrays.*
- class array_2d_alloc
  A simple class to provide an `allocate()` *function for 2-dimensional arrays.*
- class pointer_alloc
  A simple class to provide an `allocate()` *function for pointers.*
- class pointer_2d_alloc
  A simple class to provide an `allocate()` *function for pointers.*
- class array_reverse
  A simple class which reverses the order of an array.
- class array_const_reverse
  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_column
  Column of a 2d array.
- class array_2d_row
  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.
- class array_row
  Extract a row of a C-style 2d-array.

Functions

- template<class type>
  type ** new_2d_array (size_t nr, size_t nc)
  Create a new C-style 2-dimensional array.
- template<class type>
  int delete_2d_array (type **t, size_t nr)
  Create a new C-style 2-dimensional array.

8.2 cblas_base.h File Reference

8.2.1 Detailed Description

O2scl basic linear algebra function templates.

Todo

Finish dgemm()

Idea for future

Convert to size_t and add float and complex versions

Definition in file cblas_base.h.

Namespaces

- namespace o2scl_cblas

Enumerations

- enum O2CBLAS_ORDER { **O2cblasRowMajor** = 101, **O2cblasColMajor** = 102 }
  Matrix order, either column-major or row-major.
- enum O2CBLAS_TRANSPOSE { **O2cblasNoTrans** = 111, **O2cblasTrans** = 112, **O2cblasConjTrans** = 113 }
  Transpose operations.
- enum O2CBLAS_UPLO { **O2cblasUpper** = 121, **O2cblasLower** = 122 }
  Upper- or lower-triangular.
- enum O2CBLAS_DIAG { **O2cblasNonUnit** = 131, **O2cblasUnit** = 132 }
  Unit or generic diagonal.
- enum O2CBLAS_SIDE { **O2cblasLeft** = 141, **O2cblasRight** = 142 }
  Left or right sided operation.

Functions

- template<class mat_t, class vec_t>
  int dgemm (const enum O2CBLAS_ORDER Order, const enum O2CBLAS_TRANSPOSE TransA, const enum O2CBLAS_-TRANSPOSE TransB, const int M, const int N, const int K, const double alpha, const mat_t &A, const mat_t &B, const double beta, mat_t &C)
  Compute $y = \alpha \mathrm{op}(A) x + \beta y$.

Standard BLAS functions

- template<class vec_t, class vec2_t>
  void daxpy (const int N, const double alpha, const vec_t &X, vec2_t &Y)

Compute $y = \alpha x + y$.
- template<class vec_t, class vec2_t>
double ddot (const int N, const vec_t &X, const vec2_t &Y)
Compute $r = x \cdot y$.
- template<class vec_t>
void dscal (const int N, const double alpha, vec_t &X)
Compute $x = \alpha x$.
- template<class vec_t>
double dnrm2 (const int N, const vec_t &X)
Compute the squared norm of the vector `X`.
- template<class mat_t, class vec_t>
int dgemv (const enum O2CBLAS_ORDER order, const enum O2CBLAS_TRANSPOSE TransA, const int M, const int 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>
int dtrsv (const enum O2CBLAS_ORDER order, const enum O2CBLAS_UPLO Uplo, const enum O2CBLAS_TRANSPOSE TransA, const enum O2CBLAS_DIAG Diag, const int M, const int N, const mat_t &A, vec_t &X)
Compute $x = \mathrm{op}(A)^{-1} x$.

Helper BLAS functions

- template<class vec_t, class vec2_t>
void daxpy_subvec (const int N, const double alpha, const vec_t &X, vec2_t &Y, const int ie)
Compute $x = \alpha x$ beginning with index `ie` *and ending with index* `N-1`.
- template<class vec_t, class vec2_t>
double ddot_subvec (const int N, const vec_t &X, const vec2_t &Y, const int ie)
Compute $r = x \cdot y$ beginning with index `ie` *and ending with index* `N-1`.
- template<class vec_t>
void dscal_subvec (const int N, const double alpha, vec_t &X, const int ie)
Compute $x = \alpha x$ beginning with index `ie` *and ending with index* `N-1`.
- template<class vec_t>
double dnrm2_subvec (const int N, const vec_t &X, const int ie)
Compute the squared norm of the vector `X` *beginning with index* `ie` *and ending with index* `N-1`.
- template<class mat_t>
double dnrm2_subcol (const mat_t &M, const size_t ir, const size_t ic, const size_t N)
Compute the squared norm of the last `N` *rows of a column of a matrix.*
- template<class mat_t>
void dscal_subcol (mat_t &A, const size_t ir, const size_t ic, const size_t n, const double alpha)
Compute $x = \alpha x$.
- template<class mat_t, class vec_t>
void daxpy_hv_sub (const int N, const double alpha, const mat_t &X, vec_t &Y, const int ie)
Compute $x = \alpha x$ for householder_hv_sub().
- template<class mat_t, class vec_t>
double ddot_hv_sub (const int N, const mat_t &X, const vec_t &Y, const int ie)
Compute $r = x \cdot y$ for householder_hv_sub().

8.3 columnify.h File Reference

8.3.1 Detailed Description

Functions to create output in columns.

Definition in file columnify.h.

```
#include <iostream>
#include <string>
#include <vector>
#include <o2scl/misc.h>
#include <o2scl/array.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 naive matrix output.
- 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.
- template<class mat_t>
  int matrix_out (std::ostream &os, mat_t &A, size_t nrows, size_t ncols)
  A operator for simple 2-d array output.

8.3.2 Function Documentation

8.3.2.1 int matrix_out (std::ostream & *os*, mat_t & *A*, size_t *nrows*, size_t *ncols*) `[inline]`

A operator for simple 2-d array output.

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.

Todo

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 298 of file columnify.h.

8.3.2.2 int matrix_out_paren (std::ostream & *os*, mat_t & *A*, size_t *nrows*, size_t *ncols*) `[inline]`

A operator for naive matrix output.

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 227 of file columnify.h.

8.4 cx_arith.h File Reference

8.4.1 Detailed Description

Complex arithmetic.

Todo

Define operators with assignment for complex + double

Todo

Ensure all the trig functions are tested

Definition in file cx_arith.h.

```
#include <iostream>
#include <complex>
#include <cmath>
#include <gsl/gsl_math.h>
#include <gsl/gsl_complex.h>
#include <gsl/gsl_complex_math.h>
```

Namespaces

- namespace o2scl_arith

Functions

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)

8.5 err_hnd.h File Reference

8.5.1 Detailed Description

File for definitions for err_class.

Definition in file err_hnd.h.

```
#include <iostream>
#include <string>
#include <gsl/gsl_errno.h>
```

Data Structures

- class err_class
  The error handler.

Defines

- #define set_err(d, n) o2scl::set_err_fn(d,__FILE__,__LINE__,n);
  Set an error.
- #define set_err2(d, d2, n)
  Set an error.
- #define set_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 set_err2_ret(d, d2, n)
  Set an error and return the error value.
- #define add_err(d, n) o2scl::add_err_fn(d,__FILE__,__LINE__,n);
  Set an error and add the information from the last error.
- #define add_err2(d, d2, n)
  Set an error and add the information from the last error.
- #define add_err_ret(d, n) do { o2scl::add_err_fn(d,__FILE__,__LINE__,n); return n; } while(0)
  Set an error, add the information from the last error, and return the error value.
- #define add_err2_ret(d, d2, n)
  Set an error, add the information from the last error, and return the error value.
- #define err_print(ev)
  Print out error information.
- #define cerr_print(ev)
  Print out error information to `cerr`*, do nothing occured.*
- #define err_assert(ev)
  A version of `assert`*, i.e. exit if the error value is non-zero and do nothing otherwise.*
- #define 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_nobase = 33,
gsl_notfound = 34, gsl_memtype = 35, gsl_efilenotfound = 36, gsl_index = 37 }
The error definitions from GSL.

Functions

- void set_err_fn (const char ∗desc, const char ∗file, int line, int errnum)
  Set an error.
- void add_err_fn (const char ∗desc, const char ∗file, int line, int errnum)
  Set an error and add the information from the last error.
- void error_update (int &ret, int err)
  Update an error value `err` *with the value in* `ret`*.*

Variables

- err_class ∗ err_hnd
  The global error handler pointer.
- err_class def_err_hnd
  The default error handler.

8.5.2 Define Documentation

8.5.2.1 #define add_err2(d, d2, n)

Value:

```
o2scl::add_err_fn((std::string(d)+d2).c_str(), \
                                              __FILE__,__LINE__,n);
```

Set an error and add the information from the last error.

Definition at line 261 of file err_hnd.h.

8.5.2.2 #define add_err2_ret(d, d2, n)

Value:

```
do { o2scl::add_err_fn((std::string(d)+d2).c_str(),                    \
                         __FILE__,__LINE__,n); return n; } while (0)
```

Set an error, add the information from the last error, and return the error value.

Definition at line 273 of file err_hnd.h.

8.5.2.3 #define cerr_print(ev)

Value:

```
do { if (ev!=0) std::cerr << ev << " " << err_hnd->get_str() << std::endl; \
  } while (0)
```

Print out error information to `cerr`, do nothing occured.

Definition at line 305 of file err_hnd.h.

8.5.2.4 #define err_assert(ev)

A version of `assert`, i.e. exit if the error value is non-zero and do nothing otherwise.

Idea for future

Make this consistent with assert() using NDEBUG?

Definition at line 314 of file err_hnd.h.

8.5.2.5 #define err_print(ev)

Value:

```
do { if (ev!=0) std::cout << ev << " " << err_hnd->get_str() << std::endl; \
    else std::cout << "No error occured." << std::endl; } while (0)
```

Print out error information.

Definition at line 298 of file err_hnd.h.

8.5.2.6 #define set_err2(d, d2, n)

Value:

```
o2scl::set_err_fn((std::string(d)+d2).c_str(), \
                                          __FILE__,__LINE__,n);
```

Set an error.

Definition at line 241 of file err_hnd.h.

8.5.2.7 #define set_err2_ret(d, d2, n)

Value:

```
do { o2scl::set_err_fn((std::string(d)+d2).c_str(),                     \
                         __FILE__,__LINE__,n); return n; } while (0)
```

Set an error and return the error value.

Definition at line 251 of file err_hnd.h.

8.5.3 Enumeration Type Documentation

8.5.3.1 anonymous enum

The error definitions from GSL.

Enumerator:

gsl_success Success.

gsl_failure Failure.

gsl_continue iteration has not converged

gsl_edom input domain error, e.g sqrt(-1)

gsl_erange output range error, e.g. exp(1e100)

gsl_efault invalid pointer

gsl_einval invalid argument supplied by user

gsl_efailed generic failure

gsl_efactor factorization failed

gsl_esanity sanity check failed - shouldn't happen

gsl_enomem malloc failed

gsl_ebadfunc problem with user-supplied function

gsl_erunaway iterative process is out of control

gsl_emaxiter exceeded max number of iterations

gsl_ezerodiv tried to divide by zero

gsl_ebadtol user specified an invalid tolerance

gsl_etol failed to reach the specified tolerance

gsl_eundrflw underflow

gsl_eovrflw overflow

gsl_eloss loss of accuracy

gsl_eround failed because of roundoff error

gsl_ebadlen matrix, vector lengths are not conformant

gsl_enotsqr matrix not square

gsl_esing apparent singularity detected

gsl_ediverge integral or series is divergent

gsl_eunsup requested feature is not supported by the hardware

gsl_eunimpl requested feature not (yet) implemented

gsl_ecache cache limit exceeded

gsl_etable table limit exceeded

gsl_enoprog iteration is not making progress toward solution

gsl_enoprogj jacobian evaluations are not improving the solution

gsl_etolf cannot reach the specified tolerance in f

gsl_etolx cannot reach the specified tolerance in x

gsl_etolg cannot reach the specified tolerance in gradient

gsl_eof end of file

gsl_nobase a blank method in a base class has been called

gsl_notfound Generic "not found" result.

gsl_memtype Incorrect type for memory object.

gsl_efilenotfound File not found.

gsl_index Invalid index for array or matrix.

Definition at line 42 of file err_hnd.h.

8.6 givens.h File Reference

8.6.1 Detailed Description

File for Givens rotations.

Definition in file givens.h.

```
#include <o2scl/err_hnd.h>
#include <o2scl/permutation.h>
#include <o2scl/cblas.h>
#include <o2scl/vec_arith.h>
#include <o2scl/givens_base.h>
```

Namespaces

- namespace o2scl_linalg
- namespace o2scl_linalg_paren

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

- void create_givens (const double a, const double b, double &c, double &s)
  Desc.
- void **create_givens** (const double a, const double b, double &c, double &s)

8.7 givens_base.h File Reference

8.7.1 Detailed Description

File for Givens rotations.

Definition in file givens_base.h.

Namespaces

- namespace o2scl_linalg

Functions

- 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)
  Desc.
- 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)
  Desc.

- template<class vec_t>
  void apply_givens_vec (vec_t &v, size_t i, size_t j, double c, double s)
  Desc.

8.8 hh_base.h File Reference

8.8.1 Detailed Description

File for householder solver.

Definition in file hh_base.h.

```
#include <o2scl/err_hnd.h>
#include <o2scl/cblas.h>
#include <o2scl/permutation.h>
#include <o2scl/vec_arith.h>
```

Namespaces

- namespace o2scl_linalg

Functions

- template<class mat_t, class vec_t>
  int HH_solve (size_t n, mat_t &A, const vec_t &b, vec_t &x)
  Desc.
- template<class mat_t, class vec_t>
  int HH_svx (size_t N, size_t M, mat_t &A, vec_t &x)
  Desc.

8.9 householder_base.h File Reference

8.9.1 Detailed Description

File for Householder transformations.

Definition in file householder_base.h.

```
#include <o2scl/err_hnd.h>
#include <o2scl/cblas.h>
#include <o2scl/permutation.h>
#include <o2scl/vec_arith.h>
```

Namespaces

- namespace o2scl_linalg

Functions

- 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 the last `n-1` elements of `v`.

- template<class mat_t>
  double householder_transform_subcol (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` rows of a column of a matrix.
- template<class vec_t, class mat_t>
  int householder_hm (const size_t M, const size_t N, double tau, const vec_t &v, mat_t &A)
  Apply a householder transformation v,tau to matrix m.
- template<class mat_t>
  int 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 `v`, `tau` to submatrix of m.
- template<class vec_t>
  int householder_hv (const size_t N, double tau, const vec_t &v, vec_t &w)
  Apply a householder transformation `v` to vector `w`.
- template<class mat_t, class vec_t>
  int householder_hv_sub (const mat_t &M, vec_t &w, double tau, const size_t ie, const size_t N)
  Apply a householder transformation `v` to vector `w`.
- template<class mat1_t, class mat2_t>
  int 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().

8.10 lib_settings.h File Reference

8.10.1 Detailed Description

File for definitions for lib_settings_class.

Definition in file lib_settings.h.

```
#include <iostream>
#include <string>
```

Namespaces

- namespace o2scl

Data Structures

- class lib_settings_class
  A class to manage global library settings.

Variables

- lib_settings_class lib_settings
  The global library settings object.

8.11 lu_base.h File Reference

8.11.1 Detailed Description

File for LU decomposition and associated solver.

Definition in file lu_base.h.

Namespaces

- namespace o2scl_linalg

Functions

- 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 vec_t>
  int LU_solve (const size_t N, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_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 vec_t>
  int LU_refine (const size_t N, const mat_t &A, const mat_t &LU, const o2scl::permutation &p, const vec_t &b, vec_t &x, vec_t &residual)
  Refine the solution of a linear system.
- template<class mat_t, class mat_col_t>
  int LU_invert (const size_t N, const mat_t &LU, const o2scl::permutation &p, mat_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.

8.12 minimize.h File Reference

8.12.1 Detailed Description

One-dimensional minimization routines.

Definition in file minimize.h.

```
#include <o2scl/err_hnd.h>
```

Data Structures

- class minimize
  One-dimensional minimization [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`.

8.12.2 Function Documentation

8.12.2.1 double constraint (double *x*, double *center*, double *width*, double *height*) `[inline]`

Constrain `x` to be within `width` of the value given by `center`.

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 278 of file minimize.h.

8.12.2.2 double cont_constraint (double *x*, double *center*, double *width*, double *height*, double *tightness* = `40.0`, double *exp_arg_limit* = `50.0`) `[inline]`

Constrain `x` to be within `width` of the value given by `center`.

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 319 of file minimize.h.

8.12.2.3 double cont_lower_bound (double *x*, double *center*, double *width*, double *height*, double *tightness* = `40.0`, double *exp_arg_limit* = `50.0`) `[inline]`

Constrain `x` to be greater than the value given by `center`.

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 382 of file minimize.h.

8.12.2.4 double lower_bound (double *x*, double *center*, double *width*, double *height*) [inline]

Constrain `x` to be greater than the value given by `center`.

Defining $c = \texttt{center}$, $w = \texttt{width}$, $h = \texttt{height}$, 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 347 of file minimize.h.

8.13 misc.h File Reference

8.13.1 Detailed Description

Miscellaneous functions.

Definition in file misc.h.

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <o2scl/err_hnd.h>
#include <o2scl/lib_settings.h>
#include <gsl/gsl_ieee_utils.h>
```

Data Structures

- struct string_comp
  Naive string comparison.
- class gen_test_number
  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.
- void screenify (const std::string ∗in_cols, int nin, std::string ∗&outc, int &nout, int max_size=80)
  Reformat the columns for output of width `size`.
- template<class string_arr_t>
  int **screenify2** (size_t nin, const string_arr_t &in_cols, std::vector< std::string > &out_cols, size_t max_size=80)
- int count_words (std::string str)
  Count the number of words in the string `str`.
- std::string binary_to_hex (std::string s)
  Take a string of binary quads and compress them to hexadecimal digits.

8.13.2 Function Documentation

8.13.2.1 std::string binary_to_hex (std::string *s*)

Take a string of binary quads and compress them to hexadecimal digits.

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.

8.13.2.2 int count_words (std::string *str*)

Count the number of words in the 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.

8.13.2.3 double fermi_function (double *E*, double *mu*, double *T*, double *limit* = `40.0`)

Calculate a Fermi-Dirac distribution function safely.

$[1 + \exp(E/T - \mu/T)]^{-1}$

This calculates a Fermi-Dirac distribution function guaranteeing that numbers larger than $\exp(\mathrm{limit})$ and smaller than $\exp(-\mathrm{limit})$ will be avoided. The default value of `limit` 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.

8.13.2.4 void screenify (const std::string $*$ *in_cols*, int *nin*, std::string $*$& *outc*, int & *nout*, int *max_size* = `80`)

Reformat the columns for output of width `size`.

Given a string array `in_cols` of size `nin`, screenify() reformats the array into columns creating a new string array `outc` with size `nout`.

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
```

The string array given in outc must be deleted with delete[] after usage.

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`.

Todo

Convert to the new version "screenify2"

8.14 omatrix_cx_tlate.h File Reference

8.14.1 Detailed Description

File for definitions of complex matrices.

Definition in file omatrix_cx_tlate.h.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#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>
```

Data Structures

- class omatrix_cx_view_tlate
  A matrix view of double-precision numbers.
- class omatrix_cx_tlate
  A matrix of double-precision numbers.
- class omatrix_cx_row_tlate
  Create a vector from a row of a matrix.
- class omatrix_cx_const_row_tlate
  Create a vector from a row of a matrix.
- class omatrix_cx_col_tlate
  Create a vector from a column of a matrix.
- class omatrix_cx_const_col_tlate
  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

8.15 omatrix_tlate.h File Reference

8.15.1 Detailed Description

File for definitions of matrices.

Definition in file omatrix_tlate.h.

```
#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>
```

Data Structures

- class omatrix_view_tlate
  A matrix view of double-precision numbers.
- class omatrix_tlate
  A matrix of double-precision numbers.
- class omatrix_array_tlate
  Create a matrix from an array.
- class omatrix_row_tlate
  Create a vector from a row of a matrix.
- class omatrix_const_row_tlate
  Create a const vector from a row of a matrix.
- class omatrix_col_tlate
  Create a vector from a column of a matrix.
- class omatrix_const_col_tlate
  Create a const vector from a column of a matrix.
- class omatrix_diag_tlate
  Create a vector from the main diagonal.
- class omatrix_alloc
  A simple class to provide an `allocate()` *function for omatrix.*
- class ofmatrix
  A matrix where the memory allocation is performed in the constructor.

Typedefs

- typedef omatrix_tlate< double, gsl_matrix, gsl_vector, gsl_block > omatrix
  omatrix typedef
- typedef omatrix_view_tlate< double, gsl_matrix, gsl_block > omatrix_view
  omatrix_view typedef
- typedef omatrix_row_tlate< double, gsl_matrix, gsl_vector, gsl_block > omatrix_row
  omatrix_row typedef
- typedef omatrix_col_tlate< double, gsl_matrix, gsl_vector, gsl_block > omatrix_col
  omatrix_col typedef

- typedef omatrix_const_row_tlate< double, gsl_matrix, gsl_vector, gsl_block > omatrix_const_row
  omatrix_const_row typedef
- typedef omatrix_const_col_tlate< double, gsl_matrix, gsl_vector, gsl_block > omatrix_const_col
  omatrix_const_col typedef
- typedef omatrix_diag_tlate< double, gsl_matrix, gsl_vector, gsl_block > omatrix_diag
  omatrix_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_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_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_view_tlate< data_t, parent_t, block_t > &v)
  A operator for output of omatrix objects.

8.15.2 Function Documentation

8.15.2.1 std::ostream& operator<< (std::ostream & *os*, const omatrix_view_tlate< data_t, parent_t, block_t > & *v*) [inline]

A operator for output of omatrix objects.

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 919 of file omatrix_tlate.h.

8.16 ovector_cx_tlate.h File Reference

8.16.1 Detailed Description

File for definitions of complex vectors.

Definition in file ovector_cx_tlate.h.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <complex>
#include <o2scl/err_hnd.h>
#include <o2scl/ovector_tlate.h>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_complex.h>
```

Data Structures

- class ovector_cx_view_tlate
  A vector view of double-precision numbers.
- class ovector_cx_tlate
  A vector of double-precision numbers.
- class ovector_cx_array_tlate
  Create a vector from an array.
- class ovector_cx_array_stride_tlate
  Create a vector from an array with a stride.
- class ovector_cx_subvector_tlate
  Create a vector from a subvector of another.
- class ovector_cx_const_array_tlate
  Create a vector from an array.
- class ovector_cx_const_array_stride_tlate
  Create a vector from an array_stride.
- class ovector_cx_const_subvector_tlate
  Create a vector from a subvector of another.
- class ovector_cx_real_tlate
  Create a real vector from the real parts of a complex vector.
- class ovector_cx_imag_tlate
  Create a imaginary vector from the imaginary parts of a complex vector.
- class ofvector_cx
  A 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

- 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.
- 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.

8.16.2 Function Documentation

8.16.2.1 std::ostream& operator<< (std::ostream & *os*, const ovector_cx_view_tlate< data_t, vparent_t, block_t, complex_t > & *v*) `[inline]`

A operator for naive vector output.

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 1094 of file ovector_cx_tlate.h.

8.17 ovector_tlate.h File Reference

8.17.1 Detailed Description

File for definitions of vectors.

Definition in file ovector_tlate.h.

```
#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/uvector_tlate.h>
#include <o2scl/array.h>
#include <o2scl/vector.h>
```

Data Structures

- class ovector_view_tlate
  A vector view with finite stride.
- class ovector_tlate
  A vector with finite stride.
- class ovector_array_tlate
  Create a vector from an array.
- class ovector_array_stride_tlate
  Create a vector from an array with a stride.
- class ovector_subvector_tlate
  Create a vector from a subvector of another.
- class ovector_const_array_tlate
  Create a const vector from an array.
- class ovector_const_array_stride_tlate
  Create a const vector from an array with a stride.
- class ovector_const_subvector_tlate
  Create a const vector from a subvector of another vector.
- class ovector_reverse_tlate
  Reversed view of a vector.
- class ovector_const_reverse_tlate
  Reversed view of a vector.
- class ovector_subvector_reverse_tlate
  Reversed view of a subvector.
- class ovector_const_subvector_reverse_tlate
  Reversed view of a const subvector.
- 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
  A vector where the memory allocation is performed in the constructor.

Typedefs

- typedef ovector_tlate< double, gsl_vector, gsl_block > ovector
  ovector typedef
- typedef ovector_view_tlate< double, gsl_vector, gsl_block > ovector_view

ovector_view typedef

- typedef ovector_array_tlate< double, gsl_vector, gsl_block > ovector_array
  ovector_array typedef
- typedef ovector_array_stride_tlate< double, gsl_vector, gsl_block > ovector_array_stride
  ovector_array_stride typedef
- typedef ovector_subvector_tlate< double, gsl_vector, gsl_block > ovector_subvector
  ovector_subvector typedef
- typedef ovector_const_array_tlate< double, gsl_vector, gsl_block > ovector_const_array
  ovector_const_array typedef
- typedef ovector_const_array_stride_tlate< double, gsl_vector, gsl_block > ovector_const_array_stride
  ovector_const_array_stride typedef
- typedef ovector_const_subvector_tlate< double, gsl_vector, gsl_block > ovector_const_subvector
  ovector_const_subvector typedef
- typedef ovector_reverse_tlate< double, gsl_vector, gsl_block > ovector_reverse
  ovector_reverse typedef
- typedef ovector_const_reverse_tlate< double, gsl_vector, gsl_block > ovector_const_reverse
  ovector_const_reverse typedef
- typedef ovector_subvector_reverse_tlate< double, gsl_vector, gsl_block > ovector_subvector_reverse
  ovector_subvector_reverse typedef
- typedef ovector_const_subvector_reverse_tlate< double, gsl_vector, gsl_block > ovector_const_subvector_reverse
  ovector_const_subvector_reverse typedef
- typedef ovector_tlate< int, gsl_vector_int, gsl_block_int > ovector_int
  ovector_int typedef
- typedef ovector_view_tlate< int, gsl_vector_int, gsl_block_int > ovector_int_view
  ovector_int_view 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
- 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

Functions

- template<class data_t, class vparent_t, class block_t>
  std::ostream & operator<< (std::ostream &os, const ovector_view_tlate< data_t, vparent_t, block_t > &v)
  A operator for naive vector output.

8.17.2 Function Documentation

8.17.2.1 std::ostream& operator<< (std::ostream & *os*, const ovector_view_tlate< data_t, vparent_t, block_t > & *v*) `[inline]`

A operator for naive vector output.

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 1988 of file ovector_tlate.h.

8.18 permutation.h File Reference

8.18.1 Detailed Description

File containing permutation class and associated functions.

Definition in file permutation.h.

```
#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>
```

Data Structures

- class permutation
  A permutation.

Functions

- std::ostream & operator<< (std::ostream &os, const permutation &p)
  Output operator for permutations.

8.18.2 Function Documentation

8.18.2.1 std::ostream& operator<< (std::ostream & *os*, const permutation & *p*)

Output operator for permutations.

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.

8.19 poly.h File Reference

8.19.1 Detailed Description

Classes for solving polynomials.

Warning:

> We should 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
> One has to be careful about using e.g. pow(a,1.0/3.0) for complex a since if Re(a)<0 and Im(a)==0 then the function returns NaN.

Definition in file poly.h.

```
#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>
```

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.

8.20 qr_base.h File Reference

8.20.1 Detailed Description

File for QR decomposition and associated solver.

Definition in file qr_base.h.

```
#include <o2scl/householder.h>
#include <o2scl/givens.h>
```

Namespaces

- namespace o2scl_linalg

Functions

- template<class mat_t, class vec_t>
  int 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>
  int QR_solve (size_t N, const mat_t &QR, const vec_t &tau, const vec_t &b, vec_t &x)
  Solve the system A x = b using the QR factorization.
- template<class mat_t, class vec_t>
  int QR_svx (size_t M, size_t N, const mat_t &QR, const vec_t &tau, vec_t &x)
  Solve the system A x = b in place using the QR factorization.
- template<class mat_t, class vec_t>
  int QR_QTvec (const size_t M, const size_t N, const mat_t &QR, const vec_t &tau, vec_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>
  int 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>
  int 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,.

8.21 string_conv.h File Reference

8.21.1 Detailed Description

Various string conversion functions.

Definition in file string_conv.h.

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/lib_settings.h>
#include <gsl/gsl_ieee_utils.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 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)
  Convert a string to an integer.
- bool stob (std::string s)
  Convert a string to a boolean value.
- double stod (std::string s)
  Convert a string to a double.
- std::string double_to_latex (double x, int sigfigs=5, int ex_min=-2, int ex_max=3, bool pad_zeros=false)
  Convert a double to a Latex-like string.
- std::string double_to_html (double x, int sigfigs=5, int ex_min=-2, int ex_max=3)
  Convert a double to a HTML-like string.
- std::string double_to_ieee_string (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.*

8.21.2 Function Documentation

8.21.2.1 std::string btos (bool *b*)

Convert a boolean value to a string.

This returns `"1"` for true and `"0"` for false.

8.21.2.2 std::string double_to_html (double *x*, int *sigfigs* = 5, int *ex_min* = -2, int *ex_max* = 3)

Convert a double to a HTML-like string.

This uses `×` and to convert a double to a string for use on the web.

The parameter `sigfigs` is the number of significant figures to give in the string. The parameters `ex_min` and `ex_max` represent the base-10 logarithm of the smallest and largest numbers to be represented without exponential notation. The number zero is always converted to "0". Superscripts are implemented using (can't use greater than size in documentation)

Note that this function does not warn the user if the number of significant figures requested is larger than the machine precision.

Todo

Add a pad_zeros parameter as in double_to_latex().

8.21.2.3 std::string double_to_ieee_string (double * *x*)

Convert a double to a string containing IEEE representation.

Modeled after the GSL function `gsl_ieee_fprintf_double()`, but converts to a `string` instead of a `FILE *`.

8.21.2.4 std::string double_to_latex (double *x*, int *sigfigs* = 5, int *ex_min* = -2, int *ex_max* = 3, bool *pad_zeros* = false)

Convert a double to a Latex-like string.

The parameter `sigfigs` is the number of significant figures to give in the string. The parameters `ex_min` and `ex_max` represent the base-10 logarithm of the smallest and largest numbers to be represented without the string

```
$\times 10^{\mathrm{ex}}$
```

where `ex` is the relevant exponent. The number zero is always converted to "0".

If the parameter `pad_zeros` is true, this function adds zeros to the right side of the mantissa to guarantee that the requested number of significant digits is given.

Note that this function does not warn the user if the number of significant figures requested is larger than the machine precision.

Todo

Consider converting to a class so the user can modify the strings used in giving the exponents, etc.

8.21.2.5 bool has_minus_sign (double * *x*)

Find out if the number pointed to by `x` has a minus sign.

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".

8.21.2.6 std::string ptos (void * *p*)

Convert a pointer to a string.

This uses an `ostringstream` to convert a pointer to a string and is architecture-dependent.

8.21.2.7 size_t size_of_exponent (double *x*)

Returns the number of characters required to display the exponent of `x` in scientific mode.

This usually returns 2 or 3, depending on whether or not the absolute magnitude of the exponent is greater than 100.

8.21.2.8 bool stob (std::string *s*)

Convert a string to a boolean value.

This returns true 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.

This function never fails (it just returns false for an empty string).

8.21.2.9 double stod (std::string *s*)

Convert a string to a double.

If this function fails it will call set_err() and return zero.

8.21.2.10 int stoi (std::string *s*)

Convert a string to an integer.

If this function fails it will call set_err() and return zero.

8.22 tensor.h File Reference

8.22.1 Detailed Description

File for definitions of tensors.

Definition in file tensor.h.

```
#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>
```

Data Structures

- class tensor
  Tensor class with arbitrary dimensions.
- class tensor_grid
  Tensor class with arbitrary dimensions.
- class tensor1
  Rank 1 tensor.
- 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.

8.23 tridiag_base.h File Reference

8.23.1 Detailed Description

File for solving tridiagonal systems.

Definition in file tridiag_base.h.

Namespaces

- namespace o2scl

Functions

- template<class vec_t, class vec2_t>
  int solve_tridiag_sym (const vec_t &diag, const vec2_t &offdiag, const vec_t &b, vec_t &x, size_t N)
  Solve a symmetric tridiagonal linear system.
- template<class vec_t, class vec2_t>
  int solve_tridiag_nonsym (const vec_t &diag, const vec2_t &abovediag, const vec2_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)
  Solve an asymmetric tridiagonal linear system.
- template<class vec_t>
  int solve_cyc_tridiag_sym (const vec_t &diag, const vec_t &offdiag, const vec_t &b, vec_t &x, size_t N)
  Solve a symmetric cyclic tridiagonal linear system.
- template<class vec_t>
  int solve_cyc_tridiag_nonsym (const vec_t &diag, const vec_t &abovediag, const vec_t &belowdiag, const vec_t &rhs, vec_t &x, size_t N)
  Solve an asymmetric cyclic tridiagonal linear system.

8.24 umatrix_cx_tlate.h File Reference

8.24.1 Detailed Description

File for definitions of matrices.

Definition in file umatrix_cx_tlate.h.

```
#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>
```

Data Structures

- class umatrix_cx_view_tlate
  A matrix view of complex numbers.
- class umatrix_cx_tlate
  A matrix of double-precision numbers.
- class umatrix_cx_row_tlate
  Create a vector from a row of a matrix.
- class umatrix_cx_const_row_tlate
  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
  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.

8.24.2 Function Documentation

8.24.2.1 std::ostream& operator<< (std::ostream & *os*, const umatrix_cx_view_tlate< data_t, complex_t > & *v*) `[inline]`

A operator for naive matrix output.

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.

Todo

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Definition at line 666 of file umatrix_cx_tlate.h.

8.25 umatrix_tlate.h File Reference

8.25.1 Detailed Description

File for definitions of matrices.

Definition in file umatrix_tlate.h.

```
#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>
```

Data Structures

- class umatrix_view_tlate
  A matrix view of double-precision numbers.
- class umatrix_tlate
  A matrix of double-precision numbers.
- class umatrix_row_tlate
  Create a vector from a row of a matrix.
- class umatrix_const_row_tlate
  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
  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_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_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_view_tlate< data_t > &v)
  A operator for naive matrix output.

8.25.2 Function Documentation

8.25.2.1 std::ostream& operator<< (std::ostream & *os*, const umatrix_view_tlate< data_t > & *v*) `[inline]`

A operator for naive matrix output.

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.

Todo

This assumes that scientific mode is on and showpos is off. It'd be nice to fix this.

Definition at line 662 of file umatrix_tlate.h.

8.26 uvector_cx_tlate.h File Reference

8.26.1 Detailed Description

File for definitions of complex unit-stride vectors.

Definition in file uvector_cx_tlate.h.

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_vector.h>
```

Data Structures

- class uvector_cx_view_tlate
  A vector view of complex numbers with unit stride.
- class uvector_cx_tlate
  A vector of double-precision numbers with unit stride.
- class uvector_cx_array_tlate
  Create a vector from an array.
- class uvector_cx_subvector_tlate
  Create a vector from a subvector of another.
- class uvector_cx_const_array_tlate
  Create a vector from an array.
- class uvector_cx_const_subvector_tlate
  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.

8.26.2 Function Documentation

8.26.2.1 std::ostream& operator<< (std::ostream & *os*, const uvector_cx_view_tlate< data_t, complex_t > & *v*) [inline]

A operator for naive vector output.

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 670 of file uvector_cx_tlate.h.

8.27 uvector_tlate.h File Reference

8.27.1 Detailed Description

File for definitions of unit-stride vectors.

Definition in file uvector_tlate.h.

```
#include <iostream>
```

```
#include <cstdlib>
#include <string>
#include <fstream>
#include <sstream>
#include <vector>
#include <o2scl/err_hnd.h>
#include <o2scl/string_conv.h>
#include <o2scl/array.h>
#include <o2scl/vector.h>
#include <gsl/gsl_vector.h>
```

Data Structures

- class uvector_view_tlate
  A vector view with unit stride.
- class uvector_tlate
  A vector with unit stride.
- class uvector_array_tlate
  Create a vector from an array.
- class uvector_subvector_tlate
  Create a vector from a subvector of another.
- class uvector_const_array_tlate
  Create a vector from an const array.
- class uvector_const_subvector_tlate
  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 ufvector
  A vector with unit-stride where the memory allocation is performed in the constructor.

Typedefs

- typedef uvector_tlate< data_t, size_t > uvector
  uvector typedef
- typedef uvector_view_tlate< data_t, size_t > uvector_view
  uvector_view typedef
- typedef uvector_array_tlate< data_t, size_t > uvector_array
  uvector_array typedef
- typedef uvector_subvector_tlate< data_t, size_t > uvector_subvector
  uvector_subvector typedef
- typedef uvector_const_array_tlate< data_t, size_t > uvector_const_array
  uvector_const_array typedef
- typedef uvector_const_subvector_tlate< data_t, size_t > 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_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

Functions

- template<class data_t>
  std::ostream & operator<< (std::ostream &os, const uvector_view_tlate< data_t > &v)
  A operator for naive vector output.

8.27.2 Function Documentation

8.27.2.1 std::ostream& operator<< (std::ostream & *os*, const uvector_view_tlate< data_t > & *v*) [inline]

A operator for naive vector output.

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 812 of file uvector_tlate.h.

8.28 vec_arith.h File Reference

8.28.1 Detailed Description

Vector and matrix arithmetic.

Todo

Properly document the operators defined as macros

Idea for future

Define operators for complex vector $*$ real matrix

Idea for future

These should be replaced by the BLAS routines where possible?

Definition in file vec_arith.h.

```
#include <iostream>
#include <complex>
#include <o2scl/cx_arith.h>
#include <o2scl/ovector_tlate.h>
#include <o2scl/omatrix_tlate.h>
#include <o2scl/uvector_tlate.h>
#include <o2scl/umatrix_tlate.h>
#include <o2scl/ovector_cx_tlate.h>
#include <o2scl/omatrix_cx_tlate.h>
```

```
#include <o2scl/uvector_cx_tlate.h>
#include <o2scl/umatrix_cx_tlate.h>
```

Namespaces

- namespace o2scl_arith

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.

8.28.2 Define Documentation

8.28.2.1 #define O2SCL_OP_CMAT_CVEC_MULT(vec1, vec2, mat)

Value:

```
vec1 operator*   \
    (const mat &m, const vec2 &x);
```

The header macro for complex matrix-vector (right) multiplication.

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.

By default, the following operators are defined:

```
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);
```

Definition at line 221 of file vec_arith.h.

8.28.2.2 #define O2SCL_OP_CX_DOT_PROD(dtype, vec1, vec2)

Value:

```
dtype dot        \
    (const vec1 &x, const vec2 &y);
```

The header macro for complex vector scalar (dot) product.

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 338 of file vec_arith.h.

8.28.2.3 #define O2SCL_OP_DOT_PROD(dtype, vec1, vec2)

Value:

```
dtype dot        \
    (const vec1 &x, const vec2 &y);
```

The header macro for vector scalar (dot) product.

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 307 of file vec_arith.h.

8.28.2.4 #define O2SCL_OP_MAT_VEC_MULT(vec1, vec2, mat)

Value:

```
vec1 operator*  \
    (const mat &m, const vec2 &x);
```

The header macro for matrix-vector (right) multiplication.

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.

By default, the following operators are defined:

```
ovector operator*(omatrix_view &x, ovector_view &y);
ovector operator*(omatrix_view &x, uvector_view &y);
ovector operator*(umatrix_view &x, ovector_view &y);
uvector operator*(umatrix_view &x, uvector_view &y);
```

Definition at line 179 of file vec_arith.h.

8.28.2.5 #define O2SCL_OP_SCA_VEC_MULT(dtype, vecv, vec)

Value:

```
vec operator*   \
    (const dtype &x, const vecv &y);
```

The header macro for scalar-vector multiplication.

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 372 of file vec_arith.h.

8.28.2.6 #define O2SCL_OP_TRANS_MULT(vec1, vec2, mat)

Value:

```
vec1 trans_mult \
    (const vec2 &x, const mat &m);
```

The header macro for the `trans_mult` form of vector $*$ matrix.

Definition at line 277 of file vec_arith.h.

8.28.2.7 #define O2SCL_OP_VEC_MAT_MULT(vec1, vec2, mat)

Value:

```
vec1 operator*  \
    (const vec2 &x, const mat &m);
```

The header macro for vector-matrix (left) multiplication.

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 256 of file vec_arith.h.

8.28.2.8 #define O2SCL_OP_VEC_SCA_MULT(dtype, vecv, vec)

Value:

```
vec operator*   \
    (const vecv &x, const dtype &y);
```

The header macro for vector-scalar multiplication.

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 400 of file vec_arith.h.

8.28.2.9 #define O2SCL_OP_VEC_VEC_ADD(vec1, vec2, vec3)

Value:

```
vec1 operator+  \
    (const vec2 &x, const vec3 &y);
```

The header macro for vector-vector addition.

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.

By default, the following operators are defined:

```
ovector operator+(ovector_view &x, ovector_view &y);
ovector operator+(ovector_view &x, uvector_view &y);
ovector operator+(uvector_view &x, ovector_view &y);
uvector operator+(uvector_view &x, uvector_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);
```

Definition at line 75 of file vec_arith.h.

8.28.2.10 #define O2SCL_OP_VEC_VEC_EQUAL(vec1, vec2)

Value:

```
bool operator== \
      (const vec1 &x, const vec2 &y);
```

The header macro for vector==vector.

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 690 of file vec_arith.h.

8.28.2.11 #define O2SCL_OP_VEC_VEC_NEQUAL(vec1, vec2)

Value:

```
bool operator!= \
    (const vec1 &x, const vec2 &y);
```

The header macro for vector!=vector.

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 772 of file vec_arith.h.

8.28.2.12 #define O2SCL_OP_VEC_VEC_PRO(vec1, vec2, vec3)

Value:

```
vec1 pair_prod  \
    (const vec2 &x, const vec3 &y);
```

The header macro for pairwise vector $*$ vector (where either vector can be real or complex).

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 429 of file vec_arith.h.

8.28.2.13 #define O2SCL_OP_VEC_VEC_SUB(vec1, vec2, vec3)

Value:

```
vec1 operator-  \
    (const vec2 &x, const vec3 &y);
```

The header macro for vector-vector subtraction.

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.

By default, the following operators are defined:

```
ovector operator-(ovector_view &x, ovector_view &y);
ovector operator-(ovector_view &x, uvector_view &y);
ovector operator-(uvector_view &x, ovector_view &y);
uvector operator-(uvector_view &x, uvector_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);
```

Definition at line 128 of file vec_arith.h.

8.28.2.14 #define O2SCL_OPSRC_CMAT_CVEC_MULT(vec1, vec2, mat)

The source code macro for complex matrix $*$ complex vector.

This define macro generates the function definition. See the function declaration O2SCL_OP_CMAT_CVEC_MULT

Definition at line 529 of file vec_arith.h.

8.28.2.15 #define O2SCL_OPSRC_CX_DOT_PROD(dtype, vec1, vec2)

The source code macro for a complex vector dot product.

This define macro generates the function definition. See the function declaration O2SCL_OP_CX_DOT_PROD

Definition at line 614 of file vec_arith.h.

8.28.2.16 #define O2SCL_OPSRC_DOT_PROD(dtype, vec1, vec2)

The source code macro for a vector dot product.

This define macro generates the function definition. See the function declaration O2SCL_OP_DOT_PROD

Definition at line 597 of file vec_arith.h.

8.28.2.17 #define O2SCL_OPSRC_MAT_VEC_MULT(vec1, vec2, mat)

The source code macro for matrix $*$ vector.

This define macro generates the function definition. See the function declaration O2SCL_OP_MAT_VEC_MULT

Definition at line 508 of file vec_arith.h.

8.28.2.18 #define O2SCL_OPSRC_SCA_VEC_MULT(dtype, vecv, vec)

The source code macro for vector=scalar*vector.

This define macro generates the function definition. See the function declaration O2SCL_OP_SCA_VEC_MULT

Definition at line 631 of file vec_arith.h.

8.28.2.19 #define O2SCL_OPSRC_TRANS_MULT(vec1, vec2, mat)

The source code macro for the `trans_mult` form of vector * matrix.

This define macro generates the function definition. See the function declaration O2SCL_OP_TRANS_MULT

Definition at line 576 of file vec_arith.h.

8.28.2.20 #define O2SCL_OPSRC_VEC_MAT_MULT(vec1, vec2, mat)

The source code macro for the operator form of vector * matrix.

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_MAT_MULT

Definition at line 554 of file vec_arith.h.

8.28.2.21 #define O2SCL_OPSRC_VEC_SCA_MULT(dtype, vecv, vec)

The source code macro for vector=vector*scalar.

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_SCA_MULT

Definition at line 647 of file vec_arith.h.

8.28.2.22 #define O2SCL_OPSRC_VEC_VEC_ADD(vec1, vec2, vec3)

The source code macro for vector-vector addition.

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_ADD

Definition at line 474 of file vec_arith.h.

8.28.2.23 #define O2SCL_OPSRC_VEC_VEC_EQUAL(vec1, vec2)

The source code macro vector==vector.

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 746 of file vec_arith.h.

8.28.2.24 #define O2SCL_OPSRC_VEC_VEC_NEQUAL(vec1, vec2)

The source code macro vector!=vector.

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 828 of file vec_arith.h.

8.28.2.25 #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).

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_PRO

Definition at line 664 of file vec_arith.h.

8.28.2.26 #define O2SCL_OPSRC_VEC_VEC_SUB(vec1, vec2, vec3)

The source code macro for vector-vector subtraction.

This define macro generates the function definition. See the function declaration O2SCL_OP_VEC_VEC_SUB

Definition at line 491 of file vec_arith.h.

8.29 vec_stats.h File Reference

8.29.1 Detailed Description

File containing statistics template functions.

Definition in file vec_stats.h.

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_ieee_utils.h>
#include <gsl/gsl_sort.h>
```

Functions

- template<class vec_t>
  double vector_mean (const size_t n, vec_t &data)
  Compute the mean of the first `n` *elements of a vector.*
- template<class vec_t>
  double vector_variance_fmean (const size_t n, vec_t &data, double mean)
  Variance.
- template<class vec_t>
  double vector_stddev_fmean (const size_t n, vec_t &data, double mean)
  Standard deviation.
- template<class vec_t>
  double vector_variance (const size_t n, vec_t &data, double mean)
  Compute the variance of the first `n` *elements of a vector given the mean* `mean`.
- template<class vec_t>
  double vector_variance (const size_t n, vec_t &data)
  Variance.
- template<class vec_t>
  double vector_stddev (const size_t n, vec_t &data)
  Standard deviation.
- template<class vec_t>
  double vector_stddev (const size_t n, vec_t &data, double mean)

Standard deviation.

- template<class vec_t>
  double vector_absdev (const size_t n, vec_t &data, double mean)
  Absolute deviation from the mean.
- template<class vec_t>
  double vector_absdev (const size_t n, vec_t &data)
  Absolute deviation from the mean.
- template<class vec_t>
  double vector_skew (const size_t n, vec_t &data, double mean, double stddev)
  Skewness.
- template<class vec_t>
  double vector_skew (const size_t n, vec_t &data)
  Skewness.
- template<class vec_t>
  double vector_kurtosis (const size_t n, vec_t &data, double mean, double stddev)
  Kurtosis.
- template<class vec_t>
  double vector_kurtosis (const size_t n, vec_t &data)
  Kurtosis.
- template<class vec_t>
  double vector_lag1_autocorr (const size_t n, vec_t &data, double mean)
  Lag1 autocorrelation.
- template<class vec_t>
  double vector_lag1_autocorr (const size_t n, vec_t &data)
  Lag1 autocorrelation.
- template<class vec_t>
  double vector_covariance (const size_t n, vec_t &data1, vec_t &data2, double mean1, double mean2)
  Covariance.
- template<class vec_t>
  double vector_covariance (const size_t n, vec_t &data1, vec_t &data2)
  Covariance.
- template<class vec_t>
  double vector_correlation (const size_t n, vec_t &data1, vec_t &data2)
  Pearson's correlation.
- template<class vec_t>
  double vector_pvariance (const size_t n1, vec_t &data1, const size_t n2, vec_t &data2)
  Pooled variance.
- template<class vec_t>
  double vector_quantile_sorted (const size_t n, vec_t &data, const double f)
  Quantile.
- template<class vec_t>
  double vector_median_sorted (const size_t n, vec_t &data)
  Quantile.

8.29.2 Function Documentation

8.29.2.1 double vector_mean (const size_t *n*, vec_t & *data*) `[inline]`

Compute the mean of the first `n` elements of a vector.

If `n` is zero, this will set `avg` to zero and return gsl_success.

Definition at line 51 of file vec_stats.h.

8.29.2.2 double vector_variance (const size_t *n*, vec_t & *data*, double *mean*) `[inline]`

Compute the variance of the first `n` elements of a vector given the mean `mean`.

If n is zero, this will set avg to zero and return gsl_success.

Definition at line 87 of file vec_stats.h.

8.30 vector.h File Reference

8.30.1 Detailed Description

File for generic vector functions.

For overloaded 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 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.

Definition in file vector.h.

```
#include <iostream>
#include <cmath>
#include <string>
#include <fstream>
#include <sstream>
#include <o2scl/err_hnd.h>
#include <gsl/gsl_ieee_utils.h>
#include <gsl/gsl_sort.h>
```

Functions

- template<class vec_t, class vec2_t>
  void vector_copy (size_t N, vec_t &src, vec2_t &dest)
  Naive vector copy.
- template<class mat_t, class mat2_t>
  void matrix_copy (size_t M, size_t N, mat_t &src, mat2_t &dest)
  Naive 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.
- template<class vec_t>
  int vector_out (std::ostream &os, size_t n, vec_t &v, bool endline=false)
  Output a vector to a stream.
- template<class data_t, class vec_t>
  void sort_downheap (vec_t &data, const size_t N, size_t k)
  Provide a downheap() function for vector_sort().
- template<class data_t, class vec_t>
  int vector_sort (const size_t n, vec_t &data)
  Sort a vector.
- template<class data_t, class vec_t>
  int vector_rotate (const 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>
  double vector_max (const size_t n, vec_t &data)

Compute the maximum of the first `n` *elements of a vector.*

- template<class vec_t>
  double vector_min (const size_t n, vec_t &data)
  Compute the minimum of the first `n` *elements of a vector.*
- template<class vec_t>
  int vector_minmax (const size_t n, vec_t &data, double &min, double &max)
  Compute the minimum and maximum of the first `n` *elements of a vector.*
- template<class vec_t>
  size_t vector_max_index (const size_t n, vec_t &data, double &max)
  Compute the maximum of the first `n` *elements of a vector.*
- template<class vec_t>
  int vector_min_index (const size_t n, vec_t &data, double &min)
  Compute the minimum of the first `n` *elements of a vector.*
- template<class vec_t>
  int vector_minmax_index (const size_t n, vec_t &data, double &min, size_t &ix, double &max, size_t &ix2)
  Compute the minimum and maximum of the first `n` *elements of a vector.*
- template<class vec_t>
  double vector_sum (const size_t n, vec_t &data)
  Compute the sum of the first `n` *elements of a vector.*
- template<class data_t, class vec_t>
  int vector_reverse (const size_t n, vec_t &data)
  Reverse a vector.

8.30.2 Function Documentation

8.30.2.1 void matrix_copy (size_t *M*, size_t *N*, mat_t & *src*, mat2_t & *dest*) `[inline]`

Naive matrix copy.

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);
```

Definition at line 85 of file vector.h.

8.30.2.2 void matrix_cx_copy_gsl (size_t *M*, size_t *N*, mat_t & *src*, mat2_t & *dest*) `[inline]`

GSL complex matrix copy.

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 114 of file vector.h.

8.30.2.3 void vector_copy (size_t *N*, vec_t & *src*, vec2_t & *dest*) `[inline]`

Naive vector copy.

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);
```

Definition at line 67 of file vector.h.

8.30.2.4 void vector_cx_copy_gsl (size_t *N*, vec_t & *src*, vec2_t & *dest*) [inline]

GSL complex vector copy.

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 100 of file vector.h.

8.30.2.5 int vector_out (std::ostream & *os*, size_t *n*, vec_t & *v*, bool *endline* = false) [inline]

Output a vector to a stream.

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.

Definition at line 134 of file vector.h.

8.30.2.6 int vector_rotate (const size_t *n*, vec_t & *data*, size_t *k*) [inline]

"Rotate" a vector so that the kth element is now the beginning

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`

Definition at line 223 of file vector.h.

8.30.2.7 int vector_sort (const size_t *n*, vec_t & *data*) [inline]

Sort a vector.

This is a generic sorting template function. It will work for any types `data_t` and `vec_t` for which

- `data_t` has an `operator=`
- `data_t` has a less than operator to compare elements
- `vec_t::operator[]` returns a reference to an object of type `data_t`

In particular, it will work with ovector, uvector, ovector_int, uvector_int (and other related O2scl 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, std::string[3]>(3,list);
```

Definition at line 187 of file vector.h.

8.30.2.8 double vector_sum (const size_t *n*, vec_t & *data*) [inline]

Compute the sum of the first `n` elements of a vector.

If `n` is zero, this will set `avg` to zero and return gsl_success.

Definition at line 357 of file vector.h.

Index