O_2scl_eos - Equation of State Sub-Library for O_2scl

Version 0.910

Contents

1 O2scl Equation of State Sub-Library User's Guide

1.1 Feature Overview

O_2scl_eos is sub-library for working with particles and nuclei designed to work with O_2scl_part and O_2scl . It includes

- Several classes for computing the hadronic and quark equations of state of matter near and above the nuclear saturation density
- Classes to solve the TOV equations of neutron star structure
- One class to perform a rudimentary computation of the structure of closed-shell nuclei in the Hartree approximation.

1.2 Quick Reference to User's Guide

2 Equations of State of Hadronic Matter

The hadronic equations of state are all inherited from hadronic_eos: schematic_eos, skyrme_eos, rmf_eos, apr_eos, and gen_potential_eos.

hadronic_eos includes several methods that can be used to calculate the saturation properties of nuclear matter. These methods are sometimes overloaded in descendants when exact formulas are available.

There is also a set of classes to modify the quartic term of the symmetry energy: rmf4_eos, apr4_eos, skyrme4_eos, and mdi4_eos all based on sym4_eos_base which can be used in sym4_eos.

3 Equations of State of Quark Matter

The equations of state of quark matter are all inherited from quark_eos: bag_eos is a simple bag model, nambujl_eos is the Nambu--Jona-Lasinio model. The CFL and 2SC phases are described by cfl_njl_eos, and cfl6_eos adds the color-superconducting 't Hooft interaction.

4 Solution of the Tolman-Oppenheimer-Volkov equations

The class tov_solve provide a solution to the Tolman-Oppenheimer-Volkov (TOV) equations given an equation of state. This is particularly useful for static neutron star structure: given any equation of state one can calculate the mass vs. radius curve and the properties of any star of a given mass. An adaptive integration is employed and calculates the gravitational mass, the baryonic mass (if the baryon density is supplied), and the gravitational potential. The remaining columns is the equation of state are also interpolated into the solution, e.g. if a chemical potential is given, then the radial dependence of the chemical potential for a 1.4 solar mass star can be automatically computed. The equation of state may be specified in arbitrary units so long as an appropriate conversion factor is supplied. An equation of state for low densities (baryon density < 0.08 fm^{-3}) is provided and can be automatically appended to the user-defined equation of state.

This is still experimental.

5 Cold Neutron Stars

The class cold_nstar computes the structure of zero-temperature spherically-symmetric neutron stars. It uses tov_solve to compute the structure, given a hadronic equation of state (of type hadronic_eos). It also computes the adiabatic index, the speed of sound, and determines the possibility of the direct Urca process as a function of density or radius.

This class is still experimental.

6 Nuclear structure in the Hartree approximation

See class rmf_nucleus .

7 Example Source Code

7.1 Example list

- Cold neutron star example

7.2 Cold neutron star example

```
/* Example: ex_cold_nstar.cpp
   -------------------------------------------------------------------
   This example solves the TOV equations using class cold_nstar using a
   relativistic mean-field EOS from class rmf_eos.
*/

#include <o2scl/cold_nstar.h>
#include <o2scl/rmf_eos.h>
#include <o2scl/hdf_file.h>
#include <o2scl/hdf_io.h>
#include <o2scl/test_mgr.h>
```

```
using namespace std;
using namespace o2scl;
using namespace o2scl_hdf;

// For hc_mev_fm
using namespace o2scl_const;

// A simple function to load the NL3 model
int load_nl3(rmf_eos &rmf);

int main(void) {

  cout.setf(ios::scientific);

  test_mgr t;
  t.set_output_level(1);

  cold_nstar nst;

  // Initialize EOS
  rmf_eos rmf;
  load_nl3(rmf);

  rmf.saturation();
  cout << "Saturation density: " << rmf.n0 << endl;
  cout << "Binding energy: " << rmf.eoa*hc_mev_fm << endl;
  cout << "Effective mass: " << rmf.msom << endl;
  cout << "Symmetry energy: " << rmf.esym*hc_mev_fm << endl;
  cout << "Compressibility: " << rmf.comp*hc_mev_fm << endl;

  // Compute EOS, include muons
  nst.include_muons=true;
  nst.set_eos(rmf);
  nst.calc_eos();
  o2_shared_ptr<table_units>::type te=nst.get_eos_results();

  // Compute mass vs. radius
  nst.calc_nstar();
  o2_shared_ptr<table_units>::type tr=nst.get_tov_results();
  cout << "Maximum mass: " << tr->max("gm") << endl;
  cout << "Radius of maximum mass star: "
       << tr->get("r",tr->lookup("gm",tr->max("gm"))) << endl;
  cout << "Central baryon density of maximum mass star: ";
  cout << tr->get("nb",tr->lookup("gm",tr->max("gm"))) << endl;

  // Output EOS and TOV results to files
  hdf_file hf;
  hf.open("ex_cold_nstar_eos.o2");
  hdf_output(hf,*te,"eos");
  hf.close();
  hf.open("ex_cold_nstar_tov.o2");
  hdf_output(hf,*tr,"tov");
  hf.close();

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

8 Bibliography

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

Akmal98: Akmal, Pandharipande, and Ravenhall, Phys. Rev. C **58**, 1805 (1998).

Bartel79: J. Bartel, P. Quentin, M. Brack, C. Guet, and Håkansson, Nucl. Phys. A **386** (1982) 79.

Baym71: G. Baym, C. Pethick, and P. Sutherland, Astrophys. J. **170** (1971) 299.

Beiner75: M. Beiner, H. Flocard, Nguyen van Giai, and P. Quentin, Nucl. Phys. A **238** (1975) 29.

Bernard88: V. Bernard, R. L. Jaffe, and U.-G. Meissner, Nucl. Phys. B **308** (1988) 753.

Bombaci01: I. Bombaci, "Equation of State for Dense Isospin Asymmetric Nuclear Matter for Astrophysical Applications Equation of State for Isospin-Asymmetric Nuclear Matter and Neutron Star Properties", in "Isospin physics in heavy-ion collisions at interme-

diate energies" ed. by B-A. Li and W. U. Schröder (2001) Nova Science, New York.

Brack85: M. Brack, C. Guet, and H.-B. Håkansson, Phys. Rep. **123** (1985) 275.

Buballa99: M. Buballa and M. Oertel, Phys. Lett. B **457** (1999) 261.

Buballa04: M. Buballa, Phys. Rep. **407** (2005) 205-376.

Chabanat95: E. Chabanat, Ph. D. Thesis 1995

Chabanat97: E. Chabanat, J. Meyer, P. Bonche, R. Schaeffer, and P. Haensel, Nucl. Phys. A **627** (1997) 710.

Danielewicz08: P. Danielewicz, J. Lee, arXiv.org:0807.3743.

Das03: C.B. Das, S. Das Gupta, C. Gale, and B.-A. Li, Phys. Rev. C **67** (2003) 034611.

Dobaczewski94: J. Dobaczewski, H. Flocard, and J. Treiner, Nucl. Phys. A **422** (1984) 103.

Dutta86: A.K.Dutta, J.-P.Arcoragi, J.M.Pearson, R.Behrman, F.Tondeur, Nucl. Phys. A **458** (1986) 77.

Friedrich86: J. Friedrich, and P.-G. Reinhard, Phys. Rev. C **33** (1986) 335.

Gaitanos04: T. Gaitanos, M. Di Toro, S. Typel, V. Barana, C. Fuchs, V. Greco, and d H. H. Wolter, Nucl. Phys. A **732** (2004) 24.

Gale87: C. Gale, G. Bertsch, and S. Das Gupta, Phys. Rev. C **35** (1986) 1666.

Hatsuda94: T. Hatsuda and T. Kunihiro, Phys. Rep. **247** (1994) 221.

Heide94: E.K. Heide, S. Rudaz, and P.J. Ellis Nucl. Phys. A **571** (1994) 713.

Hempel10: M. Hempel and J. Schaffner-Bielich, Nucl. Phys. A **837** (2010) 210.

Hempel11: M. Hempel, T. Fischer, J. Schaffner-Bielich, and M. Liebendorfer, arxiv.org/1108.0848.

Horowitz81: C.J. Horowitz and B.D. Serot, Nucl. Phys. A 368 (1981) 503.

Horowitz01: C. J. Horowitz and J. Piekarewicz, Phys. Rev. Lett. **86** (2001) 5647.

Kubis97: S. Kubis and M. Kutschera, Phys. Lett. B **399** (1997) 191.

Lattimer85: J. M. Lattimer, C. J. Pethick, D. G. Ravenhall, and D.Q. Lamb, Nucl. Phys. A, **432** (1985) 646.

Lattimer91: J.M. Lattimer and F.D. Swesty, Nucl. Phys. A, **535** (1991) 331.

Lattimer01: J. M. Lattimer and M. Prakash, Astrophys. J. **550** (2001) 426.

Margueron02: J. Margueron, J. Navarro, and N. V. Giai, Phys. Rev. C **66** (2002) 014303.

Muller96: H. Muller and B. D. Serot, Nucl. Phys. A **606** (1996), 508.

OConnor10: E. O'Connor and C.D. Ott, Class. Quant. Grav., 27 (2010) 114103.

Onsi94: M. Onsi, H. Przysiezniak, J.M. Pearson, Phys. Rev. C **50** (1994) 460.

Pethick95: C. J. Pethick, D. G. Ravenhall, and C. P. Lorenz, Nucl. Phys. A **584** (1995) 675.

Piekarewicz09: J. Piekarewicz and M. Centelles, Phys. Rev. C 79 (2009) 054311.

Prakash87: M. Prakash, T. L. Ainsworth, J. P. Blaizot, and H. Wolter, in Windsurfing the Fermi Sea, Volume II edited by T. T. S. Kuo and J. Speth, Elsevier 1987, pg. 357.

Prakash88: M. Prakash, T. L. Ainsworth, and J. M. Lattimer, Phys. Rev. Lett. **61** (1988) 2518.

Prakash94: M. Prakash P.J. Ellis, E.K. Heide and S. Rudaz, Nucl. Phys. A **575** (1994) 583.

Prakash97: Madappa Prakash, I. Bombaci, Manju Prakash, P. J. Ellis, J. M. Lattimer, and R. Knorren, Phys. Rep. **280** (1997) 1.

Ravenhall83: D.G. Ravenhall, C.J. Pethick, J.R. Wilson, Phys. Rev. Lett., **50** (1983) 2066.

Reinhard95: P.-G. Reinhard and H. Flocard, Nucl. Phys. A **584** (1995) 467.

Reinhard99: P.-G. Reinhard, D.J. Dean, W. Nazarewicz, J. Dobaczewski, J.A. Maruhn, M.R. Strayer, Phys. Rev. C **60**, (1999)

014316.

Shapiro83: S. L. Shapiro and S. A. Teukolsky, "Black Holes, White Dwarfs, and Neutron Stars: The Physics of Compact Objects", John Wiley and Sons, New York, 1983.

Shen98: H. Shen, H. Toki, K. Oyamatsu and K. Sumiyoshi, Nuclear Physics A **637** (1998) 435-450.

Shen98b: H. Shen, H. Toki, K. Oyamatsu and K. Sumiyoshi, Progress of Theoretical Physics, 100 (1998) 1013-1031.

Shen10a: G. Shen, C. J. Horowitz, and S. Teige, Phys. Rev. C 82, 015806 (2010).

Shen10b: G. Shen, C. J. Horowitz, and S. Teige, Phys. Rev. C 82, 045802 (2010).

Shen11: G. Shen, C. J. Horowitz, S. Teige, Phys. Rev. C 83, 035802 (2011).

Skyrme59: T. H. R. Skyrme, Nucl. Phys. **9** (1959) 615.

Steiner00: A. W. Steiner, M. Prakash, and J.M. Lattimer, Phys. Lett. B, **486** (2000) 239.

Steiner02: A. W. Steiner, S. Reddy, and M. Prakash, Phys. Rev. D, **66** (2002) 094007.

Steiner05: A. W. Steiner, Phys. Rev. D, **72** (2005) 054024.

Steiner05b: A.W. Steiner, M. Prakash, J.M. Lattimer, and P.J. Ellis, Phys. Rep. (2005).

Steiner06: A. W. Steiner, Phys. Rev. C, **74** (2006) 045808.

Steiner08: A. W. Steiner, Phys. Rev. C, **77** (2008) 035805.

Souza09: S. R. Souza, A. W. Steiner, W. G. Lynch, R. Donangelo, M. A Famiano, Astrophys. J, **707** (2009) 1495.

Tondeur84: F. Tondeur, M. Brack, M. Farine, and J.M. Pearson, Nucl. Phys. A **420** (1984) 297.

Typel99: S. Typel and H. H. Wolter, Nucl. Phys. A **656** (1999) 331.

VanGiai81: Nguyen van Giai and H. Sagawa, Phys. Lett. B **106** (1981) 379.

Zimanyi90: Zimanyi and Moszkowski, Phys. Rev. C **42** (1990) 1416.

9 Other Todos

Idea for Future Right now, the equation of state classes depend on the user to input the correct value of `non_interacting` for the particle inputs. This is not very graceful...

10 Equations of State for Core-Collapse Supernovae

There are several classes designed to provide a consistent interface to several supernova EOS tables. The abstract base class is gen_sn_eos. The child classes correspond to different EOS table formats:

- ls_eos - The Lattimer-Swesty EOS tables from Jim's webpage (Lattimer91)
- stos_eos - The H. Shen et al. EOS tables (Shen98 and Shen98b)
- sht_eos - The G. Shen et al. EOS tables (Shen10a, Shen10b, Shen11)
- hfsl_eos - The M. Hempel et al. EOS tables (Hempel10 and Hempel11)
- oo_eos - The Lattimer-Swesty and H. Shen et al. tables reformatted by O'Connor and Ott (OConnor10)

The O_2scl distribution does not contain the tables themselves, as they are quite large and most are freely available. O_2scl includes code which parses these tables and puts them in **tensor_grid3** objects for analysis by the user.

All of these classes are experimental.

11 Ideas for Future Development

Class apr_eos

There might be room to improve the testing of the finite temperature part a bit.

There is some repetition between calc_e() and calc_temp_e() that possibly could be removed.

Global apr_eos::parent_method

This function is probably unnecessary, as the syntax

Class bps_eos

Can the pressure be made to match more closely?

Convert to a hadronic_eos object and offer an associated interface?

Class cfl_njl_eos

This class internally mixes ovector, omatrix, gsl_vector and **gsl_matrix** objects in a confusing and non-optimal way. Fix this.

Allow user to change derivative object? This isn't possible right now because the stepsize parameter of the derivative object is used.

Class cold_nstar

Warn if the EOS becomes pure neutron matter.

Class ddc_eos

Implement the finite temperature EOS properly.

Class gen_potential_eos

Calculate the chemical potentials analytically

Class gen_sn_eos

Create a **table** object, possibly using **tensor_grid::vector_slice**.

Show how matrix_slice and vector_slice can be used with this object.

Could this be a child of hadronic_eos_temp and then directly used in cold_nstar()?

Add option to load and store a separate lepton/photon EOS

Add muons and/or pions

Class hadronic_eos

Could write a function to compute the "symmetry free energy" or the "symmetry entropy"

Global hadronic_eos::saturation ()

It would be great to provide numerical uncertainties in the saturation properties.

Class ldrop_mass_skin

Add translational energy?

Remove excluded volume correction and compute nuclear mass relative to the gas rather than relative to the vacuum.

In principle, Tc should be self-consistently determined from the EOS.

Does this work if the nucleus is "inside-out"?

Class nse_eos

Right now calc_density() needs a very good guess. This could be fixed, probably by solving for the log(mu/T) instead of mu.

Page Other Todos

Right now, the equation of state classes depend on the user to input the correct value of `non_interacting` for the particle inputs. This is not very graceful...

Class rmf_delta_eos

Finish the finite temperature EOS

Class rmf_eos

It might be nice to remove explicit reference to the meson masses in functions which only compute nuclear matter since they are unnecessary. This might, however, demand redefining some of the couplings.

- Fix calc_p() to be better at guessing
- The number of couplings is getting large, maybe new organization is required.
- Overload hadronic_eos::fcomp() with an exact version

Global rmf_eos::calc_e (fermion &ne, fermion &pr, thermo &lth)

Improve the operation of this function when the proton density is zero.

Global rmf_eos::calc_e_fields (fermion &ne, fermion &pr, thermo &lth, double &sig, double &ome, double &rho)

Improve the operation of this function when the proton density is zero.

Global rmf_eos::calc_eq_p (fermion &neu, fermion &p, double sig, double ome, double rho, double &f1, double &f2, double &f3, thermo &th)

Probably best to have f1, f2, and f3 scaled in some sensible way, i.e. scaled to the fields?

Class rmf_nucleus

Sort energy levels at the end by energy

Improve the numerical methods

Make the neutron and proton orbitals more configurable

Generalize to $m_n \neq m_p$.

Allow more freedom in the integrations

Consider converting everything to inverse fermis.

Convert to zero-indexed arrays

Warn when the level ordering is wrong, and unoccupied levels are lower energy than occupied levels

Class skyrme_eos

There is some code duplication between calc_e() and calc_temp_e() which could be simplified.

Class stos_eos

Add the T=0 and Ye=0 data to this class

Class tov_buchdahl_eos

Figure out what to do with the buchfun() function

12 Todo List

Global cfl_njl_eos::calc_eq_temp_p (quark &u, quark &d, quark &s, double &qq1, double &qq2, double &qq3, double &gap1, double &gap2, double &gap3, double mu3, double mu8, double &n3, double &n8, thermo &qb, double temper)

It surprises me that n3 is not -res[11]. Is there a sign error in the color densities?

Global cfl_njl_eos::gapped_eigenvalues (double m1, double m2, double lmom, double mu1, double mu2, double tdelta, double lam[4], double dldmu1[4], double dldmu2[4], double dldm1[4], double dldm2[4], double dldg[4])

In the code, the equal mass case seems to be commented out. Why?

Global ddc_eos::calc_eq_e (fermion &neu, fermion &p, double sig, double ome, double rho, double &f1, double &f2, double &f3, thermo &th)

Is the thermodynamic identity is satisfied even when the field equations are not solved? Check this.

Class gen_sn_eos

Ensure all chemical potentials are based on the same rest masses?

Allow logarithmic grids for any of nb, Ye, or T.

Class ldrop_mass_skin

This is based on LPRL, but it's a little different in Lattimer and Swesty. I should document what the difference is.

The testing could be updated.

Class ls_eos

There are still a few points for which the electron/photon EOS seems to be off.

Class rmf_eos

The functions fcomp_fields(), fkprime_fields(), and fesym_fields() are not quite correct if the neutron and proton masses are different. For this reason, they are currently unused by saturation().

- The fix_saturation() and calc_cr() functions use mnuc, and should be modified to allow different neutron and proton masses.
- Check the formulas in the "Background" section
- There are two calc_e() functions that solve. One is specially designed to work without a good initial guess. Possibly the other calc_e() function should be similarly designed?
- Make sure that this class properly handles particles for which inc_rest_mass is true/false
- The error handler is called sometimes when calc_e() is used to compute pure neutron matter. This should be fixed.
- Decide whether to throw an error at [Ref. 1].
- Put the err_nonconv system into calc_p(), calc_temp_e() and fix_saturation(), etc.

Global rmf_eos::check_naturalness (rmf_eos &re)

I may have ignored some signs in the above, which are unimportant for this application, but it would be good to fix them for posterity.

Global rmf_eos::fix_saturation (double guess_cs=4.0, double guess_cw=3.0, double guess_b=0.001, double guess_c=-0.001)

Fix this for zm_mode=true

- Ensure solver is more robust

Global rmf_eos::fkprime_fields (double sig, double ome, double nb, double &k, double &kprime)

Does this work? Fix fkprime_fields() if it does not.

Global rmf_eos::n_charge

Should use hadronic_eos::proton_frac instead?

Class rmf_nucleus

Better documentation

Convert energies() to use EOS and possibly replace sigma_rhs() and related functions by the associated field equation method of rmf_eos.

Document hw=3.923+23.265/cbrt(atot);

Global schematic_eos::set_a_from_mstar (double u_msom, double mnuc)

This was computed in schematic_sym.nb, which might be added to the documentation?

Class sht_eos

More documentation

Class skyrme_eos

Make sure that this class properly handles particles for which inc_rest_mass is true/false

- What about the spin-orbit units?
- Need to write a function that calculates saturation density?
- Remove use of mnuc in calparfun()?
- The compressibility could probably use some simplification
- Make sure the finite-temperature part is properly tested
- The testing code doesn't work if err_mode is 2, probably because of problems in load().

- Document load() file format.
- Update reference list.

Global skyrme_eos::calpar (double gt0=-10.0, double gt3=70.0, double galpha=0.2, double gt1=2.0, double gt2=-1.0)

Does this work for both 'a' and 'b' non-zero?

Compare to similar formulae from Margueron02

Global skyrme_eos::landau_neutron (double n0, double m, double &f0, double &g0, double &f1, double &g1)

This needs to be checked

Global skyrme_eos::landau_nuclear (double n0, double m, double &f0, double &g0, double &f0p, double &g0p, double &f1, double &g1, double &f1p, double &g1p)

This needs to be checked.

Class tov_eos

Fix read_table_file and maybe set_low_density_eos().

Class tov_interp_eos

Warn that the pressure in the low-density eos is not strictly increasing! (see at P=4.3e-10)

It might be useful to exit more gracefully when non-finite values are obtained in interpolation, analogous to the err_nonconv mechanism elsewhere.

Class tov_solve

baryon mass doesn't work for fixed() (This may be fixed. We should make sure it's tested.)

- Combine maxoutsize and kmax?
- Document column naming issues
- Document surface gravity and redshift
- Standardize `xmev_kg`, etc.
- Use **convert_units**?
- Double check that fixed() doesn't give a solution on the unstable branch
- Ensure that this class copies over the units of the user-specified columns to the table

13 Bug List

Class gen_potential_eos

The BGBD EOS doesn't work and the effective mass for the GBD EOS doesn't work

Class sym4_eos_base

Testing was disabled in HDF conversion. Fix this.

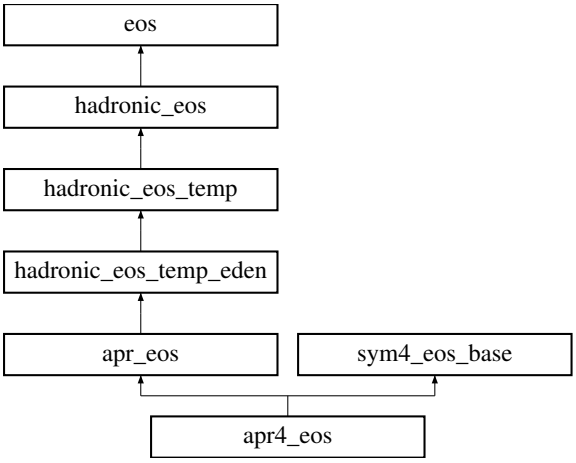
14 Data Structure Documentation

14.1 apr4_eos Class Reference

A version of apr_eos to separate potential and kinetic contributions.

`#include <sym4_eos.h>`

Inheritance diagram for apr4_eos:



14.1.1 Detailed Description

References:

Created for Steiner06.

Definition at line 125 of file sym4_eos.h.

Public Member Functions

- virtual int calc_e_sep (**fermion** &ne, **fermion** &pr, double &ed_kin, double &ed_pot, double &mu_n_kin, double &mu_p_kin, double &mu_n_pot, double &mu_p_pot)
  Compute the potential and kinetic parts separately.

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

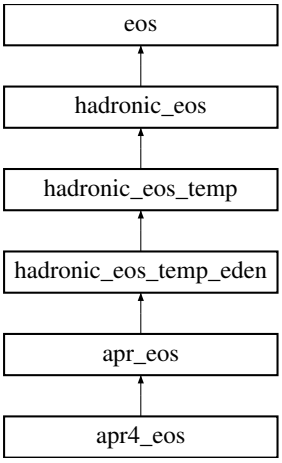
- sym4_eos.h

14.2 apr_eos Class Reference

EOS from Akmal, Pandharipande, and Ravenhall.

`#include <apr_eos.h>`

Inheritance diagram for apr_eos:



14.2.1 Detailed Description

The EOS of Akmal, Pandharipande, and Ravenhall, from Akmal98 (APR).

The Hamiltonian is:

$$\mathscr{H}_{APR} = \mathscr{H}_{kin} + \mathscr{H}_{pot}$$

$$\mathscr{H}_{kin} = \left(\frac{\hbar^2}{2m} + (p_3 + (1-x)\, p_5)\, n e^{-p_4 n} \right) \tau_n + \left(\frac{\hbar^2}{2m} + (p_3 + x p_5)\, n e^{-p_4 n} \right) \tau_p$$

$$\mathscr{H}_{pot} = g_1 \left(1 - (1-2x)^2 \right) + g_2 (1-2x)^2$$

The following are definitions for g_i in the low-density phase (LDP) or the high-density phase (HDP):

$$g_{1,LDP} = -n^2 \left(p_1 + p_2 n + p_6 n^2 + (p_{10} + p_{11} n)\, e^{-p_9^2 n^2} \right)$$

$$g_{2,LDP} = -n^2 \left(p_{12}/n + p_7 + p_8 n + p_{13} e^{-p_9^2 n^2} \right)$$

$$g_{1,HDP} = g_{1,LDP} - n^2 \left(p_{17} (n - p_{19}) + p_{21} (n - p_{19})^2 e^{p_{18}(n - p_{19})} \right)$$

$$g_{2,HDP} = g_{2,LDP} - n^2 \left(p_{15} (n - p_{20}) + p_{14} (n - p_{20})^2 e^{p_{16}(n - p_{20})} \right)$$

The chemical potentials include the rest mass energy and the energy density includes the rest mass energy density.

Note

APR seems to have been designed to be used with non-relativistic neutrons and protons with equal masses of 939 MeV. This gives a saturation density very close to 0.16.

The variables ν_n and ν_p contain the expressions $(-\mu_n + V_n)/T$ and $(-\mu_p + V_p)/T$ respectively, where V is the potential part of the single particle energy for particle i (i.e. the derivative of the Hamiltonian w.r.t. density while energy density held constant). Equivalently, ν_n is just $-k_{F_n}^2/2m^*$.

The selection between the LDP and HDP is controlled by pion. The default is to use the LDP at densities below 0.16 fm^{-3}, and for larger densities to just use whichever minimizes the energy.

The finite temperature approximations from Prakash97 are used in testing.

Note

Since this EOS uses the effective masses and chemical potentials in the fermion class, the values of **part::non_interacting** for neutrons and protons are set to false in many of the functions.
The parameter array is unit indexed, so that `par[0]` is unused.

Idea for Future There might be room to improve the testing of the finite temperature part a bit.

There is some repetition between calc_e() and calc_temp_e() that possibly could be removed.

Definition at line 123 of file apr_eos.h.

Public Member Functions

- apr_eos ()
  Create an EOS object with the default parameter set ($A18+UIX^*+\delta v$).
- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)
  Equation of state as a function of density.
- virtual int calc_temp_e (**fermion** &n, **fermion** &pr, double temper, **thermo** &th)
  Equation of state as a function of densities.
- double fcomp (double nb)
  Compute the compressibility.
- double fesym_diff (double nb)
  Calculate symmetry energy of matter as energy of neutron matter minus the energy of nuclear matter.
- void select (int model_index)
  Select model.
- int gradient_qij2 (double nn, double np, double &qnn, double &qnp, double &qpp, double &dqnndnn, double &dqnndnp, double &dqnpdnn, double &dqnpdnp, double &dqppdnn, double &dqppdnp)
  Calculate Q's for semi-infinite nuclear matter.
- double get_par (int n)
  Get the value of one of the parameters.
- int set_par (int n, double x)
  Set the value of one of the parameters.
- virtual const char * type ()
  Return string denoting type ("apr_eos")

Data Fields

- bool parent_method
  If true, use the methods from hadronic_eos for fcomp()

Protected Attributes

- double * par
  Storage for the parameters.
- int lp
  An integer to indicate which phase was used in calc_e()
- **nonrel_fermion** nrf
  Desc.
- int choice
  The variable indicating which parameter set is to be used.

Choice of phase

- static const int best = 0
  use LDP for densities less than 0.16 and for higher densities, use the phase which minimizes energy (default)
- static const int ldp = 1
  LDP (no pion condensation)
- static const int hdp = 2
  HDP (pion condensation)
- int pion
  Choice of phase (default best)
- int last_phase ()
  Return the phase of the most recent call to calc_e()

14.2.2 Member Function Documentation

14.2.2.1 double apr_eos::fcomp (double *nb*)

See general notes at hadronic_eos::fcomp(). This computes the compressibility (at fixed proton fraction = 0.5) exactly, unless parent_method is true in which case the derivative is taken numerically in hadronic_eos::fcomp().

14.2.2.2 double apr_eos::fesym_diff (double *nb*) [virtual]

This function returns the energy per baryon of neutron matter minus the energy per baryon of nuclear matter. This will deviate significantly from the results from fesym() only if the dependence of the symmetry energy on δ is not quadratic.

Reimplemented from hadronic_eos.

14.2.2.3 void apr_eos::select (int *model_index*)

Valid values for `model_index` are:

1 - A18+UIX*+deltav (preferred by Akmal, et. al. - this is the default)

2 - A18+UIX*

3 - A18+deltav

4 - A18

If any other integer is given, A18+UIX*+deltav is assumed.

14.2.2.4 int apr_eos::gradient_qij2 (double *nn*, double *np*, double & *qnn*, double & *qnp*, double & *qpp*, double & *dqnndnn*, double & *dqnndnp*, double & *dqnpdnn*, double & *dqnpdnp*, double & *dqppdnn*, double & *dqppdnp*)

For general discussion, see the documentation to hadronic_eos::qs().

For APR, we set $x_1 = x_2 = 0$ so that $Q_i = P_i/2$ and then

$$\begin{aligned} P_1 &= \left(\frac{1}{2}p_3 - p_5\right) e^{-p_4 n} \\ P_2 &= \left(\frac{1}{2}p_3 + p_5\right) e^{-p_4 n} \end{aligned}$$

This gives

$$\begin{aligned} Q_{nn} &= \frac{1}{4} e^{-p_4\rho} \left[-6p_5 - p_4(p_3 - 2p_5)(n_n + 2n_p)\right] \\ Q_{np} &= \frac{1}{8} e^{-p_4\rho} \left[4(p_3 - 4p_5) - 3p_4(p_3 - 2p_5)(n_n + n_p)\right] \\ Q_{pp} &= \frac{1}{4} e^{-p_4\rho} \left[-6p_5 - p_4(p_3 - 2p_5)(n_p + 2n_n)\right] \end{aligned}$$

See the Mathematica notebook

```
doc/o2scl/extras/apr_eos.nb
doc/o2scl/extras/apr_eos.ps
```

14.2.3 Field Documentation

14.2.3.1 bool apr_eos::parent_method

This can be set to true to check the difference in the compressibility wbetween the exact expressions and the numerical values from class hadronic_eos.

Idea for Future This function is probably unnecessary, as the syntax

```
apr_eos apr;
ccout << apr.hadronic_eos::fcomp(0.16) << endl;
```

works just as well.

Definition at line 282 of file apr_eos.h.

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

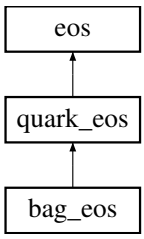
- apr_eos.h

14.3 bag_eos Class Reference

Simple bag model.

```
#include <bag_eos.h>
```

Inheritance diagram for bag_eos:



14.3.1 Detailed Description

An equation of state with $P = -B + P_{u,FG} + P_{d,FG} + P_{s,FG}$ where $P_{i,FG}$ is the Fermi gas contribution from particle i and B is a density- and temperature-independent bag constant.

The finite temperature functions run the zero temperature code if the temperature is less than or equal to 0.

Definition at line 45 of file bag_eos.h.

Public Member Functions

- virtual int calc_p (**quark** &u, **quark** &d, **quark** &s, **thermo** &th)
  Calculate equation of state as a function of chemical potentials.

- virtual int calc_e (**quark** &u, **quark** &d, **quark** &s, **thermo** &th)
  Calculate equation of state as a function of density.
- virtual int calc_temp_p (**quark** &u, **quark** &d, **quark** &s, double temper, **thermo** &th)
  Calculate equation of state as a function of the chemical potentials.
- virtual int calc_temp_e (**quark** &u, **quark** &d, **quark** &s, double temper, **thermo** &th)
  Calculate equation of state as a function of the densities.
- virtual const char * type ()
  Return string denoting type ("bag_eos")

Data Fields

- double bag_constant
  The bag constant in fm^{-4} *(default* $200/(\hbar c)$*).*

14.3.2 Member Function Documentation

14.3.2.1 virtual int bag_eos::calc_temp_p (quark & *u*, quark & *d*, quark & *s*, double *temper*, thermo & *th*) [virtual]

This function returns zero (success) unless the call to **quark::pair_mu()** fails.

Reimplemented from quark_eos.

14.3.2.2 virtual int bag_eos::calc_temp_e (quark & *u*, quark & *d*, quark & *s*, double *temper*, thermo & *th*) [virtual]

This function returns zero (success) unless the call to **quark::pair_density()** fails.

Reimplemented from quark_eos.

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

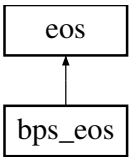
- bag_eos.h

14.4 bps_eos Class Reference

Baym-Pethick-Sutherland equation of state.

`#include <bps_eos.h>`

Inheritance diagram for bps_eos:



14.4.1 Detailed Description

This calculates the equation of state of electrons and nuclei using the approach of Baym71 (based on the discussion in Shapiro83) between about 8×10^6 g/cm^3 and 4.3×10^{11} g/cm^3. Below these densities, more complex Coulomb corrections need to be considered, and above these densities, neutron drip is important.

The default mass formula is semi-empirical

$$\begin{aligned} M(A,Z) &= (A-Z)m_n + Z(m_p+m_e) - 15.76A - 17.81A^{2/3} \\ &\quad -0.71Z^2/A^{1/3} - 94.8/A\,(A/2-Z)^2 + E_{\rm pair} \end{aligned}$$

where

$$E_{\rm pair} = \pm 39/A^{3/4}$$

if the nucleus is odd-odd (plus sign) or even-even (minus sign) and $E_{\rm pair}$ is zero for odd-even and even-odd nuclei. The nuclei are assumed not to contribute to the pressure. The electronic contribution to the pressure is assumed to be equal to the Fermi gas contribution plus a "lattice" contribution

$$\varepsilon_L = -1.444 Z^{2/3} e^2 n_e^{4/3}$$

This is Eq. 2.7.2 in Shapiro83. The rest mass energy of the nucleons is included in the energy density.

The original results from Baym71 are stored as a **table** in file `data/o2scl/bps.eos`. The testing code for this class compares the calculations to the table and matches to within .2 percent for the energy density and 9 percent for the pressure (for a fixed baryon number density).

Idea for Future Can the pressure be made to match more closely?

Convert to a hadronic_eos object and offer an associated interface?

Definition at line 81 of file bps_eos.h.

Public Member Functions

- virtual int calc_density (double barn, **thermo** &th, int &Z, int &A)
  Calculate the equation of state as a function of the baryon number density `barn`.
- virtual int calc_pressure (**thermo** &th, double &barn, int &Z, int &A)
  Calculate the equation of state as a function of the pressure.
- virtual double lattice_energy (int Z)
  The electron lattice energy.
- virtual const **fermion** & get_electron ()
  Get a pointer to the electron.
- virtual double mass_formula (int Z, int A)
  The mass formula.
- virtual const char * type ()
  Return string denoting type ("bps_eos")
- int set_mass_formula (**nuclear_mass** &nm)
  Set the nuclear mass formula to be used.
- int calc_density_fixedA (double barn, **thermo** &th, int &Z, int A)
  Compute the ground state assuming a fixed atomic number.

Data Fields

- **semi_empirical_mass** def_mass
  Default mass formula.
- **fermion** e
  The electron thermodynamics.

Protected Member Functions

- virtual int eq274 (size_t nv, const **ovector_base** &nx, **ovector_base** &ny, int &Zt)
  Solve Equation 2.7.4 for a given pressure.
- double gibbs (int Z, int A)
  The Gibbs free energy.
- double energy (double barn, int Z, int A)
  The energy density.

Protected Attributes

- fermion_zerot fzt
  Desc.
- **gsl_mroot_hybrids**< **mm_funct**<> > gs
  A solver to solve Eq. 2.7.4.
- **nuclear_mass** * nmp
  The nuclear mass formula.

14.4.2 Member Function Documentation

14.4.2.1 virtual int bps_eos::calc_density (double *barn*, thermo & *th*, int & *Z*, int & *A*) [virtual]

This calculates the equation of state as a function of the baryon number density in fm^{-3}, returning the representative nucleus with proton number `Z` and atomic number `A`. The pressure and energy density are returned in `th` in fm^{-4}.

14.4.2.2 virtual int bps_eos::calc_pressure (thermo & *th*, double & *barn*, int & *Z*, int & *A*) [virtual]

This calculates the equation of state as a function of the pressure, returning the representative nucleus with proton number `Z` and atomic number `A` and the baryon number density `barn` in fm^{-3}. The energy density is also returned in fm^{-4} in `th`.

14.4.2.3 virtual double bps_eos::mass_formula (int *Z*, int *A*) [virtual]

The nuclear mass without the contribution of the rest mass of the electrons. The electron rest mass energy is included in the electron thermodynamics elsewhere.

14.4.3 Field Documentation

14.4.3.1 fermion bps_eos::e

Note

The electron rest mass is included by default in the energy density and the chemical potential

Definition at line 145 of file bps_eos.h.

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

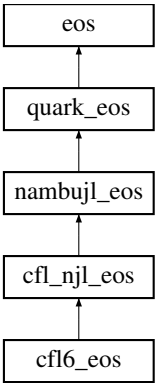
- bps_eos.h

14.5 cfl6_eos Class Reference

An EOS like cfl_njl_eos but with a color-superconducting 't Hooft interaction.

`#include <cfl6_eos.h>`

Inheritance diagram for cfl6_eos:



14.5.1 Detailed Description

Beginning with the Lagrangian:

$$\mathscr{L} = \mathscr{L}_{Dirac} + \mathscr{L}_{NJL} + \mathscr{L}_{tHooft} + \mathscr{L}_{SC} + \mathscr{L}_{SC6}$$

$$\mathscr{L}_{Dirac} = \bar{q}\left(i\partial - m - \mu\gamma^0\right)q$$

$$\mathscr{L}_{NJL} = G_S\sum_{a=0}^{8}\left[(\bar{q}\lambda^a q)^2 - \left(\bar{q}\lambda^a\gamma^5 q\right)^2\right]$$

$$\mathscr{L}_{tHooft} = G_D\left[\det{}_f\bar{q}\left(1-\gamma^5\right)q + \det{}_f\bar{q}\left(1+\gamma^5\right)q\right]$$

$$\mathscr{L}_{SC} = G_{DIQ}\left(\bar{q}_{i\alpha}i\gamma^5\varepsilon^{ijk}\varepsilon^{\alpha\beta\gamma}q^C_{j\beta}\right)\left(\bar{q}_{\ell\delta}i\gamma^5\varepsilon^{\ell mk}\varepsilon^{\delta\varepsilon\gamma}q^C_{m\varepsilon}\right)$$

$$\mathscr{L}_{SC6} = K_D\left(\bar{q}_{i\alpha}i\gamma^5\varepsilon^{ijk}\varepsilon^{\alpha\beta\gamma}q^C_{j\beta}\right)\left(\bar{q}_{\ell\delta}i\gamma^5\varepsilon^{\ell mn}\varepsilon^{\delta\varepsilon\eta}q^C_{m\varepsilon}\right)\left(\bar{q}_{k\gamma}q_{n\eta}\right)$$

We can simplify the relevant terms in $\mathscr{L}_{NJL}$:

$$\mathscr{L}_{NJL} = G_S\left[(\bar{u}u)^2 + \left(\bar{d}d\right)^2 + (\bar{s}s)^2\right]$$

and in $\mathscr{L}_{tHooft}$:

$$\mathscr{L}_{NJL} = G_D\left(\bar{u}u\bar{d}d\bar{s}s\right)$$

Using the definition:

$$\Delta^{k\gamma} = \left\langle\bar{q}i\gamma^5\varepsilon\varepsilon q^C\right\rangle$$

and the ansatzes:

$$(\bar{q}_1q_2)(\bar{q}_3q_4) \to \bar{q}_1q_2\left\langle\bar{q}_3q_4\right\rangle + \bar{q}_3q_4\left\langle\bar{q}_1q_2\right\rangle - \left\langle\bar{q}_1q_2\right\rangle\left\langle\bar{q}_3q_4\right\rangle$$

$$(\bar{q}_1q_2)(\bar{q}_3q_4)(\bar{q}_5q_6) \to \bar{q}_1q_2\left\langle\bar{q}_3q_4\right\rangle\left\langle\bar{q}_5q_6\right\rangle + \bar{q}_3q_4\left\langle\bar{q}_1q_2\right\rangle\left\langle\bar{q}_5q_6\right\rangle + \bar{q}_5q_6\left\langle\bar{q}_1q_2\right\rangle\left\langle\bar{q}_3q_4\right\rangle - 2\left\langle\bar{q}_1q_2\right\rangle\left\langle\bar{q}_3q_4\right\rangle\left\langle\bar{q}_5q_6\right\rangle$$

for the mean field approximation, we can rewrite the Lagrangian

$$\mathscr{L}_{NJL} = 2G_S\left[(\bar{u}u)\left\langle\bar{u}u\right\rangle + \left(\bar{d}d\right)\left\langle\bar{d}d\right\rangle + (\bar{s}s)\left\langle\bar{s}s\right\rangle - \left\langle\bar{u}u\right\rangle^2 - \left\langle\bar{d}d\right\rangle^2 - \left\langle\bar{s}s\right\rangle^2\right]$$

$$\mathscr{L}_{tHooft} = -2G_D\left[(\bar{u}u)\left\langle\bar{u}u\right\rangle\left\langle\bar{s}s\right\rangle + \left(\bar{d}d\right)\left\langle\bar{u}u\right\rangle\left\langle\bar{s}s\right\rangle + (\bar{s}s)\left\langle\bar{u}u\right\rangle\left\langle\bar{d}d\right\rangle - 2\left\langle\bar{u}u\right\rangle\left\langle\bar{d}d\right\rangle\left\langle\bar{s}s\right\rangle\right]$$

$$\mathscr{L}_{SC} = G_{DIQ}\left[\Delta^{k\gamma}\left(\bar{q}_{\ell\delta}i\gamma^5\varepsilon^{\ell mk}\varepsilon^{\delta\varepsilon\gamma}q^C_{m\varepsilon}\right) + \left(\bar{q}_{i\alpha}i\gamma^5\varepsilon^{ijk}\varepsilon^{\alpha\beta\gamma}q^C_{j\beta}\right)\Delta^{k\gamma\dagger} - \Delta^{k\gamma}\Delta^{k\gamma\dagger}\right]$$

$$\mathscr{L}_{SC6} = K_D\left[\left(\bar{q}_{m\varepsilon}q_{n\eta}\right)\Delta^{k\gamma}\Delta^{m\varepsilon\dagger} + \left(\bar{q}_{i\alpha}i\gamma^5\varepsilon^{ijk}\varepsilon^{\alpha\beta\gamma}q^C_{j\beta}\right)\Delta^{m\varepsilon\dagger}\left\langle\bar{q}_{m\varepsilon}q_{n\eta}\right\rangle\right]$$

$$+K_D\left[\Delta^{k\gamma}\left(\bar{q}_{\ell\delta}i\gamma^5\varepsilon^{\ell mn}\varepsilon^{\delta\varepsilon\eta}q^C_{m\varepsilon}\right)\left\langle\bar{q}_{m\varepsilon}q_{n\eta}\right\rangle - 2\Delta^{k\gamma}\Delta^{m\varepsilon\dagger}\left\langle\bar{q}_{m\varepsilon}q_{n\eta}\right\rangle\right]$$

If we make the definition $\tilde{\Delta} = 2G_{DIQ}\Delta$

References:

Created for Steiner05.

Definition at line 187 of file cfl6_eos.h.

Public Member Functions

- virtual int calc_eq_temp_p (**quark** &u, **quark** &d, **quark** &s, double &qq1, double &qq2, double &qq3, double &gap1, double &gap2, double &gap3, double mu3, double mu8, double &n3, double &n8, **thermo** &qb, double temper)
  Calculate the EOS.
- virtual int integrands (double p, double res[])
  The momentum integrands.
- virtual int test_derivatives (double lmom, double mu3, double mu8, **test_mgr** &t)
  Check the derivatives specified by eigenvalues()
- virtual int eigenvalues6 (double lmom, double mu3, double mu8, double egv[36], double dedmuu[36], double dedmud[36], double dedmus[36], double dedmu[36], double dedmd[36], double dedms[36], double dedu[36], double dedd[36], double deds[36], double dedmu3[36], double dedmu8[36])
  Calculate the energy eigenvalues and their derivatives.
- virtual int make_matrices (double lmom, double mu3, double mu8, double egv[36], double dedmuu[36], double dedmud[36], double dedmus[36], double dedmu[36], double dedmd[36], double dedms[36], double dedu[36], double dedd[36], double deds[36], double dedmu3[36], double dedmu8[36])
  Construct the matrices, but don't solve the eigenvalue problem.
- virtual const char * type ()
  Return string denoting type ("cfl6_eos")

Data Fields

- double KD
  The color superconducting 't Hooft coupling (default 0)
- double kdlimit
  The absolute value below which the CSC 't Hooft coupling is ignored(default 10^{-6})

Protected Member Functions

- int set_masses ()
  Set the quark effective masses from the gaps and the condensates.

Protected Attributes

- **omatrix_cx** iprop6
  Storage for the inverse propagator.
- **omatrix_cx** eivec6
  The eigenvectors.
- **omatrix_cx** dipdgapu
  The derivative wrt the ds gap.
- **omatrix_cx** dipdgapd
  The derivative wrt the us gap.
- **omatrix_cx** dipdgaps
  The derivative wrt the ud gap.
- **omatrix_cx** dipdqqu
  The derivative wrt the up quark condensate.
- **omatrix_cx** dipdqqd
  The derivative wrt the down quark condensate.

- **omatrix_cx** dipdqqs
  The derivative wrt the strange quark condensate.
- **ovector** eval6
  Storage for the eigenvalues.
- gsl_eigen_hermv_workspace * w6
  GSL workspace for the eigenvalue computation.

Static Protected Attributes

- static const int mat_size = 36
  The size of the matrix to be diagonalized.

Private Member Functions

- **cfl6_eos** (const cfl6_eos &)
- cfl6_eos & **operator=** (const cfl6_eos &)

14.5.2 Member Function Documentation

14.5.2.1 virtual int cfl6_eos::calc_eq_temp_p (quark & *u*, quark & *d*, quark & *s*, double & *qq1*, double & *qq2*, double & *qq3*, double & *gap1*, double & *gap2*, double & *gap3*, double *mu3*, double *mu8*, double & *n3*, double & *n8*, thermo & *qb*, double *temper*) [virtual]

Calculate the EOS from the quark condensates. Return the mass gap equations in `qq1`, `qq2`, `qq3`, and the normal gap equations in `gap1`, `gap2`, and `gap3`.

Using `fromqq=true` as in nambujl_eos and nambujl_temp_eos does not work here and will return an error.

If all of the gaps are less than gap_limit, then the nambujl_temp_eos::calc_temp_p() is used, and `gap1`, `gap2`, and `gap3` are set to equal `u.del`, `d.del`, and `s.del`, respectively.

Reimplemented from cfl_njl_eos.

14.5.2.2 virtual int cfl6_eos::eigenvalues6 (double *lmom*, double *mu3*, double *mu8*, double *egv[36]*, double *dedmuu[36]*, double *dedmud[36]*, double *dedmus[36]*, double *dedmu[36]*, double *dedmd[36]*, double *dedms[36]*, double *dedu[36]*, double *dedd[36]*, double *deds[36]*, double *dedmu3[36]*, double *dedmu8[36]*) [virtual]

Given the momentum `mom`, and the chemical potentials associated with the third and eighth gluons (`mu3` and `mu8`), this computes the eigenvalues of the inverse propagator and the assocated derivatives.

Note that this is not the same as cfl_njl_eos::eigenvalues() which returns `dedmu` rather `dedqqu`.

14.5.2.3 virtual int cfl6_eos::make_matrices (double *lmom*, double *mu3*, double *mu8*, double *egv[36]*, double *dedmuu[36]*, double *dedmud[36]*, double *dedmus[36]*, double *dedmu[36]*, double *dedmd[36]*, double *dedms[36]*, double *dedu[36]*, double *dedd[36]*, double *deds[36]*, double *dedmu3[36]*, double *dedmu8[36]*) [virtual]

This is used by check_derivatives() to make sure that the derivative entries are right.

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

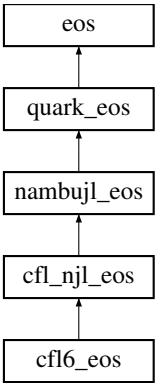
- cfl6_eos.h

14.6 cfl_njl_eos Class Reference

Nambu Jona-Lasinio model with a schematic CFL di-quark interaction at finite temperature.

`#include <cfl_njl_eos.h>`

Inheritance diagram for cfl_njl_eos:



14.6.1 Detailed Description

The variable B0 must be set before use.

The original Lagrangian is

$$\mathscr{L} = \mathscr{L}_{\rm Dirac} + \mathscr{L}_{4-{\rm fermion}} + \mathscr{L}_{6-{\rm fermion}} + \mathscr{L}_{CSC1} + \mathscr{L}_{CSC2}$$

$$\mathscr{L}_{\rm Dirac} = \bar{q}_{i\alpha}\left(i\partial\delta_{ij}\delta_{\alpha\beta} - m_{ij}\delta_{\alpha\beta} - \mu_{ij,\ \alpha\beta}\gamma^0\right)q_{j\beta}$$

$$\mathscr{L}_{4-{\rm fermion}} = G_S\sum_{a=0}^{8}\left[\left(\bar{q}\lambda_f^a q\right)^2 + \left(\bar{q}i\gamma_5\lambda_f^a q\right)^2\right]$$

$$\mathscr{L}_{6-{\rm fermion}} = -G_D\left[{\rm d}et_{ij}\,\bar{q}_{i\alpha}\left(1+i\gamma_5\right)q_{j\beta} + {\rm d}et_{ij}\,\bar{q}_{i\alpha}\left(1-i\gamma_5\right)q_{j\beta}\right]\delta_{\alpha\beta}$$

$$\mathscr{L}_{CSC1} = G_{DIQ}\sum_k\sum_\gamma\left[\left(\bar{q}_{i\alpha}\varepsilon_{ijk}\varepsilon_{\alpha\beta\gamma}q^C_{j\beta}\right)\left(\bar{q}^C_{i'\alpha'}\varepsilon_{i'j'k}\varepsilon_{\alpha'\beta'\gamma}q_{j'\beta'}\right)\right]$$

$$\mathscr{L}_{CSC2} = G_{DIQ}\sum_k\sum_\gamma\left[\left(\bar{q}_{i\alpha}i\gamma_5\varepsilon_{ijk}\varepsilon_{\alpha\beta\gamma}q^C_{j\beta}\right)\left(\bar{q}^C_{i'\alpha'}i\gamma_5\varepsilon_{i'j'k}\varepsilon_{\alpha'\beta'\gamma}q_{j'\beta'}\right)\right],$$

where μ is the quark number chemical potential. couplings G_S, G_D, and G_{DIQ} ultra-violet three-momentum cutoff, Λ

The thermodynamic potential is

$$\Omega(\mu_i, \langle\bar{q}q\rangle_i, \langle qq\rangle_i, T) = \Omega_{\rm vac} + \Omega_{\rm stat} + \Omega_0$$

where i runs over all nine (three colors times three flavors) quarks. We assume that the condensates are independent of color and that the quark chemical potentials are of the form $\mu_Q = \mu_{\rm Flavor(Q)} + \mu_{\rm Color(Q)}$ with

$$\mu_{\rm red} = \mu_3 + \mu_8/\sqrt{3} \quad \mu_{\rm green} = -\mu_3 + \mu_8/\sqrt{3} \quad \mu_{\rm blue} = -2\mu_8/\sqrt{3}$$

With these assumptions, the thermodynamic potential as given by the function thd_potential(), is a function of 12 variables

$$\Omega(\mu_u, \mu_d, \mu_s, \mu_3, \mu_8, \langle\bar{u}u\rangle, \langle\bar{d}d\rangle, \langle\bar{s}s\rangle, \langle ud\rangle, \langle us\rangle, \langle ds\rangle, T)$$

The individual terms are

$$\Omega_{\rm stat} = -\frac{1}{2}\int\frac{d^3p}{(2\pi)^3}\sum_{i=1}^{72}\left[\frac{\lambda_i}{2} + T\ln\left(1+e^{-\lambda_i/T}\right)\right]$$

$$\Omega_{\rm vac} = -2G_S \sum_{i=u,d,s} \langle \bar{q}_i q_i \rangle^2 + 4G_D \langle \bar{u}u \rangle \langle \bar{d}d \rangle \langle \bar{s}s \rangle + \sum_k \sum_\gamma \frac{|\Delta^{k\gamma}|^2}{4G_{DIQ}}$$

where λ_i are the eigenvalues of the (72 by 72) matrix (calculated by the function eigenvalues()

$$D = \begin{bmatrix} -\gamma^0 \vec{\gamma} \cdot \vec{p} - M_i \gamma^0 + \mu_{i\alpha} & \Delta i \gamma^0 \gamma_5 C \\ i\Delta \gamma^0 C \gamma_5 & -\gamma^0 \vec{\gamma}^T \cdot \vec{p} + M_i \gamma^0 - \mu_{i\alpha} \end{bmatrix}$$

and C is the charge conjugation matrix (in the Dirac representation).

The values of the various condensates are usually determined by the condition

$$\frac{\partial \Omega}{\langle \bar{q}q \rangle_i} = 0 \quad \frac{\partial \Omega}{\langle qq \rangle_i} = 0$$

Note that setting fixed_mass to `true` and setting all of the gaps to zero when `gap_limit` is less than zero will reproduce an analog of the bag model with a momentum cutoff.

The variable nambujl_eos::fromqq is automatically set to true in the constructor, as computations with `fromqq=false` are not implemented.

Idea for Future This class internally mixes ovector, omatrix, gsl_vector and **gsl_matrix** objects in a confusing and non-optimal way. Fix this.

Allow user to change derivative object? This isn't possible right now because the stepsize parameter of the derivative object is used.

References:

Created for Steiner02.

Definition at line 207 of file cfl_njl_eos.h.

Public Member Functions

- virtual int set_parameters (double lambda=0.0, double fourferm=0.0, double sixferm=0.0, double fourgap=0.0)
  Set the parameters and the bag constant 'B0'.
- virtual int calc_eq_temp_p (**quark** &u, **quark** &d, **quark** &s, double &qq1, double &qq2, double &qq3, double &gap1, double &gap2, double &gap3, double mu3, double mu8, double &n3, double &n8, **thermo** &qb, double temper)
  Calculate the EOS.
- virtual int test_derivatives (double lmom, double mu3, double mu8, **test_mgr** &t)
  Check the derivatives specified by eigenvalues()
- virtual int eigenvalues (double lmom, double mu3, double mu8, double egv[36], double dedmuu[36], double dedmud[36], double dedmus[36], double dedmu[36], double dedmd[36], double dedms[36], double dedu[36], double dedd[36], double deds[36], double dedmu3[36], double dedmu8[36])
  Calculate the energy eigenvalues as a function of the momentum.
- int set_quartic (**quartic_real_coeff** &q)
  Set the routine for solving quartics.
- int test_integration (**test_mgr** &t)
  Test the integration routines.
- int test_normal_eigenvalues (**test_mgr** &t)
  Test the routine to compute the eigenvalues of non-superfluid fermions.
- int test_gapped_eigenvalues (**test_mgr** &t)
  Test the routine to compute the eigenvalues of superfluid fermions.
- virtual const char * type ()
  Return string denoting type ("cfl_njl_eos")

Data Fields

- double eq_limit
  The equal mass threshold.
- bool integ_test
  Set to true to test the integration (default false)
- **cern_quartic_real_coeff** def_quartic
  The default quartic routine.
- double gap_limit
  Smallest allowable gap (default 0.0)
- bool zerot
  If this is true, then finite temperature corrections are ignored (default false)
- bool fixed_mass
  Use a fixed quark mass and ignore the quark condensates.
- bool color_neut
  If true, then ensure color neutrality.
- double GD
  Diquark coupling constant (default 3 G/4)
- double inte_epsabs
  The absolute precision for the integration (default 10^{-4} *)*
- double inte_epsrel
  The relative precision for the integration (default 10^{-4} *)*
- size_t inte_npoints
  The number of points used in the last integration (default 0)

Protected Member Functions

- virtual int integrands (double p, double res[])
  The integrands.
- int normal_eigenvalues (double m, double lmom, double mu, double lam[2], double dldmu[2], double dldm[2])
  Compute ungapped eigenvalues and the appropriate derivatives.
- int gapped_eigenvalues (double m1, double m2, double lmom, double mu1, double mu2, double tdelta, double lam[4], double dldmu1[4], double dldmu2[4], double dldm1[4], double dldm2[4], double dldg[4])
  Treat the simply gapped quarks in all cases gracefully.

For the integration

- double rescale_error (double err, double result_abs, double result_asc)
  The error scaling function for integ_err.
- int integ_err (double a, double b, const size_t nr, **ovector** &res, double &err2)
  A new version of **gsl_inte_qng** *to integrate several functions at the same time.*

Protected Attributes

- double temper
  Temperature.
- double smu3
  3rd gluon chemical potential
- double smu8
  8th gluon chemical potential

Numerical methods

- **quartic_real_coeff** * quartic
  The routine to solve quartics.

For computing eigenvalues

- **omatrix_cx** iprop
  Inverse propagator matrix.
- **omatrix_cx** eivec
  The eigenvectors.
- **omatrix_cx** dipdgapu
  The derivative of the inverse propagator wrt the ds gap.
- **omatrix_cx** dipdgapd
  The derivative of the inverse propagator wrt the us gap.
- **omatrix_cx** dipdgaps
  The derivative of the inverse propagator wrt the ud gap.
- **ovector** eval
  The eigenvalues.
- gsl_eigen_hermv_workspace * w
  Workspace for eigenvalue computation.

Private Member Functions

- **cfl_njl_eos** (const cfl_njl_eos &)
- cfl_njl_eos & **operator=** (const cfl_njl_eos &)

14.6.2 Member Function Documentation

14.6.2.1 virtual int cfl_njl_eos::set_parameters (double *lambda* = 0.0, double *fourferm* = 0.0, double *sixferm* = 0.0, double *fourgap* = 0.0) [virtual]

This function allows the user to specify the momentum cutoff, `lambda`, the four-fermion coupling `fourferm`, the six-fermion coupling from the 't Hooft interaction `sixferm`, and the color-superconducting coupling, `fourgap`. If 0.0 is given for any of the values, then the default is used ($\Lambda = 602.3/(\hbar c), G = 1.835/\Lambda^2, K = 12.36/\Lambda^5$).

If the four-fermion coupling that produces a gap is not specified, it is automatically set to 3/4 G, which is the value obtained from the Fierz transformation.

The value of the shift in the bag constant nambujl_eos::B0 is automatically calculated to ensure that the vacuum has zero energy density and zero pressure. The functions set_quarks() and set_thermo() must be used before hand to specify the **quark** and **thermo** objects.

14.6.2.2 virtual int cfl_njl_eos::calc_eq_temp_p (quark & *u*, quark & *d*, quark & *s*, double & *qq1*, double & *qq2*, double & *qq3*, double & *gap1*, double & *gap2*, double & *gap3*, double *mu3*, double *mu8*, double & *n3*, double & *n8*, thermo & *qb*, double *temper*) [virtual]

Calculate the EOS from the quark condensates in `u.qq`, `d.qq` and `s.qq`. Return the mass gap equations in `qq1`, `qq2`, `qq3`, and the normal gap equations in `gap1`, `gap2`, and `gap3`.

Using `fromqq=false` as in nambujl_eos and nambujl_eos does not work here and will return an error. Also, the quarks must be set through quark_eos::quark_set() before use.

If all of the gaps are less than gap_limit, then the nambujl_eos::calc_temp_p() is used, and `gap1`, `gap2`, and `gap3` are set to equal `u.del`, `d.del`, and `s.del`, respectively.

Todo It surprises me that n3 is not -res[11]. Is there a sign error in the color densities?

Reimplemented in cfl6_eos.

14.6.2.3 virtual int cfl_njl_eos::eigenvalues (double *lmom*, double *mu3*, double *mu8*, double *egv[36]*, double *dedmuu[36]*, double *dedmud[36]*, double *dedmus[36]*, double *dedmu[36]*, double *dedmd[36]*, double *dedms[36]*, double *dedu[36]*, double *dedd[36]*, double *deds[36]*, double *dedmu3[36]*, double *dedmu8[36]*) [virtual]

Given the momentum `mom`, and the chemical potentials associated with the third and eighth gluons (`mu3` and `mu8`), the energy eigenvalues are computed in egv[0] ... egv[35].

14.6.2.4 virtual int cfl_njl_eos::integrands (double *p*, double *res[]*) [protected, virtual]

- res[0] is the thermodynamic potential, Ω
- res[1] is $d-\Omega/dT$
- res[2] is $d\Omega/d\mu_u$
- res[3] is $d\Omega/d\mu_d$
- res[4] is $d\Omega/d\mu_s$
- res[5] is $d\Omega/dm_u$
- res[6] is $d\Omega/dm_d$
- res[7] is $d\Omega/dm_s$
- res[8] is $d\Omega/d\Delta_{ds}$
- res[9] is $d\Omega/d\Delta_{us}$
- res[10] is $d\Omega/d\Delta_{ud}$
- res[11] is $d\Omega/d\mu_3$
- res[12] is $d\Omega/d\mu_8$

Reimplemented in cfl6_eos.

14.6.2.5 int cfl_njl_eos::gapped_eigenvalues (double *m1*, double *m2*, double *lmom*, double *mu1*, double *mu2*, double *tdelta*, double *lam[4]*, double *dldmu1[4]*, double *dldmu2[4]*, double *dldm1[4]*, double *dldm2[4]*, double *dldg[4]*) [protected]

This function uses the quarks `q1` and `q2` to construct the eigenvalues of the inverse propagator, properly handling the either zero or finite quark mass and either zero or finite quark gaps. In the case of finite quark mass and finite quark gaps, the quartic solver is used.

The chemical potentials are separated so we can add the color chemical potentials to the quark chemical potentials if necessary.

This function is used by eigenvalues(). It does not work for the "ur-dg-sb" set of quarks which are paired in a non-trivial way.

Todo In the code, the equal mass case seems to be commented out. Why?

14.6.3 Field Documentation

14.6.3.1 cern_quartic_real_coeff cfl_njl_eos::def_quartic

Slightly better accuracy (with slower execution times) can be achieved using **gsl_poly_real_coeff** which polishes the roots of the quartics. For example

```
cfl_njl_eos cfl;
gsl_poly_real_coeff gp;
cfl.set_quartic(gp);
```

Definition at line 313 of file cfl_njl_eos.h.

14.6.3.2 double cfl_njl_eos::gap_limit

If any of the gaps are below this value, then it is assumed that they are zero and the equation of state is simplified accordingly. If all of the gaps are less than gap_limit, then the results from nambujl_eos are used in calc_eq_temp_p(), calc_temp_p() and thd_potential().

Definition at line 334 of file cfl_njl_eos.h.

14.6.3.3 bool cfl_njl_eos::zerot

This implements some simplifications in the momentum integration that are not possible at finite temperature.

Definition at line 342 of file cfl_njl_eos.h.

14.6.3.4 double cfl_njl_eos::GD

The default value is the one derived from a Fierz transformation. (Buballa04)

Definition at line 357 of file cfl_njl_eos.h.

14.6.3.5 double cfl_njl_eos::inte_epsabs

This is analogous to gsl_inte::epsabs

Definition at line 364 of file cfl_njl_eos.h.

14.6.3.6 double cfl_njl_eos::inte_epsrel

This is analogous to gsl_inte::epsrel

Definition at line 371 of file cfl_njl_eos.h.

14.6.3.7 size_t cfl_njl_eos::inte_npoints

This returns 21, 43, or 87 depending on the number of function evaluations needed to obtain the desired precision. If it the routine failes to obtain the desired precision, then this variable is set to 88.

Definition at line 381 of file cfl_njl_eos.h.

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

- cfl_njl_eos.h

14.7 cold_nstar Class Reference

Naive static cold neutron star.

`#include <cold_nstar.h>`

14.7.1 Detailed Description

This uses hadronic_eos::calc_e() to compute the equation of state of zero-temperature beta-equilibrated neutron star matter and tov_solve::mvsr() to compute the mass versus radius curve.

The electron and muon are given masses **o2scl_fm::mass_electron** and **o2scl_fm::mass_muon**, respectively.

There is an example for the usage of this class given in `examples/ex_cold_nstar.cpp`.

EOS Output

The function calc_eos() generates an object of type **table_units**, which contains the following columns

- `ed` in units of $1/\mathrm{fm}^4$, the total energy density of neutron star matter
- `pr` in units of $1/\mathrm{fm}^4$, the total pressure of neutron star matter
- `nb` in units of $1/\mathrm{fm}^3$, the baryon number density
- `mun` in units of $1/\mathrm{fm}$, the neutron chemical potential

---

- mup in units of 1/fm, the proton chemical potential
- mue in units of 1/fm, the electron chemical potential
- nn in units of 1/fm^3, the neutron number density
- np in units of 1/fm^3, the proton number density
- ne in units of 1/fm^3, the electron number density
- kfn in units of 1/fm, the neutron Fermi momentum
- kfp in units of 1/fm, the proton Fermi momentum
- kfe in units of 1/fm, the electron Fermi momentum.

If include_muons is true, the table has additional columns

- mumu in units of 1/fm, the muon chemical potential
- nmu in units of 1/fm^3, the muon number density
- kfmu in units of 1/fm, the muon Fermi momentum

If the energy density is always positive and increasing, and the pressure is always positive and increasing, then the EOS is well-formed and well_formed is true. The variable pressure_flat records the lowest baryon density where the pressure decreases with increasing density.

After computing the equation of state, calc_eos() also adds the following columns

- cs2 (unitless), the squared speed of sound
- logp, the logarithm of the pressure stored in pr
- loge, the logarithm of the energy density stored in ed
- s in units of 1/fm, the semi-perimeter of the Urca triangle
- urca in units of 1/fm^4, the squared area of the Urca triangle
- ad_index, the adiabatic index

The condition for the direct Urca process is the area of the triangle formed by the neutron, proton, and electron Fermi momenta. Using the definition of the semi-perimeter,

$$s \equiv \left(k_{F,n} + k_{F,p} + k_{F,e}\right)/2$$

Heron's formula gives the triangle area as

$$a = \sqrt{s(s-k_{F,n})(s-k_{F,p})(s-k_{F,e})}\,.$$

The column in the eos **table** labeled urca is a^2 . If this quantity is positive, then direct Urca is allowed. The variable allow_urca is the smallest density for which the direct Urca process turns on, and deny_urca is the smallest density for which the direct Urca process turns off.

The squared speed of sound (in units of c) is calculated by

$$c_s^2 = \frac{dP}{d\varepsilon}$$

and this is placed in the column labeled cs2. If the EOS is not well-formed, then this column is set to zero. If cs2 is larger than 1, the EOS is said to be "acausal". The variables acausal, acausal_ed, and acausal_pr record the baryon density, energy density, and pressure where the EOS becomes acausal. The adabatic index is calculated by

$$\Gamma = \frac{d\ln P}{d\ln \varepsilon}$$

Note that Γ must be greater than $4/3$ at the center of the neutron star for stability. (This is a necessary, but not sufficient condition.) If the EOS is not well-formed then this column is set to zero.

TOV Output

The TOV table contains all the columns typically generated for mass versus radius tables in tov_solve, as well as columns containing the central values of al the densities and chemical potentials, and all the other columns computed for the EOS above.

Idea for Future Warn if the EOS becomes pure neutron matter.

Definition at line 159 of file cold_nstar.h.

Public Member Functions

Basic operation

- int set_eos (hadronic_eos &he)
  Set the equation of state.
- int calc_eos (double np_0=0.0)
  Calculate the given equation of state.
- double calc_urca (double np_0=0.0)
  Compute the density at which the direct Urca process is allowe.
- int calc_nstar ()
  Calculate the M vs. R curve.

Data Fields

Default objects

- **fermion** def_n
  The default neutron.
- **fermion** def_p
  The default proton.
- fermion_zerot fzt
  Desc.
- tov_solve def_tov
  The default TOV equation solver.
- **cern_mroot_root**< **funct** > def_root
  The default equation solver for the EOS.
- tov_interp_eos def_tov_eos
  Default EOS object for the TOV solver.

Protected Member Functions

- double solve_fun (double x)
  Solve to ensure zero charge in β-equilibrium.

Protected Attributes

- bool eos_set
  True if equation of state has been set.
- **fermion** e
  The electron.
- **fermion** mu
  The muon.
- hadronic_eos * hep

A pointer to the equation of state.

- **fermion** ∗ np
  A pointer to the neutron.
- **fermion** ∗ pp
  A pointer to the proton.
- tov_solve ∗ tp
  A pointer to the TOV object.
- **root**< **funct** > ∗ rp
  A pointer to the solver.
- **o2_shared_ptr**< **table_units** >::type eost
  Storage for the EOS table.
- double barn
  The baryon density.

The thermodynamic information

- **thermo hb**
- **thermo h**
- **thermo l**

Output

- bool solver_success
  If true, the last call of calc_eos() succeeded.
- bool well_formed
  If true, the energy density of the EOS is monotonically increasing and the pressure is always positive.
- double pressure_flat
  The smallest baryon density where the pressure starts to decrease.
- double allow_urca
  The smallest density where Urca becomes allowed.
- double deny_urca
  The smallest density where Urca becomes disallowed.
- double acausal
  The density at which the EOS becomes acausal.
- double acausal_pr
  The pressure at which the EOS becomes acausal.
- double acausal_ed
  The energy density at which the EOS becomes acausal.
- double solver_tol
  Solver tolerance (default 10^{-4})
- int verbose
  Verbosity parameter (default 0)
- **o2_shared_ptr**< **table_units** >::type get_eos_results ()
  Get the eos table (after having called calc_eos())
- **o2_shared_ptr**< **table_units** >::type get_tov_results ()
  Get the results from the TOV (after having called calc_nstar())

Configuration

- double nb_start
  The starting baryon density (default 0.05)
- double nb_end
  The final baryon density (default 2.0)
- double dnb
  The baryon density stepsize (default 0.01)
- bool include_muons
  If true, include muons (default false)
- bool err_nonconv
  If true, throw an exception if the calculation fails (default true)

- int set_n_and_p (**fermion** &n, **fermion** &p)
  Set the neutron and proton.
- int set_root (**root**< **funct** > &rf)
  Set the equation solver for the EOS.
- int set_tov (tov_solve &ts)
  Specify the object for solving the TOV equations.

14.7.2 Member Function Documentation

14.7.2.1 int cold_nstar::set_eos (hadronic_eos & *he*) [inline]

This should be set before calling calc_eos().

Definition at line 171 of file cold_nstar.h.

14.7.2.2 double cold_nstar::calc_urca (double *np_0* = 0.0)

This is faster than using calc_eos() since it does nothing other than computes the critical density. It does not store the equation of state.

14.7.2.3 int cold_nstar::set_n_and_p (fermion & *n*, fermion & *p*) [inline]

The default objects are of type **fermion**, with mass **o2scl_fm::mass_neutron** and **o2scl_fm::mass_proton**. These defaults will give incorrect results for non-relativistic equations of state.

Definition at line 298 of file cold_nstar.h.

14.7.2.4 int cold_nstar::set_tov (tov_solve & *ts*) [inline]

The default uses the low-density equation of state with tov::verbose=0. In calc_nstar(), the units are set by calling tov_solve::set_units().

Definition at line 317 of file cold_nstar.h.

14.7.3 Field Documentation

14.7.3.1 double cold_nstar::pressure_flat

If this is zero after calling calc_eos(), then the pressure does not decrease in the specified range of baryon density

Definition at line 211 of file cold_nstar.h.

14.7.3.2 double cold_nstar::allow_urca

If this is zero after calling calc_eos(), then direct Urca is never allowed.

Definition at line 218 of file cold_nstar.h.

14.7.3.3 double cold_nstar::deny_urca

If this is zero after calling calc_eos(), then direct Urca is not disallowed at a higher density than it becomes allowed.

Definition at line 226 of file cold_nstar.h.

14.7.3.4 double cold_nstar::acausal

If this is zero, then the EOS is causal at all baryon densities in the specified range

Definition at line 233 of file cold_nstar.h.

14.7.3.5 double cold_nstar::acausal_pr

If this is zero, then the EOS is causal at all baryon densities in the specified range

Definition at line 240 of file cold_nstar.h.

14.7.3.6 double cold_nstar::acausal_ed

If this is zero, then the EOS is causal at all baryon densities in the specified range

Definition at line 247 of file cold_nstar.h.

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

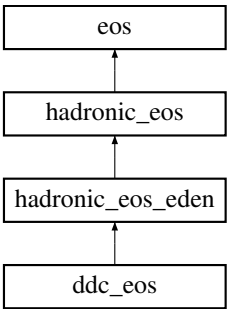
- cold_nstar.h

14.8 ddc_eos Class Reference

Relativistic mean field EOS with density dependent couplings.

`#include <ddc_eos.h>`

Inheritance diagram for ddc_eos:



14.8.1 Detailed Description

Based on Typel99.

Idea for Future Implement the finite temperature EOS properly.

Definition at line 46 of file ddc_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)
  Equation of state as a function of the densities.
- virtual int calc_eq_e (**fermion** &neu, **fermion** &p, double sig, double ome, double rho, double &f1, double &f2, double &f3, **thermo** &th)
  Equation of state and meson field equations as a function of the density.
- virtual const char * type ()
  Return string denoting type ("ddc_eos")

Data Fields

- double **rho0**

Masses

- double mnuc
  nucleon mass
- double ms
  ϕ *mass (in* fm^{-1} *)*
- double mw
  A_ω *mass (in* fm^{-1} *)*
- double mr
  A_ρ *mass (in* fm^{-1} *)*

Parameters for couplings

- double Gs
  The coupling $\Gamma_\sigma(\rho_{\mathrm{sat}})$.
- double Gw
  The coupling $\Gamma_\omega(\rho_{\mathrm{sat}})$.
- double Gr
  The coupling $\Gamma_\rho(\rho_{\mathrm{sat}})$.
- double as
  a_σ
- double aw
  a_ω
- double ar
  a_ρ
- double bs
  b_σ
- double bw
  b_ω
- double cs
  c_σ
- double cw
  c_ω
- double ds
  d_σ
- double dw
  d_ω

Protected Attributes

- fermion_zerot fzt
  Desc.

14.8.2 Member Function Documentation

14.8.2.1 virtual int ddc_eos::calc_eq_e (fermion & *neu*, fermion & *p*, double *sig*, double *ome*, double *rho*, double & *f1*, double & *f2*, double & *f3*, thermo & *th*) [virtual]

This calculates the pressure and energy density as a function of $\mu_n, \mu_p, \phi, A_\omega, A_\rho$. When the field equations have been solved, `f1`, `f2`, and `f3` are all zero.

Todo Is the thermodynamic identity is satisfied even when the field equations are not solved? Check this.

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

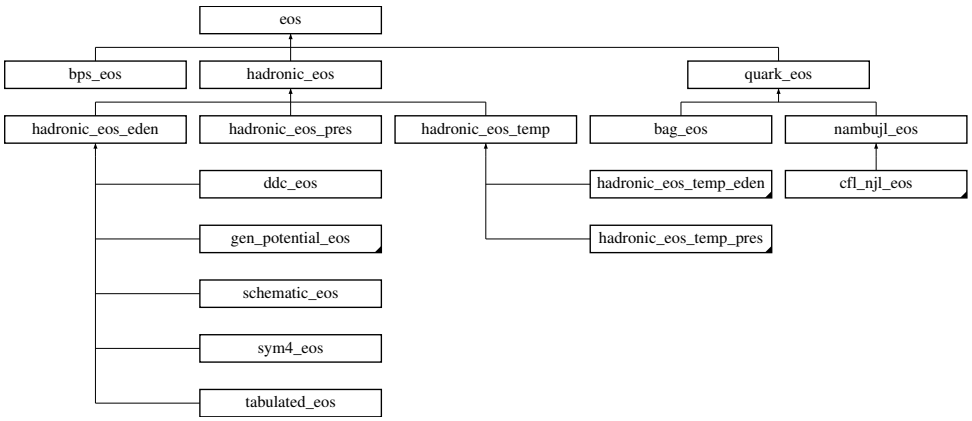
- ddc_eos.h

14.9 eos Class Reference

Equation of state base.

`#include <eos.h>`

Inheritance diagram for eos:



14.9.1 Detailed Description

A base class for the computation of an equation of state

Definition at line 37 of file eos.h.

Public Member Functions

- virtual int set_thermo (**thermo** &th)
  Set class thermo object.
- virtual int get_thermo (**thermo** ∗&th)
  Get class thermo object.
- virtual const char ∗ type ()
  Return string denoting type ("eos")

Data Fields

- **thermo** def_thermo
  The default thermo object.

Protected Attributes

- **thermo** ∗ eos_thermo
  A pointer to the thermo object.

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

- eos.h

14.10 ex_apr_eos Class Reference

Compute the APR EOS with a Gibbs construction and the mass versus radius curve [Example class].

14.10.1 Detailed Description

In succession, calculates nuclear matter, neutron matter, and then neutron star matter with Maxwell and Gibbs constructions.

We could use the more accurate masses in <o2scl/constants.h> here, but APR appears to have been designed to be used with m_n = m_p = 939 MeV.

Definition at line 55 of file ex_apr_eos.cpp.

Public Member Functions

- void run ()
  Main driver, computing the APR EOS and the associated M vs. R curve.

Protected Member Functions

- int maxwell_fig7 (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Function for the Maxwell construction in Fig. 7.
- int mixedmaxwell (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Maxwell construction of the nuclear matter mixed phase.
- int fig7fun (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Function to construct Fig. 7.
- int nstar (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Solve for neutron star matter (low-density phase)
- int nstar2 (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Solve for neutron star matter (high-density phase)
- int nstarmixed (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Solve for neutron star matter (mixed phase)
- void store_data ()
  Write a line of data to the table.
- int nucmixed (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Solve for nuclear matter (mixed phase)
- int neutmixed (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Solve for neutron matter (mixed phase)
- int nucleimat (size_t nv, const **ovector_base** &ex, **ovector_base** &ey)
  Solve for phase transition to nuclei.
- int nucleimat_pdrip (size_t nv, const **ovector_base** &ex, **ovector_base** &ey)
  Solve for phase transition to nuclei with a proton drip.

Protected Attributes

- **fermion** n
  Neutron for low-density phase.
- **fermion** p
  Proton for low-density phase.
- **fermion** n2
  Neutron for high-density phase.
- **fermion** p2
  Proton for high-density phase.
- fermion_zerot fzt
  Compute zero-temperature thermodynamics.
- **fermion** e
  Electron for low-density phase.
- **fermion** mu
  Muon for low-density phase.
- **fermion** e2
  Electron for high-density phase.
- **fermion** mu2

Muon for high-density phase.
- **thermo** hb
  Baryon thermodynamics for low-density phase.
- **thermo** l
  Leptonic thermodynamics for low-density phase.
- **thermo** hb2
  Baryon thermodynamics for high-density phase.
- **thermo** tot
  Total thermodynamics.
- **thermo** l2
  Leptonic thermodynamics for high-density phase.
- **gsl_mroot_hybrids** nd
  Solver.
- **gsl_mroot_hybrids** nd2
  Solver.
- **gsl_mroot_hybrids** sat_solver
  Solver for saturation properties.
- double nb
  Baryon density.
- double chi
  Volume fraction of low-density phase.
- double mub
  Baryon chemical potential.
- double muq
  Charge chemical potential.
- double f7x
  Proton fraction for Fig. 7.
- int choice
  Choice of model from APR.
- apr_eos ap
  Base APR EOS.
- **table_units** at
  Table for output.
- **cern_deriv**< **funct** > cd
  Derivative object.
- **hdf_file** hf
  HDF file for output.

Phase specification

- int **phase**
- static const int **low_phase** = 1
- static const int **mixed_phase** = 2
- static const int **high_phase** = 3

14.10.2 Member Function Documentation

14.10.2.1 void ex_apr_eos::run ( ) [inline]

Compute matter at densities below the maxwell construction

Definition at line 575 of file ex_apr_eos.cpp.

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

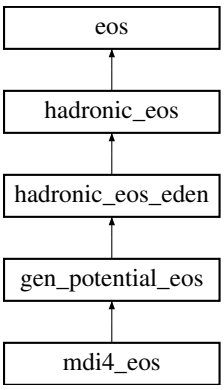
- ex_apr_eos.cpp

14.11 gen_potential_eos Class Reference

Generalized potential model equation of state.

`#include <gen_potential_eos.h>`

Inheritance diagram for gen_potential_eos:



14.11.1 Detailed Description

The single particle energy is defined by the functional derivative of the energy density with respect to the distribution function

$$e_\tau = \frac{\delta \varepsilon}{\delta f_\tau}$$

The effective mass is defined by

$$\frac{m^*}{m} = \left(\frac{m}{k} \frac{de_\tau}{dk} \right)^{-1}_{k=k_F}$$

In all of the models, the kinetic energy density is $\tau_n + \tau_p$ where

$$\tau_i = \frac{2}{(2\pi)^3} \int d^3k \left(\frac{k^2}{2m} \right) f_i(k,T)$$

and the number density is

$$\rho_i = \frac{2}{(2\pi)^3} \int d^3k \, f_i(k,T)$$

When form == mdi_form or gbd_form, the potential energy density is given by Das03 :

$$V(\rho,\delta) = \frac{Au}{\rho_0}\rho_n\rho_p + \frac{A_l}{2\rho_0}\left(\rho_n^2+\rho_p^2\right) + \frac{B}{\sigma+1}\frac{\rho^{\sigma+1}}{\rho_0^\sigma}\left(1-x\delta^2\right) + V_{mom}(\rho,\delta)$$

where $\delta = 1-2\rho_p/(\rho_n+\rho_p)$. If form == mdi_form, then

$$V_{mom}(\rho,\delta) = \frac{1}{\rho_0}\sum_{\tau,\tau'} C_{\tau,\tau'} \int\int d^3k d^3k' \frac{f_\tau(\vec{k}) f_{\tau'}(\vec{k}')}{1-(\vec{k}-\vec{k}')^2/\Lambda^2}$$

where $C_{1/2,1/2} = C_{-1/2,-1/2} = C_\ell$ and $C_{1/2,-1/2} = C_{-1/2,1/2} = C_u$. Otherwise if form == gbd_form, then

$$V_{mom}(\rho,\delta) = \frac{1}{\rho_0}\left[C_\ell\left(\rho_n g_n + \rho_p g_p\right) + C_u\left(\rho_n g_p + \rho_p g_n\right)\right]$$

where

$$g_i = \frac{\Lambda^2}{\pi^2}\left[k_{F,i} - \Lambda \tan^{-1}\left(k_{F,i}/\Lambda\right)\right]$$

Otherwise, if form == bgbd_form, bpalb_form or sl_form, then the potential energy density is given by Bombaci01 :

$$V(\rho,\delta) = V_A + V_B + V_C$$

$$V_A = \frac{2A}{3\rho_0}\left[\left(1+\frac{x_0}{2}\right)\rho^2 - \left(\frac{1}{2}+x_0\right)\left(\rho_n^2+\rho_p^2\right)\right]$$

$$V_B = \frac{4B}{3\rho_0^\sigma}\frac{T}{1+4B'T/\left(3\rho_0^{\sigma-1}\rho^2\right)}$$

where

$$T = \rho^{\sigma-1}\left[\left(1+\frac{x_3}{2}\right)\rho^2 - \left(\frac{1}{2}+x_3\right)\left(\rho_n^2+\rho_p^2\right)\right]$$

The term V_C is:

$$V_C = \sum_{i=1}^{i_{\max}} \frac{4}{5}\left(C_i+2z_i\right)\rho\left(g_{n,i}+g_{p,i}\right) + \frac{2}{5}\left(C_i-8z_i\right)\left(\rho_n g_{n,i}+\rho_p g_{p,i}\right)$$

where

$$g_{\tau,i} = \frac{2}{(2\pi)^3}\int d^3k f_\tau(k,T) g_i(k)$$

For form == bgbd_form or form == bpalb_form, the form factor is given by

$$g_i(k) = \left(1+\frac{k^2}{\Lambda_i^2}\right)^{-1}$$

while for form == sl_form, the form factor is given by

$$g_i(k) = 1 - \frac{k^2}{\Lambda_i^2}$$

where Λ_1 is specified in the parameter `Lambda` when necessary.

See Mathematica notebook at

```
doc/o2scl/extras/gen_potential_eos.nb
doc/o2scl/extras/gen_potential_eos.ps
```

Bug The BGBD EOS doesn't work and the effective mass for the GBD EOS doesn't work

Idea for Future Calculate the chemical potentials analytically

Definition at line 170 of file gen_potential_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &ne, **fermion** &pr, **thermo** &lt)
  Equation of state as a function of density.
- int set_mu_deriv (**deriv**< **funct** > &de)
  Set the derivative object to calculate the chemical potentials.
- virtual const char * type ()
  Return string denoting type ("gen_potential_eos")

Data Fields

- int form
  Form of potential.
- **gsl_deriv**< **funct** > def_mu_deriv
  The default derivative object for calculating chemical potentials.

The parameters for the various interactions

- double **x**
- double **Au**
- double **Al**
- double **rho0**
- double **B**
- double **sigma**
- double **Cl**
- double **Cu**
- double **Lambda**
- double **A**
- double **x0**
- double **x3**
- double **Bp**
- double **C1**
- double **z1**
- double **Lambda2**
- double **C2**
- double **z2**
- double **bpal_esym**
- int **sym_index**

Static Public Attributes

- static const int mdi_form = 1
  The "momentum-dependent-interaction" form.
- static const int bgbd_form = 2
  The modifed GBD form.
- static const int bpalb_form = 3
  The form from Prakash88 as formulated in Bombaci01.
- static const int sl_form = 4
  The "SL" form. See Bombaci01.
- static const int gbd_form = 5
  The Gale, Bertsch, Das Gupta from Gale87.
- static const int bpal_form = 6
  The form from Prakash88.

Protected Member Functions

- double mom_integral (double pft, double pftp)
  Compute the momentum integral for mdi_form.
- double energy (double x)
  Compute the energy.

Protected Attributes

- **nonrel_fermion** nrf
  Desc.
- bool mu_deriv_set
  True of the derivative object has been set.
- **deriv**< **funct** > ∗ mu_deriv_ptr
  The derivative object.

The mode for the energy() function [protected]

- int **mode**
- static const int **nmode** = 1
- static const int **pmode** = 2
- static const int **normal** = 0

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

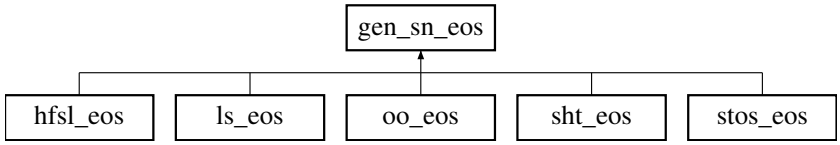
- gen_potential_eos.h

14.12 gen_sn_eos Class Reference

A base class for the supernova EOSs [abstract].

`#include <gen_sn_eos.h>`

Inheritance diagram for gen_sn_eos:



14.12.1 Detailed Description

This class is experimental.

The EOSs are stored in a set of **tensor_grid3** objects on grids with density in fm^{-3}, electron fraction (unitless) and temperature in MeV.

Some tabulated EOSs do not store data for gen_sn_eos::E, gen_sn_eos::F, gen_sn_eos::S, and gen_sn_eos::P. In that case, the grid is set for these objects but the data is set to zero. To compute these from the data after loading the EOS table, use gen_sn_eos::compute_eg().

The function load() loads the entire EOS into memory. Memory allocation is automatically performed by load(), but not deallocated until free() or the destructor is called.

After loading, you can interpolate the EOS by using **tensor_grid3::interp** directly. For example, the following returns the mass number at an arbitrary baryon density, electron fraction, and temperature assuming the table is stored in `skm.dat`:

```
ls_eos ls;
ls.load("skm.dat");
double nb=0.01, Ye=0.2, T=10.0;
cout << A.interp(nb,Ye,T) << endl;
```

Interpolation for all EOSs is linear by default.

Todo Ensure all chemical potentials are based on the same rest masses?

Allow logarithmic grids for any of nb, Ye, or T.

Idea for Future Create a **table** object, possibly using **tensor_grid::vector_slice**.

Idea for Future Show how matrix_slice and vector_slice can be used with this object.

Idea for Future Could this be a child of hadronic_eos_temp and then directly used in cold_nstar()?

Idea for Future Add option to load and store a separate lepton/photon EOS

Idea for Future Add muons and/or pions

Definition at line 84 of file gen_sn_eos.h.

Public Member Functions

- int check_free_energy (**test_mgr** &tm)
  Test the free energy and store results in `tm`.
- virtual void beta_eq_T0 (size_t i, double &nb, double &E_beta, double &P_beta, double &Ye_beta, double &Z_beta, double &A_beta)=0
  Compute energy per baryon of matter in beta equilibrium at zero temperature at a fixed grid point [abstract].
- virtual void beta_eq_sfixed (size_t i, double entr, double &nb, double &E_beta, double &P_beta, double &Ye_beta, double &Z_beta, double &A_beta, double &T_beta)
  Compute properties of matter in beta equilibrium at s=4.

Memory allocation

- int alloc ()
  Allocate memory.
- int free ()
  Free allocated memory.

Data Fields

- int verbose
  Verbosity parameter (default 1)

Data

- **tensor_grid3** F
  Total free energy per baryon in MeV.
- **tensor_grid3** Fint
  Free energy per baryon without lepton and photon contributions in MeV.
- **tensor_grid3** E
  Total internal energy per baryon in MeV.
- **tensor_grid3** Eint
  Internal energy per baryon without lepton and photon contributions in MeV.
- **tensor_grid3** P
  Total pressure in $\mathrm{MeV}/\mathrm{fm}^3$.
- **tensor_grid3** Pint
  Pressure without lepton and photon contributions in $\mathrm{MeV}/\mathrm{fm}^3$.
- **tensor_grid3** S
  Total entropy per baryon.
- **tensor_grid3** Sint
  Entry per baryon without lepton and photon contributions.
- **tensor_grid3** mun
  Neutron chemical potential in MeV.
- **tensor_grid3** mup
  Proton chemical potential in MeV.
- **tensor_grid3** Z
  Proton number.
- **tensor_grid3** A
  Mass number.
- **tensor_grid3** Xn
  Neutron fraction.

- **tensor_grid3** Xp
  Proton fraction.
- **tensor_grid3** Xalpha
  Alpha particle fraction.
- **tensor_grid3** Xnuclei
  Fraction of heavy nuclei.
- **tensor_grid3** other [20]
  Other data sets.
- **tensor_grid3** * arr [n_base+20]
  List of pointers to data.

Load table

- bool loaded
  If true, a EOS table was successfully loaded (default false)
- bool with_leptons_loaded
  True if thermodynamics with leptons has been loaded.
- bool baryons_only_loaded
  True if baryon-only thermodynamics has been loaded.
- virtual void load (std::string fname)=0
  Load table from filename `fname`.

Grid and data sizes

- size_t n_nB
  Size of baryon density grid.
- size_t n_Ye
  Size of electron fraction grid.
- size_t n_T
  Size of temperature grid.
- size_t n_oth
  Number of additional data sets.
- static const size_t n_base = 16
  Number of base data sets.

Interpolation

- **def_interp_mgr**< **uvector_base**, **linear_interp** > dim1
  Default interpolation object.
- **def_interp_mgr** < **uvector_const_subvector**, **linear_interp** > dim2
  Default interpolation object.
- int set_interp (**base_interp_mgr**< **uvector_base** > &bi1, **base_interp_mgr**< **uvector_const_subvector** > &bi2)
  Set interpolation managers.

Electron and photon contribution

- **boson** photon
  Photon.
- **fermion** electron
  Electron;.
- **rel_fermion** relf
  Desc.
- **eff_boson** effb
  Desc.
- int compute_eg ()
  Compute the electron and photon contribution for the full grid.

14.12.2 Member Function Documentation

14.12.2.1 int gen_sn_eos::compute_eg ( )

This function computes the data for `E, P, S,` and `F` by adding electrons and photons to the baryon contributions stored in `Eint, Pint, Sint,` and `Fint`.

The electron contribution to the internal energy and free energy computed by this function includes the electron rest mass.

14.12.2.2 int gen_sn_eos::check_free_energy (test_mgr & *tm*)

This checks that the data in `Fint` is consistent with that in `Eint` and `Sint` .

14.12.2.3 virtual void gen_sn_eos::beta_eq_T0 (size_t *i*, double & *nb*, double & *E_beta*, double & *P_beta*, double & *Ye_beta*, double & *Z_beta*, double & *A_beta*) [pure virtual]

Given an index `i` for the baryon grid, between 0 and n_nB (inclusive), this computes the properties of matter in beta equilibrium at zero temperature by finding the electron fraction grid point which minimizes E. The baryon density is returned in `nb`, the energy per baryon in `E_beta`, the pressure in `P_beta`, the electron fraction in `Ye_beta`, the proton number in `Z_beta` and the mass number of the nucleus in `A_beta`.

Some tables explicitly contain zero-temperature data which is used when it is available. Otherwise, linear interpolation is used to extrapolate down to zero from the lowest temperature grid points.

Implemented in hfsl_eos, sht_eos, stos_eos, oo_eos, and ls_eos.

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

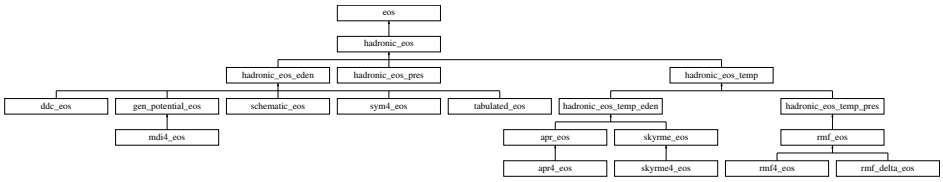
- gen_sn_eos.h

14.13 hadronic_eos Class Reference

Hadronic equation of state [abstract base].

`#include <hadronic_eos.h>`

Inheritance diagram for hadronic_eos:



14.13.1 Detailed Description

Denote the number density of neutrons as n_n, the number density of protons as n_p, the total baryon density $n_B = n_n + n_p$, the asymmetry $\alpha \equiv (n_n - n_p)/n_B$, the nuclear saturation density as $n_0 \approx 0.16$ fm^{-3}, and the quantity $\eta \equiv (n - n_0)/3n_0$. Then the energy per baryon of nucleonic matter matter can be written as an expansion around $\varepsilon = \alpha = 0$

$$E(n_B,\alpha) = -B + \frac{\tilde{K}}{2!}\varepsilon^2 + \frac{Q_0}{3!}\varepsilon^3 + \alpha^2\left(S + L\varepsilon + \frac{K_{\rm sym}}{2!}\varepsilon^2 + \frac{Q_{\rm sym}}{3!}\varepsilon^3\right) + E_4(n_B,\alpha) + \mathscr{O}(\alpha^6) \qquad \text{(Eq. 1)}$$

where E_4 represents the quartic terms

$$E_4(n_B,\alpha) = \alpha^4\left(S_4 + L_4\varepsilon + \frac{K_4}{2!}\varepsilon^2 + \frac{Q_4}{3!}\varepsilon^3\right) \qquad \text{(Eq. 2)}$$

(Adapted slightly from Piekarewicz09). From this, one can compute the energy density of nuclear matter $\varepsilon(n_B,\alpha) = n_B E(n_B,\alpha)$, the chemical potentials $\mu_i \equiv (\partial\varepsilon)/(\partial n_i)$ and the pressure $P = -\varepsilon + \mu_n n_n + \mu_p n_p$. This expansion motivates the definition of several separate terms. The binding energy B of symmetric nuclear matter ($\alpha = 0$) is around 16 MeV.

The compression modulus is usually defined by $\chi = -1/V(dV/dP) = 1/n(dP/dn)^{-1}$. In nuclear physics it has become common to use the incompressibility (or bulk) modulus with an extra factor of 9, $K = 9/(n\chi)$ and refer to K simply as the incompressibility. Here, we define the function

$$K(n_B,\alpha) \equiv 9\left(\frac{\partial P}{\partial n_B}\right) = 9n_B\left(\frac{\partial^2\varepsilon}{\partial n_B^2}\right)$$

This quantity is computed by the function fcomp() by computing the first derivative of the pressure, which is more numerically stable than the second derivative of the energy density (and most O_2scl EOSs compute the pressure exactly). This function is typically evaluated at the point $(n_B = n_0, \alpha = 0)$ and is stored in comp. This quantity is not always the same as $\tilde{K}$, defined here as

$$\tilde{K}(n_B,\alpha) = 9n_B^2\left(\frac{\partial^2 E}{\partial n_B^2}\right) = K(n_B,\alpha) - \frac{1}{n_B}18P(n_B,\alpha)$$

We denote $K \equiv K(n_B = n_0, \alpha = 0)$ and similarly for $\tilde{K}$, the quantity in Eq. 1 above. In nuclear matter at saturation, the pressure is zero and $K = \tilde{K}$. See Chabanat97 for a discussion of this distinction.

The symmetry energy $S(n_B,\alpha)$ can be defined as

$$S(n_B,\alpha) \equiv \frac{1}{2n_B}\frac{\partial^2\varepsilon}{\partial\alpha^2}$$

and the parameter S in Eq. 1 is just $S(n_0,0)$. Using

$$\left(\frac{\partial\varepsilon}{\partial\alpha}\right)_{n_B} = \frac{\partial\varepsilon}{\partial n_n}\left(\frac{\partial n_n}{\partial\alpha}\right)_{n_B} + \frac{\partial\varepsilon}{\partial n_p}\left(\frac{\partial n_p}{\partial\alpha}\right)_{n_B} = \frac{n_B}{2}(\mu_n - \mu_p)$$

this can be rewritten

$$S(n_B,\alpha) = \frac{1}{4}\frac{\partial}{\partial\alpha}(\mu_n - \mu_p)$$

where the dependence of the chemical potentials on n_B and α is not written explicitly. This quantity is computed by function fesym(). Note that many of the functions in this class are written in terms of the proton fraction $x_p = (1-\alpha)/2$ denoted as `'pf'` instead of as functions of α. Frequently, $S(n_B,\alpha)$ is evaluated at $\alpha = 0$ to give a univariate function of the baryon density. It is sometimes also evaluated at the point $(n_B = n_0, \alpha = 0)$, and this value is denoted by S above and is typically stored in esym. Alternatively, one can define the symmetry energy by

$$\tilde{S}(n_B) \approx E(n_B,\alpha = 1) - E(n_B,\alpha = 0)$$

which is computed by function fesym_diff() . The functions $S(n_B,\alpha = 0)$ and $\tilde{S}(n_B)$ are equal when $\mathcal{O}(\alpha^4)$ terms are zero. In this case, $\mu_n - \mu_p$ is proportional to α and so

$$S(n_B) = \tilde{S}(n_B) = \frac{1}{4}\frac{(\mu_n - \mu_p)}{\alpha}.$$

The symmetry energy slope parameter L, can be defined by

$$L(n_B,\alpha) \equiv 3n_B\frac{\partial S(n_B,\alpha)}{\partial n_B} = 3n_B\frac{\partial}{\partial n_B}\left[\frac{1}{2n_B}\frac{\partial^2\varepsilon}{\partial\alpha^2}\right]$$

This can be rewritten as

$$L(n_B,\alpha) = \frac{3n_B}{4}\frac{\partial}{\partial n_B}\frac{\partial}{\partial\alpha}(\mu_n - \mu_p)$$

(where the derivatives can be evaluated in either order) or alternatively using

$$\left(\frac{\partial\varepsilon}{\partial n_B}\right)_{\alpha} = \frac{\partial\varepsilon}{\partial n_n}\left(\frac{\partial n_n}{\partial n_B}\right)_{\alpha} + \frac{\partial\varepsilon}{\partial n_p}\left(\frac{\partial n_p}{\partial n_B}\right)_{\alpha} = \frac{1}{2}(\mu_n + \mu_p)$$

L can be rewritten

$$\begin{aligned} L(n_B,\alpha) &= 3n_B\left[\frac{-1}{2n_B^2}\frac{\partial^2\varepsilon}{\partial\alpha^2} + \frac{1}{4n_B}\frac{\partial^2}{\partial\alpha^2}(\mu_n + \mu_p)\right] \\ &= \frac{3}{4}\frac{\partial^2}{\partial\alpha^2}(\mu_n + \mu_p) - 3S(n_B,\alpha). \end{aligned}$$

The third derivative with respect to the baryon density is sometimes called the skewness. Here, we define

$$Q_0(n_B,\alpha) = 27 n_B^3 \frac{\partial^3}{\partial n_B^3}\left(\frac{\varepsilon}{n_B}\right) = 27 n_B^3 \frac{\partial^2}{\partial n_B^2}\left(\frac{P}{n_B^2}\right)$$

and this function is computed in fkprime() .

The second derivative of the symmetry energy with respect to the baryon density is

$$K_{\rm sym}(n_B,\alpha) = 9 n_B^2 \frac{\partial^2}{\partial n_B^2} S(n_B,\alpha)$$

The third derivative of the symmetry energy with respect to the baryon density is

$$Q_{\rm sym}(n_B,\alpha) = 27 n_B^3 \frac{\partial^3}{\partial n_B^3} S(n_B,\alpha)$$

Note that solving for the baryon density for which $P=0$ gives, to order α^2 (Piekarewicz09)

$$n_B = n_0\left[1 + \frac{6K}{Q} + \alpha^2\left(\frac{3L}{K} - \frac{6K_{\rm sym}}{Q} + \frac{6KQ_{\rm sym}}{Q^2}\right)\right]$$

The quartic symmetry energy $S_4(n_B,\alpha)$ can be defined as

$$S_4(n_B,\alpha) \equiv \frac{1}{24 n_B}\frac{\partial^4 \varepsilon}{\partial \alpha^4}$$

However, fourth derivatives are difficult numerically, and so an alternative quantity is preferable. Instead, one can evaluate the extent to which $\mathcal{O}(\alpha^4)$ terms are important from

$$\eta(n_B) \equiv \frac{E(n_B,1) - E(n_B,1/2)}{3\left[E(n_B,1/2) - E(n_B,0)\right]}$$

as described in Steiner06 . This function can be expressed either in terms of $\tilde{S}$ or S_4

$$\eta(n_B) = \frac{5\tilde{S}(n_B) - S(n_B,0)}{\tilde{S}(n_B) + 3S(n_B,0)} = \frac{5S_4(n_B,0) + 4S(n_B,0)}{S_4(n_B,0) + 4S(n_B,0)}$$

Evaluating this function at the saturation density gives

$$\eta(n_0) = \frac{4S + 5S_4}{4S + S_4}$$

(Note that S_4 is referred to as Q in Steiner06). Sometimes it is useful to separate out the kinetic and potential parts of the energy density when computing $\eta(n_B)$, and the class sym4_eos_base is useful for this purpose.

Idea for Future Could write a function to compute the "symmetry free energy" or the "symmetry entropy"

Definition at line 248 of file hadronic_eos.h.

Public Member Functions

- int gradient_qij (**fermion** &n, **fermion** &p, **thermo** &th, double &qnn, double &qnp, double &qpp, double &dqnndnn, double &dqnndnp, double &dqnpdnn, double &dqnpdnp, double &dqppdnn, double &dqppdnp)
  Calculate coefficients for gradient part of Hamiltonian.
- virtual const char * type ()
  Return string denoting type ("hadronic_eos")

Equation of state

- virtual int calc_p (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of the chemical potentials.
- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of density.

EOS properties

- virtual double fcomp (double nb, const double &alpha=0.0)
  Calculate the incompressibility in fm^{-1} *using calc_e()*
- virtual double feoa (double nb, const double &alpha=0.0)
  Calculate the energy per baryon in fm^{-1} *using calc_e()*
- virtual double fesym (double nb, const double &alpha=0.0)
  Calculate symmetry energy of matter in fm^{-1} *using calc_dmu_alpha()* .
- virtual double fesym_err (double nb, double &alpha, double &unc)
  Calculate symmetry energy of matter and its uncertainty.
- virtual double fesym_slope (double nb, const double &alpha=0.0)
  The symmetry energy slope parameter.
- virtual double fesym_curve (double nb, const double &alpha=0.0)
  The curvature of the symmetry energy.
- virtual double fesym_skew (double nb, const double &alpha=0.0)
  The skewness of the symmetry energy.
- virtual double fesym_diff (double nb)
  Calculate symmetry energy of matter as energy of neutron matter minus the energy of nuclear matter.
- virtual double feta (double nb)
  The strength parameter for quartic terms in the symmetry energy.
- virtual double fkprime (double nb, const double &alpha=0.0)
  Calculate skewness of nuclear matter using calc_e()
- virtual double fmsom (double nb, const double &alpha=0.0)
  Calculate reduced neutron effective mass using calc_e()
- virtual double fn0 (double alpha, double &leoa)
  Calculate saturation density using calc_e()
- virtual int saturation ()
  Calculates some of the EOS properties at the saturation density.

Functions for calculating physical properties

- double calc_dmu_alpha (double alpha, const double &nb)
  Compute the difference between neutron and proton chemical potentials as a function of the isospin asymmetry.
- double calc_musum_alpha (double alpha, const double &nb)
  Compute the sum of the neutron and proton chemical potentials as a function of the isospin asymmetry.
- double calc_pressure_nb (double nb, const double &alpha=0.0)
  Compute the pressure as a function of baryon density at fixed isospin asymmetry.
- double calc_edensity_nb (double nb, const double &alpha=0.0)
  Compute the energy density as a function of baryon density at fixed isospin asymmetry.
- void const_pf_derivs (double nb, double pf, double &ednb_pf, double &dPdnb_pf)
  Compute derivatives at constant proton fraction.
- double calc_press_over_den2 (double nb, const double &alpha=0.0)
  Calculate pressure / baryon density squared in nuclear matter as a function of baryon density at fixed isospin asymmetry.
- double calc_edensity_alpha (double alpha, const double &nb)
  Calculate energy density as a function of the isospin asymmetry at fixed baryon density.

Other functions

- int nuc_matter_p (size_t nv, const **ovector_base** &x, **ovector_base** &y, double ∗&pa)
  Nucleonic matter from calc_p()
- int nuc_matter_e (size_t nv, const **ovector_base** &x, **ovector_base** &y, double ∗&pa)
  Nucleonic matter from calc_e()

Set auxilliary objects

- virtual int set_mroot (**mroot**< **mm_funct**<> > &mr)
  Set class mroot object for use in calculating chemical potentials from densities.
- virtual int set_sat_root (**root**< **funct** > &mr)

Set class mroot object for use calculating saturation density.

- virtual int set_sat_deriv (**deriv**< **funct** > &de)
  Set **deriv** *object to use to find saturation properties.*
- virtual int set_sat_deriv2 (**deriv**< **funct** > &de)
  Set the second **deriv** *object to use to find saturation properties.*
- virtual int set_n_and_p (**fermion** &n, **fermion** &p)
  Set neutron and proton.

Data Fields

- double eoa
  Binding energy.
- double comp
  Compressibility.
- double esym
  Symmetry energy.
- double n0
  Saturation density.
- double msom
  Effective mass (neutron)
- double kprime
  Skewness.
- **fermion** def_neutron
  The defaut neutron.
- **fermion** def_proton
  The defaut proton.

Default solvers and derivative classes

- **gsl_deriv**< **funct** > def_deriv
  The default object for derivatives.
- **gsl_deriv**< **funct** > def_deriv2
  The second default object for derivatives.
- **gsl_mroot_hybrids**< **mm_funct**<> > def_mroot
  The default solver.
- **cern_mroot_root**< **funct** > def_sat_root
  The default solver for calculating the saturation density.

Protected Member Functions

- double t1_fun (double barn)
  Compute t1 for gradient_qij().
- double t2_fun (double barn)
  Compute t2 for gradient_qij().

Protected Attributes

- **mroot**< **mm_funct**<> > ∗ eos_mroot
  The EOS solver.
- **root**< **funct** > ∗ sat_root
  The solver to compute saturation properties.
- **deriv**< **funct** > ∗ sat_deriv
  The derivative object for saturation properties.
- **deriv**< **funct** > ∗ sat_deriv2
  The second derivative object for saturation properties.
- **fermion** ∗ neutron
  The neutron object.
- **fermion** ∗ proton
  The proton object.

14.13.2 Member Function Documentation

14.13.2.1 virtual double hadronic_eos::fcomp (double *nb*, const double & *alpha* = 0.0) [virtual]

This function computes $K(n_B,\alpha) = 9n_B\partial^2\varepsilon/(\partial n_B^2) = 9\partial P/(\partial n_B)$. The value of $K(n_0,0)$, often referred to as the "compressibility", is stored in comp by saturation() and is about 240 MeV at saturation density.

14.13.2.2 virtual double hadronic_eos::feoa (double *nb*, const double & *alpha* = 0.0) [virtual]

This function computes the energy per baryon of matter without the nucleon rest masses at the specified baryon density, `nb`, and isospin asymmetry `alpha`.

14.13.2.3 virtual double hadronic_eos::fesym (double *nb*, const double & *alpha* = 0.0) [virtual]

This function computes the symmetry energy,

$$\left(\frac{1}{2n_B}\frac{d^2\varepsilon}{d\alpha^2}\right) = \frac{1}{4}\frac{\partial}{\partial\alpha}\left(\mu_n-\mu_p\right)$$

at the value of n_B given in `nb` and α given in `alpha`. The symmetry energy at $\alpha = 0$ at the saturation density and is stored in esym by saturation().

14.13.2.4 virtual double hadronic_eos::fesym_err (double *nb*, double & *alpha*, double & *unc*) [virtual]

This estimates the uncertainty due to the numerical differentiation, assuming that difference betwen the neutron and proton chemical potentials is computed exactly by calc_dmu_alpha() .

14.13.2.5 virtual double hadronic_eos::fesym_slope (double *nb*, const double & *alpha* = 0.0) [virtual]

This returns the value of the "slope parameter" of the symmetry energy

$$L = 3n_B\left(\frac{\partial E_{sym}}{\partial n_B}\right)$$

in inverse Fermis.

where n_B is the baryon density. This ranges between about zero and 200 MeV for many EOSs.

14.13.2.6 virtual double hadronic_eos::fesym_diff (double *nb*) [virtual]

This function returns the energy per baryon of neutron matter minus the energy per baryon of nuclear matter. This will deviate significantly from the results from fesym() only if the dependence of the symmetry energy on δ is not quadratic.

Reimplemented in apr_eos.

14.13.2.7 virtual double hadronic_eos::fkprime (double *nb*, const double & *alpha* = 0.0) [virtual]

The skewness is defined to be $27n^3d^3(\varepsilon/n)/(dn^3) = 27n^3d^2(P/n^2)/(dn^2)$ and is denoted 'kprime'. This definition seems to be ambiguous for densities other than the saturation density and is not quite analogous to the compressibility.

14.13.2.8 virtual double hadronic_eos::fmsom (double *nb*, const double & *alpha* = 0.0) [virtual]

Neutron effective mass (as stored in **part::ms**) divided by vacuum mass (as stored in **part::m**) in nuclear matter at saturation density. Note that this simply uses the value of n.ms from calc_e(), so that this effective mass could be either the Landau or Dirac mass depending on the context. Note that this may not be equal to the reduced proton effective mass.

14.13.2.9 virtual double hadronic_eos::fn0 (double *alpha*, double & *leoa*) [virtual]

This function finds the baryon density for which the pressure vanishes.

14.13.2.10 virtual int hadronic_eos::saturation ( ) [virtual]

Idea for Future It would be great to provide numerical uncertainties in the saturation properties.

Reimplemented in rmf_eos, and rmf_delta_eos.

14.13.2.11 int hadronic_eos::gradient_qij (fermion & *n*, fermion & *p*, thermo & *th*, double & *qnn*, double & *qnp*, double & *qpp*, double & *dqnndnn*, double & *dqnndnp*, double & *dqnpdnn*, double & *dqnpdnp*, double & *dqppdnn*, double & *dqppdnp*)

Note

This is still somewhat experimental.

We want the gradient part of the Hamiltonian in the form

$$\mathscr{H}_{\rm grad} = \frac{1}{2} \sum_{i=n,p} \sum_{j=n,p} Q_{ij} \vec{\nabla} n_i \cdot \vec{\nabla} n_j$$

The expression for the gradient terms from Pethick95 is

$$\begin{aligned} \mathscr{H}_{\rm grad} &= -\frac{1}{4}\left(2P_1 + P_{1;f} - P_{2;f}\right) \\ &+\frac{1}{2}\left(Q_1+Q_2\right)\left(n_n\nabla^2 n_n + n_p\nabla^2 n_p\right) \\ &+\frac{1}{4}\left(Q_1-Q_2\right)\left[\left(\nabla n_n\right)^2 + \left(\nabla n_p\right)^2\right] \\ &+\frac{1}{2}\frac{dQ_2}{dn}\left(n_n\nabla n_n + n_p\nabla n_p\right)\cdot\nabla n \end{aligned}$$

This can be rewritten

$$\begin{aligned} \mathscr{H}_{\rm grad} &= \frac{1}{2}\left(\nabla n\right)^2\left[\frac{3}{2}P_1 + n\frac{dP_1}{dn}\right] \\ &-\frac{3}{4}\left[\left(\nabla n_n\right)^2 + \left(\nabla n_p\right)^2\right] \\ &-\frac{1}{2}[]\cdot\nabla n\frac{dQ_1}{dn} \\ &-\frac{1}{4}\left(\nabla n\right)^2 P_2 - \frac{1}{4}\left[\left(\nabla n_n\right)^2 + \left(\nabla n_p\right)^2\right]Q_2 \end{aligned}$$

or

$$\begin{aligned} \mathscr{H}_{\rm grad} &= \frac{1}{4}\left(\nabla n\right)^2\left[3P_1 + 2n\frac{dP_1}{dn} - P_2\right] \\ &-\frac{1}{4}\left(3Q_1+Q_2\right)\left[\left(\nabla n_n\right)^2 + \left(\nabla n_p\right)^2\right] \\ &-\frac{1}{2}\frac{dQ_1}{dn}\left[n_n\nabla n_n + n_p\nabla n_p\right]\cdot\nabla n \end{aligned}$$

or

$$\begin{aligned} \mathscr{H}_{\rm grad} &= \frac{1}{4}\left(\nabla n\right)^2\left[3P_1 + 2n\frac{dP_1}{dn} - P_2\right] \\ &-\frac{1}{4}\left(3Q_1+Q_2\right)\left[\left(\nabla n_n\right)^2 + \left(\nabla n_p\right)^2\right] \\ &-\frac{1}{2}\frac{dQ_1}{dn}\left[n_n\left(\nabla n_n\right)^2 + n_p\left(\nabla n_p\right)^2 + n\nabla n_n\cdot\nabla n_p\right] \end{aligned}$$

Generally, for Skyrme-like interactions

$$\begin{aligned} P_i &= \frac{1}{4}t_i\left(1+\frac{1}{2}x_i\right) \\ Q_i &= \frac{1}{4}t_i\left(\frac{1}{2}+x_i\right). \end{aligned}$$

for $i = 1,2$.

This function uses the assumption $x_1 = x_2 = 0$ to calculate t_1 and t_2 from the neutron and proton effective masses assuming the Skyrme form. The values of Q_{ij} and their derivatives are then computed.

The functions set_n_and_p() and set_thermo() will be called by gradient_qij(), to facilitate the use of the `n`, `p`, and `th` parameters.

14.13.2.12 double hadronic_eos::calc_pressure_nb (double *nb*, const double & *alpha* = `0.0`)

Used by fcomp().

14.13.2.13 double hadronic_eos::calc_edensity_nb (double *nb*, const double & *alpha* = `0.0`)

This function calls hadronic_eos::calc_e() with the internally stored neutron and proton objects.

14.13.2.14 double hadronic_eos::calc_press_over_den2 (double *nb*, const double & *alpha* = `0.0`)

Used by fkprime().

14.13.2.15 double hadronic_eos::calc_edensity_alpha (double *alpha*, const double & *nb*)

Used by fesym().

This function calls hadronic_eos::calc_e() with the internally stored neutron and proton objects.

14.13.2.16 virtual int hadronic_eos::set_mroot (**mroot**< **mm_funct**<> > & *mr*) `[virtual]`

Note

While in principle this allows one to use any **mroot** object, in practice some of the current EOSs require **gsl_mroot_hybrids** because it automatically avoids regions where the equations are undefined.

14.13.2.17 virtual int hadronic_eos::set_sat_root (**root**< **funct** > & *mr*) `[virtual]`

Note

While in principle this allows one to use any **mroot** object, in practice some of the current EOSs require **gsl_mroot_hybrids** because it automatically avoids regions where the equations are undefined.

14.13.2.18 virtual int hadronic_eos::set_sat_deriv2 (**deriv**< **funct** > & *de*) `[virtual]`

Computing the slope of the symmetry energy at the saturation density requires two derivative objects, because it has to take an isospin derivative and a density derivative. Thus this second **deriv** object is used in the function fesym_slope().

14.13.3 Field Documentation

14.13.3.1 gsl_deriv<funct > hadronic_eos::def_deriv

The value of **gsl_deriv::h** is set to 10^{-3} in the hadronic_eos constructor.

Definition at line 608 of file hadronic_eos.h.

14.13.3.2 gsl_deriv<funct> hadronic_eos::def_deriv2

The value of **gsl_deriv::h** is set to 10^{-3} in the hadronic_eos constructor.

Definition at line 615 of file hadronic_eos.h.

14.13.3.3 gsl_mroot_hybrids<mm_funct<> > hadronic_eos::def_mroot

Used by calc_e() to solve nuc_matter_p() (2 variables) and by calc_p() to solve nuc_matter_e() (2 variables).

Definition at line 622 of file hadronic_eos.h.

14.13.3.4 cern_mroot_root<funct > hadronic_eos::def_sat_root

Used by fn0() (which is called by saturation()) to solve saturation_matter_e() (1 variable).

Definition at line 630 of file hadronic_eos.h.

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

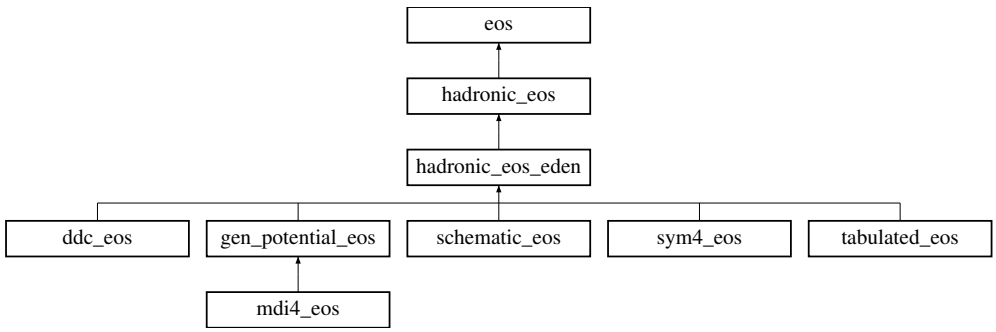
- hadronic_eos.h

14.14 hadronic_eos_eden Class Reference

A hadronic EOS based on a function of the densities [abstract base].

`#include <hadronic_eos.h>`

Inheritance diagram for hadronic_eos_eden:



14.14.1 Detailed Description

Definition at line 669 of file hadronic_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of density.
- virtual int calc_p (**fermion** &n, **fermion** &p, **thermo** &th)
  Equation of state as a function of the chemical potentials.

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

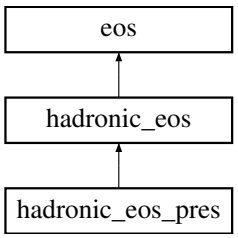
- hadronic_eos.h

14.15 hadronic_eos_pres Class Reference

A hadronic EOS based on a function of the chemical potentials [abstract base].

`#include <hadronic_eos.h>`

Inheritance diagram for hadronic_eos_pres:



14.15.1 Detailed Description

Definition at line 685 of file hadronic_eos.h.

Public Member Functions

- virtual int calc_p (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of the chemical potentials.
- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)
  Equation of state as a function of density.

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

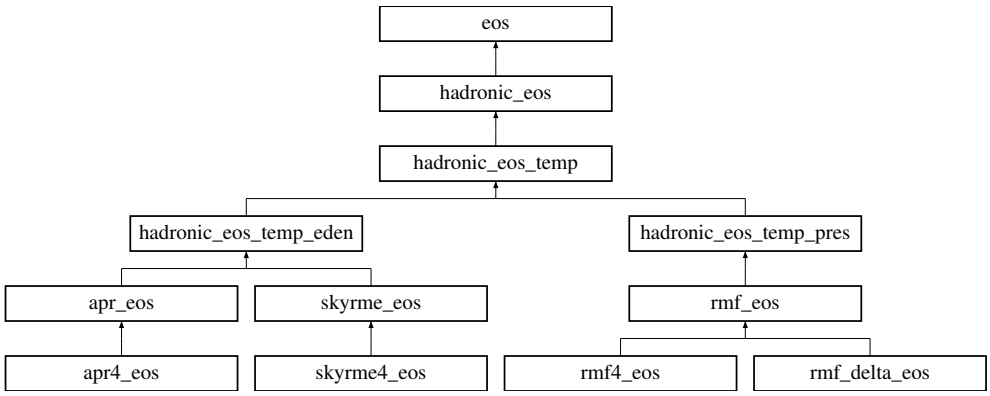
- hadronic_eos.h

14.16 hadronic_eos_temp Class Reference

A finite temperature hadronic EOS.

`#include <hadronic_eos.h>`

Inheritance diagram for hadronic_eos_temp:



14.16.1 Detailed Description

Definition at line 699 of file hadronic_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of density.
- virtual int calc_temp_e (**fermion** &n, **fermion** &p, double T, **thermo** &th)=0
  Equation of state as a function of densities at finite temperature.
- virtual int calc_p (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of the chemical potentials.
- virtual int calc_temp_p (**fermion** &n, **fermion** &p, double T, **thermo** &th)=0
  Equation of state as a function of the chemical potentials at finite temperature.

Data Fields

- **eff_fermion** def_fet
  Desc.

Protected Member Functions

- int nuc_matter_temp_e (size_t nv, const **ovector_base** &x, **ovector_base** &y, double ∗&pa)
  Solve for nuclear matter at finite temperature given density.
- int nuc_matter_temp_p (size_t nv, const **ovector_base** &x, **ovector_base** &y, double ∗&pa)
  Solve for nuclear matter at finite temperature given mu.

Protected Attributes

- fermion_eval_thermo ∗ fet
  Desc.
- double lT
  The temperature.

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

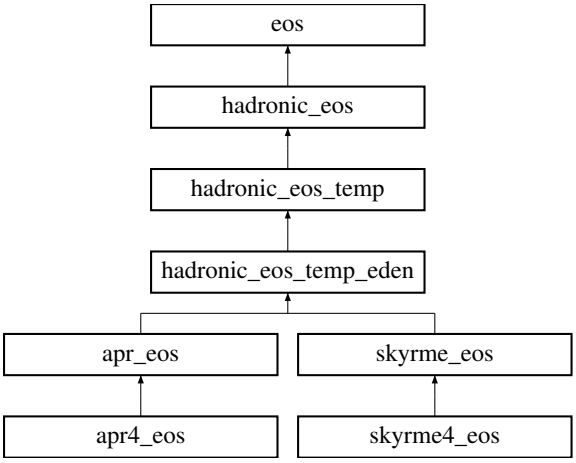
- hadronic_eos.h

14.17 hadronic_eos_temp_eden Class Reference

A hadronic EOS at finite temperature based on a function of the densities [abstract base].

`#include <hadronic_eos.h>`

Inheritance diagram for hadronic_eos_temp_eden:



14.17.1 Detailed Description

Definition at line 757 of file hadronic_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of density.
- virtual int calc_temp_e (**fermion** &n, **fermion** &p, double T, **thermo** &th)=0
  Equation of state as a function of densities at finite temperature.
- virtual int calc_p (**fermion** &n, **fermion** &p, **thermo** &th)
  Equation of state as a function of the chemical potentials.
- virtual int calc_temp_p (**fermion** &n, **fermion** &p, double T, **thermo** &th)
  Equation of state as a function of the chemical potentials at finite temperature.

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

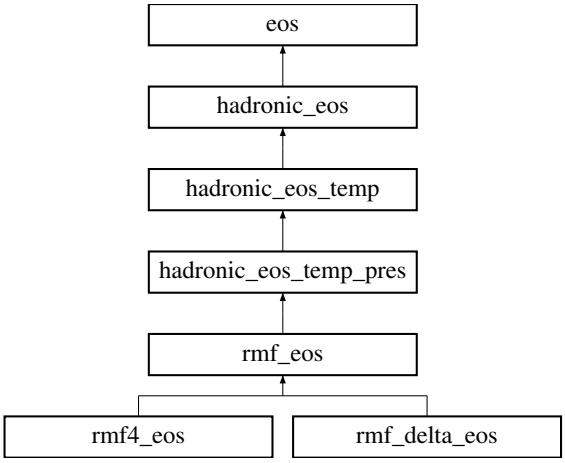
- hadronic_eos.h

14.18 hadronic_eos_temp_pres Class Reference

A hadronic EOS at finite temperature based on a function of the chemical potentials [abstract base].

`#include <hadronic_eos.h>`

Inheritance diagram for hadronic_eos_temp_pres:



14.18.1 Detailed Description

Definition at line 785 of file hadronic_eos.h.

Public Member Functions

- virtual int calc_p (**fermion** &n, **fermion** &p, **thermo** &th)=0
  Equation of state as a function of the chemical potentials.
- virtual int calc_temp_p (**fermion** &n, **fermion** &p, double T, **thermo** &th)=0
  Equation of state as a function of the chemical potentials at finite temperature.
- virtual int calc_e (**fermion** &n, **fermion** &p, **thermo** &th)
  Equation of state as a function of density.
- virtual int calc_temp_e (**fermion** &n, **fermion** &p, double T, **thermo** &th)
  Equation of state as a function of densities at finite temperature.

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

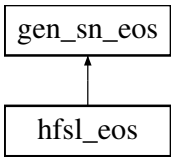
- hadronic_eos.h

14.19 hfsl_eos Class Reference

The Hempel et al. supernova EOSs.

`#include <gen_sn_eos.h>`

Inheritance diagram for hfsl_eos:



14.19.1 Detailed Description

This class is experimental.

Note

O_2scl Does not contain the EOS, only provides some code to manipulate it. This class was designed to be used with the files `dd2_frdm_eos_shen98format_v1.02.tab`, `fsg_roca_eos_shen98format_v1.0.tab`, and `nl3_lala_eos_shen98format_v1.0.tab` as obtained from http://phys-merger.physik.unibas.ch/~hempel/eos.html.

See also the documentation at gen_sn_eos.

See Hempel10 and Hempel11.

Definition at line 856 of file gen_sn_eos.h.

Public Member Functions

- virtual void load (std::string fname)
  Load table from filename `fname`.
- virtual void beta_eq_T0 (size_t i, double &nb, double &E_beta, double &P_beta, double &Ye_beta, double &Z_beta, double &A_beta)
  Compute properties of matter in beta equilibrium at zero temperature at a baryon density grid point.

Data Fields

- **tensor_grid3** & log_rho
  Logarithm of baryon number density in $\mathrm{g/cm^3}$.
- **tensor_grid3** & nB
  Baryon number density in $1/\mathrm{fm^3}$.
- **tensor_grid3** & log_Y
  Logarithm of proton fraction.
- **tensor_grid3** & Yp
  Proton fraction.
- **tensor_grid3** & M_star
  Nucleon effective mass in MeV.
- **tensor_grid3** & A_light
  Mass number of light fragments.
- **tensor_grid3** & Z_light
  Proton number of light fragments.

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

- gen_sn_eos.h

14.20 rmf_nucleus::initial_guess Struct Reference

Initial guess structure.

`#include <rmf_nucleus.h>`

14.20.1 Detailed Description

The initial guess for the meson field profiles is a set of fermi-dirac functions, i.e.

$$\sigma(r) = \mathrm{sigma0}/[1+\exp((r-\mathrm{fermi\_radius})/\mathrm{fermi\_width})]$$

Definition at line 499 of file rmf_nucleus.h.

Data Fields

- double sigma0
  Scalar field at r=0.
- double omega0
  Vector field at r=0.
- double rho0
  Isuvector field at r=0.
- double A0
  Coulomb field at r=0.
- double fermi_radius
  The radius for which the fields are half their central value.
- double fermi_width
  The "width" of the Fermi-Dirac function.

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

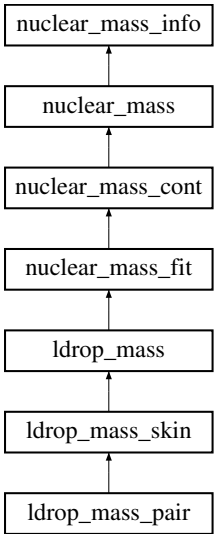
- rmf_nucleus.h

14.21 ldrop_mass Class Reference

Simple liquid drop mass formula.

`#include <ldrop_mass.h>`

Inheritance diagram for ldrop_mass:



14.21.1 Detailed Description

Includes bulk part plus surface and Coulomb (no pairing) without neutron skin and without any isospin contribution to the surface energy.

The NL4 EOS is loaded by default.

Warning

This class sets **part::inc_rest_mass** to true for the particle objects specified in set_n_and_p().

Central densities

Given a saturation density, n_0 and a transition density, n_t, we set $I = 1-2Z/A$, and then assume $\delta = I$. We further assume that the isospin-asymmetric saturation density n_L is

$$n_L = n_0 + n_1 \delta^2$$

and then we can compute $n_p = (1-\delta)/2n_L$ and $n_n = (1+\delta)/2n_L$.

Note that $\delta = I$ implies no neutron skin. A neutron skin occurs when $\delta < I$, and $\delta = 0$ implies a "maximum skin size" which is occurs when no extra neutrons are in center and all extra neutrons are located in the skin, i.e. $N_{\rm skin} = N - Z$.

Nuclear radii

The neutron and proton radii are determined from the central densities with

$$\begin{aligned} R_n &= \left(\frac{3N}{4\pi n_n}\right)^{1/3} \\ R_n &= \left(\frac{3Z}{4\pi n_p}\right)^{1/3} \end{aligned}$$

Bulk energy contribution

The bulk binding energy contribution (~ -16 MeV per nucleon) and the symmetry energy are computing using the hadronic EOS (either def_had_eos or the EOS specified in the most recent call to set_hadronic_eos_temp()). The bulk energy per baryon is

$$E_{\rm bulk}/A = \frac{\hbar c}{n_L}\left[\varepsilon(n_n,n_p) - n_n m_n - n_p m_p\right]$$

Surface energy contribution

The surface energy density is (Ravenhall83)

$$\varepsilon = \frac{\chi d\sigma}{R}$$

where σ is the surface tension. The factor χ is typically taken care of by the caller, so we ignore it for now. To compute the surface energy per baryon, we divide by the baryon density, $n_n + n_p$. We can rewrite this

$$E_{\rm surf} = \frac{3\sigma}{n_n+n_p}\left[\frac{3A}{4(n_n+n_p)\pi}\right]^{-1/3}$$

or

$$E_{\rm surf} = \frac{\sigma}{n_L}\left(\frac{36\pi n_L}{A}\right)^{1/3}$$

where the surface tension σ (in MeV) is given in surften.

Taking a typical value, $\sigma = 1.1$ MeV and $n_L = 0.16$ fm^{-3}, gives the standard result, $E_{\rm surf}/A = 18$ MeV $A^{-1/3}$.

Coulomb energy contribution

The Coulomb energy density (Ravenhall83) is

$$\varepsilon_{\rm Coul} = \frac{4\pi}{5} n_p^2 e^2 R_p^2$$

The energy per baryon is

$$E_{\rm Coul}/A = \frac{4\pi}{5n_L} n_p^2 e^2 R_p^2$$

This is the expression used in the code, except for a prefactor coul_coeff which is a fit parameter and should be close to unity.

Assuming $R_p = R$ and $Z = \frac{4\pi}{3} R^3 n_p$ and $R = [3A/(4\pi n_L)]^{1/3}$ gives

$$E_{\rm Coul}/A = \frac{6^{2/3}}{5} \pi^{1/3} e^2 n_L^{1/3} \frac{Z^2}{A^{4/3}}$$

and taking $n_L = 0.16$ fm^{-3} and $e^2 = \hbar c/137$ gives the standard result

$$E_{\rm Coul}/A = 0.76 \text{ MeV } Z^2 A^{-4/3}$$

References

Designed for Steiner08 based on Lattimer85 and Lattimer91 .

Definition at line 237 of file ldrop_mass.h.

Public Member Functions

- virtual double mass_excess_d (double Z, double N)
  Given `Z` and `N`, return the mass excess in MeV.
- virtual double drip_binding_energy_d (double Z, double N, double npout, double nnout, double chi)
  Given `Z` and `N`, return the binding energy in MeV.
- virtual const char * type ()
  Return the type, `"ldrop_mass"`.

Fitting functions

- virtual int fit_fun (size_t nv, const **ovector_base** &x)
  Fix parameters from an array for fitting.
- virtual int guess_fun (size_t nv, **ovector_base** &x)
  Fill array with guess from present values for fitting.

Data Fields

- **thermo** th
  Energy and pressure.

Input parameters

- double n1
  Density asymmetry (default 0)
- double n0
  Saturation density (The default is 0.16fm^{-3}*)*
- double surften
  Surface tension in MeV (default 1.1 MeV)
- double coul_coeff
  Coulomb coefficient (default 1.0)

Output quantities

- double nn
  Internal average neutron density.
- double np
  Internal average proton density.
- double Rn
  Neutron radius.
- double Rp
  Proton radius.
- double bulk
  Bulk part of energy.
- double surf
  Surface part of energy.
- double coul
  Coulomb part of energy.

Protected Attributes

- **eff_fermion** eff
  Desc.
- **fermion** * n
  Pointer to neutron.
- **fermion** * p
  Pointer to proton.
- hadronic_eos_temp * heos
  The base EOS for bulk matter.

EOS and particle parameters

- rmf_eos def_had_eos
  The default hadronic EOS.
- **fermion** def_neutron
  Default neutron.
- **fermion** def_proton
  Default proton.
- int set_hadronic_eos_temp (hadronic_eos_temp &uhe)
  Change the base hadronic EOS.
- void set_n_and_p (**fermion** &un, **fermion** &up)
  Change neutron and proton objects.

14.21.2 Member Function Documentation

14.21.2.1 virtual double ldrop_mass::mass_excess_d (double *Z*, double *N*) [virtual]

Implements **nuclear_mass_cont**.

14.21.2.2 virtual double ldrop_mass::drip_binding_energy_d (double *Z*, double *N*, double *npout*, double *nnout*, double *chi*) [virtual]

This function is currently independent of `npout`, `nnout`, and `chi`.

Reimplemented in ldrop_mass_skin.

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

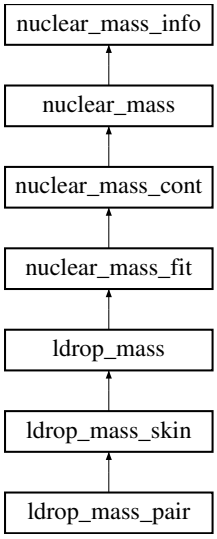
- ldrop_mass.h

14.22 ldrop_mass_pair Class Reference

Liquid drop model with pairing.

`#include <ldrop_mass.h>`

Inheritance diagram for ldrop_mass_pair:



14.22.1 Detailed Description

This class adds a pairing correction

$$E_{\rm pair}/A = -\frac{\zeta}{2A^{3/2}}\left[\cos(Z\pi)+\cos(N\pi)\right]$$

where ζ is stored in Epair. The trigonometric functions give the expected result for integer values of N and Z.

Definition at line 581 of file ldrop_mass.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "ldrop_mass_pair".
- virtual int fit_fun (size_t nv, const **ovector_base** &x)
  Fix parameters from an array for fitting.
- virtual int guess_fun (size_t nv, **ovector_base** &x)
  Fill array with guess from present values for fitting.
- virtual double drip_binding_energy_full_d (double Z, double N, double npout, double nnout, double chi, double T)
  Return the free binding energy of a nucleus in a many-body environment.

Data Fields

- double Epair
  Pairing energy coefficient (default 13 MeV)
- double pair
  Most recently computed pairing energy per baryon.

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

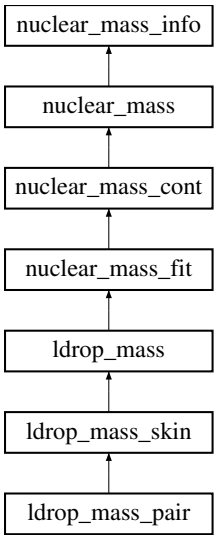
- ldrop_mass.h

14.23 ldrop_mass_skin Class Reference

More advanced liquid drop model.

`#include <ldrop_mass.h>`

Inheritance diagram for ldrop_mass_skin:



14.23.1 Detailed Description

In addition to the physics in ldrop_mass, this includes corrections for

- finite temperature
- neutron skin
- an isospin-dependent surface energy
- decrease in the Coulomb energy from external protons

Note

The input parameter T should be given in units of inverse Fermis -- this is a bit unusual since the binding energy is returned in MeV, but we keep it for now.

Bulk energy

The central densities and radii, n_n, n_p, R_n, R_p are all determined in the same way as ldrop_mass, except that now $\delta \equiv I\zeta$, where ζ is stored in doi . Note that this means $N > Z$ iff $R_n > R_p$.

If new_skin_mode is false, then the bulk energy is also computed as in ldrop_mass. Otherwise, the number of nucleons in the core is computed with

$$A_{\rm core} = Z(n_n + n_p)/n_p \text{ for } N \geq Z$$
$$A_{\rm core} = N(n_n + n_p)/n_p \text{ for } Z > N$$

and $A_{\rm skin} = A - A_{\rm core}$. The core contribution to the bulk energy is

$$E_{\rm core}/A = \left(\frac{A_{\rm core}}{A}\right)\frac{\hbar c}{n_L}\left[\varepsilon(n_n, n_p) - n_n m_n - n_p m_p\right]$$

then the skin contribution is

$$E_{\rm skin}/A = \left(\frac{A_{\rm skin}}{A}\right)\frac{\hbar c}{n_L}\left[\varepsilon(n_n, 0) - n_n m_n\right] \text{ for } N > Z$$

and

$$E_{\rm skin}/A = \left(\frac{A_{\rm skin}}{A}\right)\frac{\hbar c}{n_L}\left[\varepsilon(0,n_p) - n_p m_p\right] \text{ for } Z > N$$

Surface energy

If full_surface is false, then the surface energy is just that from ldrop_mass , with an extra factor for the surface symmetry energy

$$E_{\rm surf} = \frac{\sigma}{n_L}\left(\frac{36\pi n_L}{A}\right)^{1/3}\left(1-\sigma_\delta \delta^2\right)$$

where σ_δ is unitless and stored in ss.

If full_surface is true, then the surface energy is modified by a cubic dependence for the medium and contains finite temperature corrections.

Coulomb energy

The Coulomb energy density (Ravenhall83) is

$$\varepsilon = 2\pi e^2 R_p^2 n_p^2 f_d(\chi_p)$$

where the function $f_d(\chi)$ is

$$f_d(\chi_p) = \frac{1}{(d+2)}\left[\frac{2}{(d-2)}\left(1-\frac{d}{2}\chi_p^{(1-2/d)}\right)+\chi_p\right]$$

This class takes $d = 3$.

Todos and Future

Todo This is based on LPRL, but it's a little different in Lattimer and Swesty. I should document what the difference is.

Todo The testing could be updated.

Idea for Future Add translational energy?

Idea for Future Remove excluded volume correction and compute nuclear mass relative to the gas rather than relative to the vacuum.

Idea for Future In principle, Tc should be self-consistently determined from the EOS.

Idea for Future Does this work if the nucleus is "inside-out"?

References

Designed in Steiner08 and Souza09 based in part on Lattimer85 and Lattimer91 .

Definition at line 497 of file ldrop_mass.h.

Public Member Functions

- virtual const char * type ()
  Return the type, "ldrop_mass_skin".
- virtual int fit_fun (size_t nv, const **ovector_base** &x)
  Fix parameters from an array for fitting.
- virtual int guess_fun (size_t nv, **ovector_base** &x)
  Fill array with guess from present values for fitting.
- virtual double drip_binding_energy_d (double Z, double N, double npout, double nnout, double chi)
  Return the free binding energy of a nucleus in a many-body environment.
- virtual double drip_binding_energy_full_d (double Z, double N, double npout, double nnout, double chi, double T)
  Return the free binding energy of a nucleus in a many-body environment.

Data Fields

- bool full_surface
  If true, properly fix the surface for the pure neutron matter limit (default true)
- bool new_skin_mode
  If true, separately compute the skin for the bulk energy (default false)
- double doi
  Ratio of δ/I (default 0.8).
- double ss
  Surface symmetry energy (default 0.5)
- bool rel_vacuum
  If true, define the nuclear mass relative to the vacuum (default true)
- double Tchalf
  The critical temperature of isospin-symmetric matter in fm^{-1} (default $20.085/(\hbar c)$.)

Input parameters for temperature dependence

- double pp
  Exponent (default 1.25)
- double a0
  Coefficient (default 0.935)
- double a2
  Coefficient (default -5.1)
- double a4
  Coefficient (default -1.1)

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

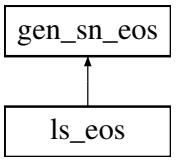
- ldrop_mass.h

14.24 ls_eos Class Reference

The Lattimer-Swesty supernova EOS.

`#include <gen_sn_eos.h>`

Inheritance diagram for ls_eos:



14.24.1 Detailed Description

This class is experimental.

Note

O_2scl Does not contain the Lattimer-Swesty EOS, only provides some code to manipulate it. This class is designed to be used with the files `ls.dat, sk1.dat, ska.dat` and `skm.dat` as provided on Jim Lattimer's website, `http://www.astro.sunysb.edu/lattimer/EOS/main.html` .

Note that the tables on this website are different than what is generated from the LS Fortran code. See oo_eos to read O'Connor and Ott's tables generated from the LS Fortran code.

The four models are

- LS (K=370, Sv=31)
- SKI' (K=371, Sv=30.4)
- SKa (K=263, Sv=34.5)
- SKM* (K=217, Sv=31.4)

Note

In the original table, the full internal energy per baryon (data section 4 of 26) is apparently based on a rest mass of $Y_e m_p + (1 - Y_e) m_n$, while the baryon part of the internal energy per baryon (data section 13 of 26) is based on a rest mass of m_n. This means that

$$E - E_{\rm int} = E_{\rm eg} - Y_e (m_n - m_p)$$

where $E_{\rm eg}$ is the energy per baryon of electrons and photons. In order to keep things consistent with the other EOS tables, when the EOS table is loaded, gen_sn_eos::Eint is rescaled to a rest mass of $Y_e m_p + (1 - Y_e) m_n$.

See also the documentation at gen_sn_eos.

See Lattimer91 and Lattimer85.

Todo There are still a few points for which the electron/photon EOS seems to be off.

Definition at line 371 of file gen_sn_eos.h.

Public Member Functions

- virtual void load (std::string fname)
  Load table from filename `fname`.
- int check_eg (**test_mgr** &tm)
  Check electrons and photons.
- virtual void beta_eq_T0 (size_t i, double &nb, double &E_beta, double &P_beta, double &Ye_beta, double &Z_beta, double &A_beta)
  Compute properties of matter in beta equilibrium at zero temperature at a baryon density grid point.

Data Fields

- **tensor_grid3** & fill
  Filling factor for nuclei.
- **tensor_grid3** & nb_in
  Baryon number density inside nuclei in fm^{-3}.
- **tensor_grid3** & dPdn
  Derivative of pressure with respect to baryon density.
- **tensor_grid3** & dPdT
  Derivative of pressure with respect to temperature.
- **tensor_grid3** & dPdY
  Derivative of pressure with respect to electron fraction.
- **tensor_grid3** & dsdT
  Derivative of entropy with respect to temperature.
- **tensor_grid3** & dsdY
  Derivative of entropy with respect to electron fraction.
- **tensor_grid3** & Nskin
  Number of neutrons in skin.
- **tensor_grid3** & nb_out
  Baryon density outside nuclei in fm^{-3}.
- **tensor_grid3** & x_out
  Proton fraction outside nuclei.
- **tensor_grid3** & mu
  Out of whackness parameter, $\mu_n - \mu_p - \mu_e + 1.293$ MeV*, in MeV.*

14.24.2 Member Function Documentation

14.24.2.1 int ls_eos::check_eg (test_mgr & *tm*)

This checks that the electron and photon thermodynamics generated by O_2scl is consistent with the data in `E`, `Eint`, `F`, `Fint`, `P`, `Pint`, `S`, and `Sint`.

14.24.2.2 virtual void ls_eos::beta_eq_T0 (size_t *i*, double & *nb*, double & *E_beta*, double & *P_beta*, double & *Ye_beta*, double & *Z_beta*, double & *A_beta*) [inline, virtual]

This EOS table doesn't have T=0 results, so we extrapolate from the two low-temperature grid points.

Implements gen_sn_eos.

Definition at line 432 of file gen_sn_eos.h.

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

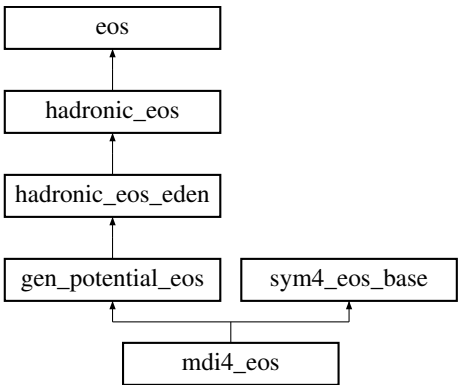
- gen_sn_eos.h

14.25 mdi4_eos Class Reference

A version of gen_potential_eos to separate potential and kinetic contributions.

`#include <sym4_eos.h>`

Inheritance diagram for mdi4_eos:



14.25.1 Detailed Description

References:

Created for Steiner06.

Definition at line 163 of file sym4_eos.h.

Public Member Functions

- virtual int calc_e_sep (**fermion** &ne, **fermion** &pr, double &ed_kin, double &ed_pot, double &mu_n_kin, double &mu_p_kin, double &mu_n_pot, double &mu_p_pot)
  Compute the potential and kinetic parts separately.
- virtual int test_separation (**fermion** &ne, **fermion** &pr, **test_mgr** &t)
  Test the separation of the potential and kinetic energy parts.

Protected Member Functions

- double energy_kin (double var)
  Compute the kinetic part of the energy density.
- double energy_pot (double var)
  Compute the potential part of the energy density.

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

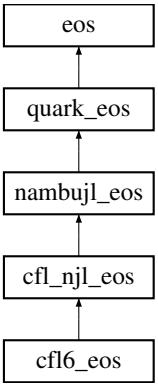
- sym4_eos.h

14.26 nambujl_eos Class Reference

Nambu Jona-Lasinio EOS at zero temperature.

`#include <nambujl_eos.h>`

Inheritance diagram for nambujl_eos:



14.26.1 Detailed Description

Calculates everything from the quark condensates ([uds].qq) and the chemical potentials ([uds].mu). If "fromqq" is set to false, then instead it calculates everything from the dynamical masses ([uds].ms) and the chemical potentials. L, G, K, and B0 are fixed constants. [uds].pr returns the pressure due to the Fermi-gas contribution plus the bag pressure contribution. [uds.ed] is the energy density for each quark so that e.g. u.ed+u.pr=u.mu∗u.n. B0 should be fixed using calc_B0() beforehand to ensure that the energy density and pressure of the vacuum is zero.

The functions set_parameters() should be called first.

The code is based on Buballa99.

The Lagrangian is

$$\mathscr{L} = \bar{q}(i\partial\!\!\!/ - \hat{m}_0)q \; + \; G\sum_{k=0}^{8}[(\bar{q}\lambda_k q)^2 + (\bar{q}i\gamma_5\lambda_k q)^2] + \mathscr{L}_6$$

$$\mathscr{L}_6 = -K\left[\det{}_f(\bar{q}(1+\gamma_5)q) + \det{}_f(\bar{q}(1-\gamma_5)q)\right].$$

And the corresponding thermodynamic potential is

$$\Omega = \Omega_{FG} + \Omega_{Int}$$

where Ω_{FG} is the Fermi gas contribution and

$$\frac{\Omega_{\rm Int}}{V} = -2N_c \sum_{i=u,d,s} \int \frac{d^3p}{(2\pi)^3}\sqrt{m_i^2+p^2} + \frac{\Omega_V}{V}$$

$$\frac{\Omega_V}{V} = \sum_{i=u,d,s} 2G\langle\bar{q}_i q_i\rangle^2 - 4K\langle\bar{q}_u q_u\rangle\langle\bar{q}_d q_d\rangle\langle\bar{q}_s q_s\rangle + B_0\,.$$

where B_0 is a constant defined to ensure that the energy density and the pressure of the vacuum is zero.

Unlike Buballa99, the bag constant, Ω_{Int}/V is defined without the term

$$\sum_{i=u,d,s} 2N_C \int_0^\Lambda \frac{d^3p}{(2\pi)^3}\sqrt{m_{0,i}^2+p^2}\,dp$$

since this allows an easier comparison to the finite temperature EOS. The constant B_0 in this case is therefore significantly larger, but the energy density and pressure are still zero in the vacuum.

The Feynman-Hellman theorem (Bernard88), gives

$$\langle\bar{q}q\rangle = \frac{\partial m^*}{\partial m}$$

The functions calc_e() and calc_p() never return a value other than zero, but will give nonsensical results for nonsensical inputs.

Finite T documentation

Calculates everything from the quark condensates ([uds].qq) and the chemical potentials ([uds].mu). If "fromqq" is set to false, then instead it calculates everything from the dynamical masses ([uds].ms) and the chemical potentials. L, G, K, and B0 are fixed constants. [uds].pr returns the pressure due to the Fermi-gas contribution plus the bag pressure contribution. [uds.ed] is the energy density for each quark so that e.g. u.ed+u.pr=u.mu∗u.n. B0 is fixed to ensure that the energy density and pressure of the vacuum is zero.

This implementation includes contributions from antiquarks.

References:

Created for Steiner00. See also Buballa99 and Hatsuda94.

Definition at line 128 of file nambujl_eos.h.

Data Structures

- struct njtp_s
  A structure for passing parameters to the integrands.

Public Types

- typedef struct nambujl_eos::njtp_s njtp
  A structure for passing parameters to the integrands.

Public Member Functions

- virtual int set_parameters (double lambda=0.0, double fourferm=0.0, double sixferm=0.0)
  Set the parameters and the bag constant `B0`.
- virtual int calc_p (**quark** &u, **quark** &d, **quark** &s, **thermo** &lth)
  Equation of state as a function of chemical potentials.
- virtual int calc_temp_p (**quark** &u, **quark** &d, **quark** &s, double T, **thermo** &th)
  Equation of state as a function of chemical potentials at finite temperature.
- virtual int calc_eq_p (**quark** &u, **quark** &d, **quark** &s, double &gap1, double &gap2, double &gap3, **thermo** &lth)
  Equation of state and gap equations as a function of chemical potential.
- virtual int calc_eq_e (**quark** &u, **quark** &d, **quark** &s, double &gap1, double &gap2, double &gap3, **thermo** &lth)
  Equation of state and gap equations as a function of the densities.
- int calc_eq_temp_p (**quark** &tu, **quark** &td, **quark** &ts, double &gap1, double &gap2, double &gap3, **thermo** &qb, double temper)

Equation of state and gap equations as a function of chemical potentials.

- int gapfunms (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Calculates gap equations in `y` *as a function of the constituent masses in* `x`*.*
- int gapfunqq (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Calculates gap equations in `y` *as a function of the quark condensates in* `x`*.*
- int gapfunmsT (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Calculates gap equations in `y` *as a function of the constituent masses in* `x`*.*
- int gapfunqqT (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Calculates gap equations in `y` *as a function of the quark condensates in* `x`*.*
- int set_quarks (**quark** &u, **quark** &d, **quark** &s)
  Set the quark objects to use.
- virtual const char * type ()
  Return string denoting type ("nambujl_eos")
- virtual int set_solver (**mroot**< **mm_funct**<> > &s)
  Set solver to use in set_parameters()
- virtual int set_inte (**inte**< **funct** > &i)
  Set integration object.

Data Fields

- double limit
  Accuracy limit for Fermi integrals for finite temperature.
- bool fromqq
  Calculate from quark condensates if true (default true)
- double L
  The momentum cutoff.
- double G
  The four-fermion coupling.
- double K
  The 't Hooft six-fermion interaction coupling.
- double B0
  The bag constant.
- **gsl_mroot_hybrids**< **mm_funct**<> > def_solver
  The default solver.
- **gsl_inte_qag**< **funct** > def_it
  The default integrator.

The default quark masses

These are the values from Buballa99 which were used to fix the pion and kaon decay constants, and the pion, kaon, and eta prime masses. They are set in the constructor and are in units of fm^{-1} .

- double **up_default_mass**
- double **down_default_mass**
- double **strange_default_mass**

The default quark objects

The masses are automatically set in the constructor to `up_default_mass`*,* `down_default_mass`*, and* `strange_default_mass.c`

- **quark def_up**
- **quark def_down**
- **quark def_strange**

Protected Member Functions

- int B0fun (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  Used by calc_B0() to compute the bag constant.
- void njbag (**quark** &q)
  Calculates the contribution to the bag constant from quark q.
- double iqq (double x, const njtp &pa)
  The integrand for the quark condensate.
- double ide (double x, const njtp &pa)
  The integrand for the density.
- double ied (double x, const njtp &pa)
  The integrand for the energy density.
- double ipr (double x, const njtp &pa)
  The integrand for the pressure.

Protected Attributes

- **inte**< **funct** > * it
  The integrator for finite temperature integrals.
- **mroot**< **mm_funct**<> > * solver
  The solver to use for set_parameters()
- **quark** * up
  The up quark.
- **quark** * down
  The down quark.
- **quark** * strange
  The strange quark.
- double cp_temp
  The temperature for calc_temp_p()

14.26.2 Member Function Documentation

14.26.2.1 virtual int nambujl_eos::set_parameters (double *lambda* = 0.0, double *fourferm* = 0.0, double *sixferm* = 0.0) [virtual]

This function allows the user to specify the momentum cutoff, `lambda`, the four-fermion coupling `fourferm` and the six-fermion coupling from the 't Hooft interaction `sixferm` . If 0.0 is given for any of the values, then the default is used ($\Lambda = 602.3/(\hbar c), G = 1.835/\Lambda^2, K = 12.36/\Lambda^5$).

The value of the shift in the bag constant `B0` is automatically calculated to ensure that the energy density and the pressure of the vacuum are zero. The functions set_quarks() and set_thermo() can be used before hand to specify the **quark** and **thermo** objects.

14.26.2.2 virtual int nambujl_eos::calc_p (**quark** & *u*, **quark** & *d*, **quark** & *s*, **thermo** & *lth*) [virtual]

This function automatically solves the gap equations

Reimplemented from quark_eos.

14.26.2.3 virtual int nambujl_eos::calc_temp_p (**quark** & *u*, **quark** & *d*, **quark** & *s*, double *T*, **thermo** & *th*) [virtual]

This function automatically solves the gap equations

Reimplemented from quark_eos.

14.26.2.4 int nambujl_eos::gapfunms (size_t *nv*, const **ovector_base** & *x*, **ovector_base** & *y*)

The function utilizes the **quark** objects which can be specified in set_quarks() and the **thermo** object which can be specified in eos::set_thermo().

14.26.2.5 int nambujl_eos::gapfunqq (size_t *nv*, const ovector_base & *x*, ovector_base & *y*)

The function utilizes the **quark** objects which can be specified in set_quarks() and the **thermo** object which can be specified in eos::set_thermo().

14.26.2.6 int nambujl_eos::gapfunmsT (size_t *nv*, const ovector_base & *x*, ovector_base & *y*)

The function utilizes the **quark** objects which can be specified in set_quarks() and the **thermo** object which can be specified in eos::set_thermo().

14.26.2.7 int nambujl_eos::gapfunqqT (size_t *nv*, const ovector_base & *x*, ovector_base & *y*)

The function utilizes the **quark** objects which can be specified in set_quarks() and the **thermo** object which can be specified in eos::set_thermo().

14.26.2.8 int nambujl_eos::set_quarks (quark & *u*, quark & *d*, quark & *s*)

The quark objects are used in gapfunms(), gapfunqq(), gapfunmsT(), gapfunqqT(), and B0fun().

14.26.3 Field Documentation

14.26.3.1 double nambujl_eos::limit

limit is used for the finite temperature integrals to ensure that no numbers larger than exp(limit) or smaller than exp(-limit) are avoided. (Default: 20)

Definition at line 156 of file nambujl_eos.h.

14.26.3.2 bool nambujl_eos::fromqq

If this is false, then computations are performed using the effective masses as inputs

Definition at line 163 of file nambujl_eos.h.

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

- nambujl_eos.h

14.27 nambujl_eos::njtp_s Struct Reference

A structure for passing parameters to the integrands.

`#include <nambujl_eos.h>`

14.27.1 Detailed Description

Definition at line 290 of file nambujl_eos.h.

Data Fields

- double **ms**
- double **m**
- double **mu**
- double **temper**
- double **limit**

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

- nambujl_eos.h

14.28 nse_eos Class Reference

Equation of state for nuclei in statistical equilibrium.

`#include <nse_eos.h>`

14.28.1 Detailed Description

This class computes the composition of matter in nuclear statistical equilibrium. The chemical potential of a nucleus X with proton number Z_X and neutron number N_X is given by

$$\mu_X = N\mu_n + Z\mu_p - E_{\mathrm{bind},X}$$

where μ_n and μ_p are the neutron and proton chemical potentials and $E_{\mathrm{bind},X}$ is the binding energy of the nucleus.

The baryon number density and electron fraction are then given by

$$n_B = n_X(N_X + Z_X) \qquad Y_e n_B = n_X Z_X$$

where n_X is the number density which is determined from the chemical potential above.

This implicitly assumes that the nuclei are non-interacting.

Idea for Future Right now calc_density() needs a very good guess. This could be fixed, probably by solving for the log(mu/T) instead of mu.

Definition at line 62 of file nse_eos.h.

Public Member Functions

- int calc_mu (double mun, double mup, double T, double &nb, double &Ye, **thermo** &th, **nuclear_dist** &nd)
  Calculate the equation of state as a function of the chemical potentials.
- int calc_density (double nb, double Ye, double T, double &mun, double &mup, **thermo** &th, **nuclear_dist** &nd)
  Calculate the equation of state as a function of the densities.
- int set_mroot (**mroot**< **mm_funct**<> > &rp)
  Set the solver for use in computing the chemical potentials.

Data Fields

- **gsl_mroot_hybrids**< **mm_funct**<> > def_root
  Default solver.

14.28.2 Member Function Documentation

14.28.2.1 int nse_eos::calc_mu (double *mun*, double *mup*, double *T*, double & *nb*, double & *Ye*, thermo & *th*, nuclear_dist & *nd*)

Given `mun`, `mup` and `T`, this computes the composition (the individual densities are stored in the distribution `nd`) the baryon number density `nb`, and the electron fraction `Ye`.

This function does not use the solver.

14.28.2.2 int nse_eos::calc_density (double *nb*, double *Ye*, double *T*, double & *mun*, double & *mup*, thermo & *th*, nuclear_dist & *nd*)

Given the baryon number density `nb`, and the electron fraction `Ye` and the temperature `T`, this computes the composition (the individual densities are stored in the distribution `nd`) and the chemical potentials are given in `mun` and `mup` .

This function uses the solver to self-consistently compute the chemical potentials.

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

- nse_eos.h

14.29 rmf_nucleus::odparms Struct Reference

A convenient struct for the solution of the Dirac equations.

`#include <rmf_nucleus.h>`

14.29.1 Detailed Description

Definition at line 244 of file rmf_nucleus.h.

Data Fields

- double eigen
  Eigenvalue.
- double kappa
  Quantum number κ .
- **umatrix** * fields
  The meson fields.
- **uvector** * varr
  Desc.

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

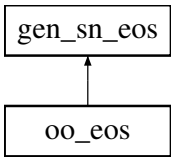
- rmf_nucleus.h

14.30 oo_eos Class Reference

The EOS tables from O'Connor and Ott.

`#include <gen_sn_eos.h>`

Inheritance diagram for oo_eos:



14.30.1 Detailed Description

This class reads the HDF5 EOS tables generated by E. O'Connor and C. Ott in OConnor10. The tables are available from

http://stellarcollapse.org/equationofstate

and are available under a creative commons attribution-noncommercial-share alike license. This O_2scl code to read those tables is licensed (along with all O_2scl code) under the GPLv3 license (with permission from Evan O'Connor).

The original README file from O'Connor and Ott's EOSdriver code is available in the O_2scl distribution in `doc/o2scl/eos/extras/scol_README` and is reproduced below

```
! ########################################################
!
! Copyright C. D. Ott and E. O'Connor, July 2009
!
!
!
! EOS constructed on the basis of the Shen et al. 1998
! relativistic-mean field nuclear EOS table. Electrons (fully
! general, based on TimmesEOS) and Photons added.
!
! Original Shen EOS table extent:
!
! Density:  10^5.1 to 10^15.4 g/cm^3
! Temperature: 0.1 to 100 MeV
! Ye: 0.01 to 0.56
!
! Table extent of current table
! (myshen_test_220r_180t_50y_extT_20090312.h5)
!
! Density: 10^3 to 10^15.36 g/cm^3
! Temperature: 0.01 to 250 MeV
! Ye: 0.015 to 0.56 MeV
!
! This bigger table is realized by extending the
! original Shen table in multiple ways in multiple
! directions:
!
! (a) density:
! Match of pure ideal gas of Ni56 + electrons/positrons +
! photons at densities below 10^7 g/ccm -- at this density
! pressures, energies and entropies match okayish with the
! values in the Shen table. The compositions (A,Z,xh,xa,xp,xn)
! are kept constant in the low-density region and n and p
! chemical potentials are set to 0.
! -- ideally, at low densities, a full NSE EOS with nuclear
! reaction network (at low T) should be stiched onto the
! Shen; working on that, but not yet ready.
!
! (b) temperature (extrapolation):
! At high density: linear extrapolation of everything in T to lower
! temperatures and higher temperatures. At low densities (below 10^7
! g/ccm), ideal gas of Ni56 + electrons/positrons + photons.
!
!
! Variable         Units             Description
! pointsrho    dimensionless     number of table points
!           in log_10(rho)
! pointstemp    dimensionless     number of table points
!           in log_10(temp)
! pointsye         dimensionless     number of table point in Y_e
! logrho    log10(rho)      index variable rho
! logtemp    log10(MeV)      index variable temperature
! ye    number fraction   index variable electron fraction
! Abar             A                 average heavy nucleus A
! Zbar             Z                 average heavy nucleus Z
! Xa               number fraction   alpha particle number frac
! Xh               number fraction   heavy nucleos number frac
! Xn               number fraction   neutron number frac
! Xn    number fraction   proton number frac
! cs2              cm^2/s^2          speed of sound squared
! dedt             erg/g/K      C_v
! dpderho    dyn g/cm^2/erg    dp/deps|rho
! dpdrhoe    dyn cm^3/cm^2/g   dp/drho|eps
```

```
! energy_shift     erg/g      Energy shift for table storage
! entropy    k_B/baryon      specific entropy
! gamma    dimensionless     Gamma_1
! logenergy    log10(erg/g)      specific internal energy
! logpress    log10(dyn/cm^2)   pressure
! mu_e    MeV/baryon      electron chemical potential
!           INCLUDING electron rest mass
! mu_p    MeV/baryon      proton chemical potential
! mu_n    MeV/baryon      neutron chemical potential
! muhat    MeV/baryon      mu_n - mu_p
! munu    MeV/baryon      mu_e - mun + mu_p
!
!
! * energy shift:
!
! In some regions the negative nuclear binding energy
! is larger in magnitude than the thermal/excitation energy. In this
! case the specific internal energy (eps) becomes negative. To allow for
! storage and interpolation of eps in logarithmic fashion, the energy
! is shifted up by an energy shift specified in the variable "energy_shift".
! This energy shift is handled internally in the EOS routines.
```

Definition at line 500 of file gen_sn_eos.h.

Public Member Functions

- virtual void load (std::string fname, size_t mode)
  Load table from filename `fname`.
- virtual void load (std::string fname)
  Load table from filename `fname`.
- virtual void beta_eq_T0 (size_t i, double &nb, double &E_beta, double &P_beta, double &Ye_beta, double &Z_beta, double &A_beta)
  Compute properties of matter in beta equilibrium at zero temperature at a baryon density grid point.

Data Fields

- **tensor_grid** & cs2
  Speed of sound in cm^2/s^2.
- **tensor_grid** & dedt
  C_V in erg/g/K.
- **tensor_grid** & dpderho
  *dpderho in dyn*g/cm^2/erg*
- **tensor_grid** & dpdrhoe
  dpdrhoe in dyn cm^3/cm^2/g
- **tensor_grid** & gamma
  Gamma.
- **tensor_grid** & mu_e
  Electron chemical potential per baryon including rest mass.
- **tensor_grid** & muhat
  mun - mup
- **tensor_grid** & munu
  mue - mun + mup
- **ovector** rho
  The original mass density grid from the table in g/cm^3.
- double energy_shift
  Energy shift for table storage in erg/g.

Static Public Attributes

- static const size_t ls_mode = 0
  Use the J. Lattimer et al. method for handling the chemical potentials.
- static const size_t stos_mode = 1
  Use the H. Shen et al. method for handling the chemical potentials.

14.30.2 Member Function Documentation

14.30.2.1 virtual void oo_eos::beta_eq_T0 (size_t *i*, double & *nb*, double & *E_beta*, double & *P_beta*, double & *Ye_beta*, double & *Z_beta*, double & *A_beta*) [inline, virtual]

This EOS table doesn't have T=0 results, so we extrapolate from the two low-temperature grid points.

Implements gen_sn_eos.

Definition at line 556 of file gen_sn_eos.h.

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

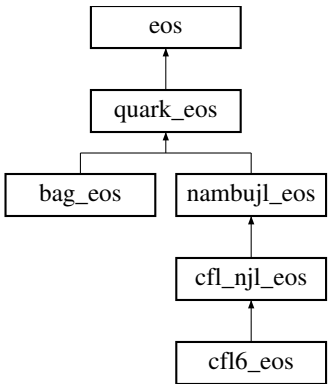
- gen_sn_eos.h

14.31 quark_eos Class Reference

Quark matter equation of state base.

`#include <quark_eos.h>`

Inheritance diagram for quark_eos:



14.31.1 Detailed Description

Definition at line 39 of file quark_eos.h.

Public Member Functions

- virtual int calc_p (**quark** &u, **quark** &d, **quark** &s, **thermo** &th)
  Calculate equation of state as a function of chemical potentials.
- virtual int calc_e (**quark** &u, **quark** &d, **quark** &s, **thermo** &th)
  Calculate equation of state as a function of density.
- virtual int calc_temp_p (**quark** &u, **quark** &d, **quark** &s, double temper, **thermo** &th)
  Calculate equation of state as a function of chemical potentials.

- virtual int calc_temp_e (**quark** &u, **quark** &d, **quark** &s, double temper, **thermo** &th)
  Calculate equation of state as a function of density.
- virtual const char * type ()
  Return string denoting type ("quark_eos")

Data Fields

- fermion_eval_thermo * fet
  Desc.
- **eff_fermion** def_fet
  Desc.

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

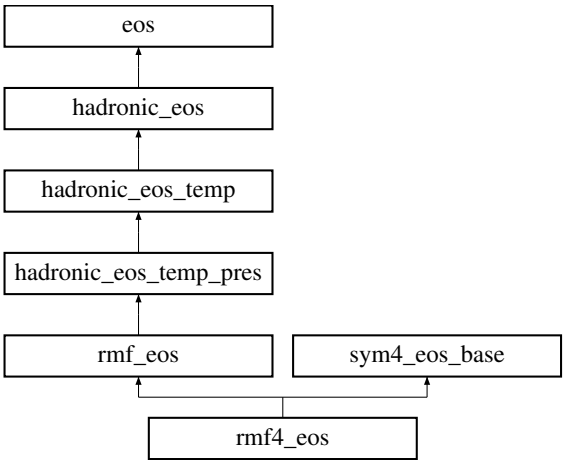
- quark_eos.h

14.32 rmf4_eos Class Reference

A version of rmf_eos to separate potential and kinetic contributions.

`#include <sym4_eos.h>`

Inheritance diagram for rmf4_eos:



14.32.1 Detailed Description

References:

Created for Steiner06.

Definition at line 103 of file sym4_eos.h.

Public Member Functions

- virtual int calc_e_sep (**fermion** &ne, **fermion** &pr, double &ed_kin, double &ed_pot, double &mu_n_kin, double &mu_p_kin, double &mu_n_pot, double &mu_p_pot)
  Compute the potential and kinetic parts separately.

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

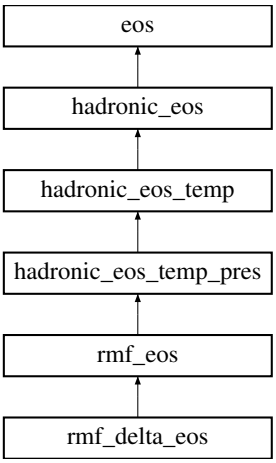
- sym4_eos.h

14.33 rmf_delta_eos Class Reference

Field-theoretical EOS with scalar-isovector meson, δ.

`#include <rmf_delta_eos.h>`

Inheritance diagram for rmf_delta_eos:



14.33.1 Detailed Description

This essentially follows the notation in Kubis97, except that our definitions of `b` and `c` follow their $\bar{b}$ and $\bar{c}$, respectively.

Also discussed in Gaitanos04, where they take $m_\delta = 980$ MeV.

The full Lagragian is:

$$\mathscr{L} = \mathscr{L}_{Dirac} + \mathscr{L}_\sigma + \mathscr{L}_\omega + \mathscr{L}_\rho + \mathscr{L}_\delta$$

$$\begin{aligned}
\mathscr{L}_{Dirac} &= \bar{\Psi}\left[i\partial\!\!\!/ - g_\omega \omega\!\!\!/ - \frac{g_\rho}{2}\vec{\rho}\!\!\!/\ \vec{\tau} - M + g_\sigma\sigma - \frac{e}{2}(1+\tau_3)A_\mu\right]\Psi \\
\mathscr{L}_\sigma &= \tfrac{1}{2}\left(\partial_\mu\sigma\right)^2 - \tfrac{1}{2}m_\sigma^2\sigma^2 - \frac{bM}{3}\left(g_\sigma\sigma\right)^3 - \frac{c}{4}\left(g_\sigma\sigma\right)^4 \\
\mathscr{L}_\omega &= -\tfrac{1}{4}f_{\mu\nu}f^{\mu\nu} + \tfrac{1}{2}m_\omega^2\omega^\mu\omega_\mu + \frac{\zeta}{24}g_\omega^4\left(\omega^\mu\omega_\mu\right)^2 \\
\mathscr{L}_\rho &= -\tfrac{1}{4}\vec{B}_{\mu\nu}\cdot\vec{B}^{\mu\nu} + \tfrac{1}{2}m_\rho^2\vec{\rho}^{\ \mu}\cdot\vec{\rho}_{\ \mu} + \frac{\xi}{24}g_\rho^4\left(\vec{\rho}^{\ \mu}\right)\cdot\vec{\rho}_{\ \mu} + g_\rho^2 f(\sigma,\omega)\vec{\rho}^{\ \mu}\cdot\vec{\rho}_{\ \mu}
\end{aligned}$$

where the additional terms are

$$\mathscr{L}_\delta = \bar{\Psi}\left(g_\delta\vec{\delta}\cdot\vec{\tau}\right)\Psi + \frac{1}{2}(\partial_\mu\vec{\delta})^2 - \frac{1}{2}m_\delta^2\vec{\delta}^{\ 2}$$

The new field equation for the delta meson is

$$m_\delta^2\delta = g_\delta(n_{s,p} - n_{s,n})$$

Idea for Future Finish the finite temperature EOS

Definition at line 92 of file rmf_delta_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &ne, **fermion** &pr, **thermo** &lth)
  Equation of state as a function of density.
- virtual int calc_p (**fermion** &neu, **fermion** &p, double sig, double ome, double rho, double delta, double &f1, double &f2, double &f3, double &f4, **thermo** &th)
  Equation of state as a function of chemical potentials.
- int calc_temp_p (**fermion** &ne, **fermion** &pr, double temper, double sig, double ome, double lrho, double delta, double &f1, double &f2, double &f3, double &f4, **thermo** &lth)
  Finite temperature (unfinished)
- virtual int set_fields (double sig, double ome, double lrho, double delta)
  Set a guess for the fields for the next call to calc_e(), calc_p(), or saturation()
- virtual int saturation ()
  Calculate saturation properties for nuclear matter at the saturation density.

Data Fields

- double md
  The mass of the scalar-isovector field.
- double cd
  The coupling of the scalar-isovector field to the nucleons.
- double del
  The value of the scalar-isovector field.

Protected Member Functions

- virtual int calc_e_solve_fun (size_t nv, const **ovector_base** &ex, **ovector_base** &ey)
  The function for calc_e()
- virtual int zero_pressure (size_t nv, const **ovector_base** &ex, **ovector_base** &ey)
  Compute matter at zero pressure (for saturation())

Private Member Functions

- virtual int set_fields (double sig, double ome, double lrho)
  Forbid setting the guesses to the fields unless all four fields are specified.

14.33.2 Member Function Documentation

14.33.2.1 virtual int rmf_delta_eos::saturation ( ) [virtual]

This requires initial guesses to the chemical potentials, etc.

Reimplemented from rmf_eos.

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

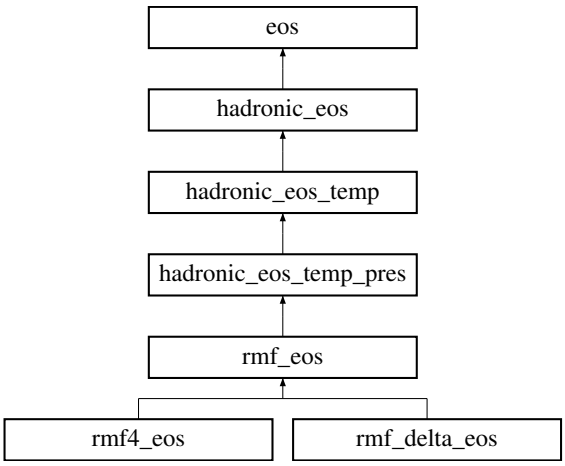
- rmf_delta_eos.h

14.34 rmf_eos Class Reference

Relativistic mean field theory EOS.

`#include <rmf_eos.h>`

Inheritance diagram for rmf_eos:



14.34.1 Detailed Description

This class computes the properties of nucleonic matter using a mean-field approximation to a field-theoretical model.

Before sending neutrons and protons to these member functions, the masses should be set to their vacuum values and the degeneracy factor should be 2. If an internal model is used (using load()), then the neutron and proton masses should be set to mnuc.

The expressions for the energy densities are often simplified in the literature using the field equations. These expressions are not used in this code since they are only applicable in infinite matter where the field equations hold, and are not suitable for use in applications (such as to finite nuclei in rmf_nucleus) where the spatial derivatives of the fields are non-zero. Notice that in the proper expressions for the energy density the similarity between terms in the pressure up to a sign. This procedure allows one to verify the thermodynamic identity even if the field equations are not solved and allows the user to add gradient terms to the energy density and pressure.

Note

Since this EOS uses the effective masses and chemical potentials in the fermion class, the values of **part::non_interacting** for neutrons and protons are set to false in many of the functions.

Todo

- The functions fcomp_fields(), fkprime_fields(), and fesym_fields() are not quite correct if the neutron and proton masses are different. For this reason, they are currently unused by saturation().
- The fix_saturation() and calc_cr() functions use mnuc, and should be modified to allow different neutron and proton masses.
- Check the formulas in the "Background" section
- There are two calc_e() functions that solve. One is specially designed to work without a good initial guess. Possibly the other calc_e() function should be similarly designed?
- Make sure that this class properly handles particles for which inc_rest_mass is true/false
- The error handler is called sometimes when calc_e() is used to compute pure neutron matter. This should be fixed.
- Decide whether to throw an error at [Ref. 1].
- Put the err_nonconv system into calc_p(), calc_temp_e() and fix_saturation(), etc.

Idea for Future

- It might be nice to remove explicit reference to the meson masses in functions which only compute nuclear matter since they are unnecessary. This might, however, demand redefining some of the couplings.
- Fix calc_p() to be better at guessing
- The number of couplings is getting large, maybe new organization is required.
- Overload hadronic_eos::fcomp() with an exact version

Background

The full Lagragian is:

$$\mathscr{L} = \mathscr{L}_{Dirac} + \mathscr{L}_{\sigma} + \mathscr{L}_{\omega} + \mathscr{L}_{\rho}$$

$$\begin{aligned}
\mathscr{L}_{Dirac} &= \bar{\Psi}\left[i\partial\!\!\!/ - g_{\omega}\omega\!\!\!/ - \frac{g_{\rho}}{2}\vec{\rho}\!\!\!/\,\vec{\tau} - M + g_{\sigma}\sigma - \frac{e}{2}(1+\tau_3)A_{\mu}\right]\Psi \\
\mathscr{L}_{\sigma} &= \tfrac{1}{2}\left(\partial_{\mu}\sigma\right)^2 - \tfrac{1}{2}m_{\sigma}^2\sigma^2 - \frac{bM}{3}\left(g_{\sigma}\sigma\right)^3 - \frac{c}{4}\left(g_{\sigma}\sigma\right)^4 \\
\mathscr{L}_{\omega} &= -\tfrac{1}{4}f_{\mu\nu}f^{\mu\nu} + \tfrac{1}{2}m_{\omega}^2\omega^{\mu}\omega_{\mu} + \frac{\zeta}{24}g_{\omega}^4\left(\omega^{\mu}\omega_{\mu}\right)^2 \\
\mathscr{L}_{\rho} &= -\tfrac{1}{4}\vec{B}_{\mu\nu}\cdot\vec{B}^{\mu\nu} + \tfrac{1}{2}m_{\rho}^2\vec{\rho}^{\,\mu}\cdot\vec{\rho}_{\,\mu} + \frac{\xi}{24}g_{\rho}^4\left(\vec{\rho}^{\,\mu}\right)\cdot\vec{\rho}_{\,\mu} + g_{\rho}^2 f(\sigma,\omega)\vec{\rho}^{\,\mu}\cdot\vec{\rho}_{\,\mu}
\end{aligned}$$

The couplings `cs`, `cw`, and `cr` are related to g_{σ}, g_{ω} and g_{ρ} above by

$$g_s = c_{\sigma}m_{\sigma} \quad g_w = c_{\omega}m_{\omega} \quad \text{and} \quad g_r = c_{\rho}m_{\rho}$$

The coefficients b and c are related to the somewhat standard κ and λ by:

$$\kappa = 2Mb \quad \lambda = 6c;$$

The function f is the coefficient of $g_r^2\rho^2$ $f(\sigma,\omega) = b_1\omega^2 + b_2\omega^4 + b_3\omega^6 + a_1\sigma + a_2\sigma^2 + a_3\sigma^3 + a_4\sigma^4 + a_5\sigma^5 + a_6\sigma^6$ where the notation from Horowitz01 is: $f(\sigma,\omega) = \lambda_4 g_s^2\sigma^2 + \lambda_v g_w^2\omega^2$ This implies $b_1 = \lambda_v g_w^2$ and $a_2 = \lambda_4 g_s^2$

The couplings, `cs`, `cw`, and `cr` all have units of fm, and the couplings `b`, `c`, `zeta` and `xi` are unitless. The additional couplings from Steiner05b, a_i have units of $\mathrm{fm}^{(i-2)}$ and the couplings b_j have units of $\mathrm{fm}^{(2*j-2)}$.

The field equations are:

$$0 = m_{\sigma}^2\sigma - g_{\sigma}\left(n_{sn} + n_{sp}\right) + bMg_{\sigma}^3\sigma^2 + cg_{\sigma}^4\sigma^3 - g_{\rho}^2\rho^2\frac{\partial f}{\partial\sigma}$$

$$0 = m_{\omega}^2\omega - g_{\omega}\left(n_n + n_p\right) + \frac{\zeta}{6}g_{\omega}^4\omega^3 + g_{\rho}^2\rho^2\frac{\partial f}{\partial\omega}$$

$$0 = m_{\rho}^2\rho + \frac{1}{2}g_{\rho}\left(n_n - n_p\right) + 2g_{\rho}^2\rho f + \frac{\xi}{6}g_{\rho}^4\rho^3$$

When the variable zm_mode is true, the effective mass is fixed using the approach of Zimanyi90 .

Defining

$$U(\sigma) = \frac{1}{2}m_{\sigma}^2\sigma^2 + \frac{bM}{3}(g_{\sigma}\sigma)^3 + \frac{c}{4}(g_{\sigma}\sigma)^4\,,$$

the binding energy per particle in symmetric matter at equilibrium is given by

$$\frac{E}{A} = \frac{1}{n_0}\left[U(\sigma_0) + \frac{1}{2}m_{\omega}\omega_0^2 + \frac{\zeta}{8}(g_{\omega}\omega_0)^4 + \frac{2}{\pi^2}\int_0^{k_F} dk k^2\sqrt{k^2 + M^{*2}}\right]\,.$$

---

where the Dirac effective mass is $M^* = M - g_\sigma \sigma_0$. The compressibility is given by

$$K = 9\frac{g_\omega^2}{m_\omega^2} n_0 + 3\frac{k_F^2}{E_F^*} - 9 n_0 \frac{M^{*2}}{E_F^{*2}} \left[\left(\frac{1}{g_\sigma^2} \frac{\partial^2}{\partial \sigma_0^2} + \frac{3}{g_\sigma M^*} \frac{\partial}{\partial \sigma_0} \right) U(\sigma_0) - 3\frac{n_0}{E_F^*} \right]^{-1} .$$

The symmetry energy of bulk matter is given by

$$E_{sym} = \frac{k_F^2}{6E_F^*} + \frac{n}{8\left(g_\rho^2/m_\rho^2 + 2f(\sigma_0, \omega_0)\right)}$$

In the above equations, the subscipt 0 denotes the mean field values of σ and ω . For the case $f = 0$, the symmetry energy varies linearly with the density at large densities. The function f permits variations in the density dependence of the symmetry energy above nuclear matter density.

See also Muller96, Zimanyi90, and Steiner05b.

Definition at line 221 of file rmf_eos.h.

Public Member Functions

- virtual const char * type ()
  Return string denoting type ("rmf_eos")
- int check_naturalness (rmf_eos &re)
  Set the coefficients of a rmf_eos object to their limits from naturalness.
- int naturalness_limits (double value, rmf_eos &re)
  Provide the maximum values of the couplings assuming a limit on naturalness.

Compute EOS

- virtual int calc_e (**fermion** &ne, **fermion** &pr, **thermo** &lth)
  Equation of state as a function of density.
- virtual int calc_e_fields (**fermion** &ne, **fermion** &pr, **thermo** &lth, double &sig, double &ome, double &rho)
  Equation of state as a function of density returning the meson fields.
- virtual int calc_p (**fermion** &ne, **fermion** &pr, **thermo** &lth)
  Equation of state as a function of chemical potential.
- virtual int calc_eq_p (**fermion** &neu, **fermion** &p, double sig, double ome, double rho, double &f1, double &f2, double &f3, **thermo** &th)
  Equation of state and meson field equations as a function of chemical potentials.
- virtual int calc_eq_temp_p (**fermion** &ne, **fermion** &pr, double temper, double sig, double ome, double rho, double &f1, double &f2, double &f3, **thermo** &th)
  Equation of state and meson field equations as a function of chemical potentials at finite temperature.
- virtual int calc_temp_p (**fermion** &ne, **fermion** &pr, double T, **thermo** &lth)
  Equation of state as a function of chemical potential.
- int calc_temp_e (**fermion** &ne, **fermion** &pr, double T, **thermo** &lth)
  Equation of state as a function of densities at finite temperature.

Saturation properties

- int fix_saturation (double guess_cs=4.0, double guess_cw=3.0, double guess_b=0.001, double guess_c=-0.001)
  Calculate cs, cw, cr, b, and c from the saturation properties.
- virtual int saturation ()
  Calculate properties of nuclear matter at the saturation density.
- double fesym_fields (double sig, double ome, double nb)
  Calculate symmetry energy assuming the field equations have already been solved.
- double fcomp_fields (double sig, double ome, double nb)
  Calculate the compressibility assuming the field equations have already been solved.
- int fkprime_fields (double sig, double ome, double nb, double &k, double &kprime)
  Calculate compressibilty and `kprime` assuming the field equations have already been solved.

Fields and field equations

- int field_eqs (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  A function for solving the field equations.
- int field_eqsT (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  A function for solving the field equations at finite temperature.
- virtual int set_fields (double sig, double ome, double lrho)
  Set a guess for the fields for the next call to calc_e(), calc_p(), or saturation()
- int get_fields (double &sig, double &ome, double &lrho)
  Return the most recent values of the meson fields.

Data Fields

- bool zm_mode
  Modifies method of calculating effective masses (default false)
- int verbose
  Verbosity parameter.
- bool err_nonconv
  If true, throw exceptions when the function calc_e() does not converge (default true)
- int last_conv
  The convergence status of the last call to calc_e()

Masses

- double mnuc
  nucleon mass
- double ms
  σ *mass (in* fm^{-1} *)*
- double mw
  ω *mass (in* fm^{-1} *)*
- double mr
  ρ *mass (in* fm^{-1} *)*

Standard couplings (including nonlinear sigma terms)

- double **cs**
- double **cw**
- double **cr**
- double **b**
- double **c**

Non-linear terms for omega and rho.

- double **zeta**
- double **xi**

Additional isovector couplings

- double **a1**
- double **a2**
- double **a3**
- double **a4**
- double **a5**
- double **a6**
- double **b1**
- double **b2**
- double **b3**

Protected Member Functions

- int fix_saturation_fun (size_t nv, const **ovector_base** &x, **ovector_base** &y)
  The function for fix_saturation()
- virtual int zero_pressure (size_t nv, const **ovector_base** &ex, **ovector_base** &ey)

Compute matter at zero pressure (for saturation())
- virtual int calc_e_solve_fun (size_t nv, const **ovector_base** &ex, **ovector_base** &ey)
  The function for calc_e()
- virtual int calc_temp_e_solve_fun (size_t nv, const **ovector_base** &ex, **ovector_base** &ey)
  The function for calc_temp_e()
- int calc_cr (double sig, double ome, double nb)
  Calculate the `cr` coupling given `sig` and `ome` at the density 'nb'.

Protected Attributes

- double n_baryon
  Desc.
- double n_charge
  Temporary charge density.
- double fe_temp
  Temperature for solving field equations at finite temperature.
- bool ce_neut_matter
  For calc_e(), if true, then solve for neutron matter.
- bool ce_prot_matter
  For calc_e(), if true, then solve for proton matter.
- bool guess_set
  True if a guess for the fields has been given.
- **mroot**< **mm_funct**<> > * sat_mroot
  The solver to compute saturation properties.
- double ce_temp
  Temperature storage for calc_temp_e()

The meson fields

- double **sigma**
- double **omega**
- double **rho**

Solver

- **gsl_mroot_hybrids**< **mm_funct**<> > def_sat_mroot
  The default solver for calculating the saturation density.
- virtual int set_sat_mroot (**mroot**< **mm_funct**<> > &mrx)
  Set class mroot object for use calculating saturation density.

14.34.2 Member Function Documentation

14.34.2.1 virtual int rmf_eos::calc_e (**fermion** & *ne*, **fermion** & *pr*, **thermo** & *lth*) [virtual]

Initial guesses for the chemical potentials are taken from the user-given values. Initial guesses for the fields can be set by set_fields(), or default values will be used. After the call to calc_e(), the final values of the fields can be accessed through get_fields().

This is a little more robust than the standard version in the parent hadronic_eos.

Idea for Future Improve the operation of this function when the proton density is zero.

Reimplemented from hadronic_eos_temp_pres.

Reimplemented in rmf_delta_eos.

14.34.2.2 virtual int rmf_eos::calc_e_fields (fermion & *ne*, fermion & *pr*, thermo & *lth*, double & *sig*, double & *ome*, double & *rho*) [virtual]

Idea for Future Improve the operation of this function when the proton density is zero.

14.34.2.3 virtual int rmf_eos::calc_p (fermion & *ne*, fermion & *pr*, thermo & *lth*) [virtual]

Solves for the field equations automatically.

Note

This may not be too robust. Fix?

Implements hadronic_eos_temp_pres.

14.34.2.4 virtual int rmf_eos::calc_eq_p (fermion & *neu*, fermion & *p*, double *sig*, double *ome*, double *rho*, double & *f1*, double & *f2*, double & *f3*, thermo & *th*) [virtual]

This calculates the pressure and energy density as a function of $\mu_n, \mu_p, \sigma, \omega, rho$. When the field equations have been solved, `f1`, `f2`, and `f3` are all zero.

The thermodynamic identity is satisfied even when the field equations are not solved.

Idea for Future Probably best to have f1, f2, and f3 scaled in some sensible way, i.e. scaled to the fields?

14.34.2.5 virtual int rmf_eos::calc_eq_temp_p (fermion & *ne*, fermion & *pr*, double *temper*, double *sig*, double *ome*, double *rho*, double & *f1*, double & *f2*, double & *f3*, thermo & *th*) [virtual]

Analogous to calc_eq_p() except at finite temperature.

14.34.2.6 virtual int rmf_eos::calc_temp_p (fermion & *ne*, fermion & *pr*, double *T*, thermo & *lth*) [virtual]

Solves for the field equations automatically.

Implements hadronic_eos_temp_pres.

14.34.2.7 int rmf_eos::fix_saturation (double *guess_cs* = `4.0`, double *guess_cw* = `3.0`, double *guess_b* = `0.001`, double *guess_c* = `-0.001`)

Note that the meson masses and mnuc must be specified before calling this function.

This function does not give correct results when bool zm_mode is true.

`guess_cs`, `guess_cw`, `guess_b`, and `guess_c` are initial guesses for `cs`, `cw`, `b`, and `c` respectively.

Todo
- Fix this for zm_mode=true
- Ensure solver is more robust

14.34.2.8 virtual int rmf_eos::saturation ( ) [virtual]

This function first constructs an initial guess, increasing the chemical potentials if required to ensure the neutron and proton densities are finite, and then uses rmf_eos::sat_mroot to solve the field equations and ensure that the neutron and proton densities are equal and the pressure is zero. The quantities hadronic_eos::n0, hadronic_eos::eoa, and hadronic_eos::msom can be computed directly, and the compressibility, the skewness, and the symmetry energy are computed using the functions fkprime_fields() and fesym_fields(). This function overrides the generic version in hadronic_eos.

If verbose is greater than zero, then then this function reports details on the initial iterations to get the initial guess for the solver.

Reimplemented from hadronic_eos.

Reimplemented in rmf_delta_eos.

14.34.2.9 double rmf_eos::fesym_fields (double *sig*, double *ome*, double *nb*)

This may only work at saturation density. Used by saturation().

14.34.2.10 double rmf_eos::fcomp_fields (double *sig*, double *ome*, double *nb*)

This may only work at saturation density.

14.34.2.11 int rmf_eos::fkprime_fields (double *sig*, double *ome*, double *nb*, double & *k*, double & *kprime*)

This may only work at saturation density. Used by saturation().

Todo Does this work? Fix fkprime_fields() if it does not.

14.34.2.12 int rmf_eos::field_eqs (size_t *nv*, const ovector_base & *x*, ovector_base & *y*)

x[0], x[1], and x[2] should be set to σ, ω , and ρ on input (in fm^{-1}) and on exit, y[0], y[1] and y[2] contain the field equations and are zero when the field equations have been solved. The `pa` parameter is ignored.

14.34.2.13 int rmf_eos::field_eqsT (size_t *nv*, const ovector_base & *x*, ovector_base & *y*)

x[0], x[1], and x[2] should be set to σ, ω , and ρ on input (in fm^{-1}) and on exit, y[0], y[1] and y[2] contain the field equations and are zero when the field equations have been solved. The `pa` parameter is ignored.

14.34.2.14 int rmf_eos::get_fields (double & *sig*, double & *ome*, double & *lrho*) [inline]

This returns the most recent values of the meson fields set by a call to saturation(), calc_e(), or calc_p(fermion &, fermion &, thermo &).

Definition at line 488 of file rmf_eos.h.

14.34.2.15 int rmf_eos::check_naturalness (rmf_eos & *re*) [inline]

As given in Muller96 .

The definition of the vector-isovector field and coupling matches what is done here. Compare the Lagrangian above with Eq. 10 from the reference.

The following couplings should all be of the same size:

$$\frac{1}{2c_s^2M^2}, \frac{1}{2c_v^2M^2}\frac{1}{8c_\rho^2M^2}, \text{ and } \frac{\bar{a}_{ijk}M^{i+2j+2k-4}}{2^{2k}}$$

which are equivalent to

$$\frac{m_s^2}{2g_s^2M^2}, \frac{m_v^2}{2g_v^2M^2}\frac{m_\rho^2}{8g_\rho^2M^2}, \text{ and } \frac{a_{ijk}M^{i+2j+2k-4}}{g_s^i g_v^{2j} g_\rho^{2k} 2^{2k}}$$

The connection the a_{ijk} 's and the coefficients that are used here is

$$\begin{aligned}
\frac{bM}{3}g_\sigma^3\sigma^3 &= a_{300}\ \sigma^3 \\
\frac{c}{4}g_\sigma^4\sigma^4 &= a_{400}\ \sigma^4 \\
\frac{\zeta}{24}g_\omega^4\omega^4 &= a_{020}\ \omega^4 \\
\frac{\xi}{24}g_\rho^4\rho^4 &= a_{002}\ \rho^4 \\
b_1 g_\rho^2\omega^2\rho^2 &= a_{011}\ \omega^2\rho^2
\end{aligned}$$

$$
\begin{aligned}
b_2 g_\rho^2 \omega^4 \rho^2 &= a_{021}\ \omega^4 \rho^2 \\
b_3 g_\rho^2 \omega^6 \rho^2 &= a_{031}\ \omega^6 \rho^2 \\
a_1 g_\rho^2 \sigma^1 \rho^2 &= a_{101}\ \sigma^1 \rho^2 \\
a_2 g_\rho^2 \sigma^2 \rho^2 &= a_{201}\ \sigma^2 \rho^2 \\
a_3 g_\rho^2 \sigma^3 \rho^2 &= a_{301}\ \sigma^3 \rho^2 \\
a_4 g_\rho^2 \sigma^4 \rho^2 &= a_{401}\ \sigma^4 \rho^2 \\
a_5 g_\rho^2 \sigma^5 \rho^2 &= a_{501}\ \sigma^5 \rho^2 \\
a_6 g_\rho^2 \sigma^6 \rho^2 &= a_{601}\ \sigma^6 \rho^2
\end{aligned}
$$

Note that Muller and Serot use the notation

$$\frac{\bar{\kappa} g_s^3}{2} = \frac{\kappa}{2} = b M g_s^3 \qquad \text{and} \qquad \frac{\bar{\lambda} g_s^4}{6} = \frac{\lambda}{6} = c g_s^4$$

which differs slightly from the "standard" notation above.

We need to compare the values of

$$
\begin{gathered}
\frac{m_s^2}{2 g_s^2 M^2}, \frac{m_v^2}{2 g_v^2 M^2} \frac{m_\rho^2}{8 g_\rho^2 M^2}, \frac{b}{3}, \frac{c}{4} \\
\frac{\zeta}{24}, \frac{\xi}{384}, \frac{b_1}{4 g_\omega^2}, \frac{b_2 M^2}{4 g_\omega^4}, \frac{b_3 M^4}{4 g_\omega^6}, \frac{a_1}{4 g_\sigma M}, \\
\frac{a_2}{4 g_\sigma^2}, \frac{a_3 M}{4 g_\sigma^3}, \frac{a_4 M^2}{4 g_\sigma^4}, \frac{a_5 M^3}{4 g_\sigma^5}, \text{ and } \frac{a_6 M^4}{4 g_\sigma^6}.
\end{gathered}
$$

These values are stored in the variables cs, cw, cr, b, c, zeta, xi, b1, etc. in the specified rmf_eos object. All of the numbers should be around 0.001 or 0.002.

For the scale M, mnuc is used.

Todo I may have ignored some signs in the above, which are unimportant for this application, but it would be good to fix them for posterity.

Definition at line 594 of file rmf_eos.h.

14.34.2.16 int rmf_eos::naturalness_limits (double *value*, rmf_eos & *re*) [inline]

The limits for the couplings are function of the nucleon and meson masses, except for the limits on `b`, `c`, `zeta`, and `xi` which are independent of the masses because of the way that these four couplings are defined.

Definition at line 631 of file rmf_eos.h.

14.34.2.17 int rmf_eos::calc_cr (double *sig*, double *ome*, double *nb*) [protected]

Used by fix_saturation().

14.34.3 Field Documentation

14.34.3.1 int rmf_eos::verbose

This is used by saturation() to report progress towards computing the properties of nuclear matter near saturation

Definition at line 234 of file rmf_eos.h.

14.34.3.2 double rmf_eos::mnuc

This need not be exactly equal to the neutron or proton mass, but provides the scale for the coupling `b`.

Definition at line 252 of file rmf_eos.h.

14.34.3.3 gsl_mroot_hybrids<mm_funct<> > rmf_eos::def_sat_mroot

Used by fn0() (which is called by saturation()) to solve saturation_matter_e() (1 variable).

Definition at line 675 of file rmf_eos.h.

14.34.3.4 double rmf_eos::n_charge [protected]

Todo Should use hadronic_eos::proton_frac instead?

Definition at line 689 of file rmf_eos.h.

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

- rmf_eos.h

14.35 rmf_nucleus Class Reference

Spherical closed-shell nuclei with a relativistic mean-field model in the Hartree approximation.

`#include <rmf_nucleus.h>`

14.35.1 Detailed Description

This code is very experimental.

This class is based on a code developed by C.J. Horowitz and B.D. Serot, and used in Horowitz81 which was then adapted by P.J. Ellis and used in Heide94 and Prakash94. Ellis and A.W. Steiner adapted it for the parameterization in in rmf_eos for Steiner05b, and then converted to C++ by Steiner afterwards.

The standard usage is something like:

```
rmf_nucleus rn;
o2scl_hdf::rmf_load(rn.rmf,"NL4");
rn.run_nucleus(82,208,0,0);
cout << rn.rnrp << endl;
```

which computes the structure of ^{208}Pb and outputs the neutron skin thickness using the model `'NL4'`.

Potential exceptions are

- Failed to converge
- Failed to solve meson field equations
- Energy not finite (usually a problem in the equation of state)
- Energy not finite in final calculation
- Function iterate() called before init_run()
- Not a closed-shell nucleus

The initial level pattern is

```
1 S 1/2
// 2 nucleons
1 P 3/2
1 P 1/2
// 8 nucleus
1 D 5/2
```

```
1 D 3/2
2 S 1/2
// 20 nucleons
1 F 7/2
// 28 nucleons
1 F 5/2
2 P 3/2
2 P 1/2
// 40 nucleons
1 G 9/2
// 50 nucleus
1 G 7/2
2 D 5/2
1 H 11/2
2 D 3/2
3 S 1/2
// 82 nucleons
1 H 9/2
2 F 7/2
1 I 13/2
2 F 5/2
3 P 3/2
3 P 1/2
// 126 nucleons
2 G 9/2
1 I 11/2
1 J 15/2
3 D 5/2
4 S 1/2
2 G 7/2
3 D 3/2
// 184 nucleons
```

Below, α is a generic index for the isospin, the radial quantum number n and the angular quantum numbers κ and m. The meson fields are $\sigma(r), \omega(r)$ and $\rho(r)$. The baryon density is $n(r)$, the neutron and proton densities are $n_n(r)$ and $n_p(r)$, and the baryon scalar density is $n_s(r)$. The nucleon field equations are

$$\begin{aligned} F'_\alpha(r) - \frac{\kappa}{r}F_\alpha(r) + \left[\varepsilon_\alpha - g_\omega\omega(r) - t_\alpha g_\rho\rho(r) - (t_\alpha + \frac{1}{2})eA(r) - M + g_\sigma\sigma(r)\right]G_\alpha(r) &= 0 \\ G'_\alpha(r) + \frac{\kappa}{r}G_\alpha(r) - \left[\varepsilon_\alpha - g_\omega\omega(r) - t_\alpha g_\rho\rho(r) - (t_\alpha + \frac{1}{2})eA(r) + M - g_\sigma\sigma(r)\right]F_\alpha(r) &= 0 \end{aligned}$$

where t_α is 1/2 for protons and -1/2 for neutrons. The meson field equations are

$$\begin{aligned} \sigma''(r) + \frac{2}{r}\sigma'(r) - m_\sigma^2\sigma &= -g_\sigma n_s(r) + bMg_\sigma^3\sigma^2 + cg_\sigma^4\sigma^3 - g_\rho^2\rho^2\frac{\partial f}{\partial\sigma} \\ \omega''(r) + \frac{2}{r}\omega'(r) - m_\omega^2\omega &= -g_\omega n(r) + \frac{\zeta}{6}g_\omega^4\omega^3 + g_\rho^2\rho^2\frac{\partial f}{\partial\omega} \\ \rho''(r) + \frac{2}{r}\rho'(r) - m_\rho^2\rho &= -\frac{g_\rho}{2}\left[n_n(r) - n_p(r)\right] + 2g_\rho^2\rho f + \frac{\xi}{6}g_\rho^4\rho^3 \end{aligned}$$

and the Coulomb field equation is

$$A''(r) + \frac{2}{r}A'(r) = -en_p(r)$$

The densities are

$$\begin{aligned} n_s &= \sum_\alpha\left\{\int d^3r\left[g(r)^2 - f(r)^2\right]\right\} \\ n &= \sum_\alpha\left\{\int d^3r\left[g(r)^2 + f(r)^2\right]\right\} \\ n_i &= \sum_\alpha\left\{t_\alpha\int d^3r\left[g(r)^2 - f(r)^2\right]\right\} \end{aligned}$$

$$n_c = \sum_\alpha \left\{ \left[t_\alpha + \frac{1}{2} \right] \int d^3r \left[g(r)^2 - f(r)^2 \right] \right\}$$

Using the Green function

$$D(r,r',m_i) = \frac{-1}{m_i r r'} \sinh(m_i r_<) \exp(-m_i r_>)$$

one can write the meson field

$$\sigma(r) = \int_0^\infty r'^2 \, dr' \left[-g_\sigma \rho_s(r) \right] D\left(r,r',m_\sigma\right)$$

When $r > r'$, and setting $x = r' - r$, the Green function is

$$D = \frac{-1}{m_i r (x+r)} \sinh\left[m_i (x+r)\right] \exp(-m_i r)$$

When $x >> r$,

The total energy is

$$E = \sum_\alpha \varepsilon_\alpha (2j_\alpha + 1) - \frac{1}{2} \int d^3r \left[-g_\sigma \sigma(r) \rho_s(r) + g_\omega \omega(r) \rho(r) + \frac{1}{2} g_\rho \rho(r) + eA(r) n_p(r) \right]$$

The charge density is the proton density folded with the charge density of the proton, i.e.

$$\rho_{\rm ch}(r) = \int d^3r' \rho_{\rm prot}(r-r') \rho_p(r)$$

where the proton charge density is assumed to be of the form

$$\rho_{\rm prot}(r) = \frac{\mu^3}{8\pi} \exp\left(-\mu |r|\right)$$

and the parameter $\mu = (0.71)^{1/2}$ GeV (see Eq. 20b in Horowitz81). The default value of a_proton is the value of μ converted into fm^{-1}.

Todo Better documentation

Convert energies() to use EOS and possibly replace sigma_rhs() and related functions by the associated field equation method of rmf_eos.

Todo Document hw=3.923+23.265/cbrt(atot);

Idea for Future Sort energy levels at the end by energy

Idea for Future Improve the numerical methods

Idea for Future Make the neutron and proton orbitals more configurable

Idea for Future Generalize to $m_n \neq m_p$.

Idea for Future Allow more freedom in the integrations

Idea for Future Consider converting everything to inverse fermis.

Idea for Future Convert to zero-indexed arrays

Idea for Future Warn when the level ordering is wrong, and unoccupied levels are lower energy than occupied levels

Definition at line 237 of file rmf_nucleus.h.

Data Structures

- struct initial_guess
  Initial guess structure.
- struct odparms
  A convenient struct for the solution of the Dirac equations.
- struct shell
  A shell of nucleons for rmf_nucleus.

Public Member Functions

Basic operation

- void run_nucleus (int nucleus_Z, int nucleus_N, int unocc_Z, int unocc_N)
  Computes the structure of a nucleus with the specified number of levels.
- void set_verbose (int v)
  Set output level.

Lower-level interface

- void init_run (int nucleus_Z, int nucleus_N, int unocc_Z, int unocc_N)
  Initialize a run.
- void iterate (int nucleus_Z, int nucleus_N, int unocc_Z, int unocc_N, int &iconverged)
  Perform an iteration.
- int post_converge (int nucleus_Z, int nucleus_N, int unocc_Z, int unocc_N)
  After convergence, make CM corrections, etc.

Data Fields

- initial_guess ig
  Parameters for initial guess.
- bool generic_ode
  If true, use the generic ODE solver instead of ...

Protected Member Functions

- int load_nl3 (rmf_eos &r)
  Load the default model NL3 into the given rmf_eos object.
- void init_meson_density ()
  Initialize the meson fields, the densities, etc.
- void energies (double xpro, double xnu, double e)
  Calculate the energy profile.
- void center_mass_corr (double atot)
  Compute the center of mass correction.
- double gunt (double x, double g1, double f1, double &funt, double eigen, double kappa, **uvector** &varr)
  Integrate the Dirac equations using a simple inline 4th order Runge-Kutta.

Protected Attributes

- double a_proton
  The factor for the charge density of the proton (default 4.2707297)
- rmf_eos * rmf
  The base EOS.
- **o2_shared_ptr**< **table_units** >::type profiles
  The radial profiles.
- **o2_shared_ptr**< **table_units** >::type chden_table
  The final charge densities.
- std::vector< shell > * levp

A pointer to the current vector of levels (either levels or unocc_levels)

- int verbose
  Control output.
- shell neutron_shells [n_internal_levels]
  The starting neutron levels.
- shell proton_shells [n_internal_levels]
  The starting proton levels.
- double step_size
  The grid step size.
- double mnuc
  The nucleon mass (automatically set in init_fun())
- **uvector** energy
  Energy profile.
- bool init_called
  True if init() has been called.
- double ode_y [2]
  ODE functions.
- double ode_dydx [2]
  ODE derivatives.
- double ode_yerr [2]
  ODE errors.

Density information (protected)

- **umatrix** xrho
  The densities.
- **uvector** xrhosp
  The proton scalar density.
- **uvector** xrhos
  The scalar field RHS.
- **uvector** xrhov
  The vector field RHS.
- **uvector** xrhor
  The iso-vector field RHS.
- **uvector** chden1
  Charge density.
- **uvector** chdenc
  Charge density.
- **uvector** arho
  Baryon density.

Gauss-Legendre integration points and weights

- double **x12** [6]
- double **w12** [6]
- double **x100** [50]
- double **w100** [50]

Static Protected Attributes

- static const int n_internal_levels = 29
  The total number of shells stored internally.
- static const int grid_size = 300
  The grid size.

Results

- int nlevels
  The number of levels.
- std::vector< shell > levels

The levels (protons first, then neutrons)

- int nuolevels
  The number of unoccupied levels (equal to `unocc_Z` + `unocc_N`)
- int last_conv
  Information on the last convergence error.
- std::vector< shell > unocc_levels
  The unoccupied levels (protons first, then neutrons)
- double stens
  Surface tension (in fm^{-3} *)*
- double rnrp
  Skin thickness (in fm)
- double rnrms
  Neutron RMS radius (in fm)
- double rprms
  Proton RMS radius (in fm)
- double etot
  Total energy (in MeV)
- double r_charge
  Charge radius (in fm)
- double r_charge_cm
  Charge radius corrected by the center of mass (in fm)
- **o2_shared_ptr**< **table_units** >::type get_profiles ()
  Get the radial profiles.
- **o2_shared_ptr**< **table_units** >::type get_chden ()
  The finaal charge densities.

Equation of state

- rmf_eos def_rmf
  The default equation of state (default NL3)
- **thermo** hb
  thermo *object for the EOS*
- **fermion** n
  The neutron.
- **fermion** p
  The proton.
- int set_eos (rmf_eos &r)
  Set the base EOS to be used.

Numeric configuration

- typedef double arr_t [2]
  The array type for the ODE solver.
- bool err_nonconv
  If true, call the error handler if the routine does not converge or reach the desired tolerance (default true)
- int itmax
  Maximum number of total iterations (default 70)
- int meson_itmax
  Maximum number of iterations for solving the meson field equations (default 10000)
- int dirac_itmax
  Maximum number of iterations for solving the Dirac equations (default 100)
- double dirac_tol
  Tolerance for Dirac equations (default 5×10^{-3} *).*
- double dirac_tol2
  Second tolerance for Dirac equations (default 5×10^{-4} *).*
- double meson_tol
  Tolerance for meson field equations (default 10^{-6} *).*
- void set_step (**ode_step**< **ode_funct**< arr_t >, arr_t > &step)
  Set the stepper for the Dirac differential equation.

The meson fields and field equations (protected)

- **umatrix** field0
  Values of the fields from the last iteration.
- **umatrix** fields
  The values of the fields.
- **umatrix** gin
  The Green's functions inside.
- **umatrix** gout
  The Green's functions outside.
- int surf_index
  The grid index corresponding to the nuclear surface (computed by init_run())
- double sigma_rhs (double sig, double ome, double rho)
  Scalar density RHS.
- double omega_rhs (double sig, double ome, double rho)
  Vector density RHS.
- double rho_rhs (double sig, double ome, double rho)
  Iso-vector density RHS.
- void mesint ()
  Calculate meson Green's functions.
- void meson (double ic)
  Calculate meson fields.
- void meson_solve ()
  Solve for the meson profiles.

Calculating the form factor, etc. (protected)

- **smart_interp_vec**< **uvector_base**, **uvector_const_subvector**, **uvector**, **uvector_alloc** > * gi
  Interpolation object.
- void pfold (double x, double &xrhof)
  Fold in proton form factor.
- double xpform (double x, double xp, double a)
  Function representing proton form factor.
- void gauss (double xmin, double xmax, double x, double &xi)
  Perform integrations for form factor.
- double xrhop (double x1)
  Desc.

Solving the Dirac equations (protected)

- **gsl_rkck**< **ode_funct**< arr_t > , arr_t, arr_t, **array_alloc** < arr_t > > def_step
  The default stepper.
- **ode_step**< **ode_funct**< arr_t > , arr_t > * ostep
  The ODE stepper.
- void dirac (int ilevel)
  Solve the Dirac equations.
- void dirac_step (double &x, double h, double eigen, double kappa, **uvector** &varr)
  Take a step in the Dirac equations.
- int odefun (double x, size_t nv, const arr_t &y, arr_t &dydx, odparms &op)
  The form of the Dirac equations for the ODE stepper.
- void field (double x, double &s, double &v, **uvector_base** &varr)
  Compute the fields for the Dirac equations.

14.35.2 Member Function Documentation

14.35.2.1 void rmf_nucleus::run_nucleus (int *nucleus_Z*, int *nucleus_N*, int *unocc_Z*, int *unocc_N*)

Note that rmf must be set before run_nucleus() is called.

This calls init_run(), and then iterate() until `iconverged` is 1, and then post_converge().

14.35.2.2 void rmf_nucleus::init_run (int *nucleus_Z*, int *nucleus_N*, int *unocc_Z*, int *unocc_N*)

Note that rmf must be set before run_nucleus() is called.

14.35.2.3 o2_shared_ptr<table_units>::type rmf_nucleus::get_profiles ( ) [inline]

The profiles are calculated each iteration by iterate().

Definition at line 329 of file rmf_nucleus.h.

14.35.2.4 int rmf_nucleus::set_eos (rmf_eos & *r*) [inline]

The equation of state must be set before run_nucleus() or init_fun() are called, including the value of rmf_eos::mnuc.

Definition at line 419 of file rmf_nucleus.h.

14.35.2.5 void rmf_nucleus::dirac (int *ilevel*) [protected]

Solves the Dirac equation in from 12 fm to the match point and then out from .04 fm and adjusts eigenvalue with

$$\Delta\varepsilon = -g(r = \text{match\_point}) \times (f^{+} - f^{-})$$

14.35.3 Field Documentation

14.35.3.1 std::vector<shell> rmf_nucleus::levels

An array of size nlevels

Definition at line 342 of file rmf_nucleus.h.

14.35.3.2 std::vector<shell> rmf_nucleus::unocc_levels

An array of size nuolevels

Definition at line 355 of file rmf_nucleus.h.

14.35.3.3 double rmf_nucleus::stens

Computed in post_converge() or automatically in run_nucleus()

Definition at line 361 of file rmf_nucleus.h.

14.35.3.4 double rmf_nucleus::rnrp

Computed every iteration in iterate() or automatically in run_nucleus()

Definition at line 368 of file rmf_nucleus.h.

14.35.3.5 double rmf_nucleus::rnrms

Computed every iteration in iterate() or automatically in run_nucleus()

Definition at line 375 of file rmf_nucleus.h.

14.35.3.6 double rmf_nucleus::rprms

Computed every iteration in iterate() or automatically in run_nucleus()

Definition at line 382 of file rmf_nucleus.h.

14.35.3.7 double rmf_nucleus::etot

Computed every iteration in iterate() or automatically in run_nucleus()

Definition at line 389 of file rmf_nucleus.h.

14.35.3.8 double rmf_nucleus::r_charge

Computed in post_converge() or automatically in run_nucleus()

Definition at line 395 of file rmf_nucleus.h.

14.35.3.9 double rmf_nucleus::r_charge_cm

Computed in post_converge() or automatically in run_nucleus()

Definition at line 401 of file rmf_nucleus.h.

14.35.3.10 rmf_eos rmf_nucleus::def_rmf

This is set in the constructor to be the default model, NL3, using the function load_nl3().

Definition at line 412 of file rmf_nucleus.h.

14.35.3.11 thermo rmf_nucleus::hb

This is just used as temporary storage.

Definition at line 428 of file rmf_nucleus.h.

14.35.3.12 fermion rmf_nucleus::n

The mass of the neutron is ignored and set by init_run() to be rmf_eos::mnuc from rmf.

Definition at line 435 of file rmf_nucleus.h.

14.35.3.13 fermion rmf_nucleus::p

The mass of the proton is ignored and set by init_run() to be rmf_eos::mnuc from rmf.

Definition at line 442 of file rmf_nucleus.h.

14.35.3.14 bool rmf_nucleus::err_nonconv

If this is false, the function proceeds normally and may provide convergence information in last_conv.

Definition at line 454 of file rmf_nucleus.h.

14.35.3.15 initial_guess rmf_nucleus::ig

Default is {310,240,-6,25.9,6.85,0.6}

Definition at line 518 of file rmf_nucleus.h.

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

- rmf_nucleus.h

14.36 rms_radius Class Reference

Compute the RMS radius of a Fermi-Dirac density distribution with fixed diffusiveness.

```
#include <ldrop_mass.h>
```

14.36.1 Detailed Description

This class computes the RMS radius given either the central density or the radius specified in the Fermi function. This class assumes the density distribution function is of the form

$$N = 4\pi\rho_0 \int r^2 \, dr \, \{1+\exp[(r-R_{\rm fermi})]\}^{-1}$$

where N is the total number of particles, and ρ_0 is the central density.

The radius assuming constant density,

$$R_{\rm cd} = \left(\frac{3N}{4\pi\rho_0}\right)^3,$$

is also given.

Definition at line 60 of file ldrop_mass.h.

Public Member Functions

- int eval_rms_rho (double rho0, double N, double d, double &Rcd, double &Rfermi, double &Rrms)
  Compute the RMS radius from the central density.
- int eval_rms_rsq (double Rfermi, double N, double d, double &rho0, double &Rcd, double &Rrms)
  Compute the RMS radius from the Fermi distribution radius.

Protected Member Functions

- double iand (double r)
  The function $4\pi r^4\rho(r)$.
- double iand2 (double r)
  The function $4\pi r^2\rho(r)$.
- double solve (double x)
  The function to fix the total number of particles.

Protected Attributes

- double urho0
  The central denstiy.
- double ud
  The diffusiveness.
- double uRfermi
  Store the user-specified value of the radius in the Fermi distribution.
- double uN
  The total number of particles.
- int pa
  Blank parameter.
- **gsl_inte_qagiu**< **funct** > it
  The integrator.
- **cern_mroot_root**< **funct** > cr
  The solver.

14.36.2 Member Function Documentation

14.36.2.1 int rms_radius::eval_rms_rho (double *rho0*, double *N*, double *d*, double & *Rcd*, double & *Rfermi*, double & *Rrms*)

Computes the RMS radius `Rrms` from the central density `rho0`, the number of particles `N`, and the diffusiveness `d`. This function also computes the radius in the Fermi distribution function, `Rfermi` and the radius assuming constant density, `Rcd`.

14.36.2.2 int rms_radius::eval_rms_rsq (double *Rfermi,* double *N,* double *d,* double & *rho0,* double & *Rcd,* double & *Rrms*)

Computes the RMS radius `Rrms` from the radius `Rfermi` in the Fermi distribution assuming a total number of particles `N`, a diffusiveness paramter `d`. This function also produces the central density `rho0`, and the radius assuming constant density, `Rcd`.

14.36.3 Field Documentation

14.36.3.1 double rms_radius::uRfermi [protected]

This is used in the integrands iand() and iand2().

Definition at line 73 of file ldrop_mass.h.

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

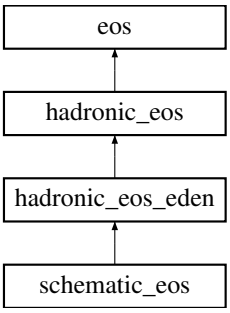
- ldrop_mass.h

14.37 schematic_eos Class Reference

Schematic hadronic equation of state.

`#include <schematic_eos.h>`

Inheritance diagram for schematic_eos:



14.37.1 Detailed Description

A schematic equation of state defined by the energy density:

$$\varepsilon = n_n m_n + n_p m_p + n\left\{eoa + \frac{comp}{18}(n/n0-1)^2 + \frac{kprime}{162}(n/n0-1)^3 + \frac{kpp}{1944}(n/n0-1)^4 + (1-2x)^2\left[a\left(\frac{n}{n0}\right)^{2/3} + b\left(\frac{n}{n0}\right)^{\gamma}\right]\right\}$$

Symmetry energy at nuclear matter density is a+b.

Note that it doesn't really matter what kind of particle object is used, since the calc_e() function doesn't use any of the particle thermodynamics functions.

Definition at line 52 of file schematic_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &ln, **fermion** &lp, **thermo** &lth)
  Equation of state as a function of density.
- virtual int set_kprime_zeroden ()
  Set kprime so that the energy per baryon of zero-density matter is zero.

- virtual int set_kpp_zeroden ()
  Set kpp so that the energy per baryon of zero-density matter is zero.
- virtual int set_a_from_mstar (double u_msom, double mnuc)
  Fix the kinetic energy symmetry coefficient from the nucleon effective mass and the saturation density.
- virtual double eoa_zeroden ()
  Return the energy per baryon of matter at zero density.
- virtual const char * type ()
  Return string denoting type ("schematic_eos")

Data Fields

- double a
  The kinetic energy symmetry coefficient in inverse fm (default 17/hc)
- double b
  The potential energy symmetry coefficient in inverse fm (default 13/hc)
- double kpp
  The coefficient of a density to the fourth term (default 0)
- double gamma
  The exponent of the high-density symmetry energy (default 1.0)

14.37.2 Member Function Documentation

14.37.2.1 virtual int schematic_eos::set_a_from_mstar (double *u_msom*, double *mnuc*) [inline, virtual]

This assumes the nucleons are non-relativistic and that the neutrons and protons have equal mass. The relativistic corrections are around 1 part in 10^6.

Todo This was computed in schematic_sym.nb, which might be added to the documentation?

Definition at line 107 of file schematic_eos.h.

14.37.2.2 virtual double schematic_eos::eoa_zeroden ( ) [inline, virtual]

This is inaccessible from calc_e() so is available separately here. Using set_kprime_zeroden() or set_kpp_zeroden() will fix kprime or kpp (respectively) to ensure that this is zero.

The result provided here does not include the nucleon mass and is given in fm^{-1}.

Definition at line 122 of file schematic_eos.h.

14.37.3 Field Documentation

14.37.3.1 double schematic_eos::a

The default value corresponds to an effective mass of about 0.7.

Definition at line 62 of file schematic_eos.h.

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

- schematic_eos.h

14.38 rmf_nucleus::shell Struct Reference

A shell of nucleons for rmf_nucleus.

`#include <rmf_nucleus.h>`

14.38.1 Detailed Description

Definition at line 270 of file rmf_nucleus.h.

Data Fields

- int twojp1
  Degeneracy $2j+1$.
- int kappa
  κ
- double energy
  Energy eigenvalue.
- double isospin
  Isospin ($+1/2$ *or* $-1/2$.
- std::string state
  Angular momentum-spin state $^{2s+1}\ell_j$.
- double match_point
  Matching radius (in fm)
- double eigen
  Desc.
- double eigenc
  Desc.
- int nodes
  Number of nodes in the wave function.

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

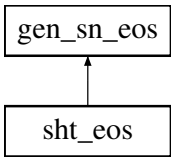
- rmf_nucleus.h

14.39 sht_eos Class Reference

A class to manipulate the G. Shen et al. EOS.

`#include <gen_sn_eos.h>`

Inheritance diagram for sht_eos:



14.39.1 Detailed Description

This class is experimental.

Note

O_2scl Does not contain the EOS, only provides some code to manipulate it. This class was designed to be used with the FSU models given at `http://cecelia.physics.indiana.edu/gang_shen_eos/FSU/fsu.html` and stored in the files named `FSU1.7eos1.01.dat`, `FSU2.1eos1.01.dat`, `FSU1.7eosb1.01.dat`, and `FSU2.1eosb1.01.dat` corresponding to mode_17, mode_21, mode_17b and mode_21b respectively.

See also the documentation at gen_sn_eos.

Todo More documentation

Definition at line 756 of file gen_sn_eos.h.

Public Member Functions

- virtual void load (std::string fname, size_t mode)
  Load table from filename `fname`.
- virtual void load (std::string fname)
  Load table from filename `fname`.
- virtual void beta_eq_T0 (size_t i, double &nb, double &E_beta, double &P_beta, double &Ye_beta, double &Z_beta, double &A_beta)
  Compute properties of matter in beta equilibrium at zero temperature at a baryon density grid point.

Data Fields

- **tensor_grid3** & T
  Temperature in MeV.
- **tensor_grid3** & Yp
  Proton fraction.
- **tensor_grid3** & nB
  Baryon number density in $1/\mathrm{fm}^3$.
- **tensor_grid3** & mue
  Electron chemical potential in MeV.
- **tensor_grid3** & M_star
  Nucleon effective mass in MeV.

Static Public Attributes

- static const size_t mode_17 = 0
  1.7 solar masses with leptons and photons
- static const size_t mode_21 = 1
  2.1 solar masses with leptons and photons
- static const size_t mode_17b = 2
  1.7 solar masses without leptons and photons
- static const size_t mode_21b = 3
  2.1 solar masses without leptons and photons
- static const size_t mode_NL3 = 4
  NL3 model with leptons and photons.
- static const size_t mode_NL3b = 5
  NL3 model with leptons and photons.

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

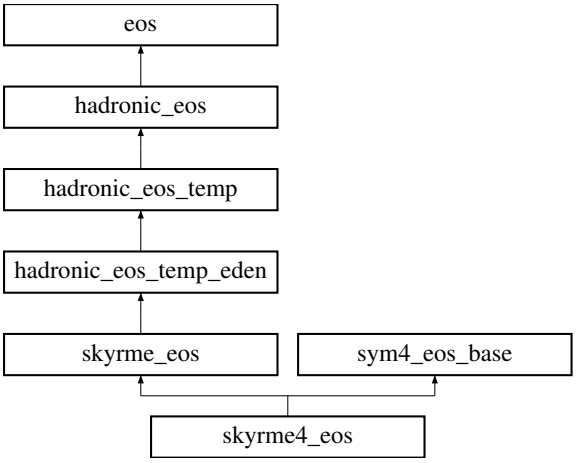
- gen_sn_eos.h

14.40 skyrme4_eos Class Reference

A version of skyrme_eos to separate potential and kinetic contributions.

`#include <sym4_eos.h>`

Inheritance diagram for skyrme4_eos:



14.40.1 Detailed Description

References:

Created for Steiner06.

Definition at line 144 of file sym4_eos.h.

Public Member Functions

- virtual int calc_e_sep (**fermion** &ne, **fermion** &pr, double &ed_kin, double &ed_pot, double &mu_n_kin, double &mu_p_kin, double &mu_n_pot, double &mu_p_pot)
  Compute the potential and kinetic parts separately.

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

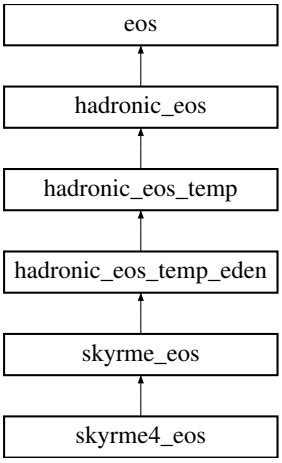
- sym4_eos.h

14.41 skyrme_eos Class Reference

Skyrme hadronic equation of state.

`#include <skyrme_eos.h>`

Inheritance diagram for skyrme_eos:



14.41.1 Detailed Description

Equation of state of nucleonic matter based on the Skryme interaction from Skyrme59 .

Background:

The Hamiltonian is defined (using the notation of Steiner05b)

$$\mathscr{H} = \mathscr{H}_{k1} + \mathscr{H}_{k2} + \mathscr{H}_{k3} + \mathscr{H}_{p1} + \mathscr{H}_{p2} + \mathscr{H}_{p3} + \mathscr{H}_{g1} + \mathscr{H}_{g2}$$

The kinetic terms are:

$$\mathscr{H}_{k1} = \frac{\tau_n}{2m_n} + \frac{\tau_p}{2m_p}$$

$$\mathscr{H}_{k2} = n(\tau_n + \tau_p)\left[\frac{t_1}{4}\left(1+\frac{x_1}{2}\right) + \frac{t_2}{4}\left(1+\frac{x_2}{2}\right)\right]$$

$$\mathscr{H}_{k3} = (\tau_n n_n + \tau_p n_p)\left[\frac{t_2}{4}\left(\frac{1}{2}+x_2\right) - \frac{t_1}{4}\left(\frac{1}{2}+x_1\right)\right]$$

where τ_i are defined

The potential terms are:

$$\mathscr{H}_{p1} = \frac{t_0}{2}\left[\left(1+\frac{x_0}{2}\right)n^2 - \left(\tfrac{1}{2}+x_0\right)\left(n_n^2+n_p^2\right)\right]$$

$$\mathscr{H}_{p2} = \frac{at_3}{6}\left[\left(1+\frac{x_3}{2}\right)n^\alpha n_n n_p + 2^{\alpha-2}(1-x_3)\left(n_n^{\alpha+2}+n_p^{\alpha+2}\right)\right]$$

$$\mathscr{H}_{p3} = \frac{bt_3}{12}\left[\left(1+\frac{x_3}{2}\right)n^{\alpha+2} - \left(\frac{1}{2}+x_3\right)n^\alpha\left(n_n^2+n_p^2\right)\right]$$

The gradient terms are displayed here for completeness even though they are not computed in the code:

$$\mathscr{H}_{g1} = \frac{3}{32}\left[t_1(1-x_1) - t_2(1+x_2)\right]\left[(\nabla n_n)^2 + (\nabla n_p)^2\right]$$

$$\mathscr{H}_{g2} = \frac{1}{8}\left[3t_1\left(1+\frac{x_1}{2}\right) - t_2\left(1+\frac{x_2}{2}\right)\right]\nabla n_n \nabla n_p$$

The values $a=0, b=1$ give the standard definition of the Skyrme Hamiltonian Skyrme59, while $a=1, b=0$ contains the modifications suggested by Onsi94.

Also, couple useful definitions

$$t_3' = (a+b)\,t_3\,,$$

$$C = \frac{3}{10m}\left(\frac{3\pi^2}{2}\right)^{2/3},$$

and

$$\beta = \frac{M}{2}\left[\frac{1}{4}\left(3t_1 + 5t_2\right) . + t_2 x_2\right]$$

Units:

Quantities which have units containing powers of energy are divided by $\hbar c$ to ensure all quantities are in units of fm. The x_i and α are unitless, while the original units of the t_i are:

- t_0 - MeV fm^3
- t_1 - MeV fm^5
- t_2 - MeV fm^5
- t_3 - MeV fm$^{3(1+\alpha)}$

These are stored internally with units of:

- t_0 - fm^2
- t_1 - fm^4
- t_2 - fm^4
- t_3 - fm$^{2+3\alpha}$

The class skyrme_eos_io uses **o2scl_const::hc_mev_fm** for I/O so that all files contain the parameters in the original units.

Misc:

The functions for the usual saturation properties are based partly on Brack85.

Models are taken from the references: Bartel79, Beiner75, Chabanat95, Chabanat97, Danielewicz08, Dobaczewski94, Dutta86, Friedrich86, Onsi94, Reinhard95, and Tondeur84, and VanGiai81 .

See Mathematica notebook at

```
doc/o2scl/extras/skyrme_eos.nb
doc/o2scl/extras/skyrme_eos.ps
```

The variables ν_n and ν_p contain the expressions $(-\mu_n + V_n)/T$ and $(-\mu_p + V_p)/T$ respectively, where V is the potential part of the single particle energy for particle i (i.e. the derivative of the Hamiltonian w.r.t. density while energy density held constant). Equivalently, ν_n is just $-k_{F_n}^2/2m^*$.

Note

The finite temperature code does not include attempt to include antiparticles and uses part::calc_density().
Since this EOS uses the effective masses and chemical potentials in the fermion class, the values of **part::non_interacting** for neutrons and protons are set to false in many of the functions.

Todo • Make sure that this class properly handles particles for which inc_rest_mass is true/false

---

- What about the spin-orbit units?
- Need to write a function that calculates saturation density?
- Remove use of mnuc in calparfun()?
- The compressibility could probably use some simplification
- Make sure the finite-temperature part is properly tested
- The testing code doesn't work if err_mode is 2, probably because of problems in load().
- Document load() file format.
- Update reference list.

Idea for Future

- There is some code duplication between calc_e() and calc_temp_e() which could be simplified.

Definition at line 218 of file skyrme_eos.h.

Public Member Functions

- int calpar (double gt0=-10.0, double gt3=70.0, double galpha=0.2, double gt1=2.0, double gt2=-1.0)
  Calculate t_0, t_1, t_2, t_3 and α from the saturation properties.
- int check_landau (double nb, double m)
  Check the Landau parameters for instabilities.
- int landau_nuclear (double n0, double m, double &f0, double &g0, double &f0p, double &g0p, double &f1, double &g1, double &f1p, double &g1p)
  Calculate the Landau parameters for nuclear matter.
- int landau_neutron (double n0, double m, double &f0, double &g0, double &f1, double &g1)
  Calculate the Landau parameters for neutron matter.
- virtual const char * type ()
  Return string denoting type ("skyrme_eos")

Basic usage

- skyrme_eos ()
  Create a blank Skyrme EOS.
- virtual ~skyrme_eos ()
  Destructor.
- virtual int calc_temp_e (**fermion** &ne, **fermion** &pr, double temper, **thermo** &th)
  Equation of state as a function of densities.
- virtual int calc_e (**fermion** &ne, **fermion** &pr, **thermo** &lt)
  Equation of state as a function of density.

Saturation properties

These calculate the various saturation properties exactly from the parameters at any density. These routines often assume that the neutron and proton masses are equal.

- virtual double feoa (double nb)
  Calculate binding energy.
- virtual double fmsom (double nb)
  Calculate effective mass.
- virtual double fcomp (double nb)
  Calculate compressibility.
- virtual double fesym (double nb, double alpha=0.0)
  Calculate symmetry energy.
- virtual double fkprime (double nb)
  skewness

Data Fields

- double W0
  Spin-orbit splitting.
- bool parent_method
  Use hadronic_eos methods for saturation properties.
- bool mu_at_zero_density
  Desc.

Basic Skyrme model parameters

- double **t0**
- double **t1**
- double **t2**
- double **t3**
- double **x0**
- double **x1**
- double **x2**
- double **x3**
- double **alpha**
- double **a**
- double **b**

14.41.2 Member Function Documentation

14.41.2.1 virtual int skyrme_eos::calc_temp_e (fermion & *ne*, fermion & *pr*, double *temper*, thermo & *th*) [virtual]

Note

Runs the zero temperature code if `temper` is less than or equal to zero.

Implements hadronic_eos_temp_eden.

14.41.2.2 virtual double skyrme_eos::feoa (double *nb*) [virtual]

$$\frac{E}{A} = Cn_B^{2/3}(1+\beta n_B) + \frac{3t_0}{8}n_B + \frac{t_3'}{16}n_B^{\alpha+1}$$

14.41.2.3 virtual double skyrme_eos::fmsom (double *nb*) [virtual]

$$M^*/M = (1+\beta n_B)^{-1}$$

14.41.2.4 virtual double skyrme_eos::fcomp (double *nb*) [virtual]

$$K = 10Cn_B^{2/3} + \frac{27}{4}t_0 n_B + 40C\beta n_B^{5/3} + \frac{9t_3'}{16}\alpha(\alpha+1)n_B^{1+\alpha} + \frac{9t_3'}{8}(\alpha+1)n_B^{1+\alpha}$$

14.41.2.5 virtual double skyrme_eos::fesym (double *nb*, double *alpha* = `0.0`) [virtual]

If pf=0.5, then the exact expression below is used. Otherwise, the method from class hadronic_eos is used.

$$E_{sym} = \frac{5}{9}Cn^{2/3} + \frac{10Cm}{3}\left[\frac{t_2}{6}\left(1+\frac{5}{4}x_2\right) - \frac{1}{8}t_1x_1\right]n^{5/3} - \frac{t_3'}{24}\left(\tfrac{1}{2}+x_3\right)n^{1+\alpha} - \frac{t_0}{4}\left(\tfrac{1}{2}+x_0\right)n$$

14.41.2.6 virtual double skyrme_eos::fkprime (double *nb*) [virtual]

$$2Cn_B^{2/3}(9-5/M^*/M)+\frac{27t_3'}{16}n^{1+\alpha}\alpha\left(\alpha^2-1\right)$$

14.41.2.7 int skyrme_eos::calpar (double *gt0* = -10.0, double *gt3* = 70.0, double *galpha* = 0.2, double *gt1* = 2.0, double *gt2* = -1.0)

In nuclear matter:

$E_b = E_b(n_0, M^*, t_0, t_3, \alpha)$

$P = P(n_0, M^*, t_0, t_3, \alpha)$

$K = K(n_0, M^*, t_3, \alpha)$ (the t_0 dependence vanishes)

$M^* = M^*(n_0, t_1, t_2, x_2)$ (the x_1 dependence cancels),

$E_{sym} = E_{sym}(x_0, x_1, x_2, x_3, t_0, t_1, t_2, t_3, \alpha)$

To fix the couplings from the saturation properties, we take n_0, M^*, E_b, K as inputs, and we can fix t_0, t_3, α from the first three relations, then use M^*, E_b to fix t_2 and t_1. The separation into two solution steps should make for better convergence. All of the x's are free parameters and should be set before the function call.

The arguments gt0, gt3, galpha, gt1, and gt2 are used as initial guesses for skyme_eos::t0, skyrme_eos::t3, skyrme_eos::alpha, skyrme_eos::t1, and skyrme_eos::t2 respectively.

Todo Does this work for both 'a' and 'b' non-zero?

Todo Compare to similar formulae from Margueron02

14.41.2.8 int skyrme_eos::check_landau (double *nb*, double *m*)

This returns zero if there are no instabilities.

14.41.2.9 int skyrme_eos::landau_nuclear (double *n0*, double *m*, double & *f0*, double & *g0*, double & *f0p*, double & *g0p*, double & *f1*, double & *g1*, double & *f1p*, double & *g1p*)

Given n0 and m, this calculates the Landau parameters in nuclear matter as given in Margueron02

Todo This needs to be checked.

(Checked once on 11/05/03)

14.41.2.10 int skyrme_eos::landau_neutron (double *n0*, double *m*, double & *f0*, double & *g0*, double & *f1*, double & *g1*)

Given 'n0' and 'm', this calculates the Landau parameters in neutron matter as given in Margueron02

Todo This needs to be checked

(Checked once on 11/05/03)

14.41.3 Field Documentation

14.41.3.1 double skyrme_eos::W0

This is unused, but included for possible future use and present in the internally stored models.

Definition at line 231 of file skyrme_eos.h.

14.41.3.2 bool skyrme_eos::parent_method

This can be set to true to check the difference between the exact expressions and the numerical values from class hadronic_eos.

Definition at line 362 of file skyrme_eos.h.

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

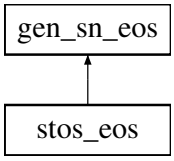
- skyrme_eos.h

14.42 stos_eos Class Reference

The Shen et al. supernova EOS.

`#include <gen_sn_eos.h>`

Inheritance diagram for stos_eos:



14.42.1 Detailed Description

This class is experimental.

Note

O_2scl Does not contain the EOS, only provides some code to manipulate it. This class is designed to be used with the file which was originally called `eos.tab` and now referred to as `eos1.tab` and stored e.g. at `http://user.numazu-ct.ac.jp/~sumi/eos/.`

In order to force the EOS to a uniform grid, linear interpolation is used to recast the variation in baryon density, choosing the grid in baryon density to be the same as the section in the table with T=0.1 MeV and $Y_p = 0.1$ for all temperature and proton fraction points.

Also, the original EOS is tabulated for constant proton fraction, and this O_2scl interface assumes that the electron fraction is equal to the proton fraction. Currently, this is a problem only at higher densities where muons might appear.

The data for gen_sn_eos::E, gen_sn_eos::F, gen_sn_eos::S, and gen_sn_eos::P is not stored in the table but can be computed with gen_sn_eos::compute_eg().

See also the documentation at gen_sn_eos.

See Shen98 and Shen98b .

Note

Thanks to Matthias Hempel for providing the correct temperature grid.

Idea for Future Add the T=0 and Ye=0 data to this class

Definition at line 642 of file gen_sn_eos.h.

Public Member Functions

- virtual void load (std::string fname)

Load table from filename `fname`.

- virtual void load (std::string fname, size_t mode)
  Load table from filename `fname`.
- virtual void beta_eq_T0 (size_t i, double &nb, double &E_beta, double &P_beta, double &Ye_beta, double &Z_beta, double &A_beta)
  Compute properties of matter in beta equilibrium at zero temperature at a baryon density grid point.

Data Fields

- **tensor_grid3** & log_rho
  Logarithm of baryon number density in g/cm^3.
- **tensor_grid3** & nB
  Baryon number density in fm^{-3}.
- **tensor_grid3** & log_Y
  Logarithm of proton fraction.
- **tensor_grid3** & Yp
  Proton fraction.
- **tensor_grid3** & M_star
  Nucleon effective mass in MeV.
- **tensor_grid3** & quark_frac
  Fraction of quark matter.

Static Public Attributes

- static const size_t **orig_mode** = 0
- static const size_t **quark_mode** = 1

14.42.2 Member Function Documentation

14.42.2.1 virtual void stos_eos::beta_eq_T0 (size_t *i*, double & *nb*, double & *E_beta*, double & *P_beta*, double & *Ye_beta*, double & *Z_beta*, double & *A_beta*) [inline, virtual]

This EOS table doesn't have T=0 results, so we extrapolate from the two low-temperature grid points.

Implements gen_sn_eos.

Definition at line 687 of file gen_sn_eos.h.

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

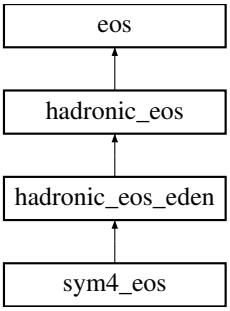
- gen_sn_eos.h

14.43 sym4_eos Class Reference

Construct an EOS with an arbitrary choice for the terms in the symmetry energy that are quartic in the isospin asymmetry.

`#include <sym4_eos.h>`

Inheritance diagram for sym4_eos:



14.43.1 Detailed Description

References:

Created for Steiner06.

Definition at line 198 of file sym4_eos.h.

Public Member Functions

- int set_base_eos (sym4_eos_base &seb)
  Set the base equation of state.
- virtual int test_eos (**fermion** &ne, **fermion** &pr, **thermo** &lth)
  Test the equation of state.
- virtual int calc_e (**fermion** &ne, **fermion** &pr, **thermo** &lth)
  Equation of state as a function of density.

Data Fields

- double alpha
  The strength of the quartic terms.

Protected Attributes

- sym4_eos_base * sp
  The base equation of state to use.

14.43.2 Member Function Documentation

14.43.2.1 virtual int sym4_eos::test_eos (**fermion** & *ne,* **fermion** & *pr,* **thermo** & *lth*) [virtual]

This compares the chemical potentials from calc_e_sep() to their finite-difference approximations in order to ensure that the separation into potential and kinetic parts is done properly.

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

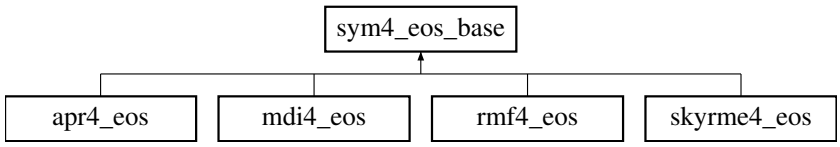
- sym4_eos.h

14.44 sym4_eos_base Class Reference

A class to compute quartic contributions to the symmetry energy [abstract base].

`#include <sym4_eos.h>`

Inheritance diagram for sym4_eos_base:



14.44.1 Detailed Description

The standard usage is that a child class implements the virtual function calc_e_sep() which is then used by calc_e_alpha() and calc_muhat(). These functions are employed by sym4_eos to compute the EOS for an arbitrary dependence of the symmetry energy on the isospin.

References:

Created for Steiner06.

Bug Testing was disabled in HDF conversion. Fix this.

Definition at line 55 of file sym4_eos.h.

Public Member Functions

- virtual int calc_e_alpha (**fermion** &ne, **fermion** &pr, **thermo** &lth, double &alphak, double &alphap, double &alphat, double &diff_kin, double &diff_pot, double &ed_kin_nuc, double &ed_pot_nuc)
  Compute alpha at the specified density.
- virtual double calc_muhat (**fermion** &ne, **fermion** &pr)
  Compute $\hat{\mu}$, the out-of-whack parameter.
- virtual int calc_e_sep (**fermion** &ne, **fermion** &pr, double &ed_kin, double &ed_pot, double &mu_n_kin, double &mu_p_kin, double &mu_n_pot, double &mu_p_pot)=0
  Compute the potential and kinetic parts separately (to be overwritten in children)

Protected Attributes

- **fermion** e
  An electron for the computation of the $\hat{\mu}$.
- fermion_zerot fzt2
  Desc.

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

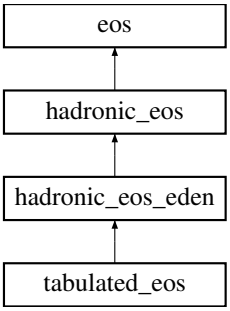
- sym4_eos.h

14.45 tabulated_eos Class Reference

EOS from a table.

`#include <tabulated_eos.h>`

Inheritance diagram for tabulated_eos:



14.45.1 Detailed Description

This assumes a symmetry energy which depends quadratically on the isospin asymmetry in order to construct an EOS from a table of baryon density and energy per baryon for both nuclear and pure neutron matter.

Note: If using a tabulated EOS to compute derivatives (like the compressibility which effectively requires a second derivative), it is important to tabulated the EOS precisely enough to ensure that the derivatives are accurate. In the case of ensuring that the compressibility at saturation density is well reproduced, I have needed the EOS to be specified with at least 6 digits of precision on a grid at least as small as 0.002 fm^{-3}.

Definition at line 52 of file tabulated_eos.h.

Public Member Functions

- virtual int calc_e (**fermion** &ne, **fermion** &pr, **thermo** &th)
  Equation of state as a function of density.
- template<class vec_t >
  int set_eos (size_t n, vec_t &rho, vec_t &Enuc, vec_t &Eneut)
  Set the EOS through vectors specifying the densities and energies.
- template<class vec_t >
  int set_eos (size_t n_nuc, vec_t &rho_nuc, vec_t &E_nuc, size_t n_neut, vec_t &rho_neut, vec_t &E_neut)
  Set the EOS through vectors specifying the densities and energies.
- **table** & get_nuc_table ()
  Return the internal table.
- **table** & get_neut_table ()
  Return the internal table.

Protected Member Functions

- int free_table ()
  Free the table memory.

Protected Attributes

- bool table_alloc
  True if the table has been allocated.
- bool one_table
  If true, then tnuc and tneut point to the same table.

The EOS tables

- **table** ∗ **tnuc**
- **table** ∗ **tneut**

Strings for the column names

- std::string **srho_nuc**
- std::string **srho_neut**
- std::string **snuc**
- std::string **sneut**

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

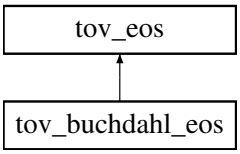
- tabulated_eos.h

14.46 tov_buchdahl_eos Class Reference

The Buchdahl EOS for the TOV solver.

`#include <tov_eos.h>`

Inheritance diagram for tov_buchdahl_eos:



14.46.1 Detailed Description

Given the eos

$$\rho = 12\sqrt{p_* P} - 5P$$

the TOV equation has an analytical solution

$$R = (1-\beta)\sqrt{\frac{\pi}{288 p_* G(1-2\beta)}}$$

where $\beta = GM/R$.

The central pressure and energy density are

$$P_c = 36 p_* \beta^2$$

$$\rho_c = 72 p_* \beta (1 - 5\beta/2)$$

Physical solutions are obtained only for $P < 25p_*/144$ and $\beta < 1/6$.

Based on Lattimer01.

Idea for Future Figure out what to do with the buchfun() function

Definition at line 363 of file tov_eos.h.

Public Member Functions

- virtual int get_eden (double P, double &e, double &nb)
  Given the pressure, produce the energy and number densities.
- virtual int get_aux (double P, size_t &np, **ovector_base** &auxp)
  Given the pressure, produce all the remaining quantities.
- virtual int get_names (size_t &np, std::vector< std::string > &pnames)
  Fill a list with strings for the names of the remaining quanities.

Data Fields

- double Pstar
  The parameter with units of pressure in units of solar masses per km cubed (default value 3.2×10^{-5} *)*

14.46.2 Member Function Documentation

14.46.2.1 virtual int tov_buchdahl_eos::get_eden (double *P*, double & *e*, double & *nb*) [inline, virtual]

If the baryon density is not specified, it should be set to zero or baryon_column should be set to false

Implements tov_eos.

Definition at line 384 of file tov_eos.h.

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

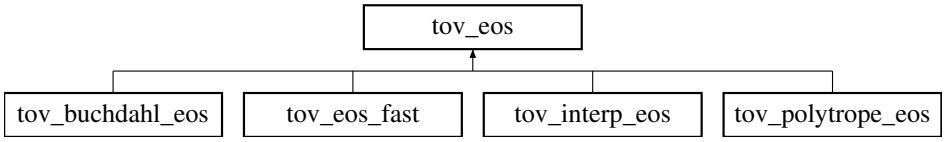
- tov_eos.h

14.47 tov_eos Class Reference

A EOS base class for the TOV solver.

`#include <tov_eos.h>`

Inheritance diagram for tov_eos:



14.47.1 Detailed Description

Todo Fix read_table_file and maybe set_low_density_eos().

Definition at line 53 of file tov_eos.h.

Public Member Functions

- virtual int get_eden (double P, double &e, double &nb)=0
  Given the pressure, produce the energy and number densities.
- virtual int get_aux (double P, size_t &np, **ovector_base** &auxp)
  Given the pressure, produce all the remaining quantities.
- virtual int get_names_units (size_t &np, std::vector< std::string > &pnames, std::vector< std::string > &punits)
  Fill a list with strings for the names of the remaining quanities.

Data Fields

- int verbose
  Control for output (default 1)
- bool baryon_column
  Set to true if the baryon density is provided in the EOS (default false)

Protected Attributes

- double mev_kg
  To convert MeV to kilograms.
- double mev_per_fm3_msun_km3
  To convert MeV / fm^3 to solar masses per cubic kilometer.

14.47.2 Member Function Documentation

14.47.2.1 virtual int tov_eos::get_eden (double *P*, double & *e*, double & *nb*) [pure virtual]

The arguments `P` and `e` should always be in $M_\odot/\mathrm{km}^3$.

If the baryon density is not specified, it should be set to zero or baryon_column should be set to false.

Implemented in tov_polytrope_eos, tov_buchdahl_eos, tov_interp_eos, and tov_eos_fast.

14.47.2.2 virtual int tov_eos::get_aux (double *P*, size_t & *np*, **ovector_base** & *auxp*) [inline, virtual]

The argument `P` should always be in $M_\odot/\mathrm{km}^3$.

Reimplemented in tov_polytrope_eos, tov_buchdahl_eos, and tov_interp_eos.

Definition at line 93 of file tov_eos.h.

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

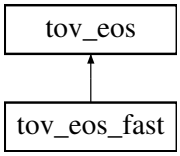
- tov_eos.h

14.48 tov_eos_fast Class Reference

An EOS.

`#include <tov_eos_fast.h>`

Inheritance diagram for tov_eos_fast:



14.48.1 Detailed Description

Definition at line 38 of file tov_eos_fast.h.

Public Member Functions

- virtual int get_eden (double pres, double &ed, double &nb)
  Given the pressure, produce the energy and number densities.
- int set_crust_core (size_t n_crust, **ovector_base** &crust_e, **ovector_base** &crust_p, **ovector_base** &crust_nb, size_t n_core, **ovector_base** &core_e, **ovector_base** &core_p, **ovector_base** &core_nb)
  Desc.

Data Fields

- bool check_eos
  If true, check the EOS for stability and ordering (default true)

Protected Member Functions

- void interp (const **ovector_base** &x, const **ovector_base** &y, double xx, double &yy, int n1, int n2)
  Linear EOS interpolation.

Protected Attributes

- size_t nlines
  Desc.
- **ovector** edv
  Desc.
- **ovector** prv
  Desc.
- **ovector** nbv
  Desc.
- bool eos_set
  Desc.

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

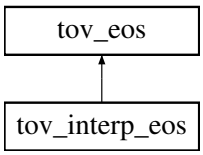
- tov_eos_fast.h

14.49 tov_interp_eos Class Reference

An EOS for the TOV solver using simple linear interpolation and a default low-density EOS.

`#include <tov_eos.h>`

Inheritance diagram for tov_interp_eos:



14.49.1 Detailed Description

Internally, energy and pressure are stored in units of solar masses per cubic kilometer and baryon density is stored in units of fm^{-3}. The user-specified EOS table is left as is, and unit conversion is performed as needed in get_eden() and other functions so that results are returned in the units specified by set_units().

Note

This stores a pointer to the user-specified table, so if that pointer becomes invalid, the interpolation will fail.

The function set_units() needs to be called before either of the functions get_eden() or get_eden_ld() are called. The function set_units() may be called after calling either the read_table() functions or the set_low_density_eos() function.

Todo Warn that the pressure in the low-density eos is not strictly increasing! (see at P=4.3e-10)

It might be useful to exit more gracefully when non-finite values are obtained in interpolation, analogous to the err_nonconv mechanism elsewhere.

Definition at line 141 of file tov_eos.h.

Public Member Functions

- virtual int get_eden (double pres, double &ed, double &nb)
  Given the pressure, produce the energy and number densities.
- virtual int get_eden_user (double pres, double &ed, double &nb)
  Given the pressure, produce the energy and number densities from the user-specified EOS.
- virtual int get_eden_ld (double pres, double &ed, double &nb)
  Given the pressure, produce the energy and number densities from the low-density EOS.
- virtual int get_aux (double P, size_t &nv, **ovector_base** &auxp)
  Given the pressure, produce all the remaining quantities.
- virtual int get_names_units (size_t &np, std::vector< std::string > &pnames, std::vector< std::string > &punits)
  Fill a list with strings for the names of the remaining quanities.
- int read_table (**table_units** &eosat, std::string s_cole, std::string s_colp, std::string s_colnb="")
  Specify the EOS through a table.
- int set_low_density_eos (std::string s_nvpath, int s_nvcole=0, int s_nvcolp=1, int s_nvcolnb=2)
  Set the low-density EOS.
- int default_low_dens_eos ()
  Default low-density EOS.
- int no_low_dens_eos ()
  Compute with no low-density EOS.
- int set_units (double s_efactor, double s_pfactor, double s_nfactor)
  Set the units of the user-specified EOS.
- int set_units (std::string leunits="", std::string lpunits="", std::string lnunits="")
  Set the units of the user-specified EOS.
- int get_transition (double &plow, double &ptrans, double &phi)
  Return limiting and transition pressures.
- int set_transition (double ptrans, double pw)
  Set the transition pressure and "width".

Protected Member Functions

- int check_eos ()
  Check that the EOS is valid.
- void interp (const **ovector_base** &x, const **ovector_base** &y, double xx, double &yy, int n1, int n2)
  Linear EOS interpolation.

Protected Attributes

Low-density EOS

- bool ldeos
  true if we are using the low-density eos (false)
- bool ldeos_read
  Low-density EOS switch.
- std::string ldpath
  the path to the low-density EOS
- int ldcole
  column in low-density eos for energy density
- int ldcolp
  column in low-density eos for pressure
- int ldcolnb
  column in low-density eos for baryon density

- **table_units** * ld_eos
  file containing low-density EOS
- double presld
  Highest pressure in low-density EOS (in $M_\odot/\mathrm{km}^3$)
- double eld
  Highest energy density in low-density EOS (in $M_\odot/\mathrm{km}^3$)
- double nbld
  Highest baryon density in low-density EOS (in $M_\odot/\mathrm{km}^3$)
- double prest
  Transition pressure (in $M_\odot/\mathrm{km}^3$)
- double pwidth
  Transition width (unitless)

User EOS

- **table_units** * eost
  file containing eos
- int nfile
  number of lines in eos file
- int cole
  column for energy density in eos file
- int colp
  column for pressure in eos file
- int colnb
  column for baryon density in eos file
- bool eos_read
  True if an EOS has been specified.

Units

- std::string eunits
  Units for energy density.
- std::string punits
  Units for pressure.
- std::string nunits
  Units for baryon density.
- double efactor
  unit conversion factor for energy density (default 1.0)
- double pfactor
  unit conversion factor for pressure (default 1.0)
- double nfactor
  unit conversion factor for baryon density (default 1.0)

14.49.2 Member Function Documentation

14.49.2.1 virtual int tov_interp_eos::get_eden (double *pres*, double & *ed*, double & *nb*) [virtual]

The arguments P and e should always be in $M_\odot/\mathrm{km}^3$.

If the baryon density is not specified, it should be set to zero or baryon_column should be set to false.

Implements tov_eos.

14.49.2.2 virtual int tov_interp_eos::get_eden_user (double *pres*, double & *ed*, double & *nb*) [virtual]

The arguments pres, ed, and nb should be in the user-specified unit system.

14.49.2.3 virtual int tov_interp_eos::get_eden_ld (double *pres*, double & *ed*, double & *nb*) [virtual]

The arguments pres, ed, and nb should be in the user-specified unit system.

14.49.2.4 virtual int tov_interp_eos::get_aux (double *P*, size_t & *nv*, ovector_base & *auxp*) [virtual]

The argument `P` should always be in $M_\odot/\mathrm{km}^3$.

Reimplemented from tov_eos.

14.49.2.5 int tov_interp_eos::get_transition (double & *plow*, double & *ptrans*, double & *phi*)

Returns, in order:

- the highest pressure in the low-density EOS
- the transition pressure
- the lowest pressure in the high-density EOS

14.49.2.6 int tov_interp_eos::set_transition (double *ptrans*, double *pw*)

Sets the transition pressure and the width (specified as a number greater than unity in `pw`) of the transition between the two EOSs. The transition is done smoothly using linear interpolation between $P = \mathrm{ptrans}/pmathrmpw$ and $P = \mathrm{ptrans} \times pmathrmpw$.

14.49.3 Field Documentation

14.49.3.1 bool tov_interp_eos::ldeos_read [protected]

This is `true` if the ldeos has been read by set_ldeos. This is useful, since then we know whether or not we need to free the memory for the LD EOS in the destructor

Definition at line 255 of file tov_eos.h.

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

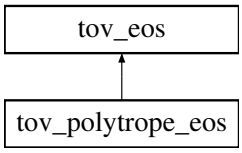
- tov_eos.h

14.50 tov_polytrope_eos Class Reference

Standard polytropic EOS $p = K\rho^{1+1/n}$.

`#include <tov_eos.h>`

Inheritance diagram for tov_polytrope_eos:



14.50.1 Detailed Description

Any units are permissible, but if this is to be used with tov_solve, then the units of K must be consistent with the units set in tov_solve::set_units().

Definition at line 426 of file tov_eos.h.

Public Member Functions

- virtual int get_eden (double P, double &e, double &nb)
  Given the pressure, produce the energy and number densities.
- virtual int get_aux (double P, size_t &np, **ovector_base** &auxp)
  Given the pressure, produce all the remaining quantities.
- virtual int get_names (size_t &np, std::vector< std::string > &pnames)
  Fill a list with strings for the names of the remaining quanities.

Data Fields

- double K
  Coefficient (default 1.0)
- double n
  Index (default 3.0)

14.50.2 Member Function Documentation

14.50.2.1 virtual int tov_polytrope_eos::get_eden (double *P*, double & *e*, double & *nb*) [inline, virtual]

If the baryon density is not specified, it should be set to zero or baryon_column should be set to false

Implements tov_eos.

Definition at line 449 of file tov_eos.h.

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

- tov_eos.h

14.51 tov_solve Class Reference

Class to solve the Tolman-Oppenheimer-Volkov equations.

`#include <tov_solve.h>`

14.51.1 Detailed Description

Mathematical background:

In units where $c = 1$, the TOV equations (i.e. Einstein's equations for a static spherically symmetric object) are

$$\frac{dm}{dr} = 4\pi r^2 \varepsilon$$

$$\frac{dP}{dr} = -\frac{G\varepsilon m}{r^2}\left(1+\frac{P}{\varepsilon}\right)\left(1+\frac{4\pi P r^3}{m}\right)\left(1-\frac{2Gm}{r}\right)^{-1}$$

where r is the radial coordinate, $m(r)$ is the gravitational mass enclosed within a radius r, and $\varepsilon(r)$ and $P(r)$ are the energy density and pressure at r, and G is the gravitational constant. The boundary conditions are $m(r=0)=0$ the condition $P(r=R)=0$ for some fixed radius R. These boundary conditions give a series of solutions to the TOV equations as a function of the radius, although they do not necessarily have a solution for all radii.

The gravitational mass is given by

$$M_G = \int_0^R 4\pi r^2 \varepsilon dr$$

The gravitational potential, $\Phi(r)$ can be determined from

$$\frac{d\Phi}{dr} = -\frac{1}{\varepsilon}\frac{dP}{dr}\left(1+\frac{P}{\varepsilon}\right)^{-1}$$

The proper boundary condition for the gravitational potential is

$$\Phi(r=R) = \frac{1}{2}\ln\left(1-\frac{2GM}{R}\right)$$

which ensures that $\phi(r) \to 0$ as $r \to \infty$.

The surface gravity is

$$g = \frac{Gm}{r^2}\left(1-\frac{2Gm}{r}\right)^{-1/2}$$

which is given in inverse kilometers and the redshift is

$$z = \left(1-\frac{2Gm}{r}\right)^{-1/2} - 1$$

which is unitless.

The baryonic mass is typically defined by

$$M_B = \int_0^R 4\pi r^2 n_B m_B \left(1-\frac{2Gm}{r}\right)^{-1/2} dr$$

where $n_B(r)$ is the baryon number density at radius r and m_B is the mass of one baryon. Then the "binding energy" of the neutron star is defined to be $M_B - M_G$. If you can specify the product $n_B m_B$ in the EOS (analogous to the rest mass energy density), it will compute the associated baryonic mass for you.

In the case of slow rigid rotation with angular velocity Ω, the moment of inertia is

$$I = \frac{8\pi}{3}\int_0^R dr\, r^4 (\varepsilon + P) e^{\Phi}\left(\frac{\bar{\omega}}{\Omega}\right)\left(1-\frac{2Gm}{r}\right)^{-1/2}$$

where the function $\bar{\omega}(r)$ is the solution of

$$\frac{d}{dr}\left(r^4 j \frac{d\bar{\omega}}{dr}\right) + 4r^3 \frac{dj}{dr}\bar{\omega} = 0$$

and the function $j(r)$ is defined by

$$j = \left(1-\frac{2Gm}{r}\right)e^{-\Phi}$$

The boundary conditions for $\bar{\omega}$ are $d\bar{\omega}/dr = 0$ at $r = 0$ and

$$\bar{\omega}(R) = \Omega - \left(\frac{R}{3}\right)\left(\frac{d\bar{\omega}}{dr}\right)_{r=R} .$$

General usage notes:

The equation of state may be changed at any time, by specifying the appropriate tov_eos object

Screen output:

- `verbose=0` - Nothing
- `verbose=1` - Basic information

- verbose=2 - For each profile computation, report solution information at every kilometer.
- verbose=3 - Report profile information at every 20 grid points. A keypress is required after each profile.

Output tables:

The functions fixed() and max() produce output tables which represent the profile of the neutron star of the requested mass. The columns are

- gm, the enclosed gravitational mass in solar masses
- r, the radial coordinate in km
- gp, the gravitational potential (unitless) when calcgpot is true
- bm, the baryonic mass in solar masses (when tov_eos::baryon_column is true).
- pr, the pressure
- ed, the energy density
- nb, the baryon density (if tov_eos::baryon_column is true)
- sg, the local surface gravity (in g/cm^2)
- rs, the local redshift (unitless),
- dmdr, the derivative of the enclosed gravitational mass in $M_\odot/km$
- dlogpdr, the derivative of the natural logarithm of the pressure
- dgpdr, the derivative of the gravitational potential in $1/km$ (if calcgpot is true)
- dbmdr, the derivative of the enclosed baryonic mass (if tov_eos::baryon_column is true).

  The remaining columns are given by the user-defined columns from the equation of state as determined by tov_eos::get_names_units() and tov_eos::get_aux().

The function mvsr() produces a different kind of output table corresponding to the mass versus radius curve. Some points on the curve may correspond to unstable branches.

- gm, the total gravitational mass in solar masses
- r, the radius in km
- gp, the gravitational potential at the surface (unitless) when calcgpot is true
- bm, total the baryonic mass in solar masses (when tov_eos::baryon_column is true).
- pr, the central pressure
- ed, the central energy density
- nb, the central baryon density (if tov_eos::baryon_column is true)
- sg, the surface gravity (in g/cm^2)
- rs, the redshift at the surface,
- dmdr, the derivative of the gravitational mass
- dlogpdr, the derivative of the natural logarithm of the pressure
- dgpdr, the derivative of the gravitational potential in $1/km$ (if calcgpot is true)

- `dbmdr`, the derivative of the enclosed baryonic mass (if tov_eos::baryon_column is true).

  The remaining columns are given by the user-defined columns from the equation of state as determined by tov_eos::get_names_units() and tov_eos::get_aux().

Accuracy:

The present code, as demonstrated in the tests, gives the correct central pressure and energy density of the analytical solution by Buchdahl to within less than 1 part in 10^8.

Other details and todos:

Note

The function star_fun() returns `gsl_efailed` without calling the error handler in the case that the solver can recover gracefully from, for example, a negative pressure.

Todo

- baryon mass doesn't work for fixed() (This may be fixed. We should make sure it's tested.)
- Combine maxoutsize and kmax?
- Document column naming issues
- Document surface gravity and redshift
- Standardize `xmev_kg`, etc.
- Use **convert_units**?
- Double check that fixed() doesn't give a solution on the unstable branch
- Ensure that this class copies over the units of the user-specified columns to the table

---

Definition at line 243 of file tov_solve.h.

Public Member Functions

Basic operation

- int set_eos (tov_eos &ter)
  Set the EOS to use.
- int set_units (double s_efactor=1.0, double s_pfactor=1.0, double s_nbfactor=1.0)
  Set units.
- int set_units (std::string eunits="", std::string punits="", std::string nunits="")
  Set units.
- virtual int mvsr ()
  Calculate the mass vs. radius curve.
- virtual int fixed (double d_tmass)
  Calculate the profile of a star with fixed mass.
- virtual int max ()
  Calculate the profile of the maximum mass star.
- **o2_shared_ptr**< **table_units** >::type get_results ()
  Return the results data table.
- int solution_check ()
  Check the solution (unfinished)

Control numerical methods

- int set_kmax (int s_maxoutsize=400, int s_kmax=80000)
  Set maximum storage for integration.
- int set_mroot (**mroot**< **mm_funct**<> > &s_mrp)
  Set solver.
- int set_minimize (**minimize**< **funct** > &s_mp)
  Set minimizer.
- int set_stepper (**adapt_step**< **ode_funct**<> > &sap)
  Set the adaptive stepper.

Data Fields

- **gsl_min_brent**< **funct** > def_min
  The default minimizer.
- **gsl_mroot_hybrids**< **mm_funct**<> > def_solver
  The default solver.
- **gsl_astep**< **ode_funct**<> > def_stepper
  The default adaptive stepper.
- bool compute_ang_vel
  If true, compute the angular velocity (default false)
- double cap_omega
  The angular velocity.

Basic properties

- double mass
  mass
- double rad
  radius
- double bmass
  baryonic mass
- double gpot
  gravitational potential

Solution parameters

These parameters can be changed at any time.

- bool generel
  Use general relativistic version (default true)
- bool calcgpot
  calculate the gravitational potential and the enclosed baryon mass (default false)
- double hmin
  smallest allowed radial stepsize (default 1.0e-4)
- double hmax
  largest allowed radial stepsize (default 0.05)
- double hstart
  initial radial stepsize (default 4.0e-3)
- int verbose
  control for output (default 1)
- double maxradius
  maximum radius for integration in km (default 60)

Mass versus radius parameters

- double prbegin
  Beginning pressure (default 7.0e-7)
- double prend
  Ending pressure (default 8.0e-3)
- double princ
  Increment for pressure (default 1.1)
- bool logmode
  Use 'princ' as a multiplier, not an additive increment (default true)
- double prguess
  Guess for central pressure in solar masses per km3 (default 5.2×10^{-5}*)*
- double max_begin
  Beginning pressure for maximum mass guess (default 7.0e-5)
- double max_end
  Ending pressure for maximum mass guess (default 5.0e-3)
- double max_inc
  Increment for pressure for maximum mass guess (default 1.3)

Protected Member Functions

- int make_unique_name (std::string &col, std::vector< std::string > &cnames)
  Ensure `col` does not match strings in `cnames`.
- int ang_vel ()
  Compute the angular velocity.
- virtual int derivs (double x, size_t nv, const **ovector_base** &y, **ovector_base** &dydx)
  The ODE step function.
- virtual int derivs_ang_vel (double x, size_t nv, const **ovector_base** &y, **ovector_base** &dydx)
  The ODE step function for the angular velocity.
- virtual int profile_out (double xx)
  Construct a stellar profile.
- virtual double maxfun (double maxx)
  The minimizer function to compute the maximum mass.
- virtual int starfun (size_t ndvar, const **ovector_base** &ndx, **ovector_base** &ndy)
  The solver function to compute the stellar profile.

Protected Attributes

- double mev_kg
  To convert MeV to kilograms.
- double mev_per_fm3_msun_km3
  To convert MeV/fm^3 to solar masses per cubic kilometer.
- double tmass
  Target mass.
- tov_eos * te
  The EOS.
- bool eos_set
  True if the EOS has been set.
- int kmax
  maximum storage size (default 40000)
- int maxoutsize
  maximum size of output file (default 400)
- double presmin
  Smallest allowed pressure for integration (default: -100)
- **sm_interp** smi
  Interpolation object for derivs_ang_vel()
- **o2_shared_ptr**< **table_units** >::type out_table
  The output table.
- **mroot**< **mm_funct**<> > * mroot_ptr
  The solver.
- **minimize**< **funct** > * min_ptr
  The minimizer.
- **adapt_step**< **ode_funct**<> > * as_ptr
  The adaptive stepper.
- double schwarz_km
  The schwarzchild radius in km.

User EOS units

- std::string eunits
  Units for energy density.
- std::string punits
  Units for pressure.
- std::string nunits
  Units for baryon density.
- double efactor
  unit conversion factor for energy density (default 1.0)
- double pfactor
  unit conversion factor for pressure (default 1.0)
- double nfactor
  unit conversion factor for baryon density (default 1.0)

Integration storage

- **ovector** rky [6]
  ODE functions.
- **ovector** rkx
  Radial coordinate (in kilometers)
- **ovector** rkdydx [6]
  The derivatives of the ODE functions.

14.51.2 Member Function Documentation

14.51.2.1 int tov_solve::make_unique_name (std::string & *col*, std::vector< std::string > & *cnames*) [protected]

Underscores are added to `col` until it matches none of the strings in `cnames`.

14.51.2.2 virtual int tov_solve::profile_out (double *xx*) [protected, virtual]

This function constructs a stellar profile in out_table from the information in rkx and rky.

14.51.2.3 int tov_solve::set_units (std::string *eunits* = " ", std::string *punits* = " ", std::string *nunits* = " ")

Valid entries for the units of energy density and pressure are:

- "g/cm^3"
- "erg/cm^3"
- "MeV/fm^3"
- "1/fm^4"
- "Msun/km^3" (i.e. solar masses per cubic kilometer)

Valid entries for the units of baryon density are:

- "1/m^3"
- "1/cm^3"
- "1/fm^3"

14.51.2.4 int tov_solve::set_kmax (int *s_maxoutsize* = 400, int *s_kmax* = 80000)

The variable `s_kmax` is the maximum number of radial integration stepsk while `s_maxoutsize` is the maximum number of points that will be output for any profile.

If `s_kmax` is less than zero, there is no limit on the number of radial steps.

14.51.3 Field Documentation

14.51.3.1 double tov_solve::tmass [protected]

Negative values to indicate a mass measured relative to the maximum mass. For example, if the EOS has a maximum mass of 2.0, then -0.15 will give the profile of a 1.85 solar mass star.

Definition at line 261 of file tov_solve.h.

14.51.3.2 double tov_solve::presmin [protected]

This quantity can't be much smaller than -100 since we need to compute numbers near $e^{-presmin}$

Definition at line 303 of file tov_solve.h.

14.51.3.3 ovector tov_solve::rky[6] [protected]

The vector `rky`[0] is the gravitational mass in solar masses, and the vector `rky`[1] is the natural logarithm of the pressure in solar masses per cubic kilometer. When calcgpot is true, the next vector in this list is the gravitational potential (which is unitless), and when tov_eos::baryon_column is true, the next vector in this list is the baryonic mass in solar masses.

Definition at line 317 of file tov_solve.h.

14.51.3.4 double tov_solve::prguess

This guess is used in the function fixed().

Definition at line 427 of file tov_solve.h.

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

- tov_solve.h

15 File Documentation

15.1 hdf_eos_io.h File Reference

File for HDF input of the O_2scl skyrme_eos and rmf_eos data files.

```
#include <hdf5.h> #include <o2scl/constants.h> #include <o2scl/hdf_file.h> #include <o2scl/lib-
_settings.h> #include <o2scl/skyrme_eos.h> #include <o2scl/rmf_eos.h>
```

15.1.1 Detailed Description

Definition in file hdf_eos_io.h.

Functions

- int rmf_load (rmf_eos &rmf, std::string model, bool external=false)
  Input a rmf_eos object from an HDF file.
- int skyrme_load (skyrme_eos &sk, std::string model, bool external=false)
  Input a skyrme_eos object from an HDF file.