IM-SRG++

0

Generated by Doxygen 1.8.13

Contents

Chapter 1

IMSRG++: an implementation of In-Medium Similarity Renormalization Group for nuclei, written in C++

The in-medium similarity renormalization group (IM-SRG) is an ab initio method for solving many-body quantum systems. It has been thus far predominantly used in nuclear physics. For a review, see H. Hergert et al, Physics Reports 216, 165 (2016) doi: 10.1016/j.phys rep.2015.12.007, and references therein.

The main idea behind the IM-SRG is to obtain a unitary transformation U which transforms the Hamiltonian into a form which makes it easier to solve. This can either be by decoupling a single reference state from all other states, or by decoupling a valence space and diagonalizing within that space. We may parameterize the unitary transformation as the exponential of an anti-hermitian generator Ω so that the transformed Hamiltonian is

$$H' = UHU^\dagger = e^\Omega H e^{-\Omega}$$

This code contains a number of classes which may be used in the various specific schemes for implementing the above transformation. A subset of these classes are exposed as python classes, enabling rapid development of implementation schemes. The code and its documentation are very much a work in progress, so if you have any questions, or if you discover any bugs, please contact Ragnar Stroberg at sstroberg <at> triumf.ca.

Chapter 2

Code Structure/Flow

Written by Y.-H. Song.

Basic code structure and flow for rough idea. Because it is up to my understanding, there can be error and will be changes according to my understanding.

2.1 Read in arguments and input files

- Default parameters, and everything passed by command line args.

```
        2bme                          :  none
        3bme                          :  none
        LECs                          :  EM2.0_2.0
        basis                         :  HF
        core_generator                :  atan
        custom_valence_space          :
        denominator_delta_orbit       :  none
        flowfile                      :  default
        fmt2                          :  me2j
        fmt3                          :  me3j
        goose_tank                    :  false
        intfile                       :  default
        method                        :  magnus
        nucleon_mass_correction       :  false
        occ_file                      :  none
        reference                     :  default
        scratch                       :
        use_brueckner_bch             :  false
        valence_file_format           :  nushellx
        valence_generator             :  shell-model-atan
        valence_space                 :
        write_omega                   :  false
        BetaCM                        :  0
        denominator_delta             :  0
        domega                        :  0.2
        ds_0                          :  0.5
        dsmax                         :  0.5
        eta_criterion                 :  1e-06
        hw                            :  20
        hwBetaCM                      :  -1
```

```
        ode_tolerance                 :  1e-06
        omega_norm_max                :  0.25
        smax                          :  200
        A                             :  -1
        e3max                         :  12
        emax                          :  6
        file2e1max                    :  12
        file2e2max                    :  24
        file2lmax                     :  10
        file3e1max                    :  12
        file3e2max                    :  24
        file3e3max                    :  12
        lmax3                         :  -1
        nsteps                        :  -1
        Operators                     :
        OperatorsFromFile             :
        SPWF                          :
```

- set ReadWrite class 'rw'

2.2 prepare model space

- set valence space
- set ModelSpace class 'modelspace'
  - set 'reference'
  - initialize occ from file
  - set nsteps , SetHbarOmega, SetTargetMass, SetE3max,SetLmax3

2.3 prepare operator

- define Operator class Hbare from modelspace and particle rank
- Hbare.SetHermitian()
- Hbare.SetGooseTank

2.4 prepare interaction

- read two-body matrix elements in various formats
  - me2j
  - navratil
  - oslo
  - oakridge has 'bin' or 'ascii' mode
  - takayuki
  - nushellx
- Except 'nushellx' format case, add Trel operator to Hbare.
- add nucleon mass correction in Trel
- read Darmssdt 3-body interaction if particle rank >=3.
- add a Lawson term,(betaCM term)

2.5 prepare Hartree-Fock basis

- declare HartreeFock class 'hf' from Hbare
- hf.solve()

2.6 prepare Hamiltonian

- declare operator 'HNO' from Hbare
- do normal ordering to HNO. hf.GetNormalOrderedH() or Hbare.DoNormalOrdering()
- SPWF:? set radial points, Rmax, spwf indices, PSI and so on ?
- add betaCM term to HNO.
- estimate ground state energy by HF or MBPT3.

2.7 calculate operators

This part is not clear how to understand. At least, I suppose it is to define observales to be calculated.

there are loops in 'opnames', 'opsfromfile', 'ops' and so on.

```
  for (auto& opname : opnames)
{
  ops.emplace_back( imsrg_util::OperatorFromString(modelspace,opname) );
}
```

- If method is 'HF', print output and ends the code.
- If method is 'FCI', print outputs in nushellx format such as '.int','.sp','.op' . But what is exactly the output here?

2.8 IMSRG solve

This part semes to be the main IMSRG.

- declare IMSRGSolver class imsrgsolver(HNO)

```
  IMSRGSolver imsrgsolver(HNO);
  imsrgsolver.SetReadWrite(rw);
  imsrgsolver.SetEtaCriterion(eta_criterion);
  bool brueckner_restart = false;
```

- setup solver according to method.('NSmagnus', 'brueckner' etc.) and solve.

```
imsrgsolver.SetMethod(method);
imsrgsolver.SetHin(HNO);
imsrgsolver.SetSmax(smax);
imsrgsolver.SetFlowFile(flowfile);
imsrgsolver.SetDs(ds_0);
imsrgsolver.SetDsmax(dsmax);
imsrgsolver.SetDenominatorDelta(denominator_delta);
imsrgsolver.SetdOmega(domega);
imsrgsolver.SetOmegaNormMax(omega_norm_max);
imsrgsolver.SetODETolerance(ode_tolerance);
...
imsrgsolver.SetGenerator(core_generator);
...
imsrgsolver.Solve();
```

- if method is 'magnus', increase smax
- if 'brueckner restart', change HNO and solve again (imsrgsolver.Solve())
- I don't see no clear separation between IMSRG and VS-IMSRG. But, I suppose 'nsetps >1' seems to be related with VS-IMSRG.

```
    if (nsteps > 1) // two-step decoupling, do core first
    {...
     imsrgsolver.SetSmax(smax);
     imsrgsolver.Solve();
    }
```

- Transform all operators in 'magnus' method.

2.9 VS-IMSRG solve

- re-normal order wrt the core

```
   // If we're doing targeted/ensemble normal ordering
    //  we now re-normal order wrt to the core
    //  and do any remaining flow.
    ModelSpace ms2(modelspace);
    ...
    if ( renormal_order )
    {
    HNO = imsrgsolver.GetH_s();
    ...
    HNO = HNO.UndoNormalOrdering();
    ms2.SetReference(ms2.core); // change the reference
    HNO.SetModelSpace(ms2);
    HNO = HNO.DoNormalOrdering();
    imsrgsolver.SetHin(HNO);
    imsrgsolver.SetEtaCriterion(1e-4);
    imsrgsolver.Solve();
    ...
    }
```

- do similar to operators

2.10 Write output

- write shell model interaction file
- or write single reference result

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 6

Namespace Documentation

6.1 imsrg_util Namespace Reference

imsrg_util namespace. Used to define some helpful functions.

Classes

- struct FBCIntegrandParameters

Functions

- std::vector< string > **split_string** (string s, string delimiter)
- Operator **OperatorFromString** (ModelSpace &modelspace, string opname)
- Operator **NumberOp** (ModelSpace &modelspace, int n, int l, int j2, int tz2)
- Operator **NumberOpAlln** (ModelSpace &modelspace, int l, int j2, int tz2)
- double **HO_density** (int n, int l, double hw, double r)
- double **HO_Radial_psi** (int n, int l, double hw, double r)
- vector< double > **GetOccupationsHF** (HartreeFock &hf)
- vector< double > **GetOccupations** (HartreeFock &hf, IMSRGSolver &imsrgsolver)
- vector< double > **GetDensity** (vector< double > &occupation, vector< double > &R, vector< int > &orbits, ModelSpace &modelspace)
- Operator **Single_Ref_1B_Density_Matrix** (ModelSpace &modelspace)
- double **Get_Charge_Density** (Operator &DM, double r)
- Operator **KineticEnergy_Op** (ModelSpace &modelspace)
- Operator Trel_Op (ModelSpace &modelspace)
- Operator TCM_Op (ModelSpace &modelspace)
- double Calculate_p1p2 (ModelSpace &modelspace, Ket &bra, Ket &ket, int J)
- Operator Trel_Masscorrection_Op (ModelSpace &modelspace)
- void **Calculate_p1p2_all** (Operator &OpIn)
- Operator R2CM_Op (ModelSpace &modelspace)
- Operator Rp2_corrected_Op (ModelSpace &modelspace, int A, int Z)
- Operator **Rn2_corrected_Op** (ModelSpace &modelspace, int A, int Z)
- Operator **Rm2_corrected_Op** (ModelSpace &modelspace, int A, int Z)
- double Calculate_r1r2 (ModelSpace &modelspace, Ket &bra, Ket &ket, int J)
- Operator HCM_Op (ModelSpace &modelspace)
- Operator RSquaredOp (ModelSpace &modelspace)

- Operator **R2_p1_Op** (ModelSpace &modelspace)
- Operator R2_1body_Op (ModelSpace &modelspace, string option)
- Operator **R2_p2_Op** (ModelSpace &modelspace)
- Operator R2_2body_Op (ModelSpace &modelspace, string option)
- Operator **ProtonDensityAtR** (ModelSpace &modelspace, double R)
- Operator **NeutronDensityAtR** (ModelSpace &modelspace, double R)
- Operator **RpSpinOrbitCorrection** (ModelSpace &modelspace)
- Operator E0Op (ModelSpace &modelspace)
- double **FBCIntegrand** (double x, void *p)
- Operator **FourierBesselCoeff** (ModelSpace &modelspace, int nu, double R, vector< index_t > index_list)
- Operator Isospin2_Op (ModelSpace &modelspace)

  Returns the T^2 operator.
- Operator ElectricMultipoleOp (ModelSpace &modelspace, int L)
- Operator NeutronElectricMultipoleOp (ModelSpace &modelspace, int L)
- Operator MagneticMultipoleOp (ModelSpace &modelspace, int L)

  Returns a reduced magnetic multipole operator with units μ_N fm$^{\lambda-1}$.
- Operator MagneticMultipoleOp_pn (ModelSpace &modelspace, int L, string pn)
- Operator IntrinsicElectricMultipoleOp (ModelSpace &modelspace, int L)
- double RadialIntegral (int na, int la, int nb, int lb, int L)
- double **RadialIntegral_RpowK** (int na, int la, int nb, int lb, int k)
- double TalmiI (int p, double k)
- double TalmiB (int na, int la, int nb, int lb, int p)
- Operator **AllowedFermi_Op** (ModelSpace &modelspace)
- Operator AllowedGamowTeller_Op (ModelSpace &modelspace)
- Operator Sigma_Op (ModelSpace &modelspace)

  Pauli spin operator

  $$\langle f \| \sigma \| i \rangle$$

  .
- Operator Sigma_Op_pn (ModelSpace &modelspace, string pn)

  Pauli spin operator

  $$\langle f \| \sigma \| i \rangle$$

  .
- void **Reduce** (Operator &X)
- void **UnReduce** (Operator &X)
- void **SplitUp** (Operator &OpIn, Operator &OpLow, Operator &OpHi, int ecut)
- Operator **RadialOverlap** (ModelSpace &modelspace)
- Operator **LdotS_Op** (ModelSpace &modelspace)
- Operator **LCM_Op** (ModelSpace &modelspace)
- Operator **QdotQ_Op** (ModelSpace &modelspace)
- double Calculate_r1xp2 (ModelSpace &modelspace, Ket &bra, Ket &ket, int Jab, int Jcd)
- Operator M0nu_TBME_Op (ModelSpace &modelspace, int Nquad, string src)
- double CPrbmeGen (ModelSpace &modelspace, double rho, double x, int n, int l, int np, int lp, int mm, double pp)
- Operator **L2rel_Op** (ModelSpace &modelspace)
- Operator **EKKShift** (Operator &Hin, int Nlower, int Nupper)
- map< index_t, double > **GetSecondOrderOccupations** (Operator &H, int emax)
- void Embed1BodyIn2Body (Operator &op1, int A)
- double GetEmbeddedTBME (Operator &op1, index_t i, index_t j, index_t k, index_t l, int Jbra, int Jket, int Lambda)
- double **FCcoefIntegrand** (double r, void *params)
- double **FrequencyConversionCoeff** (int n1, int l1, double hw1, int n2, int l2, double hw2)
- void **CommutatorTest** (Operator &X, Operator &Y)
- Operator **PSquaredOp** (ModelSpace &modelspace)
- double **cpNorm** (int a, int b)
- template<typename T >
  T **VectorUnion** (const T &v1)
- template<typename T , typename... Args>
  T **VectorUnion** (const T &v1, const T &v2, Args... args)

6.1.1 Detailed Description

imsrg_util namespace. Used to define some helpful functions.

6.1.2 Function Documentation

6.1.2.1 AllowedGamowTeller_Op()

```
Operator imsrg_util::AllowedGamowTeller_Op (
            ModelSpace & modelspace )
```

Note that there is a literature convention to include the 1/sqrt(Lambda) factor in the reduced matrix element rather than in the expression involving the sum over one-body densities (see footnote on pg 165 of Suhonen). I do not follow this convention, and instead produce the reduced matrix element

$$\langle f \| \sigma \tau_{\pm} \| i \rangle$$

6.1.2.2 Calculate_p1p2()

```
double imsrg_util::Calculate_p1p2 (
            ModelSpace & modelspace,
            Ket & bra,
            Ket & ket,
            int J )
```

This returns the antisymmetrized J-coupled two body matrix element of $\vec{p}_1 \cdot \vec{p}_2/(m)$. The formula is

$$\frac{1}{m}\langle ab|\,\vec{p}_1\cdot\vec{p}_2\,|cd\rangle_J = \frac{1}{\sqrt{(1+\delta_{ab})(1+\delta_{cd})}} \sum_{LS} \begin{bmatrix} \ell_a & s_a & j_a \\ \ell_b & s_b & j_b \\ L & S & J \end{bmatrix} \begin{bmatrix} \ell_c & s_c & j_c \\ \ell_d & s_d & j_d \\ L & S & J \end{bmatrix} \times \sum_{\substack{N_{ab}N_{cd}\Lambda \\ n_{ab}n_{cd}\lambda}} \mathcal{A}^{\lambda S}_{abcd} \times \langle N_{ab}\Lambda n_{ab}\lambda | n_a\ell_a n_b\ell_b\rangle_L \quad \langle N_{cd}\Lambda n_{cd}\lambda | n_c\ell_c n_d\ell_d\rangle_L \times \left(\langle N_{ab}\Lambda|t_{cm}|N_{cd}\Lambda\rangle - \langle n_{ab}\lambda|t_{rel}|n_{cd}\lambda\rangle\right)$$

The antisymmetrization factor $\mathcal{A}^{\lambda S}_{abcd}$ ensures that the relative wave function is antisymmetrized. It is given by $\mathcal{A}^{\lambda S}_{abcd} = |t_{za}+t_{zc}| + |t_{za}+t_{zd}|\,(-1)^{\lambda+S+|T_z|}$.

The center-of-mass and relative kinetic energies can be found by the same equation as used in the one-body piece of TCM_Op()

6.1.2.3 Calculate_r1r2()

```
double imsrg_util::Calculate_r1r2 (
            ModelSpace & modelspace,
            Ket & bra,
            Ket & ket,
            int J )
```

Returns the normalized, anti-symmetrized, J-coupled, two-body matrix element of $\vec{r}_1 \cdot \vec{r}_2$. Calculational details are similar to Calculate_p1p2().

6.1.2.4 Calculate_r1xp2()

```
double imsrg_util::Calculate_r1xp2 (
            ModelSpace & modelspace,
            Ket & bra,
            Ket & ket,
            int Jab,
            int Jcd )
```

Returns the normalized, anti-symmetrized, J-coupled, two-body matrix element of $\frac{m\omega^2}{\hbar\omega}\vec{r}_1 \cdot \vec{r}_2$. Calculational details are similar to Calculate_p1p2().

6.1.2.5 CPrbmeGen()

```
double imsrg_util::CPrbmeGen (
            ModelSpace & modelspace,
            double rho,
            double x,
            int n,
            int l,
            int np,
            int lp,
            int mm,
            double pp )
```

testing... mm = 0,2,4 pp = 0,a,2a

6.1.2.6 E0Op()

```
Operator imsrg_util::E0Op (
            ModelSpace & modelspace )
```

Returns

$$r_e^2 = \sum_i e_i r_i^2$$

6.1.2.7 ElectricMultipoleOp()

```
Operator imsrg_util::ElectricMultipoleOp (
            ModelSpace & modelspace,
            int L )
```

Returns a reduced electric multipole operator with units e fm $^\lambda$ See Suhonen eq. (6.23)

6.1.2.8 Embed1BodyIn2Body()

```
void imsrg_util::Embed1BodyIn2Body (
            Operator & op1,
            int A )
```

Embeds the one-body operator of op1 in the two-body part, using mass number A in the embedding. Note that the embedded operator is added to the two-body part, rather than overwriting. The one-body part is left as-is.

6.1.2.9 GetEmbeddedTBME()

```
double imsrg_util::GetEmbeddedTBME (
            Operator & op1,
            index_t i,
            index_t j,
            index_t k,
            index_t l,
            int Jbra,
            int Jket,
            int Lambda )
```

Returns a normalized TBME formed by embedding the one body part of op1 in a two body operator. Assumes A=2 in the formula. For other values of A, divide by (A-1).

6.1.2.10 HCM_Op()

```
Operator imsrg_util::HCM_Op (
            ModelSpace & modelspace )
```

Center of mass Hamiltonian

$$\begin{aligned} H_{CM} &= T_{CM} + \frac{1}{2} A m \omega^2 R^2 \\ &= T_{CM} + \frac{1}{2b^2} A R^2 \hbar\omega \end{aligned}$$

6.1.2.11 IntrinsicElectricMultipoleOp()

```
Operator imsrg_util::IntrinsicElectricMultipoleOp (
            ModelSpace & modelspace,
            int L )
```

Returns a reduced electric multipole operator with units e fm $^{\lambda}$ See Suhonen eq. (6.23)

6.1.2.12 M0nu_TBME_Op()

```
Operator imsrg_util::M0nu_TBME_Op (
            ModelSpace & modelspace,
            int Nquad,
            string src )
```

This is the M^{0} TBME from Equation (1) of [PRC 87, 064315 (2013)] it was coded up by me, ie) Charlie Payne (CP) from my thesis, I employed Equations:

6.1.2.13 MagneticMultipoleOp_pn()

```
Operator imsrg_util::MagneticMultipoleOp_pn (
            ModelSpace & modelspace,
            int L,
            string pn )
```

Returns a reduced magnetic multipole operator with units μ_N fm $^{\lambda-1}$ This version allows for the selection of just proton or just neutron contributions, or both. See Suhonen eq. (6.24)

6.1.2.14 NeutronElectricMultipoleOp()

```
Operator imsrg_util::NeutronElectricMultipoleOp (
            ModelSpace & modelspace,
            int L )
```

Returns a reduced electric multipole operator with units e fm $^{\lambda}$ See Suhonen eq. (6.23)

6.1.2.15 R2_1body_Op()

```
Operator imsrg_util::R2_1body_Op (
            ModelSpace & modelspace,
            string option )
```

One-body part of the proton charge radius operator. Returns

$$\hat{R}^2_{p1} = \sum_i e_i r_i^2$$

6.1.2.16 R2_2body_Op()

```
Operator imsrg_util::R2_2body_Op (
            ModelSpace & modelspace,
            string option )
```

Two-body part of the proton charge radius operator. Returns

$$\hat{R}^2_{p2} = \sum_{i \neq j} e_i \vec{r}_i \cdot \vec{r}_j$$

evaluated in the oscillator basis.

6.1.2.17 R2CM_Op()

```
Operator imsrg_util::R2CM_Op (
            ModelSpace & modelspace )
```

Returns

$$R^2_{CM} = \left(\frac{1}{A} \sum_i \vec{r}_i \right)^2 = \frac{1}{A^2} \left(\sum_i r_i^2 + 2 \sum_{i<j} \vec{r}_i \cdot \vec{r}_j \right)$$

evaluated in the oscillator basis.

6.1.2.18 RadialIntegral()

```
double imsrg_util::RadialIntegral (
            int na,
            int la,
            int nb,
            int lb,
            int L )
```

Evaluate the radial integral

$$\tilde{\mathcal{R}}^{\lambda}_{ab} = \int_0^{\infty} dx \tilde{g}_{n_a \ell_a}(x) x^{\lambda+2} \tilde{g}_{n_b \ell_b}(x)$$

where $\tilde{g}(x)$ is the radial part of the harmonic oscillator wave function with unit oscillator length $b = 1$ and $x = r/b$. To obtain the radial integral for some other oscillator length, multiply by b^{λ}. This implementation uses eq (6.41) from Suhonen. Note this is only valid for $\ell_a + \ell_b + \lambda$ = even. If $\ell_a + \ell_b + \lambda$ is odd, RadialIntegral_RpowK() is called.

6.1.2.19 Rp2_corrected_Op()

```
Operator imsrg_util::Rp2_corrected_Op (
            ModelSpace & modelspace,
            int A,
            int Z )
```

Returns

$$R_p^2 = \frac{1}{Z}\sum_p \left(\vec{r}_p - \vec{R}_{CM}\right)^2 = R_{CM}^2 + \frac{A-2}{AZ}\sum_p r_p^2 - \frac{4}{AZ}\sum_{i<j}\vec{r}_i\cdot\vec{r}_j$$

evaluated in the oscillator basis.

6.1.2.20 RSquaredOp()

```
Operator imsrg_util::RSquaredOp (
            ModelSpace & modelspace )
```

Returns

$$r^2 = \sum_i r_i^2$$

6.1.2.21 TalmiB()

```
double imsrg_util::TalmiB (
            int na,
            int la,
            int nb,
            int lb,
            int p )
```

Calculate B coefficient for Talmi integral. Formula given in Brody and Moshinsky "Tables of Transformation Brackets for Nuclear Shell-Model Calculations"

6.1.2.22 TalmiI()

```
double imsrg_util::TalmiI (
            int p,
            double k )
```

General Talmi integral for a potential r**k 1/gamma(p+3/2) * 2*INT dr r**2 r**2p r**k exp(-r**2/b**2) This is valid for (2p+3+k) $>$ 0. The Gamma function diverges for non-positive integers.

6.1.2.23 TCM_Op()

```
Operator imsrg_util::TCM_Op (
            ModelSpace & modelspace )
```

Center of mass kinetic energy, including the hw/A factor

$$T = \frac{\hbar\omega}{A}\sum_{ij} t_{ij} a_i^\dagger a_j + \frac{\hbar\omega}{A}\frac{1}{4}\sum_{ijkl} t_{ijkl} a_i^\dagger a_j^\dagger a_l a_k$$

with a one-body piece

$$t_{ij} = \frac{1}{\hbar\omega}\langle i|T_{12}|j\rangle = \frac{1}{2}(2n_i + \ell_i + 3/2)\delta_{ij} + \frac{1}{2}\sqrt{n_j(n_j+\ell_j+\frac{1}{2})}\delta_{n_i,n_j-1}\delta_{k_i k_j}$$

where k labels all quantum numbers other than n and a two-body piece

$$t_{ijkl} = \frac{1}{\hbar\omega}\left\langle ij|(T_{12}^{CM} - T_{12}^{rel})|kl\right\rangle$$

6.1.2.24 Trel_Masscorrection_Op()

```
Operator imsrg_util::Trel_Masscorrection_Op (
            ModelSpace & modelspace )
```

Correction to Trel due to the proton-neutron mass differences

$$T_{rel} = T - T_{CM}$$

$$\delta T = \sum_i \frac{p_i^2}{2m}\left(\frac{m-m_i}{m_i}\right)$$

$$\delta T_{CM} = \left(\frac{Am}{Zm_p + Nm_n}\right)\frac{1}{2mA}P_{CM}^2$$

6.1.2.25 Trel_Op()

```
Operator imsrg_util::Trel_Op (
            ModelSpace & modelspace )
```

Relative kinetic energy, includingthe hw/A factor

$$T_{rel} = T - T_{CM}$$

Chapter 7

Class Documentation

7.1 imsrg_util::FBCIntegrandParameters Struct Reference

Public Attributes

- int **n**
- int **l**
- double **hw**

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

- imsrg_util.cc

7.2 Generator Class Reference

Public Member Functions

- void **SetType** (std::string g)
- std::string **GetType** ()
- void **Update** (Operator ∗H, Operator ∗Eta)
- void **AddToEta** (Operator ∗H, Operator ∗Eta)
- void **SetDenominatorCutoff** (double c)
- void **SetDenominatorDelta** (double d)
- void **SetDenominatorDeltaIndex** (int i)
- void **SetDenominatorDeltaOrbit** (std::string orb)
- void **ConstructGenerator_Wegner** ()
- void **ConstructGenerator_White** ()
- void **ConstructGenerator_Atan** ()
- void ConstructGenerator_ImaginaryTime ()

  Imaginary time generator.
- void **ConstructGenerator_ShellModel** ()
- void **ConstructGenerator_ShellModel_Atan** ()
- void **ConstructGenerator_ShellModel_ImaginaryTime** ()
- void **ConstructGenerator_ShellModel_Atan_NpNh** ()
- void **ConstructGenerator_HartreeFock** ()
- void **ConstructGenerator_1PA** ()
- double **Get1bDenominator** (int i, int j)
- double **Get2bDenominator** (int ch, int ibra, int iket)

Public Attributes

- std::string **generator_type**
- Operator * **H**
- Operator * **Eta**
- ModelSpace * **modelspace**
- double **denominator_cutoff**
- double **denominator_delta**
- int **denominator_delta_index**

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

- Generator.hh
- Generator.cc

7.3 HartreeFock Class Reference

Public Member Functions

- HartreeFock (Operator &hbare)

  Constructor.
- void BuildMonopoleV ()

  Only the monopole part of V is needed, so construct it.
- void BuildMonopoleV3 ()

  Only the monopole part of V3 is needed.
- void Diagonalize ()

  Diagonalize the Fock matrix.
- void UpdateF ()

  Update the Fock matrix with the new transformation coefficients C.
- void UpdateDensityMatrix ()

  Update the density matrix with the new coefficients C.
- void FillLowestOrbits ()

  Get new occupations based on the current single-particle energies.
- void UpdateReference ()

  If we got new occupations in FillLowestOrbits, then we should update the hole states in the reference.
- bool CheckConvergence ()

  Compare the current energies with those from the previous iteration.
- void Solve ()

  Diagonalize and UpdateF until convergence.
- void CalcEHF ()

  Evaluate the Hartree Fock energy.
- void PrintEHF ()

  Print out the Hartree Fock energy.
- void ReorderCoefficients ()

  Reorder the coefficients in C to eliminate phases etc.
- Operator TransformToHFBasis (Operator &OpIn)

  Transform an operator from oscillator basis to HF basis.
- Operator GetNormalOrderedH ()

  Return the Hamiltonian in the HF basis at the normal-ordered 2body level.

- Operator GetNormalOrderedH (arma::mat &Cin)

  Return the Hamiltonian in the HF basis at the normal-ordered 2body level.
- Operator GetOmega ()

  Return a generator of the Hartree Fock transformation.
- Operator **GetHbare** ()
- void PrintSPE ()

  Getter function for Hbare.
- void PrintSPEandWF ()

  Print out the single-particle energies and wave functions.
- void FreeVmon ()

  Free up the memory used to store Vmon3.
- void GetRadialWF (index_t index, std::vector< double > &R, std::vector< double > &PSI)

  Return the radial wave function of an orbit in the HF basis.
- double GetRadialWF_r (index_t index, double R)

  Return the radial wave function of an orbit in the HF basis.
- void **FreezeOccupations** ()
- void **UnFreezeOccupations** ()
- uint64_t Vmon3Hash (uint64_t a, uint64_t b, uint64_t c, uint64_t d, uint64_t e, uint64_t f)
- void Vmon3UnHash (uint64_t key, int &a, int &b, int &c, int &d, int &e, int &f)

  Take a hashed key and extract the six orbit indices that went into it.

Public Attributes

- Operator & Hbare

  Input bare Hamiltonian.
- ModelSpace * modelspace

  Model Space of the Hamiltonian.
- arma::mat C

  transformation coefficients, 1st index is ho basis, 2nd = HF basis
- arma::mat rho

  density matrix rho_ij
- arma::mat KE

  kinetic energy
- arma::mat Vij

  1 body piece of 2 body potential
- arma::mat V3ij

  1 body piece of 3 body potential
- arma::mat F

  Fock matrix.
- std::array< std::array< arma::mat, 2 >, 3 > Vmon

  Monopole 2-body interaction.
- std::array< std::array< arma::mat, 2 >, 3 > Vmon_exch

  Monopole 2-body interaction.
- arma::uvec holeorbs

  list of hole orbits for generating density matrix
- arma::rowvec **hole_occ**
- arma::vec energies

  occupations of hole orbits
- arma::vec prev_energies

  SPE's from last iteration.

- double tolerance

  tolerance for convergence
- double EHF

  Hartree-Fock energy (Normal-ordered 0-body term)
- double e1hf

  One-body contribution to EHF.
- double e2hf

  Two-body contribution to EHF.
- double e3hf

  Three-body contribution to EHF.
- int iterations

  iterations used in Solve()
- std::vector< uint64_t > **Vmon3_keys**
- std::vector< double > **Vmon3**
- IMSRGProfiler profiler

  Profiler for timing, etc.
- std::deque< double > convergence_ediff

  Save last few convergence checks for diagnostics.
- std::deque< double > convergence_EHF

  Save last few convergence checks for diagnostics.
- bool **freeze_occupations**

7.3.1 Member Function Documentation

7.3.1.1 BuildMonopoleV()

```
void HartreeFock::BuildMonopoleV ( )
```

Only the monopole part of V is needed, so construct it.

Construct an unnormalized two-body monopole interaction

$$\langle ab|\bar{V}^{(2)}|cd\rangle = \sqrt{(1+\delta_{ab})(1+\delta_{cd})}\sum_J (2J+1)\langle ab|V^{(2)}|cd\rangle_J$$

This method utilizes the operator method TwoBodyME::GetTBMEmonopole()

7.3.1.2 BuildMonopoleV3()

```
void HartreeFock::BuildMonopoleV3 ( )
```

Only the monopole part of V3 is needed.

Construct an unnormalized three-body monopole interaction

$$\langle iab|\bar{V}^{(3)}|jcd\rangle = \sum_{J,J_{12}}\sum_{Tt_{12}}(2J+1)(2T+1)\langle (ia)J_{12}t_{12};bJT|V^{(3)}|(jc)J_{12}t_{12};dJT\rangle$$

7.3.1.3 CalcEHF()

```
void HartreeFock::CalcEHF ( )
```

Evaluate the Hartree Fock energy.

Calculate the HF energy.

$$\begin{aligned} E_{HF} &= \sum_{\alpha} t_{\alpha\alpha} + \frac{1}{2}\sum_{\alpha\beta} V_{\alpha\beta\alpha\beta} + \frac{1}{6}\sum_{\alpha\beta\gamma} V_{\alpha\beta\gamma\alpha\beta\gamma} \\ &= \sum_{ij}(2j_i+1)\rho_{ij}(t_{ij} + \frac{1}{2}\tilde{V}^{(2)}_{ij} + \frac{1}{6}\tilde{V}^{(3)}_{ij}) \end{aligned}$$

Where the matrices

$$\begin{aligned} \tilde{V}^{(2)}_{ij} &= \sum_{ab} \rho_{ab}\bar{V}^{(2)}_{iajb} \\ \tilde{V}^{(3)}_{ij} &= \sum_{abcd} \rho_{ab}\rho_{cd}\bar{V}^{(3)}_{iacjbd} \end{aligned}$$

have already been calculated by UpdateF().

7.3.1.4 CheckConvergence()

```
bool HartreeFock::CheckConvergence ( )
```

Compare the current energies with those from the previous iteration.

Check for convergence using difference in s.p. energies between iterations. Converged when

$$\delta_e \equiv \sqrt{\sum_i (e_i^{(n)} - e_i^{(n-1)})^2} < \text{tolerance}$$

where $e_i^{(n)}$ is the ith eigenvalue of the Fock matrix after n iterations.

7.3.1.5 Diagonalize()

```
void HartreeFock::Diagonalize ( )
```

Diagonalize the Fock matrix.

[See Suhonen eq. 4.85] Diagonalize the fock matrix $< a|F|b >$ and put the eigenvectors in $C(i,\alpha) =< i|\alpha >$ and eigenvalues in the vector energies. Save the last vector of energies to check for convergence. Submatrices corresponding to different channels are diagonalized independently. This guarantees that J,Tz, and π remain good.

7.3.1.6 FillLowestOrbits()

```
void HartreeFock::FillLowestOrbits ( )
```

Get new occupations based on the current single-particle energies.

Get new occupation numbers by filling the orbits in order of their single-particle energies. The last proton/neutron orbit can have a fractional filling, corresponding to ensemble normal ordering.

7.3.1.7 GetNormalOrderedH()

```
Operator HartreeFock::GetNormalOrderedH ( )
```

Return the Hamiltonian in the HF basis at the normal-ordered 2body level.

Returns the normal-ordered Hamiltonian in the Hartree-Fock basis, neglecting the residual 3-body piece.

$$E_0 = E_{HF}$$

$$f = C^\dagger F C$$

$$\Gamma = D^\dagger \left(V^{(2)} + V^{(3\to 2)} \right) D$$

$$V_{ijkl}^{(2\to 3)J} \equiv \frac{1}{\sqrt{(1+\delta_{ij})(1+\delta_{kl})}} \sum_{ab} \sum_{J_3} (2J_3+1) \rho_{ab} V_{ijaklb}^{JJJ_3}$$

Where F is the Fock matrix obtained in UpdateF() and the matrix D is the same as the one defined in TransformToHFBasis().

7.3.1.8 GetOmega()

```
Operator HartreeFock::GetOmega ( )
```

Return a generator of the Hartree Fock transformation.

Get the one-body generator corresponding to the transformation to the HF basis. Since the unitary transformation for HF is given by the $U_{HF} = C^\dagger$ matrix, we have $e^{-\Omega} = C \Rightarrow \Omega = -\log(C)$. The log is evaluated by diagonalizing the one-body submatrix and taking the log of the diagonal entries. This is much slower than the other methods, but it might be useful.

7.3.1.9 PrintEHF()

```
void HartreeFock::PrintEHF ( )
```

Print out the Hartree Fock energy.

Print out the Hartree Fock energy, and the 1-, 2-, and 3-body contributions to it.

7.3.1.10 PrintSPE()

```
void HartreeFock::PrintSPE ( )
```

Getter function for Hbare.

Print out the single-particle energies

7.3.1.11 ReorderCoefficients()

```
void HartreeFock::ReorderCoefficients ( )
```

Reorder the coefficients in C to eliminate phases etc.

Eigenvectors/values come out of the diagonalization energy-ordered. We want them ordered corresponding to the input ordering, i.e. we want the l,j,tz sub-blockes of the matrix C to be energy-ordered and positive along the diagonal. For a 3x3 matrix this would be something like (this needs to be updated)

$$\begin{pmatrix} -0.8 & 0.2 & -0.6 \\ -0.3 & 0.3 & 0.9 \\ 0.2 & 0.9 & -0.4 \end{pmatrix} \rightarrow \begin{pmatrix} 0.8 & -0.6 & 0.2 \\ 0.3 & 0.9 & 0.3 \\ -0.2 & -0.4 & 0.9 \end{pmatrix}$$

7.3.1.12 Solve()

```
void HartreeFock::Solve ( )
```

Diagonalize and UpdateF until convergence.

Diagonalize and update the Fock matrix until convergence. Then, call ReorderCoefficients() to make sure the index ordering and phases are preserved in the transformation from the original basis to the Hatree-Fock basis.

7.3.1.13 TransformToHFBasis()

```
Operator HartreeFock::TransformToHFBasis (
            Operator & OpHO )
```

Transform an operator from oscillator basis to HF basis.

Takes in an operator expressed in the basis of the original Hamiltonian, and returns that operator in the Hartree-Fock basis.

$$t_{HF} = C^\dagger t_{HO} C$$
$$V^J_{HF} = D^\dagger V^J_{HO} D$$

The matrix D is defined as

$$D_{ab\alpha\beta} \equiv \sqrt{\frac{1+\delta_{ab}}{1+\delta_{\alpha\beta}}} C_{a\alpha} C_{b\beta}$$

The factor in the square root is due to the fact that we're using normalized TBME's. Since only kets with $a \leq b$ are stored, we can use the antisymmetry of the TBME's and define

$$D(J)_{ab\alpha\beta} \equiv \sqrt{\frac{1+\delta_{ab}}{1+\delta_{\alpha\beta}}} \left(C_{a\alpha} C_{b\beta} - (1-\delta_{ab})(-1)^{j_a+j_b-J} C_{b\alpha} C_{a\beta} \right)$$

7.3.1.14 UpdateDensityMatrix()

```
void HartreeFock::UpdateDensityMatrix ( )
```

Update the density matrix with the new coefficients C.

one-body density matrix $< i|\rho|j >= \sum_\beta n_\beta < i|\beta >< \beta|j >$ where n_β ensures that beta runs over HF orbits in the core (i.e. below the fermi surface)

7.3.1.15 UpdateF()

```
void HartreeFock::UpdateF ( )
```

Update the Fock matrix with the new transformation coefficients C.

[See Suhonen eq 4.85]

$$F_{ij} = t_{ij} + \frac{1}{2j_i+1}\sum_{ab}\rho_{ab}\bar{V}^{(2)}_{iajb} + \frac{1}{2(2j_i+1)}\sum_{abcd}\rho_{ab}\rho_{cd}\bar{V}^{(3)}_{iacjbd}$$

- F is the Fock matrix, to be diagonalized
- t is the kinetic energy
- ρ is the density matrix defined in UpdateDensityMatrix()
- $\bar{V}^{(2)}$ is the monopole component of the 2-body interaction defined in BuildMonopoleV().
- $\bar{V}^{(3)}$ is the monopole component of the 3-body interaction devined in BuildMonopoleV3().

7.3.1.16 Vmon3Hash()

```
uint64_t HartreeFock::Vmon3Hash (
            uint64_t a,
            uint64_t b,
            uint64_t c,
            uint64_t d,
            uint64_t e,
            uint64_t f )
```

Hashing function for rolling six orbit indices into a single long unsigned int. Each orbit gets 10 bits.

7.3.2 Member Data Documentation

7.3.2.1 energies

```
arma::vec HartreeFock::energies
```

occupations of hole orbits

vector of single particle energies

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

- HartreeFock.hh
- HartreeFock.cc

7.4 std::hash< javier_state_t > Struct Template Reference

Public Member Functions

- size_t **operator()** (const javier_state_t &st) const

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

- ReadWrite.cc

7.5 IMSRGProfiler Class Reference

`#include <IMSRGProfiler.hh>`

Public Member Functions

- std::map< std::string, size_t > CheckMem ()
- std::map< std::string, float > **GetTimes** ()
- void **PrintTimes** ()
- void **PrintCounters** ()
- void **PrintMemory** ()
- void **PrintAll** ()
- size_t **MaxMemUsage** ()

Static Public Attributes

- static std::map< std::string, double > timer

  For keeping timing information for various method calls.
- static std::map< std::string, int > **counter**
- static float **start_time** = -1

7.5.1 Detailed Description

Profiling class with all static data members. This is for keeping track of timing and memory usage, etc. IMSRG↩ Profiler methods should probably not be called inside parallel blocks.

7.5.2 Member Function Documentation

7.5.2.1 CheckMem()

```
std::map< std::string, size_t > IMSRGProfiler::CheckMem ( )
```

Check how much memory is being used.

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

- IMSRGProfiler.hh
- IMSRGProfiler.cc

7.6 IMSRGSolver Class Reference

Classes

- class ODE_Monitor

Public Member Functions

- **IMSRGSolver** (Operator &H_in)
- void **NewOmega** ()
- void **SetHin** (Operator &H_in)
- void **SetReadWrite** (ReadWrite &r)
- void **Reset** ()
- void **AddOperator** (Operator &Op)
- void **UpdateEta** ()
- void **SetMethod** (string m)
- void **Solve** ()
- void **Solve_magnus_euler** ()
- void **Solve_magnus_modified_euler** ()
- Operator Transform (Operator &OpIn)
  Returns $e^{Omega}\mathcal{O}e^{-Omega}$.
- Operator **Transform** (Operator &&OpIn)
- Operator InverseTransform (Operator &OpIn)
  Returns $e^{-Omega}\mathcal{O}e^{Omega}$.
- Operator **GetOmega** (int i)
- void **SetOmega** (size_t i, Operator &om)
- size_t **GetOmegaSize** ()
- int **GetNOmegaWritten** ()
- Operator Transform_Partial (Operator &OpIn, int n)
- Operator **Transform_Partial** (Operator &&OpIn, int n)
- void **SetFlowFile** (string s)
- void **SetDs** (double d)
- void **SetDsmax** (double d)
- void **SetdOmega** (double d)
- void **SetSmax** (double d)
- void **SetGenerator** (string g)
- void **SetOmegaNormMax** (double x)
- void **SetODETolerance** (float x)
- void **SetEtaCriterion** (float x)
- void **SetMagnusAdaptive** (bool b)

- int **GetSystemDimension** ()
- Operator & **GetH_s** ()
- Operator & **GetEta** ()
- Generator & **GetGenerator** ()
- void **UpdateOmega** ()
- void **UpdateH** ()
- void **WriteFlowStatus** (ostream &)
- void **WriteFlowStatusHeader** (ostream &)
- void **WriteFlowStatus** (string)
- void **WriteFlowStatusHeader** (string)
- void **SetDenominatorCutoff** (double c)
- void **SetDenominatorDelta** (double d)
- void **SetDenominatorDeltaIndex** (int i)
- void **SetDenominatorDeltaOrbit** (string o)
- void **CleanupScratch** ()
- void **operator()** (const deque< Operator > &x, deque< Operator > &dxdt, const double t)
- void **Solve_ode** ()
- void **Solve_ode_adaptive** ()
- void **Solve_ode_magnus** ()

Public Attributes

- ModelSpace * **modelspace**
- ReadWrite * **rw**
- Operator * **H_0**
- deque< Operator > **FlowingOps**
- Operator **H_saved**
- Operator **Eta**
- deque< Operator > **Omega**
- Generator **generator**
- int **istep**
- double **s**
- double **ds**
- double **ds_max**
- double **smax**
- double **norm_domega**
- double **omega_norm_max**
- double **eta_criterion**
- string **method**
- string **flowfile**
- IMSRGProfiler **profiler**
- int **n_omega_written**
- int **max_omega_written**
- bool **magnus_adaptive**
- ODE_Monitor **ode_monitor**
- vector< double > **times**
- vector< double > **E0**
- vector< double > **eta1**
- vector< double > **eta2**
- string **ode_mode**
- float **ode_e_abs**
- float **ode_e_rel**

7.6.1 Member Function Documentation

7.6.1.1 Transform_Partial()

```
Operator IMSRGSolver::Transform_Partial (
            Operator & OpIn,
            int n )
```

Returns $e^{\Omega}\mathcal{O}e^{-\Omega}$ for the Ω_is with index greater than or equal to n.

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

- IMSRGSolver.hh
- IMSRGSolver.cc

7.7 javier_state_t Struct Reference

Public Member Functions

- **javier_state_t** (int e12_in, int n_in, int N_in, int J_in, int S_in, int L_in, int lam_in, int LAM_in, int T_in, int Tz_in)

Public Attributes

- int **e12**
- int **n**
- int **N**
- int **J**
- int **S**
- int **L**
- int **lam**
- int **LAM**
- int **T**
- int **Tz**

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

- ReadWrite.cc

7.8 Ket Class Reference

Public Member Functions

- **Ket** (Orbit &op, Orbit &oq)
- int **Phase** (int J)
- int **delta_pq** () const

Public Attributes

- Orbit * **op**
- Orbit * **oq**
- int **p**
- int **q**

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

- ModelSpace.hh
- ModelSpace.cc

7.9 ModelSpace Class Reference

Public Member Functions

- **ModelSpace** (const ModelSpace &)
- **ModelSpace** (ModelSpace &&)
- **ModelSpace** (int emax, std::vector< std::string > hole_list, std::vector< std::string > valence_list)
- **ModelSpace** (int emax, std::vector< std::string > hole_list, std::vector< std::string > core_list, std::vector< std::string > valence_list)
- **ModelSpace** (int emax, std::string reference, std::string valence)
- **ModelSpace** (int emax, std::string reference)
- ModelSpace **operator=** (const ModelSpace &)
- ModelSpace **operator=** (ModelSpace &&)
- void **Init** (int emax, std::string reference, std::string valence)
- void **Init** (int emax, std::map< index_t, double > hole_list, std::string valence)
- void **Init** (int emax, std::map< index_t, double > hole_list, std::vector< index_t > core_list, std::vector< index_t > valence_list)
- void **Init** (int emax, std::vector< std::string > hole_list, std::vector< std::string > core_list, std::vector< std::string > valence_list)
- void **Init_occ_from_file** (int emax, std::string valence, std::string occ_file)
- std::map< index_t, double > **GetOrbitsAZ** (int A, int Z)
- void **GetAZfromString** (std::string str, int &A, int &Z)
- std::vector< index_t > **String2Index** (std::vector< std::string > vs)
- std::string **Index2String** (index_t ind)
- void Get0hwSpace (int Aref, int Zref, std::vector< index_t > &core_list, std::vector< index_t > &valence_list)
- void **ParseCommaSeparatedValenceSpace** (std::string valence, std::vector< index_t > &core_list, std::vector< index_t > &valence_list)
- void **SetupKets** ()
- void **AddOrbit** (Orbit orb)
- void **AddOrbit** (int n, int l, int j2, int tz2, double occ, int io)
- Orbit & **GetOrbit** (int i)
- Ket & **GetKet** (int i) const
- Ket & **GetKet** (int p, int q) const
- int **GetOrbitIndex** (int n, int l, int j2, int tz2) const
- int **GetKetIndex** (int p, int q) const
- int **GetKetIndex** (Ket *ket) const
- int **GetNumberOrbits** () const
- int **GetNumberKets** () const
- void **SetHbarOmega** (double hw)
- void **SetTargetMass** (int A)

- void **SetTargetZ** (int Z)
- double **GetHbarOmega** () const
- int **GetTargetMass** () const
- int **GetTargetZ** () const
- int **GetAref** () const
- int **GetZref** () const
- int **GetNumberTwoBodyChannels** () const
- TwoBodyChannel & **GetTwoBodyChannel** (int ch) const
- TwoBodyChannel_CC & **GetTwoBodyChannel_CC** (int ch) const
- int **GetTwoBodyJmax** () const
- int **GetThreeBodyJmax** () const
- void **SetReference** (std::vector< index_t >)
- void **SetReference** (std::map< index_t, double >)
- void **SetReference** (std::string)
- int **GetEmax** ()
- int **GetE2max** ()
- int **GetE3max** ()
- int **GetLmax2** ()
- int **GetLmax3** ()
- void **SetEmax** (int e)
- void **SetE2max** (int e)
- void **SetE3max** (int e)
- void **SetLmax2** (int l)
- void **SetLmax3** (int l)
- double **GetSixJ** (double j1, double j2, double j3, double J1, double J2, double J3)
- double **GetNineJ** (double j1, double j2, double j3, double j4, double j5, double j6, double j7, double j8, double j9)
- double **GetMoshinsky** (int N, int Lam, int n, int lam, int n1, int l1, int n2, int l2, int L)
- bool **SixJ_is_empty** ()
- int **GetOrbitIndex** (std::string)
- int **GetTwoBodyChannelIndex** (int j, int p, int t)
- int **phase** (int x)
- int **phase** (double x)
- int **Index1** (int n, int l, int j2, int tz2) const
- int **Index2** (int p, int q) const
- void **PreCalculateMoshinsky** ()
- void PreCalculateSixJ ()
- void **ClearVectors** ()
- void **ResetFirstPass** ()
- void **CalculatePandyaLookup** (int rank_J, int rank_T, int parity)
- std::map< std::array< int, 2 >, std::array< std::vector< int >, 2 > > & **GetPandyaLookup** (int rank_J, int rank_T, int parity)
- uint64_t **SixJHash** (double j1, double j2, double j3, double J1, double J2, double J3)
- void **SixJUnHash** (uint64_t key, uint64_t &j1, uint64_t &j2, uint64_t &j3, uint64_t &J1, uint64_t &J2, uint64_t &J3)
- uint64_t **MoshinskyHash** (uint64_t N, uint64_t Lam, uint64_t n, uint64_t lam, uint64_t n1, uint64_t l1, uint64_t n2, uint64_t l2, uint64_t L)
- void **MoshinskyUnHash** (uint64_t key, uint64_t &N, uint64_t &Lam, uint64_t &n, uint64_t &lam, uint64_t &n1, uint64_t &l1, uint64_t &n2, uint64_t &l2, uint64_t &L)

Public Attributes

- std::vector< index_t > **holes**
- std::vector< index_t > **particles**
- std::vector< index_t > **core**
- std::vector< index_t > **valence**
- std::vector< index_t > **qspace**
- std::vector< index_t > **proton_orbits**
- std::vector< index_t > **neutron_orbits**
- std::vector< index_t > **KetIndex_pp**
- std::vector< index_t > **KetIndex_ph**
- std::vector< index_t > **KetIndex_hh**
- std::vector< index_t > **KetIndex_cc**
- std::vector< index_t > **KetIndex_vc**
- std::vector< index_t > **KetIndex_qc**
- std::vector< index_t > **KetIndex_vv**
- std::vector< index_t > **KetIndex_qv**
- std::vector< index_t > **KetIndex_qq**
- std::vector< double > **Ket_occ_hh**
- std::vector< double > **Ket_unocc_hh**
- std::vector< double > **Ket_occ_ph**
- std::vector< double > **Ket_unocc_ph**
- std::array< std::array< std::unordered_map< index_t, index_t >, 3 >, 3 > **MonopoleKets**
- int **Emax**
- int **E2max**
- int **E3max**
- int **Lmax2**
- int **Lmax3**
- int **OneBodyJmax**
- int **TwoBodyJmax**
- int **ThreeBodyJmax**
- std::map< std::array< int, 3 >, std::vector< index_t > > **OneBodyChannels**
- std::vector< unsigned int > **SortedTwoBodyChannels**
- std::vector< unsigned int > **SortedTwoBodyChannels_CC**
- int **norbits**
- double **hbar_omega**
- int **target_mass**
- int **target_Z**
- int **Aref**
- int **Zref**
- int **nTwoBodyChannels**
- std::vector< Orbit > **Orbits**
- std::vector< Ket > **Kets**
- std::vector< TwoBodyChannel > **TwoBodyChannels**
- std::vector< TwoBodyChannel_CC > **TwoBodyChannels_CC**
- std::map< std::array< int, 3 >, std::map< std::array< int, 2 >, std::array< std::vector< int >, 2 > > > **PandyaLookup**
- bool **sixj_has_been_precalculated**
- bool **moshinsky_has_been_precalculated**
- bool **scalar_transform_first_pass**
- std::vector< bool > **tensor_transform_first_pass**
- IMSRGProfiler **profiler**

Static Public Attributes

- static std::map< std::string, std::vector< std::string > > **ValenceSpaces**
- static std::unordered_map< uint64_t, double > **SixJList**
- static std::unordered_map< uint64_t, double > **NineJList**
- static std::unordered_map< uint64_t, double > **MoshList**

7.9.1 Member Function Documentation

7.9.1.1 Get0hwSpace()

```
void ModelSpace::Get0hwSpace (
            int Aref,
            int Zref,
            std::vector< index_t > & core_list,
            std::vector< index_t > & valence_list )
```

Find the valence space of one single major oscillator shell each for protons and neutrons (not necessarily the same shell for both) which contains the naive shell-model ground state of the reference. For example, if we want to treat C20, with 6 protons and 14 neutrons, we take the 0p shell for protons and 1s0d shell for neutrons.

7.9.1.2 PreCalculateSixJ()

```
void ModelSpace::PreCalculateSixJ ( )
```

Loop over all the 6j symbols that we expect to encounter, and store them in a hash table. Calculate all symbols

$$\{ja \quad jb \quad J1jc \quad jd \quad J2\}$$

and

$$\{J1 \quad J2 \quad J3ja \quad jb \quad jc\}$$

where ja,jb,jc are half-integer and J1,J2,J3 are integer. ja,jb,jc run from 1/2 to emax+1/2, while jd runs higher since the 3N recoupling requires it to go up to e(emax+1/2). I haven't yet bothered using the symmetry properties of the 6j symbol.

7.9.2 Member Data Documentation

7.9.2.1 ValenceSpaces

```
std::map< std::string, std::vector< std::string > > ModelSpace::ValenceSpaces  [static]
```

Initial value:

```
{
{ "s-shell"  ,          {"vacuum", "p0s1","n0s1"}},
{ "p-shell"  ,          {"He4", "p0p3","n0p3","p0p1","n0p1"}},
{ "sp-shell"  ,         {"vacuum", "p0s1","n0s1","p0p3","n0p3","p0p1","n0p1"}},
{ "spsd-shell"  ,       {"vacuum", "p0s1","n0s1","p0p3","n0p3","p0p1","n0p1","p0d5","n0d5","p0d3","n0d3","
    p1s1","n1s1"}},
{ "spsdpf-shell"  ,     {"vacuum", "p0s1","n0s1","p0p3","n0p3","p0p1","n0p1","p0d5","n0d5","p0d3","n0d3","
    p1s1","n1s1","p0f7","n0f7","p0f5","n0f5","p1p3","n1p3","p1p1","n1p1"}},
{ "sd-shell"  ,         {"O16", "p0d5","n0d5","p0d3","n0d3","p1s1","n1s1"}},
{ "psd-shell"  ,        {"He4", "p0p3","n0p3","p0p1","n0p1","p0d5","n0d5","p0d3","n0d3","p1s1","n1s1"}},
{ "psdNR-shell"  ,      {"He10","p0p3","p0p1","n0d5","n0d3","n1s1"}},
{ "psdPR-shell"  ,      {"O10","n0p3","n0p1","p0d5","p0d3","p1s1"}},
{ "fp-shell"  ,         {"Ca40","p0f7","n0f7","p0f5","n0f5","p1p3","n1p3","p1p1","n1p1"}},
{ "sdfp-shell"  ,       {"O16", "p0d5","n0d5","p0d3","n0d3","p1s1","n1s1","p0f7","n0f7","p0f5","n0f5","p1p3"
    ,"n1p3","p1p1","n1p1"}},
{ "sdfpNR-shell"  ,     {"O28", "p0d5","p0d3","p1s1","n0f7","n0f5","n1p3","n1p1"}},
{ "sdfpPR-shell"  ,     {"Ca28", "n0d5","n0d3","n1s1","p0f7","p0f5","p1p3","p1p1"}},
{ "fpg9-shell"  ,       {"Ca40","p0f7","n0f7","p0f5","n0f5","p1p3","n1p3","p1p1","n1p1","p0g9","n0g9"}},
{ "fpg9NR-shell"  ,     {"Ca40","p0f7","n0f7","p0f5","n0f5","p1p3","n1p3","p1p1","n1p1","n0g9"}},
{ "fpgdsNR-shell"  ,    {"Ca60","p0f7","p0f5","p1p3","p1p1","n0g9","n0g7","n1d5","n1d3","n2s1"}},

{ "sd3f7p3-shell"  ,    {"Si28","p0d3","n0d3","p1s1","n1s1","p0f7","n0f7","p1p3","n1p3"}},
{ "gds-shell" ,         {"Zr80","p0g9","n0g9","p0g7","n0g7","p1d5","n1d5","p1d3","n1d3","p2s1","n2s1"}},
}
```

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

- ModelSpace.hh
- ModelSpace.cc

7.10 IMSRGSolver::ODE_Monitor Class Reference

Public Member Functions

- **ODE_Monitor** (IMSRGSolver &solver)
- void **operator()** (const deque< Operator > &x, double t)
- void **report** ()

Public Attributes

- IMSRGSolver & **imsrgsolver**
- vector< double > & **times**
- vector< double > & **E0**
- vector< double > & **eta1**
- vector< double > & **eta2**

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

- IMSRGSolver.hh

7.11 Operator Class Reference

`#include <Operator.hh>`

Public Member Functions

- Operator ()
  Default constructor.
- Operator (ModelSpace &)
  Construct a 2-body scalar operator.
- **Operator** (ModelSpace &, int Jrank, int Trank, int Parity, int part_rank)
- Operator (const Operator &rhs)
  Copy constructor.
- **Operator** (Operator &&)
- Operator & **operator=** (const Operator &rhs)
- Operator & **operator+=** (const Operator &rhs)
- Operator **operator+** (const Operator &rhs) const
- Operator & **operator+=** (const double &rhs)
- Operator **operator+** (const double &rhs) const
- Operator & **operator-=** (const Operator &rhs)
- Operator **operator-** (const Operator &rhs) const
- Operator **operator-** () const
- Operator & **operator-=** (const double &rhs)
- Operator **operator-** (const double &rhs) const
- Operator & **operator*=** (const double rhs)
- Operator **operator*** (const double rhs) const
- Operator & **operator/=** (const double rhs)
- Operator **operator/** (const double rhs) const
- Operator & **operator=** (Operator &&rhs)
- Operator & TempOp (size_t n)
  Static scratch space for calculations.
- double **GetOneBody** (int i, int j)
- void **SetOneBody** (int i, int j, double val)
- int **GetTwoBodyDimension** (int ch_bra, int ch_ket)
- double **GetTwoBody** (int ch_bra, int ch_ket, int i, int j)
- void **SetTwoBody** (int J1, int p1, int T1, int J2, int p2, int T2, int i, int j, int k, int l, double v)
- void **SetE3max** (int e)
- int **GetE3max** ()
- ModelSpace * **GetModelSpace** ()
- void **SetModelSpace** (ModelSpace &ms)
- void Erase ()
  Set all matrix elements to zero.
- void **EraseZeroBody** ()
- void EraseOneBody ()
  set zero-body term to zero
- void EraseTwoBody ()
  set all two-body terms to zero
- void EraseThreeBody ()
  set all two-body terms to zero
- void **SetHermitian** ()
- void **SetAntiHermitian** ()
- void **SetNonHermitian** ()

- bool **IsHermitian** () const
- bool **IsAntiHermitian** () const
- bool **IsNonHermitian** () const
- int **GetParticleRank** () const
- int **GetJRank** () const
- int **GetTRank** () const
- int **GetParity** () const
- void **SetParticleRank** (int pr)
- void **MakeReduced** ()
- void **MakeNotReduced** ()
- void **ChangeNormalization** (double coeff)
- void **MakeNormalized** ()
- void **MakeUnNormalized** ()
- void **ScaleZeroBody** (double x)
- void **ScaleOneBody** (double x)
- void **ScaleTwoBody** (double x)
- void Symmetrize ()

  Copy the upper-half triangle to the lower-half triangle for each matrix.
- void AntiSymmetrize ()

  Copy the upper-half triangle to the lower-half triangle with a minus sign.
- void **SetUpOneBodyChannels** ()
- size_t **Size** ()
- void **WriteBinary** (std::ofstream &ofs)
- void **ReadBinary** (std::ifstream &ifs)
- Operator DoNormalOrdering ()

  Calls DoNormalOrdering2() or DoNormalOrdering3(), depending on the rank of the operator.
- Operator DoNormalOrdering2 ()

  Returns the normal ordered two-body operator.
- Operator DoNormalOrdering3 ()

  Returns the normal ordered three-body operator.
- Operator UndoNormalOrdering () const

  Returns the operator normal-ordered wrt the vacuum.
- Operator Truncate (ModelSpace &ms_new)

  Returns the operator trunacted to the new model space.
- void **SetToCommutator** (const Operator &X, const Operator &Y)
- void CommutatorScalarScalar (const Operator &X, const Operator &Y)
- void CommutatorScalarTensor (const Operator &X, const Operator &Y)
- Operator BCH_Product (Operator &)
- Operator BCH_Transform (const Operator &)
- Operator Standard_BCH_Transform (const Operator &)
- Operator Brueckner_BCH_Transform (const Operator &)
- void **CalculateKineticEnergy** ()
- void Eye ()

  set to identity operator – unused
- double GetMP2_Energy ()
- double GetMP3_Energy ()
- double MP1_Eval (Operator &)
- void **PrintTimes** ()
- double Norm () const
- double **OneBodyNorm** () const
- double **TwoBodyNorm** () const
- double **Trace** (int Atrace, int Ztrace) const
- void **ScaleFermiDirac** (Operator &H, double T, double Efermi)

- void **PrintOneBody** () const
- void **PrintTwoBody** (int ch) const
- std::deque< arma::mat > **InitializePandya** (size_t nch, std::string orientation)
- void **DoPandyaTransformation** (std::deque< arma::mat > &, std::string orientation) const
- void DoPandyaTransformation_SingleChannel (arma::mat &X, int ch_cc, std::string orientation) const
- void **AddInversePandyaTransformation** (const std::deque< arma::mat > &)
- void **AddInversePandyaTransformation_SingleChannel** (arma::mat &Z, int ch_cc)
- void comm110ss (const Operator &X, const Operator &Y)
- void comm220ss (const Operator &X, const Operator &Y)
- void comm111ss (const Operator &X, const Operator &Y)
- void comm121ss (const Operator &X, const Operator &Y)
- void comm221ss (const Operator &X, const Operator &Y)
- void comm122ss (const Operator &X, const Operator &Y)
- void comm222_pp_hhss (const Operator &X, const Operator &Y)
- void comm222_phss (const Operator &X, const Operator &Y)
- void comm222_pp_hh_221ss (const Operator &X, const Operator &Y)
- void **GooseTankUpdate** (const Operator &Omega, const Operator &Nested)
- void **ConstructScalarMpp_Mhh** (const Operator &X, const Operator &Y, TwoBodyME &Mpp, TwoBodyME &Mhh) const
- void **ConstructScalarMpp_Mhh_GooseTank** (const Operator &X, const Operator &Y, TwoBodyME &Mpp, TwoBodyME &Mhh) const
- void DoTensorPandyaTransformation (std::map< std::array< index_t, 2 >, arma::mat > &) const
- void **DoTensorPandyaTransformation_SingleChannel** (arma::mat &X, int ch_bra_cc, int ch_ket_cc) const
- void **AddInverseTensorPandyaTransformation** (const std::map< std::array< index_t, 2 >, arma::mat > &)
- void **AddInverseTensorPandyaTransformation_SingleChannel** (arma::mat &Zbar, int ch_bra_cc, int ch↩_ket_cc)
- void **comm111st** (const Operator &X, const Operator &Y)
- void **comm121st** (const Operator &X, const Operator &Y)
- void **comm122st** (const Operator &X, const Operator &Y)
- void **comm222_pp_hh_221st** (const Operator &X, const Operator &Y)
- void comm222_phst (const Operator &X, const Operator &Y)

Static Public Member Functions

- static void **Set_BCH_Transform_Threshold** (double x)
- static void **Set_BCH_Product_Threshold** (double x)
- static void **SetUseBrueckner BCH** (bool tf)
- static void **SetUseGooseTank** (bool tf)

Public Attributes

- ModelSpace * modelspace

  Pointer to the associated modelspace.
- double ZeroBody

  The zero body piece of the operator.
- arma::mat OneBody

  The one body piece of the operator, stored in a single NxN armadillo matrix, where N is the number of single-particle orbits.
- TwoBodyME TwoBody

  The two body piece of the operator.
- ThreeBodyME ThreeBody

The three body piece of the operator.

- int rank_J

  Spherical tensor rank of the operator.
- int rank_T

  Isotensor rank of the operator.
- int parity

  Parity of the operator, 0=even 1=odd.
- int particle_rank

  Maximum particle rank. Should be 2 or 3.
- int E2max

  For two-body matrix elements, $e_i + e_j \leq$ E2max.
- int E3max

  For three-body matrix elements, $e_i + e_j + e_k \leq$ E3max.
- bool **hermitian**
- bool **antihermitian**
- int nChannels

  Number of two-body channels J, π, T_z associated with the model space.
- std::map< std::array< int, 3 >, std::vector< index_t > > **OneBodyChannels**
- IMSRGProfiler **profiler**

Static Public Attributes

- static double **bch_transform_threshold** = 1e-9
- static double **bch_product_threshold** = 1e-4
- static bool **use_brueckner_bch** = false
- static bool **use_goose_tank_correction** = false
- static bool **use_goose_tank_correction_titus** = false

Friends

- Operator Commutator (const Operator &X, const Operator &Y)

7.11.1 Detailed Description

The Operator class provides a generic operator up to three-body, scalar or tensor. The class contains lots of methods and overloaded operators so that the resulting code that uses the operators can look as close as possible to the math that is written down.

7.11.2 Member Function Documentation

7.11.2.1 BCH_Product()

```
Operator Operator::BCH_Product (
            Operator & Y )
```

X.BCH_Product(Y) returns Z such that $e^Z = e^X e^Y$ by employing the Baker-Campbell-Hausdorff formula

$$Z = X + Y + \frac{1}{2}[X,Y] + \frac{1}{12}([X,[X,Y]] + [Y,[Y,X]]) + \ldots$$

7.11.2.2 BCH_Transform()

```
Operator Operator::BCH_Transform (
            const Operator & Omega )
```

Calculates the kinetic energy operator in the harmonic oscillator basis.

$$t_{ab} = \frac{1}{2}\hbar\omega\delta_{\ell_a\ell_b}\delta_{j_aj_b}\delta_{t_{za}t_{zb}} \begin{cases} 2n_a + \ell_a + \frac{3}{2} & : n_a = n_b \\ \sqrt{n_a(n_a + \ell_a + \frac{1}{2})} & : n_a = n_b + 1 \end{cases}$$

X.BCH_Transform(Y) returns $Z = e^Y X e^{-Y}$. We use the Baker-Campbell-Hausdorff formula

$$Z = X + [Y, X] + \frac{1}{2!}[Y, [Y, X]] + \frac{1}{3!}[Y, [Y, [Y, X]]] + \ldots$$

with all commutators truncated at the two-body level.

7.11.2.3 Brueckner_BCH_Transform()

```
Operator Operator::Brueckner_BCH_Transform (
            const Operator & Omega )
```

Variation of the BCH transformation procedure requested by a one Dr. T.D. Morris

$$e^{\Omega_1+\Omega_2} X e^{-\Omega_1-\Omega_2} \rightarrow e^{\Omega_2} e^{\Omega_1} X e^{-\Omega_1} e^{-\Omega_2}$$

7.11.2.4 comm110ss()

```
void Operator::comm110ss (
            const Operator & X,
            const Operator & Y )
```

$$[X_1), Y_{(1)}]_{(0)} = \sum_a n_a(2j_a + 1) \left(X_{(1)}Y_{(1)} - Y_{(1)}X_{(1)}\right)_{aa}$$

7.11.2.5 comm111ss()

```
void Operator::comm111ss (
            const Operator & X,
            const Operator & Y )
```

$$[X_{(1)}, Y_{(1)}]_{(1)} = X_{(1)}Y_{(1)} - Y_{(1)}X_{(1)}$$

7.11.2.6 comm121ss()

```
void Operator::comm121ss (
             const Operator & X,
             const Operator & Y )
```

Returns $[X_{(1)}, Y_{(2)}] - [Y_{(1)}, X_{(2)}]$, where

$$[X_{(1)}, Y_{(2)}]_{ij} = \frac{1}{2j_i+1} \sum_{ab} (n_a \bar{n}_b) \sum_J (2J+1)(X_{ab} Y^J_{biaj} - X_{ba} Y^J_{aibj})$$

7.11.2.7 comm122ss()

```
void Operator::comm122ss (
             const Operator & X,
             const Operator & Y )
```

Returns $[X_{(1)}, Y_{(2)}]_{(2)} - [Y_{(1)}, X_{(2)}]_{(2)}$, where

$$[X_{(1)}, Y_{(2)}]^J_{ijkl} = \sum_a (X_{ia} Y^J_{ajkl} + X_{ja} Y^J_{iakl} - X_{ak} Y^J_{ijal} - X_{al} Y^J_{ijka})$$

here, all TBME are unnormalized, i.e. they should have a tilde.

7.11.2.8 comm220ss()

```
void Operator::comm220ss (
             const Operator & X,
             const Operator & Y )
```

$$[X_{(2)}, Y_{(2)}]_{(0)} = \frac{1}{2} \sum_J (2J+1) \sum_{abcd} (n_a n_b \bar{n}_c \bar{n}_d) \tilde{X}^J_{abcd} \tilde{Y}^J_{cdab}$$

may be rewritten as

$$[X_{(2)}, Y_{(2)}]_{(0)} = 2 \sum_J (2J+1) Tr(X^J_{hh'pp'} Y^J_{pp'hh'})$$

where we obtain a factor of four from converting two unrestricted sums to restricted sums, i.e. $\sum_{ab} \to \sum_{a \leq b}$, and using the normalized TBME.

7.11.2.9 comm221ss()

```
void Operator::comm221ss (
             const Operator & X,
             const Operator & Y )
```

$$[X_{(2)}, Y_{(2)}]_{ij} = \frac{1}{2(2j_i+1)} \sum_J (2J+1) \sum_{abc} (\bar{n}_a \bar{n}_b n_c + n_a n_b \bar{n}_c)(X^J_{ciab} Y^J_{abcj} - Y^J_{ciab} X^J_{abcj})$$

This may be rewritten as

$$[X_{(2)}, Y_{(2)}]_{ij} = \frac{1}{2j_i+1} \sum_c \sum_J (2J+1) \left(n_c \mathcal{M}^J_{pp,icjc} + \bar{n}_c \mathcal{M}^J_{hh,icjc} \right)$$

With the intermediate matrix

$$\mathcal{M}^J_{pp} \equiv \frac{1}{2} (X^J \mathcal{P}_{pp} Y^J - Y^J \mathcal{P}_{pp} X^J)$$

and likewise for $\mathcal{M}^J_{hh}$

7.11.2.10 comm222_phss()

```
void Operator::comm222_phss (
            const Operator & X,
            const Operator & Y )
```

Calculates the part of $[X_{(2)}, Y_{(2)}]_{ijkl}$ which involves ph intermediate states, here indicated by Z^J_{ijkl}

$$Z^J_{ijkl} = \sum_{ab}(n_a\bar{n}_b - \bar{n}_a n_b)\sum_{J'}(2J'+1)\left[\begin{Bmatrix} j_i & j_j & J \\ j_k & j_l & J' \end{Bmatrix}\left(\bar{X}^{J'}_{i\bar{l}a\bar{b}}\bar{Y}^{J'}_{a\bar{b}k\bar{j}} - \bar{Y}^{J'}_{i\bar{l}a\bar{b}}\bar{X}^{J'}_{a\bar{b}k\bar{j}}\right) - (-1)^{j_i+j_j-J}\begin{Bmatrix} j_j & j_i & J \\ j_k & j_l & J' \end{Bmatrix}\left(\bar{X}^{J'}_{j\bar{l}a\bar{b}}\bar{Y}_{a}\right.\right.$$

This is implemented by defining an intermediate matrix

$$\bar{Z}^J_{i\bar{l}k\bar{j}} \equiv \sum_{ab}(n_a\bar{n}_b)\left[\left(\bar{X}^{J'}_{i\bar{l}a\bar{b}}\bar{Y}^{J'}_{a\bar{b}k\bar{j}} - \bar{Y}^{J'}_{i\bar{l}a\bar{b}}\bar{X}^{J'}_{a\bar{b}k\bar{j}}\right) - \left(\bar{X}^{J'}_{i\bar{l}b\bar{a}}\bar{Y}^{J'}_{b\bar{a}k\bar{j}} - \bar{Y}^{J'}_{i\bar{l}b\bar{a}}\bar{X}^{J'}_{b\bar{a}k\bar{j}}\right)\right]$$

The Pandya-transformed matrix elements are obtained with DoPandyaTransformation(). The matrices $\bar{X}^{J'}_{i\bar{l}a\bar{b}}\bar{Y}^{J'}_{a\bar{b}k\bar{j}}$ and $\bar{Y}^{J'}_{i\bar{l}a\bar{b}}\bar{X}^{J'}_{a\bar{b}k\bar{j}}$ are related by a Hermitian conjugation, which saves two matrix multiplications. The commutator is then given by

$$Z^J_{ijkl} = \sum_{J'}(2J'+1)\left[\begin{Bmatrix} j_i & j_j & J \\ j_k & j_l & J' \end{Bmatrix}\bar{Z}^{J'}_{i\bar{l}k\bar{j}} - (-1)^{j_i+j_j-J}\begin{Bmatrix} j_j & j_i & J \\ j_k & j_l & J' \end{Bmatrix}\bar{Z}^{J'}_{j\bar{l}k\bar{i}}\right]$$

7.11.2.11 comm222_phst()

```
void Operator::comm222_phst (
            const Operator & X,
            const Operator & Y )
```

Calculates the part of $[X_{(2)}, \mathbb{Y}^{\Lambda}_{(2)}]_{ijkl}$ which involves ph intermediate states, here indicated by $\mathbb{Z}^{J_1J_2\Lambda}_{ijkl}$

$$\mathbb{Z}^{J_1J_2\Lambda}_{ijkl} = \sum_{abJ_3J_4}(n_a - n_b)\hat{J}_1\hat{J}_2\hat{J}_3\hat{J}_4\left[\begin{Bmatrix} j_i & j_l & J_3 \\ j_j & j_k & J_4 \\ J_1 & J_2 & \Lambda \end{Bmatrix}\left(\bar{X}^{J3}_{i\bar{l}a\bar{b}}\bar{\mathbb{Y}}^{J_3J_4\Lambda}_{a\bar{b}k\bar{j}} - \bar{\mathbb{Y}}^{J_3J_4\Lambda}_{i\bar{l}a\bar{b}}\bar{X}^{J_4}_{a\bar{b}k\bar{j}}\right) - (-1)^{j_i+j_j-J_1}\begin{Bmatrix} j_j & j_l & J_3 \\ j_i & j_k & J_4 \\ J_1 & J_2 & \Lambda \end{Bmatrix}\left(\bar{X}_{\right.\right.$$

This is implemented by defining an intermediate matrix

$$\bar{\mathbb{Z}}^{J_3J_4\Lambda}_{i\bar{l}k\bar{j}} \equiv \sum_{ab}(n_a\bar{n}_b)\left[\left(\bar{X}^{J3}_{i\bar{l}a\bar{b}}\bar{\mathbb{Y}}^{J_3J_4\Lambda}_{a\bar{b}k\bar{j}} - \bar{\mathbb{Y}}^{J_3J_4\Lambda}_{i\bar{l}a\bar{b}}\bar{X}^{J_4}_{a\bar{b}k\bar{j}}\right) - \left(\bar{X}^{J_3}_{i\bar{l}b\bar{a}}\bar{\mathbb{Y}}^{J_3J_4\Lambda}_{b\bar{a}k\bar{j}} - \bar{\mathbb{Y}}^{J_3J_4\Lambda}_{i\bar{l}b\bar{a}}\bar{X}^{J_4}_{b\bar{a}k\bar{j}}\right)\right]$$

The Pandya-transformed matrix elements are obtained with DoTensorPandyaTransformation(). The matrices $\bar{X}^{J_3}_{i\bar{l}a\bar{b}}\bar{\mathbb{Y}}^{J_3J_4\Lambda}_{a\bar{b}k\bar{j}}$ and $\bar{\mathbb{Y}}^{J_4J_3\Lambda}_{i\bar{l}a\bar{b}}\bar{X}^{J_3}_{a\bar{b}k\bar{j}}$ are related by a Hermitian conjugation, which saves two matrix multiplications, provided we take into account the phase $(-1)^{J_3-J_4}$ from conjugating the spherical tensor. The commutator is then given by

$$\mathbb{Z}^{J_1J_2\Lambda}_{ijkl} = \sum_{J_3J_4}\hat{J}_1\hat{J}_2\hat{J}_3\hat{J}_4\left[\begin{Bmatrix} j_i & j_l & J_3 \\ j_j & j_k & J_4 \\ J_1 & J_2 & \Lambda \end{Bmatrix}\bar{\mathbb{Z}}^{J_3J_4\Lambda}_{i\bar{l}k\bar{j}} - (-1)^{j_i+j_j-J}\begin{Bmatrix} j_j & j_l & J_3 \\ j_i & j_k & J_4 \\ J_1 & J_2 & \Lambda \end{Bmatrix}\bar{\mathbb{Z}}^{J_3J_4\Lambda}_{j\bar{l}k\bar{i}}\right]$$

7.11.2.12 comm222_pp_hh_221ss()

```
void Operator::comm222_pp_hh_221ss (
            const Operator & X,
            const Operator & Y )
```

Since comm222_pp_hhss() and comm221ss() both require the construction of the intermediate matrices $\mathcal{M}_{pp}$ and $\mathcal{M}_{hh}$, we can combine them and only calculate the intermediates once.

7.11.2.13 comm222_pp_hhss()

```
void Operator::comm222_pp_hhss (
            const Operator & X,
            const Operator & Y )
```

Calculates the part of the commutator $[X_{(2)}, Y_{(2)}]_{(2)}$ which involves particle-particle or hole-hole intermediate states.

$$[X_{(2)}, Y_{(2)}]^J_{ijkl} = \frac{1}{2}\sum_{ab}(\bar{n}_a\bar{n}_b - n_a n_b)(X^J_{ijab}Y^J_{ablk} - Y^J_{ijab}X^J_{abkl})$$

This may be written as

$$[X_{(2)}, Y_{(2)}]^J = \mathcal{M}^J_{pp} - \mathcal{M}^J_{hh}$$

With the intermediate matrices

$$\mathcal{M}^J_{pp} \equiv \frac{1}{2}(X^J\mathcal{P}_{pp}Y^J - Y^J\mathcal{P}_{pp}X^J)$$

and likewise for $\mathcal{M}^J_{hh}$.

7.11.2.14 CommutatorScalarScalar()

```
void Operator::CommutatorScalarScalar (
            const Operator & X,
            const Operator & Y )
```

Commutator where X and Y are scalar operators. Should be called through Commutator()

7.11.2.15 CommutatorScalarTensor()

```
void Operator::CommutatorScalarTensor (
            const Operator & X,
            const Operator & Y )
```

Commutator $[X, Y]$ where X is a scalar operator and Y is a tensor operator. Should be called through Commutator()

7.11.2.16 DoNormalOrdering2()

```
Operator Operator::DoNormalOrdering2 ( )
```

Returns the normal ordered two-body operator.

Normal ordering of a 2body operator set up for scalar or tensor operators, but the tensor part hasn't been tested

7.11.2.17 DoNormalOrdering3()

```
Operator Operator::DoNormalOrdering3 ( )
```

Returns the normal ordered three-body operator.

Normal ordering of a three body operator. Start by generating the normal ordered two body piece, then use DoNormalOrdering2() to get the rest. (Note that there are some numerical factors). The normal ordered two body piece is

$$\Gamma^J_{ijkl} = V^J_{ijkl} + \sum_a n_a \sum_K \frac{2K+1}{2J+1} V^{(3)JJK}_{ijakla}$$

Right now, this is only set up for scalar operators, but I don't anticipate handling 3body tensor operators in the near future.

7.11.2.18 DoPandyaTransformation_SingleChannel()

```
void Operator::DoPandyaTransformation_SingleChannel (
            arma::mat & X,
            int ch_cc,
            std::string orientation ) const
```

The scalar Pandya transformation is defined as

$$\bar{X}^J_{i\bar{j}k\bar{l}} = -\sum_{J'}(2J'+1)\begin{Bmatrix} j_i & j_j & J \\ j_k & j_l & J' \end{Bmatrix} X^J_{ilkj}$$

where the overbar indicates time-reversed orbits.

7.11.2.19 DoTensorPandyaTransformation()

```
void Operator::DoTensorPandyaTransformation (
            std::map< std::array< index_t, 2 >, arma::mat > &  ) const
```

The scalar Pandya transformation is defined as

$$\bar{X}^J_{i\bar{j}k\bar{l}} = -\sum_{J'}(2J'+1)\begin{Bmatrix} j_i & j_j & J \\ j_k & j_l & J' \end{Bmatrix} X^J_{ilkj}$$

where the overbar indicates time-reversed orbits. This function is designed for use with comm222_phss() and so it takes in two arrays of matrices, one for hp terms and one for ph terms.

7.11.2.20 EraseOneBody()

```
void Operator::EraseOneBody ( )
```

set zero-body term to zero

set all one-body terms to zero

7.11.2.21 GetMP2_Energy()

```
double Operator::GetMP2_Energy ( )
```

Calculate the second-order perturbation theory correction to the energy

$$E^{(2)} = \sum_{ia}(2j_a+1)\frac{|f_{ia}|^2}{f_{aa}-f_{ii}} + \sum_{i\leq j//a\leq b}\sum_J(2J+1)\frac{|\Gamma^J_{ijab}|^2}{f_{aa}+f_{bb}-f_{ii}-f_{jj}}$$

7.11.2.22 GetMP3_Energy()

```
double Operator::GetMP3_Energy ( )
```

Calculate the third order perturbation correction to energy

$$\frac{1}{8}\sum_{abijpq}\sum_J(2J+1)\frac{\Gamma^J_{abij}\Gamma^J_{ijpq}\Gamma^J_{pqab}}{(f_a+f_b-f_p-f_q)(f_a+f_b-f_i-f_j)} + \sum_{abcijk}\sum_J(2J+1)^2\frac{\bar{\Gamma}^J_{a\bar{i}j\bar{b}}\bar{\Gamma}^J_{j\bar{b}k\bar{c}}\bar{\Gamma}^J_{k\bar{c}a\bar{i}}}{(f_a+f_b-f_i-f_j)(f_a+f_c-f_i-f_k)}$$

7.11.2.23 MP1_Eval()

```
double Operator::MP1_Eval (
            Operator & H )
```

Evaluate first order perturbative correction to the operator's ground-state expectation value. A HF basis is assumed.

$$\mathcal{O}^{(1)} = 2 \sum_{abij} \frac{H_{abij} \mathcal{O}_{ijab}}{\Delta_{abij}}$$

7.11.2.24 Norm()

```
double Operator::Norm ( ) const
```

Obtain the Frobenius norm of the operator, which here is defined as

$$\|X\| = \sqrt{\|X_{(1)}\|^2 + \|X_{(2)}\|^2}$$

and

$$\|X_{(1)}\|^2 = \sum_{ij} X_{ij}^2$$

7.11.2.25 Standard_BCH_Transform()

```
Operator Operator::Standard_BCH_Transform (
            const Operator & Omega )
```

X.BCH_Transform(Y) returns $Z = e^Y X e^{-Y}$. We use the Baker-Campbell-Hausdorff formula

$$Z = X + [Y, X] + \frac{1}{2!}[Y, [Y, X]] + \frac{1}{3!}[Y, [Y, [Y, X]]] + \ldots$$

with all commutators truncated at the two-body level.

7.11.2.26 Truncate()

```
Operator Operator::Truncate (
            ModelSpace & ms_new )
```

Returns the operator trunacted to the new model space.

Truncate an operator to a smaller emax A corresponding ModelSpace object must be created at the appropriate scope. That's why the new operator is passed as a

7.11.2.27 UndoNormalOrdering()

```
Operator Operator::UndoNormalOrdering ( ) const
```

Returns the operator normal-ordered wrt the vacuum.

Convert to a basis normal ordered wrt the vacuum. This doesn't handle 3-body terms. In that case, the 2-body piece is unchanged.

7.11.3 Friends And Related Function Documentation

7.11.3.1 Commutator

```
Operator Commutator (
            const Operator & X,
            const Operator & Y )  [friend]
```

Returns $Z = [X, Y]$

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

- Operator.hh
- Operator.cc

7.12 Orbit Class Reference

Public Member Functions

- **Orbit** (int n, int l, int j, int t, double occ, int cvq, int index)
- **Orbit** (const Orbit &)

Public Attributes

- int **n**
- int **l**
- int **j2**
- int **tz2**
- double **occ**
- int **cvq**
- int **index**

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

- ModelSpace.hh
- ModelSpace.cc

7.13 Parameters Class Reference

Public Member Functions

- **Parameters** (int, char $**$)
- void **ParseCommandLineArgs** (int, char $**$)
- void **PrintOptions** ()
- string **s** (string)
- double **d** (string)
- int **i** (string)
- vector$<$ string $>$ **v** (string)
- string **DefaultFlowFile** ()
- string **DefaultIntFile** ()

Public Attributes

- bool **help_mode**

Static Public Attributes

- static map< string, string > **string_par**
- static map< string, double > **double_par**
- static map< string, int > **int_par**
- static map< string, vector< string > > **vec_par**

7.13.1 Member Data Documentation

7.13.1.1 double_par

```
map< string, double > Parameters::double_par  [static]
```

Initial value:

```
= {
  {"hw",         20.0},
  {"smax",       200.0},
  {"dsmax",      0.5},
  {"ds_0",       0.5},
  {"domega",         0.2},
  {"omega_norm_max",     0.25},
  {"ode_tolerance", 1e-6},
  {"denominator_delta",     0},
  {"BetaCM",                0},
  {"hwBetaCM",             -1},
  {"eta_criterion",      1e-6},

}
```

7.13.1.2 int_par

```
map< string, int > Parameters::int_par  [static]
```

Initial value:

```
= {
  {"A", -1},
  {"e3max",      12},
  {"emax",       6},
  {"lmax3",      -1},
  {"nsteps",         -1},
  {"file2e1max",     12},
  {"file2e2max",     24},
  {"file2lmax",      10},
  {"file3e1max",     12},
  {"file3e2max",     24},
  {"file3e3max",     12},
}
```

7.13.1.3 string_par

```
map< string, string > Parameters::string_par  [static]
```

Initial value:

```
= {
  {"2bme",          "none"},
  {"3bme",          "none"},
  {"core_generator",        "atan"},
  {"valence_generator",     "shell-model-atan"},
  {"flowfile",          "default"},
  {"intfile",           "default"},
  {"fmt2",          "me2j"},
  {"fmt3",          "me3j"},
  {"reference",         "default"},
  {"valence_space",     ""},
  {"custom_valence_space",      ""},
  {"basis",         "HF"},
  {"method",            "magnus"},
  {"denominator_delta_orbit",   "none"},
  {"LECs",          "EM2.0_2.0"},
  {"scratch",           ""},
  {"use_brueckner_bch",          "false"},
  {"valence_file_format",       "nushellx"},
  {"occ_file",          "none"},
  {"goose_tank",        "false"},
  {"write_omega",       "false"},
  {"nucleon_mass_correction",   "false"},
}
```

7.13.1.4 vec_par

```
map< string, vector< string > > Parameters::vec_par  [static]
```

Initial value:

```
= {
 {"Operators", {} },
 {"OperatorsFromFile", {} },
 {"SPWF",{} },
}
```

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

- Parameters.hh

7.14 ReadWrite Class Reference

Public Member Functions

- void **ReadSettingsFile** (std::string filename)
- void ReadTBME_Oslo (std::string filename, Operator &Hbare)

  Read two-body matrix elements from an Oslo-formatted file.
- void ReadTBME_OakRidge (std::string spname, std::string tbmename, Operator &Hbare, std::string format)

  Read two-body matrix elements from an Oslo-formatted file, as obtained from Gaute Hagen.
- void **ReadBareTBME_Jason** (std::string filename, Operator &Hbare)

- void ReadBareTBME_Navratil (std::string filename, Operator &Hbare)
- void **ReadBareTBME_Navratil_from_stream** (std::istream &infile, Operator &Hbare)
- void ReadBareTBME_Darmstadt (std::string filename, Operator &Hbare, int E1max, int E2max, int lmax)

  Decide if the file is gzipped or ascii, create a stream, then call ReadBareTBME_Darmstadt_from_stream().
- template<class T >
  void ReadBareTBME_Darmstadt_from_stream (T &infile, Operator &Hbare, int E1max, int E2max, int lmax)
- void Read_Darmstadt_3body (std::string filename, Operator &Hbare, int E1max, int E2max, int E3max)
- size_t Count_Darmstadt_3body_to_read (Operator &Hbare, int E1max, int E2max, int E3max, std::vector< int > &orbits_remap, std::vector< size_t > &nread_list)
- template<class T >
  void **Read_Darmstadt_3body_from_stream** (T &infile, Operator &Hbare, int E1max, int E2max, int E3max)
- void Store_Darmstadt_3body (const std::vector< float > &ThreeBME, const std::vector< size_t > &nread_list, const std::vector< int > &orbits_remap, Operator &Hbare, int E1max, int E2max, int E3max)
- void GetHDF5Basis (ModelSpace *modelspace, std::string filename, std::vector< std::array< int, 5 >> &Basis)
- void Read3bodyHDF5 (std::string filename, Operator &op)
- void **Read3bodyHDF5_new** (std::string filename, Operator &op)
- void **ReadOperator_Nathan** (std::string filename1b, std::string filename2b, Operator &op)
- void **ReadTensorOperator_Nathan** (std::string filename1b, std::string filename2b, Operator &op)
- void **Read2bCurrent_Navratil** (std::string filename, Operator &Op)
- void **Write_me2j** (std::string filename, Operator &op, int emax, int e2max, int lmax)
- void **Write_me3j** (std::string filename, Operator &op, int E1max, int E2max, int E3max)
- void **WriteTBME_Navratil** (std::string filename, Operator &Hbare)
- void WriteNuShellX_sps (Operator &op, std::string filename)

  *Write the valence model space to a NuShellX *.sp file.*
- void **WriteNuShellX_int** (Operator &op, std::string filename)
- void **WriteNuShellX_op** (Operator &op, std::string filename)
- void **ReadNuShellX_int** (Operator &op, std::string filename)
- void **ReadNuShellX_int_iso** (Operator &op, std::string filename)
- void **ReadNuShellX_sp** (ModelSpace &ms, std::string filename)
- void WriteNuShellX_intfile (Operator &op, std::string filename, std::string mode)
- void WriteAntoine_int (Operator &op, std::string filename)

  This now appears to be working properly.
- void WriteAntoine_input (Operator &op, std::string filename, int A, int Z)

  Generate an input file for antoine, which may be edited afterwards if need be.
- void WriteOperator (Operator &op, std::string filename)

  Write an operator to a plain-text file.
- void WriteOperatorHuman (Operator &op, std::string filename)

  Write an operator to a plain-text file.
- void ReadOperator (Operator &op, std::string filename)

  Read an operator from a plain-text file.
- void ReadOperatorHuman (Operator &op, std::string filename)

  Read an operator from a plain-text file.
- void CompareOperators (Operator &op1, Operator &op2, std::string filename)

  Write an operator to a plain-text file.
- void **ReadOneBody_Takayuki** (std::string filename, Operator &Hbare)
- void **ReadTwoBody_Takayuki** (std::string filename, Operator &Hbare)
- void **WriteOneBody_Takayuki** (std::string filename, Operator &Hbare)
- void **WriteTwoBody_Takayuki** (std::string filename, Operator &Hbare)
- void **WriteTensorOneBody** (std::string filename, Operator &H, std::string opname)
- void **WriteTensorTwoBody** (std::string filename, Operator &H, std::string opname)
- void **WriteOneBody_Simple** (std::string filename, Operator &Hbare)
- void WriteOneBody_Oslo (std::string filename, Operator &Hbare)

- void WriteTwoBody_Oslo (std::string filename, Operator &Hbare)

  Read two-body matrix elements from an Oslo-formatted file.
- void **ReadTwoBodyEngel** (std::string filename, Operator &Op)
- void **ReadTwoBodyEngel_from_stream** (std::istream &infile, Operator &Op)
- void **ReadRelCMOpFromJavier** (std::string statefile, std::string MEfile, Operator &Op)
- void **SetLECs** (double c1, double c3, double c4, double cD, double cE)
- std::array< double, 5 > **GetLECs** ()
- void **SetLECs_preset** (std::string)
- void **SetCoMCorr** (bool b)
- void **SetScratchDir** (std::string d)
- std::string **GetScratchDir** ()
- int **GetAref** ()
- int **GetZref** ()
- void **SetAref** (int a)
- void **SetZref** (int z)
- void **Set3NFormat** (std::string fmt)
- bool **InGoodState** ()

Public Attributes

- std::map< std::string, std::string > **InputParameters**
- bool **doCoM_corr**
- bool **goodstate**
- std::array< double, 5 > **LECs**
- std::string **scratch_dir**
- std::string **File2N**
- std::string **File3N**
- std::string **format3N**
- int **Aref**
- int **Zref**

7.14.1 Member Function Documentation

7.14.1.1 Count_Darmstadt_3body_to_read()

```
size_t ReadWrite::Count_Darmstadt_3body_to_read (
            Operator & Hbare,
            int E1max,
            int E2max,
            int E3max,
            std::vector< int > & orbits_remap,
            std::vector< size_t > & nread_list )
```

Read me3j format three-body matrix elements. Pass in E1max, E2max, E3max for the file, so that it can be properly interpreted. The modelspace truncation doesn't need to coincide with the file truncation. For example, you could have an emax=10 modelspace and read from an emax=14 file, and the matrix elements with emax>10 would be ignored.

7.14.1.2 GetHDF5Basis()

```
void ReadWrite::GetHDF5Basis (
            ModelSpace * modelspace,
            std::string filename,
            std::vector< std::array< int, 5 >> & Basis )
```

Read three-body basis from HDF5 formatted file. This routine was ported to C++ from a C routine by Heiko Hergert, with as little modification as possible.

7.14.1.3 Read3bodyHDF5()

```
void ReadWrite::Read3bodyHDF5 (
            std::string filename,
            Operator & op )
```

Read three-body matrix elements from HDF5 formatted file. This routine was ported to C++ from a C routine by Heiko Hergert, with as little modification as possible.

7.14.1.4 Read_Darmstadt_3body()

```
void ReadWrite::Read_Darmstadt_3body (
            std::string filename,
            Operator & Hbare,
            int E1max,
            int E2max,
            int E3max )
```

Decide the file format from the extension – .me3j (Darmstadt group format, human-readable), .gz (gzipped me3j, less storage), .bin (me3j converted to binary, faster to read), .h5 (HDF5 format). Default is to assume .me3j. For the first three, the file is converted to a stream and sent to ReadDarmstadt_3body_from_stream(). For the HDF5 format, a separate function is called: Read3bodyHDF5().

7.14.1.5 ReadBareTBME_Darmstadt_from_stream()

```
template<class T >
void ReadWrite::ReadBareTBME_Darmstadt_from_stream (
            T & infile,
            Operator & Hbare,
            int emax,
            int Emax,
            int lmax )
```

Read TBME's from a file formatted by the Darmstadt group. The file contains just the matrix elements, and the corresponding quantum numbers are inferred. This means that the model space of the file must also be specified. emax refers to the maximum single-particle oscillator shell. Emax refers to the maximum of the sum of two single particles. lmax refers to the maximum single-particle ℓ. If Emax is not specified, it should be 2 $\times$ emax. If lmax is not specified, it should be emax. Also note that the matrix elements are given in un-normalized form, i.e. they are just the M scheme matrix elements multiplied by Clebsch-Gordan coefficients for JT coupling.

7.14.1.6 ReadBareTBME_Navratil()

```
void ReadWrite::ReadBareTBME_Navratil (
            std::string filename,
            Operator & Hbare )
```

Read two body matrix elements from file formatted for Petr Navratil's no-core shell model code. These are given as normalized, JT coupled matix elements. Matrix elements corresponding to orbits outside the modelspace are ignored.

7.14.1.7 Store_Darmstadt_3body()

```
void ReadWrite::Store_Darmstadt_3body (
            const std::vector< float > & ThreeBME,
            const std::vector< size_t > & nread_list,
            const std::vector< int > & orbits_remap,
            Operator & Hbare,
            int E1max,
            int E2max,
            int E3max )
```

Read me3j format three-body matrix elements. Pass in E1max, E2max, E3max for the file, so that it can be properly interpreted. The modelspace truncation doesn't need to coincide with the file truncation. For example, you could have an emax=10 modelspace and read from an emax=14 file, and the matrix elements with emax>10 would be ignored.

7.14.1.8 WriteNuShellX_intfile()

```
void ReadWrite::WriteNuShellX_intfile (
            Operator & op,
            std::string filename,
            std::string mode )
```

Write the valence space part of the interaction to a NuShellX $*$.int file. Note that for operators other than the Hamiltonian NuShellX assumes identical orbits for protons and neutrons, so that the pnpn interaction should be equal to the pnnp interaction. This is only approximately true for interactions generated with IMSRG, so some averaging is required.

7.14.1.9 WriteOneBody_Oslo()

```
void ReadWrite::WriteOneBody_Oslo (
            std::string filename,
            Operator & Hbare )
```

Switching order here to make EKK work with the MBPT code

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

- ReadWrite.hh
- ReadWrite.cc

7.15 ThreeBodyME Class Reference

`#include <ThreeBodyME.hh>`

Public Member Functions

- **ThreeBodyME** (ModelSpace *)
- **ThreeBodyME** (ModelSpace *ms, int e3max)
- size_t KeyHash (size_t, size_t, size_t, size_t, size_t, size_t) const
- void **Allocate** ()
- void **SetModelSpace** (ModelSpace *ms)
- std::vector< std::pair< size_t, double > > AccessME (int Jab_in, int Jde_in, int J2, int tab_in, int tde_in, int T2, int i, int j, int k, int l, int m, int n) const
- ThreeBME_type **AddToME** (int Jab_in, int Jde_in, int J2, int tab_in, int tde_in, int T2, int i, int j, int k, int l, int m, int n, ThreeBME_type V)
- void SetME (int Jab_in, int Jde_in, int J2, int tab_in, int tde_in, int T2, int i, int j, int k, int l, int m, int n, ThreeBME_type V)
- ThreeBME_type GetME (int Jab_in, int Jde_in, int J2, int tab_in, int tde_in, int T2, int i, int j, int k, int l, int m, int n) const
- ThreeBME_type GetME_pn (int Jab_in, int Jde_in, int J2, int i, int j, int k, int l, int m, int n) const
- int SortOrbits (int a_in, int b_in, int c_in, int &a, int &b, int &c) const
- double RecouplingCoefficient (int recoupling_case, double ja, double jb, double jc, int Jab_in, int Jab, int J) const

  Coefficients for recoupling three body matrix elements.
- void **SetE3max** (int e)
- int **GetE3max** ()
- void **Erase** ()
- void Deallocate ()

  Free up the memory used for the matrix elements.
- size_t **size** ()
- void **WriteBinary** (std::ofstream &)
- void **ReadBinary** (std::ifstream &)

Public Attributes

- ModelSpace * **modelspace**
- std::vector< ThreeBME_type > **MatEl**
- std::unordered_map< size_t, size_t > **OrbitIndexHash**
- int **E3max**
- size_t **total_dimension**

Static Public Attributes

- static const int **ABC** = 0
- static const int **BCA** = 1
- static const int **CAB** = 2
- static const int **ACB** = 3
- static const int **BAC** = 4
- static const int **CBA** = 5

7.15.1 Detailed Description

The three-body piece of an operator, stored in nested vectors. The 3BMEs are stored in unnormalized JT coupled form $\langle (abJ_{ab}t_{ab})c|V|(deJ_{de}t_{de})f\rangle_{JT}$. To minimize the number of stored matrix elements, only elements with $a \geq b \geq c, a \geq d \geq e \geq f$ are stored. The other combinations are obtained on the fly by GetME(). The storage format is MatEl[{a,b,c,d,e,f,J,Jab,Jde}][T_index] = $\langle (abJ_{ab}t_{ab})c|V|(deJ_{de}t_{de})f\rangle_{JT}$.

7.15.2 Member Function Documentation

7.15.2.1 AccessME()

```
std::vector< std::pair< size_t, double > > ThreeBodyME::AccessME (
              int Jab_in,
              int Jde_in,
              int J2,
              int tab_in,
              int tde_in,
              int T2,
              int a_in,
              int b_in,
              int c_in,
              int d_in,
              int e_in,
              int f_in ) const
```

Since the code for setting or getting matrix elements is almost identical, do all the work here to pull out a list of indices and coefficients which are needed for setting or getting.

7.15.2.2 GetME()

```
ThreeBME_type ThreeBodyME::GetME (
              int Jab_in,
              int Jde_in,
              int J2,
              int tab_in,
              int tde_in,
              int T2,
              int a_in,
              int b_in,
              int c_in,
              int d_in,
              int e_in,
              int f_in ) const
```

Get three body matrix element in isospin formalism $V_{abcdef}^{J_{ab}J_{de}Jt_{ab}t_{de}T}$ (which is how they're stored). The elements are stored with the following restrictions: $a \geq b \geq c, d \geq e \geq f, a \geq d$. If $a = d$ then $b \geq e$, and if $b = e$ then $c \geq f$. Other orderings are obtained by recoupling on the fly.

7.15.2.3 GetME_pn()

```
ThreeBME_type ThreeBodyME::GetME_pn (
            int Jab_in,
            int Jde_in,
            int J2,
            int a,
            int b,
            int c,
            int d,
            int e,
            int f ) const
```

Get three body matrix element in proton-neutron formalism.

$$V^{(pn)}_{abcdef} = \sum_{t_{ab} t_{de} T} < t_a t_b | t_{ab} >< t_d t_e | t_{de} >< t_{ab} t_c | T >< t_{de} t_f | T > V^{t_{ab} t_{de} T}_{abcdef}$$

7.15.2.4 KeyHash()

```
size_t ThreeBodyME::KeyHash (
            size_t a,
            size_t b,
            size_t c,
            size_t d,
            size_t e,
            size_t f ) const
```

Hash function to map six indices to a single long unsigned int. Each index gets 10 bits, for a maximum of 1024 indices. If we have good isospin, this means we're ok up to emax=43.

7.15.2.5 SetME()

```
void ThreeBodyME::SetME (
            int Jab_in,
            int Jde_in,
            int J2,
            int tab_in,
            int tde_in,
            int T2,
            int a_in,
            int b_in,
            int c_in,
            int d_in,
            int e_in,
            int f_in,
            ThreeBME_type V )
```

Set a three body matrix element. Since only a subset of orbit orderings are stored, we need to recouple if the input ordering is different.

7.15.2.6 SortOrbits()

```
int ThreeBodyME::SortOrbits (
             int a_in,
             int b_in,
             int c_in,
             int & a,
             int & b,
             int & c ) const
```

Rearrange orbits (abc) so that a>=b>=c and return an int which reflects the required reshuffling

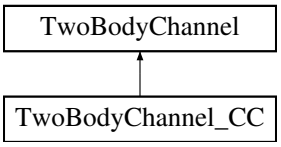
- 0: (abc)_in -> (abc)
- 1: (bca)_in -> (abc)
- 2: (cab)_in -> (abc)
- 3: (acb)_in -> (abc) – odd permutation
- 4: (bac)_in -> (abc) – odd permutation
- 5: (cba)_in -> (abc) – odd permutation

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

- ThreeBodyME.hh
- ThreeBodyME.cc

7.16 TwoBodyChannel Class Reference

Inheritance diagram for TwoBodyChannel:



Public Member Functions

- **TwoBodyChannel** (int j, int p, int t, ModelSpace *ms)
- **TwoBodyChannel** (int N, ModelSpace *ms)
- void **Initialize** (int N, ModelSpace *ms)
- int **GetNumberKets** () const
- int **GetLocalIndex** (int ketindex) const
- int **GetLocalIndex** (int p, int q) const
- int **GetKetIndex** (int i) const
- const Ket & **GetKet** (int i) const
- Ket & **GetKet** (int i)
- arma::uvec **GetKetIndexFromList** (std::vector< index_t > &vec_in)
- const arma::uvec & **GetKetIndex_pp** () const

- const arma::uvec & **GetKetIndex_hh** () const
- const arma::uvec & **GetKetIndex_ph** () const
- const arma::uvec & **GetKetIndex_cc** () const
- const arma::uvec & **GetKetIndex_vc** () const
- const arma::uvec & **GetKetIndex_qc** () const
- const arma::uvec & **GetKetIndex_vv** () const
- const arma::uvec & **GetKetIndex_qv** () const
- const arma::uvec & **GetKetIndex_qq** () const
- virtual bool **CheckChannel_ket** (Orbit *op, Orbit *oq) const
- bool **CheckChannel_ket** (Ket &ket) const

Public Attributes

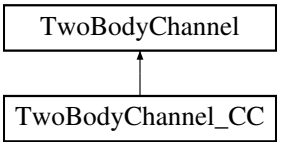
- int **J**
- int **parity**
- int **Tz**
- arma::uvec **KetIndex_pp**
- arma::uvec **KetIndex_hh**
- arma::uvec **KetIndex_ph**
- arma::uvec **KetIndex_cc**
- arma::uvec **KetIndex_vc**
- arma::uvec **KetIndex_qc**
- arma::uvec **KetIndex_vv**
- arma::uvec **KetIndex_qv**
- arma::uvec **KetIndex_qq**
- arma::vec **Ket_occ_hh**
- arma::vec **Ket_unocc_hh**
- arma::vec **Ket_occ_ph**
- arma::vec **Ket_unocc_ph**
- ModelSpace * **modelspace**
- int **NumberKets**
- std::vector< int > **KetList**
- std::vector< int > **KetMap**

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

- ModelSpace.hh
- ModelSpace.cc

7.17 TwoBodyChannel_CC Class Reference

Inheritance diagram for TwoBodyChannel_CC:



Public Member Functions

- **TwoBodyChannel_CC** (int j, int p, int t, ModelSpace *ms)
- **TwoBodyChannel_CC** (int N, ModelSpace *ms)
- bool **CheckChannel_ket** (Orbit *op, Orbit *oq) const

Additional Inherited Members

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

- ModelSpace.hh
- ModelSpace.cc

7.18 TwoBodyME Class Reference

`#include <TwoBodyME.hh>`

Public Member Functions

- **TwoBodyME** (ModelSpace *)
- **TwoBodyME** (TwoBodyME_ph &)
- **TwoBodyME** (ModelSpace *ms, int rankJ, int rankT, int parity)
- TwoBodyME & **operator*=** (const double)
- TwoBodyME & **operator+=** (const TwoBodyME &)
- TwoBodyME & **operator-=** (const TwoBodyME &)
- void **Allocate** ()
- bool **IsHermitian** ()
- bool **IsAntiHermitian** ()
- bool **IsNonHermitian** ()
- void **SetHermitian** ()
- void **SetAntiHermitian** ()
- void **SetNonHermitian** ()
- arma::mat & **GetMatrix** (int chbra, int chket)
- arma::mat & **GetMatrix** (int ch)
- arma::mat & **GetMatrix** (std::array< int, 2 > a)
- const arma::mat & **GetMatrix** (int chbra, int chket) const
- const arma::mat & **GetMatrix** (int ch) const
- double GetTBME (int ch_bra, int ch_ket, int a, int b, int c, int d) const

  This returns the matrix element times a factor $\sqrt{(1+\delta_{ij})(1+\delta_{kl})}$.
- double GetTBME_norm (int ch_bra, int ch_ket, int a, int b, int c, int d) const

  This returns the normalized matrix element.
- void **SetTBME** (int ch_bra, int ch_ket, int a, int b, int c, int d, double tbme)
- void **AddToTBME** (int ch_bra, int ch_ket, int a, int b, int c, int d, double tbme)
- double **GetTBME** (int ch_bra, int ch_ket, Ket &bra, Ket &ket) const
- void **SetTBME** (int ch_bra, int ch_ket, Ket &bra, Ket &ket, double tbme)
- void **AddToTBME** (int ch_bra, int ch_ket, Ket &bra, Ket &ket, double tbme)
- double **GetTBME_norm** (int ch_bra, int ch_ket, int ibra, int iket) const
- void **SetTBME** (int ch_bra, int ch_ket, int ibra, int iket, double tbme)
- void **AddToTBME** (int ch_bra, int ch_ket, int ibra, int iket, double tbme)

- double **GetTBME** (int j_bra, int p_bra, int t_bra, int j_ket, int p_ket, int t_ket, Ket &bra, Ket &ket) const
- void **SetTBME** (int j_bra, int p_bra, int t_bra, int j_ket, int p_ket, int t_ket, Ket &bra, Ket &ket, double tbme)
- void **AddToTBME** (int j_bra, int p_bra, int t_bra, int j_ket, int p_ket, int t_ket, Ket &bra, Ket &ket, double tbme)
- double **GetTBME** (int j_bra, int p_bra, int t_bra, int j_ket, int p_ket, int t_ket, int a, int b, int c, int d) const
- void **SetTBME** (int j_bra, int p_bra, int t_bra, int j_ket, int p_ket, int t_ket, int a, int b, int c, int d, double tbme)
- void **AddToTBME** (int j_bra, int p_bra, int t_bra, int j_ket, int p_ket, int t_ket, int a, int b, int c, int d, double tbme)
- double **GetTBME_J** (int j_bra, int j_ket, int a, int b, int c, int d) const
- void **SetTBME_J** (int j_bra, int j_ket, int a, int b, int c, int d, double tbme)
- void **AddToTBME_J** (int j_bra, int j_ket, int a, int b, int c, int d, double tbme)
- double **GetTBME_J_norm** (int j_bra, int j_ket, int a, int b, int c, int d) const
- double **GetTBME** (int ch, int a, int b, int c, int d) const
- double **GetTBME_norm** (int ch, int a, int b, int c, int d) const
- void **SetTBME** (int ch, int a, int b, int c, int d, double tbme)
- void **AddToTBME** (int ch, int a, int b, int c, int d, double tbme)
- double **GetTBME** (int ch, Ket &bra, Ket &ket) const
- double **GetTBME_norm** (int ch, Ket &bra, Ket &ket) const
- void **SetTBME** (int ch, Ket &bra, Ket &ket, double tbme)
- void **AddToTBME** (int ch, Ket &bra, Ket &ket, double tbme)
- double **GetTBME_norm** (int ch, int ibra, int iket) const
- void **SetTBME** (int ch, int ibra, int iket, double tbme)
- void **AddToTBME** (int ch, int ibra, int iket, double tbme)
- double **GetTBME** (int j, int p, int t, Ket &bra, Ket &ket) const
- void **SetTBME** (int j, int p, int t, Ket &bra, Ket &ket, double tbme)
- void **AddToTBME** (int j, int p, int t, Ket &bra, Ket &ket, double tbme)
- double **GetTBME** (int j, int p, int t, int a, int b, int c, int d) const
- double **GetTBME_norm** (int j, int p, int t, int a, int b, int c, int d) const
- void **SetTBME** (int j, int p, int t, int a, int b, int c, int d, double tbme)
- void **AddToTBME** (int j, int p, int t, int a, int b, int c, int d, double tbme)
- double **GetTBME_J** (int j, int a, int b, int c, int d) const
- void **SetTBME_J** (int j, int a, int b, int c, int d, double tbme)
- void **AddToTBME_J** (int j, int a, int b, int c, int d, double tbme)
- double **GetTBME_J_norm** (int j, int a, int b, int c, int d) const
- void Set_pn_TBME_from_iso (int j, int T, int tz, int a, int b, int c, int d, double tbme)
- double **Get_iso_TBME_from_pn** (int j, int T, int tz, int a, int b, int c, int d)
- double GetTBMEmonopole (int a, int b, int c, int d) const
- double **GetTBMEmonopole_norm** (int a, int b, int c, int d) const
- double **GetTBMEmonopole** (Ket &bra, Ket &ket) const
- void Erase ()

  Take a matrix element expressed in relative/CM frame, and add it to the lab frame TBME.
- void **Scale** (double)
- double **Norm** () const
- void **Symmetrize** ()
- void **AntiSymmetrize** ()
- void **Eye** ()
- void **PrintMatrix** (int chbra, int chket) const
- int **Dimension** ()
- int **size** ()
- void **WriteBinary** (std::ofstream &)
- void **ReadBinary** (std::ifstream &)

Public Attributes

- ModelSpace * **modelspace**
- std::map< std::array< int, 2 >, arma::mat > **MatEl**
- int **nChannels**
- bool **hermitian**
- bool **antihermitian**
- int **rank_J**
- int **rank_T**
- int **parity**

7.18.1 Detailed Description

The two-body piece of the operator, stored in a vector of maps of of armadillo matrices. The index of the vector indicates the J-coupled two-body channel of the ket state, while the map key is the two-body channel of the bra state. This is done to allow for tensor operators which connect different two-body channels without having to store all possible combinations. In the case of a scalar operator, there is only one map key for the bra state, corresponding to that of the ket state. The normalized J-coupled TBME's are stored in the matrices. However, when the TBME's are accessed by GetTBME(), they are returned as $\tilde{\Gamma}_{ijkl} \equiv \sqrt{(1+\delta_{ij})(1+\delta_{kl})}\Gamma_{ijkl}$ because the flow equations are derived in terms of $\tilde{\Gamma}$. For efficiency, only matrix elements with $i \leq j$ and $k \leq l$ are stored. When performing sums that can be cast as matrix multiplication, we have something of the form

$$\tilde{Z}_{ijkl} \sim \frac{1}{2}\sum_{ab} \tilde{X}_{ijab}\tilde{Y}_{abkl}$$

which may be rewritten as a restricted sum, or matrix multiplication

$$Z_{ijkl} \sim \sum_{a\leq b} X_{ijab}Y_{abkl} = (X \cdot Y)_{ijkl}$$

7.18.2 Member Function Documentation

7.18.2.1 Erase()

```
void TwoBodyME::Erase ( )
```

Take a matrix element expressed in relative/CM frame, and add it to the lab frame TBME.

For a given ket expressed in relative/CM coordinates, return the indices of all the lab frame kets with non-zero overlap, as well as the overlap

7.18.2.2 GetTBMEmonopole()

```
double TwoBodyME::GetTBMEmonopole (
            int a,
            int b,
            int c,
            int d ) const
```

Returns an unnormalized monopole-like (angle-averaged) term

$$\bar{V}_{ijkl} = \sqrt{(1+\delta_{ij})(1+\delta_{kl})}\frac{\sum_J (2J+1)V^J_{ijkl}}{(2j_i+1)(2j_j+1)}$$

7.18.2.3 Set_pn_TBME_from_iso()

```
void TwoBodyME::Set_pn_TBME_from_iso (
             int j,
             int T,
             int tz,
             int a,
             int b,
             int c,
             int d,
             double tbme )
```

Useful for reading in files in isospin formalism

$$\langle ab|V|cd\rangle_{pnpn} = \frac{\sqrt{(1+\delta_{ab})(1+\delta_{cd})}}{2}\left(\langle ab|V|cd\rangle_{10} + \langle ab|V|cd\rangle_{00}\right)$$

$$\langle ab|V|cd\rangle_{pnnp} = \frac{\sqrt{(1+\delta_{ab})(1+\delta_{cd})}}{2}\left(\langle ab|V|cd\rangle_{10} - \langle ab|V|cd\rangle_{00}\right)$$

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

- TwoBodyME.hh
- TwoBodyME.cc

7.19 boost::numeric::odeint::vector_space_norm_inf< deque< Operator > > Struct Template Reference

Public Types

- typedef double **result_type**

Public Member Functions

- double **operator()** (const deque< Operator > &X)

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

- IMSRGSolver.cc

7.20 VectorStream Class Reference

```
#include <ReadWrite.hh>
```

Public Member Functions

- **VectorStream** (std::vector< float > &v)
- VectorStream & **operator>>** (float &x)
- bool **good** ()
- void **getline** (char[ ], int)
- void **read** (char *buf, size_t len)

7.20.1 Detailed Description

Wrapper class so we can treat a std::vector of floats like a stream, using the extraction operator >>. This is used for the binary version of ReadWrite::Read_Darmstadt_3body_from_stream().

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

- ReadWrite.hh

Index