
EOPDev

Version 1.0.0

ELIMINATION OF ELECTRON REPULSION INTEGRALS
FROM FRAGMENT-BASED METHODS OF QUANTUM CHEMISTRY
A PLUGIN TO PSI4

BY

BARTOSZ BŁASIAK

Wrocław University of Science and Technology

FUNDED BY

NATIONAL SCIENCE CENTRE, KRAKÓW, POLAND

Grant No. 2016/23/P/ST4/01720

H2020 MARIE SKŁODOWSKA-CURIE ACTIONS CO-FUND (POLONEZ)

Grant No. 665778



NOVEMBER 2020
LICENSE: LGPL-2.1

Contents

1	Main Page	1
2	Introduction	3
2.1	Research Project Methodology	4
2.2	Expected Impact on the Development of Science, Civilization and Society	4
2.3	The EOPDev Code	5
3	EOP Design.	7
3.1	Classes of EOPs	8
3.1.1	Structure of possible EOP-based expressions and their unification	8
3.2	Density-fitting Specialized for EOPs	9
3.2.1	Fitting in Complete Space	9
3.2.2	Fitting in Incomplete Space	10
3.2.3	Fitting in Incomplete Space - Alternative Approach	11
4	Implemented Models	13
4.1	Fragment-Based Methods	13
4.2	Target, Benchmark and Competing Models	13
5	Contributing to EOPDev	15
5.1	Main Routine and Libraries	15
5.2	Header Files in Libraries	16
5.3	Environmental Variables	16
5.4	Documenting the Code	17

5.5	Naming Conventions	17
5.6	Track Timing When Evaluating the Code	18
5.7	Clean Memory Between Independent Jobs	18
5.8	Use Object-Oriented Programming	18
5.9	Implement Tests	19
6	Usage	21
6.1	Installation	21
6.1.1	Preparing Psi4	21
6.1.2	Compiltation	22
6.1.3	Step-By-Step Installation	22
6.2	EOPDev Code Structure	23
6.2.1	Main Routine	23
6.2.2	Modules	24
6.3	EOPDev Classes: Overview	25
6.3.1	EOP Module	25
6.3.2	GEFP Module	25
6.3.3	EOPDev Solver Module	26
6.4	Developing EOPs	26
6.4.1	Drafting an EOP Subclass	27
6.5	Examples	29
7	Module Index	31
7.1	Modules	31
8	Namespace Index	33
8.1	Namespace List	33
9	Hierarchical Index	35
9.1	Class Hierarchy	35
10	Class Index	39
10.1	Class List	39
11	File Index	45
11.1	File List	45
12	Module Documentation	47

12.1 The Generalized One-Electron Potentials Library	47
12.1.1 Detailed Description	48
12.2 The EOPDev Solver Library	49
12.2.1 Detailed Description	49
12.3 The Generalized Effective Fragment Potentials Library	50
12.3.1 Detailed Description	51
12.4 The Integral Package Library	52
12.4.1 Detailed Description	53
12.4.2 Hermite Operators	54
12.4.3 One-Body Integrals over Hermite Functions	55
12.4.4 Two-Body Integrals over Hermite Functions	56
12.4.5 The R(N,L,M) Coefficients	56
12.4.6 Function Documentation	57
12.5 The Three-Dimensional Vector Fields Library	62
12.5.1 Detailed Description	63
12.5.2 Function Documentation	63
12.6 The Density Functional Theory Library	65
12.7 The EOPDev Utilities	66
12.7.1 Theory	70
12.7.2 Detailed Description	71
12.7.3 Function Documentation	71
12.8 The EOPDev Testing Platform Library	81
12.8.1 Detailed Description	81
13 Namespace Documentation	83
13.1 oepdev Namespace Reference	83
13.1.1 Theory	91
13.1.2 Detailed Description	93
13.2 psi Namespace Reference	93
13.2.1 Detailed Description	93
13.2.2 Function Documentation	93
14 Class Documentation	95
14.1 oepdev::ABCD Struct Reference	95
14.1.1 Detailed Description	95
14.2 oepdev::AbInitioPolarGEFactory Class Reference	95

14.2.1 Detailed Description	96
14.3 oepdev::AllAOIntegralsIterator_2 Class Reference	97
14.3.1 Detailed Description	97
14.3.2 Constructor & Destructor Documentation	97
14.3.3 Member Function Documentation	98
14.4 oepdev::AllAOIntegralsIterator_4 Class Reference	98
14.4.1 Detailed Description	99
14.4.2 Constructor & Destructor Documentation	99
14.4.3 Member Function Documentation	100
14.5 oepdev::AllAOShellCombinationsIterator_2 Class Reference	100
14.5.1 Detailed Description	101
14.5.2 Constructor & Destructor Documentation	101
14.5.3 Member Function Documentation	103
14.6 oepdev::AllAOShellCombinationsIterator_4 Class Reference	103
14.6.1 Detailed Description	104
14.6.2 Constructor & Destructor Documentation	104
14.6.3 Member Function Documentation	105
14.7 oepdev::AOIntegralsIterator Class Reference	106
14.7.1 Detailed Description	107
14.7.2 Member Function Documentation	107
14.8 oepdev::CAMM Class Reference	108
14.8.1 Detailed Description	109
14.9 oepdev::ChargeTransferEnergyOEPotential Class Reference	110
14.9.1 Detailed Description	110
14.10 oepdev::ChargeTransferEnergySolver Class Reference	111
14.10.1 Detailed Description	111
14.10.2 Member Function Documentation	114
14.11 oepdev::CISComputer Class Reference	114
14.11.1 Detailed Description	118
14.11.2 Member Function Documentation	118
14.11.3 Member Data Documentation	121
14.12 oepdev::CISData Struct Reference	121
14.12.1 Detailed Description	122
14.13 oepdev::CPHF Class Reference	122
14.13.1 Detailed Description	125

14.13.2 Constructor & Destructor Documentation	126
14.14oepdev::CubePoints3DIterator Class Reference	126
14.14.1 Detailed Description	127
14.15oepdev::CubePointsCollection3D Class Reference	127
14.15.1 Detailed Description	128
14.16oepdev::DavidsonLiu Class Reference	128
14.16.1 Detailed Description	130
14.17oepdev::DIISManager Class Reference	132
14.17.1 Detailed Description	132
14.17.2 Constructor & Destructor Documentation	133
14.17.3 Member Function Documentation	133
14.18oepdev::DMTPole Class Reference	134
14.18.1 Detailed Description	139
14.18.2 Constructor & Destructor Documentation	139
14.18.3 Member Function Documentation	140
14.18.4 Friends And Related Function Documentation	145
14.19oepdev::DoubleGeneralizedDensityFit Class Reference	145
14.19.1 Detailed Description	146
14.19.2 Determination of the OEP matrix	146
14.19.3 Member Function Documentation	147
14.20oepdev::EETCouplingOEPotential Class Reference	148
14.20.1 Detailed Description	149
14.21oepdev::EETCouplingSolver Class Reference	149
14.21.1 Detailed Description	150
14.21.2 Member Function Documentation	155
14.22oepdev::EFP2.GEFactory Class Reference	156
14.22.1 Detailed Description	157
14.23oepdev::EFPMultipolePotentialInt Class Reference	158
14.24oepdev::ElectrostaticEnergyOEPotential Class Reference	159
14.24.1 Detailed Description	159
14.25oepdev::ElectrostaticEnergySolver Class Reference	160
14.25.1 Detailed Description	160
14.25.2 Member Function Documentation	162
14.26oepdev::ElectrostaticPotential3D Class Reference	163
14.26.1 Detailed Description	164

14.27	oepdev::ERI_1_1 Class Reference	164
14.27.1	Detailed Description	165
14.27.2	Implementation	165
14.28	oepdev::ERI_2_2 Class Reference	166
14.28.1	Detailed Description	167
14.28.2	Implementation	167
14.29	oepdev::ERI_3_1 Class Reference	167
14.29.1	Detailed Description	168
14.29.2	Implementation	169
14.30	oepdev::ESPSolver Class Reference	169
14.30.1	Detailed Description	170
14.30.2	Constructor & Destructor Documentation	171
14.31	oepdev::FFABInitioPolarGEFactory Class Reference	172
14.31.1	Detailed Description	172
14.32	oepdev::Field3D Class Reference	173
14.32.1	Detailed Description	175
14.32.2	Constructor & Destructor Documentation	175
14.32.3	Member Function Documentation	176
14.33	oepdev::Fourier5 Struct Reference	177
14.33.1	Detailed Description	177
14.34	oepdev::Fourier9 Struct Reference	177
14.34.1	Detailed Description	178
14.35	oepdev::FragmentedSystem Class Reference	178
14.35.1	Detailed Description	179
14.35.2	Member Function Documentation	179
14.36	oepdev::GenEffFrag Class Reference	182
14.36.1	Detailed Description	185
14.36.2	Member Function Documentation	186
14.37	oepdev::GenEffPar Class Reference	188
14.37.1	Detailed Description	193
14.37.2	Member Function Documentation	193
14.38	oepdev::GenEffParFactory Class Reference	201
14.38.1	Detailed Description	203
14.38.2	Member Function Documentation	204
14.39	oepdev::GeneralizedDensityFit Class Reference	206

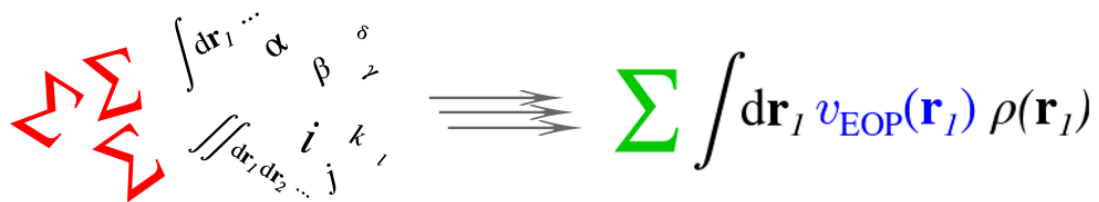
14.39.1 Detailed Description	208
14.39.2 Member Function Documentation	208
14.40oepdev::GeneralizedPolarGEFactory Class Reference	209
14.40.1 Detailed Description	213
14.41oepdev::GramSchmidt Class Reference	215
14.41.1 Detailed Description	216
14.41.2 Constructor & Destructor Documentation	217
14.41.3 Member Function Documentation	217
14.42oepdev::IntegralFactory Class Reference	218
14.42.1 Detailed Description	218
14.43oepdev::KabschSuperimposer Class Reference	219
14.43.1 Detailed Description	220
14.43.2 Member Function Documentation	220
14.44oepdev::LinearGradientNonUniformEFieldPolarGEFactory Class Reference	221
14.44.1 Detailed Description	222
14.45oepdev::LinearNonUniformEFieldPolarGEFactory Class Reference	222
14.45.1 Detailed Description	223
14.46oepdev::LinearUniformEFieldPolarGEFactory Class Reference	223
14.46.1 Detailed Description	224
14.47oepdev::MultipoleConvergence Class Reference	224
14.47.1 Detailed Description	226
14.47.2 Member Enumeration Documentation	226
14.47.3 Constructor & Destructor Documentation	227
14.47.4 Member Function Documentation	227
14.48oepdev::NonUniformEFieldPolarGEFactory Class Reference	228
14.48.1 Detailed Description	229
14.49oepdev::ObaraSaikaTwoCenterEFPRecursion_New Class Reference	229
14.49.1 Constructor & Destructor Documentation	231
14.50oepdev::OEP_EFP2_GEFactory Class Reference	231
14.50.1 Detailed Description	232
14.51oepdev::OEPDevSolver Class Reference	232
14.51.1 Detailed Description	233
14.51.2 Constructor & Destructor Documentation	240
14.51.3 Member Function Documentation	240
14.52oepdev::OEPotential Class Reference	242

14.52.1 Detailed Description	246
14.52.2 Constructor & Destructor Documentation	246
14.52.3 Member Function Documentation	247
14.53oepdev::OEPotential3D< T > Class Template Reference	248
14.53.1 Detailed Description	249
14.54oepdev::OEType Struct Reference	249
14.55oepdev::OverlapGeneralizedDensityFit Class Reference	250
14.55.1 Detailed Description	251
14.55.2 Determination of the OEP matrix	251
14.55.3 Member Function Documentation	251
14.56oepdev::PerturbCharges Struct Reference	251
14.56.1 Detailed Description	252
14.57oepdev::Points3DIterator::Point Struct Reference	252
14.58oepdev::Points3DIterator Class Reference	252
14.58.1 Detailed Description	254
14.58.2 Constructor & Destructor Documentation	254
14.58.3 Member Function Documentation	254
14.59oepdev::PointsCollection3D Class Reference	256
14.59.1 Detailed Description	257
14.59.2 Constructor & Destructor Documentation	257
14.59.3 Member Function Documentation	258
14.60oepdev::PolarGEFactory Class Reference	259
14.60.1 Detailed Description	260
14.61oepdev::PotentialInt Class Reference	260
14.61.1 Constructor & Destructor Documentation	261
14.61.2 Member Function Documentation	262
14.62oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory Class Reference	263
14.62.1 Detailed Description	264
14.63oepdev::QuadraticNonUniformEFieldPolarGEFactory Class Reference	264
14.63.1 Detailed Description	265
14.64oepdev::QuadraticUniformEFieldPolarGEFactory Class Reference	265
14.64.1 Detailed Description	266
14.65oepdev::QUAMBO Class Reference	266
14.65.1 Detailed Description	269
14.66oepdev::QUAMBOData Struct Reference	269

14.66.1 Detailed Description	270
14.67oepdev::R_CISComputer Class Reference	270
14.68oepdev::R_CISComputer_Direct Class Reference	270
14.69oepdev::R_CISComputer_DL Class Reference	271
14.69.1 Detailed Description	272
14.70oepdev::R_CISComputer_Explicit Class Reference	273
14.71oepdev::RandomPoints3DIterator Class Reference	274
14.71.1 Detailed Description	275
14.72oepdev::RandomPointsCollection3D Class Reference	275
14.72.1 Detailed Description	276
14.73oepdev::RepulsionEnergyOEPotential Class Reference	276
14.73.1 Detailed Description	277
14.74oepdev::RepulsionEnergySolver Class Reference	277
14.74.1 Detailed Description	278
14.74.2 Member Function Documentation	282
14.75oepdev::RHFPerturbed Class Reference	283
14.75.1 Detailed Description	285
14.76oepdev::ShellCombinationsIterator Class Reference	285
14.76.1 Detailed Description	287
14.76.2 Constructor & Destructor Documentation	287
14.76.3 Member Function Documentation	288
14.77oepdev::SingleGeneralizedDensityFit Class Reference	290
14.77.1 Detailed Description	290
14.77.2 Determination of the OEP matrix	291
14.77.3 Member Function Documentation	291
14.78oepdev::GeneralizedPolarGEFactory::StatisticalSet Struct Reference	291
14.79oepdev::test::Test Class Reference	292
14.80oepdev::TIData Class Reference	295
14.80.1 Detailed Description	298
14.80.2 Member Function Documentation	299
14.80.3 Member Data Documentation	303
14.81oepdev::TwoBodyAOInt Class Reference	304
14.81.1 Member Function Documentation	305
14.82oepdev::TwoElectronInt Class Reference	306
14.82.1 Detailed Description	307

14.82.2 Member Function Documentation	308
14.83 oepdev::U_CISComputer Class Reference	308
14.84 oepdev::U_CISComputer_DL Class Reference	309
14.84.1 Detailed Description	309
14.85 oepdev::U_CISComputer_Explicit Class Reference	310
14.86 oepdev::UniformEFieldPolarGEFactory Class Reference	311
14.86.1 Detailed Description	312
14.87 oepdev::UnitaryOptimizer Class Reference	312
14.87.1 Detailed Description	315
14.87.2 Constructor & Destructor Documentation	317
14.88 oepdev::UnitaryOptimizer_2 Class Reference	318
14.88.1 Detailed Description	321
14.88.2 Constructor & Destructor Documentation	322
14.89 oepdev::UnitaryOptimizer_2_1 Class Reference	323
14.89.1 Constructor & Destructor Documentation	326
14.90 oepdev::UnitaryOptimizer_4_2 Class Reference	327
14.90.1 Detailed Description	330
14.90.2 Constructor & Destructor Documentation	331
14.91 oepdev::UnitaryTransformedMOPolarGEFactory Class Reference	332
14.91.1 Detailed Description	333
14.92 oepdev::WavefunctionUnion Class Reference	333
14.92.1 Detailed Description	338
14.92.2 Constructor & Destructor Documentation	339
14.92.3 Member Function Documentation	340
15 File Documentation	343
15.1 include/oepdev_files.h File Reference	343
15.2 include/oepdev_options.h File Reference	343
15.3 main.cc File Reference	344
15.4 oepdev/lib3d/dmtp.h File Reference	345
15.5 oepdev/lib3d/esp.h File Reference	346
15.6 oepdev/libgefp/gefp.h File Reference	346
15.7 oepdev/libints/eri.h File Reference	348
15.8 oepdev/libints/recurr.h File Reference	349
15.9 oepdev/liboep/oep.h File Reference	350

15.10	oepdev/liboep/oep_gdf.h File Reference	351
15.11	oepdev/libpsi/integral.h File Reference	351
15.12	oepdev/libpsi/osrecur.h File Reference	352
15.13	oepdev/libpsi/potential.h File Reference	353
15.14	oepdev/libsolver/solver.h File Reference	353
15.15	oepdev/libsolver/ti_data.h File Reference	354
15.16	oepdev/libtest/test.h File Reference	355
15.17	oepdev/libutil/basis_rotation.h File Reference	355
15.17.1	Theory	356
15.18	oepdev/libutil/cis.h File Reference	358
15.19	oepdev/libutil/davidson_liu.h File Reference	359
15.20	oepdev/libutil/diis.h File Reference	359
15.21	oepdev/libutil/gram_schmidt.h File Reference	360
15.22	oepdev/libutil/integrals_iter.h File Reference	360
15.23	oepdev/libutil/kabsch_superimposer.h File Reference	361
15.24	oepdev/libutil/quambo.h File Reference	362
15.25	oepdev/libutil/scf_perturb.h File Reference	362
15.26	oepdev/libutil/unitary_optimizer.h File Reference	363
15.27	oepdev/libutil/util.h File Reference	364
15.28	oepdev/libutil/wavefunction_union.h File Reference	366
16	Example Documentation	369
16.1	example_cphf.cc	369
16.2	example_davidson_liu.cc	369
16.3	example_gefp.cc	370
16.4	example_integrals_iter.cc	371
16.5	example_scf_perturb.cc	372
	Index	376



$$\sum_i \sum_j \int d\mathbf{r}_1 \dots \int d\mathbf{r}_2 \dots \Rightarrow \sum \int d\mathbf{r}_1 v_{\text{EOP}}(\mathbf{r}_1) \rho(\mathbf{r}_1)$$

Effective one-electron operators can be systematically used to convert fragment-based methods into effective fragment potentials.

EOPDev

Generalized Effective One-Electron Potentials: Development Platform.

Author: Bartosz Błasiak

Contributors: Marta Chołuj, Joanna D. Bednarska, Robert W. Góra

Contact: Bartosz Błasiak (blasiak.bartosz@gmail.com)

Overview

Develop and test custom **Effective One-electron Potentials (EOP's)** for fragment-based methods of Quantum Chemistry of extended molecular aggregates.

EOPDev is a Psi4 plugin with extensive Python 3 interface. Currently, a few efficient methods that utilize EOP's and related approaches are implemented and tested against reference solutions:

1. Short-range components of interaction energy at Hartree-Fock level
2. Excitation energy transfer couplings at CIS level
3. Polarization of electronic density in non-uniform electric fields at any level

Places to go:

- [EOPDev Code](#)
- [Current Issues](#)
- [Project Website](#)

This wikipages might be updated in the future.

Funding

This project is funded by National Science Centre, Poland (grant no. 2016/23/P/ST4/01720) within the POLONEZ 3 fellowship. This project is carried out under POLONEZ programme which has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 665778 (H2020-MSCA-COFUND).

References

- [1] B. Błasiak, "One-Particle Density Matrix Polarization Susceptibility Tensors", *J. Chem. Phys.* **149**, 164115 (2018).
- [2] B. Błasiak, J. D. Bednarska, M. Chołuj, R. W. Góra, W. Bartkowiak, "Ab Initio Effective One-Electron Potential Operators: Applications for Charge-Transfer Energy in Effective Fragment Potentials", *J. Comput. Chem.*, Accepted (2020).
- [3] B. Błasiak, W. Bartkowiak, R. W. Góra, "An Effective Potential for Frenkel Excitons", Submitted (2020).

CHAPTER 2

Introduction

Exploring biological phenomena at molecular scale is oftentimes indispensable to develop new drugs and intelligent materials.

Most of relevant system properties are affected by intermolecular interactions with nearby environment such as solvent or closely bound electronic chromophores. Studying such molecular aggregates requires rigorous and accurate quantum chemistry methods, the cost of which grows very fast with the number of electrons. [Xu et al. \[2018a\]](#) [Tomasi et al. \[2005\]](#) Despite many methodologies have been devised to describe energetic and dynamical properties of **extended molecular systems** efficiently and accurately, [Warshel and Levitt \[1976\]](#) [Senn and Thiel \[2009\]](#) [Demerdash et al. \[2014\]](#) [Gordon et al. \[2012\]](#) there exist particularly difficult cases in which modelling is still challenging:

- describing electronic transitions in solution or when coupled with other electronic transition via resonance energy transfer, [Barbatti \[2014\]](#) [Szabla et al. \[2015\]](#) [Bednarska et al. \[2017\]](#) [Jedrzejewska et al. \[2018\]](#)
- performing molecular dynamics at very high level of theory including dynamic electron correlation, [Curchod and Martínez \[2018\]](#)
- vibrational frequency calculations of particular normal mode in condensed phases [Błasiak et al. \[2017\]](#) [Xu et al. \[2018b\]](#) [Lewis et al. \[2016\]](#)

and so on. The reason behind (sometimes prohibitively) high costs of fully *ab initio* calculations in the above areas is the complexity of mathematical models often based on wave functions rather than (conceptually more straightforward) electronic densities and potentials. On the other hand, it has been pointed out before that the one-electron density distributions are of particular importance in chemistry. [Kohn and Sham \[1965\]](#) [Holas and March \[1991\]](#) Thus, it can be utilized as a means of developing a general model that re-expresses the physics of intermolecular interactions in terms of effective one-electron functions that are easier to handle in practice. [Roothaan \[1951\]](#) [Hohenberg and Kohn \[1964\]](#) [Kohn and Sham \[1965\]](#) [Otto and Ladik \[1975\]](#) [Holas and](#)

[March \[1991\]](#) [Weber and Thiel \[2000\]](#) [Neese \[2005\]](#) [Cisneros et al. \[2005\]](#) [Piquemal et al. \[2006\]](#) [Li et al. \[2006\]](#) [Błasiak et al. \[2013\]](#) [Błasiak et al. \[2015\]](#)

This Project focuses on finding a unified way to simplify various fragment-based approaches of Quantum Chemistry of extended molecular systems, i.e., molecular aggregates such as interacting chromophores and molecules solvated by water and other solvents. Indeed, one of the important difficulties encountered in Quantum Chemistry of large systems is the need of evaluation of special kind of numbers known as *electron repulsion integrals*, or in short, ERIs. In a typical calculation, the amount of ERIs can be as high as tens or even hundreds of millions (!) that unfortunately prevents from application of conventional methods when the number of particles in question is too large. In the Project, the complicated expressions involving ERIs shall be greatly simplified to reduce the computational costs as much as possible while introducing no or minor approximations to the original theories.

2.1 Research Project Methodology

In this Project the new theoretical protocol based on the effective one-electron potentials (EOPs) is developed. The main principle is to rewrite arbitrary sum of functions f of electron repulsion integrals (ERIs) by defining EOPs according to the following general prescription:

$$\sum_f f \left[\left(\phi_i^A \phi_j^A || \phi_k^B \phi_l^B \right) \right] = \left(\phi_i^A | v_{kl}^B | \phi_j^A \right) \rightarrow \text{point charge or density fitting}$$

$$\sum_f f \left[\left(\phi_i^A \phi_j^B || \phi_k^B \phi_l^B \right) \right] = \left(\phi_i^A | v_{kl}^B | \phi_j^B \right) \rightarrow \text{density fitting,}$$

where A and B denote different molecules and ϕ_i is the i -th molecular orbital or basis function. Here, v_{kl}^B denotes the poeotypes *ab initio* "EOP matrix element". The above technique might be used in fragment-based *ab initio* methods including molecular dynamics protocols of new generation.

2.2 Expected Impact on the Development of Science, Civilization and Society

The proposed EOPs are expected to significantly develop the fragment-based methods that are widely used in physical chemistry and modelling of biologically important systems. Owing to universality of EOPs, they could find applications in many branches of chemical science: non-empirical* molecular dynamics, short-range resonance energy transfer in photosynthesis, electronic and vibrational solvatochromism, multidimensional spectroscopy and so on. In particular:

- the EOP-based models of Pauli repulsion energy and charge-transfer (CT) energy could be used to improve the computational performance of the second generation effective fragment potential method (EFP2). Particularly, the EFP2 CT term is relatively time consuming and due to this reason it is sometimes omitted in the applications of EFP2 to perform molecular dynamics simulations. [Błasiak et al. \[2020a\]](#)

- the EOP-based model of EET couplings could significantly improve modelling of energy transfer in the light harvesting complexes. At present, short-range phenomena (Dexter mechanisms of EET) are very difficult to efficiently and quantitatively evaluate when performing statistical averaging and applying to large molecular aggregates. Such Dexter effects could be computed by using EOPs in much more efficient manner without losing high accuracy of state-of-the-art methods such as TDFI-TI method. [Błasiak et al. \[2020b\]](#)
- the density matrix polarization (DMS) tensors could be used in new generation fragment-based *ab initio* molecular dynamics protocols that rigorously take into consideration electron correlation effects. [Błasiak \[2018\]](#)

Therefore, we believe that the application of EOPs could have an indirect impact on the design of novel drugs and materials for industry.

2.3 The EOPDev Code

To pursue the above challenges in the field of computational quantum chemistry of extended molecular aggregates, the EOPDev platform is developed. Accurate and efficient *ab initio* [models](#) based on EOPs are implemented in the EOPDev code, along with the state-of-the-art benchmark and competing methods. Written in C++ with an extensive Python interface, EOPDev is a plugin to Psi4 quantum chemistry package. Therefore, compilation and running the EOPDev code is straightforward and follows the API interface similar to the one used in Psi4 with just a few [specific programming conventions](#). The detailed discussion about using the EOPDev code can be found in [usage section](#).

Note

The 'OEP' abbreviation, rather than the 'EOP', is used throughout the code. It is because the earlier versions of EOPDev utilized the former abbreviation consecutively for the shared libraries, modules and class names. The abbreviation was changed to the latter in this public release of the code. Please treat these two abbreviations as synonyms within the project and code, both referring to the *effective one-electron potentials*.

CHAPTER 3

EOP Design.

EOP (One-Electron Potential) is associated with certain quantum one-electron operator \hat{v}^A that defines the ability of molecule A to interact in a particular way with other molecules.

It can be shown that for a two-fragment system composed of fragments A and B

$$\sum_t \mathcal{F}_t[(BX|AA)] + \sum_s (B|\hat{o}_s^A|X) = \sum_{ij \in X} (B|\hat{v}^A|i) [\mathbf{S}^{-1}]_{ij} (j|X)$$

where $S_{ij} = (i|j)$, \mathcal{F}_t is a certain linear functional of ERIs of type $(BX|AA)$, \hat{o}_s^A is a one-electron operator associated with molecule A , and $X = A$ or B . [Błasiak et al. \[2020a\]](#) Such elimination of ERIs is possible when either Coulomb-like or overlap-like interfragment ERIs are of importance. It is also possible to approximate the exchange-like ERIs and incorporate them into EOPs. [Błasiak et al. \[2020b\]](#) The above design offers substantial gain of efficiency, since complicated contractions over ERIs are effectively removed and replaced by summations over one-electron integrals (OEIs).

Technically, EOP can be understood as a **container object** (associated with the molecule in question) that stores the information about the above mentioned quantum operator. Here, it is assumed that similar EOP object is also defined for all other molecules in a molecular aggregate.

In case of interaction between molecules A and B , EOP object of molecule A interacts directly with wavefunction object of the molecule B . By defining a (i) Solver class that handles such interaction, (ii) Wavefunction class, and (iii) EOP class, the universal design of EOP-based approaches can be established and developed.

Important: EOP and Wavefunction classes should not be restricted to Hartree-Fock (HF); in general any correlated wavefunction and derived EOP's should be allowed to work with each other. However, in the current version of the project, only HF wavefunctions are considered.

3.1 Classes of EOPs

There are many types of EOPs, but the underlying principle is the same and independent of the type of intermolecular interaction. Therefore, the EOPs should be implemented by using a multi-level class design. In turn, this design depends on the way EOPs enter the mathematical expressions, i.e., on the types of matrix elements of the one-electron effective operator \hat{v}^A .

3.1.1 Structure of possible EOP-based expressions and their unification

Structure of EOP-based mathematical expressions is listed below:

Type	Matrix Element	Comment
Type 1	$(I \hat{v}^A J)$	$I \in A, J \in B$
Type 2	$(J \hat{v}^A L)$	$J, L \in B$

In the above table, I, J and K indices correspond to basis functions or molecular orbitals. Basis functions can be primary or auxiliary EOP-specialized density-fitting. Depending on the type of function and matrix element, there are many subtypes of resulting matrix elements that differ in their dimensionality. Examples are given below:

Matrix Element	DF-based form	DMTP-based form
$(\mu \hat{v}^{A[\mu]} \sigma)$	$\sum_{l \in A} v_{\mu l}^A S_{l\sigma}$	$\sum_{\alpha \in A} q_{\alpha}^{A[\mu]} V_{\mu\sigma}^{(\alpha)}$
$(i \hat{v}^{A[i]} j)$	$\sum_{l \in A} v_{il}^A S_{lj}$	$\sum_{\alpha \in A} q_{\alpha}^{A[i]} V_{ij}^{(\alpha)}$
$(j \hat{v}^{A[j]} l)$	$\sum_{l \in A} S_{jl} v_{lk}^A S_{kl}$	$\sum_{\alpha \in A} q_{\alpha}^{A[j]} V_{jl}^{(\alpha)}$

In the formulae above, the EOP-part (stored by EOP instances) and the Solver-part (to be computed by the Solver) are separated. For illustrative purpose, distributed charge approximation is assumed for the DMTP form in this table. Note however, that higher multipoles can be also used for better accuracy. It is apparent that all EOP-parts have the form of 2- or 3-index arrays with different class of axes (molecular orbitals, primary/auxiliary basis, atomic space). Therefore, they can be uniquely defined by a unified *tensor object* (storing double precision numbers) and unified *dimension object* storing the information of the axes classes.

In Psi4, a perfect candidate for the above is `psi4::Tensor` class declared in `psi4/libthce/thce.h`. Except from the numeric content its instances also store the information of the dimensions in a form of a vector of `psi4::Dimension` instances.

Another possibility is to use `psi::Matrix` objects, instead of `psi4::Tensor` objects, possibly putting them into a `std::vector` container in case there is more than two axes.

Note

Currently, the second possibility is used, i.e., matrices. For more complex data structures, other types of custom objects are defined.

3.2 Density-fitting Specialized for EOPs

To get the ab-initio representation of a EOP, one can use a procedure similar to the typical density fitting or resolution of identity, both of which are nowadays widely used to compute electron-repulsion integrals (ERIs) more efficiently. More detailed derivation and discussion of the results from this section can be found in the work of [Blasiak et al. \[2020a\]](#).

3.2.1 Fitting in Complete Space

An arbitrary one-electron potential of molecule A acting on any state vector associated with molecule A can be expanded in an *auxiliary space* centered on A as

$$v|i) = \sum_{\xi\eta} v|\xi) [S^{-1}]_{\xi\eta} (\eta|i)$$

under the necessary assumption that the auxiliary basis set is *complete*. In a special case when the basis set is orthogonal (e.g., molecular orbitals) the above relation simplifies to

$$v|i) = \sum_{\xi} v|\xi) (\xi|i)$$

It can be easily shown that the above general and exact expansion can be obtained by performing a density fitting in the complete space. We expand the LHS of the first equation on this page in a series of the auxiliary basis functions scaled by the undetermined expansion coefficients:

$$v|i) = \sum_{\xi} G_{i\xi} |\xi)$$

which we shall refer here as to the matrix form of the EOP operator. By constructing the least-squares objective function

$$Z[\{G_{\xi}^{(i)}\}] = \int d\mathbf{r}_1 \left[v(\mathbf{r}_1) \phi_i(\mathbf{r}_1) - \sum_{\xi} G_{\xi}^{(i)} \phi_{\xi}(\mathbf{r}_1) \right]^2$$

and requiring that

$$\frac{\partial Z[\{G_{\xi}^{(i)}\}]}{\partial G_{\mu}^{(i)}} = 0 \text{ for all } \mu$$

we find the coefficients $G_{\xi}^{(i)}$ to be

$$\mathbf{G}^{(i)} = \mathbf{v}^{(i)} \cdot \mathbf{S}^{-1}$$

where

$$\begin{aligned} v_{\eta}^{(i)} &= (\eta|v|i) \\ S_{\eta\xi} &= (\eta|\xi) \end{aligned}$$

or explicitly

$$G_{i\xi} = \sum_{\eta} [S^{-1}]_{\xi\eta} (\eta|v|i)$$

identical to what we obtained from application of the resolution of identity in space spanned by non-orthogonal complete set of basis vectors.

Since matrix elements of an EOP operator in auxiliary space can be computed in the same way as the matrix elements with any other basis function, one can formally write the following identity

$$(X|v|i) = \sum_{\xi\eta} S_{X\xi} [\mathbf{S}^{-1}]_{\xi\eta} (\eta|v|i)$$

where X is an arbitrary orbital. When the other orbital does not belong to molecule A but to the (changing) environment, it is straightforward to compute the resulting matrix element, which is simply given as

$$(j_{\in B}|v^A|i_{\in A}) = \sum_{\xi} S_{j\xi} G_{i\xi}$$

where j denotes the other (environmental) basis function.

In the above equation, the EOP-part (fragment parameters for molecule A only) and the Solver-part (subject to be computed by solver on the fly) are separated. This then forms a basis for fragment-based approach to solve Quantum Chemistry problems related to the extended molecular aggregates.

3.2.2 Fitting in Incomplete Space

Density fitting scheme from previous section has practical disadvantage of a nearly-complete basis set being usually very large (spanned by large amount of basis set vectors). Any non-complete basis set won't work in the previous example. Since most of basis sets used in quantum chemistry do not form a complete set, it is beneficial to design a modified scheme in which it is possible to obtain the **effective** matrix elements of the EOP operator in an **incomplete** auxiliary space. This can be achieved by minimizing the following objective function

$$Z[\{G_{\xi}^{(i)}\}] = \iint d\mathbf{r}_1 d\mathbf{r}_2 \frac{\left[v(\mathbf{r}_1) \phi_i(\mathbf{r}_1) - \sum_{\xi} G_{\xi}^{(i)} \phi_{\xi}(\mathbf{r}_1) \right] \left[v(\mathbf{r}_2) \phi_i(\mathbf{r}_2) - \sum_{\eta} G_{\eta}^{(i)} \phi_{\eta}(\mathbf{r}_1) \right]}{|\mathbf{r}_1 - \mathbf{r}_2|}$$

Thus requesting that

$$\frac{\partial Z[\{G_{\xi}^{(i)}\}]}{\partial G_{\mu}^{(i)}} = 0 \text{ for all } \mu$$

we find the coefficients $G_{\xi}^{(i)}$ to be

$$\mathbf{G}^{(i)} = \mathbf{b}^{(i)} \cdot \mathbf{A}^{-1}$$

where

$$b_{\eta}^{(i)} = (\eta||vi)$$

$$A_{\eta\xi} = (\eta||\xi)$$

The symbol $||$ is to denote the operator r_{12}^{-1} and double integration over \mathbf{r}_1 and \mathbf{r}_2 . Thus, in order to use this generalized density fitting scheme one must to compute two-centre electron repulsion integrals (implemented in [oepdev::ERI_1_1](#)).

3.2.3 Fitting in Incomplete Space - Alternative Approach

The above method of density fitting of EOPs in incomplete space might be still relatively costly since it requires two-centre ERIs. However, there exists alternative approach which requires only overlap integrals. The EOP matrix is then given by

$$\mathbf{G}_m^\dagger = \mathbf{T}_{mX} \mathbf{S}_{XX'}^{-1} \mathbf{T}_{mX}^\dagger \mathbf{S}_{ma} \mathbf{G}_a^\dagger$$

where \mathbf{G}_a and \mathbf{G}_m are the EOP matrices in complete and incomplete basis, respectively. The auxiliary matrices read

$$\begin{aligned} \mathbf{T}_{mX} &= \mathbf{S}_{mm}^{-1} \mathbf{S}_{ma} \mathbf{T}_{aX} \mathbf{S}_{XX'}^{-1} \\ \mathbf{S}_{XX'}^{-1} &= \left(\mathbf{T}_{aX}^\dagger \mathbf{S}_{am} \mathbf{S}_{mm}^{-1} \mathbf{S}_{ma} \mathbf{T}_{aX} \right)^{\frac{1}{2}} \\ \mathbf{T}_{aX} &= \mathbf{S}_{aa}^{-1} \mathcal{Q} \mathbf{U}_{aX} \end{aligned}$$

The similarity transformation matrix \mathbf{T}_{aX} is obtained from the eigenvectors of the co-variance matrix, i.e.,

$$\mathbf{C}_{aa} = \mathbf{S}_{aa}^{\frac{1}{2}} \mathbf{G}_a^\dagger \mathbf{G}_a \mathbf{S}_{aa}^{\frac{1}{2}} = \mathbf{U}_{aX} \mathbf{g}_{XX} \mathbf{U}_{aX}^\dagger$$

The operator \mathcal{Q} selects only eigenvectors \mathbf{U}_{aX} associated with the non-vanishing eigenvalues stored in the diagonal matrix \mathbf{g}_{XX} . In practice, the number of such eigenvalues is bounded by the number of rows in EOP matrix, i.e., the number of states on which the EOP operator acts. Thus, substantial reduction of the basis set size is achieved which further reduces computational cost.

Implemented Models

4.1 Fragment-Based Methods

List of most important models implemented in the EOPDev project is given below. Among the interaction energy models are the second generation of the effective potential method (EFP2) [Gordon et al. \[2013\]](#) [Li et al. \[2006\]](#) [Xu and Gordon \[2013\]](#), perturbation theories of Murrel et al. [Murrell et al. \[1965\]](#), Otto and Ladik [Otto and Ladik \[1975\]](#) and Hayes and Stone [Hayes and Stone \[1984\]](#), density decomposition scheme (DDS) [Mandado and Hermida-Ramón \[2011\]](#), reduced variational space (RVS) method [Stevens and Fink \[1987\]](#). Among the excitation energy transfer (EET) coupling methods are the TrCAMM method [Błasiak et al. \[2015\]](#) and the transfer integral (TI) method [Fujimoto \[2012\]](#).

Table 1. Theoretical fragment-based models implemented in EOPDev.

Pauli energy	CT energy	EET Coupling
EFP2	EFP2	TrCAMM
Murrel et al.	Otto-Ladik	TI
EOP-Murrel et al.	EOP-Otto-Ladik	EOP-TI
Otto-Ladik		
EOP-Otto-Ladik		
DDS		
Hayes-Stone (exact)	RVS	Exact (ESD)

4.2 Target, Benchmark and Competing Models

The target models introduced in the Project are tested against the following benchmarks and compared with the following state-of-the-art models:

Table 2. Target models vs benchmarks and competitor models.

Target Model	Benchmarks	Competing Model
EOP-Murrel et al. (Pauli)	Murrel et al., DDS, Stone	EFP2 (Pauli)
EOP-Otto-Ladik (CT)	Otto-Ladik, RVS	EFP2 (CT)
EOP-TI	Exact (ESD), TI	TI

The target models contain their EOP-based versions, that can be executed in the `OEPDevSolver::compute_oep_based`, and compared with the corresponding benchmark models `OEPDevSolver::compute_benchmark`.

Contributing to EOPDev

EOPDev is a plugin to `Psi4`.

Therefore it should follow the programming etiquette of `Psi4`. Also, EOPDev has additional programming tips to make the code more versatile and easy to develop further. Here, I emphasise on most important aspects regarding the proposed **programming rules**.

5.1 Main Routine and Libraries

EOPDev has only *one* source file in the plugin base directory, i.e., `main.cc`. This is the main driver routine that handles the functionality of the whole EOP testing platform: specifies options for `Psi4` input file and implements test routines based on the options. Include files directly related to `main.cc` are stored in the `include` directory, where only header files are present. Options are specified in `include/oepdev_options.h` whereas macros and defines in `include/oepdev_files.h`. Other sources are stored in `MODULE/libNAME*` directories where `NAME` is the name of the library with sources and header files, whereas `MODULE` is the directory of the EOPDev module.

Things to remember:

1. **No other sources in base directory.** It is not permitted to place any new source or other files in the plugin base directory (i.e., where `main.cc` resides).
2. **Sources in library directories.** Any additional source code has to be placed in `oepdev/libNAME*` directory (either existing one or a new one; in the latter case remember to add the new `*.cc` files to `CMakeLists.txt` in the plugin base directory).
3. **Miscellanea in special directories.** If you want to add additional documentation, put it in the `doc` directory. If you want to add graphics, put it in the `images` directory.

5.2 Header Files in Libraries

Header files are handy in obtaining a quick glimpse of the functionality within certain library. Each library directory should contain at least one header file in EOPDev. However, header files can be problematic if not managed properly.

Things to remember:

1. **Header preprocessor variable.** Define the preprocessor variable specifying the existence of include of the particular header file. The format of such is

```
#ifndef MODULE_LIBRARY_HEADER.h
#define MODULE_LIBRARY_HEADER.h
// rest of your code goes here
#endif // MODULE_LIBRARY_HEADER.h
```

Last line is the **end** of the header file. The preprocessor variables represents the directory tree `oepdev/MODULE/LIBRARY/HEADER.h` structure (where `oepdev` is the base plugin directory). `MODULE` is the plugin module name (e.g. `oepdev`, the name of the module directory) `LIBRARY` is the name of the library (e.g. `libutil`, should be the same as library directory name) `HEADER` is the name of the header in library directory (e.g. `diis` for `diis.h` header file)

2. **Set module namespace.** To prevent naming clashes with other modules and with Psi4 it is important to operate in separate namespace (e.g. for a module).

```
namespace MODULE {
// your code goes here
} // EndNameSpace MODULE
```

For instance, all classes and functions in `oepdev` module are implemented within the namespace of the same label. Considering addition of other local namespaces within a module can also be useful in certain cases.

5.3 Environmental Variables

Defining the set of intrinsic environmental variables can help in code management and conditional compilation. The EOPDev environmental variables are defined in `include/oepdev_files.h` file. Remember also about psi4 environmental variables defined in `psi4/psifiles.h` header. As a rule, the EOPDev environmental variable should have the following format:

```
OEPDEV_XXXX
```

where XXXX is the descriptive name of variable.

5.4 Documenting the Code

Code has to be documented (at best at a time it is being created). The place for documentation is always in header files. Additional documentation can be also placed in source files. Leaving a chunk of code for a production run without documentation is unacceptable.

Use Doxygen style for documentation all the time. Remember that it supports markdown which can make the documentation even more clear and easy to understand. Additionally you can create a nice `.rst` documentation file for Sphinx program. If you are coding equations, always include formulae in the documentation!

Things to remember:

1. **Descriptions of classes, structures, global functions, etc.** Each programming object should have a description.
2. **Documentation for function arguments and return object.** Usage of functions and class methods should be explained by providing the description of all arguments (use `\param` and `\return` Doxygen keywords).
3. **One-line description of class member variables.** Any class member variable should be preceded by a one-liner documentation (starting from `///`).
4. **Do not be afraid of long names in the code.** Self-documenting code is a bless!

5.5 Naming Conventions

Naming is important because it helps to create more readable and clear self-documented code. Some loose suggestions:

1. **Do not be afraid of long names in the code, but avoid redundancy.** Examples of good and bad names: good name: `get_density_matrix`; bad name: `get_matrix`. Unless there is only one type of matrix a particular objects can store, `matrix` is not a good name for a getter method. good name: `class Wavefunction`, bad name: `class WFN` good name: `int numberOfErrorVectors`, bad name: `int nvec`, bad name: `the_number_of_error_vectors` good name: `class EFPotential`, probably bad name: `class EffectiveFragmentPotential`. The latter might be understood by some people as a class that inherits from `EffectiveFragment` class. If it is not the case, compromise between abbreviation and long description is OK.
2. **Short names are OK in special situations.** In cases meaning of a particular variable is obvious and it is frequently used in the code locally, it can be named shortly. Examples are: `i` when iterating `no` number of occupied orbitals, `nv` number of virtual orbitals, etc.
3. **Clumped names for variables and dashed names for functions.** Try to distinguish between variable name like `sizeofEOTypeList` and a method name `get_matrix()` (neither `size_of_EOP_type_list`, nor `getMatrix()`). This is little bit cosmetics, but helps in managing the code when it grows.

4. **Class names start from capital letter.** However, avoid only capital letters in class names, unless it is obvious. Avoid also dashes in class names (they are reserved for global functions and class methods). Examples: good name: `DIISManager`, bad name: `DIIS`. good name: `EETCouplingSolver`, bad name: `EETSolver`, very bad: `EET`.

5.6 Track Timing When Evaluating the Code

It is useful to track time elapsed for performing a particular task by a computer. For this, use for example `psi::timer_on` and `psi::timer_off` functions defined in `psi4/libqt/qt.h`. Psi4 always generates the report file `timer.dat` that contains all the defined timings. For example,

```
#include "psi/libqt/qt.h"
psi::timer_on("EOP      E(Paul) Murrell-et al S1  ");
// Your code goes here
psi::timer_off("EOP      E(Paul) Murrell-et al S1  ");
```

To maintain the printout in a neat form, the timing associated with the EOPDev code can be generated via `misc/python/timing.py` utility script.

5.7 Clean Memory Between Independent Jobs

If you use scratch disk space to store integrals, clean the scratch in between independent calculations. From C++ level invoke

```
#include "psi4/libpsio/psio.hpp"
// ...
psi::PSIOManager::shared_object()->psiclean();
```

whereas from the Python level use

```
import psi4
# ..
psi4.core.clean()
```

If the scratch space is not cleaned up before next independent task begins, certain computational routines might crash with `PSIOError` or continue without error, but produce wrong results.

5.8 Use Object-Oriented Programming

Try to organise your creations in objects having special relationships and data structures. Encapsulation helps in producing self-maintaining code and is much easier to use. Use:

- **factory design** for creating objects
- **container design** for designing data structures

- **polymorphism** when dealing with various flavours of one particular feature in the data structure

Note: In Psi4, factories are frequently implemented as static methods of the base classes, for example `psi::BasisSet::build` static method. It can be followed when building object factories in EOPDev too.

5.9 Implement Tests

When a computer code is updated by new features such as methods or algorithms, it becomes important to monitor its performance in order to ensure that it works correctly. To achieve this goal, a testing platform should be established, which contains a set of tests producing certain outputs and compare them with benchmark outputs. In EOPDev, `ctest` functionality is used as a testing platform for the C++ level code. Everytime you implement new feature in the code, it is very recommended to immediately supplement the testing platform with a new test. Remember to design tests carefully so that they address all potentially vulnerable aspects of added functionalities and a valid reference output can also be defined. See the tests in `oepdev/libtest` as well as `tests/oepdev` directories.

EOPDev is addressed for developers.

Make sure you have first read [the introduction](#) and are familiar with the EOP-based ERI elimination technique before proceeding.

6.1 Installation

6.1.1 Preparing Psi4

EOPDev is a Psi4 plugin. It requires `-Psi4, version 1.2.1` (git commit 406f4de). Has to be modified (see below). `-Eigen3`, any version.

Note

Before compiling, make sure EFP is enabled in `CMakeLists.txt` (now it is not used in EOPDev but maybe in the future it would).

Recently, Psi4 introduced API visibility management. Only certain Psi4 classes and functions are *exposed* in the `core.so` library, that is further linked to Psi4 plugin shared library. Due to this reason, not all Psi4 functionalities can be directly used from outside Psi4. In order to access local API of Psi4 (also used in the EOPDev code) slight modification of Psi4 code and concomitant rebuild is necessary.

In order to expose local API used by EOPDev and hidden within Psi4 1.2, two types of small modifications are necessary:

- M1: add `PSI_API` macro after required class or function declaration in header file
- M2: add `#include "psi4/pragma.h"` line at the include section of an appropriate header file

Modification M1 is obligatory for all affected files whereas modification M2 needs to be done only in headers that do not have "psi4/pragma.h" included explicitly or implicitly. The list of some Psi4 header files along with the respective changes that need to be done are listed in the table below:

Psi4 Header File	Psi4 Class	Required Changes
libfunctional/superfunctional.h	Superfunctional	M1
libscf_solver/hf.h	HF	M1
libscf_solver/rhf.h	RHF	M1
libcubeprop/csg.h	CubicScalarGrid	M1
libmints/onebody.h	OneBodyAOInt	M1
libmints/potential.h	PotentialInt	M1
libmints/multipoles.h	MultipoleInt	M1
libmints/multipolesymmetry.h	MultipoleSymmetry	M1
libmints/fjt.h	Taylor_Fjt	M1
libmints/fjt.h	Fjt	M1
libmints/oeprop.h	EOProp	M1, M2
libmints/gshell.h	GaussianShell	M1, M2

To quickly apply these and other required modifications, use the patch files stored in `misc/patch` directory. Please make sure to use a proper patch for a chosen Psi4 version.

6.1.2 Compilation

After all the above changes have been done in Psi4 (followed by its rebuild) compile the EOPDev code by running `compile` script. Make sure Eigen3 path is set to environment variable `EIGEN3_INCLUDE_DIR` (instructions will appear on the screen). After compilation is successful, run `ctest` to check if the code works fine.

Note

It may happen that during code development there will be symbol lookup error when importing `oepdev.so` (in such case EOPDev compiles without error but Python cannot import the module `oepdev`). In such circumstance, probably there some local Psi4 feature that is needed in EOPDev is not exposed by `PSI_API` macro. To fix this, run `c++filt [name]` where `[name]` is the mangled undefined symbol. This will show you which Psi4 class or function is not exposed and requires `PSI_API` (change M1 and perhaps M2 too). Such change requires Psi4 rebuild and recompilation of EOPDev code. In any case, please contact me and report new undefined symbol (blasiak.bartosz@gmail.com).

6.1.3 Step-By-Step Installation

To summarize, follow these steps:

1. **Modify Psi4.** Create a copy of your Psi4 source, and in the directory containing the main directory of Psi4 run

```
./${EOPDEV_PATH}/misc/patch/run_patch-psi4-1.2.1-git.406f4de
```

where `EOPDEV_PATH` variable denotes here your local source directory of EOPDev. Subsequently, compile the modified Psi4.

2. **Install Eigen3.** Set the path to Eigen3:

```
export EIGEN3_INCLUDE_DIR=/your_path.to.eigen3
```

and add it to your `.bash.rc` file.

3. **Install EOPDev.** Go to your EOPDev source directory and run

```
./compile
./create_doc
evince ${EOPDEV_PATH}/doc/doxygen/latex/refman.pdf
```

The first script installs the EOPDev plugin. The second generates the HTML and documentations on your computer (e.g., see `refman.pdf` for documentation in PDF format). Generation of the documentation is optional but highly recommended.

4. **Install GEFP Python Package.** Run the following commands:

```
cd ${EOPDEV_PATH}/gefp
python setup.py install
```

Administrative privileges might be necessary. Although installing GEFP is not necessary, it contains quite a few useful Python implementations of various quantum chemistry methods or handling with molecule and wavefunction objects. In particular, GEFP contains efficient and fully automatized implementation of the [extended density fitting of EOPs](#).

6.2 EOPDev Code Structure

As a plugin to Psi4, EOPDev consists of the `main.cc` file with the plugin main routine, `include/oepdev_options.h` specifying the options of the plugin, `include/oepdev_files.h` defining all global macros and environmental variables, as well as the `oepdev` directory. The latter contains the actual EOPDev code that is divided into several subdirectories called [modules](#).

6.2.1 Main Routine

Before the actual EOPDev calculations are started, the wavefunction of the input molecular aggregate is computed by Psi4. See the plugin driver script `pymodule.py` for more details on how the calculation environment is initialized. Subsequently, one out of four types of target operations can be performed by the program:

1. `OEP_BUILD` - Compute the EOP effective parameters for one molecule.

2. `DMATPOL` - Compute the generalized density matrix susceptibility tensors (DMS's) for one molecule.
3. `SOLVER` - Perform calculations for a molecular aggregate. As for now, only dimers are handled.
4. `TEST` - Perform the testing routine.

The first two modes are single molecule calculations. `OEP_BUILD` uses the `OEPotential::build` static factory to create EOP objects whereas `DMATPOL` uses the `GenEffParFactory::build` static factory to create generalized effective fragment parameters (GEFP's) for polarization.

Note

In the future, `OEP_BUILD` will be handled also by `GenEffParFactory::build` since EOP parameters are part of the GEFP's.

`SOLVER` requires at least molecular dimer and the `WavefunctionUnion` object (being the Hartree product of the unperturbed monomer wavefunctions) is constructed at the beginning, which is then passed to the `OEPDevSolver::build` static factory. `TEST` can refer to single- or multiple-molecule calculations, whereby each of the testing routines is listed in the `cmake/CTestTestfile.cmake.in` file.

6.2.2 Modules

The source code is distributed into directories called modules:

- `liboep`
- `libgefp`
- `libsolver`
- `libints`
- `libpsi`
- `lib3d`
- `libutil`
- `libtest`

See Modules for a detailed description of each of the modules.

6.3 EOPDev Classes: Overview

6.3.1 EOP Module

The EOP module located in `oepdev/liboep` consists of the following abstract bases:

- `OEPotential` implementing the EOP,
- `GeneralizedDensityFit` implementing the GDF technique.

Each of the bases contains static factory method called `build` that creates instances of chosen subclasses. The module contains also a structure `EOPType` which is a container storing all the data associated with a particular EOP: type name, dimensions, EOP coefficients and whether is density-fitted or not.

OEPotential

It is a container and computer class of EOP. Among others, the most important public method is `OEPotential::compute` which computes all the EOPs (by iterating over all possible EOP types within a chosen EOP subclass or category). EOPs can be extracted by `OEPotential::oep` method, for instance. From protected attributes, each `OEPotential` instance stores blocks of the LCAO-MO matrices associated with the occupied (`cOcc_`) and virtual (`cVir_`) MO's. It also contains the pointers to the primary, auxiliary and intermediate basis sets (`primary_`, `auxiliary_` and `intermediate_`, accordingly). Usage example:

```
#include "oepdev/liboep/oep.h"
oep = oepdev::OEPotential::build("ELECTROSTATIC ENERGY", wfn, options);
oep->compute();
oep->write_cube("V", "oep.cube.file");
```

So far, four `OEPotential` subclasses are implemented, from which `ElectrostaticEnergyOEPotential` and `RepulsionEnergyOEPotential` are fully operative, while the rest is under development.

GeneralizedDensityFit

Implements the density fitting schemes for EOPs.

6.3.2 GEFP Module

This module deals with the effective fragments constituting an extended molecular aggregate. It builds the platform to test various generalized effective fragment potentials (GEFP).

GenEffPar

Represents generalized effective fragment parameters.

GenEffParFactory

Implements routines of calculation of effective fragment parameters of various types.

GenEffFrag

Represents one effective fragment.

6.3.3 EOPDev Solver Module

This module sets up a simple platform of comparing benchmark and EOP-based fragment-based methods.

OEPEDevSolver

This is the main solver which as for now assumes molecular dimers (or bi-fragment systems). It is based on a union of wavefunctions of unperturbed monomers, `WavefunctionUnion`.

6.4 Developing EOPs

Note

This section is for illustrative purpose. The small details of the objects such as `OEPEType` and others can change over the years due to development of the EOPDev code. However, the overall programming scheme remains unchanged and valid.

EOPs are implemented in a suitable subclass of the `OEPEPotential` base. Due to the fact that EOPs can be density-based or DMTP-based, the classes `GeneralizedDensityFit` as well as `ESPSolver` are usually necessary in the implementations. Handling the one-electron integrals (OEs) and the two-electron integrals (ERIs) in AO basis is implemented in `IntegralFactory`. In particular, potential integrals evaluated at arbitrary centres can be accessed by using the `PotentialInt` instances. Useful iterators for looping over AO ERIs the `ShellCombinationsIterator` and `AOIntegralsIterator` classes. Transformations of OEs to MO basis can be easily achieved by transforming AO integral matrices by `cOcc_` and `cVir_` members of `OEPEPotential` instances, e.g., by using the `psi::Matrix::doublet` or `psi::Matrix::triplet` static methods. Transformations of ERIs to MO basis can be performed by using the `psi4/libtrans/integraltransform.h` library.

It is recommended that the implementation of all the new EOPs follows the following steps:

1. **Write the class framework.** This includes choosing a proper name of a `OEPEPotential` subclass, sketching the constructors and a destructor, and all the necessary methods.
2. **Implement EOP types.** Each type of EOP is implemented, including the 3D vector field in case DMTP-based EOPs are of use.

3. **Update base factory method.** Add appropriate entries in the `OEPotential::build` static factory method.

Below, we shall go through each of these steps separately and discuss them in detail.

6.4.1 Drafting an EOP Subclass

This stage is the design of the overall framework of EOP subclass. The name should end with `OEPotential` to maintain the convention used so far. The template for the header file definition can be depicted as follows:

```
class SampleOEPotential : public OEPotential
{
public:
    // Purely DMTP-based EOPs
    SampleOEPotential(SharedWavefunction wfn, Options& options);

    // GDF-based EOPs
    SampleOEPotential(SharedWavefunction wfn, SharedBasisSet auxiliary, SharedBasisSet intermediate,
        Options& options);

    // Necessary destructor
    virtual ~SampleOEPotential();

    // Necessary computer
    virtual void compute(const std::string& oepType) override;

    // Necessary computer
    virtual void compute_3D(const std::string& oepType,
        const double& x, const double& y, const double& z, std::shared_ptr<psi::Vector>
        & v) override;
    // Necessary printer
    virtual void print_header() const override;

private:
    // Set defaults - good practice
    void common_init();

    // Auxiliary computers - exemplary
    double compute_3D_sample_V(const double& x, const double& y, const double& z);
};
```

The constructors need to call the abstract base constructor and then specialized initializations. It is a good practice to put the specialized common initializers in a separate private method `common_init` (which is a convention in `Psi4` and is adopted also in `EOPDev`). For instance, the exemplary constructor is show below:

```
SampleOEPotential::SampleOEPotential(SharedWavefunction wfn,
    SharedBasisSet auxiliary, SharedBasisSet intermediate, Options&
    options)
: OEPotential(wfn, auxiliary, intermediate, options)
{
    common_init();
}

void SampleOEPotential::common_init()
{
    int n1 = wfn->Ca_subset("AO","OCC")->ncol();
    int n2 = auxiliary->nbf();
    int n3 = wfn->molecule()->natom();

    psi::SharedMatrix mat_1 = std::make_shared<psi::Matrix>("G(S^{-1})", n2, n1);
```

```

psi::SharedMatrix mat_2 = std::make_shared<psi::Matrix>("G(S^{-2})", n3, n1);

OEType type_1 = {"Murrell-et al.S1", true, n1, mat_1};
OEType type_2 = {"Otto-Ladik.S2", false, n1, mat_2};

oepTypes_[type_1.name] = type_1;
oepTypes_[type_2.name] = type_2;
}

```

Note that the `OEPotential::oepTypes_` attribute, which is a `std::map` of structures `OEType`, is initialized here. All the EOP types need to be stated in the constructors. Destructors usually call nothing, unless dynamically allocated memory is also of use.

It is also a good practice to already sketch the `compute` method here by adding certain private computers, like in the example below:

```

void SampleOEPotential::compute(const std::string& oepType)
{
    if (oepType == "Murrell-et al.S1") this->compute_murrell_et_al_s1(); // calls private method
    else if (oepType == "Otto-Ladik.S2") this->compute_otto_ladik_s2(); // calls private method
    else throw psi::PSIException("EOPDEV: Error. Incorrect EOP type specified!\n"); // for safety
}
void SampleOEPotential::compute_murrell_et_al_s1()
{
    psi::timer_on ("EOP      E(Paul) Murrell-et al S1  ");
    /* Your implementation goes here */
    psi::timer_off("EOP      E(Paul) Murrell-et al S1  ");
}

```

Implementing EOP Types

Implementation of the inner body of `compute` method requires populating the members of `oepTypes_` with data. This means, that for each EOP type there has to be a specific implementation of EOP parameters. GDF-based EOPs need to create the `psi::Matrix` with EOP parameters and put them into `oepTypes_`. In the case of DMTP-based EOPs `compute_3D` method has to be additionally implemented before `compute` is fully functional. To implement `compute_3D`, `OEPotential::make_oeps3d` method is of high relevance: it creates `OEPotential3D<T>` instances, where `T` is the EOP subclass. These instances are `Field3D` objects that define EOPs in 3D Euclidean space. For example,

```

void SampleOEPotential::compute_otto_ladik_s2()
{
    // Switch on timer
    psi::timer_on("EOP      E(Paul) Otto-Ladik S2      ");

    // Create 3D field, automated through 'make_oeps3d'. Requires 'compute_3D' implementation.
    std::shared_ptr<OEPotential3D<OEPotential>> oeps3d = this->make_oeps3d("Otto-Ladik.S2");
    oeps3d->compute();

    // Perform ESP fit to get EOP effective charges
    ESPSolver esp(oeps3d);
    esp.set_charge_sums(0.5);
    esp.compute();

    // Put the EOP coefficients into 'oepTypes_'
    for (int i=0; i<esp.charges()->nrow(); ++i) {
        for (int o=0; o<oepTypes_["Otto-Ladik.S2"].n; ++o) {
            oepTypes_["Otto-Ladik.S2"].matrix->set(i, o, esp.charges()->get(i, o));
        }
    }

    // Switch off timer
    psi::timer_off("EOP      E(Paul) Otto-Ladik S2      ");
}

```

```

}
// Necessary implementation for 'make_oeps3d' to work
void SampleOEPotential::compute_3D(const std::string& oepType, const double& x, const double& y, const
    double& z, std::shared_ptr<psi::Vector>& v)
{
    // Loop over all possibilities for EOP types and exclude illegal names
    if (oepType == "Otto-Ladik.S2") {

        // this computes the actual values of EOP = v(x,y,z) and stores it in 'vec_otto_ladik_s2_'
        this->compute_3D_otto_ladik_s2(x, y, z);

        // Assign final value to the buffer vector
        for (int o = 0; o < oepTypes_["Otto-Ladik.S2"].n; ++o) v->set(o, vec_otto_ladik_s2_[o]);

    }
    else if (oepType == "Murrell-et.al.S1" ) { /* Even if it is not DMTP-based EOP, this line is necessary */}
    else {
        throw psi::PSIEXCEPTION("EOPDEV: Error. Incorrect EOP type specified!\n"); // Safety
    }
}

```

Note that `make_oeps3d` is not overridable and is fully defined in the base. Do not call `OEPotential3D` constructors in the `OEPotential` subclass (it can be done only from the level of the abstract base where all the pointers are dynamically converted to an appropriate data type due to polymorphism)!

6.5 Examples

Exemplary demos of using the `EOPDev` code in C++ level can be found in the [testing platform](#). Additional examples can also be found in `doc/examples` directory. However, the latter examples might not fully compile since they are only for illustrative purposes as the code constantly develops. They are constantly updated as much as possible.

CHAPTER 7

Module Index

7.1 Modules

Here is a list of all modules:

The Generalized One-Electron Potentials Library	47
The EOPDev Solver Library	49
The Generalized Effective Fragment Potentials Library	50
The Integral Package Library	52
The Three-Dimensional Vector Fields Library	62
The Density Functional Theory Library	65
The EOPDev Utilities	66
The EOPDev Testing Platform Library	81

Namespace Index

8.1 Namespace List

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

oepdev	OEPEv module namespace	83
psi	Psi4 package namespace	93

9.1 Class Hierarchy

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

oepdev::ABCD	95
oepdev::AOIntegralsIterator	106
oepdev::AllAOIntegralsIterator_2	97
oepdev::AllAOIntegralsIterator_4	98
oepdev::CISData	121
oepdev::CPHF	122
CubicScalarGrid	
oepdev::CubePointsCollection3D	127
oepdev::DavidsonLiu	128
oepdev::CISComputer	114
oepdev::R_CISComputer	270
oepdev::R_CISComputer_Explicit	273
oepdev::R_CISComputer_Direct	270
oepdev::R_CISComputer_DL	271
oepdev::U_CISComputer	308
oepdev::U_CISComputer_Explicit	310
oepdev::U_CISComputer_DL	309
oepdev::DIISManager	132
enable_shared_from_this	
oepdev::DMTPole	134
oepdev::CAMM	108
oepdev::GenEffFrag	182
oepdev::OEPDevSolver	232
oepdev::ChargeTransferEnergySolver	111

oepdev::EETCouplingSolver	149
oepdev::ElectrostaticEnergySolver	160
oepdev::RepulsionEnergySolver	277
oepdev::OEPotential	242
oepdev::ChargeTransferEnergyOEPotential	110
oepdev::EETCouplingOEPotential	148
oepdev::ElectrostaticEnergyOEPotential	159
oepdev::RepulsionEnergyOEPotential	276
oepdev::ESPSolver	169
oepdev::Field3D	173
oepdev::ElectrostaticPotential3D	163
oepdev::OEPotential3D < T >	248
oepdev::Fourier5	177
oepdev::Fourier9	177
oepdev::FragmentedSystem	178
oepdev::GenEffPar	188
oepdev::GenEffParFactory	201
oepdev::EFP2_GEFactory	156
oepdev::OEP_EFP2_GEFactory	231
oepdev::PolarGEFactory	259
oepdev::AbInitioPolarGEFactory	95
oepdev::UnitaryTransformedMOPolarGEFactory	332
oepdev::FFAbInitioPolarGEFactory	172
oepdev::GeneralizedPolarGEFactory	209
oepdev::NonUniformEFieldPolarGEFactory	228
oepdev::LinearGradientNonUniformEFieldPolarGEFactory	221
oepdev::LinearNonUniformEFieldPolarGEFactory	222
oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory	263
oepdev::QuadraticNonUniformEFieldPolarGEFactory	264
oepdev::UniformEFieldPolarGEFactory	311
oepdev::LinearUniformEFieldPolarGEFactory	223
oepdev::QuadraticUniformEFieldPolarGEFactory	265
oepdev::GeneralizedDensityFit	206
oepdev::DoubleGeneralizedDensityFit	145
oepdev::OverlapGeneralizedDensityFit	250
oepdev::SingleGeneralizedDensityFit	290
oepdev::GramSchmidt	215
IntegralFactory	
oepdev::IntegralFactory	218
oepdev::KabschSuperimposer	219
oepdev::MultipoleConvergence	224
oepdev::ObaraSaikaTwoCenterEFPRecursion.New	229
oepdev::OEType	249
OneBodyAOInt	
oepdev::EFPMultipolePotentialInt	158
oepdev::EFPMultipolePotentialInt	158
oepdev::PerturbCharges	251

oepdev::Points3DIterator::Point	252
oepdev::Points3DIterator	252
oepdev::CubePoints3DIterator	126
oepdev::RandomPoints3DIterator	274
oepdev::PointsCollection3D	256
oepdev::CubePointsCollection3D	127
oepdev::RandomPointsCollection3D	275
PotentialInt	
oepdev::PotentialInt	260
oepdev::QUAMBO	266
oepdev::QUAMBOData	269
RHF	
oepdev::RHFPerturbed	283
oepdev::ShellCombinationsIterator	285
oepdev::AllAOShellCombinationsIterator_2	100
oepdev::AllAOShellCombinationsIterator_4	103
oepdev::GeneralizedPolarGEFactory::StatisticalSet	291
oepdev::test::Test	292
oepdev::TIData	295
TwoBodyAOInt	
oepdev::TwoBodyAOInt	304
oepdev::TwoElectronInt	306
oepdev::ERI_1_1	164
oepdev::ERI_2_2	166
oepdev::ERI_3_1	167
oepdev::UnitaryOptimizer	312
oepdev::UnitaryOptimizer_2	318
oepdev::UnitaryOptimizer_2_1	323
oepdev::UnitaryOptimizer_4_2	327
Wavefunction	
oepdev::WavefunctionUnion	333

CHAPTER 10

Class Index

10.1 Class List

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

oepdev::ABCD	Simple structure to hold the Fourier series expansion coefficients	95
oepdev::AbInitioPolarGEFactory	Polarization GEFP Factory from First Principles. Hartree-Fock Approximation	95
oepdev::AllAOIntegralsIterator_2	Loop over all possible ERI within a particular shell doublet	97
oepdev::AllAOIntegralsIterator_4	Loop over all possible ERI within a particular shell quartet	98
oepdev::AllAOShellCombinationsIterator_2	Loop over all possible ERI shells in a shell doublet	100
oepdev::AllAOShellCombinationsIterator_4	Loop over all possible ERI shells in a shell quartet	103
oepdev::AOIntegralsIterator	Iterator for AO Integrals. Abstract Base	106
oepdev::CAMM	Cumulative Atomic Multipole Moments	108
oepdev::ChargeTransferEnergyOEPotential	Generalized One-Electron Potential for Charge-Transfer Interaction Energy . .	110
oepdev::ChargeTransferEnergySolver	Compute the Charge-Transfer interaction energy between unperturbed wave- functions	111
oepdev::CISComputer	CISComputer	114
oepdev::CISData	CIS wavefunction parameters. Container structure	121

oepdev::CPHF	
CPHF solver class	122
oepdev::CubePoints3DIterator	
Iterator over a collection of points in 3D space. g09 Cube-like order	126
oepdev::CubePointsCollection3D	
G09 cube-like ordered collection of points in 3D space	127
oepdev::DavidsonLiu	
Davidson-Liu diagonalization method	128
oepdev::DIISManager	
DIIS manager	132
oepdev::DMTPole	
Distributed Multipole Analysis Container and Computer. Abstract Base	134
oepdev::DoubleGeneralizedDensityFit	
Generalized Density Fitting Scheme - Double Fit	145
oepdev::EETCouplingOEPotential	
Generalized One-Electron Potential for EET coupling calculations	148
oepdev::EETCouplingSolver	
Compute the EET coupling energy between unperturbed wavefunctions	149
oepdev::EFP2_GEFactory	
EFP2 GEFP Factory	156
oepdev::EFPMultipolePotentialInt	
Computes potential integrals	158
oepdev::ElectrostaticEnergyOEPotential	
Generalized One-Electron Potential for Electrostatic Energy	159
oepdev::ElectrostaticEnergySolver	
Compute the Coulombic interaction energy between unperturbed wavefunctions	160
oepdev::ElectrostaticPotential3D	
Electrostatic potential of a molecule	163
oepdev::ERI_1_1	
2-centre ERI of the form $\langle a O(2) b \rangle$ where $O(2) = 1/r^{12}$	164
oepdev::ERI_2_2	
4-centre ERI of the form $\langle ab O(2) cd \rangle$ where $O(2) = 1/r^{12}$	166
oepdev::ERI_3_1	
4-centre ERI of the form $\langle abc O(2) d \rangle$ where $O(2) = 1/r^{12}$	167
oepdev::ESPSolver	
Charges from Electrostatic Potential (ESP). A solver-type class	169
oepdev::FFAbInitioPolarGEFactory	
Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory	172
oepdev::Field3D	
General Vector Field in 3D Space. Abstract base	173
oepdev::Fourier5	
Simple structure to hold the Fourier series expansion coefficients for $N=2$	177
oepdev::Fourier9	
Simple structure to hold the Fourier series expansion coefficients for $N=4$	177
oepdev::FragmentedSystem	
Molecular System for Fragment-Based Calculations	178

oepdev::GenEffFrag	
Generalized Effective Fragment. Container Class	182
oepdev::GenEffPar	
Generalized Effective Fragment Parameters. Container Class	188
oepdev::GenEffParFactory	
Generalized Effective Fragment Factory. Abstract Base	201
oepdev::GeneralizedDensityFit	
Generalized Density Fitting Scheme. Abstract Base	206
oepdev::GeneralizedPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	209
oepdev::GramSchmidt	
Gram-Schmidt orthogonalization method	215
oepdev::IntegralFactory	
Extended IntegralFactory for computing integrals	218
oepdev::KabschSuperimposer	
Compute the Cartesian rotation matrix between two structures	219
oepdev::LinearGradientNonUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	221
oepdev::LinearNonUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	222
oepdev::LinearUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	223
oepdev::MultipoleConvergence	
Multipole Convergence	224
oepdev::NonUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	228
oepdev::ObaraSaikaTwoCenterEFPRecursion.New	
Obara-Saika recursion formulae for improved EFP multipole potential integrals	229
oepdev::OEP_EFP2_GEFactory	
OEP-EFP2 GEFP Factory	231
oepdev::OEPDevSolver	
Solver of properties of molecular aggregates. Abstract base	232
oepdev::OEPotential	
Generalized One-Electron Potential: Abstract base	242
oepdev::OEPotential3D< T >	
Class template for OEP 3D fields	248
oepdev::OEType	
Container to handle the type of One-Electron Potentials	249
oepdev::OverlapGeneralizedDensityFit	
Generalized Density Fitting Scheme - Single Fit Based on Minimal Overlap in MO Basis	250
oepdev::PerturbCharges	
Structure to hold perturbing charges	251
oepdev::Points3DIterator::Point	252
oepdev::Points3DIterator	
Iterator over a collection of points in 3D space. Abstract base	252
oepdev::PointsCollection3D	
Collection of points in 3D space. Abstract base	256

oepdev::PolarGEFactory	
Polarization GEFP Factory. Abstract Base	259
oepdev::PotentialInt	
Computes potential integrals	260
oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	263
oepdev::QuadraticNonUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	264
oepdev::QuadraticUniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	265
oepdev::QUAMBO	
The Quasiatomic Minimal Basis Set Molecular Orbitals (QUAMBO)	266
oepdev::QUAMBOData	
Container to store the QUAMBO data	269
oepdev::R_CISComputer	270
oepdev::R_CISComputer_Direct	270
oepdev::R_CISComputer_DL	
CIS Computer with RHF reference: Davidson-Liu Solver	271
oepdev::R_CISComputer_Explicit	273
oepdev::RandomPoints3DIterator	
Iterator over a collection of points in 3D space. Random collection	274
oepdev::RandomPointsCollection3D	
Collection of random points in 3D space	275
oepdev::RepulsionEnergyOEPotential	
Generalized One-Electron Potential for Pauli Repulsion Energy	276
oepdev::RepulsionEnergySolver	
Compute the Pauli-Repulsion interaction energy between unperturbed wave-functions	277
oepdev::RHPerturbed	
RHF theory under electrostatic perturbation	283
oepdev::ShellCombinationsIterator	
Iterator for Shell Combinations. Abstract Base	285
oepdev::SingleGeneralizedDensityFit	
Generalized Density Fitting Scheme - Single Fit	290
oepdev::GeneralizedPolarGEFactory::StatisticalSet	
A structure to handle statistical data	291
oepdev::test::Test	
Manages test routines	292
oepdev::TIData	
Transfer Integral EET Data	295
oepdev::TwoBodyAOInt	304
oepdev::TwoElectronInt	
General Two Electron Integral	306
oepdev::U_CISComputer	308
oepdev::U_CISComputer_DL	
CIS Computer with UHF reference: Davidson-Liu Solver	309
oepdev::U_CISComputer_Explicit	310

oepdev::UniformEFieldPolarGEFactory	
Polarization GEFP Factory with Least-Squares Parameterization	311
oepdev::UnitaryOptimizer	
Find the optimum unitary matrix of quadratic matrix equation	312
oepdev::UnitaryOptimizer_2	
Find the optimum unitary matrix for quadratic matrix equation with trace	318
oepdev::UnitaryOptimizer_2_1	323
oepdev::UnitaryOptimizer_4_2	
Find the optimum unitary matrix for quartic-quadratic matrix equation with trace	327
oepdev::UnitaryTransformedMOPolarGEFactory	
Polarization GEFP Factory with Least-Squares Scaling of MO Space	332
oepdev::WavefunctionUnion	
Union of two Wavefunction objects	333

CHAPTER 11

File Index

11.1 File List

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

main.cc	344
include/ oepdev_files.h	343
include/ oepdev_options.h	343
include/doxygen/ oepdev_manual.h	??
include/doxygen/ oepdev_modules.h	??
include/doxygen/ oepdev_namespaces.h	??
oepdev/lib3d/ dmtp.h	345
oepdev/lib3d/ esp.h	346
oepdev/lib3d/ space3d.h	??
oepdev/libgefp/ gefp.h	346
oepdev/libints/ eri.h	348
oepdev/libints/ recurr.h	349
oepdev/liboep/ oep.h	350
oepdev/liboep/ oep_gdf.h	351
oepdev/libpsi/ integral.h	351
oepdev/libpsi/ multipole_potential.h	??
oepdev/libpsi/ osrecur.h	352
oepdev/libpsi/ potential.h	353
oepdev/libpsi/bck/ multipole_potential.h	??
oepdev/libsolver/ solver.h	353
oepdev/libsolver/ ti_data.h	354
oepdev/libtest/ test.h	355
oepdev/libutil/ basis_rotation.h	355
oepdev/libutil/ cis.h	358
oepdev/libutil/ cphf.h	??

oepdev/libutil/ davidson_liu.h	359
oepdev/libutil/ diis.h	359
oepdev/libutil/ gram_schmidt.h	360
oepdev/libutil/ integrals_iter.h	360
oepdev/libutil/ kabsch_superimposer.h	361
oepdev/libutil/ quambo.h	362
oepdev/libutil/ scf_perturb.h	362
oepdev/libutil/ unitary_optimizer.h	363
oepdev/libutil/ util.h	364
oepdev/libutil/ wavefunction_union.h	366

12.1 The Generalized One-Electron Potentials Library

Implements the goal of this project: The Generalized One-Electron Potentials (EOPs). You will find here EOPs for computation of Pauli repulsion energy, charge-transfer energy and others. The routines for the generalized density fitting are also implemented here. Located at `oepdev/liboep`.

Classes

- struct `oepdev::OEType`
Container to handle the type of One-Electron Potentials.
- class `oepdev::OEPotential`
Generalized One-Electron Potential: Abstract base.
- class `oepdev::ElectrostaticEnergyOEPotential`
Generalized One-Electron Potential for Electrostatic Energy.
- class `oepdev::RepulsionEnergyOEPotential`
Generalized One-Electron Potential for Pauli Repulsion Energy.
- class `oepdev::ChargeTransferEnergyOEPotential`
Generalized One-Electron Potential for Charge-Transfer Interaction Energy.
- class `oepdev::EETCouplingOEPotential`
Generalized One-Electron Potential for EET coupling calculations.
- class `oepdev::GeneralizedDensityFit`
Generalized Density Fitting Scheme. Abstract Base.
- class `oepdev::SingleGeneralizedDensityFit`
Generalized Density Fitting Scheme - Single Fit.

- class `oepdev::DoubleGeneralizedDensityFit`
Generalized Density Fitting Scheme - Double Fit.
- class `oepdev::OverlapGeneralizedDensityFit`
Generalized Density Fitting Scheme - Single Fit Based on Minimal Overlap in MO Basis.

Typedefs

- using `oepdev::SharedOEPotential` = `std::shared_ptr< OEPotential >`

12.1.1 Detailed Description

12.2 The EOPDev Solver Library

Implementations of various solvers for molecular properties as a functions of unperturbed monomeric wavefunctions. This is the place all target EOP-based models are implemented and compared with benchmark and competitor models. Located at `oepdev/libsolver`.

Classes

- class `oepdev::OEPDevSolver`
Solver of properties of molecular aggregates. Abstract base.
- class `oepdev::ElectrostaticEnergySolver`
Compute the Coulombic interaction energy between unperturbed wavefunctions.
- class `oepdev::RepulsionEnergySolver`
Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.
- class `oepdev::ChargeTransferEnergySolver`
Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.
- class `oepdev::EETCouplingSolver`
Compute the EET coupling energy between unperturbed wavefunctions.
- class `oepdev::TIData`
Transfer Integral EET Data.

12.2.1 Detailed Description

12.3 The Generalized Effective Fragment Potentials Library

Implements the GEFP method, the far goal of the EOPDev project. Here you will find the containers for GEFP parameters, the density matrix susceptibility tensors and GEFP solvers. Located at `oepdev/libgefp`.

Classes

- class `oepdev::GenEffPar`
Generalized Effective Fragment Parameters. Container Class.
- class `oepdev::GenEffFrag`
Generalized Effective Fragment. Container Class.
- class `oepdev::GenEffParFactory`
Generalized Effective Fragment Factory. Abstract Base.
- class `oepdev::EFP2_GEFactory`
EFP2 GEFP Factory.
- class `oepdev::OEP_EFP2_GEFactory`
OEP-EFP2 GEFP Factory.
- class `oepdev::PolarGEFactory`
Polarization GEFP Factory. Abstract Base.
- class `oepdev::AbInitioPolarGEFactory`
Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.
- class `oepdev::FFAbInitioPolarGEFactory`
Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.
- class `oepdev::GeneralizedPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::UniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::NonUniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::LinearUniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::QuadraticUniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::LinearNonUniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::QuadraticNonUniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::LinearGradientNonUniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.

- class `oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory`
Polarization GEFP Factory with Least-Squares Parameterization.
- class `oepdev::UnitaryTransformedMOPolarGEFactory`
Polarization GEFP Factory with Least-Squares Scaling of MO Space.
- class `oepdev::FragmentedSystem`
Molecular System for Fragment-Based Calculations.

Typedefs

- using `oepdev::SharedGenEffPar` = `std::shared_ptr< GenEffPar >`
GEFP Parameters container.
- using `oepdev::SharedGenEffParFactory` = `std::shared_ptr< GenEffParFactory >`
GEFP Parameter factory.
- using `oepdev::SharedGenEffFrag` = `std::shared_ptr< GenEffFrag >`
GEFP Fragment container.
- using `oepdev::SharedFragmentedSystem` = `std::shared_ptr< FragmentedSystem >`
Fragmented system.

12.3.1 Detailed Description

The objective is to implement the framework for the fragment-based (FB) calculations in which the system is divided into interacting fragments. The functionality relies on a few data structures:

- the `GenEffFrag` - Generalized Effective Fragment
- the `GenEffPar` - Generalized Effective Parameters
- the `GenEffParFactory` - Generalized Effective Parameters Factory Fragments can contain multiple types of parameters, e.g., ethylene fragment can have EFP2, EOP-EFP2 as well as EOP-EET parameters. Fragments can be superimposed on target structures and the class contain methods that evaluate properties based on the fragments in the system.

12.4 The Integral Package Library

Implementations of various two-, three- or four-centre two-body electron repulsion integrals via utilizing the McMurchie-Davidson recurrence scheme. Located at `oepdev/libints` and `oepdev/libpsi`.

Classes

- class `oepdev::TwoElectronInt`
General Two Electron Integral.
- class `oepdev::ERI_1_1`
2-centre ERI of the form $(a|O(2)|b)$ where $O(2) = 1/r^{12}$.
- class `oepdev::ERI_2_2`
4-centre ERI of the form $(ab|O(2)|cd)$ where $O(2) = 1/r^{12}$.
- class `oepdev::ERI_3_1`
4-centre ERI of the form $(abc|O(2)|d)$ where $O(2) = 1/r^{12}$.
- class `oepdev::EFPMultipolePotentialInt`
Computes potential integrals.
- class `oepdev::TwoBodyAOInt`
- class `oepdev::IntegralFactory`
Extended `IntegralFactory` for computing integrals.
- class `oepdev::ObaraSaikaTwoCenterEFPRecursion_New`
Obara-Saika recursion formulae for improved EFP multipole potential integrals.
- class `oepdev::PotentialInt`
Computes potential integrals.

Macros

- `#define D1_INDEX(x, i, n) ((81*(x))+(9*(i))+(n))`
Get the index of McMurchie-Davidson-Hermite D1 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momentum i of function 1, and the Hermite index n .
- `#define D2_INDEX(x, i, j, n) ((1377*(x))+(153*(i))+(17*(j))+(n))`
Get the index of McMurchie-Davidson-Hermite D2 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j of function 1 and 2, and the Hermite index n .
- `#define D3_INDEX(x, i, j, k, n) ((18225*(x))+(2025*(i))+(225*(j))+(25*(k))+(n))`
Get the index of McMurchie-Davidson-Hermite D3 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j and k of function 1, 2 and 3, and the Hermite index n .
- `#define R_INDEX(n, l, m, j) ((14739*(n))+(867*(l))+(51*(m))+(j))`
Get the index of McMurchie-Davidson R coefficient stored in the `mdh_buffer_R_` from angular momenta n, l and m and the Boys index j .

Functions

- double `oepdev::d_N_n1_n2` (int N, int n1, int n2, double PA, double PB, double aP)
Compute McMurchie-Davidson-Hermite (MDH) coefficient for binomial expansion.
- void `oepdev::make_mdh_D1_coeff` (int n1, double aPd, double *buffer)
Compute the McMurchie-Davidson-Hermite coefficients for monomial expansion.
- void `oepdev::make_mdh_D2_coeff` (int n1, int n2, double aPd, double *PA, double *PB, double *buffer)
Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion.
- void `oepdev::make_mdh_D3_coeff` (int n1, int n2, int n3, double aPd, double *PA, double *PB, double *PC, double *buffer)
Compute the McMurchie-Davidson-Hermite coefficients for trinomial expansion.
- void `oepdev::make_mdh_D2_coeff_explicit_recursion` (int n1, int n2, double aP, double *PA, double *PB, double *buffer)
Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion by explicit recursion. This function makes the same changes to buffers as `oepdev::make_mdh_D2_coeff`, but implements it through explicit recursion by calls to `oepdev::d_N_n1_n2`. Therefore, it is slightly slower. Here for debugging purposes.
- void `oepdev::make_mdh_R_coeff` (int N, int L, int M, double alpha, double a, double b, double c, double *F, double *buffer)
Compute the McMurchie-Davidson R coefficients.

12.4.1 Detailed Description

Here, we define the primitive Gaussian type functions (GTOs)

$$\begin{aligned}\phi_i(\mathbf{r}) &\equiv x_A^{n_1} y_A^{l_1} z_A^{m_1} e^{-\alpha_1 r_A^2} \\ \phi_j(\mathbf{r}) &\equiv x_B^{n_2} y_B^{l_2} z_B^{m_2} e^{-\alpha_2 r_B^2} \\ \phi_k(\mathbf{r}) &\equiv x_C^{n_3} y_C^{l_3} z_C^{m_3} e^{-\alpha_3 r_C^2}\end{aligned}$$

in which $\mathbf{r}_A \equiv \mathbf{r} - \mathbf{A}$ and so on. \mathbf{A} is the centre of the GTO, α_1 its exponent, whereas n_1, l_1, m_1 the Cartesian angular momenta, with the total angular momentum $\theta_1 = n_1 + l_1 + m_1$.

In EOPDev implementations, the following definition shall be in use:

$$\begin{aligned}\mathbf{P} &\equiv \frac{\alpha_1 \mathbf{A} + \alpha_2 \mathbf{B}}{\alpha_1 + \alpha_2} \\ \mathbf{Q} &\equiv \frac{\alpha_3 \mathbf{C} + \alpha_4 \mathbf{D}}{\alpha_3 + \alpha_4} \\ \mathbf{R} &\equiv \frac{\alpha_1 \mathbf{A} + \alpha_2 \mathbf{B} + \alpha_3 \mathbf{C}}{\alpha_1 + \alpha_2 + \alpha_3} \\ \alpha_P &\equiv \alpha_1 + \alpha_2 \\ \alpha_Q &\equiv \alpha_3 + \alpha_4 \\ \alpha_R &\equiv \alpha_1 + \alpha_2 + \alpha_3\end{aligned}$$

The unnormalized products of primitive GTOs are denoted here as

$$\begin{aligned}[ij] &\equiv \phi_i(\mathbf{r})\phi_j(\mathbf{r}) \\ [ijk] &\equiv \phi_i(\mathbf{r})\phi_j(\mathbf{r})\phi_k(\mathbf{r})\end{aligned}$$

12.4.2 Hermite Operators

It is convenient to define

$$\Lambda_j(x_P; \alpha_P) \equiv \left(\frac{\partial}{\partial P_x} \right)^j = \alpha_P^{j/2} H_j(\sqrt{\alpha_P} x_P)$$

where $H_j(x)$ is the Hermite polynomial of order j evaluated at x . Introduction of the above Hermite operator can be used by invoking the recurrence relationship due to Hermite polynomial properties:

$$x_A \Lambda_j(x_P; \alpha_P) = j \Lambda_{j-1} + |\mathbf{P} - \mathbf{A}|_x \Lambda_j + \frac{1}{2\alpha_P} \Lambda_{j+1}$$

This can be directly used to derive very useful McMurchie-Davidson-Hermite coefficients as expansion coefficients of the polynomial expansions.

Polynomial Expansions as Hermite Series

By using the previous relation, it is possible to express the following expansions in Hermite series:

$$\begin{aligned}x_A^{n_1} &= \sum_{N=0}^{n_1} d_N^{n_1} \Lambda_N(x_A; \alpha_A) \\ x_A^{n_1} x_B^{n_2} &= \sum_{N=0}^{n_1+n_2} d_N^{n_1 n_2} \Lambda_N(x_P; \alpha_P) \\ x_A^{n_1} x_B^{n_2} x_C^{n_3} &= \sum_{N=0}^{n_1+n_2+n_3} d_N^{n_1 n_2 n_3} \Lambda_N(x_R; \alpha_R)\end{aligned}$$

The recurrence relationships can be easily found and they read

$$d_N^{n_1+1} = \frac{1}{2\alpha_A} d_{N-1}^{n_1} + (N+1) d_{N+1}^{n_1}$$

as well as

$$\begin{aligned}d_N^{n_1+1, n_2} &= \frac{1}{2\alpha_P} d_{N-1}^{n_1 n_2} + |\mathbf{P} - \mathbf{A}|_x d_N^{n_1 n_2} + (N+1) d_{N+1}^{n_1 n_2} \\ d_N^{n_1, n_2+1} &= \frac{1}{2\alpha_P} d_{N-1}^{n_1 n_2} + |\mathbf{P} - \mathbf{B}|_x d_N^{n_1 n_2} + (N+1) d_{N+1}^{n_1 n_2}\end{aligned}$$

and

$$\begin{aligned} d_N^{n_1+1,n_2,n_3} &= \frac{1}{2\alpha_R} d_{N-1}^{n_1 n_2 n_3} + |\mathbf{R} - \mathbf{A}|_x d_N^{n_1 n_2 n_3} + (N+1) d_{N+1}^{n_1 n_2 n_3} \\ d_N^{n_1,n_2+1,n_3} &= \frac{1}{2\alpha_R} d_{N-1}^{n_1 n_2 n_3} + |\mathbf{R} - \mathbf{B}|_x d_N^{n_1 n_2 n_3} + (N+1) d_{N+1}^{n_1 n_2 n_3} \\ d_N^{n_1,n_2,n_3+1} &= \frac{1}{2\alpha_R} d_{N-1}^{n_1 n_2 n_3} + |\mathbf{R} - \mathbf{C}|_x d_N^{n_1 n_2 n_3} + (N+1) d_{N+1}^{n_1 n_2 n_3} \end{aligned}$$

respectively. The first elements are given by

$$\begin{aligned} d_0^0 &= 1 \\ d_0^{00} &= 1 \\ d_0^{000} &= 1 \end{aligned}$$

By using the above formalisms, it is straightforward to express the doublet of primitive GTOs as

$$[ij] = E_{ij} \sum_{N=0}^{n_1+n_2} \sum_{L=0}^{l_1+l_2} \sum_{M=0}^{m_1+m_2} d_N^{n_1 n_2} d_L^{l_1 l_2} d_M^{m_1 m_2} \Lambda_N(x_P) \Lambda_L(y_P) \Lambda_M(z_P) e^{-\alpha_P r_P^2}$$

Analogously, the triplet of primitive GTOs is given by

$$[ijk] = E_{ijk} \sum_{N=0}^{n_1+n_2+n_3} \sum_{L=0}^{l_1+l_2+l_3} \sum_{M=0}^{m_1+m_2+m_3} d_N^{n_1 n_2 n_3} d_L^{l_1 l_2 l_3} d_M^{m_1 m_2 m_3} \Lambda_N(x_R) \Lambda_L(y_R) \Lambda_M(z_R) e^{-\alpha_R r_R^2}$$

The multiplicative constants are given by

$$\begin{aligned} E_{ij}(\alpha_1, \alpha_2) &= \exp \left[-\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \\ E_{ijk}(\alpha_1, \alpha_2, \alpha_3) &= \exp \left[-\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \exp \left[-\frac{(\alpha_1 + \alpha_2) \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3} |\mathbf{P} - \mathbf{C}|^2 \right] \end{aligned}$$

12.4.3 One-Body Integrals over Hermite Functions

The fundamental Hermite integrals that appear during computations of any kind of one-body integrals over GTOs are as follows

$$[NLM|\Theta(1)] \equiv \int d\mathbf{r}_1 \Theta(\mathbf{r}_1) \Lambda_N(x_{1P}; \alpha_P) \Lambda_L(y_{1P}; \alpha_P) \Lambda_M(z_{1P}; \alpha_P) e^{-\alpha_P r_{1P}^2}$$

It immediately follows that the overlap, dipole, quadrupole and potential integrals are given as

$$\begin{aligned} [NLM|1] &= \delta_{N0} \delta_{L0} \delta_{M0} \left(\frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|x_C] &= [\delta_{N1} + |\mathbf{P}\mathbf{C}|_x \delta_{N0}] \delta_{L0} \delta_{M0} \left(\frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|x_C^2] &= \left[2\delta_{N2} + 2|\mathbf{P}\mathbf{C}|_x \delta_{N1} + \left(|\mathbf{P}\mathbf{C}|_x^2 + \frac{1}{2\alpha_P} \right) \delta_{N0} \right] \delta_{L0} \delta_{M0} \left(\frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|x_C y_C] &= (\delta_{N1} + |\mathbf{P}\mathbf{C}|_x \delta_{N0}) (\delta_{L1} + |\mathbf{P}\mathbf{C}|_y \delta_{L0}) \delta_{M0} \left(\frac{\pi}{\alpha_P} \right)^{3/2} \\ [NLM|r_C^{-1}] &= \frac{2\pi}{\alpha_P} R_{NLM} \end{aligned}$$

The coefficients R_{NLM} are discussed in separate section below.

12.4.4 Two-Body Integrals over Hermite Functions

The fundamental Hermite integrals that appear during computations of any kind of two-electron integrals over GTOs are as follows

$$[N_1 L_2 M_2 | N_2 L_2 M_2] \equiv \iint d\mathbf{r}_1 d\mathbf{r}_2 \Lambda_{N_1}(x_{1P}; \alpha_P) \Lambda_{L_1}(y_{1P}; \alpha_P) \Lambda_{M_1}(z_{1P}; \alpha_P) \\ \times \Lambda_{N_2}(x_{2Q}; \alpha_Q) \Lambda_{L_2}(y_{2Q}; \alpha_Q) \Lambda_{M_2}(z_{2Q}; \alpha_Q) e^{-\alpha_P r_{1P}^2 - \alpha_Q r_{2Q}^2}$$

The above formula dramatically reduces to the following

$$[N_1 L_2 M_2 | N_2 L_2 M_2] = \lambda (-)^{N_2+L_2+M_2} R_{N_1+N_2, L_1+L_2, M_1+M_2}$$

with

$$\lambda \equiv \frac{2\pi^{5/2}}{\alpha_P \alpha_Q \sqrt{\alpha_P + \alpha_Q}}$$

To compute the $R_{N_1+N_2, L_1+L_2, M_1+M_2}$ coefficients, the parameter T is given by

$$T = \frac{\alpha_P \alpha_Q}{\alpha_P + \alpha_Q} |\mathbf{P} - \mathbf{Q}|^2$$

12.4.5 The R(N,L,M) Coefficients

The R coefficients are defined as

$$R_{NLM} \equiv \left(\frac{\partial}{\partial a} \right)^N \left(\frac{\partial}{\partial b} \right)^L \left(\frac{\partial}{\partial c} \right)^M \int_0^1 e^{-Tu^2} du$$

with

$$T \equiv \alpha (a^2 + b^2 + c^2)$$

By extending the above definition to more general

$$R_{NLMj} \equiv (-\sqrt{\alpha})^{N+L+M} (-2\alpha)^j \int_0^1 u^{N+L+M+2j} H_N(au\sqrt{\alpha}) H_L(bu\sqrt{\alpha}) H_M(cu\sqrt{\alpha}) e^{-Tu^2} du$$

one can see that

$$R_{000j} = (-2\alpha)^j F_j(T)$$

The Boys function is here given by

$$F_j(T) \equiv \int_0^1 u^{2j} e^{-Tu^2} du$$

and its efficient implementation can be discussed elsewhere. In Psi4, `psi::Taylor.Fjt` class is used for this purpose.

Now, it is possible to show that the following recursion relationships are true:

$$\begin{aligned} R_{0,0,M+1,j} &= cR_{0,0,M,j+1} + MR_{0,0,M-1,j+1} \\ R_{0,L+1,M,j} &= bR_{0,L,M,j+1} + LR_{0,L-1,M,j+1} \\ R_{N+1,L,M,j} &= aR_{N,L,M,j+1} + NR_{N-1,L,M,j+1} \end{aligned}$$

This scheme is implemented in EOPDev.

12.4.6 Function Documentation

d_N_n1_n2()

```
double oepdev::d_N_n1_n2 (
    int N,
    int n1,
    int n2,
    double PA,
    double PB,
    double aP )
```

Parameters

<i>N</i>	- increment in the summation of MDH series
<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>PA</i>	- cartesian component of P-A distance
<i>PB</i>	- cartesian component of P-B distance
<i>aP</i>	- free parameter of MDH expansion

Returns

the McMurchie-Davidson-Hermite coefficient

make_mdh_D1_coeff()

```
void oepdev::make_mdh_D1_coeff (
    int n1,
    double aPd,
    double * buffer )
```

Parameters

<i>n1</i>	- angular momentum of first function
<i>aPd</i>	- parameter equal to 0.500/Pa where Pa is exponent
<i>buffer</i>	- the McMurchie-Davidson-Hermite 3-dimensional array (raveled to vector): <ul style="list-style-type: none"> • axis 0: dimension 3 (x, y or z Cartesian component) • axis 1: dimension n1+1 (0 to n1) • axis 2: dimension n1+1 (0 to n1)

See also

[D1_INDEX](#)

make_mdh_D2_coeff()

```
void oepdev::make_mdh_D2_coeff (
    int n1,
    int n2,
    double aPd,
    double * PA,
    double * PB,
    double * buffer )
```

Parameters

<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>aPd</i>	- parameter equal to 0.500/Pa where Pa is exponent
<i>PA</i>	- cartesian components of P-A distance
<i>PB</i>	- cartesian components of P-B distance
<i>buffer</i>	- the McMurchie-Davidson-Hermite 4-dimensional array (raveled to vector): <ul style="list-style-type: none"> • axis 0: dimension 3 (x, y or z Cartesian component) • axis 1: dimension n1+1 (0 to n1) • axis 2: dimension n2+1 (0 to n2) • axis 3: dimension n1+n2+1 (0 to n1+n2)

See also

[D2.INDEX](#)

make_mdh_D3_coeff()

```
void oepdev::make_mdh_D3_coeff (
    int n1,
    int n2,
    int n3,
    double aPd,
    double * PA,
    double * PB,
    double * PC,
    double * buffer )
```

Parameters

<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>n3</i>	- angular momentum of third function
<i>aPd</i>	- parameter equal to 0.500/Pa where Pa is exponent
<i>PA</i>	- cartesian components of P-A distance
<i>PB</i>	- cartesian components of P-B distance
<i>PC</i>	- cartesian components of P-C distance
<i>buffer</i>	- the McMurchie-Davidson-Hermite 5-dimensional array (raveled to vector): <ul style="list-style-type: none"> • axis 0: dimension 3 (x, y or z Cartesian component) • axis 1: dimension n1+1 (0 to n1) • axis 2: dimension n2+1 (0 to n2) • axis 3: dimension n3+1 (0 to n3) • axis 4: dimension n1+n2+n3+1 (0 to n1+n2+n3)

See also

[D3.INDEX](#)

make_mdh_D2_coeff_explicit_recursion()

```
void oepdev::make_mdh_D2_coeff_explicit_recursion (
```

```

int n1,
int n2,
double aP,
double * PA,
double * PB,
double * buffer )

```

Parameters

<i>n1</i>	- angular momentum of first function
<i>n2</i>	- angular momentum of second function
<i>aPd</i>	- parameter equal to 0.500/Pa where Pa is exponent
<i>PA</i>	- cartesian components of P-A distance
<i>PB</i>	- cartesian components of P-B distance
<i>buffer</i>	- the McMurchie-Davidson-Hermite 4-dimensional array (raveled to vector): <ul style="list-style-type: none"> • axis 0: dimension 3 (x, y or z Cartesian component) • axis 1: dimension n1+1 (0 to n1) • axis 2: dimension n2+1 (0 to n2) • axis 3: dimension n1+n2+1 (0 to n1+n2)

See also

[D2.INDEX](#)

make_mdh_R_coeff()

```

void oepdev::make_mdh_R_coeff (
    int N,
    int L,
    int M,
    double alpha,
    double a,
    double b,
    double c,
    double * F,
    double * buffer )

```

Parameters

<i>N</i>	- increment in the summation of MDH series along x direction
<i>L</i>	- increment in the summation of MDH series along y direction
<i>M</i>	- increment in the summation of MDH series along z direction

Parameters

<i>alpha</i>	- alpha parameter of R coefficient
<i>a</i>	- x component of PQ vector of R coefficient
<i>b</i>	- y component of PQ vector of R coefficient
<i>c</i>	- z component of PQ vector of R coefficient
<i>F</i>	- array of Boys function values for given alpha and PQ
<i>buffer</i>	- the McMurchie-Davidson 4-dimensional array (raveled to vector): <ul style="list-style-type: none">• axis 0: dimension N+1• axis 1: dimension L+1• axis 2: dimension M+1• axis 3: dimension N+L+M+1 (<i>j</i>-th element)

12.5 The Three-Dimensional Vector Fields Library

Handles all sorts of scalar distributions in 3D Euclidean space, such as general vector potentials defined at particular collection of points. In this Module, you will also find handling both random and ordered points collections in a form of a G09 cube, as well as handling G09 Cube files. You will also find solvers used to fit the generalized multipole moments of a generalized density distribution, such as the electrostatic potential (ESP) fitting method. Located at `oeplib3d`.

Classes

- class `oeplib3d::MultipoleConvergence`
Multipole Convergence.
- class `oeplib3d::DMTPole`
Distributed Multipole Analysis Container and Computer. Abstract Base.
- class `oeplib3d::CAMM`
Cumulative Atomic Multipole Moments.
- class `oeplib3d::ESPSolver`
Charges from Electrostatic Potential (ESP). A solver-type class.
- class `oeplib3d::Points3DIterator`
Iterator over a collection of points in 3D space. Abstract base.
- class `oeplib3d::CubePoints3DIterator`
Iterator over a collection of points in 3D space. g09 Cube-like order.
- class `oeplib3d::RandomPoints3DIterator`
Iterator over a collection of points in 3D space. Random collection.
- class `oeplib3d::PointsCollection3D`
Collection of points in 3D space. Abstract base.
- class `oeplib3d::RandomPointsCollection3D`
Collection of random points in 3D space.
- class `oeplib3d::CubePointsCollection3D`
G09 cube-like ordered collection of points in 3D space.
- class `oeplib3d::Field3D`
General Vector Field in 3D Space. Abstract base.
- class `oeplib3d::ElectrostaticPotential3D`
Electrostatic potential of a molecule.
- class `oeplib3d::OEPotential3D< T >`
Class template for OEP 3D fields.

Typedefs

- using `oeplib3d::SharedDMTPole` = `std::shared_ptr< DMTPole >`
DMTPole object.
- using `oeplib3d::SharedField3D` = `std::shared_ptr< oeplib3d::Field3D >`

Functions

- `oepdev::OEPotential3D< T >::OEPotential3D` (const int &ndim, const int &np, const double &padding, std::shared_ptr< T > oep, const std::string &oepType)
Construct random spherical collection of 3D field of type T.
- `oepdev::OEPotential3D< T >::OEPotential3D` (const int &ndim, const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, std::shared_ptr< T > oep, const std::string &oepType, psi::Options &options)
Construct ordered 3D collection of 3D field of type T.
- virtual `oepdev::OEPotential3D< T >::~~OEPotential3D` ()
Destructor.
- virtual void `oepdev::OEPotential3D< T >::print` () const
Print information of the object to Psi4 output.
- virtual std::shared_ptr< psi::Vector > `oepdev::OEPotential3D< T >::compute_xyz` (const double &x, const double &y, const double &z)
Compute a value of 3D field at point (x, y, z)

12.5.1 Detailed Description

12.5.2 Function Documentation

OEPotential3D() [1/2]

```
template<class T >
oepdev::OEPotential3D< T >::OEPotential3D (
    const int & ndim,
    const int & np,
    const double & padding,
    std::shared_ptr< T > oep,
    const std::string & oepType )
```

The points are drawn according to uniform distribution in 3D space.

Parameters

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<i>np</i>	- number of points to draw
<i>padding</i>	- spherical padding distance (au)
<i>oep</i>	- OEP object of type T
<i>oepType</i>	- type of OEP

OEPotential3D() [2/2]

```

template<class T >
oepdev::OEPotential3D< T >::OEPotential3D (
    const int & ndim,
    const int & nx,
    const int & ny,
    const int & nz,
    const double & px,
    const double & py,
    const double & pz,
    std::shared_ptr< T > oep,
    const std::string & oepType,
    psi::Options & options )

```

The points are generated according to Gaussian cube file format.

Parameters

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>px</i>	- padding distance along x direction
<i>py</i>	- padding distance along y direction
<i>pz</i>	- padding distance along z direction
<i>oep</i>	- OEP object of type T
<i>oepType</i>	- type of OEP
<i>options</i>	- Psi4 options object

12.6 The Density Functional Theory Library

Implements the EOPDev ab initio DFT methods. Located at `oepdev/libdft`. Currently, this library is empty.

12.7 The EOPDev Utilities

Contains utility functions such as printing EOPDev preamble to the output file, class for wavefunction union, DIIS converger, CPHF Solver, SCF solver for external electrostatic perturbations, and others. You will also find here various iterators to go through orbital shells while computing ERI, or iterators over ERI itself. Located at `oepdev/libutil`.

Classes

- struct `oepdev::CISData`
CIS wavefunction parameters. Container structure.
- class `oepdev::CISComputer`
CISComputer.
- class `oepdev::R_CISComputer`
- class `oepdev::U_CISComputer`
- class `oepdev::R_CISComputer_Explicit`
- class `oepdev::R_CISComputer_DL`
CIS Computer with RHF reference: Davidson-Liu Solver.
- class `oepdev::R_CISComputer_Direct`
- class `oepdev::U_CISComputer_Explicit`
- class `oepdev::U_CISComputer_DL`
CIS Computer with UHF reference: Davidson-Liu Solver.
- class `oepdev::CPHF`
CPHF solver class.
- class `oepdev::DavidsonLiu`
Davidson-Liu diagonalization method.
- class `oepdev::DIISManager`
DIIS manager.
- class `oepdev::GramSchmidt`
Gram-Schmidt orthogonalization method.
- class `oepdev::ShellCombinationsIterator`
Iterator for Shell Combinations. Abstract Base.
- class `oepdev::AOIntegralsIterator`
Iterator for AO Integrals. Abstract Base.
- class `oepdev::AllAOShellCombinationsIterator_4`
Loop over all possible ERI shells in a shell quartet.
- class `oepdev::AllAOShellCombinationsIterator_2`
Loop over all possible ERI shells in a shell doublet.
- class `oepdev::AllAOIntegralsIterator_4`
Loop over all possible ERI within a particular shell quartet.
- class `oepdev::AllAOIntegralsIterator_2`

- Loop over all possible ERI within a particular shell doublet.*
- class `oepdev::KabschSuperimposer`
Compute the Cartesian rotation matrix between two structures.
- struct `oepdev::QUAMBOData`
Container to store the QUAMBO data.
- class `oepdev::QUAMBO`
The Quasiatomic Minimal Basis Set Molecular Orbitals (QUAMBO)
- struct `oepdev::PerturbCharges`
Structure to hold perturbing charges.
- class `oepdev::RHFPerturbed`
RHF theory under electrostatic perturbation.
- struct `oepdev::ABCD`
Simple structure to hold the Fourier series expansion coefficients.
- struct `oepdev::Fourier5`
Simple structure to hold the Fourier series expansion coefficients for N=2.
- struct `oepdev::Fourier9`
Simple structure to hold the Fourier series expansion coefficients for N=4.
- class `oepdev::UnitaryOptimizer`
Find the optimum unitary matrix of quadratic matrix equation.
- class `oepdev::UnitaryOptimizer_4_2`
Find the optimum unitary matrix for quartic-quadratic matrix equation with trace.
- class `oepdev::UnitaryOptimizer_2`
Find the optimum unitary matrix for quadratic matrix equation with trace.
- class `oepdev::UnitaryOptimizer_2_1`
- class `oepdev::WavefunctionUnion`
Union of two Wavefunction objects.

Macros

- #define `OEPDEV_USE_PSI4_DIIS_MANAGER` 0
Use DIIS from Psi4 (1) or OEPDev (0)?
- #define `OEPDEV_MAX_AM` 8
L_max.
- #define `OEPDEV_N_MAX_AM` 17
2L_max+1
- #define `OEPDEV_CRIT_ERI` 1e-9
*ERI criterion for E12, E34, E123 and lambda*EXY coefficients.*
- #define `OEPDEV_SIZE_BUFFER_R` 250563
*Size of R buffer (OEPDEV_N_MAX_AM*OEPDEV_N_MAX_AM*OEPDEV_N_MAX_AM*OEPDEV_N_MAX_AM*3)*
- #define `OEPDEV_SIZE_BUFFER_D2` 3264

Size of D2 buffer (3(OEPDEV_MAX_AM+1)*(OEPDEV_MAX_AM+1)*OEPDEV_N_MAX_AM)*

- `#define OEPDEV_AU_KcalPerMole 627.509`

Energy converters.

- `#define OEPDEV_AU_CMRec 219474.63`
- `#define OEPDEV_AU_EV 27.21138`

Typedefs

- using `oepdev::SharedCPHF` = `std::shared_ptr< CPHF >`
CPHF object.
- using `oepdev::SharedShellsIterator` = `std::shared_ptr< ShellCombinationsIterator >`
Iterator over shells as shared pointer.
- using `oepdev::SharedAOIntsIterator` = `std::shared_ptr< AOIntegralsIterator >`
Iterator over AO integrals as shared pointer.
- using `oepdev::SharedQUAMBOData` = `std::shared_ptr< QUAMBOData >`
- using `oepdev::SharedQUAMBO` = `std::shared_ptr< QUAMBO >`
Shared QUAMBO object.
- using `oepdev::SharedWavefunctionUnion` = `std::shared_ptr< WavefunctionUnion >`
WavefunctionUnion.

Functions

- PSI_API void `oepdev::preamble` (void)
Print preamble for module OEPDEV.
- template<typename... Args>
`std::string oepdev::string_sprintf` (const char *format, Args... args)
Format string output. Example: std::string text = oepdev::string_sprintf("Test %3d, %13.5f", 5, -10.5425);
- PSI_API `std::shared_ptr< SuperFunctional >` `oepdev::create_superfunctional` (std::string name, Options &options)
Set up DFT functional.
- PSI_API `SharedBasisSet` `oepdev::create_basisset_by_copy` (SharedBasisSet basis_ref, SharedMolecule molecule_target)
Build BasisSet by Copy.
- PSI_API `SharedBasisSet` `oepdev::create_atom_basisset_by_copy` (SharedBasisSet basis_ref, SharedMolecule molecule_target, int idx_atom)
Build BasisSet by Copy for a Particular Atom.
- PSI_API `std::shared_ptr< Molecule >` `oepdev::extract_monomer` (std::shared_ptr< const Molecule > molecule_dimer, int id)
Extract molecule from dimer.
- PSI_API double `oepdev::compute_distance` (psi::SharedVector v1, psi::SharedVector v2)

Compute distance between two points in nD space.

- PSI_API std::shared_ptr< Wavefunction > [oepdev::solve_scf](#) (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< BasisSet > auxiliary, std::shared_ptr< BasisSet > guess, std::shared_ptr< SuperFunctional > functional, Options &options, std::shared_ptr< PSIO > psio, bool compute_mints=false)

Solve RHF-SCF equations for a given molecule in a given basis set.

- PSI_API std::shared_ptr< Wavefunction > [oepdev::solve_scf_sad](#) (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< BasisSet > auxiliary, std::vector< std::shared_ptr< BasisSet >> sad, std::vector< std::shared_ptr< BasisSet >> sad_fit, std::shared_ptr< SuperFunctional > functional, Options &options, std::shared_ptr< PSIO > psio, bool compute_mints=false)

Solve RHF-SCF equations for a given molecule in a given basis set.

- PSI_API double [oepdev::average_moment](#) (std::shared_ptr< psi::Vector > moment)

Compute the scalar magnitude of multipole moment.

- PSI_API std::vector< std::shared_ptr< psi::Matrix > > [oepdev::calculate_JK](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::Matrix > C)

Compute the Coulomb and exchange integral matrices in MO basis.

- PSI_API std::vector< std::shared_ptr< psi::Matrix > > [oepdev::calculate_JK_ints](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr)
- PSI_API std::vector< std::shared_ptr< psi::Matrix > > [oepdev::calculate_JK_r](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr, std::shared_ptr< psi::Matrix > Dij)

Compute the Coulomb and exchange integral matrices in MO basis.

- PSI_API std::vector< std::shared_ptr< psi::Matrix > > [oepdev::calculate_JK_rb](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr, std::shared_ptr< psi::Matrix > Dij)
- std::shared_ptr< psi::Matrix > [oepdev::calculate_DFI_Vel](#) (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::IntegralFactory > f_abab, std::shared_ptr< psi::Matrix > d.b)

Compute the Effective DFI Potential Matrix Due To Electrons.

- PSI_API std::shared_ptr< psi::Matrix > [oepdev::calculate_DFI_Vel_JK](#) (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::IntegralFactory > f_abab, std::shared_ptr< psi::Matrix > d.b)

Compute the Effective DFI Coulomb+Exchange Potential Matrix Due To Electrons.

- PSI_API std::shared_ptr< psi::Matrix > [oepdev::calculate_DFI_Vel_J](#) (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::Matrix > d.b)

Compute the Effective DFI Coulomb Potential Matrix Due To Electrons.

- PSI_API std::shared_ptr< psi::Matrix > [oepdev::calculate_OEP_basisopt_V](#) (const int &nt, std::shared_ptr< psi::IntegralFactory > f_pppt, std::shared_ptr< psi::Matrix > ca, std::shared_ptr< psi::Matrix > da)

Compute the 2-Electron Part of the Effective OEP Matrix for Auxiliary Basis Set Optimization.

- PSI_API double [oepdev::bs_optimize_projection](#) (std::shared_ptr< psi::Matrix > ti, std::shared_ptr< psi::MintsHelper > mints, std::shared_ptr< psi::BasisSet > bsf_m, std::shared_ptr< psi::BasisSet > bsf_i)

Compute the objective function value for auxiliary basis set optimization of OEPs.

Rotation of AO Space

12.7.1 Theory

The objective is to find the formulae for rotation matrices of the AO spaces as functions of the Cartesian 3 x 3 rotation matrices. It is obvious that p -type functions transform as a usual Cartesian vectors. However, higher angular momentum functions transform in a more complex way.

Problem

Define a vectorized AO space M of rank $r > 1$ that is constructed from unique tensor components of fully symmetric r -th rank AO tensor populated in standard order,

$$M_{\{ab\dots k\}} = M_{ab\dots k} \quad \text{for } a \leq b \leq \dots \leq k$$

Given a general rotation of Cartesian tensors

$$M_{ab\dots k} = \sum_{a'b'\dots k'} M_{a'b'\dots k'} r_{a'a} r_{b'b} \dots r_{k'k}$$

find closed expressions for the rotation matrix in reduced composite AO space obeying

$$M_{[ab\dots k]} = \sum_{\{a'b'\dots k'\}} M_{\{a'b'\dots k'\}} R_{\{a'b'\dots k'\}, [ab\dots k]}$$

In the derivations below the following identity of first-order partitioning will be of use:

$$\sum_{ab} M_{ab} \hat{s}_{ab} = \sum_{\{ab\}} M_{\{ab\}} (\hat{s}_{ab} + \Delta_{ab} \hat{s}_{ba})$$

where

$$\Delta_{ab} \equiv 1 - \delta_{ab}$$

and the operator s of rank r acts as follows

$$s_{a'b'\dots k'}^{ab\dots k} \equiv \hat{s}_{a'b'\dots k'} \underbrace{\mathbf{r} \otimes \mathbf{r} \otimes \dots \otimes \mathbf{r}}_r = r_{a'a} r_{b'b} \dots r_{k'k}$$

Rotation of 6D functions

The rotation of the full tensor AO space of rank 2 and dimensions (3,3) is given by

$$M_{ab} = \sum_{a'b'} M_{a'b'} r_{a'a} r_{b'b}$$

Applying the identity of first-order partitioning directly leads to the formula for a reduced 6D tensor rotation of rank 1 and dimension (6),

$$M_{[ab]} = \sum_{\{a'b'\}} M_{\{a'b'\}} R_{\{a'b'\}, [ab]}$$

where the 6 x 6 rotation matrix is given by

$$R_{\{a'b'\}, [ab]} = r_{a'a} r_{b'b} + \Delta_{a'b'} r_{b'a} r_{a'b}$$

Rotation of 10F functions

The rotation of the full tensor AO space of rank 3 and dimensions (3,3,3) is given by

$$M_{abc} = \sum_{a'b'c'} M_{a'b'c'} r_{a'b} r_{b'b} r_{c'c}$$

First of all, notice that one can perform the following partitioning

$$\sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} = \sum_{\{abc\}} M_{\{abc\}} (\hat{s}_{abc} + \hat{s}_{acb} + \hat{s}_{bac} + \hat{s}_{bca} + \hat{s}_{cab} + \hat{s}_{cba})$$

Then, perform a partitioning of the triple sum,

$$\begin{aligned} \sum_{abc} M_{abc} \hat{s}_{abc} &= \sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} \\ &+ \sum_a \sum_{b \geq a} M_{abb} \hat{s}_{abb} + \sum_a \sum_{b < a} M_{abb} \hat{s}_{abb} \\ &+ \sum_a \sum_{b > a} M_{aba} \hat{s}_{aba} + \sum_a \sum_{b < a} M_{aba} \hat{s}_{aba} \\ &+ \sum_a \sum_{b > a} M_{bba} \hat{s}_{bba} + \sum_a \sum_{b < a} M_{bba} \hat{s}_{bba} \end{aligned}$$

Using the first-order partitioning theorem and interchanging the dummy indices one finds that

$$M_{[abc]} = \sum_{\{a'b'c'\}} M_{\{a'b'c'\}} R_{\{a'b'c'\},[abc]}$$

where the 10 x 10 rotation matrix is given by

$$\begin{aligned} R_{\{a'b'c'\},[abc]} &= \delta_{b'c'} \left(s_{a'b'b'}^{abc} + \Delta_{a'b'} \left\{ s_{b'a'b'}^{abc} + s_{b'b'a'}^{abc} \right\} \right) \\ &+ \delta_{a'b'} \Delta_{b'c'} \left(s_{c'a'a'}^{abc} + s_{a'c'a'}^{abc} + s_{a'a'c'}^{abc} \right) \\ &+ \Delta_{a'b'} \Delta_{b'c'} \left(s_{a'b'c}^{abc} + s_{a'c'b'}^{abc} + s_{b'a'c'}^{abc} + s_{b'c'a'}^{abc} + s_{c'a'b'}^{abc} + s_{c'b'a'}^{abc} \right) \end{aligned}$$

and

$$s_{a'b'c'}^{abc} \equiv \hat{s}_{a'b'c'} \mathbf{r} \otimes \mathbf{r} \otimes \mathbf{r} = r_{a'a} r_{b'b} r_{c'c}$$

- `psi::SharedMatrix oepdev::r6 (psi::SharedMatrix r)`
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `void oepdev::populate (double **R, double **r, std::vector< int > idx_am, const int &nam)`
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `psi::SharedMatrix oepdev::ao_rotation_matrix (psi::SharedMatrix r, psi::SharedBasisSet b)`
Compute the full rotation matrix of AO orbital space.

12.7.2 Detailed Description

12.7.3 Function Documentation

r6()

```
psi::SharedMatrix oepdev::r6 (
    psi::SharedMatrix r )
```

Compute the 10 x 10 rotation matrix of the 10F orbitals.

Parameters

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

Returns

6 x 6 rotation matrix of the 6D orbitals

Parameters

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

Returns

10 x 10 rotation matrix of the 10F orbitals

populate()

```
void oepdev::populate (
    double ** R,
    double ** r,
    std::vector< int > idx_am,
    const int & nam )
```

Compute the 10 x 10 rotation matrix of the 10F orbitals.

Parameters

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

Returns

6 x 6 rotation matrix of the 6D orbitals

Parameters

<i>r</i>	- Cartesian 3 x 3 rotation matrix
----------	-----------------------------------

Returns

10 x 10 rotation matrix of the 10F orbitals

ao_rotation_matrix()

```
psi::SharedMatrix oepdev::ao_rotation_matrix (
    psi::SharedMatrix r,
    psi::SharedBasisSet b )
```

Parameters

<i>r</i>	- Cartesian 3 x 3 rotation matrix
<i>b</i>	- Basis set

create_superfunctional()

```
PSI_API std::shared_ptr< SuperFunctional > oepdev::create_superfunctional (
    std::string name,
    Options & options )
```

Now it accepts only pure HF functional.

Parameters

<i>name</i>	name of the functional ("HF" is now only available)
<i>options</i>	psi::Options object

Returns

psi::SharedSuperFunctional object with functional.

Examples:

[example_scf_perturb.cc](#).

create_basisset_by_copy()

```
PSI_API SharedBasisSet oepdev::create_basisset_by_copy (
    SharedBasisSet basis_ref,
    SharedMolecule molecule_target )
```

Parameters

<i>basis_ref</i>	- reference basis set
<i>molecule_target</i>	- target molecule

Returns

psi::SharedBasisSet object.

create_atom_basisset_by_copy()

```
PSI_API SharedBasisSet oepdev::create_atom_basisset_by_copy (
    SharedBasisSet basis_ref,
    SharedMolecule molecule_target,
    int idx_atom )
```

Parameters

<i>basis_ref</i>	- reference basis set
<i>molecule_target</i>	- target molecule (atom in this case)
<i>idx_atom</i>	- index of an atom in <i>basis_ref</i> -> <i>molecule</i> ()

Returns

psi::SharedBasisSet object.

extract_monomer()

```
PSI_API std::shared_ptr< Molecule > oepdev::extract_monomer (
    std::shared_ptr< const Molecule > molecule_dimer,
    int id )
```

Parameters

<i>molecule_dimer</i>	psi::SharedMolecule object with dimer
<i>id</i>	index of a molecule (starts from 1)

Returns

psi::SharedMolecule object with indicated monomer

compute_distance()

```
PSI_API double oepdev::compute_distance (
    psi::SharedVector v1,
    psi::SharedVector v2 )
```

Parameters

<i>v1</i>	- vector 1
<i>v2</i>	- vector 2

Returns

distance The vectors have to have the same length.

solve_scf()

```
PSI_API std::shared_ptr< Wavefunction > oepdev::solve_scf (
    std::shared_ptr< Molecule > molecule,
    std::shared_ptr< BasisSet > primary,
    std::shared_ptr< BasisSet > auxiliary,
    std::shared_ptr< BasisSet > guess,
    std::shared_ptr< SuperFunctional > functional,
    Options & options,
    std::shared_ptr< PSIO > psio,
    bool compute_mints = false )
```

Parameters

<i>molecule</i>	psi::SharedMolecule object with molecule
<i>primary</i>	basis set
<i>auxiliary</i>	basis set
<i>guess</i>	basis set
<i>functional</i>	DFT functional
<i>options</i>	psi::Options object
<i>psio</i>	psi::PSIO object
<i>compute_mints</i>	Compute integrals (write IWL TOC entry - necessary when transforming integrals)

Returns

psi::SharedWavefunction SCF wavefunction of the molecule

solve_scf_sad()

```
PSI_API std::shared_ptr< Wavefunction > oepdev::solve_scf_sad (
    std::shared_ptr< Molecule > molecule,
    std::shared_ptr< BasisSet > primary,
    std::shared_ptr< BasisSet > auxiliary,
    std::vector< std::shared_ptr< BasisSet >> sad,
    std::vector< std::shared_ptr< BasisSet >> sad_fit,
    std::shared_ptr< SuperFunctional > functional,
    Options & options,
    std::shared_ptr< PSIO > psio,
    bool compute_mints = false )
```

Parameters

<i>molecule</i>	psi::SharedMolecule object with molecule
<i>primary</i>	shared primary basis set
<i>auxiliary</i>	shared auxiliary basis set
<i>sad</i>	SAD basis set list
<i>sad_fit</i>	SAD DF fitting basis set list
<i>functional</i>	DFT functional
<i>options</i>	psi::Options object
<i>psio</i>	psi::PSIO object
<i>compute_mints</i>	Compute integrals (write IWL TOC entry - necessary when transforming integrals)

Returns

psi::SharedWavefunction SCF wavefunction of the molecule

average_moment()

```
PSI_API double oepdev::average_moment (
    std::shared_ptr< psi::Vector > moment )
```

Parameters

<i>moment</i>	- multipole moment vector with unique matrix elements. Now supported only for dipole and quadrupole.
---------------	--

Returns

- the average multipole moment value.

The magnitudes of multipole moments are defined here as follows:

- The dipole moment magnitude is just a norm

$$|\mu| \equiv \sqrt{\mu_x^2 + \mu_y^2 + \mu_z^2}$$

- The quadrupole moment magnitude refers to the traceless moment in Buckingham convention

$$|\Theta| \equiv \sqrt{\Theta_{zz}^2 + \frac{1}{3}(\Theta_{xx} - \Theta_{yy})^2 + \frac{4}{3}(\Theta_{xy}^2 + \Theta_{xz}^2 + \Theta_{yz}^2)}$$

In the above equation, the quadrupole moment elements refer to its traceless form.

calculate_JK()

```
PSI_API std::vector< std::shared_ptr< psi::Matrix > > oepdev::calculate_JK
(
    std::shared_ptr< psi::Wavefunction > wfn,
    std::shared_ptr< psi::Matrix > C )
```

Transforms the AO ERI's based on provided C matrix.

Parameters

<i>wfn</i>	- Wavefunction object
<i>C</i>	- molecular orbital coefficients (AO x MO)

Returns

- vector with J_{ij} and K_{ij} matrix

calculate_JK_r()

```
PSI_API std::vector< std::shared_ptr< psi::Matrix > > oepdev::calculate_JK_r
(
    std::shared_ptr< psi::Wavefunction > wfn,
    std::shared_ptr< psi::IntegralTransform > tr,
    std::shared_ptr< psi::Matrix > Dij )
```

Reads the existing MO ERI's.

Parameters

<i>wfn</i>	- Wavefunction object
<i>tr</i>	- IntegralTransform object
<i>D</i>	- density matrix in MO basis

Returns

- vector with J_{ij} and K_{ij} matrix

`_calculate_DFI_Vel()`

```
std::shared_ptr< psi::Matrix > oepdev::_calculate_DFI_Vel (
    std::shared_ptr< psi::IntegralFactory > f_aabb,
    std::shared_ptr< psi::IntegralFactory > f_abab,
    std::shared_ptr< psi::Matrix > d_b )
```

Potential is felt by molecule A and induced by electrons in molecule B.

Parameters

<i>f_aabb</i>	- IntegralFactory of type (AA BB)
<i>f_abab</i>	- IntegralFactory of type (AB AB)
<i>d_b</i>	- one-particle density matrix in AO basis of B

Returns

- $V_{el}(B)$ matrix in AO basis set of A

If *f_abab* is nullptr, then only Coulomb matrix is computed. Otherwise, also exchange contribution is computed.

`calculate_DFI_Vel_JK()`

```
PSI_API std::shared_ptr< psi::Matrix > oepdev::calculate_DFI_Vel_JK (
    std::shared_ptr< psi::IntegralFactory > f_aabb,
    std::shared_ptr< psi::IntegralFactory > f_abab,
    std::shared_ptr< psi::Matrix > d_b )
```

Potential is felt by molecule A and induced by electrons in molecule B.

Parameters

<i>f_aabb</i>	- IntegralFactory of type (AA BB)
<i>f_abab</i>	- IntegralFactory of type (AB AB)
<i>d_b</i>	- one-particle density matrix in AO basis of B

Returns

- $V_{el}(B)$ matrix in AO basis set of A

calculate_DFI_Vel_J()

```
PSI_API std::shared_ptr< psi::Matrix > oepdev::calculate_DFI_Vel_J (
    std::shared_ptr< psi::IntegralFactory > f_aabb,
    std::shared_ptr< psi::Matrix > d_b )
```

Potential is felt by molecule A and induced by electrons in molecule B.

Parameters

<i>f_aabb</i>	- IntegralFactory of type (AA BB)
<i>d_b</i>	- one-particle density matrix in AO basis of B

Returns

- $V_{el}(B)$ matrix in AO basis set of A

calculate_OEP_basisopt_V()

```
PSI_API std::shared_ptr< psi::Matrix > oepdev::calculate_OEP_basisopt_V (
    const int & nt,
    std::shared_ptr< psi::IntegralFactory > f_pppt,
    std::shared_ptr< psi::Matrix > ca,
    std::shared_ptr< psi::Matrix > da )
```

Parameters

<i>nt</i>	- number of test basis functions
<i>f_pppt</i>	- IntegralFactory of type (PP PT)
<i>ca</i>	- target MOs
<i>da</i>	- one-particle density matrix in AO basis

Returns

- V matrix

bs_optimize_projection()

```
PSI_API double oepdev::bs_optimize_projection (
    std::shared_ptr< psi::Matrix > ti,
    std::shared_ptr< psi::MintsHelper > mints,
    std::shared_ptr< psi::BasisSet > bsf_m,
    std::shared_ptr< psi::BasisSet > bsf_i )
```

Parameters

<i>ti</i>	- Ti matrix
<i>mints</i>	- integral helper (instantiated with bsf_i)
<i>bsf_m</i>	- auxiliary AO basis to optimize
<i>bsf_i</i>	- intermediate AO basis

Returns

value of objective function equal to negative trace of overlap matrix

12.8 The EOPDev Testing Platform Library

Testing platform at C++ level of code. You should add more tests here when developing new functionalities, theories or models. Located at `oepdev/libtest`.

Classes

- class `oepdev::test::Test`
Manages test routines.

12.8.1 Detailed Description

Namespace Documentation

13.1 oepdev Namespace Reference

OEPEDev module namespace.

Classes

- struct [ABCD](#)
Simple structure to hold the Fourier series expansion coefficients.
- class [AbInitioPolarGEFactory](#)
Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.
- class [AllAOIntegralsIterator_2](#)
Loop over all possible ERI within a particular shell doublet.
- class [AllAOIntegralsIterator_4](#)
Loop over all possible ERI within a particular shell quartet.
- class [AllAOShellCombinationsIterator_2](#)
Loop over all possible ERI shells in a shell doublet.
- class [AllAOShellCombinationsIterator_4](#)
Loop over all possible ERI shells in a shell quartet.
- class [AOIntegralsIterator](#)
Iterator for AO Integrals. Abstract Base.
- class [CAMM](#)
Cumulative Atomic Multipole Moments.
- class [ChargeTransferEnergyOEPotential](#)
Generalized One-Electron Potential for Charge-Transfer Interaction Energy.

- class [ChargeTransferEnergySolver](#)
Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.
- class [CISComputer](#)
CISComputer.
- struct [CISData](#)
CIS wavefunction parameters. Container structure.
- class [CPHF](#)
CPHF solver class.
- class [CubePoints3DIterator](#)
Iterator over a collection of points in 3D space. g09 Cube-like order.
- class [CubePointsCollection3D](#)
G09 cube-like ordered collection of points in 3D space.
- class [DavidsonLiu](#)
Davidson-Liu diagonalization method.
- class [DIISManager](#)
DIIS manager.
- class [DMTPole](#)
Distributed Multipole Analysis Container and Computer. Abstract Base.
- class [DoubleGeneralizedDensityFit](#)
Generalized Density Fitting Scheme - Double Fit.
- class [EETCouplingOEPotential](#)
Generalized One-Electron Potential for EET coupling calculations.
- class [EETCouplingSolver](#)
Compute the EET coupling energy between unperturbed wavefunctions.
- class [EFP2_GEFactory](#)
EFP2 GEFP Factory.
- class [EFPMultipolePotentialInt](#)
Computes potential integrals.
- class [ElectrostaticEnergyOEPotential](#)
Generalized One-Electron Potential for Electrostatic Energy.
- class [ElectrostaticEnergySolver](#)
Compute the Coulombic interaction energy between unperturbed wavefunctions.
- class [ElectrostaticPotential3D](#)
Electrostatic potential of a molecule.
- class [ERI_1_1](#)
2-centre ERI of the form $(a|O(2)|b)$ where $O(2) = 1/r_{12}$.
- class [ERI_2_2](#)
4-centre ERI of the form $(ab|O(2)|cd)$ where $O(2) = 1/r_{12}$.
- class [ERI_3_1](#)
4-centre ERI of the form $(abc|O(2)|d)$ where $O(2) = 1/r_{12}$.

- class [ESPSolver](#)
Charges from Electrostatic Potential (ESP). A solver-type class.
- class [FFAbInitioPolarGEFactory](#)
Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.
- class [Field3D](#)
General Vector Field in 3D Space. Abstract base.
- struct [Fourier5](#)
Simple structure to hold the Fourier series expansion coefficients for N=2.
- struct [Fourier9](#)
Simple structure to hold the Fourier series expansion coefficients for N=4.
- class [FragmentedSystem](#)
Molecular System for Fragment-Based Calculations.
- class [GenEffFrag](#)
Generalized Effective Fragment. Container Class.
- class [GenEffPar](#)
Generalized Effective Fragment Parameters. Container Class.
- class [GenEffParFactory](#)
Generalized Effective Fragment Factory. Abstract Base.
- class [GeneralizedDensityFit](#)
Generalized Density Fitting Scheme. Abstract Base.
- class [GeneralizedPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
- class [GramSchmidt](#)
Gram-Schmidt orthogonalization method.
- class [IntegralFactory](#)
Extended [IntegralFactory](#) for computing integrals.
- class [KabschSuperimposer](#)
Compute the Cartesian rotation matrix between two structures.
- class [LinearGradientNonUniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
- class [LinearNonUniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
- class [LinearUniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
- class [MultipoleConvergence](#)
Multipole Convergence.
- class [NonUniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
- class [ObaraSaikaTwoCenterEFPRecursion_New](#)

- Obara-Saika recursion formulae for improved EFP multipole potential integrals.*
- class [OEP_EFP2_GEFactory](#)
OEP-EFP2 GEFP Factory.
 - class [OEPDevSolver](#)
Solver of properties of molecular aggregates. Abstract base.
 - class [OEPotential](#)
Generalized One-Electron Potential: Abstract base.
 - class [OEPotential3D](#)
Class template for OEP 3D fields.
 - struct [OEPTYPE](#)
Container to handle the type of One-Electron Potentials.
 - class [OverlapGeneralizedDensityFit](#)
Generalized Density Fitting Scheme - Single Fit Based on Minimal Overlap in MO Basis.
 - struct [PerturbCharges](#)
Structure to hold perturbing charges.
 - class [Points3DIterator](#)
Iterator over a collection of points in 3D space. Abstract base.
 - class [PointsCollection3D](#)
Collection of points in 3D space. Abstract base.
 - class [PolarGEFactory](#)
Polarization GEFP Factory. Abstract Base.
 - class [PotentialInt](#)
Computes potential integrals.
 - class [QuadraticGradientNonUniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
 - class [QuadraticNonUniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
 - class [QuadraticUniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
 - class [QUAMBO](#)
The Quasiatomic Minimal Basis Set Molecular Orbitals ([QUAMBO](#))
 - struct [QUAMBOData](#)
Container to store the [QUAMBO](#) data.
 - class [R_CISComputer](#)
 - class [R_CISComputer_Direct](#)
 - class [R_CISComputer_DL](#)
CIS Computer with RHF reference: Davidson-Liu Solver.
 - class [R_CISComputer_Explicit](#)
 - class [RandomPoints3DIterator](#)
Iterator over a collection of points in 3D space. Random collection.

- class [RandomPointsCollection3D](#)
Collection of random points in 3D space.
- class [RepulsionEnergyOEPotential](#)
Generalized One-Electron Potential for Pauli Repulsion Energy.
- class [RepulsionEnergySolver](#)
Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.
- class [RHPerturbed](#)
RHF theory under electrostatic perturbation.
- class [ShellCombinationsIterator](#)
Iterator for Shell Combinations. Abstract Base.
- class [SingleGeneralizedDensityFit](#)
Generalized Density Fitting Scheme - Single Fit.
- class [TIData](#)
Transfer Integral EET Data.
- class [TwoBodyAOInt](#)
- class [TwoElectronInt](#)
General Two Electron Integral.
- class [U_CISComputer](#)
- class [U_CISComputer_DL](#)
CIS Computer with UHF reference: Davidson-Liu Solver.
- class [U_CISComputer_Explicit](#)
- class [UniformEFieldPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Parameterization.
- class [UnitaryOptimizer](#)
Find the optimum unitary matrix of quadratic matrix equation.
- class [UnitaryOptimizer_2](#)
Find the optimum unitary matrix for quadratic matrix equation with trace.
- class [UnitaryOptimizer_2_1](#)
- class [UnitaryOptimizer_4_2](#)
Find the optimum unitary matrix for quartic-quadratic matrix equation with trace.
- class [UnitaryTransformedMOPolarGEFactory](#)
Polarization GEFP Factory with Least-Squares Scaling of MO Space.
- class [WavefunctionUnion](#)
Union of two Wavefunction objects.

Typedefs

- using [SharedDMTPole](#) = std::shared_ptr< [DMTPole](#) >
DMTPole object.
- using **SharedField3D** = std::shared_ptr< [oepdev::Field3D](#) >

- using **SharedOEPotential** = std::shared_ptr< [OEPotential](#) >
- using [SharedGenEffPar](#) = std::shared_ptr< [GenEffPar](#) >
GEFP Parameters container.
- using [SharedGenEffParFactory](#) = std::shared_ptr< [GenEffParFactory](#) >
GEFP Parameter factory.
- using [SharedGenEffFrag](#) = std::shared_ptr< [GenEffFrag](#) >
GEFP Fragment container.
- using [SharedFragmentedSystem](#) = std::shared_ptr< [FragmentedSystem](#) >
Fragmented system.
- using **SharedWavefunction** = std::shared_ptr< [Wavefunction](#) >
- using **SharedBasisSet** = std::shared_ptr< [BasisSet](#) >
- using **SharedMatrix** = std::shared_ptr< [Matrix](#) >
- using **SharedVector** = std::shared_ptr< [Vector](#) >
- using **SharedLocalizer** = std::shared_ptr< [Localizer](#) >
- using **SharedCISData** = std::shared_ptr< [CISData](#) >
- using [SharedWavefunctionUnion](#) = std::shared_ptr< [WavefunctionUnion](#) >
WavefunctionUnion.
- using **SharedDMTPConvergence** = std::shared_ptr< [oepdev::MultipoleConvergence](#) >
- using **SharedMolecule** = std::shared_ptr< [psi::Molecule](#) >
- using **SharedMOSpace** = std::shared_ptr< [psi::MOSpace](#) >
- using **SharedMOSpaceVector** = std::vector< std::shared_ptr< [psi::MOSpace](#) > >
- using **SharedIntegralTransform** = std::shared_ptr< [psi::IntegralTransform](#) >
- using [SharedCPHF](#) = std::shared_ptr< [CPHF](#) >
CPHF object.
- using **SharedIntegralFactory** = std::shared_ptr< [IntegralFactory](#) >
- using **SharedTwoBodyAOInt** = std::shared_ptr< [TwoBodyAOInt](#) >
- using [SharedShellsIterator](#) = std::shared_ptr< [ShellCombinationsIterator](#) >
Iterator over shells as shared pointer.
- using [SharedAOIntsIterator](#) = std::shared_ptr< [AOIntegralsIterator](#) >
Iterator over AO integrals as shared pointer.
- using **SharedQUAMBOData** = std::shared_ptr< [QUAMBOData](#) >
- using [SharedQUAMBO](#) = std::shared_ptr< [QUAMBO](#) >
Shared QUAMBO object.
- using **SharedSuperFunctional** = std::shared_ptr< [SuperFunctional](#) >

Functions

- double [d.N.n1.n2](#) (int N, int n1, int n2, double PA, double PB, double aP)
Compute McMurchie-Davidson-Hermite (MDH) coefficient for binomial expansion.
- void [make_mdh_D2_coeff_explicit_recursion](#) (int n1, int n2, double aP, double *PA, double *PB, double *buffer)

Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion by explicit recursion. This function makes the same changes to buffers as `oepdev::make_mdh_D2_coeff`, but implements it through explicit recursion by calls to `oepdev::d_N_n1_n2`. Therefore, it is slightly slower. Here for debugging purposes.

- void `make_mdh_D1_coeff` (int n1, double aPd, double *buffer)

Compute the McMurchie-Davidson-Hermite coefficients for monomial expansion.

- void `make_mdh_D2_coeff` (int n1, int n2, double aPd, double *PA, double *PB, double *buffer)

Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion.

- void `make_mdh_D3_coeff` (int n1, int n2, int n3, double aPd, double *PA, double *PB, double *PC, double *buffer)

Compute the McMurchie-Davidson-Hermite coefficients for trinomial expansion.

- void `make_mdh_R_coeff` (int N, int L, int M, double alpha, double a, double b, double c, double *F, double *buffer)

Compute the McMurchie-Davidson R coefficients.

- double *** `init_box` (int a, int b, int c)
- void `zero_box` (double ***box, int a, int b, int c)
- void `free_box` (double ***box, int a, int b)
- psi::SharedMatrix `r10` (psi::SharedMatrix r3)
- constexpr std::complex< double > `operator""_i` (unsigned long long d)
- constexpr std::complex< double > `operator""_i` (long double d)
- PSI.API void `preamble` (void)

Print preamble for module OEPDEV.

- PSI.API std::shared_ptr< SuperFunctional > `create_superfunctional` (std::string name, Options &options)

Set up DFT functional.

- PSI.API SharedBasisSet `create_basiset_by_copy` (SharedBasisSet basis_ref, SharedMolecule molecule_target)

Build BasisSet by Copy.

- PSI.API SharedBasisSet `create_atom_basiset_by_copy` (SharedBasisSet basis_ref, SharedMolecule molecule_target, int idx_atom)

Build BasisSet by Copy for a Particular Atom.

- PSI.API std::shared_ptr< Molecule > `extract_monomer` (std::shared_ptr< const Molecule > molecule_dimer, int id)

Extract molecule from dimer.

- PSI.API double `compute_distance` (psi::SharedVector v1, psi::SharedVector v2)

Compute distance between two points in nD space.

- PSI.API std::shared_ptr< Wavefunction > `solve_scf` (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< BasisSet > auxiliary, std::shared_ptr< BasisSet > guess, std::shared_ptr< SuperFunctional > functional, Options &options, std::shared_ptr< PSIO > psio, bool compute_mints=false)

Solve RHF-SCF equations for a given molecule in a given basis set.

- PSI.API std::shared_ptr< Wavefunction > [solve_scf_sad](#) (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< BasisSet > auxiliary, std::vector< std::shared_ptr< BasisSet >> sad, std::vector< std::shared_ptr< BasisSet >> sad_fit, std::shared_ptr< SuperFunctional > functional, Options &options, std::shared_ptr< PSIO > psio, bool compute_mints=false)
Solve RHF-SCF equations for a given molecule in a given basis set.
- PSI.API double [average_moment](#) (std::shared_ptr< psi::Vector > moment)
Compute the scalar magnitude of multipole moment.
- PSI.API std::vector< std::shared_ptr< psi::Matrix > > [calculate_JK](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::Matrix > C)
Compute the Coulomb and exchange integral matrices in MO basis.
- PSI.API std::vector< std::shared_ptr< psi::Matrix > > **calculate_JK_ints** (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr)
- PSI.API std::vector< std::shared_ptr< psi::Matrix > > [calculate_JK_r](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr, std::shared_ptr< psi::Matrix > Dij)
Compute the Coulomb and exchange integral matrices in MO basis.
- PSI.API std::vector< std::shared_ptr< psi::Matrix > > **calculate_JK_rb** (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr, std::shared_ptr< psi::Matrix > Dij)
- std::shared_ptr< psi::Matrix > [_calculate_DFI_Vel](#) (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::IntegralFactory > f_abab, std::shared_ptr< psi::Matrix > d.b)
Compute the Effective DFI Potential Matrix Due To Electrons.
- PSI.API std::shared_ptr< psi::Matrix > [calculate_DFI_Vel_JK](#) (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::IntegralFactory > f_abab, std::shared_ptr< psi::Matrix > d.b)
Compute the Effective DFI Coulomb+Exchange Potential Matrix Due To Electrons.
- PSI.API std::shared_ptr< psi::Matrix > [calculate_DFI_Vel_J](#) (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::Matrix > d.b)
Compute the Effective DFI Coulomb Potential Matrix Due To Electrons.
- PSI.API std::shared_ptr< psi::Matrix > [calculate_OEP_basisopt_V](#) (const int &nt, std::shared_ptr< psi::IntegralFactory > f_pppt, std::shared_ptr< psi::Matrix > ca, std::shared_ptr< psi::Matrix > da)
Compute the 2-Electron Part of the Effective OEP Matrix for Auxiliary Basis Set Optimization.
- PSI.API double [bs_optimize_projection](#) (std::shared_ptr< psi::Matrix > ti, std::shared_ptr< psi::MintsHelper > mints, std::shared_ptr< psi::BasisSet > bsf_m, std::shared_ptr< psi::BasisSet > bsf_i)
Compute the objective function value for auxiliary basis set optimization of OEPs.
- template<typename... Args>
std::string [string_sprintf](#) (const char *format, Args... args)
Format string output. Example: std::string text = oepdev::string_sprintf("Test %3d, %13.5f", 5, -10.5425);

Rotation of AO Space

13.1.1 Theory

The objective is to find the formulae for rotation matrices of the AO spaces as functions of the Cartesian 3 x 3 rotation matrices. It is obvious that p-type functions transform as a usual Cartesian vectors. However, higher angular momentum functions transform in a more complex way.

Problem

Define a vectorized AO space M of rank $r > 1$ that is constructed from unique tensor components of fully symmetric r -th rank AO tensor populated in standard order,

$$M_{\{ab\dots k\}} = M_{ab\dots k} \quad \text{for } a \leq b \leq \dots \leq k$$

Given a general rotation of Cartesian tensors

$$M_{ab\dots k} = \sum_{a'b'\dots k'} M_{a'b'\dots k'} r_{a'a} r_{b'b} \dots r_{k'k}$$

find closed expressions for the rotation matrix in reduced composite AO space obeying

$$M_{[ab\dots k]} = \sum_{\{a'b'\dots k'\}} M_{\{a'b'\dots k'\}} R_{\{a'b'\dots k'\}, [ab\dots k]}$$

In the derivations below the following identity of first-order partitioning will be of use:

$$\sum_{ab} M_{ab} \hat{s}_{ab} = \sum_{\{ab\}} M_{\{ab\}} (\hat{s}_{ab} + \Delta_{ab} \hat{s}_{ba})$$

where

$$\Delta_{ab} \equiv 1 - \delta_{ab}$$

and the operator s of rank r acts as follows

$$s_{a'b'\dots k'}^{ab\dots k} \equiv \hat{s}_{a'b'\dots k'} \underbrace{\mathbf{r} \otimes \mathbf{r} \otimes \dots \otimes \mathbf{r}}_r = r_{a'a} r_{b'b} \dots r_{k'k}$$

Rotation of 6D functions

The rotation of the full tensor AO space of rank 2 and dimensions (3,3) is given by

$$M_{ab} = \sum_{a'b'} M_{a'b'} r_{a'a} r_{b'b}$$

Applying the identity of first-order partitioning directly leads to the formula for a reduced 6D tensor rotation of rank 1 and dimension (6),

$$M_{[ab]} = \sum_{\{a'b'\}} M_{\{a'b'\}} R_{\{a'b'\}, [ab]}$$

where the 6 x 6 rotation matrix is given by

$$R_{\{a'b'\}, [ab]} = r_{a'a} r_{b'b} + \Delta_{a'b'} r_{b'a} r_{a'b}$$

Rotation of 10F functions

The rotation of the full tensor AO space of rank 3 and dimensions (3,3,3) is given by

$$M_{abc} = \sum_{a'b'c'} M_{a'b'c'} r_{a'b} r_{b'b} r_{c'c}$$

First of all, notice that one can perform the following partitioning

$$\sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} = \sum_{\{abc\}} M_{\{abc\}} (\hat{s}_{abc} + \hat{s}_{acb} + \hat{s}_{bac} + \hat{s}_{bca} + \hat{s}_{cab} + \hat{s}_{cba})$$

Then, perform a partitioning of the triple sum,

$$\begin{aligned} \sum_{abc} M_{abc} \hat{s}_{abc} &= \sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} \\ &+ \sum_a \sum_{b \geq a} M_{abb} \hat{s}_{abb} + \sum_a \sum_{b < a} M_{abb} \hat{s}_{abb} \\ &+ \sum_a \sum_{b > a} M_{aba} \hat{s}_{aba} + \sum_a \sum_{b < a} M_{aba} \hat{s}_{aba} \\ &+ \sum_a \sum_{b > a} M_{bba} \hat{s}_{bba} + \sum_a \sum_{b < a} M_{bba} \hat{s}_{bba} \end{aligned}$$

Using the first-order partitioning theorem and interchanging the dummy indices one finds that

$$M_{[abc]} = \sum_{\{a'b'c'\}} M_{\{a'b'c'\}} R_{\{a'b'c'\}, [abc]}$$

where the 10 x 10 rotation matrix is given by

$$\begin{aligned} R_{\{a'b'c'\}, [abc]} &= \delta_{b'c'} \left(s_{a'b'b'}^{abc} + \Delta_{a'b'} \left\{ s_{b'a'b'}^{abc} + s_{b'b'a'}^{abc} \right\} \right) \\ &+ \delta_{a'b'} \Delta_{b'c'} \left(s_{c'a'a'}^{abc} + s_{a'c'a'}^{abc} + s_{a'a'c'}^{abc} \right) \\ &+ \Delta_{a'b'} \Delta_{b'c'} \left(s_{a'b'b'c}^{abc} + s_{a'c'b'}^{abc} + s_{b'a'c'}^{abc} + s_{b'b'c'a'}^{abc} + s_{c'a'b'}^{abc} + s_{c'b'a'}^{abc} \right) \end{aligned}$$

and

$$s_{a'b'c'}^{abc} \equiv \hat{s}_{a'b'c'} \mathbf{r} \otimes \mathbf{r} \otimes \mathbf{r} = r_{a'a} r_{b'b} r_{c'c}$$

- `psi::SharedMatrix r6` (`psi::SharedMatrix r`)
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `void populate` (`double **R`, `double **r`, `std::vector< int > idx_am`, `const int &nam`)
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `psi::SharedMatrix ao_rotation_matrix` (`psi::SharedMatrix r`, `psi::SharedBasisSet b`)
Compute the full rotation matrix of AO orbital space.

Variables

- `double dfxxx` [`MAX_DF`]

13.1.2 Detailed Description

Contains all the functionalities for the development of the Generalized One-Electrode Potentials (OEP's).

13.2 psi Namespace Reference

Psi4 package namespace.

Typedefs

- using **SharedBasisSet** = std::shared_ptr< BasisSet >
- using **SharedMolecule** = std::shared_ptr< Molecule >
- using **SharedMatrix** = std::shared_ptr< Matrix >
- using **SharedWavefunction** = std::shared_ptr< Wavefunction >

Functions

- PSI_API int [read_options](#) (std::string name, Options &options)
Options for the OEPDev plugin.
- void **export_dmtp** (py::module &)
- void **export_cphf** (py::module &)
- void **export_solver** (py::module &)
- void **export_util** (py::module &)
- void **export_oep** (py::module &)
- void **export_gefp** (py::module &)
- PSI_API SharedWavefunction [oepdev](#) (SharedWavefunction ref_wfn, Options &options)
Main routine of the OEPDev plugin.
- **PYBIND11_MODULE** ([oepdev](#), m)

13.2.1 Detailed Description

Contains all Psi4 functionalities.

13.2.2 Function Documentation

read_options()

```
PSI_API int psi::read_options (
    std::string name,
    Options & options )
```

Parameters

<i>name</i>	name of driver function
<i>options</i>	psi::Options object

Returns

true

oepdev()

```
PSI_API SharedWavefunction psi::oepdev (
    SharedWavefunction ref_wfn,
    Options & options )
```

Created with intention to test various models of the interaction energy between two molecules, described by the Hartree-Fock-Roothaan-Hall theory or the configuration interaction with singles theory.

In particular, the plugin tests the models of:

1. the Pauli repulsion and CT interaction energy (Project II)
2. the Induction interaction energy (Project III)
3. the excitation energy transfer couplings (Project I)

against benchmarks (exact or reference solutions). The list of implemented models can be found in [Implemented Models](#) .

Parameters

<i>ref_wfn</i>	shared wavefunction of a dimer
<i>options</i>	psi::Options object

Returns

psi::SharedWavefunction (either ref_wfn or wavefunction union)

14.1 oepdev::ABCD Struct Reference

Simple structure to hold the Fourier series expansion coefficients.

```
#include <unitary_optimizer.h>
```

Public Attributes

- double **A**
- double **B**
- double **C**
- double **D**

14.1.1 Detailed Description

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

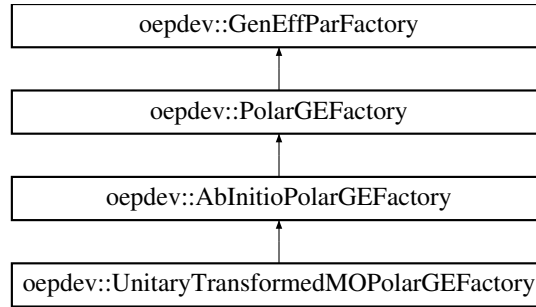
- oepdev/libutil/[unitary_optimizer.h](#)

14.2 oepdev::AbInitioPolarGEFactory Class Reference

Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::AbInitioPolarGEFactory:



Public Member Functions

- **AbInitioPolarGEFactory** (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- virtual std::shared_ptr< [GenEffPar](#) > [compute](#) (void)
Compute the density matrix susceptibility tensors.

Additional Inherited Members

14.2.1 Detailed Description

Implements creation of the density matrix susceptibility tensors for which $\mathbf{X} = \mathbf{1}$. Guarantees the idempotency of the density matrix up to first-order in LCAO-MO variation. The density matrix susceptibility tensor is represented by:

$$\delta D_{\alpha\beta} = \sum_i \mathbf{B}_{\alpha\beta}^{(i;1)} \cdot \mathbf{F}(\mathbf{r}_i)$$

where $\mathbf{B}_{\alpha\beta}^{(i;1)}$ is the density matrix dipole polarizability defined for the distributed LMO site at \mathbf{r}_i . Its explicit form is given by

$$\mathbf{B}_{\alpha\beta}^{(i;1)} = C_{\alpha i}^{(0)} \mathbf{b}_{\beta}^{(i;1)} C_{\beta i}^{(0)} - \sum_{\gamma} \left(D_{\alpha\gamma}^{(0)} C_{\beta i}^{(0)} + D_{\beta\gamma}^{(0)} C_{\alpha i}^{(0)} \right) \mathbf{b}_{\gamma}^{(i;1)}$$

where the susceptibility of the LCAO-MO coefficient is given by

$$b_{\alpha;w}^{(i;1)} = \frac{1}{4} \sum_u^{x,y,z} [\alpha_i]_{uw} \left[[\mathbb{L}_i]_{\text{Left}}^{-1} \right]_{u;\alpha}$$

for $w = x, y, z$. The auxiliary tensor \mathbb{L} is defined as

$$\mathbb{L} = \mathbf{C}^{(0)\text{T}} \cdot \mathbb{M} \cdot (\mathbf{1} - \mathbf{D}^{(0)})$$

where \mathbb{M} is the dipole integral vector of matrices in AO representation. The left inverse of the i -th element is defined as

$$[\mathbb{L}_i]_{\text{Left}}^{-1} \equiv [\mathbf{L}_i^{\text{T}} \cdot \mathbf{L}_i]^{-1} \cdot \mathbf{L}_i^{\text{T}}$$

Note that $\mathbf{L}_i \equiv [\mathbb{L}]_i$ is a $n \times 3$ matrix, whereas its left inverse is a $3 \times n$ matrix with n being the size of the AO basis set.

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

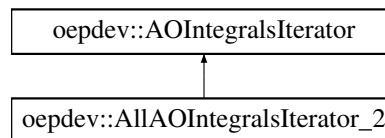
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp_polar_abinitio.cc

14.3 oepdev::AllAOIntegralsIterator_2 Class Reference

Loop over all possible ERI within a particular shell doublet.

```
#include <integrals_iter.h>
```

Inheritance diagram for oepdev::AllAOIntegralsIterator_2:



Public Member Functions

- [AllAOIntegralsIterator_2](#) (const [ShellCombinationsIterator](#) *shellIter)
Construct by shell iterator (const object)
- [AllAOIntegralsIterator_2](#) (std::shared_ptr< [ShellCombinationsIterator](#) > shellIter)
Construct by shell iterator (pointed by shared pointer)
- void [first](#) ()
First iteration.
- void [next](#) ()
Next iteration.
- int [i](#) () const
Grab the current integral i index.
- int [j](#) () const
Grab the current integral j index.
- int [index](#) () const

Additional Inherited Members

14.3.1 Detailed Description

Constructed by providing a const reference or shared pointer to an AllAOShellCombinationsIterator object.

See also

[AllAOShellCombinationsIterator_2](#)

14.3.2 Constructor & Destructor Documentation

AllAOIntegralsIterator_2() [1/2]

```
AllAOIntegralsIterator_2::AllAOIntegralsIterator_2 (
    const ShellCombinationsIterator * shellIter )
```

Parameters

<i>shellIter</i>	- shell iterator object
------------------	-------------------------

AllAOIntegralsIterator_2() [2/2]

```
AllAOIntegralsIterator_2::AllAOIntegralsIterator_2 (
    std::shared_ptr< ShellCombinationsIterator > shellIter )
```

Parameters

<i>shellIter</i>	- shell iterator object
------------------	-------------------------

14.3.3 Member Function Documentation**index()**

```
int oepdev::AllAOIntegralsIterator_2::index (
    void ) const [inline], [virtual]
```

Grab the current index of integral value stored in the buffer

Implements [oepdev::AOIntegralsIterator](#).

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

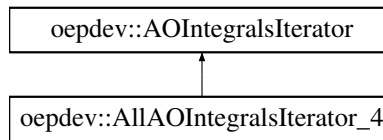
- oepdev/libutil/[integrals_iter.h](#)
- oepdev/libutil/integrals_iter.cc

14.4 oepdev::AllAOIntegralsIterator_4 Class Reference

Loop over all possible ERI within a particular shell quartet.

```
#include <integrals_iter.h>
```

Inheritance diagram for oepdev::AllAOIntegralsIterator_4:



Public Member Functions

- [AllAOIntegralsIterator_4](#) (const [ShellCombinationsIterator](#) *shellIter)
Construct by shell iterator (const object)
- [AllAOIntegralsIterator_4](#) (std::shared_ptr< [ShellCombinationsIterator](#) > shellIter)
Construct by shell iterator (pointed by shared pointer)
- void [first](#) ()
First iteration.
- void [next](#) ()
Next iteration.
- int [i](#) () const
Grab the current integral i index.
- int [j](#) () const
Grab the current integral j index.
- int [k](#) () const
Grab the current integral k index.
- int [l](#) () const
Grab the current integral l index.
- int [index](#) () const

Additional Inherited Members

14.4.1 Detailed Description

Constructed by providing a const reference or shared pointer to an AllAOShellCombinationsIterator object.

See also

[AllAOShellCombinationsIterator_4](#)

14.4.2 Constructor & Destructor Documentation

AllAOIntegralsIterator_4() [1/2]

```
AllAOIntegralsIterator_4::AllAOIntegralsIterator_4 (
    const ShellCombinationsIterator * shellIter )
```

Parameters

<i>shellIter</i>	- shell iterator object
------------------	-------------------------

AllAOIntegralsIterator_4() [2/2]

```
AllAOIntegralsIterator_4::AllAOIntegralsIterator_4 (
    std::shared_ptr< ShellCombinationsIterator > shellIter )
```

Parameters

<i>shellIter</i>	- shell iterator object
------------------	-------------------------

14.4.3 Member Function Documentation**index()**

```
int oepdev::AllAOIntegralsIterator_4::index (
    void ) const [inline], [virtual]
```

Grab the current index of integral value stored in the buffer

Implements [oepdev::AOIntegralsIterator](#).

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

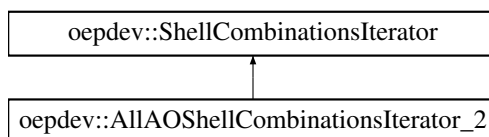
- oepdev/libutil/[integrals_iter.h](#)
- oepdev/libutil/integrals_iter.cc

14.5 oepdev::AllAOShellCombinationsIterator_2 Class Reference

Loop over all possible ERI shells in a shell doublet.

```
#include <integrals_iter.h>
```

Inheritance diagram for oepdev::AllAOShellCombinationsIterator_2:



Public Member Functions

- [AllAOShellCombinationsIterator_2](#) (SharedBasisSet [bs_1](#), SharedBasisSet [bs_2](#))
Iterate over shell doublets. Construct by providing basis sets for each axis. The basis sets must be defined for the same molecule.
- [AllAOShellCombinationsIterator_2](#) (std::shared_ptr< [IntegralFactory](#) > integrals)
Construct by providing integral factory.
- [AllAOShellCombinationsIterator_2](#) (const [IntegralFactory](#) &integrals)
- [AllAOShellCombinationsIterator_2](#) (std::shared_ptr< psi::IntegralFactory > integrals)
Construct by providing integral factory.
- [AllAOShellCombinationsIterator_2](#) (const psi::IntegralFactory &integrals)
- void [first](#) ()
First iteration.
- void [next](#) ()
Next iteration.
- void [compute_shell](#) (std::shared_ptr< [oepdev::TwoBodyAOInt](#) > tei) const
Compute ERI's for the current shell. The eris are stored in the buffer of the argument object.
- void **compute_shell** (std::shared_ptr< psi::TwoBodyAOInt > tei) const
- int [P](#) () const
Grab the current shell P index.
- int [Q](#) () const
Grab the current shell Q index.

Additional Inherited Members

14.5.1 Detailed Description

Constructed by providing [IntegralFactory](#) object or shared pointers to two basis set spaces.

14.5.2 Constructor & Destructor Documentation

AllAOShellCombinationsIterator_2() [1/5]

```
AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 (
    SharedBasisSet bs_1,
    SharedBasisSet bs_2 )
```

Parameters

<i>bs_1</i>	- basis set of axis 1
<i>bs_2</i>	- basis set of axis 2

AllAOShellCombinationsIterator_2() [2/5]

```
oepdev::AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 (
    std::shared_ptr< IntegralFactory > integrals )
```

Parameters

<i>integrals</i>	- OepDev integral factory object
------------------	----------------------------------

AllAOShellCombinationsIterator_2() [3/5]

```
AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 (
    const IntegralFactory & integrals )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

AllAOShellCombinationsIterator_2() [4/5]

```
AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 (
    std::shared_ptr< psi::IntegralFactory > integrals )
```

Parameters

<i>integrals</i>	- Psi4 integral factory object
------------------	--------------------------------

AllAOShellCombinationsIterator_2() [5/5]

```
AllAOShellCombinationsIterator_2::AllAOShellCombinationsIterator_2 (
```

```
const psi::IntegralFactory & integrals )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

14.5.3 Member Function Documentation

compute_shell()

```
void AllAOShellCombinationsIterator_2::compute_shell (
    std::shared_ptr< oepdev::TwoBodyAOInt > tei ) const [virtual]
```

Parameters

<i>tei</i>	- two electron AO integral
------------	----------------------------

Implements [oepdev::ShellCombinationsIterator](#).

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

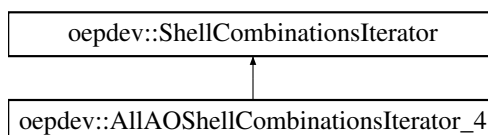
- oepdev/libutil/[integrals_iter.h](#)
- oepdev/libutil/integrals_iter.cc

14.6 oepdev::AllAOShellCombinationsIterator_4 Class Reference

Loop over all possible ERI shells in a shell quartet.

```
#include <integrals_iter.h>
```

Inheritance diagram for oepdev::AllAOShellCombinationsIterator_4:



Public Member Functions

- [AllAOShellCombinationsIterator_4](#) (SharedBasisSet [bs_1](#), SharedBasisSet [bs_2](#), SharedBasisSet [bs_3](#), SharedBasisSet [bs_4](#))

Iterate over shell quartets. Construct by providing basis sets for each axis. The basis sets must be defined for the same molecule.

- [AllAOShellCombinationsIterator_4](#) (std::shared_ptr< [IntegralFactory](#) > integrals)

Construct by providing integral factory.

- [AllAOShellCombinationsIterator_4](#) (const [IntegralFactory](#) &integrals)
- [AllAOShellCombinationsIterator_4](#) (std::shared_ptr< [psi::IntegralFactory](#) > integrals)

Construct by providing integral factory.

- [AllAOShellCombinationsIterator_4](#) (const [psi::IntegralFactory](#) &integrals)
- void [first](#) ()

Do the first iteration.

- void [next](#) ()

Do the next iteration.

- void [compute_shell](#) (std::shared_ptr< [oepdev::TwoBodyAOInt](#) > tei) const
- void [compute_shell](#) (std::shared_ptr< [psi::TwoBodyAOInt](#) > tei) const
- int [P](#) () const

Grab the current shell P index.

- int [Q](#) () const

Grab the current shell Q index.

- int [R](#) () const

Grab the current shell R index.

- int [S](#) () const

Grab the current shell S index.

Additional Inherited Members

14.6.1 Detailed Description

Constructed by providing [IntegralFactory](#) object or shared pointers to four basis set spaces.

14.6.2 Constructor & Destructor Documentation

AllAOShellCombinationsIterator_4() [1/5]

```
AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    SharedBasisSet bs_1,
    SharedBasisSet bs_2,
    SharedBasisSet bs_3,
    SharedBasisSet bs_4 )
```

Parameters

<i>bs_1</i>	- basis set of axis 1
<i>bs_2</i>	- basis set of axis 2
<i>bs_3</i>	- basis set of axis 3
<i>bs_4</i>	- basis set of axis 4

AllAOShellCombinationsIterator_4() [2/5]

```
oepdev::AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    std::shared_ptr< IntegralFactory > integrals )
```

Parameters

<i>integrals</i>	- OepDev integral factory object
------------------	----------------------------------

AllAOShellCombinationsIterator_4() [3/5]

```
AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    const IntegralFactory & integrals )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

AllAOShellCombinationsIterator_4() [4/5]

```
AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    std::shared_ptr< psi::IntegralFactory > integrals )
```

Parameters

<i>integrals</i>	- OepDev integral factory object
------------------	----------------------------------

AllAOShellCombinationsIterator_4() [5/5]

```
AllAOShellCombinationsIterator_4::AllAOShellCombinationsIterator_4 (
    const psi::IntegralFactory & integrals )
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

14.6.3 Member Function Documentation

compute_shell() [1/2]

```
void AllAOShellCombinationsIterator_4::compute_shell (
    std::shared_ptr< oepdev::TwoBodyAOInt > tei ) const [virtual]
```

Compute integrals in a current shell. Works both for [oepdev::TwoBodyAOInt](#) and [psi::TwoBodyAOInt](#)

Parameters

<i>tei</i>	- two body integral object
------------	----------------------------

Implements [oepdev::ShellCombinationsIterator](#).

compute_shell() [2/2]

```
void oepdev::AllAOShellCombinationsIterator_4::compute_shell (
    std::shared_ptr< psi ::TwoBodyAOInt > tei ) const [virtual]
```

Compute integrals in a current shell. Works both for [oepdev::TwoBodyAOInt](#) and [psi::TwoBodyAOInt](#)

Parameters

<i>tei</i>	- two body integral object
------------	----------------------------

Implements [oepdev::ShellCombinationsIterator](#).

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

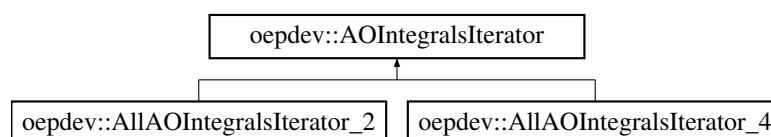
- [oepdev/libutil/integrals_iter.h](#)
- [oepdev/libutil/integrals_iter.cc](#)

14.7 oepdev::AOIntegralsIterator Class Reference

Iterator for AO Integrals. Abstract Base.

```
#include <integrals_iter.h>
```

Inheritance diagram for oepdev::AOIntegralsIterator:



Public Member Functions

- [AOIntegralsIterator](#) ()
Base Constructor.
- virtual [~AOIntegralsIterator](#) ()
Base Destructor.
- virtual void [first](#) (void)=0
Do the first iteration.
- virtual void [next](#) (void)=0
Do the next iteration.
- virtual int [i](#) (void) const
Grab i-th index.
- virtual int [j](#) (void) const
Grab j-th index.
- virtual int [k](#) (void) const
Grab k-th index.
- virtual int [l](#) (void) const
Grab l-th index.
- virtual int [index](#) (void) const =0
Grab index in the integral buffer.
- virtual bool [is_done](#) (void)
Returns the status of an iterator.

Static Public Member Functions

- static std::shared_ptr< [AOIntegralsIterator](#) > [build](#) (const [ShellCombinationsIterator](#) *shellIter, std::string mode="ALL")
- static std::shared_ptr< [AOIntegralsIterator](#) > [build](#) (std::shared_ptr< [ShellCombinationsIterator](#) > shellIter, std::string mode="ALL")

Protected Attributes

- bool [done](#)
The status of an iterator.

14.7.1 Detailed Description

14.7.2 Member Function Documentation

build() [1/2]

```
std::shared_ptr< AOIntegralsIterator > AOIntegralsIterator::build (
    const ShellCombinationsIterator * shellIter,
    std::string mode = "ALL" ) [static]
```

Build AO integrals iterator from current state of iterator over shells

Parameters

<i>shellIter</i>	- iterator over shells - either "ALL" or "UNIQUE" (iterate over all or unique integrals)
------------------	--

Returns

iterator over AO integrals

build() [2/2]

```
std::shared_ptr< AOIntegralsIterator > AOIntegralsIterator::build (
    std::shared_ptr< ShellCombinationsIterator > shellIter,
    std::string mode = "ALL" ) [static]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

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

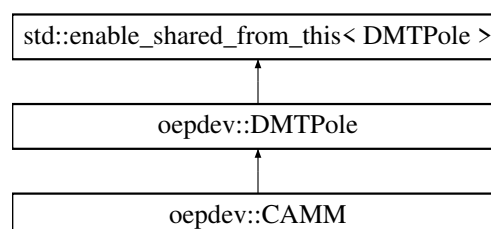
- oepdev/libutil/[integrals_iter.h](#)
- oepdev/libutil/integrals_iter.cc

14.8 oepdev::CAMM Class Reference

Cumulative Atomic Multipole Moments.

```
#include <dmt.h>
```

Inheritance diagram for oepdev::CAMM:



Public Member Functions

- **CAMM** (psi::SharedWavefunction wfn, int n)
- **CAMM** (const **CAMM** *other)
- virtual void **compute** (psi::SharedMatrix D, bool transition, int n)
Compute DMTP's from the one-particle density matrix.
- virtual void **print_header** (void) const
Print the header.
- virtual std::shared_ptr< **DMTPole** > **clone** (void) const override
Make a deep copy (wfn_, mol_, and primary_ are shallow-copied)

Additional Inherited Members

14.8.1 Detailed Description

Cumulative atomic multipole representation of the molecular charge distribution. Method of Sokalski and Poirier. Ref.: W. A. Sokalski and R. A. Poirier, *Chem. Phys. Lett.*, 98(1) **1983**

Methodology.

The distributed multipole moments are computed in the following way:

- first the atomic additive multipole moments (AAMM's) with origins set to the global coordinate system origin are computed. AO basis set partitioning is used to distribute the AAMM's onto the atomic centres.
- subsequently, the AAMM's origins are moved to the corresponding atomic site.

The computation of the AAMM's is performed according to the following prescription:

$$M_{uw\dots z}^{(A)}(\mathbf{0}) = \sum_{\alpha \in A} \sum_{\beta \in \text{allAO's}} D_{\alpha\beta}^{\text{OED}} \langle \alpha | \mathcal{M}_{uw\dots z}(\mathbf{0}) | \beta \rangle$$

where $M_{uw\dots z}^{(A)}$ denotes the $(uw\dots z)$ -th component of the multipole centered at atomic site A , the symbol $\mathcal{M}(\mathbf{0})$ is the associated quantum mechanical operator and $D_{\alpha\beta}^{\text{OED}}$ is the (generalized) one-particle density matrix element in AO basis (Greek indices).

Recentering of the multipole moments is described in the documentation of [oepdev::DMTPole::recenter](#).

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

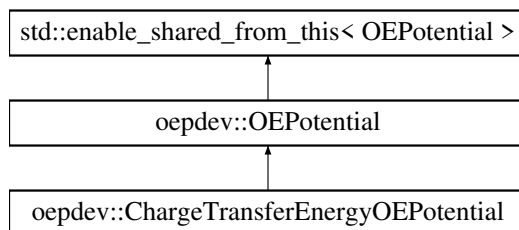
- oepdev/lib3d/[dmtp.h](#)
- oepdev/lib3d/[dmtp_camm.cc](#)

14.9 oepdev::ChargeTransferEnergyOEPotential Class Reference

Generalized One-Electron Potential for Charge-Transfer Interaction Energy.

```
#include <oep.h>
```

Inheritance diagram for oepdev::ChargeTransferEnergyOEPotential:



Public Member Functions

- **ChargeTransferEnergyOEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)
- **ChargeTransferEnergyOEPotential** (SharedWavefunction [wfn](#), Options &options)
- **ChargeTransferEnergyOEPotential** (const [ChargeTransferEnergyOEPotential](#) *f)
- virtual void [compute](#) (const std::string &oepType) override
Compute matrix forms of all OEP's within a specified OEP type.
- virtual void [compute_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared_ptr< psi::Vector > &v) override
Compute value of potential in point x, y, z and save at v.
- virtual void [print_header](#) () const override
Header information.
- virtual std::shared_ptr< [OEPotential](#) > [clone](#) (void) const override
Make a deep copy of this object.
- virtual void [initialize](#) () override
Initialize the object (expert)

Protected Member Functions

- virtual void **rotate_oep** (psi::SharedMatrix, psi::SharedMatrix, psi::SharedMatrix) override
- virtual void **translate_oep** (psi::SharedVector) override

Additional Inherited Members

14.9.1 Detailed Description

Contains the following OEP types:

- `Otto-Ladik.V1.GDF` - DF-based term (group I)
- `Otto-Ladik.V3.CAMM-nj` - CAMM-based term (group III; truncated on distributed charges)

Group II terms do not require any particular OEP's due to great simplification of this term. Atomic numbers and LMO centroids are sufficient.

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

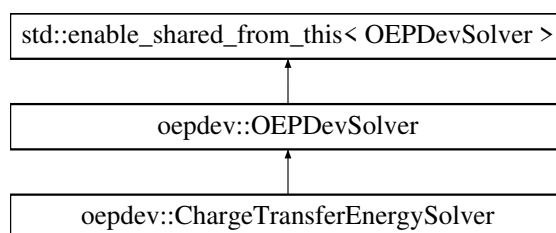
- `oepdev/liboep/oep.h`
- `oepdev/liboep/oep_energy_ct.cc`

14.10 oepdev::ChargeTransferEnergySolver Class Reference

Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.

```
#include <solver.h>
```

Inheritance diagram for `oepdev::ChargeTransferEnergySolver`:



Public Member Functions

- **ChargeTransferEnergySolver** ([SharedWavefunctionUnion](#) wfn_union)
- virtual double [compute_oep_based](#) (const std::string &method="DEFAULT")
Compute property by using OEPs.
- virtual double [compute_benchmark](#) (const std::string &method="DEFAULT")
Compute property by using benchmark method.

Additional Inherited Members

14.10.1 Detailed Description

The implemented methods are shown below

Table 14.15: Methods available in the Solver

Keyword	Method Description
Benchmark Methods	
OTTO_LADIK	<i>Default.</i> CT energy at HF level from Otto and Ladik (1975).
EFP2	CT energy at HF level from EFP2 model.
OEP-Based Methods	
OTTO_LADIK	<i>Default.</i> OEP-based Otto-Ladik expressions.

In order to construct this solver, **always** use the `OEPDevSolver::build` static factory method.

Below the detailed description of the implemented equations is given for each of the above provided methods. In the formulae across, it is assumed that the orbitals are real. The Coulomb notation for electron repulsion integrals (ERIs) is adopted; i.e,

$$(ac|bd) = \iint d\mathbf{r}_1 d\mathbf{r}_2 \phi_a(\mathbf{r}_1) \phi_c(\mathbf{r}_1) \frac{1}{r_{12}} \phi_b(\mathbf{r}_2) \phi_d(\mathbf{r}_2)$$

Greek subscripts denote basis set orbitals whereas Italic subscripts denote the occupied molecular orbitals.

The CT energy between molecules *A* and *B* is given by

$$E^{\text{CT}} = E^{A^+B^-} + E^{A^-B^+}$$

Benchmark Methods

CT energy at HF level by Otto and Ladik (1975).

For a closed-shell system, CT energy equation of Otto and Ladik becomes

$$E^{A^+B^-} \approx 2 \sum_{i \in A}^{\text{Occ}_A} \sum_{n \in B}^{\text{Vir}_B} \frac{V_{in}^2}{\epsilon_i - \epsilon_n}$$

where

$$V_{in} = V_{in}^B + 2 \sum_{j \in B}^{\text{Occ}_B} (in|jj) - \sum_{k \in A}^{\text{Occ}_A} S_{kn} \left\{ V_{ik}^B + 2 \sum_{j \in B}^{\text{Occ}_B} (ik|jj) \right\} \\ - \sum_{j \in B}^{\text{Occ}_B} \left[S_{ij} \left\{ V_{nj}^A + 2 \sum_{k \in A}^{\text{Occ}_A} (1 - \delta_{ik})(nj|kk) \right\} + (nj|ij) \right] + \sum_{k \in A}^{\text{Occ}_A} \sum_{j \in B}^{\text{Occ}_B} S_{kj} (1 - \delta_{ik})(ik|nj)$$

and analogously the twin term.

CT energy at HF level by EFP2.

In EFP2 method, CT energy is given as

$$E^{A+B^-} \approx 2 \sum_{i \in A}^{\text{Occ}_A} \sum_{n \in B}^{\text{Vir}_B} \frac{V_{in}^2}{F_{ii} - T_{nn}}$$

where

$$V_{in}^2 = \frac{V_{in}^{EF,B} - \sum_{m \in A}^{\text{All}_A} V_{im}^{EF,B} S_{mn}^B}{1 - \sum_{m \in A}^{\text{All}_A} S_{mn}^2} \left\{ V_{in}^{EF,B} - \sum_{m \in A}^{\text{All}_A} V_{im}^{EF,B} S_{mn} + \sum_{j \in B}^{\text{Occ}_B} S_{ij} \left(T_{nj} - \sum_{m \in A}^{\text{All}_A} S_{nm} T_{mj} \right) \right\}$$

and analogously the twin term.

OEP-Based Methods

OEP-Based Otto-Ladik's theory

After introducing OEPs, the original Otto-Ladik's theory is reformulated *without* approximation as

$$E^{A+B^-} \approx 2 \sum_{i \in A}^{\text{Occ}_A} \sum_{n \in B}^{\text{Vir}_B} \frac{\left(V_{in}^{\text{DF}} + V_{in}^{\text{ESP,A}} + V_{in}^{\text{ESP,B}} \right)^2}{\epsilon_i - \epsilon_n}$$

where

$$\begin{aligned} V_{in}^{\text{DF}} &= \sum_{\eta \in B}^{\text{Aux}_B} S_{i\eta} G_{\eta n}^B \\ V_{in}^{\text{ESP,A}} &= \sum_{k \in A}^{\text{Occ}_A} \sum_{j \in B}^{\text{Occ}_B} S_{kj} \sum_{x \in A} V_{nj}^{(x)} q_{ik}^{(x)} \\ V_{in}^{\text{ESP,B}} &= - \sum_{k \in A}^{\text{Occ}_A} S_{kn} V_{ik}^B \end{aligned}$$

The OEP matrix for density fitted part is given by

$$G_{\eta n}^B = \sum_{\eta' \in B}^{\text{Aux}_B} [\mathbf{S}^{-1}]_{\eta\eta'} \left\{ V_{\eta'n}^B + \sum_{j \in B}^{\text{Occ}_B} [2(\eta'n|jj) - (\eta'j|nj)] \right\}$$

The OEP ESP-A charges are fit to reproduce the OEP potential

$$v_{ik}^A(\mathbf{r}) \equiv (1 - \delta_{ik}) \int \frac{\phi_i(\mathbf{r}') \phi_k(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' - \delta_{ik} \left(\sum_{x \in A} \frac{-Z_x}{|\mathbf{r} - \mathbf{r}_x|} + 2 \sum_{k \in A}^{\text{Occ}_A} \int \frac{\phi_k(\mathbf{r}') \phi_k(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' - 2 \int \frac{\phi_i(\mathbf{r}') \phi_i(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' \right)$$

so that

$$v_{ik}^A(\mathbf{r}) \cong \sum_{x \in A} \frac{q_{ik}^{(x)}}{|\mathbf{r} - \mathbf{r}_x|}$$

The OEP ESP-B charges are fit to reproduce the electrostatic potential of molecule *B* (they are standard ESP charges).

14.10.2 Member Function Documentation

compute_oep_based()

```
double ChargeTransferEnergySolver::compute_oep_based (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one DEFAULT OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implements [oepdev::OEPDevSolver](#).

compute_benchmark()

```
double ChargeTransferEnergySolver::compute_benchmark (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one DEFAULT benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implements [oepdev::OEPDevSolver](#).

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

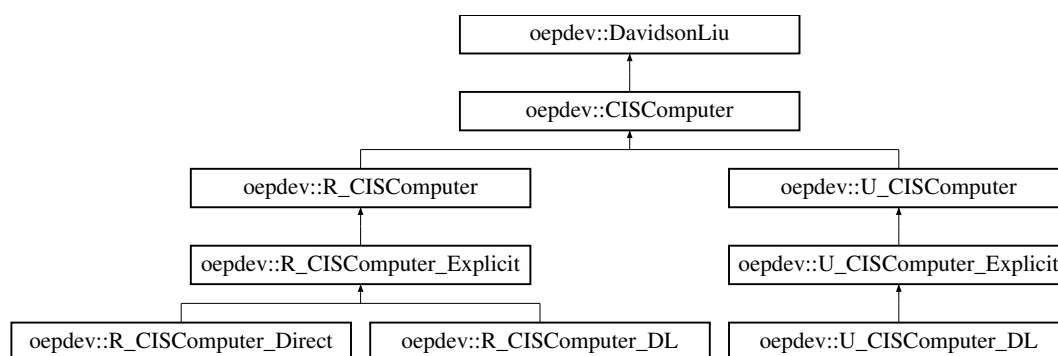
- [oepdev/libsolver/solver.h](#)
- [oepdev/libsolver/solver_energy_ct.cc](#)

14.11 oepdev::CISComputer Class Reference

[CISComputer](#).

```
#include <cis.h>
```

Inheritance diagram for [oepdev::CISComputer](#):



Public Member Functions

- virtual `~CISComputer` ()
Destructor.
- virtual void `compute` (void)
Solve the CIS problem.
- virtual void `clear_dpd` (void)
Clear DPD instance.
- int `nstates` (void) const
Get the total number of excited states.
- psi::SharedVector `eigenvalues` () const
Get the CIS eigenvalues.
- psi::SharedVector `E` () const
- psi::SharedMatrix `eigenvectors` () const
Get the CIS eigenvectors.
- psi::SharedMatrix `U` () const
- std::pair< double, double > `U_homo_lumo` (int l, int h=0, int l=0) const
*Get the HOMO+*h*->LUMO+*l* CIS coefficient for a given excited state l for spin alpha and beta.*
- SharedMatrix `Da_mo` (int i) const
Compute MO one-particle alpha density matrix for state i
- SharedMatrix `Db_mo` (int i) const
Compute MO one-particle beta density matrix for state i
- SharedMatrix `Da_ao` (int i) const
Compute AO one-particle alpha density matrix for state i
- SharedMatrix `Db_ao` (int i) const
Compute AO one-particle beta density matrix for state i
- `SharedDMTPole camm` (int j, bool symmetrize=false) const
Compute CAMM for j excited state.
- SharedMatrix `Ta_ao` (int j) const
*Compute MO one-particle alpha 0->*j* transition density matrix.*

- SharedMatrix [Tb_ao](#) (int j) const
*Compute MO one-particle beta 0->*j* transition density matrix.*
- SharedMatrix [Ta_ao](#) (int i, int j) const
*Compute MO one-particle alpha i->*j* transition density matrix.*
- SharedMatrix [Tb_ao](#) (int i, int j) const
*Compute MO one-particle beta i->*j* transition density matrix.*
- [SharedDMTPole trcamm](#) (int j, bool symmetrize=true) const
*Compute TrCAMM for 0->*j* transition.*
- [SharedDMTPole trcamm](#) (int i, int j, bool symmetrize=true) const
*Compute TrCAMM for i->*j* transition.*
- SharedVector [transition_dipole](#) (int j) const
*Compute transition dipole moment for 0->*j* transition.*
- SharedVector [transition_dipole](#) (int i, int j) const
*Compute transition dipole moment for i->*j* transition.*
- double [oscillator_strength](#) (int j) const
*Compute oscillator strength for 0->*j* transition.*
- double [oscillator_strength](#) (int i, int j) const
*Compute oscillator strength for i->*j* transition.*
- double [s2](#) (int i) const
*Compute <S2> expectation value for the *i*th state.*
- void [determine_electronic_state](#) (int &l)
Determine electronic state.
- std::shared_ptr< [CISData](#) > [data](#) (int l, int h, int l, bool symmetrize_trcamm=false)
Return CIS data structure for a given excited state l

Static Public Member Functions

- static std::shared_ptr< [CISComputer](#) > [build](#) (const std::string &type, std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt, const std::string &reference="")
Build CIS Computer.

Static Public Attributes

- static const std::vector< std::string > [reference_types](#) = {"RHF", "UHF"}
Slater determinant possible references, that are implemented.

Protected Member Functions

- **CISComputer** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt, psi::IntegralTransform::TransformType trans_type)
- virtual void **print_header_** (void)
- virtual void **set_nstates_** (void)
- virtual void **allocate_memory** (void)
- virtual void **allocate_hamiltonian_** (void)
- virtual void **prepare_for_cis_** (void)
- virtual void **build_hamiltonian_** (void)=0
- virtual void **diagonalize_hamiltonian_** (void)
- virtual void **standardize_amplitudes_** (void)
- virtual void **print_excited_states_** (void)
- virtual void **print_excited_state_character_** (int l)=0
- virtual void **set_beta_** (void)=0
- virtual void **transform_integrals_** (void)
- virtual void **davidson_liu_compute_diagonal_hamiltonian** (void)
- virtual void **davidson_liu_compute_sigma** (void)

Protected Attributes

- std::shared_ptr< psi::Wavefunction > [ref_wfn_](#)
Reference wavefunction.
- const int [nmo_](#)
Psi4 Options.
- const int [naocc_](#)
Number of alpha occupied MO's.
- const int [nbocc_](#)
Number of beta occupied MO's.
- const int [navir_](#)
Number of alpha virtual MO's.
- const int [nbvir_](#)
Number of beta virtual MO's.
- int [ndets_](#)
Number of excited determinants.
- int [nstates_](#)
Number of excited states.
- SharedMatrix [H_](#)
CIS Excited State Hamiltonian in Slater determinantal basis.
- SharedMatrix [U_](#)
- SharedVector [E_](#)
- std::shared_ptr< psi::JK > [jk_](#)

Computer of generalized JK objects.

- SharedVector **eps_a_o_**
- SharedVector **eps_a_v_**
- SharedVector **eps_b_o_**
- SharedVector **eps_b_v_**
- const psi::IntegralTransform::TransformationType [transformation_type_](#)

MO Integral Transformation Type.

- std::shared_ptr< psi::IntegralTransform > **inttrans_**

14.11.1 Detailed Description

14.11.2 Member Function Documentation

build()

```
std::shared_ptr< CISComputer > oepdev::CISComputer::build (
    const std::string & type,
    std::shared_ptr< psi::Wavefunction > wfn,
    psi::Options & opt,
    const std::string & reference = "" ) [static]
```

Parameters

<i>type</i>	- Type of computer
<i>wfn</i>	- Psi4 wavefunction
<i>opt</i>	- Psi4 options
<i>reference</i>	- Reference Slater determinant (RHF, UHF available).

Available computer types:

- RESTRICTED or RCIS - RHF wavefunction is used as reference state
- UNRESTRICTED or UCIS - UHF wavefunction is used as reference state

Implementation

The CIS Hamiltonian in the basis space of singly-excited Slater determinants is constructed from canonical molecular orbitals (CMO's)

$$\begin{aligned}\langle \Phi_0 | \mathcal{H} | \Phi_i^a \rangle &= 0 \\ \langle \Phi_j^b | \mathcal{H} | \Phi_i^a \rangle &= \delta_{ij} \delta_{ab} (\epsilon_a - \epsilon_i) + \langle aj | ib \rangle - \langle aj | bi \rangle\end{aligned}$$

where i labels the occupied CMO's whereas a labels the virtual CMO's. In the above equation, $\langle aj|ib \rangle$ is the 2-electron 4-centre integral in physicist's notation. After integrating out the spin coordinate, four blocks of Hamiltonian are explicitly given as

$$\begin{aligned}\langle \Phi_j^b | \mathcal{H} | \Phi_i^a \rangle &= \delta_{ij} \delta_{ab} (\epsilon_a - \epsilon_i) + [ia|jb] - [ab|ij] \\ \langle \Phi_j^{\bar{b}} | \mathcal{H} | \Phi_i^{\bar{a}} \rangle &= \delta_{i\bar{j}} \delta_{\bar{a}\bar{b}} (\epsilon_{\bar{a}} - \epsilon_{\bar{i}}) + [\bar{i}\bar{a}|\bar{j}\bar{b}] - [\bar{a}\bar{b}|\bar{i}\bar{j}] \\ \langle \Phi_j^{\bar{b}} | \mathcal{H} | \Phi_i^a \rangle &= [ia|\bar{j}\bar{b}] \\ \langle \Phi_j^b | \mathcal{H} | \Phi_i^{\bar{a}} \rangle &= [\bar{i}\bar{a}|jb]\end{aligned}$$

where the $[ia|jb]$ is the 2-electron 4-centre integral in the chemist's (Coulomb) notation.

Such matrix is diagonalized yielding the excitation energies (wrt HF ground state) as well as the CIS coefficients

$$\sum_{ij} \sum_{ab} t_{i,I}^a H_{ij}^{ab} t_{j,J}^b = E_I \delta_{IJ}$$

where the summations above extend over alpha and beta electron spin labels and $t_{i,I}^a$ is the CIS amplitude for the i th excited state, associated with the $i \rightarrow a$ excitation with respect to the HF reference determinant. Note that E_I is *not* the excited state energy, but the energy relative the the HF reference energy.

See also

For Davidson-Liu solution to CIS problem, see [oepdev::R_CISComputer_DL](#) and [oepdev::U_CISComputer_DL](#).

Transition density matrix

AO basis transition density from ground (HF) to excited (CIS) state is given by

$$P_{\mu\nu}^{(g \rightarrow e)} = \sum_i \sum_a^{\text{Occ Vir}} t_{i,e}^a C_{vi} C_{\mu a} + \sum_{\bar{i}} \sum_{\bar{a}}^{\text{Occ Vir}} t_{\bar{i},e}^{\bar{a}} C_{v\bar{i}} C_{\mu\bar{a}}$$

Excited state density matrix

CMO basis excited state density matrix for alpha spin is given by

Analogous expression is given for the beta spin.

AO representation of the CMO excited state density matrix is

$$P_{\mu\nu}^{(e)} = \sum_{pq} C_{\mu p} P_{pq}^{(e)} C_{vq} + \sum_{\bar{p}\bar{q}} C_{\mu\bar{p}} P_{\bar{p}\bar{q}}^{(e)} C_{v\bar{q}}$$

which is the sum of alpha and beta density matrices in CMO basis transformed to AO basis.

The CMO excited state density matrix for spin alpha is given by

$$P_{pq}^{(e)} = \begin{cases} \delta_{pq} - \sum_a^{\text{Vir}} t_{p,e}^a t_{q,e}^a & \text{for } p, q \in \text{Occ} \\ \sum_i^{\text{Occ}} t_{i,e}^p t_{i,e}^q & \text{for } p, q \in \text{Vir} \\ 0 & \text{otherwise} \end{cases}$$

The beta spin density matrix is generated analogously as above.

The cumulative atomic multipole moments (CAMM) are computed from the excited state density matrices in AO basis. The nuclear contribution is included.

Transition multipole moments

The transition dipole moment is computed from the AO transition density matrix and the dipole integrals in AO basis, i.e.,

$$\langle \Phi_0 | \hat{\mu}_u | \Psi_e \rangle = \text{Tr} \left[\mathbf{d}^{(u)} \cdot \mathbf{P}^{g \rightarrow e} \right]$$

Oscillator strength is computed from the transition dipole moment via

$$f^{g \rightarrow e} = \frac{2}{3} E_e \left| \langle \Phi_0 | \hat{\mu} | \Psi_e \rangle \right|^2$$

Transition cumulative atomic multipole moments (TrCAMM) are computed from the transition density matrices in AO basis. The nuclear contribution is not included.

Spin angular momentum

The expectation value of the \hat{S}^2 operator is calculated from the CIS amplitudes and MOs of the reference wavefunction according to D. Maurice and M. Head-Gordon, *Int. J. Quant. Chem.*, **1995**, 95, 010361-10:

$$\begin{aligned} \langle \hat{S}^2 \rangle_{\text{UCIS}} = \langle \hat{S}^2 \rangle_{\text{UHF}} &- \text{Tr} \left[\mathbf{Q}_{\text{Occ}}^{(\alpha)} \cdot \left\{ \mathbf{P}_{\text{Occ}}^{(e,\alpha)} - \mathbf{1} \right\} \right] - \text{Tr} \left[\mathbf{Q}_{\text{Occ}}^{(\beta)} \cdot \left\{ \mathbf{P}_{\text{Occ}}^{(e,\beta)} - \mathbf{1} \right\} \right] \\ &- \text{Tr} \left[\mathbf{Q}_{\text{Vir}}^{(\alpha)} \cdot \mathbf{P}_{\text{Vir}}^{(e,\alpha)} \right] - \text{Tr} \left[\mathbf{Q}_{\text{Vir}}^{(\beta)} \cdot \mathbf{P}_{\text{Vir}}^{(e,\beta)} \right] - 2 \sum_i^{\text{Occ}} \sum_a^{\text{Vir}} \sum_{\bar{j}}^{\text{Occ}} \sum_{\bar{b}}^{\text{Vir}} \Delta_{i\bar{j}}^* \Delta_{a\bar{b}} t_{i,e}^a t_{\bar{j},e}^{\bar{b}} \end{aligned}$$

where

$$\begin{aligned} [\mathbf{Q}_{\text{Occ}}^{(\alpha)}]_{ij} &= \sum_{\bar{k}}^{\text{Occ}} \Delta_{\bar{k}i}^* \Delta_{\bar{k}j} \\ [\mathbf{Q}_{\text{Occ}}^{(\beta)}]_{i\bar{j}} &= \sum_k^{\text{Occ}} \Delta_{ki}^* \Delta_{k\bar{j}} \\ [\mathbf{Q}_{\text{Vir}}^{(\alpha)}]_{ab} &= \sum_{\bar{k}}^{\text{Occ}} \Delta_{\bar{k}a}^* \Delta_{\bar{k}b} \\ [\mathbf{Q}_{\text{Vir}}^{(\beta)}]_{a\bar{b}} &= \sum_k^{\text{Occ}} \Delta_{k\bar{a}}^* \Delta_{kb} \end{aligned}$$

and

$$\Delta_{pq} = \sum_{\mu\nu} C_{\mu p} S_{\mu\nu} C_{\nu p}$$

The diagnostic for UHF spin contamination is given by

$$\langle \hat{S}^2 \rangle_{\text{UHF}} = \langle \hat{S}^2 \rangle_{\text{exact}} + N_{\beta} - \sum_i^{\text{Occ}} \sum_{\bar{j}}^{\text{Occ}} |\Delta_{i\bar{j}}|^2$$

with

$$\langle \hat{S}^2 \rangle_{\text{exact}} = \frac{N_{\alpha} - N_{\beta}}{2} \left(\frac{N_{\alpha} - N_{\beta} + 2}{2} \right)$$

and is also printed out to the output file.

Note

Useful options:

- `CIS_TYPE` - Algorithm of CIS. Available: `DAVIDSON_LIU` (Default), `DIRECT_EXPLICIT` (only RHF reference), `EXPLICIT`.
- `CIS_SCHWARTZ_CUTOFF` - Cutoff for Schwartz ERI screening. Default: 0.0. Relevant if `DAVIDSON_LIU` or `DIRECT_EXPLICIT` are chosen as CIS type.
- `CIS_STANDARDIZE_AMPLITUDES` - If true, CIS amplitudes of each excited state are rephased so that the leading amplitude is positive. Default: true.
- `OEPDEV_AMPLITUDE_PRINT_THRESHOLD` - Control threshold how many CIS amplitudes to print to the output. Default: 0.1.
- For UHF references, SAD guess might lead to triplet instabilities. It is then better to set `CORE` as the UHF guess

14.11.3 Member Data Documentation

`nmo_`

```
const int oepdev::CISComputer::nmo_ [protected]
```

Number of MO's

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

- `oepdev/libutil/cis.h`
- `oepdev/libutil/cis_base.cc`

14.12 oepdev::CISData Struct Reference

CIS wavefunction parameters. Container structure.

```
#include <cis.h>
```

Public Member Functions

- [CISData](#) (void)=default
Null Constructor.
- [CISData](#) (const [CISData](#) *)
Copy Constructor.

Public Attributes

- double [E_ex](#)
Excitation energy.
- double [t.homo.lumo](#)
CIS HOMO-LUMO amplitude.
- SharedMatrix [Pe](#)
Excited state density matrix (sum of alpha and beta)
- SharedMatrix [Peg](#)
Transition ground-to-excited state density matrix (sum of alpha and beta)
- [SharedDMTPole](#) [trcamm](#)
TrCAMM.
- [SharedDMTPole](#) [camm.homo](#)
CAMM for HOMO orbital.
- [SharedDMTPole](#) [camm.lumo](#)
CAMM for LUMO orbital.

14.12.1 Detailed Description

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

- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis_base.cc

14.13 oepdev::CPHF Class Reference

[CPHF](#) solver class.

```
#include <cphf.h>
```

Public Member Functions

Constructor and Destructor

- [CPHF](#) (SharedWavefunction ref_wfn, Options &[options](#))
Constructor.
- [~CPHF](#) ()
Destructor.

Executor

- void [compute](#) (void)
run the calculations

Printer

- void [print](#) (void) const
print to output file

Accessors

- int [nocc](#) (void) const
get the number of occupied orbitals
- std::shared_ptr< Wavefunction > [wfn](#) (void) const
grab the wavefunction
- Options & [options](#) (void) const
grab the Psi4 options
- std::shared_ptr< Matrix > [polarizability](#) (void) const
retrieve the molecular (total) polarizability
- std::shared_ptr< Matrix > [polarizability](#) (int i) const
retrieve the i-th orbital-associated polarizability
- std::shared_ptr< Matrix > [polarizability](#) (int i, int j) const
retrieve the charge-transfer polarizability associated with orbitals i and j
- std::shared_ptr< Matrix > [X](#) (int x) const
retrieve the X operator O-V perturbation matrix in AO basis for x-th component
- std::vector< std::shared_ptr< Matrix > > [X](#) (void) const
retrieve the X operator O-V perturbation matrix in AO basis for all three Cartesian components
- std::shared_ptr< Matrix > [X_mo](#) (int x) const
retrieve the X operator O-V perturbation matrix in MO basis for x-th component
- std::vector< std::shared_ptr< Matrix > > [X_mo](#) (void) const
retrieve the X operator O-V perturbation matrix in MO basis for all three Cartesian components
- std::shared_ptr< Matrix > [F_mo](#) (int x) const
retrieve the F operator O-V perturbation matrix in MO basis for x-th component
- std::vector< std::shared_ptr< Matrix > > [F_mo](#) (void) const
retrieve the F operator O-V perturbation matrix in MO basis for all three Cartesian components

- `std::shared_ptr< Matrix > T` (void) const
retrieve the transformation from old to new MO's
- `std::shared_ptr< Matrix > Cocc` (void) const
retrieve the Cocc (always Canonical)
- `std::shared_ptr< Matrix > Cvir` (void) const
retrieve the Cvir
- `std::shared_ptr< Vector > epsocc` (void) const
retrieve the epsocc (always Canonical)
- `std::shared_ptr< Vector > epsvir` (void) const
retrieve the epsvir
- `std::shared_ptr< Vector > lmo_centroid` (int i) const
retrieve the i-th orbital (LMO) centroid
- `std::shared_ptr< Localizer > localizer` (void) const
retrieve the orbital localizer

Protected Attributes

Basic Data

- `std::shared_ptr< psi::Wavefunction > _wfn`
Wavefunction object.
- Options & `_options`
Options.
- `std::shared_ptr< BasisSet > _primary`
Primary Basis Set.
- `std::shared_ptr< Localizer > _localizer`
Orbital localizer.

Sizing Information

- `const int _no`
Number of occupied orbitals.
- `const int _nv`
Number of virtual orbitals.
- `const int _nn`
Number of basis functions.
- `long int _memory`
Memory.

Parameters of CPHF Calculations

- `int _maxiter`
Maximum number of iterations.
- `double _conv`
CPHF convergence threshold.
- `bool _with_diis`

whether use DIIS or not

- `const int _diis_dim`

Size of subspace.

Molecular Orbitals

- `std::shared_ptr< Matrix > _cocc`
Occupied orbitals.
- `std::shared_ptr< Matrix > _cvir`
Virtual orbitals.
- `std::shared_ptr< Vector > _eps_occ`
Occupied orbital energies.
- `std::shared_ptr< Vector > _eps_vir`
Virtual orbital energies.
- `std::shared_ptr< psi::Matrix > _T`
Transformation from old to new MO's.

DIIS Manager

- `std::vector< std::shared_ptr< oepdev::DIISManager > > _diis`
the DIIS managers for each perturbation operator x, y and z

Response Properties

- `std::shared_ptr< Matrix > _molecularPolarizability`
Total (molecular) polarizability tensor.
- `std::vector< std::shared_ptr< Vector > > _orbitalCentroids`
LMO centroids.
- `std::vector< std::shared_ptr< Matrix > > _orbitalPolarizabilities`
orbital-associated polarizability tensors
- `std::vector< std::vector< std::shared_ptr< Matrix > > > _orbitalChargeTransferPolarizabilities`
orbital-orbital charge-transfer polarizability tensors
- `std::vector< std::shared_ptr< Matrix > > _X_OV_ao_matrices`
Perturbation X Operator O->V matrices in AO basis.
- `std::vector< std::shared_ptr< Matrix > > _X_OV_mo_matrices`
Perturbation X Operator O->V matrices in MO basis.
- `std::vector< std::shared_ptr< Matrix > > _F_OV_mo_matrices`
Electric Field Operator O->V matrices in MO basis.

14.13.1 Detailed Description

Solves CPHF equations (now only for RHF wavefunction). Computes molecular and polarizabilities associated with the localized molecular orbitals (LMO).

Note

Useful options:

- CPHF_CONVER - convergence of CPHF. Default: 1e-8 (au)
- CPHF_CONVER - maximum number of iterations. Default: 50
- CPHF_DIIS - whether use DIIS or not. Default: true
- CPHF_DIIS_DIM - dimension of iterative subspace. Default: 3
- CPHF_LOCALIZE - localize the molecular orbitals? Default: true
- CPHF_LOCALIZER - set orbital localization method. Available: BOYS and PIPEK_MEZEY. Default: BOYS

14.13.2 Constructor & Destructor Documentation

CPHF()

```
oepdev::CPHF::CPHF (
    SharedWavefunction ref_wfn,
    Options & options )
```

Parameters

<i>ref_wfn</i>	reference HF wavefunction
<i>options</i>	set of Psi4 options

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

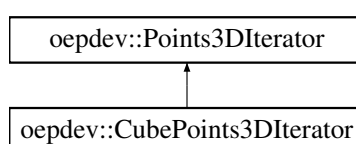
- oepdev/libutil/cphf.h
- oepdev/libutil/cphf.cc

14.14 oepdev::CubePoints3DIterator Class Reference

Iterator over a collection of points in 3D space. g09 Cube-like order.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::CubePoints3DIterator:



Public Member Functions

- **CubePoints3DIterator** (const int &nx, const int &ny, const int &nz, const double &dx, const double &dy, const double &dz, const double &ox, const double &oy, const double &oz)
Initialize first iteration.
- virtual void [first](#) ()
Step to next iteration.
- virtual void [next](#) ()
Step to next iteration.

Protected Attributes

- const int **nx_**
- const int **ny_**
- const int **nz_**
- const double **dx_**
- const double **dy_**
- const double **dz_**
- const double **ox_**
- const double **oy_**
- const double **oz_**
- int **ii_**
- int **jj_**
- int **kk_**

Additional Inherited Members

14.14.1 Detailed Description

Note: Always create instances by using static factory method from [Points3DIterator](#). Do not use constructor of this class.

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

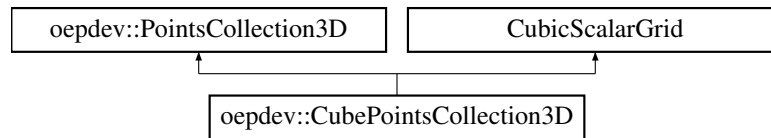
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.15 oepdev::CubePointsCollection3D Class Reference

G09 cube-like ordered collection of points in 3D space.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::CubePointsCollection3D:



Public Member Functions

- **CubePointsCollection3D** ([Collection](#) collectionType, const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedBasisSet bs, psi::Options &options)
- virtual void [print](#) () const
Print the information to Psi4 output file.
- virtual void **write_cube_file** (psi::SharedMatrix v, const std::string &name, const int &col=0)

Additional Inherited Members

14.15.1 Detailed Description

Note: Do not use constructors of this class explicitly. Instead, use static factory methods of the superclass to create instances.

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

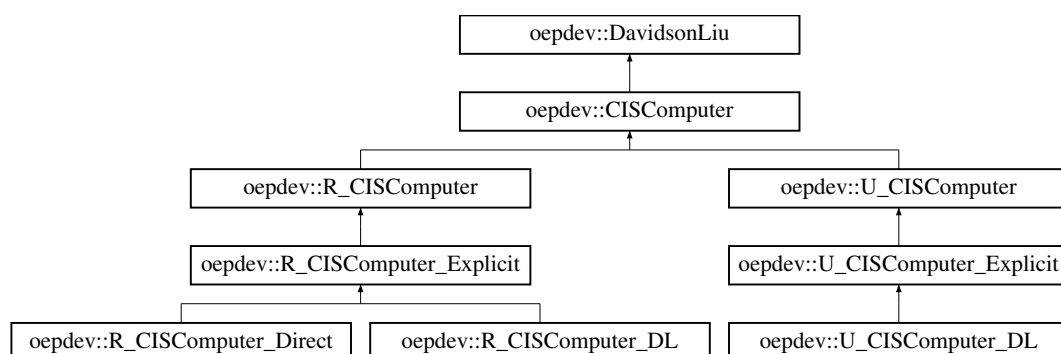
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.16 oepdev::DavidsonLiu Class Reference

Davidson-Liu diagonalization method.

```
#include <davidson_liu.h>
```

Inheritance diagram for oepdev::DavidsonLiu:



Public Member Functions

- [DavidsonLiu](#) (psi::Options &opt)
Constructor.
- virtual [~DavidsonLiu](#) ()
Destructor.
- virtual void [run_davidson_liu](#) ()
Run the Davidson-Liu solver.
- psi::SharedVector [eigenvalues_davidson_liu](#) () const
Get the eigenvalues.
- psi::SharedVector **E_davidson_liu** () const
- psi::SharedMatrix [eigenvectors_davidson_liu](#) () const
Get the eigenvectors.
- psi::SharedMatrix **U_davidson_liu** () const

Protected Member Functions

- virtual void [davidson_liu_initialize](#) (int N, int L, int M)
Helper interface.
- virtual void **davidson_liu_initialize_guess_vectors** ()
- virtual void **davidson_liu_initialize_guess_vectors_by_random** ()
- virtual void **davidson_liu_initialize_guess_vectors_by_custom** ()
- virtual void **davidson_liu_compute_diagonal_hamiltonian** ()=0
- virtual void **davidson_liu_compute_sigma** ()=0
- virtual void **davidson_liu_add_guess_vectors** ()
- virtual double **davidson_liu_compute_convergence** ()
- virtual void **davidson_liu_finalize** (bool)

Protected Attributes

- int [N_davidson_liu_](#)
Dimensionality of Hamiltonian.
- int [L_davidson_liu_](#)
Number of guess vectors.
- int [M_davidson_liu_](#)
Number of roots of interest.
- psi::Options & [options_](#)
Psi4 options.
- psi::SharedVector [E_davidson_liu_](#)
Eigenvalues.
- psi::SharedMatrix [U_davidson_liu_](#)

Eigenvectors.

- `psi::SharedVector H_diag_davidson_liu_`
Diagonal elements of the matrix to diagonalize.
- `psi::SharedVector E_old_davidson_liu_`
Old estimation of eigenvalues.
- `bool davidson_liu_initialized_`
Is Davidson-Liu computer ready for calculations?
- `bool davidson_liu_finalized_`
Is Davidson-Liu computer finished with calculations?
- `int davidson_liu_n_sigma_computed_`
- `std::vector< psi::SharedVector > sigma_vectors_davidson_liu_`
Sigma vectors stored.
- `std::shared_ptr< oepdev::GramSchmidt > guess_vectors_davidson_liu_`
Object storing guess vectors.

14.16.1 Detailed Description

Find the lowest M eigenvalues and associated eigenvectors of the real, symmetric (square) matrix \mathbf{H} .

Associated options:

- `DAVIDSON_LIU_NROOTS` - number of roots of interest. Default: 1.
- `DAVIDSON_LIU_CONVER` - convergence of the iterative procedure as RMS of old and current eigenvalues. Default: 1.0E-10.
- `DAVIDSON_LIU_MAXITER` - maximum number of iterations. Default: 500.
- `DAVIDSON_LIU_GUESS` - Type of guess vectors. Default: RANDOM, which is constructing random vectors.
- `DAVIDSON_LIU_THRESH_LARGE` - Small correction vector threshold (see description below). Default: 1.0E-03.
- `DAVIDSON_LIU_THRESH_SMALL` - Small correction vector threshold (see description below). Default: 1.0E-06.
- `DAVIDSON_LIU_SPACE_MAX` - Maximum number of guess vectors. Default: 200.
- `DAVIDSON_LIU_SPACE_START` - Starting amount of guess vectors. Must be larger or equal to number of roots. Default: -1, which means that number of roots is taken.
- `DAVIDSON_LIU_STOP_WHEN_UNCONVERGED` - Raise error when iterations do not converge. Default: True.

Usage in C++ programming

This class is an abstract base. In order to use the Davidson-Liu method fully implemented here, one must define a child class inheriting from [oepdev::DavidsonLiu](#) and implementing two of the pure methods:

- `davidson_liu_compute_diagonal_hamiltonian` - method specifying the calculation of the σ vectors, which are stored in the `std::vector<psi::SharedVector> sigma_vectors_davidson_liu;`
- `davidson_liu_compute_diagonal_hamiltonian` - method specifying the calculation of the diagonal elements of the Hamiltonian, stored in the `psi::SharedVector H_diag_davidson_liu.`

See also

Examples for demo use.

Implementation

The implementation follows Figure 5, Section 3.2.1 in Ref.[1]. Dimensionality:

- N - number of rows/columns of matrix to diagonalize
- L - current number of guess vectors
- M - number of roots of interest

Sigma vectors are defined to be

$$\mathbf{S} = \mathbf{H}\mathbf{B}$$

where \mathbf{B} are the guess vectors stored as a matrix of size (N, L) in core memory. Subspace Hamiltonian is then given by

$$\mathbf{G} = \mathbf{B}^T \mathbf{S}$$

and is diagonalized using standard diagonalization technique,

$$\mathbf{G} = \mathbf{U}\mathbf{z}\mathbf{U}^T$$

where \mathbf{z} are the eigenvalues. First M lowest eigenvalues and associated eigenvectors are saved in \mathbf{E} and \mathbf{A} , respectively (with the latter having size of (L, M)). The current eigenvector matrix \mathbf{C} containing roots is given by

$$\mathbf{C} = \mathbf{B}\mathbf{A}$$

Once this step is completed, the correction vectors are computed for each eigenvalue according to

$$\delta_{Ik} = \frac{1}{E_k - H_{II}} \left[-E_k C_{Ik} + \sum_l^L \sigma_{Il} A_{lk} \right]$$

and they are orthonormalized against all the columns of \mathbf{B} by using the Gram-Schmidt procedure. If the norm of such orthonormalized correction vector is larger than threshold value, it is appended to \mathbf{B} as new guess vector.

Note

Note that the current implementation uses the original Davidson's preconditioner, which might have problems with breaking spin symmetry of the solution.

Treatment of correction vector threshold.

In the current implementation, two threshold values are defined:

- larger threshold, controlled by `DAVIDSON_LIU_THRESH_LARGE` Psi4 option, is used for the first lowest eigenvalue.
- smaller threshold, controlled by `DAVIDSON_LIU_THRESH_SMALL` Psi4 option, is used for the next eigenvalues if $M > 1$.

References

[1] C. David Sherrill and Henry F. Schaefer III, *Adv. Quant. Chem.* **1999** (34), pp. 94720-1460.

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

- `oepdev/libutil/davidson_liu.h`
- `oepdev/libutil/davidson_liu.cc`

14.17 oepdev::DIISManager Class Reference

DIIS manager.

```
#include <diis.h>
```

Public Member Functions

- [DIISManager](#) (int dim, int na, int nb)
- [~DIISManager](#) ()
Destructor.
- void [put](#) (const std::shared_ptr< const Matrix > &error, const std::shared_ptr< const Matrix > &vector)
- void [compute](#) (void)
- void [update](#) (std::shared_ptr< Matrix > &other)

14.17.1 Detailed Description

Instance can interact directly with the process of solving vector quantities in iterative manner. One needs to pass the dimensions of solution vector as well as the DIIS subspace size. The iterative procedure requires providing the current vector and also an estimate of the error vector. The updated DIIS vector can be copied to an old vector through the Instance.

14.17.2 Constructor & Destructor Documentation

DIISManager()

```
oepdev::DIISManager::DIISManager (
    int dim,
    int na,
    int nb )
```

Constructor.

Parameters

<i>dim</i>	Size of DIIS subspace
<i>na</i>	Number of solution rows
<i>nb</i>	Number of solution columns

14.17.3 Member Function Documentation

put()

```
void oepdev::DIISManager::put (
    const std::shared_ptr< const Matrix > & error,
    const std::shared_ptr< const Matrix > & vector )
```

Put the current solution to the DIIS manager.

Parameters

<i>error</i>	Shared matrix with current solution error
<i>vector</i>	Shared matrix with current solution vector

compute()

```
void oepdev::DIISManager::compute (
    void )
```

Perform DIIS interpolation.

update()

```
void oepdev::DIISManager::update (
    std::shared_ptr< Matrix > & other )
```

Update solution vector. Pass the Shared pointer to current solution. Then it will be overridden by the updated DIIS solution.

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

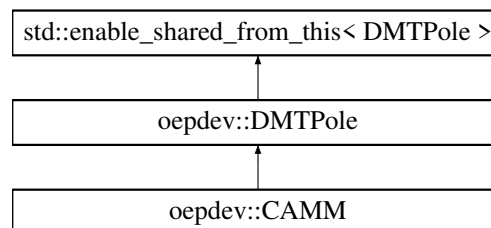
- oepdev/libutil/[diis.h](#)
- oepdev/libutil/[diis.cc](#)

14.18 oepdev::DMTPole Class Reference

Distributed Multipole Analysis Container and Computer. Abstract Base.

```
#include <dmp.h>
```

Inheritance diagram for oepdev::DMTPole:



Public Member Functions

Accessors

- virtual bool [has_charges](#) () const
Has distributed charges?
- virtual bool [has_dipoles](#) () const
Has distributed dipoles?
- virtual bool [has_quadrupoles](#) () const
Has distributed quadrupoles?
- virtual bool [has_octupoles](#) () const
Has distributed octupoles?
- virtual bool [has_hexadecapoles](#) () const
Has distributed hexadecapoles?
- virtual psi::SharedMatrix [centres](#) () const
Get the positions of distribution centres.
- virtual psi::SharedMatrix [origins](#) () const
Get the positions of distribution origins.
- virtual psi::SharedVector [centre](#) (int x) const

- Get the position of the *x*th distribution centre.*
- virtual psi::SharedVector [origin](#) (int x) const
- Get the position of the *x*th distribution origin.*
- virtual std::vector< psi::SharedMatrix > [charges](#) () const
- Get the distributed charges.*
- virtual std::vector< psi::SharedMatrix > [dipoles](#) () const
- Get the distributed dipoles.*
- virtual std::vector< psi::SharedMatrix > [quadrupoles](#) () const
- Get the distributed quadrupoles.*
- virtual std::vector< psi::SharedMatrix > [octupoles](#) () const
- Get the distributed octupoles.*
- virtual std::vector< psi::SharedMatrix > [hexadecapoles](#) () const
- Get the distributed hexadecapoles.*
- virtual psi::SharedMatrix [charges](#) (int i) const
- Get the distributed charges for the ith distribution.*
- virtual psi::SharedMatrix [dipoles](#) (int i) const
- Get the distributed dipoles for the ith distribution.*
- virtual psi::SharedMatrix [quadrupoles](#) (int i) const
- Get the distributed quadrupoles for the ith distribution.*
- virtual psi::SharedMatrix [octupoles](#) (int i) const
- Get the distributed octupoles for the ith distribution.*
- virtual psi::SharedMatrix [hexadecapoles](#) (int i) const
- Get the distributed hexadecapoles for the ith distribution.*
- virtual int [n_sites](#) () const
- Get the number of distributed sites.*
- virtual int [n_dmtip](#) () const
- Get the number of distributions.*

Mutators

- void [set_charges](#) (std::vector< psi::SharedMatrix > M)
- Set the distributed charges.*
- void [set_dipoles](#) (std::vector< psi::SharedMatrix > M)
- Set the distributed dipoles.*
- void [set_quadrupoles](#) (std::vector< psi::SharedMatrix > M)
- Set the distributed quadrupoles.*
- void [set_octupoles](#) (std::vector< psi::SharedMatrix > M)
- Set the distributed octupoles.*
- void [set_hexadecapoles](#) (std::vector< psi::SharedMatrix > M)
- Set the distributed hexadecapoles.*
- void [set_charges](#) (psi::SharedMatrix M, int i)
- Set the distributed charges for the ith distribution.*
- void [set_dipoles](#) (psi::SharedMatrix M, int i)
- Set the distributed dipoles for the ith distribution.*
- void [set_quadrupoles](#) (psi::SharedMatrix M, int i)
- Set the distributed quadrupoles for the ith distribution.*
- void [set_octupoles](#) (psi::SharedMatrix M, int i)

Set the distributed octupoles for the i th distribution.

- void [set_hexadecapoles](#) (psi::SharedMatrix M, int i)
Set the distributed hexadecapoles for the i th distribution.

Transformators

- virtual void [recenter](#) (psi::SharedMatrix new_origins)
Change origins of the distributed multipole moments of all sets.
- void [translate](#) (psi::SharedVector transl)
Translate the DMTP sets.
- void [rotate](#) (psi::SharedMatrix rotmat)
Rotate the DMTP sets.
- double [superimpose](#) (psi::SharedMatrix ref_xyz, std::vector< int > suplist={})
Superimpose the DMTP sets.

Computers

- void [compute](#) (std::vector< psi::SharedMatrix > D, std::vector< bool > t)
Compute DMTP's from the set of the one-particle density matrices.
- void [compute](#) (void)
Compute ground state DMTP.
- std::shared_ptr< [MultipoleConvergence](#) > [energy](#) (std::shared_ptr< [DMTPole](#) > other, [MultipoleConvergence::ConvergenceLevel](#) max_clevel=[MultipoleConvergence::R5](#))
Evaluate the generalized interaction energy.
- std::shared_ptr< [MultipoleConvergence](#) > [potential](#) (const double &x, const double &y, const double &z, [MultipoleConvergence::ConvergenceLevel](#) max_clevel=[MultipoleConvergence::R5](#))
Evaluate the generalized potential at a given point.
- std::shared_ptr< [MultipoleConvergence](#) > [field](#) (const double &x, const double &y, const double &z, [MultipoleConvergence::ConvergenceLevel](#) max_clevel=[MultipoleConvergence::R5](#))
Evaluate the generalized field at a given point.

Printers

- virtual void [print_header](#) () const =0
Print the header.
- void [print](#) () const
Print the contents.

Static Public Member Functions

- static [MultipoleConvergence::ConvergenceLevel](#) [determine_dmtip_convergence_level](#) (const std::string &option)

Protected Member Functions

Protected Interface

- [DMTPole](#) (std::shared_ptr< psi::Wavefunction > wfn, int n)
Construct an empty DMTP object from the wavefunction.
- virtual void [compute](#) (psi::SharedMatrix D, bool transition, int i)
Compute DMTP's from the one-particle density matrix.
- void [compute_integrals](#) ()
Compute multipole integrals.
- void [compute_order](#) ()
Compute maximum order of the integrals.
- virtual void [recenter](#) (psi::SharedMatrix new_origins, int i)
Change origins of the distributed multipole moments of ith set.
- virtual void [allocate](#) ()
Initialize and allocate memory.
- virtual void [copy_from](#) (const [DMTPole](#) *)
Deep-copy the matrix and DMTP data.

Protected Attributes

Basic

- std::string [name_](#)
Name of the distribution method.
- psi::SharedMolecule [mol_](#)
Molecule associated with this DMTP.
- psi::SharedWavefunction [wfn_](#)
Wavefunction associated with this DMTP.
- psi::SharedBasisSet [primary_](#)
Basis set (primary)
- std::vector< psi::SharedMatrix > [mplnts_](#)
Multipole integrals.

Sizing

- int [nDMTPs_](#)
Number of DMTP's.
- int [nSites_](#)
Number of DMTP sites.
- int [order_](#)
Maximum order of the multipole.

Descriptors

- bool [hasCharges_](#)
Has distributed charges?

- bool [hasDipoles_](#)
Has distributed dipoles?
- bool [hasQuadrupoles_](#)
Has distributed quadrupoles?
- bool [hasOctupoles_](#)
Has distributed octupoles?
- bool [hasHexadecapoles_](#)
Has distributed hexadecapoles?

Geometry

- psi::SharedMatrix [centres_](#)
DMTP centres.
- psi::SharedMatrix [origins_](#)
DMTP origins.

Multipoles

- std::vector< psi::SharedMatrix > [charges_](#)
DMTP charges.
- std::vector< psi::SharedMatrix > [dipoles_](#)
DMTP dipoles.
- std::vector< psi::SharedMatrix > [quadrupoles_](#)
DMTP quadrupoles.
- std::vector< psi::SharedMatrix > [octupoles_](#)
DMTP octupoles.
- std::vector< psi::SharedMatrix > [hexadecapoles_](#)
DMTP hexadecapoles.

Friends

- class [MultipoleConvergence](#)

Constructors and Destructor

- static std::shared_ptr< [DMTPole](#) > [build](#) (const std::string &type, std::shared_ptr< psi::Wavefunction > wfn, int n=1)
Build an empty DMTP object from the wavefunction.
- static std::shared_ptr< [DMTPole](#) > [empty](#) (std::string type)
Build an empty DMTP object of no type.
- [DMTPole](#) (void)
Construct an empty DMTP object of no type.
- [DMTPole](#) (const [DMTPole](#) *)
Copy constructor.

- virtual std::shared_ptr< [DMTPole](#) > [clone](#) (void) const =0
*Make a deep copy (*wfn*_, *mol*_, and *primary*_ are shallow-copied)*
- virtual [~DMTPole](#) ()
Destructor.

14.18.1 Detailed Description

Handles the distributed multipole expansions up to hexadecapoles. Distributed centres as well as DMTP origins are allowed to be located in arbitrary points in space. The object describes a set of N DMTP's, that can be generated by providing one-particle density matrices in AO basis. Nuclear contributions can be switched on or off separately for each DMTP within a set. The following operations on the DMTP sets are available through the API:

- translation
- rotation
- superimposition
- recentering the origins
- computing the generalized property from another DMTP set

See also

[MultipoleConvergence](#)

14.18.2 Constructor & Destructor Documentation

DMTPole() [1/2]

```
oepdev::DMTPole::DMTPole (
    void )
```

Do not use this constructor. Use the [DMTPole::empty](#) method.

DMTPole() [2/2]

```
oepdev::DMTPole::DMTPole (
    std::shared_ptr< psi::Wavefunction > wfn,
    int n ) [protected]
```

Parameters

<i>wfn</i>	- wavefunction
<i>n</i>	- number of DMTP sets

Do not use this constructor. Use the [DMTPole::build](#) method.

14.18.3 Member Function Documentation

build()

```
std::shared_ptr< DMTPole > oepdev::DMTPole::build (
    const std::string & type,
    std::shared_ptr< psi::Wavefunction > wfn,
    int n = 1 ) [static]
```

Parameters

<i>type</i>	- DMTP method. Available: CAMM .
<i>wfn</i>	- wavefunction
<i>n</i>	- number of DMTP sets

Returns

DMTP distribution

empty()

```
std::shared_ptr< DMTPole > oepdev::DMTPole::empty (
    std::string type ) [static]
```

Returns

Blank DMTP distribution with memory allocated by no data.

determine_dmtp_convergence_level()

```
MultipoleConvergence::ConvergenceLevel oepdev::DMTPole::determine_dmtp_convergence_level
(
    const std::string & option ) [static]
```

Determine the [CAMM](#) convergence for a given global option

Parameters

<i>option</i>	- string for option
---------------	---------------------

recenter()

```
void oepdev::DMTPole::recenter (
    psi::SharedMatrix new_origins ) [virtual]
```

Parameters

<i>new_origins</i>	- matrix with coordinates of the new origins $\{\mathbf{r}_{\text{new}}\}$.
--------------------	--

Note

The number of origins has to be equal to the number of distributed centres.

Recentering of the multipoles affects the distributed dipoles and higher moments. The moments are given as

$$\begin{aligned}
 q_{\text{new}} &= q_{\text{old}} \\
 \boldsymbol{\mu}_{\text{new}} &= \boldsymbol{\mu}_{\text{old}} - q_{\text{old}} \Delta^{(1)} \\
 \boldsymbol{\Theta}_{\text{new}} &= \boldsymbol{\Theta}_{\text{old}} + q_{\text{old}} \Delta^{(2)} - \sum_{\mathcal{P}_2} \mathcal{P}_2 \left[(q_{\text{old}} \mathbf{r}_{\text{old}} + \boldsymbol{\mu}_{\text{old}}) \otimes \Delta^{(1)} \right] \\
 \boldsymbol{\Omega}_{\text{new}} &= \boldsymbol{\Omega}_{\text{old}} - q_{\text{old}} \Delta^{(3)} + \sum_{\mathcal{P}_3} \mathcal{P}_3 \left[(q_{\text{old}} \mathbf{r}_{\text{old}} + \boldsymbol{\mu}_{\text{old}}) \otimes \Delta^{(2)} \right] - \sum_{\mathcal{P}_6} \mathcal{P}_6 \left[(q_{\text{old}} \mathbf{r}_{\text{old}}^2 + \boldsymbol{\mu}_{\text{old}} \otimes \mathbf{r}_{\text{old}} + \boldsymbol{\Theta}_{\text{old}}) \otimes \Delta^{(1)} \right] \\
 \boldsymbol{\Xi}_{\text{new}} &= \boldsymbol{\Xi}_{\text{old}} + q_{\text{old}} \Delta^{(4)} - \sum_{\mathcal{P}_3} \mathcal{P}_3 \left[(q_{\text{old}} \mathbf{r}_{\text{old}} + \boldsymbol{\mu}_{\text{old}}) \otimes \Delta^{(3)} \right] + \sum_{\mathcal{P}_3} \mathcal{P}_3 \left[(q_{\text{old}} \mathbf{r}_{\text{old}}^2 + \boldsymbol{\mu}_{\text{old}} \otimes \mathbf{r}_{\text{old}} + \boldsymbol{\Theta}_{\text{old}}) \otimes \Delta^{(2)} \right] \\
 &\quad - \sum_{\mathcal{P}_3} \mathcal{P}_3 \left[(q_{\text{old}} \mathbf{r}_{\text{old}}^3 + \boldsymbol{\mu}_{\text{old}} \otimes \mathbf{r}_{\text{old}}^2 + \boldsymbol{\Theta}_{\text{old}} \otimes \mathbf{r}_{\text{old}} + \boldsymbol{\Omega}_{\text{old}}) \otimes \Delta^{(1)} \right]
 \end{aligned}$$

where

$$\begin{aligned}
 \Delta^{(1)} &\equiv \mathbf{r}_{\text{new}} - \mathbf{r}_{\text{old}} \\
 \Delta^{(2)} &\equiv \mathbf{r}_{\text{new}}^2 - \mathbf{r}_{\text{old}}^2 \\
 \Delta^{(3)} &\equiv \mathbf{r}_{\text{new}}^3 - \mathbf{r}_{\text{old}}^3 \\
 \Delta^{(4)} &\equiv \mathbf{r}_{\text{new}}^4 - \mathbf{r}_{\text{old}}^4
 \end{aligned}$$

In the above equations, the distributed centre label was omitted (redundant) as each distributed site of multipoles is independent of the others. TODO - Finish for octupoles and hexadecapoles! -> define the permutation operators!

rotate()

```
void oepdev::DMTPole::rotate (
    psi::SharedMatrix rotmat )
```

Parameters

<i>rotmat</i>	- Cartesian rotation matrix r
---------------	--------------------------------------

Centers and origins, as well as dipole, quadrupole, octupole and hexadecapole moments are transformed according to:

$$\begin{aligned}
 x_a^{(i)} &\rightarrow \sum_{a'} x_{a'}^{(i)} r_{a'a} \\
 o_a^{(i)} &\rightarrow \sum_{a'} o_{a'}^{(i)} r_{a'a} \\
 \mu_a^{(i)} &\rightarrow \sum_{a'} \mu_{a'}^{(i)} r_{a'a} \\
 \Theta_a^{(i)} &\rightarrow \sum_{a'b'} \Theta_{a'b'}^{(i)} r_{a'a} r_{b'b} \\
 \Omega_a^{(i)} &\rightarrow \sum_{a'b'c'} \Omega_{a'b'c'}^{(i)} r_{a'a} r_{b'b} r_{c'c} \\
 \Xi_a^{(i)} &\rightarrow \sum_{a'b'c'd'} \Xi_{a'b'c'd'}^{(i)} r_{a'a} r_{b'b} r_{c'c} r_{d'd}
 \end{aligned}$$

where the definition of $r_{a'a}$ is consistent with the Kabsch algorithm implemented in [KabschSuperimposer](#).

See also

[KabschSuperimposer](#)

superimpose()

```
double oepdev::DMTPole::superimpose (
    psi::SharedMatrix ref_xyz,
    std::vector< int > suplist = {} )
```

Parameters

<i>ref_xyz</i>	- target geometry to superimpose
<i>suplist</i>	- superimposition list

Returns

the RMS of superimposition Kabsch algorithm is used for superimposition.

See also

[KabschSuperimposer](#)

compute() [1/2]

```
void oepdev::DMTPole::compute (
    std::vector< psi::SharedMatrix > D,
    std::vector< bool > t )
```

Parameters

<i>D</i>	- list of one-particle density matrices
<i>t</i>	- list of flags determining if density is of transition type or not

compute() [2/2]

```
void oepdev::DMTPole::compute (
    void )
```

Compute DMTP's from the *sum* of the ground-state alpha and beta one-particle density matrices (t=false, i=0). Results in a usual DMTP analysis of a molecule's charge density distribution.

energy()

```
std::shared_ptr< MultipoleConvergence > oepdev::DMTPole::energy (
    std::shared_ptr< DMTPole > other,
    MultipoleConvergence::ConvergenceLevel max_clevel = MultipoleConvergence::R5
)
```

Parameters

<i>other</i>	- interacting DMTP distribution.
<i>max_clevel</i>	- maximum convergence level (see below).

Returns

The generalized interaction energy convergence (A.U. units)

The following convergence levels are available:

- `MultipoleConvergence::R1`: includes qq terms.
- `MultipoleConvergence::R2`: includes dq terms and above.
- `MultipoleConvergence::R3`: includes qQ, dd terms and above.
- `MultipoleConvergence::R4`: includes qO, dQ terms and above.
- `MultipoleConvergence::R5`: includes qH, dO, QQ terms and above.

potential()

```
std::shared_ptr< MultipoleConvergence > oepdev::DMTPole::potential (
    const double & x,
    const double & y,
    const double & z,
    MultipoleConvergence::ConvergenceLevel max_clevel = MultipoleConvergence::R5
)
```

Parameters

<i>x</i>	- location x-th Cartesian component
<i>y</i>	- location y-th Cartesian component
<i>z</i>	- location z-th Cartesian component
<i>max_clevel</i>	- maximum convergence level (see below).

Returns

The generalized potential convergence (A.U. units)

The following convergence levels are available:

- `MultipoleConvergence::R1`: includes qq terms.
- `MultipoleConvergence::R2`: includes dq terms and above.
- `MultipoleConvergence::R3`: includes qQ, dd terms and above.
- `MultipoleConvergence::R4`: includes qO, dQ terms and above.
- `MultipoleConvergence::R5`: includes qH, dO, QQ terms and above.

field()

```
std::shared_ptr< MultipoleConvergence > oepdev::DMTPole::field (
    const double & x,
    const double & y,
    const double & z,
    MultipoleConvergence::ConvergenceLevel max_clevel = MultipoleConvergence::R5
)
```

Parameters

<i>x</i>	- location x-th Cartesian component
<i>y</i>	- location y-th Cartesian component
<i>z</i>	- location z-th Cartesian component
<i>max_clevel</i>	- maximum convergence level (see below).

Returns

The generalized field convergence (A.U. units)

The following convergence levels are available:

- `MultipoleConvergence::R1`: includes qq terms.
- `MultipoleConvergence::R2`: includes dq terms and above.
- `MultipoleConvergence::R3`: includes qQ, dd terms and above.
- `MultipoleConvergence::R4`: includes qO, dQ terms and above.
- `MultipoleConvergence::R5`: includes qH, dO, QQ terms and above.

14.18.4 Friends And Related Function Documentation

MultipoleConvergence

```
friend class MultipoleConvergence [friend]
```

Convergence of multipole moment series.

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

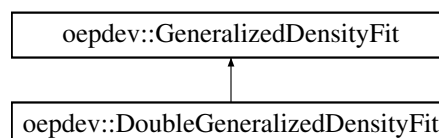
- `oepdev/lib3d/dmtp.h`
- `oepdev/lib3d/dmtp_base.cc`

14.19 oepdev::DoubleGeneralizedDensityFit Class Reference

Generalized Density Fitting Scheme - Double Fit.

```
#include <oep_gdf.h>
```

Inheritance diagram for `oepdev::DoubleGeneralizedDensityFit`:



Public Member Functions

- **DoubleGeneralizedDensityFit** (std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::BasisSet > bs_intermediate, std::shared_ptr< psi::Matrix > v_vector)
- std::shared_ptr< psi::Matrix > **compute** (void)

Perform the generalized density fit.

Additional Inherited Members

14.19.1 Detailed Description

The density fitting map projects the OEP onto an arbitrary (not necessarily complete) auxiliary basis set space through application of the self energy minimization technique. The resulting three-electron repulsion integrals are computed by utilizing the resolution of identity in an intermediate, nearly-complete basis set space, hence performing an internal density fitting in nearly complete basis. Refer to [density fitting specialized for OEP's](#) for more details.

14.19.2 Determination of the OEP matrix

Coefficients \mathbf{G} are computed by using the following relation

$$\mathbf{G} = \mathbf{A}^{-1} \cdot \mathbf{R} \cdot \mathbf{H}$$

where the intermediate projection matrix is given by

$$\mathbf{H} = \mathbf{S}^{-1} \cdot \mathbf{V}$$

In the above equations,

$$\begin{aligned} A_{\xi\xi'} &= (\xi||\xi') \\ R_{\xi\varepsilon} &= (\xi||\varepsilon) \\ S_{\varepsilon\varepsilon'} &= (\varepsilon|\varepsilon') \\ V^{\varepsilon i} &= (\varepsilon|\hat{v}i) \end{aligned}$$

The following labeling convention is used here:

- i denotes the arbitrary state vector
- ξ denotes the auxiliary basis set element
- ε denotes the intermediate (nearly complete) basis set element

In the above, $|$ denotes the single integration over electron coordinate, i.e.,

$$(a|b) \equiv \int d\mathbf{r} \phi_a^*(\mathbf{r}) \phi_b(\mathbf{r})$$

whereas $||$ acts as shown below:

$$(a||b) \equiv \iint d\mathbf{r}' d\mathbf{r}'' \frac{\phi_a^*(\mathbf{r}') \phi_b(\mathbf{r}'')}{|\mathbf{r}' - \mathbf{r}''|}$$

The spatial form of the potential operator \hat{v} can be expressed by

$$v(\mathbf{r}) \equiv \int d\mathbf{r}' \frac{\rho(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|}$$

with $\rho(\mathbf{r})$ being the effective one-electron density associated with \hat{v} .

Theory behind the double GDF scheme

In order to perform the generalized density fitting in an incomplete auxiliary basis set, one must apply the following formula:

$$\mathbf{G} = \mathbf{A}^{-1} \cdot \mathbf{B}$$

where one encounters the need of evaluation of the following *three-electron integrals*

$$B_{\xi i} = (\xi || \hat{v} i) \equiv \iiint d\mathbf{r}' d\mathbf{r}'' d\mathbf{r}''' \phi_{\xi}^*(\mathbf{r}') \frac{1}{|\mathbf{r}' - \mathbf{r}''|} \rho(\mathbf{r}''') \frac{1}{|\mathbf{r}''' - \mathbf{r}''|} \phi_i(\mathbf{r}'')$$

Computation of all the necessary integrals of this kind is very costly and impractical for larger molecules. However, one can use the same trick that is a kernel of the OEP technique introduced in the OEPDev project, i.e., introduce the effective potential in order to get rid of one integration. This can be done by performing the generalized density fitting in the nearly complete intermediate basis

$$\hat{v}|i) \cong \sum_{\varepsilon} H_{\varepsilon i} |\varepsilon)$$

Note that this is done just for the sake of factorizing the triple integral and computing the OEP matrix for the incomplete auxiliary basis. Therefore, the intermediate basis set is used just for a while during density fitting and is no longer necessary later on. By inserting the above identity to the triple integral one can transform it into a sum of the two-electron integrals that are much easier to evaluate. This leads to equations given in the beginning of this section.

14.19.3 Member Function Documentation

compute()

```
std::shared_ptr< psi::Matrix > DoubleGeneralizedDensityFit::compute (
    void ) [virtual]
```

Returns

The OEP coefficients G_{ξ_i}

Implements [oepdev::GeneralizedDensityFit](#).

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

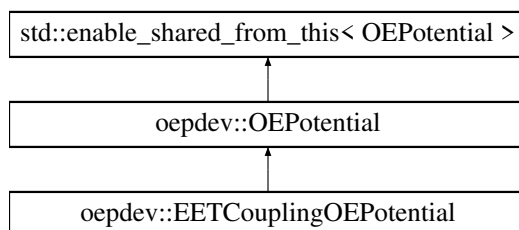
- oepdev/liboep/[oep_gdf.h](#)
- oepdev/liboep/oep_gdf.cc

14.20 oepdev::EETCouplingOEPotential Class Reference

Generalized One-Electron Potential for EET coupling calculations.

```
#include <oep.h>
```

Inheritance diagram for oepdev::EETCouplingOEPotential:



Public Member Functions

- **EETCouplingOEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)
- **EETCouplingOEPotential** (SharedWavefunction [wfn](#), Options &options)
- **EETCouplingOEPotential** (const [EETCouplingOEPotential](#) *f)
- virtual void [compute](#) (const std::string &oepType) override
Compute matrix forms of all OEP's within a specified OEP type.
- virtual void [compute_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared_ptr< psi::Vector > &v) override
Compute value of potential in point x, y, z and save at v.
- virtual void [print_header](#) () const override
Header information.
- virtual std::shared_ptr< [OEPotential](#) > [clone](#) (void) const override
Make a deep copy of this object.
- virtual void [initialize](#) () override
Initialize the object (expert)

Protected Member Functions

- virtual void **rotate_oep** (psi::SharedMatrix, psi::SharedMatrix, psi::SharedMatrix) override
- virtual void **translate_oep** (psi::SharedVector) override

Additional Inherited Members

14.20.1 Detailed Description

Contains the following OEP types:

- Fujimoto.GDF - Joint OEP type for ET(L), ET(HL), HT(H) and HT(HL)
- Fujimoto.CIS - CIS data
- Fujimoto.EXCH- Pure-exchange coupling matrix $G_{\mu\nu} \equiv (\mu\mu|vv)$
- Fujimoto.CT_M- ($HH|LL$) integral for the H_34 Hamiltonian matrix elements (CT)

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

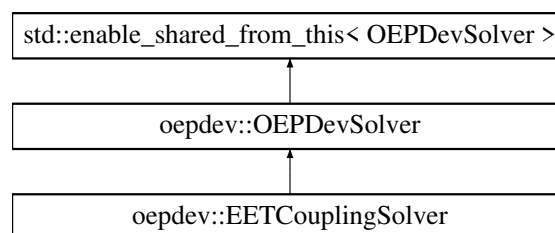
- oepdev/liboep/oep.h
- oepdev/liboep/oep_coupling_eet.cc

14.21 oepdev::EETCouplingSolver Class Reference

Compute the EET coupling energy between unperturbed wavefunctions.

```
#include <solver.h>
```

Inheritance diagram for oepdev::EETCouplingSolver:



Public Member Functions

- **EETCouplingSolver** (SharedWavefunctionUnion wfn_union)
- virtual double **compute_oep_based** (const std::string &method="DEFAULT")
Compute property by using OEPs.
- virtual double **compute_benchmark** (const std::string &method="DEFAULT")
Compute property by using benchmark method.

Additional Inherited Members

14.21.1 Detailed Description

The implemented methods are shown below

Table 14.32: Methods available in the Solver

Keyword	Method Description
Benchmark Methods	
FUJIMOTO_TI_CIS	<i>Default.</i> EET Coupling by Fujimoto JPC 2012.
OEP-Based Methods	
FUJIMOTO_TI_CIS	<i>Default.</i> OEP-based TI/CIS expressions.

In order to construct this solver, **always** use the `OEPDevSolver::build` static factory method.

Below the detailed description of the implemented equations is given for each of the above provided methods. In the formulae across, it is assumed that the orbitals are real. The Coulomb notation for electron repulsion integrals (ERIs) is adopted; i.e,

$$(ac|bd) = \iint d\mathbf{r}_1 d\mathbf{r}_2 \phi_a(\mathbf{r}_1) \phi_c(\mathbf{r}_1) \frac{1}{r_{12}} \phi_b(\mathbf{r}_2) \phi_d(\mathbf{r}_2)$$

Greek subscripts denote basis set orbitals whereas Italic subscripts denote the occupied molecular orbitals.

Benchmark Methods

TI/CIS Method (Fujimoto JPC 2012).

In the simplest version of TI/CIS approach, the Hamiltonian of the molecular aggregate (dimer) is constructed from the CIS approximation and 4 basis functions constructed as follows:

$$\begin{aligned} |\Phi_1\rangle &= |\Psi_A^{(e)} \otimes \Psi_B^{(g)}\rangle \\ |\Phi_2\rangle &= |\Psi_A^{(g)} \otimes \Psi_B^{(e)}\rangle \\ |\Phi_3\rangle &= |\Psi_A^{(+)} \otimes \Psi_B^{(-)}\rangle \\ |\Phi_4\rangle &= |\Psi_A^{(-)} \otimes \Psi_B^{(+)}\rangle \end{aligned}$$

where *g* and *e* superscripts denote the ground and excited state of a molecule, + and – label the cationic and anionic state, respectively, whereas $|\Psi_X \otimes \Psi_Y\rangle$ denotes the antisymmetrized

Hartree product of the monomer wavefunctions. The associated diagonal Hamiltonian matrix elements can be defined as

$$\begin{aligned}\langle \Phi_1 | \mathcal{H} - E_0 | \Phi_1 \rangle &\equiv E_1 = E_{e \rightarrow g}^A + \sum_{\mu \nu \in A} \left(P_{\nu \mu}^{A(e)} - P_{\nu \mu}^{A(g)} \right) \times \left\{ V_{\mu \nu}^{B(\text{nuc})} + \sum_{\lambda \sigma \in B} P_{\lambda \sigma}^{B(g)} \left[(\mu \nu | \sigma \lambda) - \frac{1}{2} (\mu \lambda | \sigma \nu) \right] \right\} \\ \langle \Phi_2 | \mathcal{H} - E_0 | \Phi_2 \rangle &\equiv E_2 = E_{e \rightarrow g}^B + \sum_{\mu \nu \in B} \left(P_{\nu \mu}^{B(e)} - P_{\nu \mu}^{B(g)} \right) \times \left\{ V_{\mu \nu}^{A(\text{nuc})} + \sum_{\lambda \sigma \in A} P_{\lambda \sigma}^{A(g)} \left[(\mu \nu | \sigma \lambda) - \frac{1}{2} (\mu \lambda | \sigma \nu) \right] \right\} \\ \langle \Phi_3 | \mathcal{H} - E_0 | \Phi_3 \rangle &\equiv E_3 = -\varepsilon_H^A + \varepsilon_L^B - (H^A H^A | L^B L^B) \\ \langle \Phi_4 | \mathcal{H} - E_0 | \Phi_4 \rangle &\equiv E_4 = \varepsilon_L^A - \varepsilon_H^B - (L^A L^A | H^B H^B)\end{aligned}$$

The associated off-diagonal Hamiltonian matrix elements can be defined as

$$\begin{aligned}\langle \Phi_1 | \mathcal{H} | \Phi_2 \rangle &\equiv V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovrl}} \\ \langle \Phi_1 | \mathcal{H} | \Phi_3 \rangle &\equiv V^{\text{ET1}} \\ \langle \Phi_2 | \mathcal{H} | \Phi_4 \rangle &\equiv V^{\text{ET2}} \\ \langle \Phi_1 | \mathcal{H} | \Phi_4 \rangle &\equiv V^{\text{HT1}} \\ \langle \Phi_2 | \mathcal{H} | \Phi_3 \rangle &\equiv V^{\text{HT2}} \\ \langle \Phi_3 | \mathcal{H} | \Phi_4 \rangle &\equiv V^{\text{CT}}\end{aligned}$$

where the Forster-type Coulombic (Coul), Dexter-type exchange (Exch), remaining overlap correction (Ovrl), as well as the electron, hole and charge (ET, HT, CT) transfer contributions are defined. The exchange-Coulomb coupling takes the form

$$\begin{aligned}V^{\text{Coul}} &= \frac{V^{\text{Coul},(0)}}{1 - S_{12}^2} \\ V^{\text{Exch}} &= \frac{V^{\text{Exch},(0)}}{1 - S_{12}^2} \\ V^{\text{Ovrl}} &= -\frac{(E_1 + E_2)S_{12}}{2(1 - S_{12}^2)}\end{aligned}$$

The overlap-corrected ET, HT and CT matrix elements read

$$\begin{aligned}V^{\text{ET1}} &= [1 - S_{13}^2]^{-1} \left\{ V^{\text{ET1},(0)} - \frac{1}{2} (E_1 + E_2) S_{13} \right\} \\ V^{\text{ET2}} &= [1 - S_{24}^2]^{-1} \left\{ V^{\text{ET2},(0)} - \frac{1}{2} (E_1 + E_2) S_{24} \right\} \\ V^{\text{HT1}} &= [1 - S_{14}^2]^{-1} \left\{ V^{\text{HT1},(0)} - \frac{1}{2} (E_1 + E_2) S_{14} \right\} \\ V^{\text{HT2}} &= [1 - S_{23}^2]^{-1} \left\{ V^{\text{HT2},(0)} - \frac{1}{2} (E_1 + E_2) S_{23} \right\} \\ V^{\text{CT}} &= [1 - S_{34}^2]^{-1} \left\{ V^{\text{CT},(0)} - \frac{1}{2} (E_1 + E_2) S_{34} \right\}\end{aligned}$$

In the above equations, the superscript (0) denotes that the matrix elements are not affected by the overlap between molecular wavefunctions, and are given by

$$\begin{aligned}
 V^{\text{Coul},(0)} &= \sum_{\mu\nu\in A} \sum_{\lambda\sigma\in B} P_{\nu\mu}^{g\rightarrow e(A)} P_{\lambda\sigma}^{g\rightarrow e(B)} (\mu\nu|\sigma\lambda) \\
 V^{\text{Exch},(0)} &= -\frac{1}{2} \sum_{\mu\nu\in A} \sum_{\lambda\sigma\in B} P_{\nu\mu}^{g\rightarrow e(A)} P_{\lambda\sigma}^{g\rightarrow e(B)} (\mu\lambda|\sigma\nu) \\
 V^{\text{ET1},(0)} &= t_{H\rightarrow L}^A \left\{ (L^A|\mathcal{F}|L^B) + 2(L^A H^A|H^A L^B) - (L^A L^B|H^A H^A) \right\} \\
 V^{\text{ET2},(0)} &= t_{H\rightarrow L}^B \left\{ (L^A|\mathcal{F}|L^B) + 2(L^A H^B|H^B L^B) - (L^A L^B|H^B H^B) \right\} \\
 V^{\text{HT1},(0)} &= t_{H\rightarrow L}^A \left\{ -(H^A|\mathcal{F}|H^B) + 2(H^A L^A|L^A H^B) - (H^A H^B|L^A L^A) \right\} \\
 V^{\text{HT2},(0)} &= t_{H\rightarrow L}^B \left\{ -(H^A|\mathcal{F}|H^B) + 2(H^A L^B|L^B H^B) - (H^A H^B|L^B L^B) \right\} \\
 V^{\text{CT},(0)} &= 2(H^A L^B|L^A H^B) - (H^A H^B|L^A L^B)
 \end{aligned}$$

In the above, \mathcal{F} is the Fock operator whereas H and L denote the HOMO and LUMO orbitals, respectively. The overlap integrals between the basis states are approximated by

$$\begin{aligned}
 S_{12} &\equiv (\Phi_1|\Phi_2) \cong -\frac{1}{N_{el}^{AB}} \text{Tr} \left[\mathbf{p}^{g\rightarrow e(A)} \mathbf{s}^{AB} \mathbf{p}^{g\rightarrow e(B)} \mathbf{s}^{BA} \right] \\
 S_{13} &\equiv (\Phi_1|\Phi_3) \cong -\frac{t_{H\rightarrow L}^A}{N_{el}^{AB}} S_{LL}^{AB} \\
 S_{14} &\equiv (\Phi_1|\Phi_4) \cong +\frac{t_{H\rightarrow L}^A}{N_{el}^{AB}} S_{HH}^{AB} \\
 S_{24} &\equiv (\Phi_2|\Phi_4) \cong -\frac{t_{H\rightarrow L}^B}{N_{el}^{AB}} S_{LL}^{AB} \\
 S_{23} &\equiv (\Phi_2|\Phi_3) \cong +\frac{t_{H\rightarrow L}^B}{N_{el}^{AB}} S_{HH}^{AB} \\
 S_{34} &\equiv (\Phi_3|\Phi_4) \cong -\frac{1}{N_{el}^{AB}} S_{HH}^{AB} S_{LL}^{AB}
 \end{aligned}$$

where the overlap between molecular orbitals U and W is given by

$$S_{UW}^{AB} \equiv \mathbf{s}^{AB} : \mathbf{c}_U^A \otimes \mathbf{c}_W^B$$

and \mathbf{s}^{AB} is the AO overlap matrix between molecule A and B atomic basis functions.

For a closed-shell system, the EET coupling constant for two electronic transitions can be given approximately by

$$V \approx V^{\text{Direct}} + V^{\text{Indirect}}$$

where the overlap-corrected direct and indirect coupling constants are

$$\begin{aligned}
 V^{\text{Direct}} &= V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovrl}} \\
 V^{\text{Indirect}} &= V^{\text{TI-2}} + V^{\text{TI-3}}
 \end{aligned}$$

with

$$V^{\text{TI-2}} = -\frac{V^{\text{ET1}}V^{\text{HT2}}}{E_3 - E_1} - \frac{V^{\text{ET2}}V^{\text{HT1}}}{E_4 - E_1}$$

$$V^{\text{TI-3}} = \frac{V^{\text{CT}}(V^{\text{ET1}}V^{\text{ET2}} + V^{\text{HT1}}V^{\text{HT2}})}{(E_3 - E_1)(E_4 - E_1)}$$

Fock matrix in AB space

In the current implementation, Fock matrix in the AB space, that is necessary to evaluate ET and HT matrix elements, can be defined as

1. the AB block of full Hartree-Fock SCF Fock matrix for entire system;
2. the zeroth-order Fock matrix that is composed of monomer's unperturbed ground-state 1-particle density matrices.

In the latter case, the Fock matrix in AO representation is given by:

$$F_{\alpha \in A, \beta \in B}^{AB} \approx T_{\alpha\beta} + V_{\alpha\beta}^{A(\text{nuc})} + V_{\alpha\beta}^{B(\text{nuc})} + \sum_{\mu \nu \in A} P_{\nu\mu}^{A(g)} G_{\alpha\beta, \mu\nu} + \sum_{\sigma \lambda \in B} P_{\lambda\sigma}^{B(g)} G_{\alpha\beta, \sigma\lambda}$$

where

$$G_{\alpha\beta, \gamma\delta} \equiv (\alpha\beta|\gamma\delta) - \frac{1}{2}(\alpha\delta|\gamma\beta)$$

Mulliken approximated exchange-like contributions.

Exchange and CT contributions require ERIs of type (AB,AB). It is instructive to approximate these contributions in terms of the Coulomb-like ERIs for the sake of testing of OEP-based approximations which are given in the next Section.

Application of the Mulliken approximation

$$(ij|kl) \approx \frac{1}{4} S_{ij} S_{kl} [(ii|kk) + (jj|kk) + (ii|ll) + (jj|ll)]$$

results in the following approximations to the exchange-like terms

$$V^{\text{Exch},(0)} \approx -\frac{1}{8} \sum_{\mu \nu \in A} \sum_{\lambda \sigma \in B} P_{\nu\mu}^{g \rightarrow e(A)} P_{\lambda\sigma}^{g \rightarrow e(B)} S_{\mu\lambda} S_{\sigma\nu} [(\mu\mu|\sigma\sigma) + (\lambda\lambda|\nu\nu) + (\mu\mu|\nu\nu) + (\lambda\lambda|\sigma\sigma)]$$

$$V^{\text{CT},(0)} \approx \frac{1}{2} S_{HL}^{AB} S_{LH}^{AB} [r_{HL}^A + r_{HL}^B + \rho_H^A \odot \rho_H^B + \rho_L^A \odot \rho_L^B]$$

$$- \frac{1}{4} S_{HH}^{AB} S_{LL}^{AB} [r_{HL}^A + r_{HL}^B + \rho_H^A \odot \rho_L^B + \rho_L^A \odot \rho_H^B]$$

The former can be rewritten in a more convenient to implement formula:

$$V^{\text{Exch},(0)} \approx -\frac{1}{4} \sum_{\mu \in A} \sum_{\nu \in B} (\mu\mu|\sigma\sigma) [\mathbf{P}^A \mathbf{s}^{AB}]_{\mu\sigma} [\mathbf{P}^B \mathbf{s}^{BA}]_{\sigma\mu} - \frac{1}{8} \sum_{\mu \nu \in A} P_{\nu\mu}^A (\mu\mu|\nu\nu) [\mathbf{s}^{AB} \mathbf{P}^B \mathbf{s}^{BA}]_{\mu\nu}$$

$$- \frac{1}{8} \sum_{\sigma \lambda \in B} P_{\lambda\sigma}^B (\lambda\lambda|\sigma\sigma) [\mathbf{s}^{BA} \mathbf{P}^A \mathbf{s}^{AB}]_{\sigma\lambda}$$

In the CT term,

$$\begin{aligned} r_{HL}^A &\equiv \rho_H^A \odot \rho_L^A \\ r_{HL}^B &\equiv \rho_H^B \odot \rho_L^B \end{aligned}$$

where the effective Coulombic interaction energies are defined by

$$\rho_U^A \odot \rho_W^B \equiv (U^A U^A | W^A W^A)$$

OEP-Based Methods

OEP method of interfragment ERI elimination applied to the direct Coulombic coupling yields the TrCAMM coupling. Direct exchange coupling requires using the Mulliken approximation. The TI contributions can be treated after certain fragmentation of the Fock operator of a dimer is assumed. In that case, OEP method yields the complete TI/CIS model. [Błasiak et al. \[2020b\]](#)

OEP-Based TI/CIS theory

Application of the OEP method yields the following expressions for the off-diagonal Hamiltonian matrix elements:

$$\begin{aligned} V^{\text{ET1},(0)} &= t_{H \rightarrow L}^A \left\{ \sum_{\zeta \in A} S_{\zeta L}^{AB} V_{\zeta;HL}^{A;ET} + \sum_{\eta \in B} S_{\eta L}^{BA} V_{\eta;L}^{B;ET} \right\} \\ V^{\text{ET2},(0)} &= t_{H \rightarrow L}^B \left\{ \sum_{\zeta \in A} S_{\zeta L}^{AB} V_{\zeta;L}^{A;ET} + \sum_{\eta \in B} S_{\eta L}^{BA} V_{\eta;HL}^{B;ET} \right\} \\ V^{\text{HT1},(0)} &= t_{H \rightarrow L}^A \left\{ \sum_{\zeta \in A} S_{\zeta H}^{AB} V_{\zeta;HL}^{A;HT} + \sum_{\eta \in B} S_{\eta H}^{BA} V_{\eta;H}^{B;HT} \right\} \\ V^{\text{HT2},(0)} &= t_{H \rightarrow L}^B \left\{ \sum_{\zeta \in A} S_{\zeta H}^{AB} V_{\zeta;H}^{A;HT} + \sum_{\eta \in B} S_{\eta H}^{BA} V_{\eta;HL}^{B;HT} \right\} \\ V^{\text{CT},(0)} &= \frac{1}{2} S_{HL}^{AB} S_{LH}^{AB} \left\{ r_{HL}^A + r_{HL}^B + \frac{1}{|\mathbf{r}_H^A - \mathbf{r}_H^B|} + \sum_{x \in A} \sum_{y \in B} \frac{q_{x,L}^A q_{y,L}^B}{|\mathbf{r}_x - \mathbf{r}_y|} \right\} \\ &\quad - \frac{1}{4} S_{HH}^{AB} S_{LL}^{AB} \left\{ r_{HL}^A + r_{HL}^B - \sum_{y \in B} \frac{q_{y,L}^B}{|\mathbf{r}_H^A - \mathbf{r}_y|} - \sum_{x \in A} \frac{q_{x,L}^A}{|\mathbf{r}_H^B - \mathbf{r}_x|} \right\} \end{aligned}$$

where the charge centroids are given by $\mathbf{r}_Y^X \equiv (Y^X | \hat{\mathbf{r}} | Y^X)$. The OEP matrices can be written in a density fitting form utilizing a nearly-complete auxiliary basis set as

$$V_{\zeta;N}^{X;M} = \sum_{\eta \in X} [\mathbf{S}^{-1}]_{\zeta\eta} a_{\eta;N}^{X;M}$$

for N = H, L or HL and M = ET1, ET2, HT1 or HT2, respectively. The explicit formulae for the auxiliary vectors **a** are as follows:

$$\begin{aligned}
 a_{\alpha;L}^{X;ET} &= \sum_{\beta} C_{\beta L}^X Q_{\alpha\beta}^X \\
 a_{\alpha;HL}^{X;ET} &= a_{\alpha;L}^{X;ET} + \sum_{\beta\gamma\delta} (\alpha\beta|\gamma\delta) \left\{ 2C_{\beta H}^X C_{\gamma L}^X - C_{\beta L}^X C_{\gamma H}^X \right\} C_{\delta H}^X \\
 a_{\alpha;H}^{X;HT} &= - \sum_{\beta} C_{\beta H}^X Q_{\alpha\beta}^X \\
 a_{\alpha;HL}^{X;HT} &= a_{\alpha;H}^{X;HT} + \sum_{\beta\gamma\delta} (\alpha\beta|\gamma\delta) \left\{ 2C_{\beta L}^X C_{\gamma H}^X - C_{\beta H}^X C_{\gamma L}^X \right\} C_{\delta L}^X
 \end{aligned}$$

The auxiliary matrix **Q** can be found by inverting the following matrix equation

$$\sum_{\beta} C_{\beta Y}^X Q_{\alpha\beta}^X = \sum_{\beta} C_{\beta Y}^X \left\{ \frac{1}{2} T_{\alpha\beta} + V_{\text{nuc};\alpha\beta}^X \right\} + \sum_{\beta\gamma\delta} (\alpha\beta|\gamma\delta) \left\{ P_{\delta\gamma}^{X(g)} C_{\beta Y}^X - \frac{1}{2} P_{\beta\gamma}^{X(g)} C_{\delta Y}^X \right\}$$

The matrices **C** are the LCAO-MO matrices in primary AO basis.

Direct exchange

Neglecting interfragment ERIs from the Mulliken-approximated direct exchange coupling results in the following expression:

$$V^{\text{Exch.}(0)} \approx -\frac{1}{8} \sum_{\mu\nu \in A} P_{\nu\mu}^A (\mu\mu|\nu\nu) [s^{AB} \mathbf{P}^B s^{BA}]_{\mu\nu} - \frac{1}{8} \sum_{\sigma\lambda \in B} P_{\lambda\sigma}^B (\lambda\lambda|\sigma\sigma) [s^{BA} \mathbf{P}^A s^{AB}]_{\sigma\lambda}$$

which enables implementing it within the EFP methodology. Note that here, the ERIs $(\mu\mu|\nu\nu)$ refer only to one fragment, therefore they constitute a two-index EFP parameters which are relatively inexpensive to handle.

The OEP-based model also approximates the diagonal Hamiltonian matrix elements. The environmental contributions are usually small and are neglected here. The HOMO-LUMO interaction is approximated by using the DMTP approximation. Hence, the diagonal Hamiltonian matrix elements read approximately:

$$\begin{aligned}
 \langle \Phi_1 | \mathcal{H} - E_0 | \Phi_1 \rangle &\equiv E_1 \cong E_{e \rightarrow g}^A \\
 \langle \Phi_2 | \mathcal{H} - E_0 | \Phi_2 \rangle &\equiv E_2 \cong E_{e \rightarrow g}^B \\
 \langle \Phi_3 | \mathcal{H} - E_0 | \Phi_3 \rangle &\equiv E_3 \cong -\varepsilon_H^A + \varepsilon_L^B - \rho_H^A \odot \rho_L^B \\
 \langle \Phi_4 | \mathcal{H} - E_0 | \Phi_4 \rangle &\equiv E_4 \cong \varepsilon_L^A - \varepsilon_H^B - \rho_L^A \odot \rho_H^B
 \end{aligned}$$

where the ' \odot ' symbol is treated via the CAMM expansion, Sokalski and Poirier [1983] or the LMTP expansion Etchebest et al. [1982] truncated on charges.

14.21.2 Member Function Documentation

compute_oep_based()

```
double EETCouplingSolver::compute_oep_based (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one `DEFAULT` OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implements [oepdev::OEPDevSolver](#).

compute_benchmark()

```
double EETCouplingSolver::compute_benchmark (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one `DEFAULT` benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implements [oepdev::OEPDevSolver](#).

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

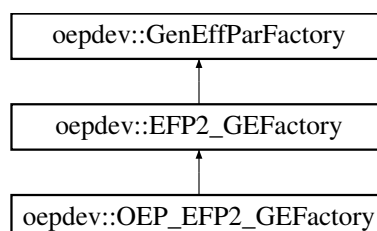
- [oepdev/libsolver/solver.h](#)
- [oepdev/libsolver/solver_coupling_eet.cc](#)

14.22 oepdev::EFP2_GEFactory Class Reference

EFP2 GEFP Factory.

```
#include <gefp.h>
```

Inheritance diagram for `oepdev::EFP2_GEFactory`:



Public Member Functions

- [EFP2_GEFactory](#) (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
Construct from Psi4 options.
- virtual [~EFP2_GEFactory](#) ()
Destruct.
- virtual std::shared_ptr< [GenEffPar](#) > [compute](#) (void)
Compute the EFP2 parameters.

Protected Member Functions

- virtual std::shared_ptr< [oepdev::DMTPole](#) > [compute_dmtip](#) (void)
- virtual void [compute_lmoc](#) (void)
- virtual std::shared_ptr< [oepdev::CPHF](#) > [compute_cphf](#) (void)
- virtual std::shared_ptr< [oepdev::QUAMBO](#) > [compute_quambo](#) (void)
- virtual void [assemble_efp2_parameters](#) (void)
- virtual void [assemble_geometry_data](#) (void)
- virtual void [assemble_dmtip_data](#) (void)
- virtual void [assemble_lmo_centroids](#) (void)
- virtual void [assemble_fock_matrix](#) (void)
- virtual void [assemble_canonical_orbitals](#) (void)
- virtual void [assemble_distributed_polarizabilities](#) (void)

Protected Attributes

- std::shared_ptr< [oepdev::GenEffPar](#) > [EFP2Parameters_](#)

Additional Inherited Members

14.22.1 Detailed Description

Basic interface for the EFP2 parameters.

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

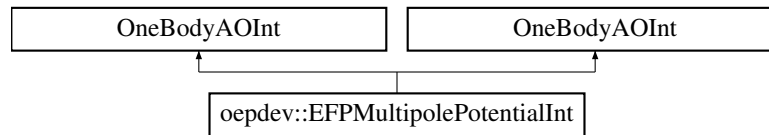
- [oepdev/libgefp/gefp.h](#)
- [oepdev/libgefp/gefp_efp2.cc](#)

14.23 oepdev::EFPMultipolePotentialInt Class Reference

Computes potential integrals.

```
#include <multipole_potential.h>
```

Inheritance diagram for oepdev::EFPMultipolePotentialInt:



Public Member Functions

- [EFPMultipolePotentialInt](#) (std::vector< psi::SphericalTransform > &, std::shared_ptr< psi::BasisSet >, std::shared_ptr< psi::BasisSet >, int max_k=3, int deriv=0)
Constructor. Do not call directly use an [IntegralFactory](#).
- [~EFPMultipolePotentialInt](#) () override
Virtual destructor.
- [EFPMultipolePotentialInt](#) (std::vector< psi::SphericalTransform > &, std::shared_ptr< psi::BasisSet >, std::shared_ptr< psi::BasisSet >, int max_k=3, int deriv=0)
Constructor. Do not call directly use an [IntegralFactory](#).
- [~EFPMultipolePotentialInt](#) () override
Virtual destructor.

Protected Member Functions

- void [compute_pair](#) (const psi::GaussianShell &, const psi::GaussianShell &) override
Computes the electric field between two gaussian shells.
- void [compute_pair](#) (const psi::GaussianShell &, const psi::GaussianShell &) override
Computes the electric field between two gaussian shells.

Protected Attributes

- oepdev::ObaraSaikaTwoCenterMultipolePotentialRecursion **mvi_recur_**
- int **max_k_**
- [oepdev::ObaraSaikaTwoCenterEFPRecursion.New](#) **mvi_recur_**
- bool **do_octupoles_**
- int **nchunk_**

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

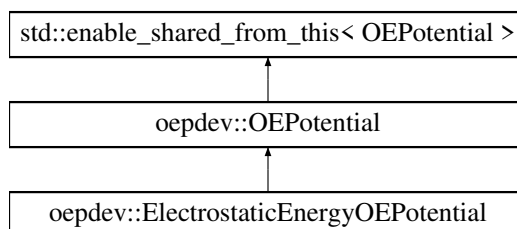
- oepdev/libpsi/bck/multipole_potential.h
- oepdev/libpsi/bck/multipole_potential.cc

14.24 oepdev::ElectrostaticEnergyOEPotential Class Reference

Generalized One-Electron Potential for Electrostatic Energy.

```
#include <oep.h>
```

Inheritance diagram for oepdev::ElectrostaticEnergyOEPotential:



Public Member Functions

- [ElectrostaticEnergyOEPotential](#) (SharedWavefunction [wfn](#), Options &options)
Only ESP-based potential is worth implementing.
- **ElectrostaticEnergyOEPotential** (const [ElectrostaticEnergyOEPotential](#) *f)
- virtual void [compute](#) (const std::string &oepType) override
Compute matrix forms of all OEP's within a specified OEP type.
- virtual void [compute_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared_ptr< psi::Vector > &v) override
Compute value of potential in point x, y, z and save at v.
- virtual void [print_header](#) () const override
Header information.
- virtual std::shared_ptr< [OEPotential](#) > [clone](#) (void) const override
Make a deep copy of this object.
- virtual void [initialize](#) () override
Initialize the object (expert)

Protected Member Functions

- virtual void **rotate_oep** (psi::SharedMatrix r, psi::SharedMatrix R_prim, psi::SharedMatrix R_aux) override
- virtual void **translate_oep** (psi::SharedVector t) override

Additional Inherited Members

14.24.1 Detailed Description

Contains the following OEP types:

- [V](#)

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

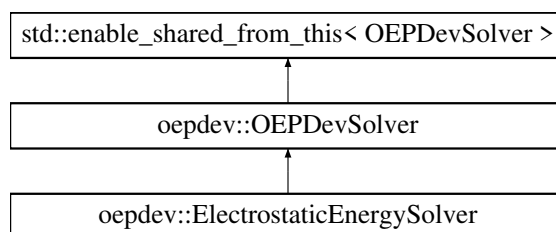
- [oepdev/liboep/oep.h](#)
- [oepdev/liboep/oep_energy_coul.cc](#)

14.25 oepdev::ElectrostaticEnergySolver Class Reference

Compute the Coulombic interaction energy between unperturbed wavefunctions.

```
#include <solver.h>
```

Inheritance diagram for oepdev::ElectrostaticEnergySolver:



Public Member Functions

- **ElectrostaticEnergySolver** ([SharedWavefunctionUnion](#) wfn_union)
- virtual double [compute_oep_based](#) (const std::string &method="DEFAULT")
Compute property by using OEPs.
- virtual double [compute_benchmark](#) (const std::string &method="DEFAULT")
Compute property by using benchmark method.

Additional Inherited Members

14.25.1 Detailed Description

The implemented methods are shown in below

Table 14.35: Methods available in the Solver

Keyword	Method Description
Benchmark Methods	
AO_EXPANDED	<i>Default.</i> Exact Coulombic energy from atomic orbital expansions.
MO_EXPANDED	Exact Coulombic energy from molecular orbital expansions

Keyword	Method Description
OEP-Based Methods	
ESP_SYMMETRIZED	<i>Default.</i> Coulombic energy from ESP charges interacting with nuclei and electronic density. Symmetrized with respect to monomers.
CAMM	Coulombic energy from CAMM distributions.

Below the detailed description of the above methods is given.

Benchmark Methods

Exact Coulombic energy from atomic orbital expansions.

The Coulombic interaction energy is given by

$$E^{\text{Coul}} = E^{\text{Nuc-Nuc}} + E^{\text{Nuc-El}} + E^{\text{El-El}}$$

where the nuclear-nuclear repulsion energy is

$$E^{\text{Nuc-Nuc}} = \sum_{x \in A} \sum_{y \in B} \frac{Z_x Z_y}{|\mathbf{r}_x - \mathbf{r}_y|}$$

the nuclear-electronic attraction energy is

$$E^{\text{Nuc-El}} = \sum_{x \in A} \sum_{\lambda \sigma \in B} Z_x V_{\lambda \sigma}^{(x)} \left(D_{\lambda \sigma}^{(\alpha)} + D_{\lambda \sigma}^{(\beta)} \right) + \sum_{y \in B} \sum_{\mu \nu \in A} Z_y V_{\mu \nu}^{(y)} \left(D_{\mu \nu}^{(\alpha)} + D_{\mu \nu}^{(\beta)} \right)$$

and the electron-electron repulsion energy is

$$E^{\text{El-El}} = \sum_{\mu \nu \in A} \sum_{\lambda \sigma \in B} \left\{ D_{\mu \nu}^{(\alpha)} + D_{\mu \nu}^{(\beta)} \right\} \left\{ D_{\lambda \sigma}^{(\alpha)} + D_{\lambda \sigma}^{(\beta)} \right\} (\mu \nu | \lambda \sigma)$$

In the above equations,

$$V_{\lambda \sigma}^{(x)} \equiv \int \frac{\phi_{\lambda}^*(\mathbf{r}) \phi_{\sigma}(\mathbf{r})}{|\mathbf{r} - \mathbf{r}_x|} d\mathbf{r}$$

Exact Coulombic energy from molecular orbital expansion.

This approach is fully equivalent to the atomic orbital expansion shown above. For the closed shell case, the Coulombic interaction energy is given by

$$E^{\text{Coul}} = E^{\text{Nuc-Nuc}} + E^{\text{Nuc-El}} + E^{\text{El-El}}$$

where the nuclear-nuclear repulsion energy is

$$E^{\text{Nuc-Nuc}} = \sum_{x \in A} \sum_{y \in B} \frac{Z_x Z_y}{|\mathbf{r}_x - \mathbf{r}_y|}$$

the nuclear-electronic attraction energy is

$$E^{\text{Nuc-EI}} = 2 \sum_{i \in A} \sum_{y \in B} V_{ii}^{(y)} + 2 \sum_{j \in B} \sum_{x \in A} V_{jj}^{(x)}$$

and the electron-electron repulsion energy is

$$E^{\text{EI-EI}} = 4 \sum_{i \in A} \sum_{j \in B} (ii|jj)$$

OEP-Based Methods

Coulombic energy from ESP charges interacting with nuclei and electronic density.

In this approach, nuclear and electronic density of either species is approximated by ESP charges. In order to achieve symmetric expression, the interaction is computed twice (ESP of A interacting with density matrix and nuclear charges of B and vice versa) and then divided by 2. Thus,

$$E^{\text{Coul}} \approx \frac{1}{2} \left[\sum_{x \in A} \sum_{y \in B} \frac{Z_x q_y}{|\mathbf{r}_x - \mathbf{r}_y|} + \sum_{y \in B} \sum_{\mu \nu \in A} q_y V_{\mu \nu}^{(y)} \left(D_{\mu \nu}^{(\alpha)} + D_{\mu \nu}^{(\beta)} \right) \right. \\ \left. + \sum_{y \in B} \sum_{x \in A} \frac{q_x Z_y}{|\mathbf{r}_x - \mathbf{r}_y|} + \sum_{x \in A} \sum_{\lambda \sigma \in B} q_x V_{\lambda \sigma}^{(x)} \left(D_{\lambda \sigma}^{(\alpha)} + D_{\lambda \sigma}^{(\beta)} \right) \right]$$

If the basis set is large and the number of ESP centres $q_{x(y)}$ is sufficient, the sum of first two contributions equals the sum of the latter two contributions.

Notes:

- This solver also computes and prints the ESP-ESP point charge interaction energy,

$$E^{\text{Coul,ESP}} \approx \sum_{x \in A} \sum_{y \in B} \frac{q_x q_y}{|\mathbf{r}_x - \mathbf{r}_y|}$$

for reference purposes.

- In order to construct this solver, **always** use the `OEPDevSolver::build` static factory method.

14.25.2 Member Function Documentation

`compute_oep_based()`

```
double ElectrostaticEnergySolver::compute_oep_based (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one `DEFAULT` OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implements [oepdev::OEPDevSolver](#).

compute_benchmark()

```
double ElectrostaticEnergySolver::compute_benchmark (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one `DEFAULT` benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implements [oepdev::OEPDevSolver](#).

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

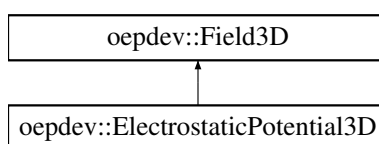
- oepdev/libsolver/[solver.h](#)
- oepdev/libsolver/solver_energy_coul.cc

14.26 oepdev::ElectrostaticPotential3D Class Reference

Electrostatic potential of a molecule.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::ElectrostaticPotential3D:



Public Member Functions

- **ElectrostaticPotential3D** (const int &np, const double &padding, psi::SharedWavefunction [wfn](#), psi::Options &options)
- **ElectrostaticPotential3D** (const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedWavefunction [wfn](#), psi::Options &options)
- virtual std::shared_ptr< psi::Vector > [compute_xyz](#) (const double &x, const double &y, const double &z)

Compute a value of 3D field at point (x, y, z)

- virtual void `print ()` const

Print information of the object to Psi4 output.

Additional Inherited Members

14.26.1 Detailed Description

Computes the electrostatic potential of a molecule directly from the wavefunction. The electrostatic potential $v(\mathbf{r})$ at point \mathbf{r} is computed from the following formula:

$$v(\mathbf{r}) = v_{\text{nuc}}(\mathbf{r}) + v_{\text{el}}(\mathbf{r})$$

where the nuclear and electronic contributions are defined accordingly as

$$v_{\text{nuc}}(\mathbf{r}) = \sum_x \frac{Z_x}{|\mathbf{r} - \mathbf{r}_x|}$$

$$v_{\text{el}}(\mathbf{r}) = \sum_{\mu\nu} \left\{ D_{\mu\nu}^{(\alpha)} + D_{\mu\nu}^{(\beta)} \right\} V_{\nu\mu}(\mathbf{r})$$

In the above equations, Z_x denotes the charge of x th nucleus, $D_{\mu\nu}^{(\omega)}$ is the one-particle (relaxed) density matrix element in AO basis associated with the ω electron spin, and $V_{\mu\nu}(\mathbf{r})$ is the potential one-electron integral defined by

$$V_{\nu\mu}(\mathbf{r}) \equiv \int d\mathbf{r}' \phi_{\nu}^*(\mathbf{r}') \frac{1}{|\mathbf{r} - \mathbf{r}'|} \phi_{\mu}(\mathbf{r}')$$

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

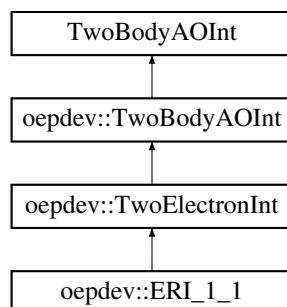
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.27 oepdev::ERI_1_1 Class Reference

2-centre ERI of the form $(a|O(2)|b)$ where $O(2) = 1/r_{12}$.

```
#include <eri.h>
```

Inheritance diagram for oepdev::ERI_1_1:



Public Member Functions

- [ERI_1_1](#) (const [IntegralFactory](#) *integral, int deriv=0, bool use_shell_pairs=false)
Constructor. Use [oepdev::IntegralFactory](#) to generate this object.
- [~ERI_1_1](#) ()
Destructor.

Protected Member Functions

- size_t [compute_doublet](#) (int, int)
Compute ERI's between 2 shells.

Protected Attributes

- double * [mdh_buffer_1_](#)
Buffer for McMurchie-Davidson-Hermite coefficients for monomial expansion (shell 1)
- double * [mdh_buffer_2_](#)
Buffer for McMurchie-Davidson-Hermite coefficients for monomial expansion (shell 2)

14.27.1 Detailed Description

ERI's are computed for a shell doublet (P|Q) and stored in the `target_full_buffer`, accessible through `buffer()` method:

$$\begin{aligned} &\text{For each } (n_1, l_1, m_1) \in P : \\ &\quad \text{For each } (n_2, l_2, m_2) \in Q : \\ &\quad \text{ERI} = (A|B)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] \end{aligned}$$

For detailed description of the McMurchie-Davidson scheme, refer to [The Integral Package Library](#).

14.27.2 Implementation

A set of ERI's in a shell is decontracted as

$$(A|B)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{ij} c_i(\alpha_1) c_j(\alpha_2) (i|j)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

where the primitive ERI is given by

$$(i|j)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{N_1=0}^{n_1} \sum_{L_1=0}^{l_1} \sum_{M_1=0}^{m_1} \sum_{N_2=0}^{n_2} \sum_{L_2=0}^{l_2} \sum_{M_2=0}^{m_2} d_{N_1}^{n_1} d_{L_1}^{l_1} d_{M_1}^{m_1} d_{N_2}^{n_2} d_{L_2}^{l_2} d_{M_2}^{m_2} [N_1 L_1 M_1 | N_2 L_2 M_2]$$

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

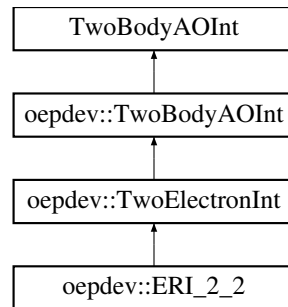
- [oepdev/libints/eri.h](#)
- [oepdev/libints/eri.cc](#)

14.28 oepdev::ERI_2_2 Class Reference

4-centre ERI of the form $(ab|O(2)|cd)$ where $O(2) = 1/r_{12}$.

```
#include <eri.h>
```

Inheritance diagram for oepdev::ERI_2_2:



Public Member Functions

- [ERI_2_2](#) (const [IntegralFactory](#) *integral, int deriv=0, bool use_shell_pairs=false)

Constructor. Use [oepdev::IntegralFactory](#) to generate this object.

- [~ERI_2_2](#) ()

Destructor.

Protected Member Functions

- size_t [compute_quartet](#) (int, int, int, int)

Compute ERI's between 4 shells.

Protected Attributes

- double * [mdh_buffer_12_](#)

Buffer for McMurchie-Davidson-Hermite coefficients for binomial expansion (shells 1 and 2)

- double * [mdh_buffer_34_](#)

Buffer for McMurchie-Davidson-Hermite coefficients for binomial expansion (shells 3 and 4)

14.28.1 Detailed Description

ERI's are computed for a shell quartet (PQ|RS) and stored in the `target_full_buffer`, accessible through `buffer()` method:

For each $(n_1, l_1, m_1) \in P$:
 For each $(n_2, l_2, m_2) \in Q$:
 For each $(n_3, l_3, m_3) \in R$:
 For each $(n_4, l_4, m_4) \in S$:
 $ERI = (AB|CD)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$

For detailed description of the McMurchie-Davidson scheme, refer to [The Integral Package Library](#).

14.28.2 Implementation

A set of ERI's in a shell is decontracted as

$$(AB|CD)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{ijkl} c_i(\alpha_1) c_j(\alpha_2) c_k(\alpha_3) c_l(\alpha_4) (ij|kl)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

where the primitive ERI is given by

$$(ij|kl)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = E_{ij}(\alpha_1, \alpha_2) E_{kl}(\alpha_3, \alpha_4) \\ \times \sum_{N_1=0}^{n_1+n_2} \sum_{L_1=0}^{l_1+l_2} \sum_{M_1=0}^{m_1+m_2} \sum_{N_2=0}^{n_3+n_4} \sum_{L_2=0}^{l_3+l_4} \sum_{M_2=0}^{m_3+m_4} d_{N_1}^{n_1 n_2} d_{L_1}^{l_1 l_2} d_{M_1}^{m_1 m_2} d_{N_2}^{n_3 n_4} d_{L_2}^{l_3 l_4} d_{M_2}^{m_3 m_4} [N_1 L_1 M_1 | N_2 L_2 M_2]$$

In the above equation, the multiplicative constants are given as

$$E_{ij}(\alpha_1, \alpha_2) = \exp \left[-\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \\ E_{kl}(\alpha_3, \alpha_4) = \exp \left[-\frac{\alpha_3 \alpha_4}{\alpha_3 + \alpha_4} |\mathbf{C} - \mathbf{D}|^2 \right]$$

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

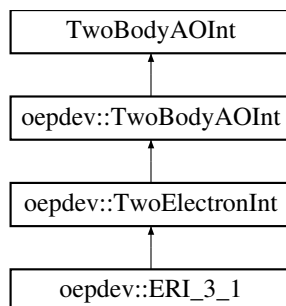
- oepdev/libints/[eri.h](#)
- oepdev/libints/eri.cc

14.29 oepdev::ERI_3_1 Class Reference

4-centre ERI of the form (abc|O(2)|d) where O(2) = 1/r12.

```
#include <eri.h>
```

Inheritance diagram for oepdev::ERI_3_1:



Public Member Functions

- [ERI_3_1](#) (const [IntegralFactory](#) *integral, int deriv=0, bool use_shell_pairs=false)
Constructor. Use [oepdev::IntegralFactory](#) to generate this object.
- [~ERI_3_1](#) ()
Destructor.

Protected Member Functions

- size_t [compute_quartet](#) (int, int, int, int)
Compute ERI's between 4 shells.

Protected Attributes

- double * [mdh_buffer_123_](#)
Buffer for McMurchie-Davidson-Hermite coefficients for trinomial expansion (shells 1, 2 and 3)
- double * [mdh_buffer_4_](#)
Buffer for McMurchie-Davidson-Hermite coefficients for monomial expansion (shell 4)

14.29.1 Detailed Description

ERI's are computed for a shell quartet (PQR|S) and stored in the `target_full_buffer`, accessible through `buffer()` method:

For each $(n_1, l_1, m_1) \in P$:
 For each $(n_2, l_2, m_2) \in Q$:
 For each $(n_3, l_3, m_3) \in R$:
 For each $(n_4, l_4, m_4) \in S$:

$$\text{ERI} = (ABC|D)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

For detailed description of the McMurchie-Davidson scheme, refer to [The Integral Package Library](#).

14.29.2 Implementation

A set of ERI's in a shell is decontracted as

$$(ABC|D)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = \sum_{ijkl} c_i(\alpha_1) c_j(\alpha_2) c_k(\alpha_3) c_l(\alpha_4) (ijk|l)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}]$$

where the primitive ERI is given by

$$(ijk|l)[\{\alpha\}, \mathbf{n}, \mathbf{l}, \mathbf{m}] = E_{ijk}(\alpha_1, \alpha_2, \alpha_3) \times \sum_{N_1=0}^{n_1+n_2+n_3} \sum_{L_1=0}^{l_1+l_2+l_3} \sum_{M_1=0}^{m_1+m_2+m_3} \sum_{N_2=0}^{n_4} \sum_{L_2=0}^{l_4} \sum_{M_2=0}^{m_4} d_{N_1}^{n_1 n_2 n_3} d_{L_1}^{l_1 l_2 l_3} d_{M_1}^{m_1 m_2 m_3} d_{N_2}^{n_4} d_{L_2}^{l_4} d_{M_2}^{m_4} [N_1 L_1 M_1 | N_2 L_2 M_2]$$

In the above equation, the multiplicative constants are given as

$$E_{ijk}(\alpha_1, \alpha_2, \alpha_3) = \exp \left[-\frac{\alpha_1 \alpha_2}{\alpha_1 + \alpha_2} |\mathbf{A} - \mathbf{B}|^2 \right] \exp \left[-\frac{(\alpha_1 + \alpha_2) \alpha_3}{\alpha_1 + \alpha_2 + \alpha_3} |\mathbf{P} - \mathbf{C}|^2 \right]$$

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

- oepdev/libints/eri.h
- oepdev/libints/eri.cc

14.30 oepdev::ESPSolver Class Reference

Charges from Electrostatic Potential (ESP). A solver-type class.

```
#include <esp.h>
```

Public Member Functions

- [ESPSolver](#) (SharedField3D field)
Construct from 3D vector field.
- [ESPSolver](#) (SharedField3D field, psi::SharedMatrix [centres](#))
Construct from 3D vector field.
- virtual [~ESPSolver](#) ()
Destructor.
- virtual psi::SharedMatrix [charges](#) () const
Get the (fit) charges.
- virtual psi::SharedMatrix [centres](#) () const
Get the charge distribution centres.
- virtual void [set_charge_sums](#) (psi::SharedVector s)
Set the charge sums Q_p .
- virtual void [set_charge_sums](#) (const double &s)
Set the charge sums Q_p (equal to all fields)
- virtual void [compute](#) ()
Perform fitting of effective charges.

Protected Attributes

- const int `nCentres_`
Number of fit centres.
- const int `nFields_`
Number of fields to fit.
- SharedField3D `field_`
Scalar field.
- psi::SharedMatrix `charges_`
Charges to be fit.
- psi::SharedMatrix `centres_`
Centres, at which fit charges will reside.
- psi::SharedVector `charge_sums_`
Vector of sums of partial charges.

14.30.1 Detailed Description

Solves the least-squares problem to fit the generalized charges $q_{m;p}$, that reproduce the reference generalized potential $v_p^{\text{ref}}(\mathbf{r})$ supplied by the `Field3D` object:

$$\int d\mathbf{r}' \left[v_p^{\text{ref}}(\mathbf{r}') - \sum_m \frac{q_{m;p}}{|\mathbf{r}' - \mathbf{r}_m|} \right]^2 \rightarrow \text{minimize}$$

The charges are subject to the following constraint:

$$\sum_m q_{m;p} = Q_p \text{ for all } p$$

Method description.

M generalized charges is found by solving the matrix equation

$$\begin{pmatrix} \mathbf{A} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} \end{pmatrix}^{-1} \cdot \begin{pmatrix} \mathbf{b}_p \\ Q_p \end{pmatrix} = \begin{pmatrix} \mathbf{q}_p \\ \lambda \end{pmatrix}$$

where the \mathbf{A} matrix of dimension $(M+1) \times (M+1)$ and \mathbf{b}_p vector of length $M+1$ are given as

$$A_{mn} = \sum_i \frac{1}{r_{im} r_{in}}$$

$$b_{m;p} = \sum_i \frac{v_p^{\text{ref}}(\mathbf{r}_m)}{r_{im}}$$

In the above equation, summations run over all sample points, at which reference potential is known. The solution is stored in the $M \times N$ matrix, where N is the dimensionality of the 3D vector field (i.e., the number of potentials supplied, p_{max}). As a default, $Q_p = 0$ for all potentials. This can be set by `oepdev::ESPSolver::set_charge_sums` method.

Note

Useful options:

- `ESP_PAD_SPHERE` - Padding spherical radius for random points selection. Default: 10.0 [A.U.]
- `ESP_NPOINTS_PER_ATOM` - Number of random points per atom in a molecule. Default: 1500
- `ESP_VDW_RADIUS_C` - The vdW radius for carbon atom. Default: 3.0 [A.U.]
- `ESP_VDW_RADIUS_H` - The vdW radius for hydrogen atom. Default: 4.0 [A.U.]
- `ESP_VDW_RADIUS_N` - The vdW radius for nitrogen atom. Default: 2.4 [A.U.]
- `ESP_VDW_RADIUS_O` - The vdW radius for oxygen atom. Default: 5.6 [A.U.]
- `ESP_VDW_RADIUS_F` - The vdW radius for fluorium atom. Default: 2.3 [A.U.]
- `ESP_VDW_RADIUS_CL` - The vdW radius for chlorium atom. Default: 2.9 [A.U.]

14.30.2 Constructor & Destructor Documentation**ESPSolver()** [1/2]

```
oepdev::ESPSolver::ESPSolver (
    SharedField3D field )
```

Assume that the centres are on atoms associated with the 3D vector field.

Parameters

<i>field</i>	- oepdev 3D vector field object
--------------	---------------------------------

ESPSolver() [2/2]

```
oepdev::ESPSolver::ESPSolver (
    SharedField3D field,
    psi::SharedMatrix centres )
```

Solve ESP equations for a custom set of charge distribution centres.

Parameters

<i>field</i>	- oepdev 3D vector field object
<i>centres</i>	- matrix with coordinates of charge distribution centres

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

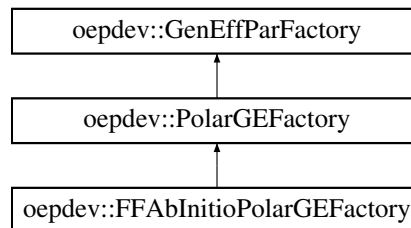
- [oepdev/lib3d/esp.h](#)
- [oepdev/lib3d/esp.cc](#)

14.31 oepdev::FFAbInitioPolarGEFactory Class Reference

Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::FFAbInitioPolarGEFactory:



Public Member Functions

- **FFAbInitioPolarGEFactory** (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- virtual std::shared_ptr< [GenEffPar](#) > [compute](#) (void)
Compute the density matrix susceptibility tensors.

Additional Inherited Members

14.31.1 Detailed Description

Implements creation of the density matrix susceptibility tensors. Does not guarantee the idempotency of the density matrix in LCAO-MO variation, but for weak electric fields the idempotency is to be expected up to first order. The density matrix susceptibility tensor is represented by:

$$\delta D_{\alpha\beta} = \mathbf{B}_{\alpha\beta}^{(1)} \cdot \mathbf{F} + \mathbf{B}_{\alpha\beta}^{(2)} : \mathbf{F} \otimes \mathbf{F}$$

where $\mathbf{B}_{\alpha\beta}^{(1)}$ is the density matrix dipole polarizability defined as

$$\mathbf{B}_{\alpha\beta}^{(1)} = \left. \frac{\partial D_{\alpha\beta}}{\partial \mathbf{F}} \right|_{\mathbf{F}=\mathbf{0}}$$

whereas $\mathbf{B}_{\alpha\beta}^{(2)}$ is the density matrix dipole-dipole hyperpolarizability,

$$\mathbf{B}_{\alpha\beta}^{(2)} = \left. \frac{1}{2} \frac{\partial^2 D_{\alpha\beta}}{\partial \mathbf{F} \otimes \partial \mathbf{F}} \right|_{\mathbf{F}=\mathbf{0}}$$

The first derivative is evaluated numerically from central finite-field 3-point formula,

$$f' = \frac{f(h) - f(-h)}{2h} + \mathcal{O}(h^2)$$

where h is the differentiation step. Second derivatives are evaluated from the following formulae:

$$f_{uu} = \frac{f(h) + f(-h) - 2f(0)}{h^2} + \mathcal{O}(h^2)$$

$$f_{uvw} = \frac{f(h, h) + f(-h, -h) + 2f(0) - f(h, 0) - f(-h, 0) - f(0, h) - f(0, -h)}{2h^2} + \mathcal{O}(h^2)$$

As long as the second-order susceptibility is considered, this susceptibility model works well for uniform weak, moderate and strong electric fields.

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

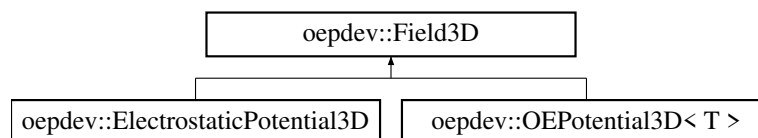
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp_polar_ffabinitio.cc

14.32 oepdev::Field3D Class Reference

General Vector Field in 3D Space. Abstract base.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::Field3D:



Public Member Functions

- [Field3D](#) (const int &ndim, const int &np, const double &pad, psi::SharedWavefunction [wfn](#), psi::Options &options)
Construct potential on random grid by providing wavefunction. Excludes space within vdW volume.
- [Field3D](#) (const int &ndim, const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &options)
Construct potential on cube grid by providing wavefunction.
- virtual [~Field3D](#) ()
Destructor.
- virtual int [npoints](#) () const
Get the number of points at which the 3D field is defined.

- virtual std::shared_ptr< [PointsCollection3D](#) > [points_collection](#) () const
Get the collection of points.
- virtual std::shared_ptr< psi::Matrix > [data](#) () const
Get the data matrix in a form $\{ [x, y, z, f_1(x, y, z), f_2(x, y, z), \dots, f_n(x, y, z)] \}$ where $n = ndim$.
- virtual std::shared_ptr< psi::Wavefunction > [wfn](#) () const
Get the wavefunction.
- virtual bool [is_computed](#) () const
Get the information if data is already computed or not.
- int [dimension](#) () const
Get the number of fields.
- virtual void [compute](#) ()
Compute the 3D field in each point from the point collection.
- virtual std::shared_ptr< psi::Vector > [compute_xyz](#) (const double &x, const double &y, const double &z)=0
Compute a value of 3D field at point (x, y, z)
- virtual void [write_cube_file](#) (const std::string &name)
Write the cube file (only for Cube collections, otherwise does nothing)
- virtual void [print](#) () const =0
Print information of the object to Psi4 output.

Static Public Member Functions

- static shared_ptr< [Field3D](#) > [build](#) (const std::string &type, const int &np, const double &pad, psi::SharedWavefunction [wfn](#), psi::Options &options, const int &ndim=1)
Build 3D field of random points. vdW volume is excluded.
- static shared_ptr< [Field3D](#) > [build](#) (const std::string &type, const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedWavefunction [wfn](#), psi::Options &options, const int &ndim=1)
Build 3D field of points on a g09-cube grid.

Protected Attributes

- std::shared_ptr< [PointsCollection3D](#) > [pointsCollection_](#)
Collection of points at which the 3D field is to be computed.
- std::shared_ptr< psi::Matrix > [data_](#)
The data matrix in a form $\{ [x, y, z, f_1(x, y, z), f_2(x, y, z), \dots, f_n(x, y, z)] \}$ where $n = nDim$.
- std::shared_ptr< psi::Wavefunction > [wfn_](#)
Wavefunction.
- psi::Matrix [geom_](#)
Geometry of a molecule.
- std::shared_ptr< psi::IntegralFactory > [fact_](#)

- Integral factory.*
- `std::shared_ptr< psi::Matrix > pot_`
Matrix of potential one-electron integrals.
- `std::shared_ptr< psi::OneBodyAOInt > oneInt_`
One-electron integral shared pointer.
- `std::shared_ptr< PotentialInt > potInt_`
One-electron potential shared pointer.
- `std::shared_ptr< psi::BasisSet > primary_`
Basis set.
- `int nbf_`
Number of basis functions.
- `int nDim_`
Dimensionality of the 3D field (1: scalar field, 2>: vector field)
- `bool isComputed_`
Has data already computed?

14.32.1 Detailed Description

Create vector field defined at points distributed randomly or as an ordered g09 cube-like collection. Currently implemented fields are:

- Electrostatic potential - computes electrostatic potential (requires wavefunction)
- Template of generic classes - compute custom vector fields (requires generic object that is able to compute the field in 3D space)

Note: Always create instances by using static factory methods `build`. The following types of 3D vector fields are currently implemented:

- ELECTROSTATIC POTENTIAL

14.32.2 Constructor & Destructor Documentation

Field3D()

```
oepdev::Field3D::Field3D (
    const int & ndim,
    const int & nx,
    const int & ny,
    const int & nz,
    const double & px,
```

```

    const double & py,
    const double & pz,
    std::shared_ptr< psi::Wavefunction > wfn,
    psi::Options & options )

```

Construct potential on random grid by providing molecule. Excludes space within vdW volume Field3D(const int& ndim, const int& np, const double& pad, psi::SharedMolecule mol, psi::Options& options);

14.32.3 Member Function Documentation

build() [1/2]

```

std::shared_ptr< Field3D > oepdev::Field3D::build (
    const std::string & type,
    const int & np,
    const double & pad,
    psi::SharedWavefunction wfn,
    psi::Options & options,
    const int & ndim = 1 ) [static]

```

Parameters

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<i>type</i>	- type of 3D field
<i>np</i>	- number of points
<i>pad</i>	- radius padding of a minimal sphere enclosing the molecule
<i>wfn</i>	- Psi4 Wavefunction containing the molecule
<i>options</i>	- Psi4 options

build() [2/2]

```

std::shared_ptr< Field3D > oepdev::Field3D::build (
    const std::string & type,
    const int & nx,
    const int & ny,
    const int & nz,
    const double & px,
    const double & py,
    const double & pz,
    psi::SharedWavefunction wfn,
    psi::Options & options,

```



```
const int & ndim = 1 ) [static]
```

Parameters

<i>ndim</i>	- dimensionality of 3D field (1: scalar field, >2: vector field)
<i>type</i>	- type of 3D field
<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>px</i>	- padding distance along x direction
<i>py</i>	- padding distance along y direction
<i>pz</i>	- padding distance along z direction
<i>wfn</i>	- Psi4 Wavefunction containing the molecule
<i>options</i>	- Psi4 options

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

- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.33 oepdev::Fourier5 Struct Reference

Simple structure to hold the Fourier series expansion coefficients for $N=2$.

```
#include <unitary_optimizer.h>
```

Public Attributes

- double **a0**
- double **a1**
- double **a2**
- double **b1**
- double **b2**

14.33.1 Detailed Description

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

- oepdev/libutil/[unitary_optimizer.h](#)

14.34 oepdev::Fourier9 Struct Reference

Simple structure to hold the Fourier series expansion coefficients for $N=4$.

```
#include <unitary_optimizer.h>
```

Public Attributes

- double **a0**
- double **a1**
- double **a2**
- double **a3**
- double **a4**
- double **b1**
- double **b2**
- double **b3**
- double **b4**

14.34.1 Detailed Description

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

- oepdev/libutil/[unitary_optimizer.h](#)

14.35 oepdev::FragmentedSystem Class Reference

Molecular System for Fragment-Based Calculations.

```
#include <gefp.h>
```

Public Member Functions

Mutators

- void [set_geometry](#) (std::vector< psi::SharedMolecule > aggregate)
Set the current atomic coordinates of the system.
- void [set_primary](#) (std::vector< psi::SharedBasisSet > p)
Set the current atomic coordinates of the system.
- void [set_auxiliary](#) (std::vector< psi::SharedBasisSet > a)
Set the auxiliary basis sets (TO BE DEPRECATED)

Transformators

- void [superimpose](#) ()
Superimpose all the fragments onto the current atomic coordinates.

Computers

- double [compute_energy](#) (std::string theory)
Compute a total energy.
- double [compute_energy_term](#) (std::string theory, bool manybody)
Compute a single energy term.

Protected Attributes

Working Attributes

- std::vector< std::shared_ptr< [GenEffFrag](#) > > [bsm_](#)
List of Base Fragments (BSMs)
- std::vector< int > [ind_](#)
List of fragment assignment indices.
- const int [nfrag_](#)
Number of all fragments in the system.
- std::vector< std::shared_ptr< [GenEffFrag](#) > > [fragments_](#)
List of all fragments in the system.
- std::vector< psi::SharedMolecule > [aggregate_](#)
List of molecules currently representing all fragments in the system.
- std::vector< psi::SharedBasisSet > [basis_prim_](#)
List of current primary basis sets (TO BE DEPRECATED)
- std::vector< psi::SharedBasisSet > [basis_aux_](#)
List of current auxiliary basis sets (TO BE DEPRECATED)

Constructors and Destructor.

- static std::shared_ptr< [FragmentedSystem](#) > [build](#) (std::vector< std::shared_ptr< [GenEffFrag](#) > > bsm, std::vector< int > ind)
Build from the list of base molecules (BSM) and fragment assignment vector.
- [FragmentedSystem](#) (std::vector< std::shared_ptr< [GenEffFrag](#) > > bsm, std::vector< int > ind)
Constructor.
- virtual [~FragmentedSystem](#) ()
Destructor.

14.35.1 Detailed Description

Implements interface of running fragment-based calculations on molecular systems defined in terms of independent but interacting fragments.

14.35.2 Member Function Documentation

build()

```
oepdev::SharedFragmentedSystem oepdev::FragmentedSystem::build (
    std::vector< std::shared_ptr< GenEffFrag >> bsm,
    std::vector< int > ind ) [static]
```

Parameters

<i>bsm</i>	- list of base molecules
<i>ind</i>	- list of fragment assignments indices

Returns

system of fragments

After initialization, the list of fragments f_i is created within the object, where the i -th fragment is given by

$$f_i = \text{copy}(m_{d_i})$$

In the above, m and d denote the lists of BSMs and fragment assignment indices, respectively.

set_geometry()

```
void oepdev::FragmentedSystem::set_geometry (
    std::vector< psi::SharedMolecule > aggregate ) [inline]
```

Parameters

<i>aggregate</i>	- list of all molecules in the system
------------------	---------------------------------------

set_primary()

```
void oepdev::FragmentedSystem::set_primary (
    std::vector< psi::SharedBasisSet > p ) [inline]
```

Parameters

<i>aggregate</i>	- molecule object of the whole systemSet the current atomic coordinates of the system.
<i>aggregate</i>	- molecule object of the whole systemSet the primary basis sets (TO BE DEPRECATED)
<i>p</i>	- list of all primary basis sets in the system

Note

This will be deprecated once basis sets can be rotated and embedded in [oepdev::GenEffFrag](#).

set_auxiliary()

```
void oepdev::FragmentedSystem::set_auxiliary (
    std::vector< psi::SharedBasisSet > a ) [inline]
```

Parameters

<i>a</i>	- list of all auxiliary basis sets in the system
----------	--

Note

This will be deprecated once basis sets can be rotated and embedded in [oepdev::GenEffFrag](#).

compute_energy()

```
double oepdev::FragmentedSystem::compute_energy (
    std::string theory )
```

Parameters

<i>theory</i>	- theory to use for calculations
---------------	----------------------------------

Returns

energy in a.u.

compute_energy_term()

```
double oepdev::FragmentedSystem::compute_energy_term (
    std::string theory,
    bool manybody )
```

Parameters

<i>theory</i>	- theory to use for calculations
<i>manybody</i>	- whether to use many body routines.

Returns

energy in a.u.

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

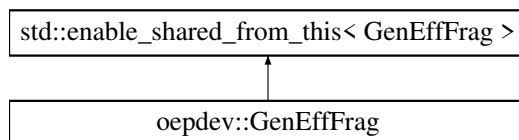
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/fragmented_system.cc

14.36 oepdev::GenEffFrag Class Reference

Generalized Effective Fragment. Container Class.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::GenEffFrag:

**Public Member Functions****Transformators**

- void [rotate](#) (std::shared_ptr< psi::Matrix > R)
Rotate.
- void [translate](#) (std::shared_ptr< psi::Vector > T)
Translate.
- void [superimpose](#) (std::shared_ptr< psi::Matrix > targetXYZ, std::vector< int > supList)
Superimpose.
- void [superimpose](#) (psi::SharedMolecule targetMol, std::vector< int > supList)
Superimpose.
- void [superimpose](#) (void)
Superimpose to the structure held in frag_

Mutators

- void [set_parameters](#) (const std::string &type, std::shared_ptr< [GenEffPar](#) > par)
Set the parameters.
- void [set_ndocc](#) (int n)
Set the number of doubly occupied MOs.
- void [set_nbf](#) (int n)
Set the number of primary basis functions.
- void [set_molecule](#) (const psi::SharedMolecule mol)

Set the fragment molecule.

- void [set_basisset](#) (std::string key, psi::SharedBasisSet basis)

Set the basis set.

- void [set_gefp_polarization](#) (const std::shared_ptr< [GenEffPar](#) > &par)

Set the Density Matrix Susceptibility Tensor Object.

- void [set_dmat_dipole_polarizability](#) (const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> &susc)

Set the Density Matrix Dipole Polarizability.

- void [set_dmat_dipole_dipole_hyperpolarizability](#) (const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> &susc)

Set the Density Matrix Dipole-Dipole Hyperpolarizability.

- void [set_dmat_quadrupole_polarizability](#) (const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> &susc)

Set the Density Matrix Quadrupole Polarizability.

Accessors

- int [nbf](#) (void) const

Grab the number of primary basis functions.

- int [natom](#) (void) const

Grab the number of atoms.

- int [ndocc](#) (void) const

Grab the number of doubly occupied molecular orbitals.

- psi::SharedMolecule [molecule](#) (void) const

Grab the molecule attached to this fragment.

- std::shared_ptr< psi::Matrix > [susceptibility](#) (int fieldRank, int fieldGradientRank, int i, int x) const

Grab the Density Matrix Susceptibility.

- std::vector< std::shared_ptr< psi::Matrix > > [susceptibility](#) (int fieldRank, int fieldGradientRank, int i) const

Grab the Density Matrix Susceptibility.

- std::vector< std::vector< std::shared_ptr< psi::Matrix > > > [susceptibility](#) (int fieldRank, int fieldGradientRank) const

Grab the Density Matrix Susceptibility.

Public Attributes

Parameters

- std::map< std::string, std::shared_ptr< [GenEffPar](#) > > [parameters](#)

Dictionary of All GEF Parameters.

- std::map< std::string, psi::SharedBasisSet > [basissets](#)

Dictionary of All Basis Sets.

Protected Member Functions

- `psi::SharedVector extract_xyz (psi::SharedMolecule) const`
Extract XYZ.
- `psi::SharedVector extract_dmtp (std::shared_ptr< oepdev::DMTPole >) const`
Extract DMTP.
- `psi::SharedVector compute_u_vector (psi::SharedMatrix rmo_1, psi::SharedMatrix rmo_2, psi::SharedMolecule mol_2) const`
Compute u vector for OEP-CT calculations.
- `psi::SharedMatrix compute_w_matrix (psi::SharedMolecule mol_1, psi::SharedMolecule mol_2, psi::SharedMatrix rmo_1) const`
Compute w matrix for OEP-CT calculations.
- `double compute_ct_component (psi::SharedVector eps_occ_X, psi::SharedVector eps_vir_Y, psi::SharedMatrix V) const`
Compute OEP-CT energy component.

Interface Computers

- `double compute_pairwise_energy (std::string theory, std::shared_ptr< GenEffFrag > other) const`
- `double compute_pairwise_energy_efp2_coul (std::shared_ptr< GenEffFrag > other) const`
- `double compute_pairwise_energy_efp2_exrep (std::shared_ptr< GenEffFrag > other) const`
- `double compute_pairwise_energy_efp2_ind (std::shared_ptr< GenEffFrag > other) const`
- `double compute_pairwise_energy_efp2_ct (std::shared_ptr< GenEffFrag > other) const`
- `double compute_pairwise_energy_efp2_disp (std::shared_ptr< GenEffFrag > other) const`
- `double compute_pairwise_energy_oep_efp2_exrep (std::shared_ptr< GenEffFrag > other) const`
- `double compute_pairwise_energy_oep_efp2_ct (std::shared_ptr< GenEffFrag > other) const`

Protected Attributes

- `std::string name_`
Name of GEFP.
- `psi::SharedMolecule frag_`
Structure.
- `int nbf_`
Number of primary basis functions.
- `int natom_`

Number of atoms.

- int `ndocc_`

Number of doubly occupied MOs.

- `std::shared_ptr< GenEffPar > densityMatrixSusceptibilityGEF_`

Constructors and Destructor

- `GenEffFrag ()`

Initialize with default name of GEFP (Default)

- `GenEffFrag (std::string name)`

Initialize with custom name of GEFP.

- `GenEffFrag (const GenEffFrag *)`

Copy Constructor.

- `std::shared_ptr< GenEffFrag > clone (void) const`

Make a deep copy.

- `~GenEffFrag ()`

Destruct.

- `static std::shared_ptr< GenEffFrag > build (std::string name)`

Create an empty fragment.

Computers

- `double energy_term (std::string theory, std::shared_ptr< GenEffFrag > other) const`

Compute interaction energy between this and other fragment.

- `static double compute_energy (std::string theory, std::vector< std::shared_ptr< GenEffFrag >> fragments)`

Compute the total interaction energy term in a cluster of fragments.

- `static double compute_energy_term (std::string theory, std::vector< std::shared_ptr< GenEffFrag >> fragments, bool manybody)`

Compute a single interaction energy term in a cluster of fragments.

- `static double compute_many_body_energy_term (std::string theory, std::vector< std::shared_ptr< GenEffFrag >> fragments)`

Compute a single interaction energy term in a cluster of fragments by using manybody routine.

14.36.1 Detailed Description

Describes the GEFP fragment that is in principle designed to work at correlated levels of theory.

See also

[GenEffPar](#), [GenEffParFactory](#)

14.36.2 Member Function Documentation

susceptibility() [1/3]

```
std::shared_ptr<psi::Matrix> oepdev::GenEffFrag::susceptibility (
    int fieldRank,
    int fieldGradientRank,
    int i,
    int x ) const [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient
<i>i</i>	- id of the distributed site
<i>x</i>	- id of the composite Cartesian component

susceptibility() [2/3]

```
std::vector<std::shared_ptr<psi::Matrix> > oepdev::GenEffFrag::susceptibility
(
    int fieldRank,
    int fieldGradientRank,
    int i ) const [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient
<i>i</i>	- id of the distributed site

susceptibility() [3/3]

```
std::vector<std::vector<std::shared_ptr<psi::Matrix> > > oepdev::GenEffFrag::susceptibility
(
    int fieldRank,
    int fieldGradientRank ) const [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
------------------	---

Parameters

<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient
--------------------------	--

energy_term()

```
double oepdev::GenEffFrag::energy_term (
    std::string theory,
    std::shared_ptr< GenEffFrag > other ) const
```

Parameters

<i>theory</i>	- theory used to compute energy
<i>other</i>	- other fragment

Returns

interaction energy in [A.U.]

compute_energy()

```
double oepdev::GenEffFrag::compute_energy (
    std::string theory,
    std::vector< std::shared_ptr< GenEffFrag >> fragments ) [static]
```

Parameters

<i>theory</i>	- theory used to compute energy
<i>fragments</i>	- list of fragments in the system

Returns

interaction energy in [A.U.]

compute_energy_term()

```
double oepdev::GenEffFrag::compute_energy_term (
    std::string theory,
    std::vector< std::shared_ptr< GenEffFrag >> fragments,
    bool manybody ) [static]
```

Parameters

<i>theory</i>	- theory used to compute energy
<i>fragments</i>	- list of fragments in the system
<i>manybody</i>	- use the manybody routine? If not, pairwise routine is utilized.

Returns

interaction energy in [A.U.]

compute_many_body_energy_term()

```
double oepdev::GenEffFrag::compute_many_body_energy_term (
    std::string theory,
    std::vector< std::shared_ptr< GenEffFrag >> fragments ) [static]
```

Parameters

<i>theory</i>	- theory used to compute energy
<i>fragments</i>	- list of fragments in the system

Returns

interaction energy in [A.U.]

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

- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp_frag.cc

14.37 oepdev::GenEffPar Class Reference

Generalized Effective Fragment Parameters. Container Class.

```
#include <gefp.h>
```

Public Member Functions**Transformators**

- void [rotate](#) (psi::SharedMatrix R)
Rotate the parameters in 3D Euclidean space.

- void [translate](#) (psi::SharedVector t)
Translate the parameters in 3D Euclidean space.
- void [superimpose](#) (psi::SharedMatrix targetXYZ, std::vector< int > supList)
Superimpose the parameters in 3D Euclidean space onto a target geometry.

Mutators

- void [set_vector](#) (std::string key, psi::SharedVector mat)
Set the vector data.
- void [set_matrix](#) (std::string key, psi::SharedMatrix mat)
Set the matrix data.
- void [set_dmtip](#) (std::string key, std::shared_ptr< oepdev::DMTPole > mat)
Set the DMTP data.
- void [set_oep](#) (std::string key, oepdev::SharedOEPotential oep)
Set the OEP data.
- void [set_dpol](#) (std::string key, std::vector< psi::SharedMatrix > mats)
Set the DPOL data.
- void [set_basisset](#) (std::string key, psi::SharedBasisSet basis)
Set the basis set data.
- void [set_susceptibility](#) (int fieldRank, int fieldGradientRank, const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> &susc)
Set the Density Matrix Susceptibility.
- void [set_dipole_polarizability](#) (const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> &susc)
Set The Density Matrix Dipole Polarizability.
- void [set_dipole_dipole_hyperpolarizability](#) (const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> &susc)
Set The Density Matrix Dipole-Dipole Hyperpolarizability.
- void [set_quadrupole_polarizability](#) (const std::vector< std::vector< std::shared_ptr< psi::Matrix >>> &susc)
Set The Density Matrix Quadrupole Polarizability.
- void [set_centres](#) (const std::vector< std::shared_ptr< psi::Vector >> ¢res)
Set the distributed centres' positions.

Allocators

- void [allocate](#) (int fieldRank, int fieldGradientRank, int nsites, int nbf)
Allocate the Density Matrix Susceptibility.
- void [allocate_dipole_polarizability](#) (int nsites, int nbf)
Allocate The Density Matrix Dipole Polarizability.
- void [allocate_dipole_dipole_hyperpolarizability](#) (int nsites, int nbf)
Allocate The Density Matrix Dipole-Dipole Hyperpolarizability.
- void [allocate_quadrupole_polarizability](#) (int nsites, int nbf)
Allocate The Density Matrix Quadrupole Polarizability.

Descriptors

- `std::string type () const`
Type of Parameters.
- `std::string name () const`
Name of Parameters.
- `bool hasDensityMatrixDipolePolarizability () const`
Does it has dipole polarizability DMS?
- `bool hasDensityMatrixDipoleDipoleHyperpolarizability () const`
Does it has dipole-dipole hyperpolarizability DMS?
- `bool hasDensityMatrixQuadrupolePolarizability () const`
Does it has quadrupole polarizability DMS?

Accessors

- `psi::SharedVector vector (std::string key) const`
Get the vector data.
- `psi::SharedMatrix matrix (std::string key) const`
Get the matrix data.
- `std::shared_ptr< oepdev::DMTPole > dmp (std::string key) const`
Get the DMTP data.
- `oepdev::SharedOEPotential oep (std::string key) const`
Get the OEP data.
- `std::vector< psi::SharedMatrix > dpol (std::string key) const`
Get the DPOL data.
- `psi::SharedBasisSet basisset (std::string key) const`
Get the basis set data.
- `std::shared_ptr< psi::Matrix > susceptibility (int fieldRank, int fieldGradientRank, int i, int x) const`
Grab the Density Matrix Susceptibility.
- `std::vector< std::shared_ptr< psi::Matrix > > susceptibility (int fieldRank, int fieldGradientRank, int i) const`
Grab the Density Matrix Susceptibility.
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > susceptibility (int fieldRank, int fieldGradientRank) const`
Grab the Density Matrix Susceptibility.
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > dipole_polarizability () const`
Grab the density matrix dipole polarizability tensor.
- `std::vector< std::shared_ptr< psi::Matrix > > dipole_polarizability (int i) const`
Grab the density matrix dipole polarizability tensor's x-th component.
- `std::shared_ptr< psi::Matrix > dipole_polarizability (int i, int x) const`
Grab the density matrix dipole polarizability tensor's x-th component of the i-th distributed site.
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > dipole_dipole_hyperpolarizability () const`
Grab the density matrix dipole-dipole hyperpolarizability tensor.
- `std::vector< std::shared_ptr< psi::Matrix > > dipole_dipole_hyperpolarizability (int i) const`

- Grab the density matrix dipole-dipole hyperpolarizability tensor's x-th component.*

• `std::shared_ptr< psi::Matrix > dipole_dipole_hyperpolarizability (int i, int x) const`

Grab the density matrix dipole-dipole hyperpolarizability tensor's x-th component of the i-th distributed site.
- `std::vector< std::vector< std::shared_ptr< psi::Matrix > > > quadrupole_polarizability () const`

Grab the density matrix quadrupole polarizability tensor.
- `std::vector< std::shared_ptr< psi::Matrix > > quadrupole_polarizability (int i) const`

Grab the density matrix quadrupole polarizability tensor's x-th component.
- `std::shared_ptr< psi::Matrix > quadrupole_polarizability (int i, int x) const`

Grab the density matrix quadrupole polarizability tensor's x-th component of the i-th distributed site.
- `std::vector< std::shared_ptr< psi::Vector > > centres () const`

Grab the centres' positions.
- `std::shared_ptr< psi::Vector > centre (int i) const`

Grab the position of the i-th distributed site.

DMS Computers

- `std::shared_ptr< psi::Matrix > compute_density_matrix (std::shared_ptr< psi::Vector > field)`

Compute the density matrix due to the uniform electric field perturbation.
- `std::shared_ptr< psi::Matrix > compute_density_matrix (double fx, double fy, double fz)`

Compute the density matrix due to the uniform electric field perturbation.
- `std::shared_ptr< psi::Matrix > compute_density_matrix (std::vector< std::shared_ptr< psi::Vector > > fields)`

Compute the density matrix due to the non-uniform electric field perturbation.
- `std::shared_ptr< psi::Matrix > compute_density_matrix (std::vector< std::shared_ptr< psi::Vector > > fields, std::vector< std::shared_ptr< psi::Matrix > > grads)`

Compute the density matrix due to the non-uniform electric field perturbation.

Protected Attributes

Qualifiers

Compute the interaction energy between this and other EFP2 fragment.

Parameters

par	- other parameters object
-----	---------------------------

- `std::string name_`

The Name of Parameter.
- `std::string type_`

The Type of Parameter.
- `bool hasDensityMatrixDipolePolarizability_`

The Name of Parameter.

- bool [hasDensityMatrixDipoleDipoleHyperpolarizability_](#)
The Name of Parameter.
- bool [hasDensityMatrixQuadrupolePolarizability_](#)
The Name of Parameter.

Matrices and Multipoles

- std::vector< std::shared_ptr< psi::Vector > > [distributedCentres_](#)
The Positions of the Distributed Centres.
- std::map< std::string, psi::SharedVector > [data_vector_](#)
Data for Vector Types by Keyword.
- std::map< std::string, psi::SharedMatrix > [data_matrix_](#)
Data for Matrix Types by Keyword.
- std::map< std::string, std::shared_ptr< oepdev::DMTPole > > [data_dmtpl_](#)
Data for DMTP Types by Keyword.
- std::map< std::string, oepdev::SharedOEPotential > [data_oep_](#)
Data for OEP Types by Keyword.
- std::map< std::string, std::vector< psi::SharedMatrix > > [data_dpole_](#)
Data for DMTP Types by Keyword.
- std::map< std::string, psi::SharedBasisSet > [data_basisset_](#)
Data for AO Basis Set by Keyword.

Density Matrix Susceptibility

- std::vector< std::vector< std::shared_ptr< psi::Matrix > > > [densityMatrixDipolePolarizability_](#)
The Density Matrix Dipole Polarizability.
- std::vector< std::vector< std::shared_ptr< psi::Matrix > > > [densityMatrixDipoleDipoleHyperpolarizability_](#)
The Density Matrix Dipole-Dipole Hyperpolarizability.
- std::vector< std::vector< std::shared_ptr< psi::Matrix > > > [densityMatrixQuadrupolePolarizability_](#)
The Density Matrix Quadrupole Polarizability.

Constructor and Destructor

- [GenEffPar](#) (std::string name)
Create with name of this parameter.
- [GenEffPar](#) (const [GenEffPar](#) *)
Copy Constructor.
- std::shared_ptr< [GenEffPar](#) > [clone](#) (void) const
Make a deep copy.
- [~GenEffPar](#) ()
Destruct.
- virtual void [copy_from](#) (const [GenEffPar](#) *)
Deep-copy the matrix and DMTP data.

14.37.1 Detailed Description

See also

[GenEffFrag](#), [GenEffParFactory](#)

14.37.2 Member Function Documentation

rotate()

```
void oepdev::GenEffPar::rotate (
    psi::SharedMatrix R )
```

Parameters

R	- the rotation matrix
-----	-----------------------

translate()

```
void oepdev::GenEffPar::translate (
    psi::SharedVector t )
```

Parameters

t	- the translation vector
-----	--------------------------

superimpose()

```
void oepdev::GenEffPar::superimpose (
    psi::SharedMatrix targetXYZ,
    std::vector< int > supList )
```

Parameters

<i>targetXYZ</i>	- the target geometry
<i>suplist</i>	- the superimposition list

set_vector()

```
void oepdev::GenEffPar::set_vector (
    std::string key,
    psi::SharedVector mat ) [inline]
```

Parameters

<i>key</i>	- keyword for a vector
<i>mat</i>	- vector

This sets the item in the map `data_vector_`.

set_matrix()

```
void oepdev::GenEffPar::set_matrix (
    std::string key,
    psi::SharedMatrix mat ) [inline]
```

Parameters

<i>key</i>	- keyword for a matrix
<i>mat</i>	- matrix

This sets the item in the map `data_matrix_`.

set_dmtip()

```
void oepdev::GenEffPar::set_dmtip (
    std::string key,
    std::shared_ptr< oepdev::DMTPole > mat ) [inline]
```

Parameters

<i>key</i>	- keyword for a DMTP
<i>dmtip</i>	- DMTP object

This sets the item in the map `data_dmtip_`.

set_oep()

```
void oepdev::GenEffPar::set_oep (
    std::string key,
    oepdev::SharedOEPotential oep ) [inline]
```

Parameters

<i>key</i>	- keyword for a OEP
<i>oep</i>	- OEP object

This sets the item in the map `data_oep_`.

set_dpole()

```
void oepdev::GenEffPar::set_dpole (
    std::string key,
    std::vector< psi::SharedMatrix > mats ) [inline]
```

Parameters

<i>key</i>	- keyword for a DPOL
<i>dmp</i>	- DPOL object

This sets the item in the map `data_dpole_`.

set_basiset()

```
void oepdev::GenEffPar::set_basiset (
    std::string key,
    psi::SharedBasisSet basis ) [inline]
```

Parameters

<i>key</i>	- keyword for a matrix
<i>mat</i>	- matrix

This sets the item in the map `data_basiset_`.

set_susceptibility()

```
void oepdev::GenEffPar::set_susceptibility (
    int fieldRank,
    int fieldGradientRank,
    const std::vector< std::vector< std::shared_ptr< psi::Matrix >>>
    & susc ) [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field \mathbf{F}
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient $\nabla \otimes \mathbf{F}$
<i>susc</i>	- the susceptibility tensor

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with \mathbf{F}
- (2, 0) - dipole-dipole hyperpolarizability, interacts with $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with $\nabla \otimes \mathbf{F}$

allocate()

```
void oepdev::GenEffPar::allocate (
    int fieldRank,
    int fieldGradientRank,
    int nsites,
    int nbf ) [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field \mathbf{F}
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient $\nabla \otimes \mathbf{F}$
<i>nsites</i>	- number of distributed sites
<i>nbf</i>	- number of basis functions in the basis set

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with \mathbf{F}
- (2, 0) - dipole-dipole hyperpolarizability, interacts with $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with $\nabla \otimes \mathbf{F}$

vector()

```
psi::SharedVector oepdev::GenEffPar::vector (
    std::string key ) const [inline]
```

Parameters

<i>key</i>	- keyword for a vector
------------	------------------------

Returns

vector data type

matrix()

```
psi::SharedMatrix oepdev::GenEffPar::matrix (  
    std::string key ) const [inline]
```

Parameters

<i>key</i>	- keyword for a matrix
------------	------------------------

Returns

matrix data type

dmtp()

```
std::shared_ptr<oepdev::DMTPole> oepdev::GenEffPar::dmtp (  
    std::string key ) const [inline]
```

Parameters

<i>key</i>	- keyword for a DMTP
------------	----------------------

Returns

DMTP data type

oep()

```
oepdev::SharedOEPotential oepdev::GenEffPar::oep (  
    std::string key ) const [inline]
```

Parameters

<i>key</i>	- keyword for a OEP
------------	---------------------

Returns

OEP data type

dpol()

```
std::vector<psi::SharedMatrix> oepdev::GenEffPar::dpol (
    std::string key ) const [inline]
```

Parameters

<i>key</i>	- keyword for a DPOL
------------	----------------------

Returns

DPOL data type

basisset()

```
psi::SharedBasisSet oepdev::GenEffPar::basisset (
    std::string key ) const [inline]
```

Parameters

<i>key</i>	- keyword for a basis set
------------	---------------------------

Returns

basis set data type

susceptibility() [1/3]

```
std::shared_ptr<psi::Matrix> oepdev::GenEffPar::susceptibility (
    int fieldRank,
    int fieldGradientRank,
    int i,
    int x ) const [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
------------------	---

Parameters

<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient
<i>i</i>	- id of the distributed site
<i>x</i>	- id of the composite Cartesian component

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with \mathbf{F}
- (2, 0) - dipole-dipole hyperpolarizability, interacts with $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with $\nabla \otimes \mathbf{F}$

The distributed sites are assumed to be atomic sites or molecular orbital centroids (depending on the polarization factory used). For the electric field, the composite Cartesian index is just an ordinary Cartesian index. For the electric field gradient and electric field squared, the composite Cartesian index is given as

$$I(x,y) = 3x + y$$

where the values of 0, 1 and 2 correspond to x, y and z Cartesian components, respectively. Therefore, in the latter case, there is 9 distinct composite Cartesian components.

susceptibility() [2/3]

```
std::vector<std::shared_ptr<psi::Matrix> > oepdev::GenEffPar::susceptibility
(
    int fieldRank,
    int fieldGradientRank,
    int i ) const [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient
<i>i</i>	- id of the distributed site

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with \mathbf{F}
- (2, 0) - dipole-dipole hyperpolarizability, interacts with $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with $\nabla \otimes \mathbf{F}$

The distributed sites are assumed to be atomic sites or molecular orbital centroids (depending on the polarization factory used).

susceptibility() [3/3]

```
std::vector<std::vector<std::shared_ptr<psi::Matrix> > > oepdev::GenEffPar::susceptibility
(
    int fieldRank,
    int fieldGradientRank ) const [inline]
```

Parameters

<i>fieldRank</i>	- power dependency with respect to the electric field
<i>fieldGradientRank</i>	- power dependency with respect to the electric field gradient

The following susceptibilities are supported (fieldRank, fieldGradientRank):

- (1, 0) - dipole polarizability, interacts with \mathbf{F}
- (2, 0) - dipole-dipole hyperpolarizability, interacts with $\mathbf{F} \otimes \mathbf{F}$
- (0, 1) - quadrupole polarizability, interacts with $\nabla \otimes \mathbf{F}$

compute_density_matrix() [1/4]

```
std::shared_ptr<psi::Matrix> oepdev::GenEffPar::compute_density_matrix (
    std::shared_ptr< psi::Vector > field )
```

Parameters

<i>field</i>	- the uniform electric field vector (A.U.)
--------------	--

compute_density_matrix() [2/4]

```
psi::SharedMatrix oepdev::GenEffPar::compute_density_matrix (
    double fx,
    double fy,
    double fz )
```

Parameters

<i>fx</i>	- x-th Cartesian component of the uniform electric field vector (A.U.)
<i>fy</i>	- y-th Cartesian component of the uniform electric field vector (A.U.)
<i>fz</i>	- z-th Cartesian component of the uniform electric field vector (A.U.)

Compute the fragment parameters.

Accessors

- virtual std::shared_ptr< psi::Wavefunction > [wfn](#) (void) const
Grab wavefunction.
- virtual psi::Options & [options](#) (void) const
Grab options.
- std::shared_ptr< [oepdev::CPHF](#) > [cphf_solver](#) () const
Grab the CPHF object.
- std::shared_ptr< [oepdev::DMTPole](#) > [dmtp](#) () const
Grab the DMTP object.

Protected Attributes

Basic data

- std::shared_ptr< psi::Wavefunction > [wfn_](#)
Wavefunction.
- psi::Options & [options_](#)
Psi4 Options.
- const int [nbf_](#)
Number of basis functions.

Padding of box

- double [cx_](#)
Centre-of-mass coordinates.
- double [cy_](#)
Centre-of-mass coordinates.
- double [cz_](#)
Centre-of-mass coordinates.
- double [radius_](#)
Radius of padding sphere around the molecule.

Container objects

- std::shared_ptr< [oepdev::CPHF](#) > [cphfSolver_](#)
The CPHF object.
- std::shared_ptr< [oepdev::DMTPole](#) > [dmtpSolver_](#)
The DMTP object.
- std::shared_ptr< [oepdev::QUAMBO](#) > [quamboSolver_](#)
The QUAMBO object.

Other Factories

- std::shared_ptr< [oepdev::GenEffParFactory](#) > [abInitioPolarizationSusceptibilitiesFactory_](#)
Ab initio polarization susceptibility factory.

Constructors and Desctructor

- static std::shared_ptr< [GenEffParFactory](#) > [build](#) (const std::string &type, std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
Build Density Matrix Susceptibility Generalized Factory.
- static std::shared_ptr< [GenEffParFactory](#) > [build](#) (const std::string &type, std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt, psi::SharedBasisSet aux, psi::SharedBasisSet intermed)
Build Density Matrix Susceptibility Generalized Factory.
- [GenEffParFactory](#) (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
Construct from wavefunction and Psi4 options.
- virtual [~GenEffParFactory](#) ()
Destruct.

Random number generation

- std::default_random_engine [randomNumberGenerator_](#)
Draw random number.
- std::uniform_real_distribution< double > [randomDistribution_](#)
Draw random number.
- virtual double [random_double](#) ()
Draw random number.
- virtual std::shared_ptr< psi::Vector > [draw_random_point](#) ()
Draw random point in 3D space, excluding the vdW region.

Van der Waals region

- std::shared_ptr< psi::Matrix > [excludeSpheres_](#)
Matrix with vdW sphere information.
- std::map< std::string, double > [vdwRadius_](#)
Map with vdW radii.
- virtual bool [is_in_vdWsphere](#) (double x, double y, double z) const
Is the point inside a vdW region?

14.38.1 Detailed Description

Describes the GEFP fragment that is in principle designed to work at correlated levels of theory.

See also

[GenEffPar](#), [GenEffFrag](#)

14.38.2 Member Function Documentation

build() [1/2]

```
static std::shared_ptr<GenEffParFactory> oepdev::GenEffParFactory::build (
    const std::string & type,
    std::shared_ptr< psi::Wavefunction > wfn,
    psi::Options & opt ) [static]
```

Parameters

<i>type</i>	- Type of factory
<i>wfn</i>	- Psi4 wavefunction
<i>opt</i>	- Psi4 options

Available factory types:

- POLARIZATION - creates the polarization generalized effective fragment parameters' factory Factory subtype is specified in Psi4 options (input file).

Note

Useful options:

- POLARIZATION factory type:
 - DMATPOL_TRAINING_MODE - training mode. Default: EFIELD
 - DMATPOL_NSAMPLES - number of random samples (field or test charges sets). Default: 30
 - DMATPOL_FIELD_SCALE - electric field scale factor (relevant if training mode is EFIELD). Default: 0.01 [au]
 - DMATPOL_NTEST_CHARGE - number of test charges per sample (relevant if training mode is CHARGES). Default: 1
 - DMATPOL_TEST_CHARGE - test charge value (relevant if training mode is CHARGES). Default: 0.001 [au]
 - DMATPOL_FIELD_RANK - electric field rank. Default: 1
 - DMATPOL_GRADIENT_RANK - electric field gradient rank. Default: 0
 - DMATPOL_TEST_FIELD_X - test electric field in X direction. Default: 0.000 [au]
 - DMATPOL_TEST_FIELD_Y - test electric field in Y direction. Default: 0.000 [au]
 - DMATPOL_TEST_FIELD_Z - test electric field in Z direction. Default: 0.008 [au]
 - DMATPOL_OUT_STATS - output file name for statistical evaluation results. Default: dmatpol.stats.dat

- `DMATPOL_DO_AB_INITIO` - compute ab initio susceptibilities and evaluate statistics for it. Default: `false`
- `DMATPOL_OUT_STATS_AB_INITIO` - output file name for statistical evaluation results of ab initio model. Default: `dmatpol.stats.abinitio.dat`

build() [2/2]

```
static std::shared_ptr<GenEffParFactory> oepdev::GenEffParFactory::build (
    const std::string & type,
    std::shared_ptr<psi::Wavefunction > wfn,
    psi::Options & opt,
    psi::SharedBasisSet aux,
    psi::SharedBasisSet intermed ) [static]
```

Parameters

<i>type</i>	- Type of factory
<i>wfn</i>	- Psi4 wavefunction
<i>opt</i>	- Psi4 options

Available factory types:

- `POLARIZATION` - creates the polarization generalized effective fragment parameters' factory Factory subtype is specified in Psi4 options (input file).

Note**Useful options:**

- **POLARIZATION factory type:**
 - `DMATPOL_TRAINING_MODE` - training mode. Default: `EFIELD`
 - `DMATPOL_NSAMPLES` - number of random samples (field or test charges sets). Default: `30`
 - `DMATPOL_FIELD_SCALE` - electric field scale factor (relevant if training mode is `EFIELD`). Default: `0.01 [au]`
 - `DMATPOL_NTEST_CHARGE` - number of test charges per sample (relevant if training mode is `CHARGES`). Default: `1`
 - `DMATPOL_TEST_CHARGE` - test charge value (relevant if training mode is `CHARGES`). Default: `0.001 [au]`
 - `DMATPOL_FIELD_RANK` - electric field rank. Default: `1`
 - `DMATPOL_GRADIENT_RANK` - electric field gradient rank. Default: `0`
 - `DMATPOL_TEST_FIELD_X` - test electric field in X direction. Default: `0.000 [au]`
 - `DMATPOL_TEST_FIELD_Y` - test electric field in Y direction. Default: `0.000 [au]`

- DMATPOL_TEST_FIELD_Z - test electric field in Z direction. Default: 0.008 [au]
- DMATPOL_OUT_STATS - output file name for statistical evaluation results. Default: dmatpol.stats.dat
- DMATPOL_DO_AB_INITIO - compute ab initio susceptibilities and evaluate statistics for it. Default: false
- DMATPOL_OUT_STATS_AB_INITIO - output file name for statistical evaluation results of ab initio model. Default: dmatpol.stats.abinitio.dat

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

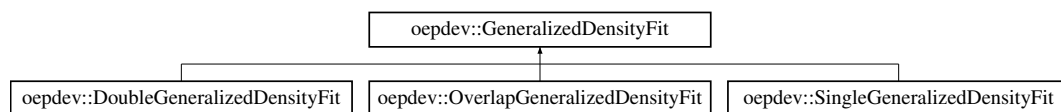
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp.cc

14.39 oepdev::GeneralizedDensityFit Class Reference

Generalized Density Fitting Scheme. Abstract Base.

```
#include <oep_gdf.h>
```

Inheritance diagram for oepdev::GeneralizedDensityFit:



Public Member Functions

- [GeneralizedDensityFit](#) ()
Constructor. Initializes the pointers.
- virtual [~GeneralizedDensityFit](#) ()
Destructor.
- virtual std::shared_ptr< psi::Matrix > [compute](#) (void)=0
Perform the generalized density fit.
- std::shared_ptr< psi::Matrix > [G](#) (void) const
Extract the G_{ξ_i} coefficients.

Static Public Member Functions

- static std::shared_ptr< [GeneralizedDensityFit](#) > [build](#) (std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::Matrix > v_vector)
Factory for Single GDF Computer.

- static std::shared_ptr< [GeneralizedDensityFit](#) > [build](#) (std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::BasisSet > bs_intermediate, std::shared_ptr< psi::Matrix > v_vector)

Factory for Double GDF Computer.

- static std::shared_ptr< [GeneralizedDensityFit](#) > [build](#) (std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::BasisSet > bs_intermediate, std::shared_ptr< psi::Matrix > v_vector, int dummy)

Factory for Overlap GDF Computer.

Protected Member Functions

- void [invert_matrix](#) (std::shared_ptr< psi::Matrix > &M)
- Invert a square matrix and check if the inverse is acceptable.*

Protected Attributes

- std::shared_ptr< psi::Matrix > [G_](#)
The OEP coefficients $G_{\xi i}$.
- std::shared_ptr< psi::Matrix > [H_](#)
The intermediate DF coefficients for $\hat{v}|i$.
- std::shared_ptr< psi::Matrix > [V_](#)
The V matrix ($\xi|\hat{v}i$).
- int [n_a_](#)
Number of auxiliary basis set functions.
- int [n_i_](#)
Number of intermediate basis set functions.
- int [n_o_](#)
Number of OEP's.
- std::shared_ptr< psi::BasisSet > [bs_a_](#)
Basis set: auxiliary.
- std::shared_ptr< psi::BasisSet > [bs_i_](#)
Basis set: intermediate.
- std::shared_ptr< [oepdev::IntegralFactory](#) > [ints_aa_](#)
Integral factory: aux - aux.
- std::shared_ptr< [oepdev::IntegralFactory](#) > [ints_ai_](#)
Integral factory: aux - int.
- std::shared_ptr< [oepdev::IntegralFactory](#) > [ints_ii_](#)
Integral factory: int - int.

14.39.1 Detailed Description

Performs the following map:

$$\hat{v}|i\rangle \cong \sum_{\eta} G_{\eta i} |\eta\rangle$$

where \hat{v} is the effective one-electron potential (OEP) operator, $|i\rangle$ is an arbitrary state vector and $|\eta\rangle$ is an auxiliary basis vector. The coefficients $G_{\eta i}$ are stored and define the OEP acting on the state i . The mapping onto the auxiliary space can be done in two ways:

- **Single Density Fit.** [This method](#) requires the auxiliary basis set to be nearly complete.
- **Double Density Fit.** [This method](#) can be used to arbitrary auxiliary basis sets.

14.39.2 Member Function Documentation

build() [1/3]

```
std::shared_ptr< GeneralizedDensityFit > GeneralizedDensityFit::build (
    std::shared_ptr< psi::BasisSet > bs_auxiliary,
    std::shared_ptr< psi::Matrix > v_vector ) [static]
```

Parameters

<i>bs_auxiliary</i>	- auxiliary basis set
<i>v_vector</i>	- the matrix with $V_{\xi i}$ elements

Returns

Generalized Density Fit Computer.

build() [2/3]

```
std::shared_ptr< GeneralizedDensityFit > GeneralizedDensityFit::build (
    std::shared_ptr< psi::BasisSet > bs_auxiliary,
    std::shared_ptr< psi::BasisSet > bs_intermediate,
    std::shared_ptr< psi::Matrix > v_vector ) [static]
```

Parameters

<i>bs_auxiliary</i>	- auxiliary basis set
<i>bs_intermediate</i>	- intermediate basis set
<i>v_vector</i>	- the matrix with $V_{\epsilon i}$ elements

Returns

Generalized Density Fit Computer.

build() [3/3]

```
std::shared_ptr< GeneralizedDensityFit > GeneralizedDensityFit::build (
    std::shared_ptr< psi::BasisSet > bs_auxiliary,
    std::shared_ptr< psi::BasisSet > bs_intermediate,
    std::shared_ptr< psi::Matrix > v_vector,
    int dummy ) [static]
```

Parameters

<i>bs_auxiliary</i>	- auxiliary basis set
<i>bs_intermediate</i>	- intermediate basis set
<i>v_vector</i>	- the matrix with $V_{\epsilon i}$ elements
<i>dummy</i>	- a dummy variable (not used)

Returns

Generalized Density Fit Computer.

compute()

```
std::shared_ptr< psi::Matrix > GeneralizedDensityFit::compute (
    void ) [pure virtual]
```

Returns

The OEP coefficients $G_{\xi i}$

Implemented in [oepdev::OverlapGeneralizedDensityFit](#), [oepdev::DoubleGeneralizedDensityFit](#), and [oepdev::SingleGeneralizedDensityFit](#).

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

- [oepdev/liboep/oep_gdf.h](#)
- [oepdev/liboep/oep_gdf.cc](#)

14.40 oepdev::GeneralizedPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::GeneralizedPolarGEFactory:



Classes

- struct [StatisticalSet](#)
A structure to handle statistical data.

Public Member Functions

- [GeneralizedPolarGEFactory](#) (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
Construct from Psi4 wavefunction and options.
- virtual [~GeneralizedPolarGEFactory](#) ()
Destruct.
- virtual std::shared_ptr< [GenEffPar](#) > [compute](#) (void)
Perform Least-Squares Fit.
- bool [has_dipole_polarizability](#) () const
Dipole Polarizability (interacting with \mathbf{F})
- bool [has_dipole_dipole_hyperpolarizability](#) () const
Dipole-Dipole Hyperpolarizability (interacting with \mathbf{F}^2)
- bool [has_quadrupole_polarizability](#) () const
Quadrupole Polarizability (interacting with $\nabla \otimes \mathbf{F}$)
- bool [has_ab_initio_dipole_polarizability](#) () const
Ab Initio Dipole Polarizability (interacting with \mathbf{F})
- double [Zinit](#) () const
Grab initial summaric Z value.
- double [Z](#) () const
Grab final summaric Z value.

Protected Member Functions

- void [allocate](#) (void)
Allocate memory.
- void [invert_hessian](#) (void)
Invert Hessian (do also the identity test)

- void [compute_electric_field_sums](#) (void)
Compute electric field sum set.
- void [compute_electric_field_gradient_sums](#) (void)
Compute electric field gradient sum set.
- void [compute_statistics](#) (void)
Run the statistical evaluation of results.
- void [set_distributed_centres](#) (void)
Set the distributed centres.
- void [compute_parameters](#) (void)
Compute the parameters.
- void [fit](#) (void)
Perform least-squares fit.
- void [compute_ab_initio](#) (void)
Compute ab initio parameters.
- void [save](#) (int i, int j)
Save susceptibility tensors associated with the i-th and j-th basis set function.
- virtual void [compute_samples](#) (void)=0
Compute samples of density matrices and select electric field distributions.
- virtual void [compute_gradient](#) (int i, int j)=0
Compute Gradient vector associated with the i-th and j-th basis set function.
- virtual void [compute_hessian](#) (void)=0
Compute Hessian matrix (independent on the parameters)

Protected Attributes

- int [nBlocks_](#)
Number of parameter blocks.
- int [nSites_](#)
Number of distributed sites.
- int [nSitesAbInitio_](#)
Number of distributed sites of Ab Initio model (FF - single site (com); distributed: LMO sites)
- int [nParameters_](#)
Dimensionality of entire parameter space.
- std::vector< int > [nParametersBlock_](#)
Dimensionality of parameter space per block.
- const int [nSamples_](#)
Number of statistical samples.
- const double [symmetryNumber_](#) [6]
Symmetry number for matrix susceptibilities.
- std::shared_ptr< psi::Matrix > [Gradient_](#)

- Gradient.*
- `std::shared_ptr< psi::Matrix > Hessian_`
Hessian.
- `std::shared_ptr< psi::Matrix > Parameters_`
Parameters.
- `std::shared_ptr< oepdev::GenEffPar > PolarizationSusceptibilities_`
Density Matrix Susceptibility Tensors Object.
- `std::shared_ptr< oepdev::GenEffPar > abInitioPolarizationSusceptibilities_`
Density Matrix Susceptibility Tensors Object for Ab Initio Model.
- `bool hasDipolePolarizability_`
Has Dipole Polarizability?
- `bool hasDipoleDipoleHyperpolarizability_`
Has Dipole-Dipole Hyperpolarizability?
- `bool hasQuadrupolePolarizability_`
Has Quadrupole Polarizability?
- `bool hasAbInitioDipolePolarizability_`
Has Ab Initio Dipole Polarizability?
- `StatisticalSet referenceStatisticalSet_`
Reference statistical data.
- `StatisticalSet referenceDpolStatisticalSet_`
Multipole reference statistical data.
- `StatisticalSet modelStatisticalSet_`
Model statistical data.
- `StatisticalSet abInitioModelStatisticalSet_`
Ab Initio Model statistical data.
- `std::vector< std::shared_ptr< psi::Matrix > > VMatrixSet_`
Potential matrix set.
- `std::vector< std::vector< std::shared_ptr< Vector > > > electricFieldSet_`
Electric field set.
- `std::vector< std::vector< std::shared_ptr< Matrix > > > electricFieldGradientSet_`
Electric field gradient set.
- `std::vector< std::vector< double > > electricFieldSumSet_`
Electric field sum set.
- `std::vector< std::vector< std::shared_ptr< psi::Vector > > > electricFieldGradientSum-Set_`
Electric field gradient sum set.
- `std::vector< std::vector< std::shared_ptr< Vector > > > abInitioModelElectricFieldSet_`
Electric field set for Ab Initio Model (LMO-distributed)
- `const double mField_`
Level shifters for Hessian blocks.

- double [Zinit_](#)
Initial summaric Z value.
- double [Z_](#)
Final summaric Z value.
- std::shared_ptr< psi::JK > [jk_](#)
Computer of generalized JK objects.

Additional Inherited Members

14.40.1 Detailed Description

Implements a general class of methods for the density matrix susceptibility tensors represented by:

$$\delta D_{\alpha\beta} = \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(20)} : \mathbf{F}(\mathbf{r}_i) \otimes \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(01)} : \nabla_i \otimes \mathbf{F}(\mathbf{r}_i) + \dots \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$ is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(20)}$ is the density matrix dipole-dipole hyperpolarizability
- $\mathbf{B}_{i;\alpha\beta}^{(01)}$ is the density matrix quadrupole polarizability

all defined for the generalized distributed site at \mathbf{r}_i .

Available models:

1. Training against uniform electric fields

- [oepdev::LinearUniformEFieldPolarGEFactory](#) - linear with respect to electric field
- [oepdev::QuadraticUniformEFieldPolarGEFactory](#) - quadratic with respect to electric field

2. Training against non-uniform electric fields

- [oepdev::LinearNonUniformEFieldPolarGEFactory](#) - linear with respect to electric field, distributed site model
- [oepdev::QuadraticNonUniformEFieldPolarGEFactory](#) - quadratic with respect to electric field, distributed site model
- [oepdev::LinearGradientNonUniformEFieldPolarGEFactory](#) - linear with respect to electric field and linear with respect to electric field gradient, distributed site model. This model does not function now.
- [oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory](#) - linear with respect to electric field and linear with respect to electric field gradient, distributed site model. This model does not function now.

For the non-linear field training, a set of point charges in each training sample is assumed. Distributed models use atomic centers as expansion points.

Determination of the generalized susceptibilities

Let $\{\mathbf{F}^{(1)}(\mathbf{r}), \mathbf{F}^{(2)}(\mathbf{r}), \dots, \mathbf{F}^{(N)}(\mathbf{r}), \dots\}$ be a set of N_{\max} distinct and randomly sampled spatial distributions of electric field. It is assumed that the exact difference one-particle density matrices (with respect to the unperturbed state) defined as

$$\delta\overline{\mathbf{D}}^{(N)} \equiv \overline{\mathbf{D}}^{(N)} - \overline{\mathbf{D}}^{(0)}$$

are known for each sample (overline symbolizes the exact estimate). Now, for each pair of the AO indices the following parameterization is constructed:

$$\delta D^{(N)} = \sum_i^M \left\{ \sum_u^{x,y,z} s_{iu}^{[1]} F_{iu}^{(N)} + \sum_u^{x,y,z} \sum_{w < u} r_{uw} s_{iww}^{[2]} F_{iu}^{(N)} F_{iw}^{(N)} + \dots \right\}$$

(the Greek subscripts were omitted here for notational simplicity). In the above equation, $B_u^{(i;1)} = s_{iu}^{[1]}$ and $B_{uw}^{(i;2)} = r_{uw} s_{iww}^{[2]}$, where r_{uw} is the symmetry factor equal to 1 for diagonal elements and 2 for off-diagonal elements of $B_{uw}^{(i;2)}$. The multiple parameter blocks ($s^{[1]}$, $s^{[2]}$ and so on) appear in the first power, allowing for linear least-squares regression. The square bracket superscripts denote the block of the parameter space.

To determine the optimum set, $\mathbf{s} = (s^{[1]} \ s^{[2]} \ \dots)^T$, a loss function Z that is subject to the least-squares minimization, is defined as

$$Z(\mathbf{s}) = \sum_N^{N_{\max}} \left(\delta D^{(N)} - \delta\overline{D}^{(N)} \right)^2.$$

The Hessian of Z computed with respect to the parameters is parameter-independent (constant) and generally non-singular as long as the electric fields on all distributed sites are different. Therefore, the exact solution for the optimal parameters is given by the Newton equation

$$\mathbf{s} = -\mathbf{H}^{-1} \cdot \mathbf{g},$$

where \mathbf{g} and \mathbf{H} are the gradient vector and the Hessian matrix, respectively. Note that in this case the dimensions of parameter space for the block 1 and 2 are equal to $3M$ and $6M$, respectively. The explicit forms of the gradient and Hessian up to second-order are given in the next section.

Explicit Formulae for Gradient and Hessian Blocks in Linear Regression DMS Model

The gradient vector \mathbf{g} and the Hessian matrix \mathbf{H} are built from blocks associated with a particular type of parameters, i.e.,

$$\mathbf{g} = \begin{pmatrix} \mathbf{g}^{[1]} \\ \mathbf{g}^{[2]} \end{pmatrix}, \quad \mathbf{H} = \begin{pmatrix} \mathbf{H}^{[11]} & \mathbf{H}^{[12]} \\ \mathbf{H}^{[21]} & \mathbf{H}^{[22]} \end{pmatrix},$$

where the block indices 1 and 2 correspond to the first- and second-order susceptibilities, respectively. Note that the second derivatives of $\delta D^{(N)}$ with respect to the adjustable parameters vanish due to the linear functional form of the parameterization formula given in the previous section. Thus, the gradient element of the r -th block and Hessian element of the (rs) -th block read

$$g^{[r]} \equiv \frac{\partial Z}{\partial s^{[r]}} = -2 \sum_N \overline{\delta D}^{(N)} \frac{\partial [\delta D^{(N)}]}{\partial s^{[r]}},$$

$$H^{[rs]} \equiv \frac{\partial^2 Z}{\partial s^{[r]} \partial s^{[s]}} = 2 \sum_N \frac{\partial [\delta D^{(N)}]}{\partial s^{[r]}} \frac{\partial [\delta D^{(N)}]}{\partial s^{[s]}}.$$

The explicit formulae for the gradient are

$$g_{ku}^{[1]} = -2 \sum_N \overline{\delta D}^{(N)} F_{ku}^{(N)} ,$$

$$g_{kuw}^{[2]} = -2 r_{uw} \sum_N \overline{\delta D}^{(N)} F_{ku}^{(N)} F_{kw}^{(N)} .$$

The Hessian subsequently follows to be %

$$H_{ku,lw}^{[11]} = 2 \sum_N F_{ku}^{(N)} F_{lw}^{(N)} ,$$

$$H_{ku,lu'w'}^{[12]} = 2 r_{u'w'} \sum_N F_{ku}^{(N)} F_{lu'}^{(N)} F_{lw'}^{(N)} ,$$

$$H_{kuw,lu'w'}^{[22]} = 2 r_{uw} r_{u'w'} \sum_N F_{ku}^{(N)} F_{kw}^{(N)} F_{lu'}^{(N)} F_{lw'}^{(N)} .$$

Note that due to the symmetry of the Hessian matrix, the block 21 is a transpose of the block 12. The composite indices ku and kuw are constructed from the distributed site index k and the appropriate symmetry-adapted ($w < u$) Cartesian component of a particular DMS tensor: u for the first-order, and uw for the second-order susceptibility tensor, respectively. The method described above can be easily extended to third and higher orders.

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

- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp_polar_base.cc

14.41 oepdev::GramSchmidt Class Reference

Gram-Schmidt orthogonalization method.

```
#include <gram_schmidt.h>
```

Public Member Functions

- [GramSchmidt](#) ()
Construct the blank Gram-Schmidt Orthonormalizer.
- [GramSchmidt](#) (std::vector< psi::SharedVector > vectors)
Construct the Gram-Schmidt Orthonormalizer.
- virtual [~GramSchmidt](#) ()
Destructor.
- virtual std::vector< psi::SharedVector > [V](#) (void) const
Retrieve all the vectors.
- virtual int [L](#) (void) const
Retrieve the number of vectors.
- virtual psi::SharedVector [V](#) (int i) const

- Retrieve the i -th vector.*

 - void `normalize` (void)

Normalize all the vectors.

 - void `orthonormalize` (void)

Orthonormalize all the vectors.

 - void `orthogonalize` (void)

Orthogonalize all the vectors.

 - void `orthogonalize_vector` (psi::SharedVector &d, bool `normalize`=false) const

*Orthogonalize vector with respect to the vector set. Modifies **d**.*

 - psi::SharedVector `projection` (psi::SharedVector u, psi::SharedVector v) const
 - void `append` (psi::SharedVector d)

Append new vector to the list.

 - void `reset` (std::vector< psi::SharedVector > `V`)

Reset by providing new vectors.

 - void `reset` (void)

Reset to empty state.

Protected Attributes

- std::vector< psi::SharedVector > `V_`
- Vectors stored.*
- int `L_`
- Number of vectors.*

14.41.1 Detailed Description

Orthonormalize a set of L vectors, i.e.,

$$\{\mathbf{v}_k\} \rightarrow \{\mathbf{u}_k\} \text{ for } k = 1, 2, \dots, L$$

Implementation

The orthogonalized vectors are generated according to

$$\mathbf{u}_k = \left[1 - \sum_{i=1}^{k-1} \hat{P}_{\mathbf{u}_i} \right] \mathbf{v}_k$$

where the projection operator is given by

$$\hat{P}_{\mathbf{u}} = \frac{1}{u^2} \mathbf{u} [\square \cdot \mathbf{u}]$$

14.41.2 Constructor & Destructor Documentation

GramSchmidt() [1/2]

```
oepdev::GramSchmidt::GramSchmidt ( )
```

GramSchmidt() [2/2]

```
oepdev::GramSchmidt::GramSchmidt (
    std::vector< psi::SharedVector > vectors )
```

Parameters

<i>vectors</i>	- list of vectors to be orthogonalized.
----------------	---

14.41.3 Member Function Documentation

projection()

```
psi::SharedVector oepdev::GramSchmidt::projection (
    psi::SharedVector u,
    psi::SharedVector v ) const
```

Compute the projection vector.

Parameters

<i>u</i>	- projected direction
<i>v</i>	- projected vector

Returns

a new vector \mathbf{v}' such that

$$\mathbf{v}' = \hat{P}_{\mathbf{u}} \mathbf{v}$$

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

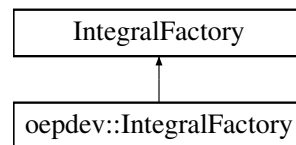
- oepdev/libutil/[gram_schmidt.h](#)
- oepdev/libutil/gram_schmidt.cc

14.42 oepdev::IntegralFactory Class Reference

Extended [IntegralFactory](#) for computing integrals.

```
#include <integral.h>
```

Inheritance diagram for oepdev::IntegralFactory:



Public Member Functions

- [IntegralFactory](#) (std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, std::shared_ptr< psi::BasisSet > bs3, std::shared_ptr< psi::BasisSet > bs4)
Initialize integral factory given a BasisSet for each center. Becomes (bs1 bs2 | bs3 bs4).
- [IntegralFactory](#) (std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2)
Initialize integral factory given a BasisSet for two centres. Becomes (bs1 bs2 | bs1 bs2).
- [IntegralFactory](#) (std::shared_ptr< psi::BasisSet > bs1)
Initialize integral factory given a BasisSet for two centres. Becomes (bs1 bs1 | bs1 bs1).
- virtual [~IntegralFactory](#) ()
Destructor.
- virtual psi::OneBodyAOInt * [ao_efp_multipole_potential_new](#) (int max_k=3, int deriv=0)
Returns an improved [EFPMultipolePotentialInt](#).
- virtual [oepdev::TwoBodyAOInt](#) * [eri_1_1](#) (int deriv=0, bool use_shell_pairs=false)
Returns an [ERI_1_1](#) integral object.
- virtual [oepdev::TwoBodyAOInt](#) * [eri_2_1](#) (int deriv=0, bool use_shell_pairs=false)
Returns an [ERI_2_1](#) integral object.
- virtual [oepdev::TwoBodyAOInt](#) * [eri_2_2](#) (int deriv=0, bool use_shell_pairs=false)
Returns an [ERI_2_2](#) integral object.
- virtual [oepdev::TwoBodyAOInt](#) * [eri_3_1](#) (int deriv=0, bool use_shell_pairs=false)
Returns an [ERI_3_1](#) integral object.

14.42.1 Detailed Description

In addition to integrals available in Psi4, [oepdev::IntegralFactory](#) enables to compute also:

- OEI's:
 - none at that moment

- ERI's:
 - integrals of type (a|b) - `oepdev::ERI_1_1`
 - integrals of type (ab|c) - `oepdev::ERI_2_1`
 - integrals of type (abc|d) - `oepdev::ERI_3_1`
 - integrals of type (ab|cd) - `oepdev::ERI_2_2` (also in Psi4 as `psi::ERI`)

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

- `oepdev/libpsi/integral.h`
- `oepdev/libpsi/integral.cc`

14.43 oepdev::KabschSuperimposer Class Reference

Compute the Cartesian rotation matrix between two structures.

```
#include <kabsch_superimposer.h>
```

Public Member Functions

- `KabschSuperimposer ()`
Constructor.
- `~KabschSuperimposer ()`
Destructor.
- `void compute (psi::SharedMatrix initial_xyz, psi::SharedMatrix final_xyz)`
Run the Kabsch algorithm.
- `void compute (psi::SharedMolecule initial_mol, psi::SharedMolecule final_mol)`
Run the Kabsch algorithm.
- `psi::SharedMatrix get_transformed (void)`
Return transformed coordinates \mathbf{X}' .
- `double rms (void)`
Compute RMS or superimposition.
- `void clear (void)`
Clear all previous calculations.

Public Attributes

- `psi::SharedMatrix rotation`
Rotation matrix \mathbf{r} .
- `psi::SharedVector translation`
Translation vector \mathbf{t} .

- psi::SharedMatrix [initial_xyz](#)
Initial xyz \mathbf{X} .
- psi::SharedMatrix [final_xyz](#)
Final xyz \mathbf{X}_0 .

14.43.1 Detailed Description

The superimposition is defined as:

$$\mathbf{X}' = \mathbf{t} + \mathbf{X} \cdot \mathbf{r} \approx \mathbf{X}_0$$

where X_{iu} is the u -th Cartesian component of the i -th atom's position, \mathbf{t} is the superimposition translation vector, \mathbf{r} is the superimposition rotation matrix, and prime denotes transformed coordinates.

The superimposition uses the Kabsch algorithm.

The Kabsch Algorithm.

Rotation matrix is calculated from

$$\mathbf{r} = \mathbf{U} \cdot \mathbf{V}^T$$

where

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{S} \cdot \mathbf{V}^T$$

is the singular value decomposition of the covariance matrix

$$\mathbf{A} = [\mathbf{X} - \langle \mathbf{X} \rangle]^T \cdot [\mathbf{X}_0 - \langle \mathbf{X}_0 \rangle]$$

The average of position is given by

$$\langle \mathbf{X} \rangle_u = \frac{1}{N} \sum_i X_{iu}$$

where N is the number of atoms. If determinant of rotation matrix is negative (indicating inversion), rotation matrix is recomputed by inverting the sign of the third column of \mathbf{V} .

The translation vector is then calculated by

$$\mathbf{t} = \langle \mathbf{X}_0 \rangle - \langle \mathbf{X} \rangle \cdot \mathbf{r}$$

14.43.2 Member Function Documentation

compute() [1/2]

```
void oepdev::KabschSuperimposer::compute (
    psi::SharedMatrix initial_xyz,
    psi::SharedMatrix final_xyz )
```

Parameters

<i>initial_xyz</i>	- position vectors \mathbf{X}
<i>final_xyz</i>	- position vectors \mathbf{X}_0

`compute()` [2/2]

```
void oepdev::KabschSuperimposer::compute (
    psi::SharedMolecule initial_mol,
    psi::SharedMolecule final_mol ) [inline]
```

Parameters

<i>initial_mol</i>	- molecule with atomic positions at \mathbf{X}
<i>final_mol</i>	- molecule with atomic positions at \mathbf{X}_0

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

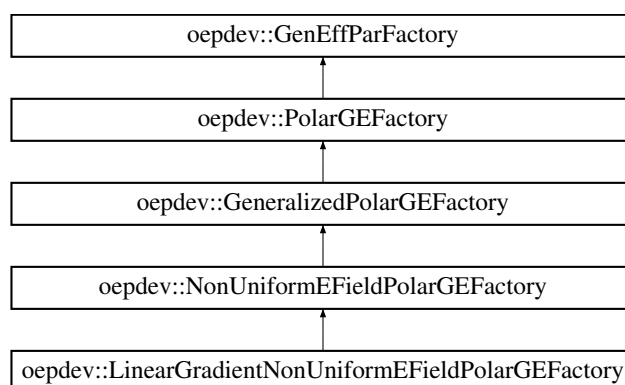
- oepdev/libutil/[kabsch_superimposer.h](#)
- oepdev/libutil/kabsch_superimposer.cc

14.44 oepdev::LinearGradientNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::LinearGradientNonUniformEFieldPolarGEFactory:



Public Member Functions

- **LinearGradientNonUniformEFieldPolarGEFactory** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)
- void **compute_gradient** (int i, int j)
Compute Gradient vector associated with the i-th and j-th basis set function.
- void **compute_hessian** (void)
Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.44.1 Detailed Description

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(01)} : \nabla_i \otimes \mathbf{F}(\mathbf{r}_i) \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$ is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(01)}$ is the density matrix quadrupole polarizability all defined for the distributed site at \mathbf{r}_i .

Note

This model is not available now and probably will be deprecated in the future.

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

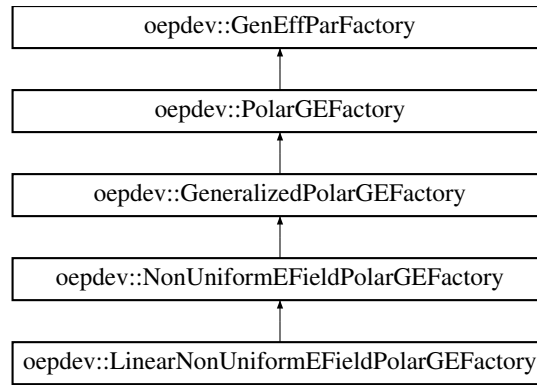
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp_polar_nonuniform_field_1_grad_1.cc

14.45 oepdev::LinearNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::LinearNonUniformEFieldPolarGEFactory:



Public Member Functions

- **LinearNonUniformEFieldPolarGEFactory** (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute_gradient](#) (int i, int j)
Compute Gradient vector associated with the i-th and j-th basis set function.
- void [compute_hessian](#) (void)
Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.45.1 Detailed Description

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i)$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$ is the density matrix dipole polarizability defined for the distributed site at \mathbf{r}_i .

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

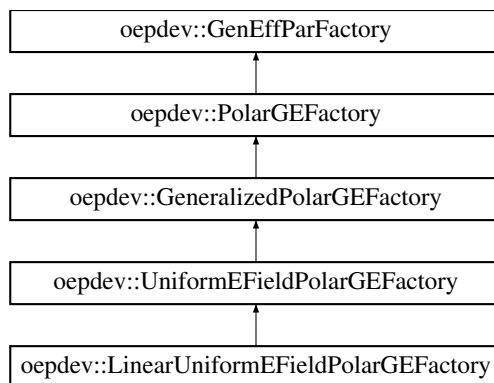
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp_polar_nonuniform_field_1.cc

14.46 oepdev::LinearUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::LinearUniformEFieldPolarGEFactory:



Public Member Functions

- **LinearUniformEFieldPolarGEFactory** (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute_gradient](#) (int i, int j)
Compute Gradient vector associated with the i-th and j-th basis set function.
- void [compute_hessian](#) (void)
Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.46.1 Detailed Description

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \mathbf{B}_{\alpha\beta}^{(10)} \cdot \mathbf{F}$$

where:

- $\mathbf{B}_{\alpha\beta}^{(10)}$ is the density matrix dipole polarizability

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

- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp_polar_uniform_field_1.cc

14.47 oepdev::MultipoleConvergence Class Reference

Multipole Convergence.

```
#include <dmt.h>
```


Public Types

- enum [ConvergenceLevel](#) {
 R1, R2, R3, R4,
 R5 }
- enum [Property](#) { **Energy, Potential, Field** }

Public Member Functions

- [MultipoleConvergence](#) (std::shared_ptr< [DMTPole](#) > dmt1, std::shared_ptr< [DMTPole](#) > dmt2, [ConvergenceLevel](#) max_clevel=R5)
Construct from two shared [DMTPole](#) objects.
- virtual [~MultipoleConvergence](#) ()
Destructor.
- void [compute](#) ([Property](#) property=Energy)
- void [compute](#) (const double &x, const double &y, const double &z, [Property](#) property=Potential)
- std::shared_ptr< psi::Matrix > [level](#) ([ConvergenceLevel](#) clevel=R5)

Protected Member Functions

- void [compute_energy](#) ()
Compute the generalized energy.
- void [compute_potential](#) (const double &x, const double &y, const double &z)
Compute the generalized potential.
- void [compute_field](#) (const double &x, const double &y, const double &z)
Compute the generalized field potential.

Protected Attributes

- [ConvergenceLevel max_clevel_](#)
Maximum allowed convergence level.
- std::shared_ptr< [DMTPole](#) > [dmt1_](#)
First DMTP set.
- std::shared_ptr< [DMTPole](#) > [dmt2_](#)
Second DMTP set.
- std::map< std::string, std::shared_ptr< psi::Matrix > > [convergenceList_](#)
Dictionary of available convergence level results.
- std::map< std::string, std::shared_ptr< psi::Matrix > > [energyConvergencePairs_](#)
Dictionary of available energy convergence pairs.

14.47.1 Detailed Description

Handles the convergence of the distributed multipole expansions up to hexadecapole. Takes shared pointers to existing [DMTPole](#) objects and computes the generalized property:

- energy
- potential from the DMTP sets. The results are stored in matrix of size (N1, N2) where N1 and N2 are equal to the number of DMTP's in a set described by according [DMTPole](#) object given.

Note

Useful options:

- DMTP_CONVER - level of multipole series convergence (available: R1, R2, R3, R4 and R5). Default: R5.

See also

[DMTPole](#)

14.47.2 Member Enumeration Documentation

ConvergenceLevel

```
enum oepdev::MultipoleConvergence::ConvergenceLevel
```

Convergence level of the multipole expansion:

Parameters

<i>R1</i>	- qq term
<i>R2</i>	- qd and sum of the above
<i>R3</i>	- qQ, dd and sum of the above
<i>R4</i>	- qO, dQ and sum of the above
<i>R5</i>	- qH, dO, QQ and sum of the above

Property

```
enum oepdev::MultipoleConvergence::Property
```

Property to be evaluated from DMTP's:

Parameters

<i>Energy</i>	- generalized energy
<i>Field</i>	- generalized field
<i>Potential</i>	- generalized potential

14.47.3 Constructor & Destructor Documentation

MultipoleConvergence()

```
oepdev::MultipoleConvergence::MultipoleConvergence (
    std::shared_ptr< DMTPole > dmt1,
    std::shared_ptr< DMTPole > dmt2,
    MultipoleConvergence::ConvergenceLevel max_clevel = R5 )
```

Parameters

<i>dmt1</i>	- first DMTPole object
<i>dmt2</i>	- second DMTPole object
<i>max_clevel</i>	- maximul allowed convergence level

14.47.4 Member Function Documentation

compute() [1/2]

```
void oepdev::MultipoleConvergence::compute (
    MultipoleConvergence::Property property = Energy )
```

Compute the generalized interaction property

Parameters

<i>property</i>	- generalized Property
-----------------	------------------------

compute() [2/2]

```
void oepdev::MultipoleConvergence::compute (
```

```

const double & x,
const double & y,
const double & z,
MultipoleConvergence::Property property = Potential )

```

Compute the generalized generator property

Parameters

<i>x</i>	- location x-th Cartesian component
<i>y</i>	- location y-th Cartesian component
<i>z</i>	- location z-th Cartesian component
<i>property</i>	- generalized Property

level()

```

std::shared_ptr< psi::Matrix > oepdev::MultipoleConvergence::level (
    MultipoleConvergence::ConvergenceLevel clevel = R5 )

```

Grab the generalized property at specified level of convergence

Parameters

<i>clevel</i>	- ConvergenceLevel
---------------	--------------------

Returns

vector of results (each element corresponds to each DMTP pair in a set)

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

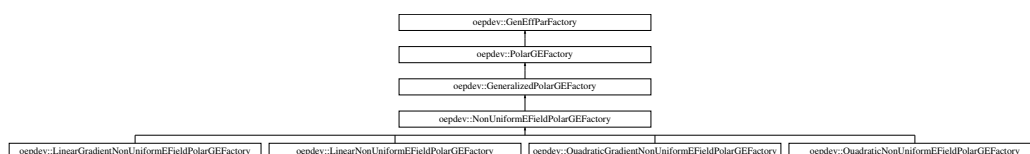
- oepdev/lib3d/dmtp.h
- oepdev/lib3d/dmtp_base.cc

14.48 oepdev::NonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::NonUniformEFieldPolarGEFactory:



Public Member Functions

- **NonUniformEFieldPolarGEFactory** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)
- void [compute_samples](#) (void)
Compute samples of density matrices and select electric field distributions.
- virtual void [compute_gradient](#) (int i, int j)=0
Compute Gradient vector associated with the i-th and j-th basis set function.
- virtual void [compute_hessian](#) (void)=0
Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.48.1 Detailed Description

Implements a class of density matrix susceptibility models for parameterization in the non-uniform electric field generated by point charges.

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

- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp_polar_nonuniform_base.cc

14.49 oepdev::ObaraSaikaTwoCenterEFPRecursion_New Class Reference

Obara-Saika recursion formulae for improved EFP multipole potential integrals.

```
#include <osrecur.h>
```

Public Member Functions

- [ObaraSaikaTwoCenterEFPRecursion_New](#) & **operator=** (const [ObaraSaikaTwoCenterEFPRecursion_New](#) &)
- [ObaraSaikaTwoCenterEFPRecursion_New](#) (int max_am1, int max_am2, int max_k)
- double *** [q](#) () const
Returns the potential integral 3D matrix.
- double *** [x](#) () const
- double *** [y](#) () const
- double *** [z](#) () const
- double *** [xx](#) () const
- double *** [yy](#) () const
- double *** [zz](#) () const
- double *** [xy](#) () const

- double *** **xz** () const
- double *** **yz** () const
- double *** **xxx** () const
- double *** **yyy** () const
- double *** **zzz** () const
- double *** **xyx** () const
- double *** **xxz** () const
- double *** **xyy** () const
- double *** **yyz** () const
- double *** **xzz** () const
- double *** **yyz** () const
- double *** **xyz** () const
- virtual void [compute](#) (double PA[3], double PB[3], double PC[3], double zeta, int am1, int am2)

Computes the potential integral 3D matrix using the data provided.

Protected Member Functions

- void **calculate_f** (double *F, int n, double t)

Protected Attributes

- int **max_am1_**
- int **max_am2_**
- int **size_**
- bool **do_octupoles_**
- double *** **q_**
- double *** **x_**
- double *** **y_**
- double *** **z_**
- double *** **xx_**
- double *** **xy_**
- double *** **xz_**
- double *** **yy_**
- double *** **yz_**
- double *** **zz_**
- double *** **xxx_**
- double *** **xyx_**
- double *** **xxz_**
- double *** **xyy_**
- double *** **xyz_**
- double *** **xzz_**

- double *** **yyy**_
- double *** **yyz**_
- double *** **yzz**_
- double *** **zzz**_

14.49.1 Constructor & Destructor Documentation

ObaraSaikaTwoCenterEFPRecursion.New()

```
oepdev::ObaraSaikaTwoCenterEFPRecursion.New::ObaraSaikaTwoCenterEFPRecursion.New
(
    int max_am1,
    int max_am2,
    int max_k )
```

Constructor, max_am1 and max_am2 are the max angular momentum on center 1 and 2. Needed to allocate enough memory.

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

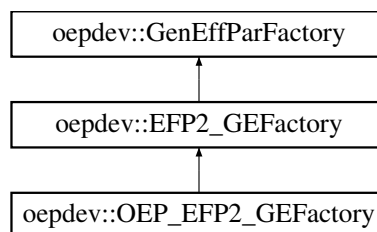
- oepdev/libpsi/[osrecur.h](#)
- oepdev/libpsi/osrecur.cc

14.50 oepdev::OEP_EFP2_GEFactory Class Reference

OEP-EFP2 GEFP Factory.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::OEP_EFP2_GEFactory:



Public Member Functions

- [OEP_EFP2_GEFactory](#) (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
Construct from Psi4 options.

- [OEP_EFP2_GEFactory](#) (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt, psi::SharedBasisSet aux, psi::SharedBasisSet intermed)
Construct from Psi4 options and additional basis sets.
- virtual [~OEP_EFP2_GEFactory](#) ()
Destruct.
- virtual std::shared_ptr< [GenEffPar](#) > [compute](#) (void)
Compute the OEP-EFP2 parameters.

Protected Member Functions

- virtual void **assemble_canonical_orbitals** (void) override
- virtual void **assemble_oep_efp2_parameters** (void)
- virtual void **assemble_oep_lmo_centroids** (void)

Protected Attributes

- psi::SharedBasisSet **auxiliary_**
- psi::SharedBasisSet **intermediate_**
- oepdev::SharedOEPotential **oep_rep_**
- oepdev::SharedOEPotential **oep_ct_**

Additional Inherited Members

14.50.1 Detailed Description

Basic interface for the OEP-EFP2 parameters.

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

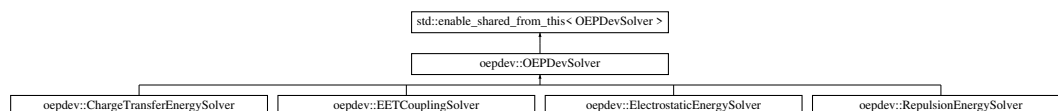
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp_oep_efp2.cc

14.51 oepdev::OEPDevSolver Class Reference

Solver of properties of molecular aggregates. Abstract base.

```
#include <solver.h>
```

Inheritance diagram for oepdev::OEPDevSolver:



Public Member Functions

- [OEPDevSolver](#) ([SharedWavefunctionUnion](#) wfn_union)
Take wavefunction union and initialize the Solver.
- virtual [~OEPDevSolver](#) ()
Destructor.
- virtual double [compute_oep_based](#) (const std::string &method="DEFAULT")=0
Compute property by using OEPs.
- virtual double [compute_benchmark](#) (const std::string &method="DEFAULT")=0
Compute property by using benchmark method.

Static Public Member Functions

- static std::shared_ptr< [OEPDevSolver](#) > [build](#) (const std::string &target, [SharedWavefunctionUnion](#) wfn_union)
Build a solver of a particular property for given molecular cluster.

Protected Attributes

- [SharedWavefunctionUnion](#) wfn_union_
Wavefunction union.
- psi::Options & [options_](#)
Options.
- std::vector< std::string > [methods_oepBased_](#)
Names of all OEP-based methods implemented for a solver.
- std::vector< std::string > [methods_benchmark_](#)
Names of all benchmark methods implemented for a solver.

14.51.1 Detailed Description

Uses only a wavefunction union object to initialize.

Available solvers

- ELECTROSTATIC ENERGY
- REPULSION ENERGY
- CHARGE TRANSFER ENERGY
- EET COUPLING CONSTANT

Options

Interaction Property Method

- `OEPDEV_SOLVER_EINT_COUL_AO` - Coulombic energy: AO expanded
- `OEPDEV_SOLVER_EINT_COUL_MO` - Coulombic energy: MO expanded
- `OEPDEV_SOLVER_EINT_COUL_ESP` - Coulombic energy: ESP
- `OEPDEV_SOLVER_EINT_COUL_CAMM` - Coulombic energy: [CAMM](#)
- `OEPDEV_SOLVER_EINT_REP_HS` - Exchange-repulsion energy: Hayes-Stone
- `OEPDEV_SOLVER_EINT_REP_DDS` - Exchange-repulsion energy: DDS
- `OEPDEV_SOLVER_EINT_REP_MRW` - Exchange-repulsion energy: Murrell et al.
- `OEPDEV_SOLVER_EINT_REP_OL` - Exchange-repulsion energy: Otto-Ladik
- `OEPDEV_SOLVER_EINT_REP_OEP1` - Exchange-repulsion energy: OEP (S1: GDF, S2: ESP)
- `OEPDEV_SOLVER_EINT_REP_OEP2` - Exchange-repulsion energy: OEP (S1: GDF, S2: [CAMM](#))
- `OEPDEV_SOLVER_EINT_REP_EFP2` - Exchange-repulsion energy: EFP2
- `OEPDEV_SOLVER_EINT_CT_OL` - Charge-transfer energy: Otto-Ladik
- `OEPDEV_SOLVER_EINT_CT_OEP` - Charge-transfer energy: OEP
- `OEPDEV_SOLVER_EINT_CT_EFP2` - Charge-transfer energy: EFP2

Generalized density fitting (GDF) options:

- `OEPDEV_DF_TYPE` - type of the GDF. Default: `DOUBLE`. Other: `SINGLE`.
- `DF_BASIS_OEP` - auxiliary basis set. Default: `sto-3g`.
- `DF_BASIS_INT` - intermediate basis set. Relevant only if double GDF is used. Default: `aug-cc-pVDZ-jkfit`. Note that intermediate basis set should be nearly complete.

EFP2 Charge transfer energy options:

- `EFP2_CT_POTENTIAL_INTS` - Type of potential one-electron operator. Default: `'DMTP'`. Other: `'ERI'`.
- `EFP2_CT_NO_OCTUPOLES` - Ignore octupole moments from potential integrals? Default: `True`.

Excited States

- `EXCITED_STATE` - ID of state for all monomers to consider. If $-n$, then the n th bright state is taken. Default: -1 .
- `EXCITED_STATE_A` - ID of state for monomer A to consider. If $-n$, then the n th bright state is taken. Default: -1 .
- `EXCITED_STATE_B` - ID of state for monomer B to consider. If $-n$, then the n th bright state is taken. Default: -1 .
- `OSCILLATOR_STRENGTH_THRESHOLD` - Threshold for oscillator strength for bright states selection. Default: 0.01 .
- `TrCAMM_SYMMETRIZE` - Whether to use the 'symmetrized transition density' or not. Default: `true`.
- `TI_CIS_SCF_FOCK_MATRIX` - Whether to compute the full SCF Fock matrix for the dimer or approximate it from monomer OPDM's. Default: `false`.
- `TI_CIS_PRINT_FOCK_MATRIX` - Whether to print the Fock matrix (AB block in AO basis) or not. Default: `false`.

Environmental variables

One can easily access those variables from Python level by calling

```
psi4.get_variable("name of variable")
```

in your Python script.

Table 14.95: Environmental variables in the OEPDev solver.

Keyword	Description
Coulombic Interaction Energy	
<i>Distributed Multipole Series</i>	
EINT COUL CAMM R-1	CAMM charge-charge terms
EINT COUL CAMM R-2	CAMM charge-dipole terms + all above
EINT COUL CAMM R-3	CAMM charge-quadrupole, dipole-dipole + all above
EINT COUL CAMM R-4	CAMM charge-octupole, dipole-quadrupole + all above
EINT COUL CAMM R-5	CAMM charge-hexadecapole, dipole-octupole, quadrupole-quadrupole + all above

Keyword	Description
EINT COUL ESP	ESP charge-charge terms
<i>Exact First-Order Perturbation Theory</i>	
EINT COUL EXACT	MO or AO expanded Coulombic energy. Both give same results but MO is much faster.
Exchange-Repulsion Interaction Energy	
<i>Density Decomposition Scheme</i>	
EINT REP DDS KCAL	Pauli repulsion
EINT EXC DDS KCAL	DDS exchange
EINT EXR DDS KCAL	Sum of the above
<i>Hayes-Stone model</i>	
EINT REP HAYES-STONE KCAL	Pauli repulsion
EINT EXC HAYES-STONE KCAL	Pure exchange
EINT EXR HAYES-STONE KCAL	Sum of the above
<i>Murrell et al. model</i>	
EINT REP MURRELL-ETAL KCAL	Pauli repulsion
EINT EXC MURRELL-ETAL KCAL	Pure exchange (same as Hayes-Stone)
EINT EXR MURRELL-ETAL KCAL	Sum of the above
EINT REP MURRELL-ETAL:S1 KCAL	Pauli repulsion: $S^{\{-1\}}$ term
EINT REP MURRELL-ETAL:S2 KCAL	Pauli repulsion: $S^{\{-2\}}$ term
<i>Otto-Ladik model</i>	
EINT REP OTTO-LADIK KCAL	Pauli repulsion
EINT EXC OTTO-LADIK KCAL	Pure exchange (same as Hayes-Stone)
EINT EXR OTTO-LADIK KCAL	Sum of the above
EINT REP OTTO-LADIK:S1 KCAL	Pauli repulsion: $S^{\{-1\}}$ term
EINT REP OTTO-LADIK:S2 KCAL	Pauli repulsion: $S^{\{-2\}}$ term
<i>EFP2 model</i>	
EINT REP EFP2 KCAL	Pauli repulsion
EINT EXC EFP2 KCAL	Exchange: SGO approximation of Jensen
EINT EXR EFP2 KCAL	Sum of the above
EINT REP EFP2:S1 KCAL	Pauli repulsion: $S^{\{-1\}}$ term

Keyword	Description
EINT REP EFP2:S2 KCAL	Pauli repulsion: S^{-2} term
OEP-based models	
EINT REP OEP-MURRELL-ETAL-1 KCAL	Pauli repulsion: S1 term using GDF, S2 term using CAMM
EINT REP OEP-MURRELL-ETAL-1 S1 KCAL	S^{-1} term of the above total term
EINT REP OEP-MURRELL-ETAL-1 S2 KCAL	S^{-2} term of the above total term
EINT REP OEP-MURRELL-ETAL-2 KCAL	Pauli repulsion: S1 term using GDF, S2 term using ESP
EINT REP OEP-MURRELL-ETAL-2 S1 KCAL	S^{-1} term of the above total term
EINT REP OEP-MURRELL-ETAL-2 S1 KCAL	S^{-2} term of the above total term
Charge-Transfer Interaction Energy	
EFP2 Model	
EINT CT EFP2 KCAL	Total charge-transfer energy (kcal/mole)
Otto-Ladik Model	
EINT CT OTTO-LADIK KCAL	Total charge-transfer energy (kcal/mole)
OEP-Based Otto-Ladik Model	
EINT CT OEP-OTTO-LADIK KCAL	Total charge-transfer energy (kcal/mole)
EET Coupling Constant	
TrCAMM Model	
EET V0 TRCAMM R1 CM-1	Overlap-uncorrected, converged to R1 (cm-1)
EET V TRCAMM R1 CM-1	Overlap-corrected, converged to R1 (cm-1)
EET V0 TRCAMM R2 CM-1	Overlap-uncorrected, converged to R2 (cm-1)
EET V TRCAMM R2 CM-1	Overlap-corrected, converged to R2 (cm-1)
EET V0 TRCAMM R3 CM-1	Overlap-uncorrected, converged to R3 (cm-1)
EET V TRCAMM R3 CM-1	Overlap-corrected, converged to R3 (cm-1)
EET V0 TRCAMM R4 CM-1	Overlap-uncorrected, converged to R4 (cm-1)

Keyword	Description
EET V TRCAMP R4 CM-1	Overlap-corrected, converged to R4 (cm-1)
EET V0 TRCAMP R5 CM-1	Overlap-uncorrected, converged to R5 (cm-1)
EET V TRCAMP R5 CM-1	Overlap-corrected, converged to R5 (cm-1)
<i>TI/CIS Model</i>	
EET V0 COUL CM-1	Overlap-uncorrected Coulomb (Forster) coupling (cm-1)
EET V0 EXCH CM-1	Overlap-uncorrected exchange (Dexter) coupling (cm-1)
EET V COUL CM-1	Overlap-corrected Coulomb (Forster) coupling (cm-1)
EET V EXCH CM-1	Overlap-corrected exchange (Dexter) coupling (cm-1)
EET V OVRL CM-1	Remaining overlap correction to direct coupling(cm-1)
EET V0 ET1 CM-1	Overlap-uncorrected H_13 matrix element (cm-1)
EET V0 ET2 CM-1	Overlap-uncorrected H_24 matrix element (cm-1)
EET V0 HT1 CM-1	Overlap-uncorrected H_14 matrix element (cm-1)
EET V0 HT2 CM-1	Overlap-uncorrected H_23 matrix element (cm-1)
EET V0 CT CM-1	Overlap-uncorrected H_34 matrix element (cm-1)
EET V ET1 CM-1	Overlap-corrected H_13 matrix element (cm-1)
EET V ET2 CM-1	Overlap-corrected H_24 matrix element (cm-1)
EET V HT1 CM-1	Overlap-corrected H_14 matrix element (cm-1)
EET V HT2 CM-1	Overlap-corrected H_23 matrix element (cm-1)
EET V CT CM-1	Overlap-corrected H_34 matrix element (cm-1)
EET V0 TI-2 CM-1	Approximate 2nd-order indirect coupling (cm-1)
EET V0 TI-3 CM-1	Approximate 3rd-order indirect coupling (cm-1)
EET V TI-2 CM-1	2nd-order indirect coupling (cm-1)
EET V TI-3 CM-1	3rd-order indirect coupling (cm-1)
EET V0 DIRECT CM-1	Approximate direct coupling (cm-1)
EET V0 INDIRECT CM-1	Approximate indirect coupling (cm-1)
EET V DIRECT CM-1	Direct coupling (cm-1)

Keyword	Description
EET V INDIRECT CM-1	Indirect coupling (cm-1)
EET V0 TI-CIS CM-1	Approximate total coupling (cm-1)
EET V TI-CIS CM-1	Total coupling (cm-1)
EET V0 EXCH-M CM-1	Overlap-uncorrected exchange (Dexter) coupling in Mulliken approximation (cm-1)
EET V EXCH-M CM-1	Overlap-corrected exchange (Dexter) coupling in Mulliken approximation (cm-1)
EET V0 CT-M CM-1	Overlap-uncorrected H_34 matrix element in Mulliken approximation (cm-1)
EET V CT-M CM-1	Overlap-corrected H_34 matrix element in Mulliken approximation (cm-1)
<i>OEP-Based TI/CIS Model</i>	
EET V OEP:COUL CM-1	Overlap-corrected Coulomb (Forster) coupling (TrCamm; cm-1)
EET V OEP:EXCH CM-1	Overlap-corrected exchange (Dexter) coupling (Mulliken approximation of AO ERIs; cm-1)
EET V OEP:OVRL CM-1	Remaining overlap correction to direct coupling (cm-1)
EET V0 OEP:ET1 CM-1	Overlap-uncorrected H_13 matrix element (cm-1)
EET V0 OEP:ET2 CM-1	Overlap-uncorrected H_24 matrix element (cm-1)
EET V0 OEP:HT1 CM-1	Overlap-uncorrected H_14 matrix element (cm-1)
EET V0 OEP:HT2 CM-1	Overlap-uncorrected H_23 matrix element (cm-1)
EET V0 OEP:CT:CAMM CM-1	Overlap-uncorrected H_34 matrix element: CAMM approximation of ionic interaction (cm-1)
EET V0 OEP:CT:CC CM-1	Overlap-uncorrected H_34 matrix element: Point-charge approximation of ionic interaction (cm-1)
EET V OEP:ET1 CM-1	Overlap-corrected H_13 matrix element (cm-1)
EET V OEP:ET2 CM-1	Overlap-corrected H_24 matrix element (cm-1)
EET V OEP:HT1 CM-1	Overlap-corrected H_14 matrix element (cm-1)
EET V OEP:HT2 CM-1	Overlap-corrected H_23 matrix element (cm-1)

Keyword	Description
EET V OEP:CT:CAMM CM-1	Overlap-corrected H ₃₄ matrix element: CAMM approximation of ionic interaction (cm-1)
EET V OEP:CT:CC CM-1	Overlap-corrected H ₃₄ matrix element: Point-charge approximation of ionic interaction (cm-1)
EET V OEP:TI-2 CM-1	2nd-order indirect coupling (cm-1)
EET V OEP:TI-3:CAMM CM-1	3rd-order indirect coupling with CAMM approximation for V _{CT} (cm-1)
EET V OEP:TI-3:CC CM-1	3rd-order indirect coupling with point-charge approximation for V _{CT} (cm-1)
EET V OEP:DIRECT CM-1	Direct coupling (cm-1)
EET V OEP:INDIRECT:CAMM CM-1	Indirect coupling with CAMM approximation for V _{CT} (cm-1)
EET V OEP:INDIRECT:CC CM-1	Indirect coupling with point-charge approximation for V _{CT} (cm-1)
EET V OEP:TI-CIS:CAMM CM-1	Total coupling with CAMM approximation for V _{CT} (cm-1)
EET V OEP:TI-CIS:CC CM-1	Total coupling with point-charge approximation for V _{CT} (cm-1)

14.51.2 Constructor & Destructor Documentation

OEPEvSolver()

```
OEPEvSolver::OEPEvSolver (
    SharedWavefunctionUnion wfn_union )
```

Parameters

<i>wfn_union</i>	- wavefunction union of isolated molecular wavefunctions
------------------	--

14.51.3 Member Function Documentation

build()

```
std::shared_ptr< OEPEvSolver > OEPEvSolver::build (
    const std::string & target,
    SharedWavefunctionUnion wfn_union ) [static]
```


Parameters

<i>target</i>	- target property
<i>wfn_union</i>	- wavefunction union of isolated molecular wavefunctions

Implemented target properties:

- `ELECTROSTATIC_ENERGY` - Coulombic interaction energy between unperturbed wavefunctions.
- `REPULSION_ENERGY` - Pauli repulsion interaction energy between unperturbed wavefunctions.

See also

[ElectrostaticEnergySolver](#)

compute_oep_based()

```
double OEPDevSolver::compute_oep_based (
    const std::string & method = "DEFAULT" ) [pure virtual]
```

Each solver object has one `DEFAULT` OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implemented in [oepdev::EETCouplingSolver](#), [oepdev::ChargeTransferEnergySolver](#), [oepdev::RepulsionEnergySolver](#) and [oepdev::ElectrostaticEnergySolver](#).

compute_benchmark()

```
double OEPDevSolver::compute_benchmark (
    const std::string & method = "DEFAULT" ) [pure virtual]
```

Each solver object has one `DEFAULT` benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implemented in [oepdev::EETCouplingSolver](#), [oepdev::ChargeTransferEnergySolver](#), [oepdev::RepulsionEnergySolver](#) and [oepdev::ElectrostaticEnergySolver](#).

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

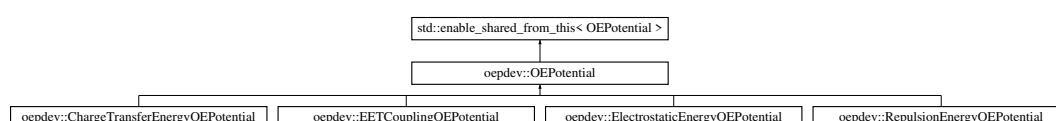
- oepdev/libsolver/[solver.h](#)
- oepdev/libsolver/solver_base.cc

14.52 oepdev::OEPotential Class Reference

Generalized One-Electron Potential: Abstract base.

```
#include <oep.h>
```

Inheritance diagram for oepdev::OEPotential:



Public Member Functions

- [OEPotential](#) (SharedWavefunction [wfn](#), Options &options)
Fully ESP-based OEP object.
- [OEPotential](#) (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)
General OEP object.
- [OEPotential](#) (const [OEPotential](#) *)
Copy constructor.
- virtual std::shared_ptr< [OEPotential](#) > [clone](#) (void) const =0
Make a deep copy of this object.
- virtual [~OEPotential](#) ()
Destructor.
- virtual void [compute](#) (void)
Compute matrix forms of all OEP's within all OEP types.
- virtual void [compute](#) (const std::string &oepType)=0
Compute matrix forms of all OEP's within a specified OEP type.
- virtual void [compute_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared_ptr< psi::Vector > &v)=0
Compute value of potential in point x, y, z and save at v.
- std::shared_ptr< [OEPotential3D](#)< [OEPotential](#) > > [make_oeps3d](#) (const std::string &oepType)
Create 3D vector field with OEP.
- virtual void [write_cube](#) (const std::string &oepType, const std::string &fileName)
Write potential to a cube file.

- virtual void [localize](#) (void)
Localize Occupied MO's.
- virtual std::vector< psi::SharedVector > [mo_centroids](#) (psi::SharedMatrix C)
Compute MO centroids from LCAO-MO matrix.
- virtual void [rotate](#) (psi::SharedMatrix r, psi::SharedMatrix R_prim, psi::SharedMatrix R_aux)
Rotate.
- virtual void [translate](#) (psi::SharedVector t)
Translate.
- virtual void [superimpose](#) (const Matrix &refGeometry, const std::vector< int > &supList, const std::vector< int > &reordList)
Superimpose.
- std::string [name](#) () const
Retrieve name of this OEP.
- [OEType oep](#) (const std::string &oepType) const
Retrieve the potentials.
- SharedMatrix [matrix](#) (const std::string &oepType) const
Retrieve the potentials of a particular OEP type in a matrix form.
- int [n](#) (const std::string &oepType) const
Retrieve the number of a particular OEP type.
- SharedWavefunction [wfn](#) () const
Retrieve wavefunction object.
- SharedMatrix [cOcc](#) () const
Retrieve Canonical occupied MOs.
- SharedMatrix [cVir](#) () const
Retrieve Canonical virtual MOs.
- SharedVector [epsOcc](#) () const
Retrieve Canonical occupied MO energies.
- SharedVector [epsVir](#) () const
Retrieve Canonical virtual MO energies.
- SharedMatrix [lOcc](#) () const
Retrieve Localized occupied MOs.
- SharedMatrix [T](#) () const
Retrieve Canonical to Localized occupied MO transformation matrix.
- SharedLocalizer [localizer](#) () const
Retrieve MO Localizer.
- std::vector< std::shared_ptr< psi::Vector > > [lmoc](#) () const
Retrieve LMO Centroids.
- void [set_name](#) (const std::string &name)
Set the name of this OEP.
- void [set_localized_orbitals](#) (std::shared_ptr< psi::Localizer > [localizer](#))

- Set the localized molecular orbitals in OEP calculation.*

 - void `set_localized_orbitals` (std::shared_ptr< [OEPotential](#) > `oep`)
- Set the localized molecular orbitals in OEP calculation.*

 - void `set_occupied_canonical_orbitals` (std::shared_ptr< [OEPotential](#) > `oep`)
- Set the occupied canonical orbitals in OEP calculations.*

 - virtual void `print_header` () const
- Header information.*

 - void `print` () const
- Print the contents (OEP data)*

 - virtual void `initialize` ()=0
- Initialize the object (expert)*

Static Public Member Functions

- static std::shared_ptr< [OEPotential](#) > `build` (const std::string &category, SharedWavefunction [wfn](#), Options &options)

Build fully ESP-based OEP object.
- static std::shared_ptr< [OEPotential](#) > `build` (const std::string &category, SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)

Build general OEP object.

Public Attributes

- bool `use_localized_orbitals`

Whether to use localized molecular orbitals in OEP calculation; Default: False.
- bool `use_quambo_orbitals`

Whether to use [QUAMBO](#) orbitals to construct VVOs; Default: False.

Protected Member Functions

- virtual void `copy_from` (const [OEPotential](#) *)

Deep-copy the data.
- virtual void `rotate_basic` (psi::SharedMatrix r, psi::SharedMatrix R_prim, psi::SharedMatrix R_aux)

Rotate basic data.
- virtual void `translate_basic` (psi::SharedVector t)

Translate basic data.
- virtual void `rotate_oep` (psi::SharedMatrix r, psi::SharedMatrix R_prim, psi::SharedMatrix R_aux)
- virtual void `translate_oep` (psi::SharedVector t)
- virtual void `compute_molecular_orbitals` ()

Compute MOs (used in initialization stage)

Protected Attributes

- Options [options_](#)
Psi4 options.
- SharedWavefunction [wfn_](#)
Wavefunction.
- SharedBasisSet [primary_](#)
Promary Basis set.
- SharedBasisSet [auxiliary_](#)
Auxiliary Basis set.
- SharedBasisSet [intermediate_](#)
Intermediate Basis set.
- SharedLocalizer [localizer_](#)
Molecular Orbital Localizer.
- std::string [name_](#)
Name of this OEP;.
- std::map< std::string, OEType > [oepTypes_](#)
Types of OEP's within the scope of this object.
- std::shared_ptr< psi::IntegralFactory > [intsFactory_](#)
Integral factory.
- psi::SharedMatrix [potMat_](#)
Matrix of potential one-electron integrals.
- std::shared_ptr< psi::OneBodyAOInt > [OEInt_](#)
One-electron integral shared pointer.
- std::shared_ptr< oepdev::PotentialInt > [potInt_](#)
One-electron potential shared pointer.
- psi::SharedMatrix [cOcc_](#)
Occupied orbitals: Canonical (CMO)
- psi::SharedMatrix [cVir_](#)
Virtual orbitals (Canonical or Valence)
- psi::SharedVector [epsOcc_](#)
Occupied orbital energies: Canonical (CMO)
- psi::SharedVector [epsVir_](#)
Virtual orbital energies (Canonical or Valence)
- psi::SharedMatrix [lOcc_](#)
Occupied orbitals: Localized (LMO)
- psi::SharedMatrix [T_](#)
Canonical to Occupied orbitals transformation.
- std::vector< psi::SharedVector > [lmoc_](#)
LMO Centroids.
- bool [initialized_](#)
Is the object initialized? (MOs computed)

14.52.1 Detailed Description

Manages OEP's in matrix and 3D forms. Available OEP categories:

- ELECTROSTATIC ENERGY
- REPULSION ENERGY
- CHARGE TRANSFER ENERGY
- EET COUPLING CONSTANT

14.52.2 Constructor & Destructor Documentation

OEPotential() [1/2]

```
OEPotential::OEPotential (
    SharedWavefunction wfn,
    Options & options )
```

Parameters

<i>wfn</i>	- wavefunction
<i>options</i>	- Psi4 options

OEPotential() [2/2]

```
OEPotential::OEPotential (
    SharedWavefunction wfn,
    SharedBasisSet auxiliary,
    SharedBasisSet intermediate,
    Options & options )
```

Parameters

<i>wfn</i>	- wavefunction
<i>auxiliary</i>	- auxiliary basis set for density fitting of OEP's
<i>intermediate</i>	- intermediate basis set for density fitting of OEP's
<i>options</i>	- Psi4 options

14.52.3 Member Function Documentation

build() [1/2]

```
std::shared_ptr< OEPotential > OEPotential::build (
    const std::string & category,
    SharedWavefunction wfn,
    Options & options ) [static]
```

Parameters

<i>type</i>	- OEP category
<i>wfn</i>	- wavefunction
<i>options</i>	- Psi4 options

build() [2/2]

```
std::shared_ptr< OEPotential > OEPotential::build (
    const std::string & category,
    SharedWavefunction wfn,
    SharedBasisSet auxiliary,
    SharedBasisSet intermediate,
    Options & options ) [static]
```

Parameters

<i>type</i>	- OEP category
<i>wfn</i>	- wavefunction
<i>auxiliary</i>	- auxiliary basis set for density fitting of OEP's
<i>intermediate</i>	- intermediate basis set for density fitting of OEP's
<i>options</i>	- Psi4 options

make_oeps3d()

```
std::shared_ptr< OEPotential3D< OEPotential > > OEPotential::make_oeps3d (
    const std::string & oepType )
```

Parameters

<i>oepType</i>	- type of OEP. ESP-based OEP is assumed.
----------------	--

Returns

Vector field 3D with the OEP values.

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

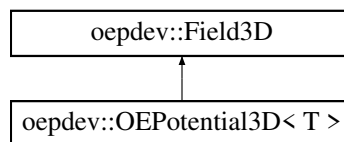
- [oepdev/liboep/oep.h](#)
- [oepdev/liboep/oep_base.cc](#)

14.53 oepdev::OEPotential3D< T > Class Template Reference

Class template for OEP 3D fields.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::OEPotential3D< T >:



Public Member Functions

- [OEPotential3D](#) (const int &ndim, const int &np, const double &padding, std::shared_ptr< T > oep, const std::string &oepType)
Construct random spherical collection of 3D field of type T.
- [OEPotential3D](#) (const int &ndim, const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, std::shared_ptr< T > oep, const std::string &oepType, psi::Options &options)
Construct ordered 3D collection of 3D field of type T.
- virtual [~OEPotential3D](#) ()
Destructor.
- virtual std::shared_ptr< psi::Vector > [compute_xyz](#) (const double &x, const double &y, const double &z)
Compute a value of 3D field at point (x, y, z)
- virtual void [print](#) () const
Print information of the object to Psi4 output.

Protected Attributes

- std::shared_ptr< T > [oep_](#)
Shared pointer to the instance of class T
- std::string [oepType_](#)
Descriptor of the 3D field type stored in instance of T

Additional Inherited Members

14.53.1 Detailed Description

template<class T>
class oepdev::OEPotential3D< T >

Used for special type of classes T that contain following public member functions:

```
class T : public std::enable_shared_from_this<T> {
public:
    void compute_3D(const std::string& descriptor,
                   const double& x, const double& y, const double& z,
                   std::shared_ptr<psi::Vector> &v);

    shared_ptr<psi::Wavefunction> wfn() const {return wfn-;}
};
```

with the `descriptor` of a certain 3D field type, `x`, `y`, `z` the points in 3D space in which the scalar or vector field has to be computed and stored at `v`. Instances of `T` should store shared pointer to wavefunction object. List of classes `T` that are compatible with this class template and are currently implemented in `oepdev` is given below:

- `oepdev::OEPotential` abstract base (do not use derived classes as `T`)

Template parameters:

Template Parameters

<code>T</code>	the compatible class (e.g. <code>oepdev::OEPotential</code>)
----------------	---

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

- `oepdev/lib3d/space3d.h`

14.54 oepdev::OEType Struct Reference

Container to handle the type of One-Electron Potentials.

```
#include <oep.h>
```

Public Member Functions

- `OEType` ()=default
Initializer.
- `OEType` (std::string, bool, int, SharedMatrix, `SharedDMTPole`, SharedCISData)

Initializer from list.

- [OEPTYPE](#) (const [OEPTYPE](#) *)

Copy constructor.

Public Attributes

- std::string [name](#)
Name of this type of OEP.
- bool [is_density_fitted](#)
Is this OEP DF-based?
- int [n](#)
Number of OEP's within a type.
- SharedMatrix [matrix](#)
All OEP's of this type gathered in a matrix form.
- [SharedDMTPole](#) [dmtp](#)
Distributed Multipole Object.
- SharedCISData [cis_data](#)
CIS data.

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

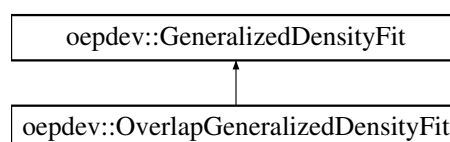
- oepdev/liboep/[oep.h](#)
- oepdev/liboep/oep_base.cc

14.55 oepdev::OverlapGeneralizedDensityFit Class Reference

Generalized Density Fitting Scheme - Single Fit Based on Minimal Overlap in MO Basis.

```
#include <oep_gdf.h>
```

Inheritance diagram for oepdev::OverlapGeneralizedDensityFit:



Public Member Functions

- **OverlapGeneralizedDensityFit** (std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::BasisSet > bs_intermediate, std::shared_ptr< psi::Matrix > v_vector)
- std::shared_ptr< psi::Matrix > [compute](#) (void)
Perform the generalized density fit.

Additional Inherited Members

14.55.1 Detailed Description

The density fitting map projects the OEP onto an arbitrary (not necessarily complete) auxiliary basis set space through application of the basis projection technique. Refer to [density fitting specialized for OEP's](#) for more details.

14.55.2 Determination of the OEP matrix

Coefficients \mathbf{G} are computed by using the following relation

$$\mathbf{G} = \mathbf{T}_{m\tilde{B}} \cdot \mathbf{S}_{\tilde{B}\tilde{B}}^{-1} \cdot \mathbf{T}_{m\tilde{B}}^{\dagger} \cdot \mathbf{S}_{mi} \cdot \mathbf{G}_i$$

where the intermediate projection matrix is given by

$$\mathbf{G}_i = \mathbf{S}_{ii}^{-1} \cdot \mathbf{V}_i$$

See [density fitting of OEPs](#) for more details regarding the matrices involved in the above expressions.

14.55.3 Member Function Documentation

compute()

```
std::shared_ptr< psi::Matrix > OverlapGeneralizedDensityFit::compute (
    void ) [virtual]
```

Returns

The OEP coefficients G_{ξ_i}

Implements [oepdev::GeneralizedDensityFit](#).

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

- oepdev/liboep/oep_gdf.h
- oepdev/liboep/oep_gdf.cc

14.56 oepdev::PerturbCharges Struct Reference

Structure to hold perturbing charges.

```
#include <scf_perturb.h>
```

Public Attributes

- `std::vector< double > charges`
Vector of charge values.
- `std::vector< std::shared_ptr< psi::Vector > > positions`
Vector of charge position vectors.

14.56.1 Detailed Description

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

- `oepdev/libutil/scf_perturb.h`

14.57 `oepdev::Points3DIterator::Point` Struct Reference

Public Attributes

- `double x`
- `double y`
- `double z`
- `int index`

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

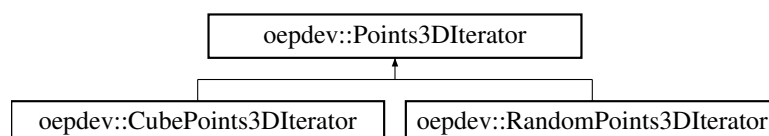
- `oepdev/lib3d/space3d.h`

14.58 `oepdev::Points3DIterator` Class Reference

Iterator over a collection of points in 3D space. Abstract base.

```
#include <space3d.h>
```

Inheritance diagram for `oepdev::Points3DIterator`:



Classes

- struct [Point](#)

Public Member Functions

- [Points3DIterator](#) (const int &np)
Plain constructor. Initializes the abstract features.
 - virtual [~Points3DIterator](#) ()
Destructor.
 - virtual bool [is_done](#) ()
Check if iteration is finished.
 - virtual void [first](#) ()=0
Initialize first iteration.
 - virtual void [next](#) ()=0
Step to next iteration.
 - virtual void [rewind](#) ()
Rewind to the beginning.
-
- virtual double **x** () const
 - virtual double **y** () const
 - virtual double **z** () const
 - virtual int **index** () const

Static Public Member Functions

- static shared_ptr< [Points3DIterator](#) > [build](#) (const int &nx, const int &ny, const int &nz, const double &dx, const double &dy, const double &dz, const double &ox, const double &oy, const double &oz)
Build G09 Cube collection iterator.
- static shared_ptr< [Points3DIterator](#) > [build](#) (const int &np, const double &radius, const double &cx, const double &cy, const double &cz)
Build random collection iterator.
- static shared_ptr< [Points3DIterator](#) > [build](#) (const int &np, const double &pad, psi::SharedMolecule mol)
Build random collection iterator.

Protected Attributes

- const int [np_](#)
Number of points.
- bool [done_](#)
Status of the iterator.
- int [index_](#)

Current index.

- [Point](#) **current_**

14.58.1 Detailed Description

Points3DIterators are constructed either as iterators over:

- a random collections or
- an ordered (g09 cube-like) collections. **Note:** Always create instances by using static factory methods.

14.58.2 Constructor & Destructor Documentation

Points3DIterator()

```
oepdev::Points3DIterator::Points3DIterator (
    const int & np )
```

Parameters

<i>np</i>	- number of points this iterator is constructed for
-----------	---

14.58.3 Member Function Documentation

build() [1/3]

```
std::shared_ptr< Points3DIterator > oepdev::Points3DIterator::build (
    const int & nx,
    const int & ny,
    const int & nz,
    const double & dx,
    const double & dy,
    const double & dz,
    const double & ox,
    const double & oy,
    const double & oz ) [static]
```

The points are generated according to Gaussian cube file format.

Parameters

<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>dx</i>	- spacing distance along x direction
<i>dy</i>	- spacing distance along y direction
<i>dz</i>	- spacing distance along y direction
<i>ox</i>	- coordinate x of cube origin
<i>oy</i>	- coordinate y of cube origin
<i>oz</i>	- coordinate z of cube origin

build() [2/3]

```
std::shared_ptr< Points3DIterator > oepdev::Points3DIterator::build (
    const int & np,
    const double & radius,
    const double & cx,
    const double & cy,
    const double & cz ) [static]
```

The points are drawn according to uniform distrinution in 3D space.

Parameters

<i>np</i>	- number of points to draw
<i>radius</i>	- sphere radius inside which points are to be drawn
<i>cx</i>	- coordinate x of sphere's centre
<i>cy</i>	- coordinate y of sphere's centre
<i>cz</i>	- coordinate z of sphere's centre

build() [3/3]

```
shared_ptr< Points3DIterator > oepdev::Points3DIterator::build (
    const int & np,
    const double & pad,
    psi::SharedMolecule mol ) [static]
```

The points are drawn according to uniform distrinution in 3D space enclosing a molecule given. All drawn points lie outside the van der Waals volume.

Parameters

<i>np</i>	- number of points to draw
<i>pad</i>	- radius padding of a minimal sphere enclosing the molecule
<i>mol</i>	- Psi4 molecule object

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

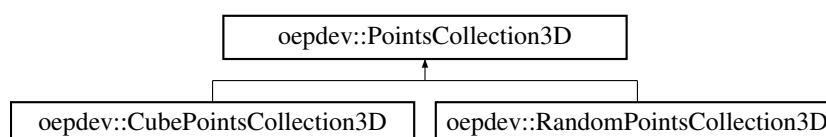
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.59 oepdev::PointsCollection3D Class Reference

Collection of points in 3D space. Abstract base.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::PointsCollection3D:



Public Types

- enum [Collection](#) { **Random**, **Cube** }
- Public descriptor of collection type.*

Public Member Functions

- [PointsCollection3D](#) ([Collection](#) collectionType, int &np)
Initialize abstract features.
- **PointsCollection3D** ([Collection](#) collectionType, const int &np)
- virtual [~PointsCollection3D](#) ()
Destructor.
- virtual int [npoints](#) () const
Get the number of points.
- virtual shared_ptr< [Points3DIterator](#) > [points_iterator](#) () const
Get the iterator over this collection of points.
- virtual [Collection](#) [get_type](#) () const
Get the collection type.
- virtual void [print](#) () const =0
Print the information to Psi4 output file.

Static Public Member Functions

- static shared_ptr< [PointsCollection3D](#) > [build](#) (const int &npoints, const double &radius, const double &cx=0.0, const double &cy=0.0, const double &cz=0.0)

Build random collection of points.

- static shared_ptr< [PointsCollection3D](#) > [build](#) (const int &npoints, const double &padding, psi::SharedMolecule mol)

Build random collection of points.

- static shared_ptr< [PointsCollection3D](#) > [build](#) (const int &nx, const int &ny, const int &nz, const double &px, const double &py, const double &pz, psi::SharedBasisSet bs, psi::Options &options)

Build G09 Cube collection of points.

Protected Attributes

- const int [np_](#)
Number of points.
- [Collection](#) [collectionType_](#)
Collection type.
- shared_ptr< [Points3DIterator](#) > [pointsIterator_](#)
iterator over points collection

14.59.1 Detailed Description

Create random or ordered (g09 cube-like) collections of points in 3D space.

Note: Always create instances by using static factory methods.

14.59.2 Constructor & Destructor Documentation

PointsCollection3D()

```
oepdev::PointsCollection3D::PointsCollection3D (
    Collection collectionType,
    int & np )
```

Parameters

<i>np</i>	- number of points to be created
-----------	----------------------------------

14.59.3 Member Function Documentation

build() [1/3]

```
std::shared_ptr< PointsCollection3D > oepdev::PointsCollection3D::build (
    const int & npoints,
    const double & radius,
    const double & cx = 0.0,
    const double & cy = 0.0,
    const double & cz = 0.0 ) [static]
```

Points uniformly span a sphere.

Parameters

<i>npoints</i>	- number of points to draw
<i>radius</i>	- sphere radius inside which points are to be drawn
<i>cx</i>	- coordinate x of sphere's centre
<i>cy</i>	- coordinate y of sphere's centre
<i>cz</i>	- coordinate z of sphere's centre

build() [2/3]

```
std::shared_ptr< PointsCollection3D > oepdev::PointsCollection3D::build (
    const int & npoints,
    const double & padding,
    psi::SharedMolecule mol ) [static]
```

Points uniformly span space inside a sphere enclosing a molecule. exluding the van der Waals volume.

Parameters

<i>np</i>	- number of points to draw
<i>padding</i>	- radius padding of a minimal sphere enclosing the molecule
<i>mol</i>	- Psi4 molecule object

build() [3/3]

```
std::shared_ptr< PointsCollection3D > oepdev::PointsCollection3D::build (
    const int & nx,
```

```

const int & ny,
const int & nz,
const double & px,
const double & py,
const double & pz,
psi::SharedBasisSet bs,
psi::Options & options ) [static]

```

The points span a parallelepiped according to Gaussian cube file format.

Parameters

<i>nx</i>	- number of points along x direction
<i>ny</i>	- number of points along y direction
<i>nz</i>	- number of points along z direction
<i>px</i>	- padding distance along x direction
<i>py</i>	- padding distance along y direction
<i>pz</i>	- padding distance along z direction
<i>bs</i>	- Psi4 basis set object
<i>options</i>	- Psi4 options object

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

- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.60 oepdev::PolarGEFactory Class Reference

Polarization GEFP Factory. Abstract Base.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::PolarGEFactory:



Public Member Functions

- [PolarGEFactory](#) (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
Construct from Psi4 options.
- virtual [~PolarGEFactory](#) ()
Destruct.
- virtual std::shared_ptr< [GenEffPar](#) > [compute](#) (void)=0
Compute the density matrix susceptibility tensors.

Protected Member Functions

- `std::shared_ptr< psi::Vector > draw_field ()`
Randomly draw electric field value.
- `double draw_charge ()`
Randomly draw charge value.
- `std::shared_ptr< oepdev::RHFPerturbed > perturbed_state (const std::shared_ptr< psi::Vector > &field)`
Solve SCF equations to find perturbed state due to uniform electric field.
- `std::shared_ptr< oepdev::RHFPerturbed > perturbed_state (const std::shared_ptr< psi::Vector > &pos, const double &charge)`
Solve SCF equations to find perturbed state due to point charge.
- `std::shared_ptr< oepdev::RHFPerturbed > perturbed_state (const std::shared_ptr< psi::Matrix > &charges)`
Solve SCF equations to find perturbed state due to set of point charges.
- `std::shared_ptr< psi::Vector > field_due_to_charges (const std::shared_ptr< psi::Matrix > &charges, const double &x, const double &y, const double &z)`
Evaluate electric field at point (x,y,z) due to point charges.
- `std::shared_ptr< psi::Vector > field_due_to_charges (const std::shared_ptr< psi::Matrix > &charges, const std::shared_ptr< psi::Vector > &pos)`
- `std::shared_ptr< psi::Matrix > field_gradient_due_to_charges (const std::shared_ptr< psi::Matrix > &charges, const double &x, const double &y, const double &z)`
Evaluate electric field gradient at point (x,y,z) due to point charges.
- `std::shared_ptr< psi::Matrix > field_gradient_due_to_charges (const std::shared_ptr< psi::Matrix > &charges, const std::shared_ptr< psi::Vector > &pos)`

Additional Inherited Members

14.60.1 Detailed Description

Basic interface for the polarization density matrix susceptibility parameters.

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

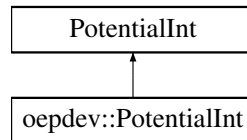
- `oepdev/libgefp/gefp.h`
- `oepdev/libgefp/gefp_polar_base.cc`

14.61 oepdev::PotentialInt Class Reference

Computes potential integrals.

```
#include <potential.h>
```

Inheritance diagram for `oepdev::PotentialInt`:



Public Member Functions

- [PotentialInt](#) (std::vector< psi::SphericalTransform > &st, std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, int deriv=0)

Constructor. Initialize identically like in psi::Potentillnt.

- [PotentialInt](#) (std::vector< psi::SphericalTransform > &st, std::shared_ptr< psi::BasisSet > bs1, std::shared_ptr< psi::BasisSet > bs2, std::shared_ptr< psi::Matrix > Qxyz, int deriv=0)

Constructor. Takes an arbitrary collection of charges.

- [PotentialInt](#) (std::vector< psi::SphericalTransform > &, std::shared_ptr< psi::BasisSet >, std::shared_ptr< psi::BasisSet >, const double &x, const double &y, const double &z, const double &q=1.0, int deriv=0)

Constructor. Computes potential for one point x, y, z for a test particle of charge q.

- void [set_charge_field](#) (const double &x, const double &y, const double &z, const double &q=1.0)

Mutator. Set the charge field to be a x, y, z point of charge q.

14.61.1 Constructor & Destructor Documentation

PotentialInt() [1/3]

```

oepdev::PotentialInt::PotentialInt (
    std::vector< psi::SphericalTransform > & st,
    std::shared_ptr< psi::BasisSet > bs1,
    std::shared_ptr< psi::BasisSet > bs2,
    int deriv = 0 )
  
```

Parameters

<i>st</i>	- Spherical transform object
<i>bs1</i>	- basis set for first space
<i>bs2</i>	- basis set for second space
<i>deriv</i>	- derivative level

PotentialInt() [2/3]

```
oepdev::PotentialInt::PotentialInt (
    std::vector< psi::SphericalTransform > & st,
    std::shared_ptr< psi::BasisSet > bs1,
    std::shared_ptr< psi::BasisSet > bs2,
    std::shared_ptr< psi::Matrix > Qxyz,
    int deriv = 0 )
```

Parameters

<i>st</i>	- Spherical transform object
<i>bs1</i>	- basis set for first space
<i>bs2</i>	- basis set for second space
<i>Qxyz</i>	- matrix with charges and their positions
<i>deriv</i>	- derivative level

PotentialInt() [3/3]

```
oepdev::PotentialInt::PotentialInt (
    std::vector< psi::SphericalTransform > & st,
    std::shared_ptr< psi::BasisSet > bs1,
    std::shared_ptr< psi::BasisSet > bs2,
    const double & x,
    const double & y,
    const double & z,
    const double & q = 1.0,
    int deriv = 0 )
```

Parameters

<i>st</i>	- Spherical transform object
<i>bs1</i>	- basis set for first space
<i>bs2</i>	- basis set for second space
<i>x</i>	- x coordinate of q
<i>y</i>	- y coordinate of q
<i>z</i>	- z coordinate of q
<i>q</i>	- value of the probe charge
<i>deriv</i>	- derivative level

14.61.2 Member Function Documentation

set_charge_field()

```
void oepdev::PotentialInt::set_charge_field (
    const double & x,
    const double & y,
    const double & z,
    const double & q = 1.0 )
```

Parameters

<i>x</i>	- x coordinate of q
<i>y</i>	- y coordinate of q
<i>z</i>	- z coordinate of q
<i>q</i>	- value of the probe charge

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

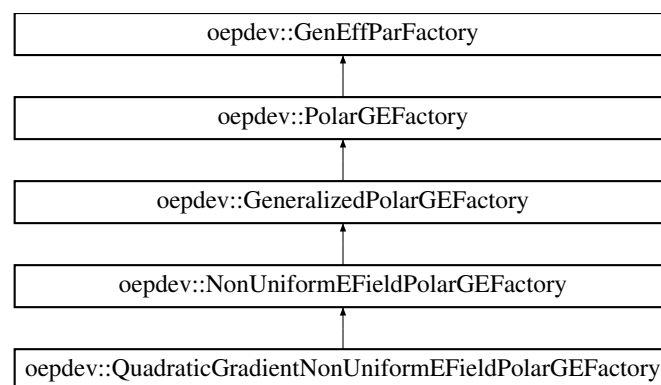
- oepdev/libpsi/[potential.h](#)
- oepdev/libpsi/potential.cc

14.62 oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory:

**Public Member Functions**

- **QuadraticGradientNonUniformEFieldPolarGEFactory** (`std::shared_ptr< psi::Wavefunction > wfn`, `psi::Options &opt`)
- void [compute_gradient](#) (`int i`, `int j`)

Compute Gradient vector associated with the i -th and j -th basis set function.

- void `compute_hessian` (void)

Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.62.1 Detailed Description

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(20)} : \mathbf{F} \otimes \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(01)} : \nabla_i \otimes \mathbf{F}(\mathbf{r}_i) \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$ is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(20)}$ is the density matrix dipole-dipole hyperpolarizability
- $\mathbf{B}_{i;\alpha\beta}^{(01)}$ is the density matrix quadrupole polarizability all defined for the distributed site at \mathbf{r}_i .

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

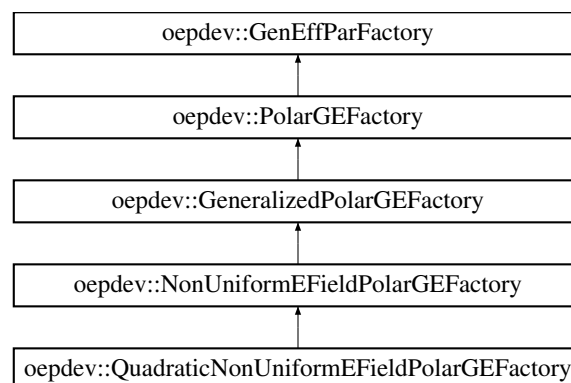
- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp_polar_nonuniform_field_2_grad_1.cc

14.63 oepdev::QuadraticNonUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::QuadraticNonUniformEFieldPolarGEFactory:



Public Member Functions

- **QuadraticNonUniformEFieldPolarGEFactory** (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute_gradient](#) (int i, int j)
Compute Gradient vector associated with the i-th and j-th basis set function.
- void [compute_hessian](#) (void)
Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.63.1 Detailed Description

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \sum_i \left\{ \mathbf{B}_{i;\alpha\beta}^{(10)} \cdot \mathbf{F}(\mathbf{r}_i) + \mathbf{B}_{i;\alpha\beta}^{(20)} : \mathbf{F}(\mathbf{r}_i) \otimes \mathbf{F}(\mathbf{r}_i) \right\}$$

where:

- $\mathbf{B}_{i;\alpha\beta}^{(10)}$ is the density matrix dipole polarizability
- $\mathbf{B}_{i;\alpha\beta}^{(20)}$ is the density matrix dipole-dipole hyperpolarizability all defined for the distributed site at \mathbf{r}_i .

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

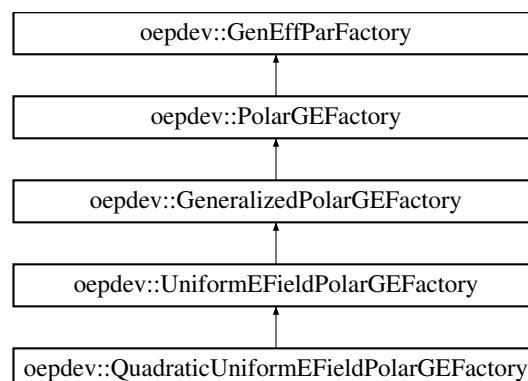
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp_polar_nonuniform_field_2.cc

14.64 oepdev::QuadraticUniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::QuadraticUniformEFieldPolarGEFactory:



Public Member Functions

- **QuadraticUniformEFieldPolarGEFactory** (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute_gradient](#) (int i, int j)
Compute Gradient vector associated with the i-th and j-th basis set function.
- void [compute_hessian](#) (void)
Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.64.1 Detailed Description

Implements the generalized density matrix susceptibility model of the form

$$\delta D_{\alpha\beta} \approx \mathbf{B}_{\alpha\beta}^{(10)} \cdot \mathbf{F} + \mathbf{B}_{\alpha\beta}^{(20)} : \mathbf{F} \otimes \mathbf{F}$$

where:

- $\mathbf{B}_{\alpha\beta}^{(10)}$ is the density matrix dipole polarizability
- $\mathbf{B}_{\alpha\beta}^{(20)}$ is the density matrix dipole-dipole hyperpolarizability

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

- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp_polar_uniform_field_2.cc

14.65 oepdev::QUAMBO Class Reference

The Quasiatomic Minimal Basis Set Molecular Orbitals ([QUAMBO](#))

```
#include <quambo.h>
```

Public Member Functions

- [QUAMBO](#) (psi::SharedWavefunction wfn, bool [acbs](#)=true)
Constructor.
- virtual [~QUAMBO](#) ()
Destructor.
- void [compute](#) (void)
Compute QUAMBOs and VVOs.
- psi::SharedMatrix [quambo](#) (const std::string &spin, const std::string &type="ORTHOGONAL")

- Get the QUAMBOs in AO representation (AOs: rows, QUAMBOs: columns)*

 - psi::SharedVector [epsilon_a_subset](#) (const std::string &space, const std::string &subset)

Get SCF alpha orbital energies in minimal MO basis.

 - psi::SharedVector [epsilon_b_subset](#) (const std::string &space, const std::string &subset)

Get SCF beta orbital energies in minimal MO basis.

 - psi::SharedMatrix [Ca_subset](#) (const std::string &space, const std::string &subset)

Get SCF alpha orbitals in minimal MO basis.

 - psi::SharedMatrix [Cb_subset](#) (const std::string &space, const std::string &subset)

Get SCF beta orbitals in minimal MO basis.

 - int [nbas](#) () const

Size of QUAMBO basis.

 - int [naocc](#) () const

Number of Alpha occupied MOs in minimal basis (same as in original basis)

 - int [nbocc](#) () const

Number of Beta occupied MOs in minimal basis (same as in original basis)

 - int [navir](#) () const

Number of Alpha virtual MOs in minimal basis (number of Alpha VVOs)

 - int [nbvir](#) () const

Number of Beta virtual MOs in minimal basis (number of Beta VVOs)

Static Public Member Functions

- static std::shared_ptr< [QUAMBO](#) > [build](#) (psi::SharedWavefunction wfn, bool [acbs](#)=true)
- Static factory method.*

Public Attributes

- const bool [acbs](#)
- Is ACBS mode selected?*

Protected Member Functions

- double [compute_error_between_two_vectors_](#) (psi::SharedVector a, psi::SharedVector b)
- int [calculate_nbas_mini_](#) (void)
- std::vector< psi::SharedMolecule > [atomize_](#) (void)
- SharedQUAMBOData [compute_quambo_data_](#) (psi::SharedMatrix, psi::SharedMatrix, psi::SharedVector, psi::SharedVector, psi::SharedMatrix, psi::SharedMatrix, psi::SharedMatrix, int, int, std::string)
- psi::SharedVector [epsilon_subset_helper_](#) (psi::SharedVector C_full, const std::string &label, const int &n, const std::string &space, const std::string &subset)
- psi::SharedMatrix [C_subset_helper_](#) (psi::SharedMatrix C_full, const std::string &label, const int &n, const std::string &space, const std::string &subset)

Protected Attributes

- psi::Options & [options_](#)
Psi4 options.
- psi::SharedMolecule [mol_](#)
Molecule.
- psi::SharedWavefunction [wfn_](#)
Wavefunction.
- std::map< std::string, int > [nbas_atom_mini_](#)
numbers of minimal basis functions of free atoms
- std::map< std::string, int > [unpe_atom_](#)
numbers of unpaired electrons in free atoms
- psi::SharedMatrix [Sao_](#)
AO Overlap Matrix.
- psi::SharedMatrix [quambo_a_nonorthogonal_](#)
QUAMBO (Alpha, non-orthogonal)
- psi::SharedMatrix [quambo_a_orthogonal_](#)
QUAMBO (Alpha, orthogonal)
- psi::SharedMatrix [quambo_b_nonorthogonal_](#)
QUAMBO (Beta, non-orthogonal)
- psi::SharedMatrix [quambo_b_orthogonal_](#)
QUAMBO (Beta, orthogonal)
- psi::SharedMatrix [c_a_mini_vir_](#)
Virtual Valence Molecular Orbitals (Alpha, VVO)
- psi::SharedMatrix [c_b_mini_vir_](#)
Virtual Valence Molecular Orbitals (Beta, VVO)
- psi::SharedVector [e_a_mini_vir_](#)
VVO Energies (Alpha)
- psi::SharedVector [e_b_mini_vir_](#)
VVO Energies (Beta)
- psi::SharedMatrix [c_a_mini_](#)
All Molecular orbitals (Alpha, OCC + VVO)
- psi::SharedMatrix [c_b_mini_](#)
All Molecular orbitals (Beta, OCC + VVO)
- psi::SharedVector [e_a_mini_](#)
Energies of All Molecular Orbitals (Alpha)
- psi::SharedVector [e_b_mini_](#)
Energies of All Molecular Orbitals (Beta)
- int [nbas_mini_](#)
Size of QUAMBO basis per orbital group (Alpha, Beta)
- int [naocc_mini_](#)

- *Number of Alpha occupied MOs.*
- int [nbocc_mini_](#)
- *Number of Beta occupied MOs.*
- int [navir_mini_](#)
- *Number of Alpha virtual MOs.*
- int [nbvir_mini_](#)
- *Number of Beta virtual MOs.*
- int [nbf_](#)
- *Number of AO basis functions.*

14.65.1 Detailed Description

References:

- [1] W. C. Lu, C. Z. Wang, M.W. Schmidt, L. Bytautas, K. M. Ho, K. Reudenberg, *J. Chem. Phys.* **120**, 2629 (2004). [original [QUAMBO](#) paper]
- [2] P. Xu, M. S. Gordon, *J. Chem. Phys.* **139**, 194104 (2013). [application of [QUAMBO](#) in EFP2 CT term]

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

- oepdev/libutil/[quambo.h](#)
- oepdev/libutil/quambo.cc

14.66 oepdev::QUAMBOData Struct Reference

Container to store the [QUAMBO](#) data.

```
#include <quambo.h>
```

Public Attributes

- psi::SharedMatrix [quambo_nonorthogonal](#)
QUAMBO (non-orthogonal)
- psi::SharedMatrix [quambo_orthogonal](#)
QUAMBO (orthogonal)
- psi::SharedMatrix [c_mini_vir](#)
Virtual Valence Molecular Orbitals (VVO)
- psi::SharedVector [e_mini_vir](#)
VVO Energies.
- psi::SharedMatrix [c_mini](#)
All Molecular orbitals (OCC + VVO)
- psi::SharedVector [e_mini](#)
Energies of All Molecular Orbitals.

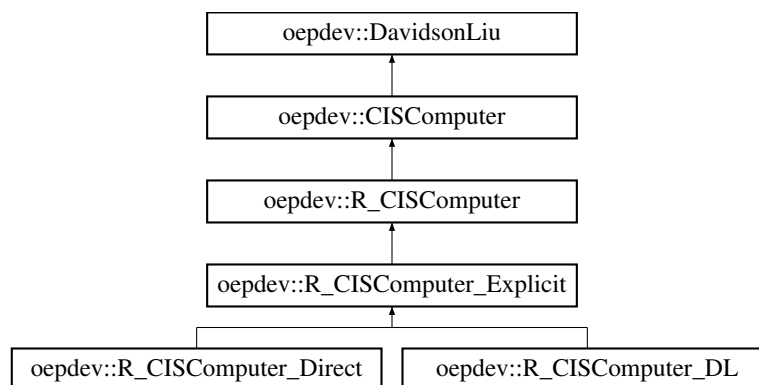
14.66.1 Detailed Description

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

- oepdev/libutil/[quambo.h](#)

14.67 oepdev::R_CISComputer Class Reference

Inheritance diagram for oepdev::R_CISComputer:



Public Member Functions

- **R_CISComputer** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)

Protected Member Functions

- virtual void **print_excited_state_character_** (int l)

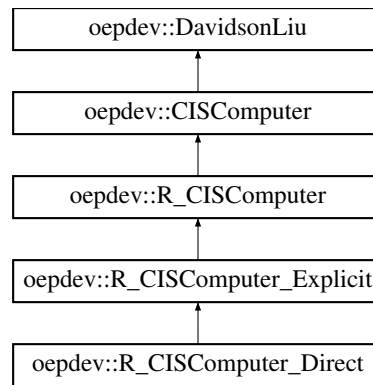
Additional Inherited Members

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

- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis_rhf.cc

14.68 oepdev::R_CISComputer_Direct Class Reference

Inheritance diagram for oepdev::R_CISComputer_Direct:



Public Member Functions

- **R_CISComputer_Direct** (`std::shared_ptr< psi::Wavefunction > wfn`, `psi::Options &opt`)

Protected Member Functions

- virtual void **build_hamiltonian_** (void)
- virtual void **transform_integrals_** (void)

Additional Inherited Members

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

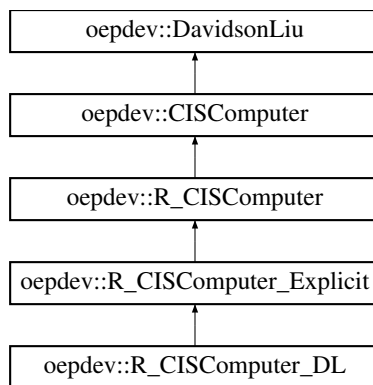
- `oepdev/libutil/cis.h`
- `oepdev/libutil/cis_rhf_direct.cc`

14.69 oepdev::R_CISComputer_DL Class Reference

CIS Computer with RHF reference: Davidson-Liu Solver.

```
#include <cis.h>
```

Inheritance diagram for `oepdev::R_CISComputer_DL`:



Public Member Functions

- **R_CISComputer_DL** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)

Protected Member Functions

- virtual void **set_nstates_** (void)
- virtual void **transform_integrals_** (void)
- virtual void **allocate_hamiltonian_** (void)
- virtual void **build_hamiltonian_** (void)
- virtual void **diagonalize_hamiltonian_** (void)
- virtual void **davidson_liu_compute_diagonal_hamiltonian** (void)
- virtual void **davidson_liu_compute_sigma** (void)

Additional Inherited Members

14.69.1 Detailed Description

Associated options:

- **CIS_TYPE** - must be set to **DAVIDSON_LIU** (Default).
- **CIS_SCHWARTZ_CUTOFF** - Cutoff for Schwartz ERI screening. Default: 0.0.

Implementation

Diagonal Hamiltonian elements

They are computed by using direct method with Schwartz screening of AO ERI's. The implementation formula is

$$H_{ii}^{aa} = \epsilon_a - \epsilon_i + \sum_{\alpha\beta\gamma\delta} (\alpha\beta|\gamma\delta) C_{\alpha i} C_{\delta a} (C_{\beta a} C_{\gamma i} - C_{\beta i} C_{\gamma a})$$

The block associated with beta spin is equal to alpha block.

Sigma vectors

The sigma vectors are computed from

$$\begin{aligned}\sigma_i^{a,k} &= (\varepsilon_a - \varepsilon_i) b_i^{a,k} + J_i^a(\mathbf{T}^{(k)}) + J_i^a(\overline{\mathbf{T}}^{(k)}) - K_i^a(\mathbf{T}^{(k)}) \\ \sigma_i^{\bar{a},k} &= (\varepsilon_a - \varepsilon_i) b_i^{\bar{a},k} + J_i^a(\mathbf{T}^{(k)}) + J_i^a(\overline{\mathbf{T}}^{(k)}) - K_i^a(\overline{\mathbf{T}}^{(k)})\end{aligned}$$

where k labels the vectors and where the generalized one-particle density matrices are defined by

$$\begin{aligned}T_{\gamma\delta}^{(k)} &= \sum_{jb} C_{\delta b} b_j^{b,k} C_{\gamma j} \\ \bar{T}_{\gamma\delta}^{(k)} &= \sum_{\bar{j}\bar{b}} C_{\delta\bar{b}} \bar{b}_{\bar{j}}^{\bar{b},k} C_{\gamma\bar{j}}\end{aligned}$$

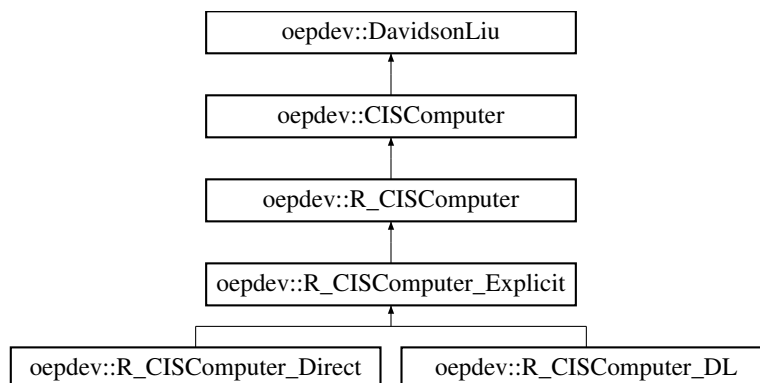
The **J** and **K** matrices in AO basis are computed by using the `psi::JK` object, and subsequently transformed to CMO's.

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

- oepdev/libutil/cis.h
- oepdev/libutil/cis_rhf_dl.cc

14.70 oepdev::R_CISComputer_Explicit Class Reference

Inheritance diagram for oepdev::R_CISComputer_Explicit:



Public Member Functions

- **R_CISComputer_Explicit** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)

Protected Member Functions

- virtual void **set_beta_** (void)
- virtual void **build_hamiltonian_** (void)

Additional Inherited Members

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

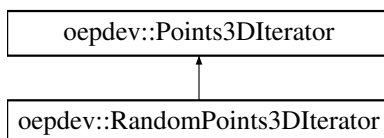
- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis_rhf_explicit.cc

14.71 oepdev::RandomPoints3DIterator Class Reference

Iterator over a collection of points in 3D space. Random collection.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::RandomPoints3DIterator:



Public Member Functions

- **RandomPoints3DIterator** (const int &np, const double &radius, const double &cx, const double &cy, const double &cz)
- **RandomPoints3DIterator** (const int &np, const double &pad, psi::SharedMolecule mol)
- virtual void [first](#) ()
Initialize first iteration.
- virtual void [next](#) ()
Step to next iteration.

Protected Member Functions

- virtual double **random_double** ()
- virtual void **draw_random_point** ()
- virtual bool **is_in_vdWsphere** (double x, double y, double z) const

Protected Attributes

- double **cx_**
- double **cy_**
- double **cz_**
- double **radius_**
- double **r_**

- double **phi**_
- double **theta**_
- double **x**_
- double **y**_
- double **z**_
- psi::SharedMatrix **excludeSpheres**_
- std::map< std::string, double > **vdwRadius**_
- std::default_random_engine **randomNumberGenerator**_
- std::uniform_real_distribution< double > **randomDistribution**_

Additional Inherited Members

14.71.1 Detailed Description

Note: Always create instances by using static factory method from [Points3DIterator](#). Do not use constructors of this class.

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

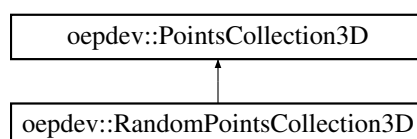
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.72 oepdev::RandomPointsCollection3D Class Reference

Collection of random points in 3D space.

```
#include <space3d.h>
```

Inheritance diagram for oepdev::RandomPointsCollection3D:



Public Member Functions

- **RandomPointsCollection3D** ([Collection](#) collectionType, const int &[npoints](#), const double &radius, const double &cx, const double &cy, const double &cz)
- **RandomPointsCollection3D** ([Collection](#) collectionType, const int &[npoints](#), const double &padding, psi::SharedMolecule mol)
- virtual void [print](#) () const

Print the information to Psi4 output file.

Additional Inherited Members

14.72.1 Detailed Description

Note: Do not use constructors of this class explicitly. Instead, use static factory methods of the superclass to create instances.

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

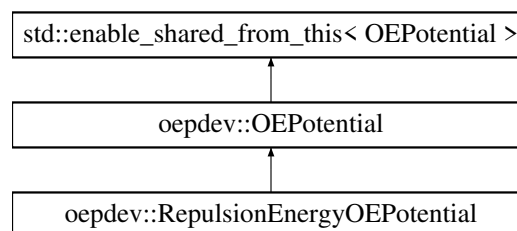
- oepdev/lib3d/space3d.h
- oepdev/lib3d/space3d.cc

14.73 oepdev::RepulsionEnergyOEPotential Class Reference

Generalized One-Electron Potential for Pauli Repulsion Energy.

```
#include <oep.h>
```

Inheritance diagram for oepdev::RepulsionEnergyOEPotential:



Public Member Functions

- **RepulsionEnergyOEPotential** (SharedWavefunction [wfn](#), SharedBasisSet auxiliary, SharedBasisSet intermediate, Options &options)
- **RepulsionEnergyOEPotential** (SharedWavefunction [wfn](#), Options &options)
- **RepulsionEnergyOEPotential** (const [RepulsionEnergyOEPotential](#) *f)
- virtual void [compute](#) (const std::string &oepType) override
Compute matrix forms of all OEP's within a specified OEP type.
- virtual void [compute_3D](#) (const std::string &oepType, const double &x, const double &y, const double &z, std::shared_ptr< psi::Vector > &v) override
Compute value of potential in point x, y, z and save at v.
- virtual void [print_header](#) () const override
Header information.
- virtual std::shared_ptr< [OEPotential](#) > [clone](#) (void) const override
Make a deep copy of this object.
- virtual void [initialize](#) () override
Initialize the object (expert)

Protected Member Functions

- virtual void **rotate_oep** (psi::SharedMatrix r, psi::SharedMatrix R_prim, psi::SharedMatrix R_aux) override
- virtual void **translate_oep** (psi::SharedVector t) override

Additional Inherited Members

14.73.1 Detailed Description

Contains the following OEP types:

- Murrell-etal.S1
- Otto-Ladik.S2

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

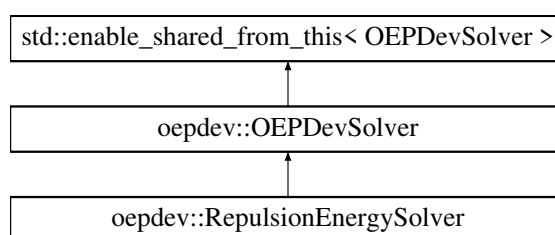
- oepdev/liboep/[oep.h](#)
- oepdev/liboep/oep_energy_pauli.cc

14.74 oepdev::RepulsionEnergySolver Class Reference

Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.

```
#include <solver.h>
```

Inheritance diagram for oepdev::RepulsionEnergySolver:



Public Member Functions

- **RepulsionEnergySolver** ([SharedWavefunctionUnion](#) wfn_union)
- virtual double [compute_oep_based](#) (const std::string &method="DEFAULT")
Compute property by using OEPs.
- virtual double [compute_benchmark](#) (const std::string &method="DEFAULT")
Compute property by using benchmark method.

Additional Inherited Members

14.74.1 Detailed Description

The implemented methods are shown below

Table 14.118: Methods available in the Solver

Keyword	Method Description
Benchmark Methods	
HAYES_STONE	<i>Default.</i> Pauli Repulsion energy at HF level from Hayes and Stone (1984).
DDS	Pauli Repulsion energy at HF level from Mandado and Hermida-Ramon (2012).
MURRELL_ETAL	Approximate Pauli Repulsion energy at HF level from Murrell et al (1967).
OTTO_LADIK	Approximate Pauli Repulsion energy at HF level from Otto and Ladik (1975).
EFP2	Approximate Pauli Repulsion energy at HF level from EFP2 model.
OEP-Based Methods	
MURRELL_ETAL_GDF_ESP	<i>Default.</i> OEP-Murrell et al's: S1 term via DF-OEP, S2 term via ESP-OEP.
MURRELL_ETAL_GDF_CAMM	OEP-Murrell et al's: S1 term via DF-OEP, S2 term via CAMM-OEP.
MURRELL_ETAL_ESP	OEP-Murrell et al's: S1 and S2 via ESP-OEP (not implemented)

Note:

- This solver also computes and prints the exchange energy at HF level (formulae are given below) for reference purposes.
- In order to construct this solver, **always** use the `OEPDevSolver::build` static factory method.

Below the detailed description of the implemented equations is given for each of the above provided methods. In the formulae across, it is assumed that the orbitals are real. The Coulomb notation for electron repulsion integrals (ERIs) is adopted; i.e,

$$(ac|bd) = \iint d\mathbf{r}_1 d\mathbf{r}_2 \phi_a(\mathbf{r}_1) \phi_c(\mathbf{r}_1) \frac{1}{r_{12}} \phi_b(\mathbf{r}_2) \phi_d(\mathbf{r}_2)$$

Greek subscripts denote basis set orbitals whereas Italic subscripts denote the occupied molecular orbitals.

Benchmark Methods

Pauli Repulsion energy at HF level by Hayes and Stone (1984).

For a closed-shell system, equation of Hayes and Stone (1984) becomes

$$E^{\text{Rep}} = 2 \sum_{kl} \left(V_{kl}^A + V_{kl}^B + T_{kl} \right) \left[[\mathbf{S}^{-1}]_{lk} - \delta_{lk} \right] \\ + \sum_{klmn} (kl|mn) \left\{ 2[\mathbf{S}^{-1}]_{kl}[\mathbf{S}^{-1}]_{mn} - [\mathbf{S}^{-1}]_{kn}[\mathbf{S}^{-1}]_{lm} - 2\delta_{kl}\delta_{mn} + \delta_{kn}\delta_{lm} \right\}$$

where \mathbf{S} is the overlap matrix between the doubly-occupied orbitals. The exact, pure exchange energy is for a closed shell case given as

$$E^{\text{Ex,pure}} = -2 \sum_{a \in A} \sum_{b \in B} (ab|ba)$$

Similarity transformation of molecular orbitals does not affect the resulting energies. The overall exchange-repulsion interaction energy is then (always net repulsive)

$$E^{\text{Ex-Rep}} = E^{\text{Ex,pure}} + E^{\text{Rep}}$$

Repulsion energy of Mandado and Hermida-Ramon (2011)

At the Hartree-Fock level, the exchange-repulsion energy from the density-based scheme of Mandado and Hermida-Ramon (2011) is fully equivalent to the method by Hayes and Stone (1984). However, density-based method enables to compute exchange-repulsion energy at any level of theory. It is derived based on the Pauli deformation density matrix,

$$\Delta \mathbf{D}^{\text{Pauli}} \equiv \mathbf{D}^{oo} - \mathbf{D}$$

where \mathbf{D}^{oo} and \mathbf{D} are the density matrix formed from mutually orthogonal sets of molecular orbitals within the entire aggregate (formed by symmetric orthogonalization of MO's) and the density matrix of the unperturbed system (that can be understood as a Hadamard sum $\mathbf{D} \equiv \mathbf{D}^A \oplus \mathbf{D}^B$).

At HF level, the Pauli deformation density matrix is given by

$$\Delta \mathbf{D}^{\text{Pauli}} = \mathbf{C} [\mathbf{S}^{-1} - \mathbf{1}] \mathbf{C}^\dagger$$

whereas the density matrix constructed from mutually orthogonal orbitals is

$$\mathbf{D}^{oo} = \mathbf{C} \mathbf{S}^{-1} \mathbf{C}^\dagger$$

In the above equations, \mathbf{S} is the overlap matrix between doubly occupied molecular orbitals of the entire aggregate.

Here, the expressions for the exchange-repulsion energy at any level of theory are shown for the case of open-shell system. The net repulsive energy is given as

$$E^{\text{Ex-Rep}} = E^{\text{Rep},1} + E^{\text{Rep},2} + E^{\text{Ex}}$$

where the one- and two-electron part of the repulsion energy is

$$\begin{aligned} E^{\text{Rep},1} &= E^{\text{Rep},\text{Kin}} + E^{\text{Rep},\text{Nuc}} \\ E^{\text{Rep},2} &= E^{\text{Rep},\text{el}-\Delta} + E^{\text{Rep},\Delta-\Delta} \end{aligned}$$

The kinetic and nuclear contributions are

$$\begin{aligned} E^{\text{Rep},\text{Kin}} &= 2 \sum_{\alpha\beta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} T_{\alpha\beta} \\ E^{\text{Rep},\text{Nuc}} &= 2 \sum_{\alpha\beta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} \sum_{z \in A,B} V_{\alpha\beta}^{(z)} \end{aligned}$$

whereas the electron-deformation and deformation-deformation interaction contributions are

$$\begin{aligned} E^{\text{Rep},\text{el}-\Delta} &= 4 \sum_{\alpha\beta\gamma\delta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} D_{\gamma\delta}(\alpha\beta|\gamma\delta) \\ E^{\text{Rep},\Delta-\Delta} &= 2 \sum_{\alpha\beta\gamma\delta \in A,B} \Delta D_{\alpha\beta}^{\text{Pauli}} \Delta D_{\gamma\delta}^{\text{Pauli}}(\alpha\beta|\gamma\delta) \end{aligned}$$

The associated exchange energy is given by

$$E^{\text{Ex}} = - \sum_{\alpha\beta\gamma\delta \in A,B} \left[D_{\alpha\delta}^{oo} D_{\beta\gamma}^{oo} - D_{\alpha\delta}^A D_{\beta\gamma}^A - D_{\alpha\delta}^B D_{\beta\gamma}^B \right] (\alpha\beta|\gamma\delta)$$

It is important to emphasise that, although, at HF level, the particular 'repulsive' and 'exchange' energies computed by using either Hayes and Stone or Mandado and Hermida-Ramon methods are not equal to each other, they sum up to exactly the same exchange-repulsion energy, $E^{\text{Ex-Rep}}$. Therefore, these methods at HF level are fully equivalent but the nature of partitioning of repulsive and exchange parts is different. It is also noted that the orbital localization does *not* affect the resulting energies, as opposed to the few approximate methods described below (Otto-Ladik and EFP2 methods).

Approximate Pauli Repulsion energy at HF level from Murrell et al.

By expanding the overlap matrix in a Taylor series one can show that the Pauli repulsion energy is approximately given as

$$E^{\text{Rep}} = E^{\text{Rep}}(\mathcal{O}(S)) + E^{\text{Rep}}(\mathcal{O}(S^2))$$

where the first-order term is

$$E^{\text{Rep}}(\mathcal{O}(S)) = -2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ V_{ab}^A + \sum_{c \in A} [2(ab|cc) - (ac|bc)] + V_{ab}^B + \sum_{d \in B} [2(ab|dd) - (ad|bd)] \right\}$$

whereas the second-order term is

$$\begin{aligned} E^{\text{Rep}}(\mathcal{O}(S^2)) &= 2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ \sum_{c \in A} S_{bc} \left[V_{ac}^B + 2 \sum_{d \in B} (ac|dd) \right] + \sum_{d \in B} S_{ad} \left[V_{bd}^A + 2 \sum_{x \in A} (bd|cx) \right] \right. \\ &\quad \left. - \sum_{c \in A} \sum_{d \in B} S_{cd} (ac|bd) \right\} \end{aligned}$$

Thus derived repulsion energy is invariant with respect to transformation of molecular orbitals, similarly as Hayes-Stone's method and density-based method. By using OEP technique, the above theory can be exactly re-cast *without* any further approximations.

Approximate Pauli Repulsion energy at HF level from Otto and Ladik (1975).

The Pauli repulsion energy is approximately given as

$$E^{\text{Rep}} = E^{\text{Rep}}(\mathcal{O}(S)) + E^{\text{Rep}}(\mathcal{O}(S^2))$$

where the first-order term is

$$E^{\text{Rep}}(\mathcal{O}(S)) = -2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ V_{ab}^A + 2 \sum_{c \in A} (ab|cc) - (ab|aa) + V_{ab}^B + 2 \sum_{d \in B} (ab|dd) - (ab|bb) \right\}$$

whereas the second-order term is

$$E^{\text{Rep}}(\mathcal{O}(S^2)) = 2 \sum_{a \in A} \sum_{b \in B} S_{ab}^2 \left\{ V_{aa}^B + V_{bb}^A + 2 \sum_{c \in A} (cc|bb) + 2 \sum_{d \in B} (aa|dd) - (aa|bb) \right\}$$

Thus derived repulsion energy is *not* invariant with respect to transformation of molecular orbitals, in contrast to Hayes-Stone's method and density-based method. It was shown that good results are obtained when using localized molecular orbitals, whereas using canonical molecular orbitals brings poor results. By using OEP technique, the above theory can be exactly re-cast *without* any further approximations.

Approximate Pauli Repulsion energy at HF level from Jensen and Gordon (1996).

The Pauli repulsion energy used within the EFP2 approach is approximately given as

$$E^{\text{Rep}} = E^{\text{Rep}}(\mathcal{O}(S)) + E^{\text{Rep}}(\mathcal{O}(S^2))$$

where the first-order term is

$$E^{\text{Rep}}(\mathcal{O}(S)) = -2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ \sum_{c \in A} F_{ac}^A S_{cb} + \sum_{d \in B} F_{bd}^B S_{da} - 2T_{ab} \right\}$$

whereas the second-order term is

$$E^{\text{Rep}}(\mathcal{O}(S^2)) = 2 \sum_{a \in A} \sum_{b \in B} S_{ab}^2 \left\{ \sum_{x \in A} \frac{-Z_x}{R_{xb}} + \sum_{y \in B} \frac{-Z_y}{R_{ya}} + \sum_{c \in A} \frac{2}{R_{bc}} + \sum_{d \in B} \frac{2}{R_{ad}} - \frac{1}{R_{ab}} \right\}$$

Thus derived repulsion energy is *not* invariant with respect to transformation of molecular orbitals, in contrast to Hayes-Stone's method and density-based method. It was shown that good results are obtained when using localized molecular orbitals, whereas using canonical molecular orbitals brings poor results.

In EFP2, exchange energy is approximated by spherical Gaussian approximation (SGO). The result of this is the following formula for the exchange energy:

$$E^{\text{Ex}} \approx -4 \sum_{a \in A} \sum_{b \in B} \sqrt{\frac{-2 \ln |S_{ab}|}{\pi}} \frac{S_{ab}^2}{R_{ab}}$$

In all the above formulas, R_{ij} are distances between position vectors of i -th and j -th point. The LMO centroids are defined by

$$\mathbf{r}_a = (a|\mathbf{r}|a)$$

where a denotes the occupied molecular orbital.

OEP-Based Methods

The Murrell et al's theory of Pauli repulsion for S-1 term and the Otto-Ladik's theory for S-2 term is here re-cast by introducing OEPs. The S-1 term is expressed via DF-OEP, whereas the S-2 term via ESP-OEP.

S-1 term (Murrell et al.)

The OEP reduction without any approximations leads to the following formula

$$E^{\text{Rep}}(\mathcal{O}(S^1)) = -2 \sum_{a \in A} \sum_{b \in B} S_{ab} \left\{ \sum_{\xi \in A} S_{b\xi} G_{\xi a}^A + \sum_{\eta \in B} S_{a\eta} G_{\eta b}^B \right\}$$

where the OEP matrices are given as

$$G_{\xi a}^A = \sum_{\xi' \in A} [\mathbf{S}^{-1}]_{\xi\xi'} \sum_{\alpha \in A} \left\{ C_{\alpha a} V_{\alpha\xi'}^A + \sum_{\mu \nu \in A} [2C_{\alpha a} D_{\mu\nu} - C_{\nu a} D_{\alpha\mu}] (\alpha\xi' | \mu\nu) \right\}$$

and analogously for molecule *B*. Here, the nuclear attraction integrals are denoted by $V_{\alpha\xi'}^A$.

S-2 term (Otto-Ladik)

After the OEP reduction, this contribution under Otto-Ladik approximation has the following form:

$$E^{\text{Rep}}(\mathcal{O}(S^2)) = 2 \sum_{a \in A} \sum_{b \in B} S_{ab}^2 \left\{ \sum_{x \in A} q_{xa} V_{bb}^{(x)} + \sum_{y \in B} q_{yb} V_{aa}^{(y)} \right\}$$

where the ESP charges associated with each occupied molecular orbital reproduce the *effective potential* of molecule in question, i.e.,

$$\sum_{x \in A} \frac{q_{xa}}{|\mathbf{r} - \mathbf{r}_x|} \cong v_a^A(\mathbf{r})$$

where the potential is given by

$$v_a^A(\mathbf{r}) = \sum_{x \in A} \frac{-Z_x}{|\mathbf{r} - \mathbf{r}_x|} + 2 \sum_{c \in A} \int \frac{\phi_c(\mathbf{r}') \phi_c(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}' - \frac{1}{2} \int \frac{\phi_a(\mathbf{r}') \phi_a(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} d\mathbf{r}'$$

14.74.2 Member Function Documentation

compute_oep_based()

```
double RepulsionEnergySolver::compute_oep_based (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one DEFAULT OEP-based method.

Parameters

<i>method</i>	- flavour of OEP model
---------------	------------------------

Implements [oepdev::OEPDevSolver](#).

compute_benchmark()

```
double RepulsionEnergySolver::compute_benchmark (
    const std::string & method = "DEFAULT" ) [virtual]
```

Each solver object has one `DEFAULT` benchmark method

Parameters

<i>method</i>	- benchmark method
---------------	--------------------

Implements [oepdev::OEPDevSolver](#).

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

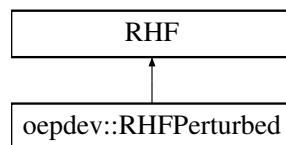
- oepdev/libsolver/[solver.h](#)
- oepdev/libsolver/solver_energy_pauli.cc

14.75 oepdev::RHPerturbed Class Reference

RHF theory under electrostatic perturbation.

```
#include <scf_perturb.h>
```

Inheritance diagram for oepdev::RHPerturbed:

**Public Member Functions**

- [RHPerturbed](#) (std::shared_ptr< psi::Wavefunction > ref_wfn, std::shared_ptr< psi::SuperFunctional > functional)
Build from wavefunction and superfunctional.
- [RHPerturbed](#) (std::shared_ptr< psi::Wavefunction > ref_wfn, std::shared_ptr< psi::SuperFunctional > functional, psi::Options &options, std::shared_ptr< psi::PSIO > psio)

Build from wavefunction and superfunctional + options and psio.

- virtual `~RHFPerturbed ()`

Clear memory.

- virtual double `compute_energy ()`

Compute total energy.

- virtual void `set_perturbation (std::shared_ptr< psi::Vector > field)`

Perturb the system with external electric field.

- virtual void `set_perturbation (const double &fx, const double &fy, const double &fz)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- virtual void `set_perturbation (std::shared_ptr< psi::Vector > position, const double &charge)`

Perturb the system with a point charge.

- virtual void `set_perturbation (const double &rx, const double &ry, const double &rz, const double &charge)`

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- `std::shared_ptr< psi::Matrix > Vpert () const`

Get a copy of the perturbation potential one-electron matrix.

- double `nuclear_interaction_energy () const`

Get the interaction energy of the nuclei with the perturbing potential.

Protected Member Functions

- virtual void `perturb_Hcore ()`

Add the electrostatic perturbation to the Hcore matrix.

Protected Attributes

- `std::shared_ptr< psi::Vector > perturbField_`

Perturbing electric field.

- `std::shared_ptr< PerturbCharges > perturbCharges_`

Perturbing charges.

- `std::shared_ptr< psi::Matrix > Vpert_`

Perturbation potential one-electron matrix.

- double `nuclearInteractionEnergy_`

Electrostatic interaction energy due to nuclei.

14.75.1 Detailed Description

Compute RHF wavefunction under the following conditions:

- external uniform electric field
- set of point charges The mixed conditions can also be used.

Theory

The electrostatic perturbation is here understood as a distribution of external (generally non-uniform) electric field. It is assumed that this perturbation is one-electron in nature. Therefore, the one-electron Hamiltonian is changed according to the following

$$\mathbf{H}^{\text{core}} \rightarrow \mathbf{H}^{\text{core}} + \sum_n q_n \mathbf{V}^{(n)} - \mathbb{M} \cdot \mathbf{F}$$

where q_n is the external classical point charge, $\mathbf{V}^{(n)}$ is the associated matrix of potential integrals, \mathbb{M} is the vector of dipole integrals and \mathbf{F} is an external uniform electric field. The total energy is then computed by performing an SCF procedure on the above one-electron Hamiltonian. The contribution due to nuclei is included, i.e.,

$$E_{\text{Nuc}} \rightarrow E_{\text{Nuc-Nuc}} + \sum_{In} \frac{q_n Z_I}{r_{In}} - \mu_{\text{Nuc}} \cdot \mathbf{F}$$

where μ_{Nuc} is the nuclear dipole moment and Z_I is the atomic number of the I th nucleus. It is added in the nuclear repulsion energy $E_{\text{Nuc-Nuc}}$ (note that the resulting energy can be negative as well depending on the electric field direction and configuration of point charges).

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

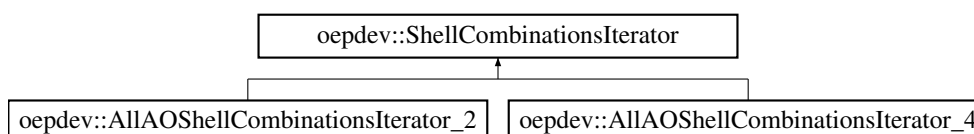
- oepdev/libutil/[scf_perturb.h](#)
- oepdev/libutil/scf_perturb.cc

14.76 oepdev::ShellCombinationsIterator Class Reference

Iterator for Shell Combinations. Abstract Base.

```
#include <integrals_iter.h>
```

Inheritance diagram for oepdev::ShellCombinationsIterator:



Public Member Functions

- [ShellCombinationsIterator](#) (int `nshell`)
Constructor.
- virtual [~ShellCombinationsIterator](#) ()
Destructor.
- virtual void [first](#) (void)=0
First iteration.
- virtual void [next](#) (void)=0
Next iteration.
- virtual std::shared_ptr< psi::BasisSet > [bs_1](#) (void) const
Grab the basis set of axis 1.
- virtual std::shared_ptr< psi::BasisSet > [bs_2](#) (void) const
Grab the basis set of axis 2.
- virtual std::shared_ptr< psi::BasisSet > [bs_3](#) (void) const
Grab the basis set of axis 3.
- virtual std::shared_ptr< psi::BasisSet > [bs_4](#) (void) const
Grab the basis set of axis 4.
- virtual int [P](#) (void) const
Grab the current shell P index.
- virtual int [Q](#) (void) const
Grab the current shell Q index.
- virtual int [R](#) (void) const
Grab the current shell R index.
- virtual int [S](#) (void) const
Grab the current shell S index.
- virtual bool [is_done](#) (void)
Return status of an iterator.
- virtual int [nshell](#) (void) const
Return number of shells this iterator is for.
- virtual std::shared_ptr< [AOIntegralsIterator](#) > [ao_iterator](#) (std::string mode="ALL") const

- virtual void [compute_shell](#) (std::shared_ptr< oepdev::TwoBodyAOInt > tei) const =0
- virtual void [compute_shell](#) (std::shared_ptr< psi::TwoBodyAOInt > tei) const =0

Static Public Member Functions

- static std::shared_ptr< [ShellCombinationsIterator](#) > [build](#) (const [IntegralFactory](#) &ints, std::string mode="ALL", int [nshell](#)=4)

Build shell iterator from [oepdev::IntegralFactory](#).

- static std::shared_ptr< [ShellCombinationsIterator](#) > [build](#) (std::shared_ptr< [IntegralFactory](#) > ints, std::string mode="ALL", int [nshell](#)=4)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

- static std::shared_ptr< [ShellCombinationsIterator](#) > [build](#) (const psi::IntegralFactory &ints, std::string mode="ALL", int [nshell](#)=4)

Build shell iterator from [psi::IntegralFactory](#).

- static std::shared_ptr< [ShellCombinationsIterator](#) > [build](#) (std::shared_ptr< [psi::IntegralFactory](#) > ints, std::string mode="ALL", int [nshell](#)=4)

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Protected Attributes

- SharedBasisSet [bs_1_](#)

Basis set of axis 1.

- SharedBasisSet [bs_2_](#)

Basis set of axis 2.

- SharedBasisSet [bs_3_](#)

Basis set of axis 3.

- SharedBasisSet [bs_4_](#)

Basis set of axis 4.

- const int [nshell_](#)

Number of shells this iterator is for.

- bool [done](#)

Status of an iterator.

14.76.1 Detailed Description

Date

2018/03/01 17:22:00

14.76.2 Constructor & Destructor Documentation

ShellCombinationsIterator()

```
ShellCombinationsIterator::ShellCombinationsIterator (
    int nshell )
```

Parameters

<i>nshell</i>	- number of shells this iterator is for
---------------	---

14.76.3 Member Function Documentation**build()** [1/2]

```
std::shared_ptr< ShellCombinationsIterator > ShellCombinationsIterator::build
(
    const IntegralFactory & ints,
    std::string mode = "ALL",
    int nshell = 4 ) [static]
```

Parameters

<i>ints</i>	- integral factory
<i>mode</i>	- mode of iteration (either ALL or UNIQUE)
<i>nshell</i>	- number of shells to iterate through

Returns

shell iterator

Examples:

example.integrals.iter.cc.

build() [2/2]

```
std::shared_ptr< ShellCombinationsIterator > ShellCombinationsIterator::build
(
    const psi::IntegralFactory & ints,
    std::string mode = "ALL",
    int nshell = 4 ) [static]
```


Parameters

<i>ints</i>	- integral factory
<i>mode</i>	- mode of iteration (either ALL or UNIQUE)
<i>nshell</i>	- number of shells to iterate through

Returns

shell iterator

compute_shell() [1/2]

```
void ShellCombinationsIterator::compute_shell (
    std::shared_ptr< oepdev::TwoBodyAOInt > tei ) const [pure virtual]
```

Compute integrals in a current shell. Works both for [oepdev::TwoBodyAOInt](#) and [psi::TwoBodyAOInt](#)

Parameters

<i>tei</i>	- two body integral object
------------	----------------------------

Implemented in [oepdev::AllAOShellCombinationsIterator_2](#), and [oepdev::AllAOShellCombinationsIterator_4](#).

compute_shell() [2/2]

```
void ShellCombinationsIterator::compute_shell (
    std::shared_ptr< psi::TwoBodyAOInt > tei ) const [pure virtual]
```

Compute integrals in a current shell. Works both for [oepdev::TwoBodyAOInt](#) and [psi::TwoBodyAOInt](#)

Parameters

<i>tei</i>	- two body integral object
------------	----------------------------

Implemented in [oepdev::AllAOShellCombinationsIterator_4](#).

ao_iterator()

```
std::shared_ptr< AOIntegralsIterator > ShellCombinationsIterator::ao_iterator
(
    std::string mode = "ALL" ) const [virtual]
```

Make an AO integral iterator based on current shell

Parameters

<i>mode</i>	- either "ALL" or "UNIQUE" (iterate over all or unique integrals)
-------------	---

Returns

iterator over AO integrals

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

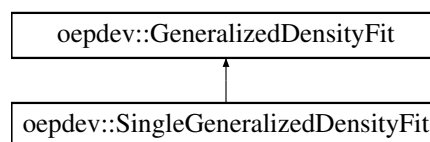
- [oepdev/libutil/integrals_iter.h](#)
- [oepdev/libutil/integrals_iter.cc](#)

14.77 oepdev::SingleGeneralizedDensityFit Class Reference

Generalized Density Fitting Scheme - Single Fit.

```
#include <oep_gdf.h>
```

Inheritance diagram for oepdev::SingleGeneralizedDensityFit:



Public Member Functions

- **SingleGeneralizedDensityFit** (std::shared_ptr< psi::BasisSet > bs_auxiliary, std::shared_ptr< psi::Matrix > v_vector)
- std::shared_ptr< psi::Matrix > [compute](#) (void)

Perform the generalized density fit.

Additional Inherited Members

14.77.1 Detailed Description

The density fitting map projects the OEP onto the auxiliary, nearly complete basis set space through application of the resolution of identity. Refer to [density fitting in complete space](#) for more details.

14.77.2 Determination of the OEP matrix

Coefficients \mathbf{G} are computed by using the following relation

$$\mathbf{G}^{(i)} = \mathbf{v}^{(i)} \cdot \mathbf{S}^{-1}$$

where

$$S_{\xi\eta} = (\xi|\eta)$$

$$v_{\xi}^{(i)} = (\xi|\hat{v}i)$$

In the above, $|$ denotes the single integration over electron coordinate, i.e.,

$$(a|b) \equiv \int d\mathbf{r} \phi_a^*(\mathbf{r}) \phi_b(\mathbf{r})$$

whereas the spatial form of the potential operator \hat{v} can be expressed by

$$v(\mathbf{r}) \equiv \int d\mathbf{r}' \frac{\rho(\mathbf{r}')}{|\mathbf{r}' - \mathbf{r}|}$$

with $\rho(\mathbf{r})$ being the effective one-electron density associated with \hat{v} .

14.77.3 Member Function Documentation

compute()

```
std::shared_ptr< psi::Matrix > SingleGeneralizedDensityFit::compute (
    void ) [virtual]
```

Returns

The OEP coefficients $G_{\xi i}$

Implements [oepdev::GeneralizedDensityFit](#).

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

- oepdev/liboep/[oep_gdf.h](#)
- oepdev/liboep/oep_gdf.cc

14.78 oepdev::GeneralizedPolarGEFactory::StatisticalSet Struct Reference

A structure to handle statistical data.

```
#include <gefp.h>
```

Public Attributes

- `std::vector< double >` [InducedInteractionEnergySet](#)
Interaction energy set.
- `std::vector< std::shared_ptr< psi::Matrix > >` [DensityMatrixSet](#)
Density matrix set.
- `std::vector< std::shared_ptr< psi::Vector > >` [InducedDipoleSet](#)
Induced dipole moment set.
- `std::vector< std::shared_ptr< psi::Vector > >` [InducedQuadrupoleSet](#)
Induced quadrupole moment set.
- `std::vector< std::shared_ptr< psi::Matrix > >` [JKMatrixSet](#)
Sum of J and K matrix set.

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

- `oepdev/libgefp/gefp.h`

14.79 oepdev::test::Test Class Reference

Manages test routines.

```
#include <test.h>
```

Public Member Functions

- [Test](#) (`std::shared_ptr< psi::Wavefunction > wfn`, `psi::Options &options`)
Construct the tester.
- [~Test](#) ()
Destructor.
- `double run` (void)
Perform the test.

Protected Member Functions

- `double test_basic` (void)
Test the basic functionalities of OEPEv.
- `double test_basis_rotation` (void)
Test the AO basis set rotation from [oepdev::ao_rotation_matrix](#).
- `double test_cis_rhf` (void)
Test the CIS(RHF) method.
- `double test_cis_uhf` (void)

- Test the CIS(UHF) method.*

 - double `test_cis_rhf_dl` (void)

Test the CIS(RHF) method with Davidson-Liu algorithm.

 - double `test_cis_uhf_dl` (void)

Test the CIS(UHF) method with Davidson-Liu algorithm.

 - double `test_cphf` (void)

Test the CPHF method.

 - double `test_dmatPol` (void)

Test the density matrix susceptibility ($X = 1$)

 - double `test_dmatPolX` (void)

Test the density matrix susceptibility.

 - double `test_eri_1_1` (void)

Test the oepdev::ERI_1_1 class against psi::ERI.

 - double `test_eri_2_2` (void)

Test the oepdev::ERI_2_2 class against psi::ERI.

 - double `test_eri_3_1` (void)

Test the oepdev::ERI_3_1 class against psi::ERI.

 - double `test_unitaryOptimizer` (void)

Test the oepdev::UnitaryOptimizer class.

 - double `test_unitaryOptimizer_2` (void)

Test the oepdev::UnitaryOptimizer_2 class.

 - double `test_unitaryOptimizer_4_2` (void)

Test the oepdev::UnitaryOptimizer_4_2 class.

 - double `test_scf_perturb` (void)

Test the oepdev::RHFPerturbed class.

 - double `test_quambo` (void)

Test the oepdev::QUAMBO class.

 - double `test_camm` (void)

Test the oepdev::CAMM class.

 - double `test_dmtp_pot_field` (void)

Test the oepdev::MultipoleConvergence class: potential and field calculations.

 - double `test_dmtp_energy` (void)

Test the oepdev::DMTP class for energy calculations.

 - double `test_efp2_energy` (void)

Test the oepdev::EFP2_GenEffPar and oepdev::EFP2_Computer classes.

 - double `test_oep_efp2_energy` (void)

Test the oepdev::EFP2_GenEffPar and oepdev::EFP2_Computer classes.

 - double `test_kabsch_superimposition` (void)

Test the oepdev::KabschSuperimposer.

 - double `test_dmtp_superimposition` (void)

- *Test the `oepdev::DMTP` class for superimposition.*
- double `test_esp_solver` (void)
 - *Test the `oepdev::ESPSolver`.*
- double `test_points_collection3d` (void)
 - *Test the cube file generation (`oepdev::Field3D` electrostatic potential and `oepdev::Points3DIterator` for cube collection)*
- double `test_ct_energy_benchmark_ol` (void)
 - *Test the Charge-transfer Energy Solver (benchmark method Otto-Ladik)*
- double `test_ct_energy_oep_based_ol` (void)
 - *Test the Charge-transfer Energy Solver (oep-based method Otto-Ladik)*
- double `test_rep_energy_benchmark_hs` (void)
 - *Test the Repulsion Energy Solver: (benchmark method Hayes-Stone)*
- double `test_rep_energy_benchmark_dds` (void)
 - *Test the Repulsion Energy Solver: (benchmark method Density-Based - DDS/HF)*
- double `test_rep_energy_benchmark_murrell_etal` (void)
 - *Test the Repulsion Energy Solver: (benchmark method Murrell-etal)*
- double `test_rep_energy_oep_based_murrell_etal` (void)
 - *Test the Repulsion Energy Solver: (OEP-based method Murrell-etal)*
- double `test_rep_energy_benchmark_ol` (void)
 - *Test the Repulsion Energy Solver: (benchmark method Otto-Ladik)*
- double `test_rep_energy_benchmark_efp2` (void)
 - *Test the Repulsion Energy Solver: (benchmark method EFP2)*
- double `test_custom` (void)
 - *Test the custom code (to be deprecated)*

Protected Attributes

- `std::shared_ptr< psi::Wavefunction > wfn_`
Wavefunction object.
- `psi::Options & options_`
Psi4 Options.

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

- `oepdev/libtest/test.h`
- `oepdev/libtest/basic.cc`
- `oepdev/libtest/basis_rotation.cc`
- `oepdev/libtest/camm.cc`
- `oepdev/libtest/cis_rhf_dl.cc`
- `oepdev/libtest/cis_rhf_explicit.cc`
- `oepdev/libtest/cis_uhf_dl.cc`

- oepdev/libtest/cis_uhf_explicit.cc
- oepdev/libtest/cphf.cc
- oepdev/libtest/ct_energy_benchmark_ol.cc
- oepdev/libtest/ct_energy_oep_based_ol.cc
- oepdev/libtest/dmatpol.cc
- oepdev/libtest/dmatpolX.cc
- oepdev/libtest/dmtp_energy.cc
- oepdev/libtest/dmtp_pot_field.cc
- oepdev/libtest/dmtp_superimposition.cc
- oepdev/libtest/efp2_energy.cc
- oepdev/libtest/eri_1_1.cc
- oepdev/libtest/eri_2_2.cc
- oepdev/libtest/eri_3_1.cc
- oepdev/libtest/esp_solver.cc
- oepdev/libtest/kabsch_superimposition.cc
- oepdev/libtest/oep_efp2_energy.cc
- oepdev/libtest/points_collection3d.cc
- oepdev/libtest/quambo.cc
- oepdev/libtest/rep_energy_benchmark_dds.cc
- oepdev/libtest/rep_energy_benchmark_efp2.cc
- oepdev/libtest/rep_energy_benchmark_hs.cc
- oepdev/libtest/rep_energy_benchmark_murrell_etal.cc
- oepdev/libtest/rep_energy_benchmark_ol.cc
- oepdev/libtest/rep_energy_oep_based_murrell_etal.cc
- oepdev/libtest/scf_perturb.cc
- oepdev/libtest/test.cc
- oepdev/libtest/test_custom.cc
- oepdev/libtest/unitary_optimizer.cc
- oepdev/libtest/unitary_optimizer_2.cc
- oepdev/libtest/unitary_optimizer_4_2.cc

14.80 oepdev::TIData Class Reference

Transfer Integral EET Data.

```
#include <ti_data.h>
```

Public Member Functions

- [TIData](#) ()
Constructor.
- virtual [~TIData](#) ()

Destructor.

- void [set_s](#) (double, double, double, double, double, double)
Set the overlap integrals between basis states, S_{ij} , for $ij=12,13,14,23,24,34$.
- void [set_e](#) (double, double, double, double)
Set the diagonal exciton Hamiltonian matrix elements E_n for $n=1,2,3,4$.
- void [set_de](#) (double, double)
Set environmental corrections ΔE_1 and ΔE_2 .
- void [set_trcamm_coupling](#) (oepdev::SharedDMTPConvergence)
Set the convergence object for TrCamm-based $V^{\text{Coul},(0)}$.
- virtual double [coupling_trcamm](#) (const std::string &rn)
- virtual double [coupling_direct](#) (void)
- virtual double [coupling_direct_coul](#) (void)
- virtual double [coupling_direct_exch](#) (void)
- virtual double [coupling_indirect](#) (void)
- virtual double [coupling_indirect_ti2](#) (void)
- virtual double [coupling_indirect_ti3](#) (void)
- virtual double [coupling_total](#) (void)
- virtual double [overlap_corrected](#) (const std::string &type)
- virtual double [overlap_corrected_direct](#) (void)
- virtual double [overlap_corrected_direct](#) (double v)
- virtual double [overlap_corrected_indirect](#) (double v, double s)

Public Attributes

- [oepdev::MultipoleConvergence::ConvergenceLevel trcamm_convergence](#)
Convergence object for Coulombic coupling under TrCamm approximation.
- bool [diagonal_correction](#)
Environmental correction activated?
- bool [mulliken_approximation](#)
Mulliken approximation activated?
- bool [overlap_correction](#)
Overlap correction activated?
- bool [trcamm_approximation](#)
TrCamm approximation activated?
- std::map< std::string, double > [v0](#)
- oepdev::SharedDMTPConvergence [v0_trcamm](#)
 V_0 Coul multipole convergence.
- double [s12](#)

Overlap matrix element between basis functions.

- double [s13](#)

Overlap matrix element between basis functions.

- double [s14](#)

Overlap matrix element between basis functions.

- double [s23](#)

Overlap matrix element between basis functions.

- double [s24](#)

Overlap matrix element between basis functions.

- double [s34](#)

Overlap matrix element between basis functions.

- double [e1](#)

Diagonal Hamiltonian matrix element.

- double [e2](#)

Diagonal Hamiltonian matrix element.

- double [e3](#)

Diagonal Hamiltonian matrix element.

- double [e4](#)

Diagonal Hamiltonian matrix element.

- double [de1](#)

Environmental correction to the E_n for $n=1,2$.

- double [de2](#)

Environmental correction to the E_n for $n=1,2$.

Protected Attributes

- double [c_](#)

Conversion factor (unused now)

14.80.1 Detailed Description

Container for storing and managing TI data for EET coupling calculations, according to Fujimoto JCP 2012:

- exciton Hamiltonian matrix elements
- overlap integrals between basis states
- TrCamm EET coupling convergence object

Contains useful methods to process exciton Hamiltonian matrix elements:

- compute direct and indirect EET coupling constants
- compute overlap-corrected exciton Hamiltonian off-diagonal matrix elements
- include or exclude environmental correction in the diagonal exciton Hamiltonian
- activate TrCamm approximation of $V^{\text{Coul},(0)}$
- activate Mulliken approximation for $V^{\text{Exch},(0)}$ and $V^{\text{CT},(0)}$

To activate/deactivate the various approximations and corrections listed above, set the following attributes

- `diagonal_correction`
- `mulliken_approximation`
- `overlap_correction`
- `trcamm_approximation`

to `true/false`, according to your need.

Example of usage.

```
{c++}
// Set up exciton Hamiltonian
TIData data = TIData();
data.set_s(S12, S13, S14, S32, S42, S34);
data.set_e(E1, E2, E3, E4);
data.set_de(E1 - E01, E2 - E02);
data.v0["COUL"] = V0.Coul;
data.v0["EXCH"] = V0.Exch;
data.v0["ET1"] = V0.ET1;
data.v0["ET2"] = V0.ET2;
data.v0["HT1"] = V0.HT1;
data.v0["HT2"] = V0.HT2;
data.v0["CT"] = V0.CT;
data.v0["EXCH.M"] = V0.Exch.M;
data.v0["CT.M"] = V0.CT.M;

// Set up approximations and corrections
data.diagonal_correction = true;
data.mulliken_approximation = false;
```

```

data.trcamm_approximation = false;
data.overlap_correction = true;

// Compute overlap-corrected indirect coupling matrix elements
double V_ET1 = data.overlap_corrected("ET1");
double V_ET2 = data.overlap_corrected("ET2");
double V_HT1 = data.overlap_corrected("HT1");
double V_HT2 = data.overlap_corrected("HT2");
double V_CT = data.overlap_corrected("CT");
double V_CTM = data.overlap_corrected("CTM");

// Compute final coupling contributions
double V_Coul = data.overlap_corrected("COUL");
double V_Exch = data.overlap_corrected("EXCH");
double V_Ovrl = data.overlap_corrected("OVRL");

double V_ExchM = data.overlap_corrected("EXCH.M");

double V_TI_2 = data.coupling_indirect_ti2();
double V_TI_3 = data.coupling_indirect_ti3();

data.diagonal_correction = false;
double V0_TI_2 = data.coupling_indirect_ti2();
double V0_TI_3 = data.coupling_indirect_ti3();

data.mulliken_approximation = true;
double V0_TI_3_M = data.coupling_indirect_ti3();
data.diagonal_correction = true;
double V_TI_3_M = data.coupling_indirect_ti3();

double V_direct = V_Coul + V_Exch + V_Ovrl;
double V_indirect = V_TI_2 + V_TI_3;

```

See also

[oepdev::EETCouplingSolver](#)

14.80.2 Member Function Documentation

coupling_trcamm()

```
double TIData::coupling_trcamm (
    const std::string & rn ) [virtual]
```

Compute Coulombic coupling approximated by TrCAMM.

Parameters

<i>rn</i>	- convergence of TrCAMM coupling. Can be from R1 to R5, which corresponds to the R^{-n} series expansion of distributed multipoles.
-----------	---

Returns

$$V^{\text{Coul},(0)} \approx V^{\text{TrCAMM},(0)}(R^{-n})$$

coupling_direct()

```
double TIData::coupling_direct (
    void ) [virtual]
```

Compute the direct EET coupling constant.

Returns

$$V^{\text{Dir}} = V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovr}}$$

Overlap and diagonal corrections as well as TrCamm and Mulliken approximations for Coulomb and pure exchange parts can be set.

coupling_direct_coul()

```
double TIData::coupling_direct_coul (
    void ) [virtual]
```

Compute the direct EET coupling constant in Forster limit (Coulombic approximation)

Returns

$$V^{\text{Dir}} = V^{\text{Coul}}$$

Overlap correction as well as TrCamm approximation for Coulomb coupling can be set.

coupling_direct_exch()

```
double TIData::coupling_direct_exch (
    void ) [virtual]
```

Compute the direct EET coupling constant due to pure exchange.

Returns

$$V^{\text{Dir}} = V^{\text{Exch}}$$

Overlap correction as well as Mulliken approximation for pure exchange coupling can be set.

coupling_indirect()

```
double TIData::coupling_indirect (
    void ) [virtual]
```

Compute the indirect EET coupling constant.

Returns

$$V^{\text{Indir}} = V^{\text{TI},(2)} + V^{\text{TI},(3)}$$

Overlap and diagonal corrections as well as Mulliken approximations for $V^{\text{CT},(0)}$ can be set.

coupling_indirect_ti2()

```
double TIData::coupling_indirect_ti2 (
    void ) [virtual]
```

Compute the indirect EET coupling constant in second-order with respect to TI.

Returns

$$V^{TI,(2)} = -\frac{V^{ET1}V^{HT2}}{E_3-E_1} - \frac{V^{ET2}V^{HT1}}{E_4-E_1}$$

Overlap and diagonal corrections can be set.

coupling_indirect_ti3()

```
double TIData::coupling_indirect_ti3 (
    void ) [virtual]
```

Compute the indirect EET coupling constant in third-order with respect to TI.

Returns

$$V^{TI,(3)} = \frac{V^{CT}(V^{ET1}V^{ET2}+V^{HT1}V^{HT2})}{(E_3-E_1)(E_4-E_1)}$$

Overlap and diagonal corrections as well as Mulliken approximations for $V^{CT,(0)}$ can be set.

coupling_total()

```
double TIData::coupling_total (
    void ) [virtual]
```

Compute the *total* EET coupling constant.

Returns

$$V^{\text{Total}} = V^{\text{Dir}} + V^{\text{Indir}}$$

Overlap and diagonal corrections, TrCamm approximation for Coulomb coupling, and Mulliken approximations for pure exchange coupling and $V^{CT,(0)}$ can be set.

overlap_corrected()

```
double TIData::overlap_corrected (
    const std::string & type ) [virtual]
```

Compute overlap corrected matrix elements.

Parameters

<i>type</i>	<p>- matrix element $V^{\text{type},(0)}$ subject to overlap correction, where type is one of the following:</p> <ul style="list-style-type: none"> • COUL - $V^{\text{Coul},(0)}$, • EXCH - $V^{\text{Exch},(0)}$, • TrCamm_R1 - $V^{\text{TrCamm},(0)}(R^{-1})$, • TrCamm_R2 - $V^{\text{TrCamm},(0)}(R^{-2})$, • TrCamm_R3 - $V^{\text{TrCamm},(0)}(R^{-3})$, • TrCamm_R4 - $V^{\text{TrCamm},(0)}(R^{-4})$, • TrCamm_R5 - $V^{\text{TrCamm},(0)}(R^{-5})$, • ET1 - $V^{\text{ET1},(0)}$, • ET2 - $V^{\text{ET2},(0)}$, • HT1 - $V^{\text{HT1},(0)}$, • HT2 - $V^{\text{HT2},(0)}$, • CT - $V^{\text{CT},(0)}$, • CT_M - Mulliken-approximated $V^{\text{CT},(0)}$, • EXCH_M - Mulliken-approximated $V^{\text{Exch},(0)}$.
-------------	---

If type = OVRL, the overlap-correction to the direct EET coupling constant is returned, V^{Ovrl} .

Returns

overlap-corrected exciton Hamiltonian matrix element contribution of selected type

Diagonal correction can be set.

overlap_corrected_direct() [1/2]

```
double TIData::overlap_corrected_direct (
    void ) [virtual]
```

Compute overlap-corrected direct EET coupling constant.

Returns

$$V^{\text{Dir}} = V^{\text{Coul}} + V^{\text{Exch}} + V^{\text{Ovrl}}$$

Diagonal correction, TrCamm approximation and Mulliken approximation can be set.

overlap_corrected_direct() [2/2]

```
double TIData::overlap_corrected_direct (
    double v ) [virtual]
```

Compute overlap-corrected direct EET coupling constant from value v .

Returns

$$\frac{v}{1-S_{12}^2}$$

overlap_corrected_indirect()

```
double TIData::overlap_corrected_indirect (
    double v,
    double s ) [virtual]
```

Compute overlap-corrected coupling constant from value v and associated overlap integral s .

Returns

$$\frac{1}{1-s^2} \left(v - \frac{(E_1+E_2)s}{2} \right)$$

Diagonal correction can be set.

14.80.3 Member Data Documentation**v0**

```
std::map<std::string, double> oepdev::TIData::v0
```

Dictionary of all zeroth-order off-diagonal matrix elements.

Use only the following keywords:

- COUL - $V^{\text{Coul},(0)}$,
- EXCH - $V^{\text{Exch},(0)}$,
- TrCAMM_R1 - $V^{\text{TrCAMM},(0)}(R^{-1})$,
- TrCAMM_R2 - $V^{\text{TrCAMM},(0)}(R^{-2})$,
- TrCAMM_R3 - $V^{\text{TrCAMM},(0)}(R^{-3})$,
- TrCAMM_R4 - $V^{\text{TrCAMM},(0)}(R^{-4})$,

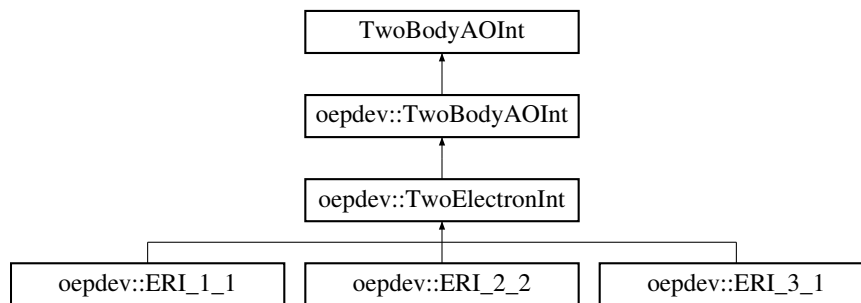
- $\text{TrCAMM_R5} - V^{\text{TrCAMM},(0)}(R^{-5}),$
- $\text{ET1} - V^{\text{ET1},(0)},$
- $\text{ET2} - V^{\text{ET2},(0)},$
- $\text{HT1} - V^{\text{HT1},(0)},$
- $\text{HT2} - V^{\text{HT2},(0)},$
- $\text{CT} - V^{\text{CT},(0)},$
- $\text{CT_M} - \text{Mulliken-approximated } V^{\text{CT},(0)},$
- $\text{EXCH_M} - \text{Mulliken-approximated } V^{\text{Exch},(0)},$
- $\text{OVR} - V^{\text{Ovr}}.$

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

- [oepdev/libsolver/ti_data.h](#)
- [oepdev/libsolver/ti_data.cc](#)

14.81 oepdev::TwoBodyAOInt Class Reference

Inheritance diagram for oepdev::TwoBodyAOInt:



Public Member Functions

- virtual void [compute](#) (std::shared_ptr< psi::Matrix > &result, int ibs1=0, int ibs2=2)
Compute two-body two-centre integral and put it into matrix.
- virtual void [compute](#) (psi::Matrix &result, int ibs1=0, int ibs2=2)
- virtual size_t [compute_shell](#) (int, int, int, int)=0
- virtual size_t [compute_shell](#) (int, int, int)=0
- virtual size_t [compute_shell](#) (int, int)=0
- virtual size_t [compute_shell_deriv1](#) (int, int, int, int)=0
- virtual size_t [compute_shell_deriv2](#) (int, int, int, int)=0

- virtual size_t **compute_shell_deriv1** (int, int, int)=0
- virtual size_t **compute_shell_deriv2** (int, int, int)=0
- virtual size_t **compute_shell_deriv1** (int, int)=0
- virtual size_t **compute_shell_deriv2** (int, int)=0

Protected Member Functions

- **TwoBodyAOInt** (const [IntegralFactory](#) *intsfactory, int deriv=0)
- **TwoBodyAOInt** (const [TwoBodyAOInt](#) &rhs)

14.81.1 Member Function Documentation

compute() [1/2]

```
void oepdev::TwoBodyAOInt::compute (
    std::shared_ptr< psi::Matrix > & result,
    int ibs1 = 0,
    int ibs2 = 2 ) [virtual]
```

Parameters

<i>result</i>	- matrix where to store (i j) two-body integrals
<i>ibs1</i>	- first basis set axis
<i>ibs2</i>	- second basis set axis

compute() [2/2]

```
void oepdev::TwoBodyAOInt::compute (
    psi::Matrix & result,
    int ibs1 = 0,
    int ibs2 = 2 ) [virtual]
```

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

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

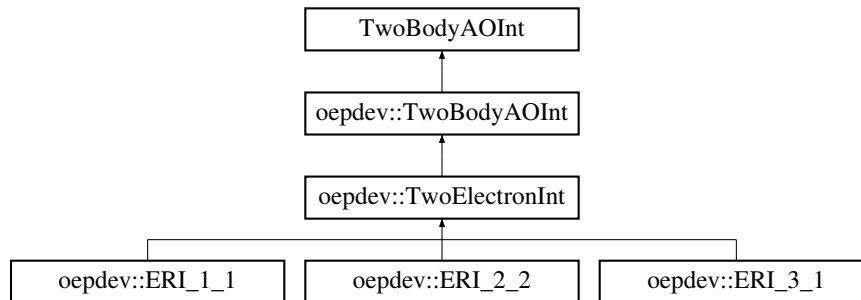
- oepdev/libpsi/[integral.h](#)
- oepdev/libpsi/integral.cc

14.82 oepdev::TwoElectronInt Class Reference

General Two Electron Integral.

```
#include <eri.h>
```

Inheritance diagram for oepdev::TwoElectronInt:



Public Member Functions

- **TwoElectronInt** (const [IntegralFactory](#) *integral, int deriv, bool use_shell_pairs)
- virtual size_t [compute_shell](#) (int, int)

Compute ERI's between 2 shells. Result is stored in buffer.
- virtual size_t [compute_shell](#) (int, int, int)

Compute ERI's between 3 shells. Result is stored in buffer.
- virtual size_t [compute_shell](#) (int, int, int, int)

Compute ERI's between 4 shells. Result is stored in buffer.
- virtual size_t [compute_shell](#) (const psi::AOShellCombinationsIterator &)
- virtual size_t [compute_shell_deriv1](#) (int, int)

Compute first derivatives of ERI's between 2 shells.
- virtual size_t [compute_shell_deriv2](#) (int, int)

Compute second derivatives of ERI's between 2 shells.
- virtual size_t [compute_shell_deriv1](#) (int, int, int)

Compute first derivatives of ERI's between 3 shells.
- virtual size_t [compute_shell_deriv2](#) (int, int, int)

Compute second derivatives of ERI's between 3 shells.
- virtual size_t [compute_shell_deriv1](#) (int, int, int, int)

Compute first derivatives of ERI's between 4 shells.
- virtual size_t [compute_shell_deriv2](#) (int, int, int, int)

Compute second derivatives of ERI's between 4 shells.

Protected Member Functions

- int [get_cart_am](#) (int am, int n, int x)
Get the angular momentum per Cartesian component.
- double [get_R](#) (int N, int L, int M)
Get the (N,L,M)th McMurchie-Davidson coefficient.
- virtual size_t [compute_doublet](#) (int, int)
Computes the ERI's between three shells.
- virtual size_t [compute_triplet](#) (int, int, int)
Computes the ERI's between three shells.
- virtual size_t [compute_quartet](#) (int, int, int, int)
Computes the ERI's between four shells.

Protected Attributes

- const int [max_am_](#)
Maximum angular momentum.
- const int [n_max_am_](#)
Maximum number of angular momentum functions.
- psi::Fjt * [fjt_](#)
Computes the fundamental: Boys function value at T for degree v.
- bool [use_shell_pairs_](#)
Should we use shell pair information?
- const double [cartMap_](#) [60]
Map of Cartesian components per each am.
- const double [df_](#) [8]
Double factorial array.
- double * [mdh_buffer_R_](#)
Buffer for the McMurchie-Davidson-Hermite R coefficients.

14.82.1 Detailed Description

Implements the McMurchie-Davidson recursive scheme for all integral types. The integral can be defined for any number of Gaussian centres, thus it is not limited to 2-by-2 four-centre ERI. Currently implemented subtypes are:

- [oepdev::ERI_1_1](#) - 2-centre electron-repulsion integral (i|j)
- [oepdev::ERI_2_2](#) - 4-centre electron-repulsion integral (ij|kl)
- [oepdev::ERI_3_1](#) - 4-centre electron-repulsion integral (ijk|l)

See also

[The Integral Package Library](#)

14.82.2 Member Function Documentation

compute_shell()

```
size_t oepdev::TwoElectronInt::compute_shell (
    const psi::AOShellCombinationsIterator & shellIter ) [virtual]
```

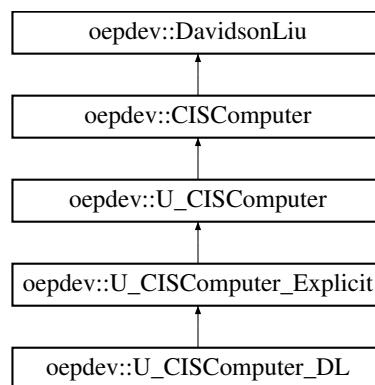
Compute ERIs between 4 shells. Result is stored in buffer. Only for use with [ERI_2.2](#) and the same basis sets, otherwise shell pairs won't be compatible.

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

- oepdev/libints/[eri.h](#)
- oepdev/libints/eri.cc

14.83 oepdev::U_CISComputer Class Reference

Inheritance diagram for oepdev::U_CISComputer:



Public Member Functions

- **U_CISComputer** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)

Protected Member Functions

- virtual void **print_excited_state_character_** (int l)

Additional Inherited Members

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

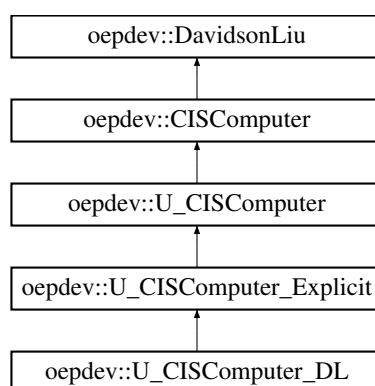
- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis_uhf.cc

14.84 oepdev::U_CISComputer_DL Class Reference

CIS Computer with UHF reference: Davidson-Liu Solver.

```
#include <cis.h>
```

Inheritance diagram for oepdev::U_CISComputer_DL:



Public Member Functions

- **U_CISComputer_DL** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)

Protected Member Functions

- virtual void **set_nstates_** (void)
- virtual void **transform_integrals_** (void)
- virtual void **allocate_hamiltonian_** (void)
- virtual void **build_hamiltonian_** (void)
- virtual void **diagonalize_hamiltonian_** (void)
- virtual void **davidson_liu_compute_diagonal_hamiltonian** (void)
- virtual void **davidson_liu_compute_sigma** (void)

Additional Inherited Members

14.84.1 Detailed Description

Associated options:

- `CIS_TYPE` - must be set to `DAVIDSON_LIU` (Default).
- `CIS_SCHWARTZ_CUTOFF` - Cutoff for Schwartz ERI screening. Default: 0.0.

Implementation

Diagonal Hamiltonian elements

They are computed by using direct method with Schwartz screening of AO ERI's. The implementation formula is

$$H_{ii}^{aa} = \epsilon_a - \epsilon_i + \sum_{\alpha\beta\gamma\delta} (\alpha\beta|\gamma\delta) C_{\alpha i} C_{\delta a} (C_{\beta a} C_{\gamma i} - C_{\beta i} C_{\gamma a})$$

$$H_{ii}^{\bar{a}\bar{a}} = \epsilon_{\bar{a}} - \epsilon_{\bar{i}} + \sum_{\alpha\beta\gamma\delta} (\alpha\beta|\gamma\delta) C_{\alpha \bar{i}} C_{\delta \bar{a}} (C_{\beta \bar{a}} C_{\gamma \bar{i}} - C_{\beta \bar{i}} C_{\gamma \bar{a}})$$

Sigma vectors

The sigma vectors are computed from

$$\sigma_i^{a,k} = (\epsilon_a - \epsilon_i) b_i^{a,k} + J_i^a(\mathbf{T}^{(k)}) + J_i^a(\overline{\mathbf{T}^{(k)}}) - K_i^a(\mathbf{T}^{(k)})$$

$$\sigma_{\bar{i}}^{\bar{a},k} = (\epsilon_{\bar{a}} - \epsilon_{\bar{i}}) b_{\bar{i}}^{\bar{a},k} + J_{\bar{i}}^{\bar{a}}(\mathbf{T}^{(k)}) + J_{\bar{i}}^{\bar{a}}(\overline{\mathbf{T}^{(k)}}) - K_{\bar{i}}^{\bar{a}}(\mathbf{T}^{(k)})$$

where k labels the vectors and where the generalized one-particle density matrices are defined by

$$T_{\gamma\delta}^{(k)} = \sum_{jb} C_{\delta b} b_j^{b,k} C_{\gamma j}$$

$$\overline{T}_{\gamma\delta}^{(k)} = \sum_{\bar{j}\bar{b}} C_{\delta \bar{b}} \bar{b}_{\bar{j}}^{\bar{b},k} C_{\gamma \bar{j}}$$

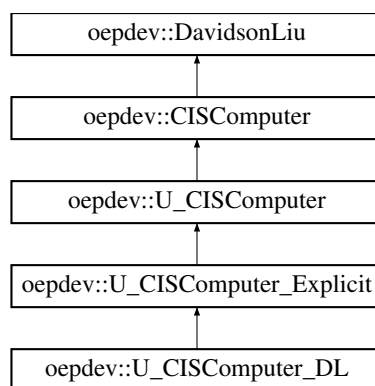
The **J** and **K** matrices in AO basis are computed by using the `psi::JK` object, and subsequently transformed to CMO's.

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

- `oepdev/libutil/cis.h`
- `oepdev/libutil/cis_uhf_dl.cc`

14.85 oepdev::U_CISComputer_Explicit Class Reference

Inheritance diagram for `oepdev::U_CISComputer_Explicit`:



Public Member Functions

- **U_CISComputer_Explicit** (std::shared_ptr< psi::Wavefunction > wfn, psi::Options &opt)

Protected Member Functions

- virtual void **set_beta_** (void)
- virtual void **build_hamiltonian_** (void)

Additional Inherited Members

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

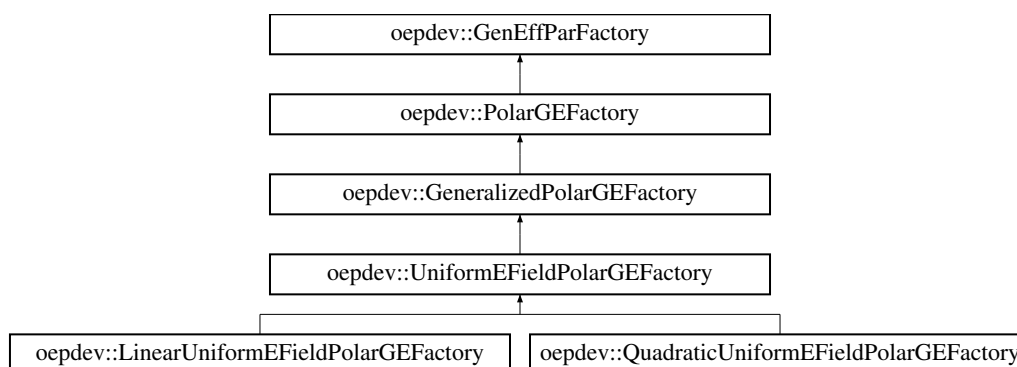
- oepdev/libutil/[cis.h](#)
- oepdev/libutil/cis_uhf_explicit.cc

14.86 oepdev::UniformEFieldPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Parameterization.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::UniformEFieldPolarGEFactory:



Public Member Functions

- **UniformEFieldPolarGEFactory** (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
- void [compute_samples](#) (void)
Compute samples of density matrices and select electric field distributions.
- virtual void [compute_gradient](#) (int i, int j)=0
Compute Gradient vector associated with the i-th and j-th basis set function.
- virtual void [compute_hessian](#) (void)=0
Compute Hessian matrix (independent on the parameters)

Additional Inherited Members

14.86.1 Detailed Description

Implements a class of density matrix susceptibility models for parameterization in the uniform electric field.

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

- oepdev/libgefp/gefp.h
- oepdev/libgefp/gefp_polar_uniform_base.cc

14.87 oepdev::UnitaryOptimizer Class Reference

Find the optimum unitary matrix of quadratic matrix equation.

```
#include <unitary_optimizer.h>
```

Public Member Functions

- [UnitaryOptimizer](#) (double *R, double *P, int n, double conv=1.0e-6, int maxiter=100, bool verbose=true)
Create from R and P matrices and optimization options.
- [UnitaryOptimizer](#) (std::shared_ptr< psi::Matrix > R, std::shared_ptr< psi::Vector > P, double conv=1.0e-6, int maxiter=100, bool verbose=true)
Create from R and P matrices and optimization options.
- [~UnitaryOptimizer](#) ()
Clear memory.
- bool [maximize](#) ()
Run the minimization.
- bool [minimize](#) ()
Run the maximization.

- `std::shared_ptr< psi::Matrix > X ()`
Get the unitary matrix (solution)
- `double * get_X () const`
Get the unitary matrix (pointer to solution)
- `double Z ()`
Get the actual value of Z function.
- `bool success () const`
Get the status of the optimization.

Protected Member Functions

- `UnitaryOptimizer (int n, double conv, int maxiter, bool verbose)`
Initialize the basic memory.
- `void common_init_ ()`
Prepare the optimizer.
- `void run_ (const std::string &opt)`
Run the optimization (intermediate interface)
- `void optimize_ (const std::string &opt)`
Run the optimization (inner interface)
- `void refresh_ ()`
Restore the initial state of the optimizer.
- `void update_conv_ ()`
Update the convergence.
- `void update_iter_ ()`
Update the iterates.
- `void update_Z_ ()`
Update Z value.
- `void update_RP_ ()`
Uptade R and P matrices.
- `void update_X_ ()`
Update the solution matrix X.
- `double eval_Z_ (double *X, double *R, double *P)`
Evaluate the objective Z function.
- `double eval_Z_ ()`
- `double eval_dZ_ (double g, double *R, double *P, int i, int j)`
Evaluate the change in Z.
- `double eval_Z_trial_ (int i, int j, double gamma)`
Evaluate the trial Z value.
- `void form_X0_ ()`
Create identity matrix.

- void [form_X_](#) (int i, int j, double gamma)
Form unitary matrix X (store in buffer Xnew_)
- void [form_next_X_](#) (const std::string &opt)
Form the next unitary matrix X.
- [ABCD get_ABCD_](#) (int i, int j)
Retrieve [ABCD](#) parameters for root search.
- void [find_roots_boyd_](#) (const [ABCD](#) &abcd)
Solve for all roots of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$ -> implements Boyd's method.*
- double [find_root_halley_](#) (double x0, const [ABCD](#) &abcd)
Solve for root of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$ -> implements Halley's method.*
- double [find_gamma_](#) (const [ABCD](#) &abcd, int i, int j, const std::string &opt)
Compute gamma from roots of base equations.
- bool [lt_](#) (double a, double b)
less-than function
- bool [gt_](#) (double a, double b)
greater-than function
- double [func_0_](#) (double g, const [ABCD](#) &abcd)
Function $f(\text{gamma}) = d(dZ)/d\text{gamma}$.
- double [func_1_](#) (double g, const [ABCD](#) &abcd)
Gradient of $f(\text{gamma})$
- double [func_2_](#) (double g, const [ABCD](#) &abcd)
Hessian of $f(\text{gamma})$ - used only for Halley method (not implemented since Boyd method is more suitable here)
- std::shared_ptr< psi::Matrix > [psi_X_](#) ()
Form the Psi4 matrix with the transformation matrix.

Protected Attributes

- const int [n_](#)
Dimension of the problem.
- const double [conv_](#)
Convergence.
- const int [maxiter_](#)
Maximum number of iterations.
- const bool [verbose_](#)
Verbose mode.
- double * [R_](#)
R matrix.
- double * [P_](#)

- P* vector.
- double * [R0_](#)
Reference R matrix.
- double * [P0_](#)
Reference P vector.
- double * [X_](#)
X Matrix (accumulated solution)
- double * [W_](#)
Work place.
- double * [Xold_](#)
Temporary X matrix.
- double * [Xnew_](#)
Temporary X matrix.
- int [niter_](#)
Current number of iterations.
- double [S_](#) [4]
Current solutions.
- double [Zinit_](#)
Initial Z value.
- double [Zold_](#)
Old Z value.
- double [Znew_](#)
New Z value.
- double [conv_current_](#)
Current convergence.
- bool [success_](#)
Status of optimization.

14.87.1 Detailed Description

The objective function of the orthogonal matrix \mathbf{X}

$$Z(\mathbf{X}) \equiv \sum_{ijkl} X_{ij} X_{kl} R_{jl} - \sum_{ij} X_{ij} P_j$$

is optimized by using the Jacobi iteration algorithm. In the above equation, \mathbf{R} is a square, general real matrix of size $N \times N$ whereas \mathbf{P} is a real vector of length N .

Algorithm.

Optimization of \mathbf{X} is factorized into a sequence of 2-dimensional rotations with one real parameter γ :

$$\mathbf{X}^{\text{New}} = \mathbf{U}(\gamma) \cdot \mathbf{X}^{\text{Old}}$$

where

$$\mathbf{U}(\gamma) \equiv \begin{pmatrix} \ddots & & & & \\ & \cos(\gamma) & \cdots & \sin(\gamma) & \\ & \vdots & \ddots & \vdots & \\ & -\sin(\gamma) & \cdots & \cos(\gamma) & \\ & & & & \ddots \end{pmatrix}$$

is the Jacobi transformation matrix constructed for the I th and J th element from the entire N -dimensional set. For the sake of algorithmic simplicity, every iteration after $\mathbf{U}(\gamma)$ has been formed, \mathbf{X}^{Old} is for a while assumed to be an identity matrix and the \mathbf{R} matrix and \mathbf{P} vector are transformed according to the following formulae

$$\begin{aligned} \mathbf{R} &\rightarrow \mathbf{U}\mathbf{R}\mathbf{U}^T \\ \mathbf{P} &\rightarrow \mathbf{U}\mathbf{P} \end{aligned}$$

The full transformation matrix is accumulated in the memory buffer until convergence.

In each iteration, the optimum angle γ is found as follows: First, the roots of the finite Fourier series

$$A \sin(\gamma) + B \cos(\gamma) + C \sin(2\gamma) + D \cos(2\gamma) = 0$$

are found. In the above equations, the expansion coefficients are given as

$$\begin{aligned} A &= P_I + P_J - \sum_{k \neq I, J} (R_{Ik} + R_{Jk} + R_{kI} + R_{kJ}) \\ B &= P_I - P_J - \sum_{k \neq I, J} (R_{Ik} - R_{Jk} + R_{kI} - R_{kJ}) \\ C &= -2(R_{IJ} + R_{JI}) \\ D &= -2(R_{II} - R_{JJ}) \end{aligned}$$

and I, J are the chosen indices in the Jacobi iteration subspace. The roots are evaluated by applying the Boyd's method[1], in which they are given as

$$\gamma_n = \Re[-i \ln(\lambda_n)]$$

where λ_n is an eigenvalue of the following 4 by 4 complex matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{D+iC}{D-iC} & -\frac{B+iC}{D-iC} & 0 & -\frac{B-iC}{D-iC} \end{pmatrix}$$

Once the four roots of the Fourier series equation are found, one solution out of four is chosen which satisfies the global optimum condition, i.e., the largest increase/decrease in the objective function given by

$$\delta Z = A(1 - \cos(\gamma)) + B \sin(\gamma) + C \sin^2(\gamma) + \frac{D}{2} \sin(2\gamma)$$

The discrimination between the minima/maxima is performed based on the evaluation of the Hessian of Z with respect to γ ,

$$\frac{\partial^2 Z}{\partial \gamma^2} = A \cos(\gamma) - B \sin(\gamma) + 2C \cos(2\gamma) - 2D \sin(2\gamma)$$

All the $N(N-1)/2$ unique pairs of molecular orbitals are checked and the optimal set of γ, I, J is chosen to construct \mathbf{X}^{New} .

References:

[1] Boyd, J.P.; J. Eng. Math. (2006) 56, pp. 203-219

14.87.2 Constructor & Destructor Documentation

UnitaryOptimizer() [1/3]

```
oepdev::UnitaryOptimizer::UnitaryOptimizer (
    double * R,
    double * P,
    int n,
    double conv = 1.0e-6,
    int maxiter = 100,
    bool verbose = true )
```

Parameters

<i>R</i>	- R matrix
<i>P</i>	- P vector
<i>n</i>	- dimensionality of the problem (<i>N</i>)
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

UnitaryOptimizer() [2/3]

```
oepdev::UnitaryOptimizer::UnitaryOptimizer (
    std::shared_ptr< psi::Matrix > R,
    std::shared_ptr< psi::Vector > P,
    double conv = 1.0e-6,
    int maxiter = 100,
```

```
bool verbose = true )
```

Parameters

R	- \mathbf{R} matrix
P	- \mathbf{P} vector
<i>conv</i>	- convergence in the Z function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

UnitaryOptimizer() [3/3]

```
oepdev::UnitaryOptimizer::UnitaryOptimizer (
    int n,
    double conv,
    int maxiter,
    bool verbose ) [protected]
```

Parameters

n	- dimensionality of the problem (N)
<i>conv</i>	- convergence in the Z function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

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

- [oepdev/libutil/unitary_optimizer.h](#)
- [oepdev/libutil/unitary_optimizer.cc](#)

14.88 oepdev::UnitaryOptimizer_2 Class Reference

Find the optimim unitary matrix for quadratic matrix equation with trace.

```
#include <unitary_optimizer.h>
```

Public Member Functions

- [UnitaryOptimizer_2](#) (double *P, int n, double conv=1.0e-6, int maxiter=100, bool verbose=true)
Create from P tensor and optimization options.
- [~UnitaryOptimizer_2](#) ()

- *Clear memory.*
- bool [maximize](#) ()
- *Run the minimization.*
- bool [minimize](#) ()
- *Run the maximization.*
- std::shared_ptr< psi::Matrix > [X](#) ()
- *Get the unitary matrix (solution)*
- double * [get_X](#) () const
- *Get the unitary matrix (pointer to solution)*
- double [Z](#) ()
- *Get the actual value of Z function.*
- bool [success](#) () const
- *Get the status of the optimization.*

Protected Member Functions

- [UnitaryOptimizer_2](#) (int n, double conv, int maxiter, bool verbose)
- *Initialize the basic memory.*
- void [common_init_](#) ()
- *Prepare the optimizer.*
- void [run_](#) (const std::string &opt)
- *Run the optimization (intermediate interface)*
- void [optimize_](#) (const std::string &opt)
- *Run the optimization (inner interface)*
- void [refresh_](#) ()
- *Restore the initial state of the optimizer.*
- void [update_conv_](#) ()
- *Update the convergence.*
- void [update_iter_](#) ()
- *Update the iterates.*
- void [update_Z_](#) ()
- *Update Z value.*
- void [update_P_](#) ()
- *Uptade P tensor.*
- void [update_X_](#) ()
- *Update the solution matrix X.*
- double [eval_Z_](#) (double *X, double *P)
- *Evaluate the objective Z function.*
- double [eval_Z_](#) ()
- double [eval_dZ_](#) (double g, double *P, int I, int J)

- Evaluate the change in Z.*

 - double `eval_Z_trial_` (int I, int J, double gamma)

Evaluate the trial Z value.
- void `form_X0_` ()

Create identity matrix.
- void `form_X_` (int I, int J, double gamma)

Form unitary matrix X (store in buffer Xnew_)
- void `form_next_X_` (const std::string &opt)

Form the next unitary matrix X.
- `Fourier5` `get_fourier_` (int I, int J)

Retrieve ABCD parameters for root search.
- void `find_roots_boyd_` (const `Fourier5` &abcd)

Solve for all roots of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) + E = 0$ -> implements Boyd's method.*
- double `find_root_halley_` (double x0, const `Fourier5` &abcd)

Solve for root of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$ -> implements Halley's method.*
- double `find_gamma_` (const `Fourier5` &abcd, int i, int j, const std::string &opt)

Compute gamma from roots of base equations.
- bool `lt_` (double a, double b)

less-than function
- bool `gt_` (double a, double b)

greater-than function
- std::shared_ptr< psi::Matrix > `psi_X_` ()

Form the Psi4 matrix with the transformation matrix.

Protected Attributes

- const int `n_`

Dimension of the problem.
- const double `conv_`

Convergence.
- const int `maxiter_`

Maximum number of iterations.
- const bool `verbose_`

Verbose mode.
- double * `P_`

P tensor.
- double * `P0_`

Reference P tensor.
- double * `X_`

- *X Matrix (accumulated solution)*
- double * [W_](#)
Work place.
- double * [Xold_](#)
Temporary X matrix.
- double * [Xnew_](#)
Temporary X matrix.
- int [niter_](#)
Current number of iterations.
- double [S_](#) [4]
Current solutions.
- double [Zinit_](#)
Initial Z value.
- double [Zold_](#)
Old Z value.
- double [Znew_](#)
New Z value.
- double [conv_current_](#)
Current convergence.
- bool [success_](#)
Status of optimization.

14.88.1 Detailed Description

The objective function of the orthogonal matrix \mathbf{X}

$$Z(\mathbf{X}) \equiv \sum_{ijk} X_{ji} X_{ki} P_{ijk}$$

is optimized by using the Jacobi iteration algorithm. In the above equation, \mathbf{P} is a general real third-rank tensor of size N^3 . The solver is equivalent to [UnitaryOptimizer_4.2](#) in mathematical sense, in which the sixth-rank tensor is zero, hence costly N^6 memory allocation is avoided.

Algorithm.

Optimization of \mathbf{X} is factorized into a sequence of 2-dimensional rotations with one real parameter γ :

$$\mathbf{X}^{\text{New}} = \mathbf{X}^{\text{Old}} \cdot \mathbf{U}(\gamma)$$

where

$$\mathbf{U}(\gamma) \equiv \begin{pmatrix} \ddots & & & \\ & \cos(\gamma) & \cdots & \sin(\gamma) \\ & \vdots & \ddots & \vdots \\ & -\sin(\gamma) & \cdots & \cos(\gamma) \\ & & & \ddots \end{pmatrix}$$

is the Jacobi transformation matrix constructed for the I th and J th element from the entire N -dimensional set. For the sake of algorithmic simplicity, every iteration after $\mathbf{U}(\gamma)$ has been formed, \mathbf{X}^{Old} is for a while assumed to be an identity matrix and the \mathbf{P} tensor are transformed according to the following formulae

$$P_{ijk} \rightarrow \sum_{j'k'} P_{ij'k'} X_{j'j} X_{k'k}$$

The full transformation matrix is accumulated in the memory buffer until convergence.

In each iteration, the optimum angle γ is found as follows: First, the roots of the finite Fourier series

$$a_0 + \sum_{p=1}^2 \{a_p \cos(px) + b_p \sin(px)\} = 0$$

are found. In the above equations, the expansion coefficients are calculated analytically as a function of I, J - the chosen indices in the Jacobi iteration subspace. The roots are evaluated by applying the Boyd's method[1], in which they are given as

$$\gamma_n = \Re[-i \ln(\lambda_n)]$$

where λ_n is an eigenvalue of the following 4 by 4 complex matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -\frac{a_2+ib_2}{a_2-ib_2} & -\frac{a_1+ib_1}{a_2-ib_2} & -\frac{2a_0}{a_2-ib_2} & -\frac{a_1-ib_1}{a_2-ib_2} \end{pmatrix}$$

Once the four roots of the Fourier series equation are found, one solution out of four is chosen which satisfies the global optimum condition, i.e., the largest increase/decrease in the objective function given by

$$\delta Z = Z(\mathbf{U}(\gamma)) - Z(\mathbf{1})$$

The Hessian is not computed. All the $N(N-1)/2$ unique pairs of molecular orbitals are checked and the optimal set of γ, I, J is chosen to construct \mathbf{X}^{New} .

References:

[1] Boyd, J.P.; J. Eng. Math. (2006) 56, pp. 203-219

14.88.2 Constructor & Destructor Documentation

UnitaryOptimizer_2() [1/2]

```
oepdev::UnitaryOptimizer_2::UnitaryOptimizer_2 (
    double * P,
    int n,
    double conv = 1.0e-6,
    int maxiter = 100,
    bool verbose = true )
```

Parameters

P	- \mathbf{P} tensor (flattened row-wise)
n	- dimensionality of the problem (N)
$conv$	- convergence in the Z function
$maxiter$	- maximum number of iterations
$verbose$	- whether print information of iteration process or not Sets up the optimizer.

UnitaryOptimizer_2() [2/2]

```
oepdev::UnitaryOptimizer_2::UnitaryOptimizer_2 (
    int  $n$ ,
    double  $conv$ ,
    int  $maxiter$ ,
    bool  $verbose$  ) [protected]
```

Parameters

n	- dimensionality of the problem (N)
$conv$	- convergence in the Z function
$maxiter$	- maximum number of iterations
$verbose$	- whether print information of iteration process or not Sets up the optimizer.

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

- oepdev/libutil/[unitary_optimizer.h](#)
- oepdev/libutil/unitary_optimizer.cc

14.89 oepdev::UnitaryOptimizer_2_1 Class Reference

Public Member Functions

- [UnitaryOptimizer_2_1](#) (psi::SharedMatrix P , psi::SharedVector p , double $conv=1.0e-6$, int $maxiter=100$, bool $verbose=true$)
Create from P matrix and p vector and optimization options.
- [~UnitaryOptimizer_2_1](#) ()
Clear memory.
- bool [maximize](#) ()
Run the minimization.
- bool [minimize](#) ()
Run the maximization.

- `psi::SharedMatrix X ()`
Get the unitary matrix (solution)
- `double ** get_X () const`
Get the unitary matrix (pointer to solution)
- `double Z ()`
Get the actual value of Z function.
- `bool success () const`
Get the status of the optimization.

Protected Member Functions

- `UnitaryOptimizer_2_1 (int n, double conv, int maxiter, bool verbose)`
Initialize the basic memory.
- `void common_init_ ()`
Prepare the optimizer.
- `void run_ (const std::string &opt)`
Run the optimization (intermediate interface)
- `void optimize_ (const std::string &opt)`
Run the optimization (inner interface)
- `void refresh_ ()`
Restore the initial state of the optimizer.
- `void update_conv_ ()`
Update the convergence.
- `void update_iter_ ()`
Update the iterates.
- `void update_Z_ ()`
Update Z value.
- `void update_P_ ()`
Uptade P tensor.
- `void update_X_ ()`
Update the solution matrix X.
- `double eval_Z_ (psi::SharedMatrix X, psi::SharedMatrix P)`
Evaluate the objective Z function.
- `double eval_Z_ ()`
- `double eval_dZ_ (double g, psi::SharedMatrix, int I, int J)`
Evaluate the change in Z.
- `double eval_Z_trial_ (int I, int J, double gamma)`
Evaluate the trial Z value.
- `void form_X0_ ()`
Create identity matrix.

- void [form_X_](#) (int I, int J, double gamma)
Form unitary matrix X (store in buffer Xnew_)
- void [form_next_X_](#) (const std::string &opt)
Form the next unitary matrix X.
- [Fourier5 get_fourier_](#) (int I, int J)
Retrieve [ABCD](#) parameters for root search.
- void [find_roots_boyd_](#) (const [Fourier5](#) &abcd)
Solve for all roots of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) + E = 0$ -> implements Boyd's method.*
- double [find_root_halley_](#) (double x0, const [Fourier5](#) &abcd)
Solve for root of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$ -> implements Halley's method.*
- double [find_gamma_](#) (const [Fourier5](#) &abcd, int i, int j, const std::string &opt)
Compute gamma from roots of base equations.
- bool [lt_](#) (double a, double b)
less-than function
- bool [gt_](#) (double a, double b)
greater-than function
- psi::SharedMatrix [psi_X_](#) ()
Form the Psi4 matrix with the transformation matrix.

Protected Attributes

- const int [n_](#)
Dimension of the problem.
- const double [conv_](#)
Convergence.
- const int [maxiter_](#)
Maximum number of iterations.
- const bool [verbose_](#)
Verbose mode.
- psi::SharedMatrix [P_](#)
P tensor.
- psi::SharedMatrix [P0_](#)
Reference P tensor.
- psi::SharedVector [p_](#)
p vector
- psi::SharedMatrix [X_](#)
X Matrix (accumulated solution)
- psi::SharedMatrix [W_](#)
Work place 1.

- `psi::SharedMatrix Y_`
Work place 2.
- `psi::SharedMatrix Xold_`
Temporary X matrix.
- `psi::SharedMatrix Xnew_`
Temporary X matrix.
- `int niter_`
Current number of iterations.
- `double S_ [4]`
Current solutions.
- `double Zinit_`
Initial Z value.
- `double Zold_`
Old Z value.
- `double Znew_`
New Z value.
- `double conv_current_`
Current convergence.
- `bool success_`
Status of optimization.

14.89.1 Constructor & Destructor Documentation

UnitaryOptimizer_2_1() [1/2]

```
oepdev::UnitaryOptimizer_2_1::UnitaryOptimizer_2_1 (
    psi::SharedMatrix P,
    psi::SharedVector p,
    double conv = 1.0e-6,
    int maxiter = 100,
    bool verbose = true )
```

Parameters

<i>P</i>	- P matrix
<i>p</i>	- p vector
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

UnitaryOptimizer_2_1() [2/2]

```
oepdev::UnitaryOptimizer_2_1::UnitaryOptimizer_2_1 (
    int n,
    double conv,
    int maxiter,
    bool verbose ) [protected]
```

Parameters

<i>n</i>	- dimensionality of the problem (<i>N</i>)
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

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

- oepdev/libutil/[unitary_optimizer.h](#)
- oepdev/libutil/unitary_optimizer.cc

14.90 oepdev::UnitaryOptimizer_4_2 Class Reference

Find the optimim unitary matrix for quartic-quadratic matrix equation with trace.

```
#include <unitary_optimizer.h>
```

Public Member Functions

- [UnitaryOptimizer_4_2](#) (double *R, double *P, int n, double conv=1.0e-6, int maxiter=100, bool verbose=true)
Create from R and P matrices and optimization options.
- [~UnitaryOptimizer_4_2](#) ()
Clear memory.
- bool [maximize](#) ()
Run the minimization.
- bool [minimize](#) ()
Run the maximization.
- std::shared_ptr< psi::Matrix > [X](#) ()
Get the unitary matrix (solution)
- double * [get_X](#) () const
Get the unitary matrix (pointer to solution)

- double [Z](#) ()
Get the actual value of Z function.
- bool [success](#) () const
Get the status of the optimization.

Protected Member Functions

- [UnitaryOptimizer_4_2](#) (int n, double conv, int maxiter, bool verbose)
Initialize the basic memory.
- void [common_init](#) ()
Prepare the optimizer.
- void [run](#) (const std::string &opt)
Run the optimization (intermediate interface)
- void [optimize](#) (const std::string &opt)
Run the optimization (inner interface)
- void [refresh](#) ()
Restore the initial state of the optimizer.
- void [update_conv](#) ()
Update the convergence.
- void [update_iter](#) ()
Update the iterates.
- void [update_Z](#) ()
Update Z value.
- void [update_RP](#) ()
Uptade R and P matrices.
- void [update_X](#) ()
Update the solution matrix X.
- double [eval_Z](#) (double *X, double *R, double *P)
Evaluate the objective Z function.
- double [eval_Z](#) ()
- double [eval_dZ](#) (double g, double *R, double *P, int I, int J)
Evaluate the change in Z.
- double [eval_Z_trial](#) (int I, int J, double gamma)
Evaluate the trial Z value.
- void [form_X0](#) ()
Create identity matrix.
- void [form_X](#) (int I, int J, double gamma)
Form unitary matrix X (store in buffer Xnew.)
- void [form_next_X](#) (const std::string &opt)
Form the next unitary matrix X.

- [Fourier9 get_fourier_](#) (int I, int J)
Retrieve ABCD parameters for root search.
- void [find_roots_boyd_](#) (const [Fourier9](#) &abcd)
Solve for all roots of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$ -> implements Boyd's method.*
- double [find_root_halley_](#) (double x0, const [Fourier9](#) &abcd)
Solve for root of equation $A\sin(g) + B*\cos(g) + C*\sin(2*g) + D*\cos(2*g) = 0$ -> implements Halley's method.*
- double [find_gamma_](#) (const [Fourier9](#) &abcd, int i, int j, const std::string &opt)
Compute gamma from roots of base equations.
- bool [lt_](#) (double a, double b)
less-than function
- bool [gt_](#) (double a, double b)
greater-than function
- std::shared_ptr< psi::Matrix > [psi_X_](#) ()
Form the Psi4 matrix with the transformation matrix.

Protected Attributes

- const int [n_](#)
Dimension of the problem.
- const double [conv_](#)
Convergence.
- const int [maxiter_](#)
Maximum number of iterations.
- const bool [verbose_](#)
Verbose mode.
- double * [R_](#)
R tensor.
- double * [P_](#)
P tensor.
- double * [R0_](#)
Reference R tensor.
- double * [P0_](#)
Reference P tensor.
- double * [X_](#)
X Matrix (accumulated solution)
- double * [W_](#)
Work place.
- double * [Xold_](#)
Temporary X matrix.

- double * `Xnew_`
Temporary X matrix.
- int `niter_`
Current number of iterations.
- double `S_` [8]
Current solutions.
- double `Zinit_`
Initial Z value.
- double `Zold_`
Old Z value.
- double `Znew_`
New Z value.
- double `conv_current_`
Current convergence.
- bool `success_`
Status of optimization.

14.90.1 Detailed Description

The objective function of the orthogonal matrix \mathbf{X}

$$Z(\mathbf{X}) \equiv \sum_{ijklmn} X_{ki} X_{lj} X_{mi} X_{nj} R_{ijklmn} + \sum_{ijk} X_{ji} X_{ki} P_{ijk}$$

is optimized by using the Jacobi iteration algorithm. In the above equation, \mathbf{R} is a general real sixth-rank tensor of size N^6 whereas \mathbf{P} is a general real third-rank tensor of size N^3 .

Algorithm.

Optimization of \mathbf{X} is factorized into a sequence of 2-dimensional rotations with one real parameter γ :

$$\mathbf{X}^{\text{New}} = \mathbf{X}^{\text{Old}} \cdot \mathbf{U}(\gamma)$$

where

$$\mathbf{U}(\gamma) \equiv \begin{pmatrix} \ddots & & & & \\ & \cos(\gamma) & \cdots & \sin(\gamma) & \\ & \vdots & \ddots & \vdots & \\ & -\sin(\gamma) & \cdots & \cos(\gamma) & \\ & & & & \ddots \end{pmatrix}$$

is the Jacobi transformation matrix constructed for the I th and J th element from the entire N -dimensional set. For the sake of algorithmic simplicity, every iteration after $\mathbf{U}(\gamma)$ has been formed,

\mathbf{X}^{Old} is for a while assumed to be an identity matrix and the \mathbf{R} as well as \mathbf{P} tensors are transformed according to the following formulae

$$R_{ijklmn} \rightarrow \sum_{k'l'm'n'} R_{ijk'l'm'n'} X_{k'k} X_{l'l} X_{m'm} X_{n'n}$$

$$P_{ijk} \rightarrow \sum_{j'k'} P_{ij'k'} X_{j'j} X_{k'k}$$

The full transformation matrix is accumulated in the memory buffer until convergence.

In each iteration, the optimum angle γ is found as follows: First, the roots of the finite Fourier series

$$a_0 + \sum_{p=1}^4 \{a_p \cos(px) + b_p \sin(px)\} = 0$$

are found. In the above equations, the expansion coefficients are calculated analytically as a function of I, J - the chosen indices in the Jacobi iteration subspace. The roots are evaluated by applying the Boyd's method[1], in which they are given as

$$\gamma_n = \Re[-i \ln(\lambda_n)]$$

where λ_n is an eigenvalue of the following 8 by 8 complex matrix:

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -\frac{a_4+ib_4}{a_4-ib_4} & -\frac{a_3+ib_3}{a_4-ib_4} & -\frac{a_2+ib_2}{a_4-ib_4} & -\frac{a_1+ib_1}{a_4-ib_4} & -\frac{2a_0}{a_4-ib_4} & -\frac{a_1-ib_1}{a_4-ib_4} & -\frac{a_2-ib_2}{a_4-ib_4} & -\frac{a_3-ib_3}{a_4-ib_4} \end{pmatrix}$$

Once the eight roots of the Fourier series equation are found, one solution out of eight is chosen which satisfies the global optimum condition, i.e., the largest increase/decrease in the objective function given by

$$\delta Z = Z(\mathbf{U}(\gamma)) - Z(\mathbf{1})$$

The Hessian is not computed. All the $N(N-1)/2$ unique pairs of molecular orbitals are checked and the optimal set of γ, I, J is chosen to construct \mathbf{X}^{New} .

References:

[1] Boyd, J.P.; J. Eng. Math. (2006) 56, pp. 203-219

14.90.2 Constructor & Destructor Documentation

UnitaryOptimizer_4_2() [1/2]

```
oepdev::UnitaryOptimizer_4_2::UnitaryOptimizer_4_2 (
    double * R,
    double * P,
    int n,
    double conv = 1.0e-6,
    int maxiter = 100,
    bool verbose = true )
```

Parameters

<i>R</i>	- R tensor (flattened row-wise)
<i>P</i>	- P tensor (flattened row-wise)
<i>n</i>	- dimensionality of the problem (<i>N</i>)
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

UnitaryOptimizer_4_2() [2/2]

```
oepdev::UnitaryOptimizer_4_2::UnitaryOptimizer_4_2 (
    int n,
    double conv,
    int maxiter,
    bool verbose ) [protected]
```

Parameters

<i>n</i>	- dimensionality of the problem (<i>N</i>)
<i>conv</i>	- convergence in the <i>Z</i> function
<i>maxiter</i>	- maximum number of iterations
<i>verbose</i>	- whether print information of iteration process or not Sets up the optimizer.

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

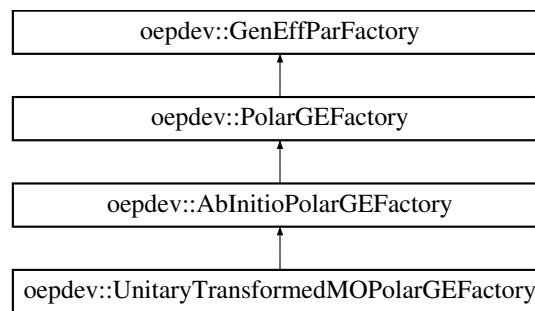
- [oepdev/libutil/unitary_optimizer.h](#)
- [oepdev/libutil/unitary_optimizer.cc](#)

14.91 oepdev::UnitaryTransformedMOPolarGEFactory Class Reference

Polarization GEFP Factory with Least-Squares Scaling of MO Space.

```
#include <gefp.h>
```

Inheritance diagram for oepdev::UnitaryTransformedMOPolarGEFactory:



Public Member Functions

- [UnitaryTransformedMOPolarGEFactory](#) (std::shared_ptr< psi::Wavefunction > [wfn](#), psi::Options &opt)
Construct from [CPHF](#) object and Psi4 options.
- virtual [~UnitaryTransformedMOPolarGEFactory](#) ()
Destruct.
- std::shared_ptr< [GenEffPar](#) > [compute](#) (void)
Perform Least-Squares Fit.

Additional Inherited Members

14.91.1 Detailed Description

Implements creation of the density matrix susceptibility tensors for which $\mathbf{X} \neq \mathbf{1}$. Guarantees the idempotency of the density matrix up to first-order in LCAO-MO variation.

Note

This method does not give better results than the $X=1$ method and is extremely time and memory consuming. Therefore, it is placed here only for future reference about solving unitary optimization problem in case it occurs.

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

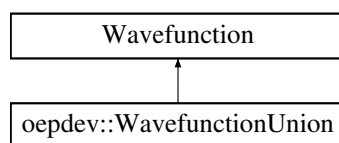
- oepdev/libgefp/[gefp.h](#)
- oepdev/libgefp/gefp_polar_abinitio.cc

14.92 oepdev::WavefunctionUnion Class Reference

Union of two Wavefunction objects.

```
#include <wavefunction.union.h>
```

Inheritance diagram for oepdev::WavefunctionUnion:



Public Member Functions

- [WavefunctionUnion](#) (SharedWavefunction ref_wfn, Options &options)

Constructor.

- [WavefunctionUnion](#) (SharedMolecule dimer, SharedBasisSet [primary](#), SharedBasisSet auxiliary_df, SharedBasisSet guess, SharedBasisSet primary_1, SharedBasisSet primary_2, SharedBasisSet auxiliary_1, SharedBasisSet auxiliary_2, SharedBasisSet auxiliary_df_1, SharedBasisSet auxiliary_df_2, SharedBasisSet intermediate_1, SharedBasisSet intermediate_2, SharedBasisSet guess_1, SharedBasisSet guess_2, SharedWavefunction wfn_1, SharedWavefunction wfn_2, Options &options)

Constructor.

- virtual [~WavefunctionUnion](#) ()

Destructor.

- virtual double [compute_energy](#) ()

Compute Energy (now blank)

- virtual double [nuclear_repulsion_interaction_energy](#) ()

Compute Nuclear Repulsion Energy between unions.

- void [localize_orbitals](#) ()

Localize Molecular Orbitals.

- void [transform_integrals](#) ()

Transform Integrals (2- and 4-index transformations)

- void [clear_dpd](#) ()

Close the DPD instance.

- int [l_nmo](#) (int n) const

*Get number of molecular orbitals of the *n*th fragment.*

- int [l_nso](#) (int n) const

*Get number of symmetry orbitals of the *n*th fragment.*

- int [l_ndocc](#) (int n) const

*Get number of doubly occupied orbitals of the *n*th fragment.*

- int [l_nvir](#) (int n) const

*Get number of virtual orbitals of the *n*th fragment.*

- int [l_nalpha](#) (int n) const

*Get the number of the alpha electrons of the *n*th fragment.*

- int [l_nbeta](#) (int n) const

- Get the number of the beta electrons of the *n*th fragment.*

 - int [L_nbf](#) (int n) const
- Get number of basis functions of the *n*th fragment.*

 - int [L_noffs_ao](#) (int n) const
- Get the basis set offset of the *n*th fragment.*

 - double [L_energy](#) (int n) const
- Get the reference energy of the *n*th fragment.*

 - SharedMolecule [L_molecule](#) (int n) const
- Get the molecule object of the *n*th fragment.*

 - SharedBasisSet [L_primary](#) (int n) const
- Get the primary basis set object of the *n*th fragment.*

 - SharedBasisSet [L_auxiliary](#) (int n) const
- Get the auxiliary basis set object of the *n*th fragment.*

 - SharedBasisSet [L_intermediate](#) (int n) const
- Get the intermediate basis set object of the *n*th fragment.*

 - SharedBasisSet [L_guess](#) (int n) const
- Get the guess basis set object of the *n*th fragment.*

 - SharedWavefunction [L_wfn](#) (int n) const
- Get the wavefunction object of the *n*th fragment.*

 - SharedMOSpace [L_mospace](#) (int n, const std::string &label) const
- Get the MO space named label (either OCC or VIR) of the *n*th fragment.*

 - SharedLocalizer [L_localizer](#) (int n) const
- Get the orbital localizer object of the *n*th fragment.*

 - psi::SharedMatrix [L_ca_occ](#) (int n) const
- Get the occupied molecular orbitals of the *n*th fragment.*

 - psi::SharedMatrix [L_ca_vir](#) (int n) const
- Get the virtual molecular orbitals of the *n*th fragment.*

 - psi::SharedVector [L_eps_a_occ](#) (int n) const
- Get the occupied molecular orbital energies of the *n*th fragment.*

 - psi::SharedVector [L_eps_a_vir](#) (int n) const
- Get the virtual molecular orbital energies of the *n*th fragment.*

 - SharedIntegralTransform [integrals](#) (void) const
- Get the integral transform object of the entire union.*

 - bool [has_localized_orbitals](#) (void) const
- If union got its molecular orbital localized or not.*

 - SharedBasisSet [primary](#) (void) const
- Get the primary basis set for the entire union.*

 - SharedMOSpace [mospace](#) (const std::string &label) const
- Get the MO space named label (either OCC or VIR)*

 - SharedMatrix [Ca_subset](#) (const std::string &basis="SO", const std::string &subset="ALL")

- SharedMatrix [Cb_subset](#) (const std::string &basis="SO", const std::string &subset="ALL")
- SharedMatrix [C_subset_helper](#) (SharedMatrix C, const Dimension &noccp, SharedVector epsilon, const std::string &basis, const std::string &subset)

Helpers for Ca_ and Cb_ matrix transformers.

- SharedVector [epsilon_subset_helper](#) (SharedVector epsilon, const Dimension &noccp, const std::string &basis, const std::string &subset)

Helper for epsilon transformer.

- void [print_header](#) (void)

Print information about this wavefunction union.

- void [print_mo_integrals](#) (void)

Print the MO integrals.

Protected Attributes

- int [nIsolatedMolecules_](#)

Number of isolated molecules.

- SharedWavefunction [dimer_wavefunction_](#)

The wavefunction for a dimer (electrons relaxed in the field of monomers)

- SharedIntegralTransform [integrals_](#)

Integral transform object (2- and 4-index transformations)

- bool [hasLocalizedOrbitals_](#)

whether orbitals of the union were localized (or not)

- std::map< const std::string, SharedMOSpace > [mospacesUnion_](#)

Dictionary of MO spaces for the entire union (OCC and VIR)

- std::vector< SharedMolecule > [l_molecule_](#)

List of molecules.

- std::vector< SharedBasisSet > [l_primary_](#)

List of primary basis functions per molecule.

- std::vector< SharedBasisSet > [l_auxiliary_](#)

List of auxiliary basis functions per molecule.

- std::vector< SharedBasisSet > [l_intermediate_](#)

List of intermediate basis functions per molecule.

- std::vector< SharedBasisSet > [l_guess_](#)

List of guess basis functions per molecule.

- std::vector< SharedWavefunction > [l_wfn_](#)

List of original isolated wavefunctions (electrons unrelaxed)

- std::vector< std::string > [l_name_](#)

List of names of isolated wavefunctions.

- std::vector< int > [l_nbf_](#)

List of basis function numbers per molecule.

- std::vector< int > [l_nmo_](#)

List of numbers of molecular orbitals (MO's) per molecule.

- `std::vector< int > l_nso_`

List of numbers of SO's per molecule.

- `std::vector< int > l_ndocc_`

List of numbers of doubly occupied orbitals per molecule.

- `std::vector< int > l_nvir_`

List of numbers of virtual orbitals per molecule.

- `std::vector< int > l_noffs_ao_`

List of basis set offsets per molecule.

- `std::vector< double > l_energy_`

List of energies of isolated wavefunctions.

- `std::vector< double > l_efzc_`

List of frozen-core energies per isolated wavefunction.

- `std::vector< bool > l_density_fitted_`

List of information per wfn whether it was obtained using DF or not.

- `std::vector< int > l_nalpha_`

List of numbers of alpha electrons per isolated wavefunction.

- `std::vector< int > l_nbeta_`

List of numbers of beta electrons per isolated wavefunction.

- `std::vector< int > l_nfrzc_`

List of numbers of frozen-core orbitals per isolated molecule.

- `std::vector< psi::SharedMatrix > l_ca_occ_`

List of occupied orbitals.

- `std::vector< psi::SharedMatrix > l_ca_vir_`

List of virtual orbitals.

- `std::vector< psi::SharedVector > l_eps_a_occ_`

List of occupied orbital energies.

- `std::vector< psi::SharedVector > l_eps_a_vir_`

List of virtual orbital energies.

- `std::vector< SharedLocalizer > l_localizer_`

List of orbital localizers.

- `std::vector< std::map< const std::string, SharedMOSpace > > l_mospace_`

List of dictionaries of MO spaces.

- `std::shared_ptr< psi::OEProp > oeprop_`

One-Electron Property.

14.92.1 Detailed Description

The [WavefunctionUnion](#) is the union of two unperturbed Wavefunctions.

Notes:

1. Works only for C1 symmetry! Therefore `this->nirrep() = 1`.
 2. Does not set `reference_wavefunction_`
 3. Sets `oeprop_` for the union of uncoupled molecules
-
1. Performs Hadamard sums on `H_`, `Fa_`, `Da_`, `Ca_` and `S_` based on uncoupled wavefunctions.
 2. Since it is based on shallow copy of the original Wavefunction, it **changes** contents of this wavefunction. Reallocate and copy if you want to keep the original wavefunction.

Warnings:

1. Gradients, Hessians and frequencies are not touched, hence they are **wrong**!
2. Lagrangian (if present) is not touched, hence its **wrong**!
3. `Ca/Cb` and `epsilon` subsets were reimplemented from `psi::Wavefunction` to remove sorting of orbitals. However, the corresponding member functions are not virtual in `psi::Wavefunction`. This could bring problems when upcasting.

The following variables are *shallow* copies of variables inside the Wavefunction object, that is created for the *whole* molecule cluster:

- `basissets_` (DF/RI/F12/etc basis sets)
- `basisset_` (ORBITAL basis set)
- `sobasisset_` (Primary basis set for SO integrals)
- `AO2SO_` (AO2SO conversion matrix (AO in rows, SO in cols))
- `molecule_` (Molecule that this wavefunction is run on)
- `options_` (Options object)
- `psio_` (PSI file access variables)
- `integral_` (Integral factory)
- `factory_` (Matrix factory for creating standard sized matrices)
- `memory_` (How much memory you have access to)
- `nalpha_, nbeta_` (Total alpha and beta electrons)
- `nfrzc_` (Total frozen core orbitals)

- `doccpi_` (Number of doubly occupied per irrep)
- `soccpi_` (Number of singly occupied per irrep)
- `frzcpi_` (Number of frozen core per irrep)
- `frzvp_` (Number of frozen virtuals per irrep)
- `nalphapi_` (Number of alpha electrons per irrep)
- `nbetapi_` (Number of beta electrons per irrep)
- `nsopi_` (Number of so per irrep)
- `nmopi_` (Number of mo per irrep)
- `nso_` (Total number of SOs)
- `nmo_` (Total number of MOs)
- `nirrep_` (Number of irreps; must be equal to 1 due to symmetry reasons)
- `same_a_b_dens_` and `same_a_b_orbs_` The rest is altered so that the Wavefunction parameters reflect a cluster of non-interacting (uncoupled, isolated, unrelaxed) molecular electron densities.

14.92.2 Constructor & Destructor Documentation

WavefunctionUnion() [1/2]

```
oepdev::WavefunctionUnion::WavefunctionUnion (
    SharedWavefunction ref_wfn,
    Options & options )
```

Provide wavefunction with molecule containing at least 2 fragments.

Parameters

<i>ref_wfn</i>	- reference wavefunction
<i>options</i>	- Psi4 options

This constructor is used for C++ internal interface.

WavefunctionUnion() [2/2]

```
oepdev::WavefunctionUnion::WavefunctionUnion (
    SharedMolecule dimer,
    SharedBasisSet primary,
```

```

SharedBasisSet auxiliary_df,
SharedBasisSet guess,
SharedBasisSet primary_1,
SharedBasisSet primary_2,
SharedBasisSet auxiliary_1,
SharedBasisSet auxiliary_2,
SharedBasisSet auxiliary_df_1,
SharedBasisSet auxiliary_df_2,
SharedBasisSet intermediate_1,
SharedBasisSet intermediate_2,
SharedBasisSet guess_1,
SharedBasisSet guess_2,
SharedWavefunction wfn_1,
SharedWavefunction wfn_2,
Options & options )

```

Provide molecule dimer and all the required monomer basis sets and wavefunctions.

Parameters

<i>dimer</i>	- molecule object
<i>primary</i>	- basis set object: dimer (primary)
<i>auxiliary_df</i>	- basis set object: dimer (DF SCF)
<i>guess</i>	- basis set object: dimer (guess)
<i>primary_1</i>	- basis set object for 1st monomer
<i>primary_2</i>	- basis set object for 2nd monomer
<i>auxiliary_1</i>	- basis set object for 1st monomer
<i>auxiliary_2</i>	- basis set object for 2nd monomer
<i>auxiliary_df_1</i>	- basis set object for 1st monomer
<i>auxiliary_df_2</i>	- basis set object for 2nd monomer
<i>intermediate_1</i>	- basis set object for 1st monomer
<i>intermediate_2</i>	- basis set object for 2nd monomer
<i>guess_1</i>	- basis set object for 1st monomer
<i>guess_2</i>	- basis set object for 2nd monomer
<i>wfn_1</i>	- unperturbed wavefunction object
<i>wfn_2</i>	- unperturbed wavefunction object
<i>options</i>	- Psi4 options

This constructor is for interface with Python level.

14.92.3 Member Function Documentation

Ca_subset()

```
SharedMatrix oepdev::WavefunctionUnion::Ca_subset (
    const std::string & basis = "SO",
    const std::string & subset = "ALL" )
```

Return a subset of the Ca matrix in a desired basis

Parameters

<i>basis</i>	the symmetry basis to use AO, SO
<i>subset</i>	the subset of orbitals to return ALL, ACTIVE, FROZEN, OCC, VIR, FROZEN_OCC, ACTIVE_OCC, ACTIVE_VIR, FROZEN_VIR

Returns

the matrix in Pitzer order in the desired basis

Cb_subset()

```
SharedMatrix oepdev::WavefunctionUnion::Cb_subset (
    const std::string & basis = "SO",
    const std::string & subset = "ALL" )
```

Return a subset of the Cb matrix in a desired basis

Parameters

<i>basis</i>	the symmetry basis to use AO, SO
<i>subset</i>	the subset of orbitals to return ALL, ACTIVE, FROZEN, OCC, VIR, FROZEN_OCC, ACTIVE_OCC, ACTIVE_VIR, FROZEN_VIR

Returns

the matrix in Pitzer order in the desired basis

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

- oepdev/libutil/[wavefunction_union.h](#)
- oepdev/libutil/wavefunction_union.cc

15.1 include/oepdev_files.h File Reference

Macros

- #define `OEPDEV_USE_PSI4_DIIS_MANAGER` 0
Use DIIS from Psi4 (1) or OEPDev (0)?
- #define `OEPDEV_MAX_AM` 8
L_{max}.
- #define `OEPDEV_N_MAX_AM` 17
2L_{max}+1
- #define `OEPDEV_CRIT_ERI` 1e-9
*ERI criterion for E12, E34, E123 and lambda*EXY coefficients.*
- #define `OEPDEV_SIZE_BUFFER_R` 250563
*Size of R buffer (OEPDEV_N_MAX_AM*OEPDEV_N_MAX_AM*OEPDEV_N_MAX_AM*OEPDEV_N_MAX_AM*3)*
- #define `OEPDEV_SIZE_BUFFER_D2` 3264
Size of D2 buffer (3(OEPDEV_MAX_AM+1)*(OEPDEV_MAX_AM+1)*OEPDEV_N_MAX_AM)*
- #define `OEPDEV_AU_KcalPerMole` 627.509
Energy converters.
- #define `OEPDEV_AU_CMRec` 219474.63
- #define `OEPDEV_AU_EV` 27.21138

15.2 include/oepdev_options.h File Reference

Namespaces

- [psi](#)

Psi4 package namespace.

Functions

- PSI API int [psi::read_options](#) (std::string name, Options &options)

Options for the OEPDev plugin.

15.3 main.cc File Reference

```
#include <string>
#include "include/oepdev_files.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libmints/wavefunction.h"
#include "include/oepdev_options.h"
#include "oepdev/liboep/oep.h"
#include "oepdev/libgefp/gefp.h"
#include "oepdev/libsolver/solver.h"
#include "oepdev/libtest/test.h"
#include <pybind11/pybind11.h>
```

Namespaces

- [psi](#)

Psi4 package namespace.

Typedefs

- using **SharedWavefunction** = std::shared_ptr< psi::Wavefunction >
- using **SharedUnion** = std::shared_ptr< [oepdev::WavefunctionUnion](#) >
- using **SharedOEPotential** = std::shared_ptr< [oepdev::OEPotential](#) >
- using **SharedGEFPFactory** = std::shared_ptr< [oepdev::GenEffParFactory](#) >
- using **SharedGEFPParameters** = std::shared_ptr< [oepdev::GenEffPar](#) >

Functions

- void **psi::export_dmtf** (py::module &)
- void **psi::export_cphf** (py::module &)
- void **psi::export_solver** (py::module &)

- void **psi::export_util** (py::module &)
- void **psi::export_oep** (py::module &)
- void **psi::export_gefp** (py::module &)
- PSI_API SharedWavefunction **psi::oepdev** (SharedWavefunction ref_wfn, Options &options)

Main routine of the OEPCDev plugin.

- **psi::PYBIND11_MODULE** (oepdev, m)

15.4 oepdev/lib3d/dmtp.h File Reference

```
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
```

Classes

- class **oepdev::MultipoleConvergence**
Multipole Convergence.
- class **oepdev::DMTPole**
Distributed Multipole Analysis Container and Computer. Abstract Base.
- class **oepdev::CAMM**
Cumulative Atomic Multipole Moments.

Namespaces

- **psi**
Psi4 package namespace.
- **oepdev**
OEPCDev module namespace.

Typedefs

- using **psi::SharedBasisSet** = std::shared_ptr< BasisSet >
- using **oepdev::SharedDMTPole** = std::shared_ptr< DMTPole >
DMTPole object.

15.5 oepdev/lib3d/esp.h File Reference

```
#include "space3d.h"
```

Classes

- class [oepdev::ESPSolver](#)
Charges from Electrostatic Potential (ESP). A solver-type class.

Namespaces

- [oepdev](#)
OEPEv module namespace.

Typedefs

- using [oepdev::SharedField3D](#) = std::shared_ptr< [oepdev::Field3D](#) >

15.6 oepdev/libgefp/gefp.h File Reference

```
#include <vector>
#include <string>
#include <random>
#include <cmath>
#include <map>
#include "psi4/libpsi4util/PsiOutStream.h"
#include "psi4/libpsi4util/process.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/vector3.h"
#include "../liboep/oep.h"
#include "../libutil/util.h"
#include "../libutil/cphf.h"
#include "../libutil/scf_perturb.h"
#include "../libutil/quambo.h"
#include "../libpsi/integral.h"
```

Classes

- class [oepdev::GenEffPar](#)

- Generalized Effective Fragment Parameters. Container Class.*
- class [oepdev::GenEffFrag](#)
- Generalized Effective Fragment. Container Class.*
- class [oepdev::GenEffParFactory](#)
- Generalized Effective Fragment Factory. Abstract Base.*
- class [oepdev::EFP2_GEFactory](#)
- EFP2 GEFP Factory.*
- class [oepdev::OEP_EFP2_GEFactory](#)
- OEP-EFP2 GEFP Factory.*
- class [oepdev::PolarGEFactory](#)
- Polarization GEFP Factory. Abstract Base.*
- class [oepdev::AbInitioPolarGEFactory](#)
- Polarization GEFP Factory from First Principles. Hartree-Fock Approximation.*
- class [oepdev::FFAbInitioPolarGEFactory](#)
- Polarization GEFP Factory from First Principles: Finite-Difference Model. Arbitrary level of theory.*
- class [oepdev::GeneralizedPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- struct [oepdev::GeneralizedPolarGEFactory::StatisticalSet](#)
- A structure to handle statistical data.*
- class [oepdev::UniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::NonUniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::LinearUniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::QuadraticUniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::LinearNonUniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::QuadraticNonUniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::LinearGradientNonUniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Parameterization.*
- class [oepdev::UnitaryTransformedMOPolarGEFactory](#)
- Polarization GEFP Factory with Least-Squares Scaling of MO Space.*
- class [oepdev::FragmentedSystem](#)
- Molecular System for Fragment-Based Calculations.*

Namespaces

- [oepdev](#)

OEPPDev module namespace.

Typedefs

- using [oepdev::SharedOEPotential](#) = std::shared_ptr< OEPotential >
- using [oepdev::SharedGenEffPar](#) = std::shared_ptr< GenEffPar >
GEFP Parameters container.
- using [oepdev::SharedGenEffParFactory](#) = std::shared_ptr< GenEffParFactory >
GEFP Parameter factory.
- using [oepdev::SharedGenEffFrag](#) = std::shared_ptr< GenEffFrag >
GEFP Fragment container.
- using [oepdev::SharedFragmentedSystem](#) = std::shared_ptr< FragmentedSystem >
Fragmented system.

15.7 oepdev/libints/eri.h File Reference

```
#include "psi4/libpsi4util/exception.h"
#include "psi4/libmints/integral.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/fjt.h"
#include "../libpsi/integral.h"
#include "recurr.h"
```

Classes

- class [oepdev::TwoElectronInt](#)
General Two Electron Integral.
- class [oepdev::ERI_1_1](#)
2-centre ERI of the form $(a|O(2)|b)$ where $O(2) = 1/r_{12}$.
- class [oepdev::ERI_2_2](#)
4-centre ERI of the form $(ab|O(2)|cd)$ where $O(2) = 1/r_{12}$.
- class [oepdev::ERI_3_1](#)
4-centre ERI of the form $(abc|O(2)|d)$ where $O(2) = 1/r_{12}$.

Namespaces

- [oepdev](#)

OEPPDev module namespace.

15.8 oepdev/libints/recur.h File Reference

Namespaces

- [oepdev](#)
OEPEv module namespace.

Macros

- `#define D1_INDEX(x, i, n) ((81*(x))+(9*(i))+(n))`
Get the index of McMurchie-Davidson-Hermite D1 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momentum i of function 1, and the Hermite index n .
- `#define D2_INDEX(x, i, j, n) ((1377*(x))+(153*(i))+(17*(j))+(n))`
Get the index of McMurchie-Davidson-Hermite D2 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j of function 1 and 2, and the Hermite index n .
- `#define D3_INDEX(x, i, j, k, n) ((18225*(x))+(2025*(i))+(225*(j))+(25*(k))+(n))`
Get the index of McMurchie-Davidson-Hermite D3 coefficient stored in the `mdh_buffer_`, that is attributed to the x Cartesian coordinate from angular momenta i, j and k of function 1, 2 and 3, and the Hermite index n .
- `#define R_INDEX(n, l, m, j) ((14739*(n))+(867*(l))+(51*(m))+(j))`
Get the index of McMurchie-Davidson R coefficient stored in the `mdh_buffer_R_` from angular momenta n, l and m and the Boys index j .

Functions

- `double oepdev::d_N_n1_n2 (int N, int n1, int n2, double PA, double PB, double aP)`
Compute McMurchie-Davidson-Hermite (MDH) coefficient for binomial expansion.
- `void oepdev::make_mdh_D1_coeff (int n1, double aPd, double *buffer)`
Compute the McMurchie-Davidson-Hermite coefficients for monomial expansion.
- `void oepdev::make_mdh_D2_coeff (int n1, int n2, double aPd, double *PA, double *PB, double *buffer)`
Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion.
- `void oepdev::make_mdh_D3_coeff (int n1, int n2, int n3, double aPd, double *PA, double *PB, double *PC, double *buffer)`
Compute the McMurchie-Davidson-Hermite coefficients for trinomial expansion.
- `void oepdev::make_mdh_D2_coeff_explicit_recursion (int n1, int n2, double aP, double *PA, double *PB, double *buffer)`
Compute the McMurchie-Davidson-Hermite coefficients for binomial expansion by explicit recursion. This function makes the same changes to buffers as `oepdev::make_mdh_D2_coeff`, but implements it through explicit recursion by calls to `oepdev::d_N_n1_n2`. Therefore, it is slightly slower. Here for debugging purposes.

- void [oepdev::make_mdh_R_coeff](#) (int N, int L, int M, double alpha, double a, double b, double c, double *F, double *buffer)

Compute the McMurchie-Davidson R coefficients.

15.9 oepdev/liboep/oep.h File Reference

```
#include <cstdio>
#include <string>
#include <vector>
#include <map>
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsi4util/PsiOutputStream.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/local.h"
#include "../libutil/cis.h"
#include "../libpsi/integral.h"
#include "../libpsi/potential.h"
#include "../lib3d/space3d.h"
#include "../lib3d/dmtp.h"
```

Classes

- struct [oepdev::OEType](#)
Container to handle the type of One-Electron Potentials.
- class [oepdev::OEPotential](#)
Generalized One-Electron Potential: Abstract base.
- class [oepdev::ElectrostaticEnergyOEPotential](#)
Generalized One-Electron Potential for Electrostatic Energy.
- class [oepdev::RepulsionEnergyOEPotential](#)
Generalized One-Electron Potential for Pauli Repulsion Energy.
- class [oepdev::ChargeTransferEnergyOEPotential](#)
Generalized One-Electron Potential for Charge-Transfer Interaction Energy.
- class [oepdev::EETCouplingOEPotential](#)
Generalized One-Electron Potential for EET coupling calculations.

Namespaces

- [oepdev](#)
OEPDev module namespace.

Typedefs

- using **oepdev::SharedWavefunction** = std::shared_ptr< Wavefunction >
- using **oepdev::SharedBasisSet** = std::shared_ptr< BasisSet >
- using **oepdev::SharedMatrix** = std::shared_ptr< Matrix >
- using **oepdev::SharedVector** = std::shared_ptr< Vector >
- using **oepdev::SharedLocalizer** = std::shared_ptr< Localizer >
- using **oepdev::SharedCISData** = std::shared_ptr< CISData >

15.10 oepdev/liboep/oep_gdf.h File Reference

```
#include <cstdio>
#include <string>
#include "psi4/liboptions/liboptions.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/matrix.h"
#include "../libpsi/integral.h"
```

Classes

- class [oepdev::GeneralizedDensityFit](#)
Generalized Density Fitting Scheme. Abstract Base.
- class [oepdev::SingleGeneralizedDensityFit](#)
Generalized Density Fitting Scheme - Single Fit.
- class [oepdev::DoubleGeneralizedDensityFit](#)
Generalized Density Fitting Scheme - Double Fit.
- class [oepdev::OverlapGeneralizedDensityFit](#)
Generalized Density Fitting Scheme - Single Fit Based on Minimal Overlap in MO Basis.

Namespaces

- [oepdev](#)
OEPEv module namespace.

15.11 oepdev/libpsi/integral.h File Reference

```
#include "psi4/libmints/integral.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/matrix.h"
```

```
#include "multipole_potential.h"
```

Classes

- class [oepdev::TwoBodyAOInt](#)
- class [oepdev::IntegralFactory](#)

Extended [IntegralFactory](#) for computing integrals.

Namespaces

- [oepdev](#)

OEPPDev module namespace.

15.12 oepdev/libpsi/osrecur.h File Reference

Classes

- class [oepdev::ObaraSaikaTwoCenterEFPRecursion_New](#)

Obara-Saika recursion formulae for improved EFP multipole potential integrals.

Namespaces

- [oepdev](#)

OEPPDev module namespace.

Macros

- `#define MAX_DF 500`
- `#define MAX_FAC 100`

Functions

- double *** **oepdev::init_box** (int a, int b, int c)
- void **oepdev::zero_box** (double ***box, int a, int b, int c)
- void **oepdev::free_box** (double ***box, int a, int b)

15.13 oepdev/libpsi/potential.h File Reference

```
#include <vector>
#include "psi4/psi4-dec.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libmints/typedefs.h"
#include "psi4/libmints/onebody.h"
#include "psi4/libmints/potential.h"
#include "psi4/libmints/sointegral_onebody.h"
#include "psi4/libmints/osrecur.h"
```

Classes

- class [oepdev::PotentialInt](#)
Computes potential integrals.

Namespaces

- [oepdev](#)
OEPEv module namespace.

15.14 oepdev/libsolver/solver.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "psi4/psi4-dec.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libmints/potential.h"
#include "psi4/libmints/integral.h"
#include "psi4/libpsi4util/PsiOutputStream.h"
#include "../libutil/wavefunction_union.h"
#include "../libutil/integrals_iter.h"
#include "../libpsi/integral.h"
#include "../liboep/oep.h"
#include "../../include/oepdev_files.h"
```

Classes

- class [oepdev::OEPEvSolver](#)

Solver of properties of molecular aggregates. Abstract base.

- class [oepdev::ElectrostaticEnergySolver](#)

Compute the Coulombic interaction energy between unperturbed wavefunctions.

- class [oepdev::RepulsionEnergySolver](#)

Compute the Pauli-Repulsion interaction energy between unperturbed wavefunctions.

- class [oepdev::ChargeTransferEnergySolver](#)

Compute the Charge-Transfer interaction energy between unperturbed wavefunctions.

- class [oepdev::EETCouplingSolver](#)

Compute the EET coupling energy between unperturbed wavefunctions.

Namespaces

- [oepdev](#)

OEPEv module namespace.

Typedefs

- using [oepdev::SharedWavefunctionUnion](#) = std::shared_ptr< WavefunctionUnion >
[WavefunctionUnion](#).

15.15 oepdev/libsolver/ti_data.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "../lib3d/dmtp.h"
```

Classes

- class [oepdev::TIData](#)

Transfer Integral EET Data.

Namespaces

- [oepdev](#)

OEPEv module namespace.

Typedefs

- using **oepdev::SharedDMTPConvergence** = std::shared_ptr< [oepdev::MultipoleConvergence](#) >

15.16 oepdev/libtest/test.h File Reference

```
#include <vector>
#include "psi4/psi4-dec.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libpsi4util/PsiOutStream.h"
#include "psi4/libmints/integral.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libqt/qt.h"
#include "../libpsi/integral.h"
#include "../libutil/integrals_iter.h"
#include "../../include/oepdev_files.h"
```

Classes

- class [oepdev::test::Test](#)

Manages test routines.

Namespaces

- [oepdev](#)

OEPEv module namespace.

15.17 oepdev/libutil/basis_rotation.h File Reference

```
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/basisset.h"
```

Namespaces

- [psi](#)

Psi4 package namespace.

- [oepdev](#)

OEPDev module namespace.

Functions

Rotation of AO Space

15.17.1 Theory

The objective is to find the formulae for rotation matrices of the AO spaces as functions of the Cartesian 3 x 3 rotation matrices. It is obvious that p-type functions transform as a usual Cartesian vectors. However, higher angular momentum functions transform in a more complex way.

Problem

Define a vectorized AO space M of rank $r > 1$ that is constructed from unique tensor components of fully symmetric r -th rank AO tensor populated in standard order,

$$M_{\{ab\dots k\}} = M_{ab\dots k} \quad \text{for } a \leq b \leq \dots \leq k$$

Given a general rotation of Cartesian tensors

$$M_{ab\dots k} = \sum_{a'b'\dots k'} M_{a'b'\dots k'} r_{a'a} r_{b'b} \dots r_{k'k}$$

find closed expressions for the rotation matrix in reduced composite AO space obeying

$$M_{[ab\dots k]} = \sum_{\{a'b'\dots k'\}} M_{\{a'b'\dots k'\}} R_{\{a'b'\dots k'\}, [ab\dots k]}$$

In the derivations below the following identity of first-order partitioning will be of use:

$$\sum_{ab} M_{ab} \hat{s}_{ab} = \sum_{\{ab\}} M_{\{ab\}} (\hat{s}_{ab} + \Delta_{ab} \hat{s}_{ba})$$

where

$$\Delta_{ab} \equiv 1 - \delta_{ab}$$

and the operator s of rank r acts as follows

$$s_{a'b'\dots k'}^{ab\dots k} \equiv \hat{s}_{a'b'\dots k'} \underbrace{\mathbf{r} \otimes \mathbf{r} \otimes \dots \otimes \mathbf{r}}_r = r_{a'a} r_{b'b} \dots r_{k'k}$$

Rotation of 6D functions

The rotation of the full tensor AO space of rank 2 and dimensions (3,3) is given by

$$M_{ab} = \sum_{a'b'} M_{a'b'} r_{a'b} r_{b'b}$$

Applying the identity of first-order partitioning directly leads to the formula for a reduced 6D tensor rotation of rank 1 and dimension (6),

$$M_{[ab]} = \sum_{\{a'b'\}} M_{\{a'b'\}} R_{\{a'b'\},[ab]}$$

where the 6 x 6 rotation matrix is given by

$$R_{\{a'b'\},[ab]} = r_{a'a} r_{b'b} + \Delta_{a'b'} r_{b'a} r_{a'b}$$

Rotation of 10F functions

The rotation of the full tensor AO space of rank 3 and dimensions (3,3,3) is given by

$$M_{abc} = \sum_{a'b'c'} M_{a'b'c'} r_{a'b} r_{b'b} r_{c'c}$$

First of all, notice that one can perform the following partitioning

$$\sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} = \sum_{\{abc\}} M_{\{abc\}} (\hat{s}_{abc} + \hat{s}_{acb} + \hat{s}_{bac} + \hat{s}_{bca} + \hat{s}_{cab} + \hat{s}_{cba})$$

Then, perform a partitioning of the triple sum,

$$\begin{aligned} \sum_{abc} M_{abc} \hat{s}_{abc} &= \sum_a \sum_{b \neq a} \sum_{c \neq b \neq a} M_{abc} \hat{s}_{abc} \\ &+ \sum_a \sum_{b \geq a} M_{abb} \hat{s}_{abb} + \sum_a \sum_{b < a} M_{abb} \hat{s}_{abb} \\ &+ \sum_a \sum_{b > a} M_{aba} \hat{s}_{aba} + \sum_a \sum_{b < a} M_{aba} \hat{s}_{aba} \\ &+ \sum_a \sum_{b > a} M_{bba} \hat{s}_{bba} + \sum_a \sum_{b < a} M_{bba} \hat{s}_{bba} \end{aligned}$$

Using the first-order partitioning theorem and interchanging the dummy indices one finds that

$$M_{[abc]} = \sum_{\{a'b'c'\}} M_{\{a'b'c'\}} R_{\{a'b'c'\},[abc]}$$

where the 10 x 10 rotation matrix is given by

$$\begin{aligned} R_{\{a'b'c'\},[abc]} &= \delta_{b'c'} \left(s_{a'b'b'}^{abc} + \Delta_{a'b'} \left\{ s_{b'a'b'}^{abc} + s_{b'b'a'}^{abc} \right\} \right) \\ &+ \delta_{a'b'} \Delta_{b'c'} \left(s_{c'a'a'}^{abc} + s_{a'c'a'}^{abc} + s_{a'a'c'}^{abc} \right) \\ &+ \Delta_{a'b'} \Delta_{b'c'} \left(s_{a'b'c}^{abc} + s_{a'c'b'}^{abc} + s_{b'a'c'}^{abc} + s_{b'c'a'}^{abc} + s_{c'a'b'}^{abc} + s_{c'b'a'}^{abc} \right) \end{aligned}$$

and

$$s_{a'b'c'}^{abc} \equiv \hat{s}_{a'b'c'} \mathbf{r} \otimes \mathbf{r} \otimes \mathbf{r} = r_{a'a} r_{b'b} r_{c'c}$$

- `psi::SharedMatrix oepdev::r6` (`psi::SharedMatrix r`)
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `void oepdev::populate` (`double **R`, `double **r`, `std::vector< int > idx_am`, `const int &nam`)
Compute the 6 x 6 rotation matrix of the 6D orbitals.
- `psi::SharedMatrix oepdev::ao_rotation_matrix` (`psi::SharedMatrix r`, `psi::SharedBasisSet b`)
Compute the full rotation matrix of AO orbital space.

15.18 oepdev/libutil/cis.h File Reference

```
#include <string>
#include <utility>
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libpsio/psio.hpp"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libtrans/integraltransform.h"
#include "psi4/libtrans/mospace.h"
#include "psi4/libdpd/dpd.h"
#include "psi4/libfock/jk.h"
#include "../lib3d/dmtp.h"
#include "davidson_liu.h"
```

Classes

- struct `oepdev::CISData`
CIS wavefunction parameters. Container structure.
- class `oepdev::CISComputer`
CISComputer.
- class `oepdev::R_CISComputer`
- class `oepdev::U_CISComputer`
- class `oepdev::R_CISComputer_Explicit`
- class `oepdev::R_CISComputer_DL`
CIS Computer with RHF reference: Davidson-Liu Solver.
- class `oepdev::R_CISComputer_Direct`
- class `oepdev::U_CISComputer_Explicit`
- class `oepdev::U_CISComputer_DL`
CIS Computer with UHF reference: Davidson-Liu Solver.

Namespaces

- [oepdev](#)

OEPDev module namespace.

Typedefs

- using **oepdev::SharedMolecule** = std::shared_ptr< psi::Molecule >
- using **oepdev::SharedMOSpace** = std::shared_ptr< psi::MOSpace >
- using **oepdev::SharedMOSpaceVector** = std::vector< std::shared_ptr< psi::MOSpace > >
- using **oepdev::SharedIntegralTransform** = std::shared_ptr< psi::IntegralTransform >

15.19 oepdev/libutil/davidson_liu.h File Reference

```
#include "psi4/liboptions/liboptions.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "gram_schmidt.h"
```

Classes

- class [oepdev::DavidsonLiu](#)

Davidson-Liu diagonalization method.

Namespaces

- [oepdev](#)

OEPDev module namespace.

15.20 oepdev/libutil/diis.h File Reference

```
#include <cstdio>
#include <string>
#include <vector>
#include "psi4/libciomr/libciomr.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libqt/qt.h"
#include "psi4/libpsi4util/PsiOutputStream.h"
```

Classes

- class [oepdev::DIISManager](#)
DIIS manager.

Namespaces

- [oepdev](#)
OEPEv module namespace.

15.21 oepdev/libutil/gram_schmidt.h File Reference

```
#include "psi4/libmints/vector.h"
```

Classes

- class [oepdev::GramSchmidt](#)
Gram-Schmidt orthogonalization method.

Namespaces

- [oepdev](#)
OEPEv module namespace.

15.22 oepdev/libutil/integrals_iter.h File Reference

```
#include <cstdio>
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/integral.h"
#include "../libpsi/integral.h"
```

Classes

- class [oepdev::ShellCombinationsIterator](#)
Iterator for Shell Combinations. Abstract Base.
- class [oepdev::AOIntegralsIterator](#)
Iterator for AO Integrals. Abstract Base.
- class [oepdev::AllAOShellCombinationsIterator_4](#)

Loop over all possible ERI shells in a shell quartet.

- class [oepdev::AllAOShellCombinationsIterator_2](#)

Loop over all possible ERI shells in a shell doublet.

- class [oepdev::AllAOIntegralsIterator_4](#)

Loop over all possible ERI within a particular shell quartet.

- class [oepdev::AllAOIntegralsIterator_2](#)

Loop over all possible ERI within a particular shell doublet.

Namespaces

- [oepdev](#)

OEPEv module namespace.

Typedefs

- using [oepdev::SharedIntegralFactory](#) = std::shared_ptr< IntegralFactory >
- using [oepdev::SharedTwoBodyAOInt](#) = std::shared_ptr< TwoBodyAOInt >
- using [oepdev::SharedShellsIterator](#) = std::shared_ptr< ShellCombinationsIterator >
Iterator over shells as shared pointer.
- using [oepdev::SharedAOIntsIterator](#) = std::shared_ptr< AOIntegralsIterator >
Iterator over AO integrals as shared pointer.

15.23 oepdev/libutil/kabsch_superimposer.h File Reference

```
#include <string>
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/molecule.h"
```

Classes

- class [oepdev::KabschSuperimposer](#)

Compute the Cartesian rotation matrix between two structures.

Namespaces

- [oepdev](#)

OEPEv module namespace.

15.24 oepdev/libutil/quambo.h File Reference

```
#include <string>
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/wavefunction.h"
```

Classes

- struct [oepdev::QUAMBOData](#)
Container to store the [QUAMBO](#) data.
- class [oepdev::QUAMBO](#)
The Quasiatomic Minimal Basis Set Molecular Orbitals ([QUAMBO](#))

Namespaces

- [oepdev](#)
OEPEv module namespace.

Typedefs

- using [oepdev::SharedQUAMBO](#) = std::shared_ptr< QUAMBO >
Shared [QUAMBO](#) object.

15.25 oepdev/libutil/scf_perturb.h File Reference

```
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libscf_solver/rhf.h"
```

Classes

- struct [oepdev::PerturbCharges](#)
Structure to hold perturbing charges.
- class [oepdev::RHFPerturbed](#)
RHF theory under electrostatic perturbation.

Namespaces

- [oepdev](#)

OEPEv module namespace.

15.26 oepdev/libutil/unitary_optimizer.h File Reference

```
#include <string>
#include <complex>
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
```

Classes

- struct [oepdev::ABCD](#)

Simple structure to hold the Fourier series expansion coefficients.

- struct [oepdev::Fourier5](#)

Simple structure to hold the Fourier series expansion coefficients for N=2.

- struct [oepdev::Fourier9](#)

Simple structure to hold the Fourier series expansion coefficients for N=4.

- class [oepdev::UnitaryOptimizer](#)

Find the optimim unitary matrix of quadratic matrix equation.

- class [oepdev::UnitaryOptimizer_4_2](#)

Find the optimim unitary matrix for quartic-quadratic matrix equation with trace.

- class [oepdev::UnitaryOptimizer_2](#)

Find the optimim unitary matrix for quadratic matrix equation with trace.

- class [oepdev::UnitaryOptimizer_2_1](#)

Namespaces

- [oepdev](#)

OEPEv module namespace.

Macros

- `#define IDX(i, j, n) ((n)*(i)+(j))`
- `#define IDX3(i, j, k) (n2_*(i)+n_*(j)+(k))`
- `#define IDX6(i, j, k, l, m, n) (n5_*(i)+n4_*(j)+n3_*(k)+n2_*(l)+n_*(m)+(n))`

Functions

- `constexpr std::complex< double > oepdev::operator""i` (unsigned long long d)
- `constexpr std::complex< double > oepdev::operator""i` (long double d)

15.27 oepdev/libutil/util.h File Reference

```
#include <cstdio>
#include <string>
#include <cmath>
#include <map>
#include <cassert>
#include "psi4/psi4-dec.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libciomr/libciomr.h"
#include "psi4/libpsio/psio.hpp"
#include "psi4/libiwl/iwl.h"
#include "psi4/libqt/qt.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/writer.h"
#include "psi4/libmints/writer_file_prefix.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/mintshelper.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/oeprop.h"
#include "psi4/libmints/local.h"
#include "psi4/libfunctional/superfunctional.h"
#include "psi4/libtrans/mospace.h"
#include "psi4/libtrans/integraltransform.h"
#include "psi4/libscf_solver/rhf.h"
#include "psi4/libdpd/dpd.h"
```

Namespaces

- [oepdev](#)
OEPEv module namespace.

Typedefs

- using **oepdev::SharedSuperFunctional** = `std::shared_ptr< SuperFunctional >`

Functions

- PSI_API void [oepdev::preamble](#) (void)
Print preamble for module OEPDEV.
- template<typename... Args>
std::string [oepdev::string_sprintf](#) (const char *format, Args... args)
Format string output. Example: std::string text = oepdev::string_sprintf("Test %3d, %13.5f", 5, -10.5425);.
- PSI_API std::shared_ptr< SuperFunctional > [oepdev::create_superfunctional](#) (std::string name, Options &options)
Set up DFT functional.
- PSI_API SharedBasisSet [oepdev::create_basisset_by_copy](#) (SharedBasisSet basis_ref, SharedMolecule molecule_target)
Build BasisSet by Copy.
- PSI_API SharedBasisSet [oepdev::create_atom_basisset_by_copy](#) (SharedBasisSet basis_ref, SharedMolecule molecule_target, int idx_atom)
Build BasisSet by Copy for a Particular Atom.
- PSI_API std::shared_ptr< Molecule > [oepdev::extract_monomer](#) (std::shared_ptr< const Molecule > molecule_dimer, int id)
Extract molecule from dimer.
- PSI_API double [oepdev::compute_distance](#) (psi::SharedVector v1, psi::SharedVector v2)
Compute distance between two points in nD space.
- PSI_API std::shared_ptr< Wavefunction > [oepdev::solve_scf](#) (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< BasisSet > auxiliary, std::shared_ptr< BasisSet > guess, std::shared_ptr< SuperFunctional > functional, Options &options, std::shared_ptr< PSIO > psio, bool compute_mints=false)
Solve RHF-SCF equations for a given molecule in a given basis set.
- PSI_API std::shared_ptr< Wavefunction > [oepdev::solve_scf_sad](#) (std::shared_ptr< Molecule > molecule, std::shared_ptr< BasisSet > primary, std::shared_ptr< BasisSet > auxiliary, std::vector< std::shared_ptr< BasisSet >> sad, std::vector< std::shared_ptr< BasisSet >> sad_fit, std::shared_ptr< SuperFunctional > functional, Options &options, std::shared_ptr< PSIO > psio, bool compute_mints=false)
Solve RHF-SCF equations for a given molecule in a given basis set.
- PSI_API double [oepdev::average_moment](#) (std::shared_ptr< psi::Vector > moment)
Compute the scalar magnitude of multipole moment.
- PSI_API std::vector< std::shared_ptr< psi::Matrix >> [oepdev::calculate_JK](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::Matrix > C)
Compute the Coulomb and exchange integral matrices in MO basis.
- PSI_API std::vector< std::shared_ptr< psi::Matrix >> [oepdev::calculate_JK_ints](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr)
Compute the Coulomb and exchange integral matrices in MO basis.
- PSI_API std::vector< std::shared_ptr< psi::Matrix >> [oepdev::calculate_JK_r](#) (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr, std::shared_ptr< psi::Matrix > Dij)

Compute the Coulomb and exchange integral matrices in MO basis.

- PSI_API std::vector< std::shared_ptr< psi::Matrix > > **oepdev::calculate_JK_rb** (std::shared_ptr< psi::Wavefunction > wfn, std::shared_ptr< psi::IntegralTransform > tr, std::shared_ptr< psi::Matrix > Dij)
- std::shared_ptr< psi::Matrix > **oepdev::calculate_DFI_Vel** (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::IntegralFactory > f_abab, std::shared_ptr< psi::Matrix > d_b)

Compute the Effective DFI Potential Matrix Due To Electrons.

- PSI_API std::shared_ptr< psi::Matrix > **oepdev::calculate_DFI_Vel_JK** (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::IntegralFactory > f_abab, std::shared_ptr< psi::Matrix > d_b)

Compute the Effective DFI Coulomb+Exchange Potential Matrix Due To Electrons.

- PSI_API std::shared_ptr< psi::Matrix > **oepdev::calculate_DFI_Vel_J** (std::shared_ptr< psi::IntegralFactory > f_aabb, std::shared_ptr< psi::Matrix > d_b)

Compute the Effective DFI Coulomb Potential Matrix Due To Electrons.

- PSI_API std::shared_ptr< psi::Matrix > **oepdev::calculate_OEP_basisopt_V** (const int &nt, std::shared_ptr< psi::IntegralFactory > f_pppt, std::shared_ptr< psi::Matrix > ca, std::shared_ptr< psi::Matrix > da)

Compute the 2-Electron Part of the Effective OEP Matrix for Auxiliary Basis Set Optimization.

- PSI_API double **oepdev::bs_optimize_projection** (std::shared_ptr< psi::Matrix > ti, std::shared_ptr< psi::MintsHelper > mints, std::shared_ptr< psi::BasisSet > bsf_m, std::shared_ptr< psi::BasisSet > bsf_i)

Compute the objective function value for auxiliary basis set optimization of OEPs.

15.28 oepdev/libutil/wavefunction_union.h File Reference

```
#include <cstdio>
#include <string>
#include <map>
#include "psi4/psi4-dec.h"
#include "psi4/liboptions/liboptions.h"
#include "psi4/libpsio/psio.h"
#include "psi4/libciomr/libciomr.h"
#include "psi4/libpsio/psio.hpp"
#include "psi4/libqt/qt.h"
#include "psi4/libmints/molecule.h"
#include "psi4/libmints/writer.h"
#include "psi4/libmints/writer_file_prefix.h"
#include "psi4/libmints/wavefunction.h"
#include "psi4/libmints/basisset.h"
#include "psi4/libmints/vector.h"
#include "psi4/libmints/matrix.h"
#include "psi4/libmints/oeprop.h"
#include "psi4/libmints/local.h"
#include "psi4/libfunctional/superfunctional.h"
```

```
#include "psi4/libtrans/mospace.h"
#include "psi4/libtrans/integraltransform.h"
#include "psi4/libscf_solver/rhf.h"
#include "psi4/libdpd/dpd.h"
```

Classes

- class [oepdev::WavefunctionUnion](#)

Union of two Wavefunction objects.

Namespaces

- [oepdev](#)

OEPEv module namespace.

CHAPTER 16

Example Documentation

16.1 example_cphf.cc

Shows how to use the `oepdev::CPHF` solver to compute molecular and LMO-distributed polarizabilities at RHF level of theory.

```
void example_cphf(std::shared_ptr<psi::Wavefunction> wfn, psi::Options& opt){  
    // build the solver  
    std::shared_ptr<oepdev::CPHF> solver = std::make_shared<oepdev::CPHF>(wfn, opt);  
  
    // run the solver to converge CPHF equations  
    solver->compute();  
  
    // print the LMO-distributed polarizabilities  
    for (int i=0; i<solver->nocc(); i++) {  
        solver->polarizability(i)->print();  
    }  
  
    // print the molecular polarizability  
    solver->polarizability()->print();  
  
    // grab 4th LMO-distributed polarizability and its associated LMO centroid  
    psi::SharedMatrix pol_4 = solver->polarizability(3);  
    psi::SharedVector rmo_4 = solver->lmo.centroid(3);  
};
```

16.2 example_davidson_liu.cc

This example is a trivial demo to use `oepdev::DavidsonLiu` in order to diagonalize a real, symmetric matrix **H**, stored in a `psi::SharedMatrix H`. This can help you to construct more complicated classes that need to solve eigenpairs for very large, sparse matrices such as CI Hamiltonians.

Note

This example might need compile properly (it's only a draft). Debug if necessary.

```
// Define a class that inherits from DavidsonLiu
class Diagonalize : public DavidsonLiu {

public:
    Diagonalize(psi::SharedMatrix H, psi::Options& opt, int M);
    virtual void ~Diagonalize() {};

protected:
    // Desired number of roots to be found
    int M_;
    // Matrix to be diagonalized (explicitly stored)
    psi::SharedMatrix matrix_;

    // Implementation of pure methods must be declared
    void davidson_liu_compute_diagonal_hamiltonian();
    void davidson_liu_compute_sigma();
};

Diagonalize::Diagonalize(psi::SharedMatrix H, psi::Options& opt, int M) :
    DavidsonLiu(opt) , matrix_(nullptr), M_(M)
{
    matrix_ = std::make_shared<psi::Matrix>(H);
    int N = H->ncol();
    int L = M_;

    // Must be run in order to allocate memory
    this->davidson_liu_initialize(N, L, M_);
}

// Implementation of pure methods
void Diagonalize::davidson_liu_compute_diagonal_hamiltonian() {
    for (int i=0; i<this->matrix_->ncol(); ++i) {
        double v = matrix_->get(i, i);
        this->H.diag_davidson_liu->set(i, v);
    }
}

void Diagonalize::davidson_liu_compute_sigma() {
    for (int k=this->davidson_liu_n_sigma_computed; k<this->L.davidson_liu; ++k) {
        psi::SharedVector Sigma = std::make_shared<psi::Vector>("", this->N_davidson_liu);
        Sigma->gemv(false, 1.0, *this->matrix_, *Sigma, 0.0);
        this->sigma_vectors.davidson_liu.push_back(Sigma);
    }
}

// Testing function
void example_davidson_liu(psi::SharedMatrix H, int M, psi::Options& opt){

    // Construct the solver object
    Diagonalize solver(H, opt, M);

    // Find *M* lowest eigenpairs of a given matrix **H**
    solver.run_davidson_liu();
};
```

16.3 example_gefp.cc

Working with GenEffFrag objects

At the moment, `psi::Molecule` and `psi::BasisSet` objects do not have Cartesian rotation implemented which prohibits using them as containers in OEPDev. On the other hand, many calculations in FB approaches require molecule and basis set rotation. Therefore, to tem-

porarily overcome this technical difficulty, molecule and basis set objects need to be supplied for each fragment in the system by building them from scratch. Below, the guideline for fragment generation and manipulation is given:

```
// Create empty fragment
SharedGenEffFrag fragment = oepdev::GenEffFrag::build("Ethylene");
// Set the parameters
fragment->parameters["efp2"] = par_efp2;
fragment->parameters["eet"] = par_eet;
// Set the number of doubly occupied MOs and number of primary basis functions at the end
fragment->set_ndocc(ndocc);
fragment->set_nbf(nbf);
// Set the current molecule and basis set
fragment->set_molecule(mol);
fragment->set_basisset("primary", basis_prim);
fragment->set_basisset("auxiliary", basis_aux);
```

Creating the parameters can be done by using an appropriate factory

```
SharedGenEffParFactory factory = GenEffParFactory::build("OEP-EFP2", wfn, options,
    auxiliary, intermediate);
SharedGenEffPar parameters = factory->compute();
```

Currently, parameters are not created with allocated basis set objects due to the above mentioned problem in Psi4 regarding lack of functionality of basis set rotation. Therefore, **it is important to first set the parameters before setting the basis set** when constructing the fragments. It is because using the `set_basisset` method for the fragment sets the basis set for all parameters as well, and if the parameters were set after the basis set, they would not have any basis sets allocated leading to errors in FB calculations. This problem will not emerge once a rotation of `psi::BasisSet` is implemented (either in Psi4 or in OEPDev).

```
void example_gefp() {
//TODO
}
```

16.4 example_integrals_iter.cc

Iterations over electron repulsion integrals in AO basis. This is an example of how to use

- the `oepdev::ShellCombinationsIterator` class
- the `oepdev::AOIntegralsIterator` class.

```
void iterate(std::shared_ptr<oepdev::IntegralFactory> ints)
{
    // Prepare for direct calculation of ERI's (shell by shell)
    std::shared_ptr<psi::TwoBodyAOInt> tei(ints->eri());

    // Grab the buffer where the integrals for a current shell will be placed
    const double* buffer = tei->buffer();

    // Create iterator to go through all shell quartet combinations
    oepdev::SharedShellsIterator shellIter =
        oepdev::ShellCombinationsIterator::build(ints, "ALL", 4);

    // Iterate over shells, and then over all integrals in each shell quartet
    for (shellIter->first(); shellIter->is.done() == false; shellIter->next())
```

```

{
    // Compute all integrals between shells in the current quartet
    shellIter->compute_shell(tei);

    // Create iterator to go through all integrals within a shell quartet
    oepdev::SharedAOIntsIterator intsIter = shellIter->ao_iterator("ALL");

    for (intsIter->first(); intsIter->is_done() == false; intsIter->next())
    {
        // Grab current (ij|kl) indices here
        int i = intsIter->i();
        int j = intsIter->j();
        int k = intsIter->k();
        int l = intsIter->l();

        // Grab the (ij|kl) integral
        double integral = buffer[intsIter->index()];
    }
}
}

```

16.5 example_scf_perturb.cc

Perturb HF Hamiltonian with external electrostatic potential. This is an example of how to use the `oepdev::RHFPerturbed` class.

```

void scf_perturb(std::shared_ptr<psi::Wavefunction> wfn, psi::Options& opt)
{
    // Set up HF superfunctional
    std::shared_ptr<psi::SuperFunctional> func = oepdev::create_superfunctional
        ("HF", opt);

    // Initialize the perturbed wavefunction
    std::shared_ptr<oepdev::RHFPerturbed> scf = std::make_shared<oepdev::RHFPerturbed>(wfn, func, opt, wfn->
        psio());

    /* Perturb the system with the uniform electric field [Fx, Fy, Fz].
       Then, add two point charges of charge qi placed at [Rxi, Ryi, Rzi].
       Provide all these values in atomic units! */
    const double Fx = 0.04, Fy = 0.05, Fz = -0.09;
    const double Rx1= 0.00, Rx2= 1.30, Rx3= -1.00;
    const double Rx1= 0.10, Rx2=-0.30, Rx3= 3.50;
    const double q1 = 0.30, q2 = -0.09;

    scf->set_perturbation(Fx, Fy, Fz);          /* set it only once, setting it again will overwrite the
        field, not add */
    scf->set_perturbation(Rx1, Ry1, Rz1, q1);
    scf->set_perturbation(Rx2, Ry2, Rz2, q2); /* more charges can be added */

    // Solve perturbed SCF equations
    scf->compute_energy();

    // Grab some data
    double energy = scf->reference_energy();    // Total energy of the system
    std::shared_ptr<psi::Matrix> Da = scf->Da(); // One-particle density matrix

    /* Note that the external field and charges perturb only one-electron Hamiltonian.*/
}

```

Bibliography

- Peng Xu, Emilie B. Guidez, Colleen Bertoni, and Mark S. Gordon. Perspective: Ab initio force field methods derived from quantum mechanics. *J. Chem. Phys.*, 148(9):090901, 2018a. [3](#)
- Jacopo Tomasi, Benedetta Mennucci, and Roberto Cammi. Quantum mechanical continuum solvation models. *Chem. Rev.*, 105(8):2999–3094, 2005. [3](#)
- A. Warshel and M. Levitt. Theoretical studies of enzymic reactions: Dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme. *J. Mol. Biol.*, 103(2):227 – 249, 1976. [3](#)
- Hans Martin Senn and Walter Thiel. QM/MM methods for biomolecular systems. *Angew. Chem. Int. Ed.*, 48(7):1198–1229, 2009. [3](#)
- Omar Demerdash, Eng-Hui Yap, and Teresa Head-Gordon. Advanced potential energy surfaces for condensed phase simulation. *Annu. Rev. Phys. Chem.*, 65(1):149–174, 2014. [3](#)
- Mark S. Gordon, Dmitri G. Fedorov, Spencer R. Pruitt, and Lyudmila V. Slipchenko. Fragmentation methods: A route to accurate calculations on large systems. *Chem. Rev.*, 112(1):632–672, 2012. [3](#)
- Mario Barbatti. Photorelaxation induced by water-chromophore electron transfer. *J. Am. Chem. Soc.*, 136(29):10246–10249, 2014. [3](#)
- Rafał Szabla, Jiří Šponer, and Robert W. Góra. Electron-driven proton transfer along H₂O wires enables photorelaxation of $\pi\sigma^*$ states in chromophore-water clusters. *J. Phys. Chem. Lett.*, 6(8):1467–1471, 2015. [3](#)
- Joanna Bednarska, Robert Zaleśny, Guangjun Tian, Natarajan Arul Murugan, Hans Ågren, and Wojciech Bartkowiak. Nonempirical simulations of inhomogeneous broadening of electronic transitions in solution: Predicting band shapes in one- and two-photon absorption spectra of chalcones. *Molecules*, 22(10):1643, 2017. [3](#)
- Beata Jedrzejewska, Anna Grabarz, Wojciech Bartkowiak, and Borys Ośmiałowski. Spectral and physicochemical properties of difluoroboranyls containing n,n-dimethylamino group studied by solvatochromic methods. *Spectrochim. Acta A*, 199:86 – 95, 2018. [3](#)

- Basile F. E. Curchod and Todd J. Martínez. Ab initio nonadiabatic quantum molecular dynamics. *Chem. Rev.*, 118(7):3305–3336, 2018. [3](#)
- Bartosz Błasiak, Casey H. Londergan, Lauren J. Webb, and Minhaeng Cho. Vibrational probes: From small molecule solvatochromism theory and experiments to applications in complex systems. *Acc. Chem. Res.*, 50(4):968–976, 2017. [3](#)
- Rosalind J. Xu, Bartosz Błasiak, Minhaeng Cho, Joshua P. Layfield, and Casey H. Londergan. A direct, quantitative connection between molecular dynamics simulations and vibrational probe line shapes. *J. Phys. Chem. Lett.*, 9(10):2560–2567, 2018b. [3](#)
- Nicholas H. C. Lewis, Natalie L. Gruenke, Thomas A. A. Oliver, Matteo Ballottari, Roberto Bassi, and Graham R. Fleming. Observation of electronic excitation transfer through light harvesting complex II using two-dimensional electronic–vibrational spectroscopy. *J. Phys. Chem. Lett.*, 7(20):4197–4206, 2016. [3](#)
- W. Kohn and L. J. Sham. Self-consistent equations including exchange and correlation effects. *Phys. Rev.*, 140:A1133–A1138, 1965. [3](#)
- A. Holas and N. H. March. Construction of the pauli potential, pauli energy, and effective potential from the electron density. *Phys. Rev. A*, 44:5521–5536, 1991. [3](#)
- C. C. J. Roothaan. New developments in molecular orbital theory. *Rev. Mod. Phys.*, 23:69–89, 1951. [3](#)
- P. Hohenberg and W. Kohn. Inhomogeneous electron gas. *Phys. Rev.*, 136:B864–B871, 1964. [3](#)
- P. Otto and J. Ladik. Investigation of the interaction between molecules at medium distances: I. scf lcao mo supermolecule, perturbational and mutually consistent calculations for two interacting hf and ch2o molecules. *Chem. Phys.*, 8(1):192 – 200, 1975. [3](#), [13](#)
- Wolfgang Weber and Walter Thiel. Orthogonalization corrections for semiempirical methods. *Theor. Chem. Acc.*, 103(6):495–506, 2000. [4](#)
- Frank Neese. Efficient and accurate approximations to the molecular spin-orbit coupling operator and their use in molecular g-tensor calculations. *J. Chem. Phys.*, 122(3):034107, 2005. [4](#)
- G. Andrés Cisneros, Jean-Philip Piquemal, and Thomas A. Darden. Intermolecular electrostatic energies using density fitting. *J. Chem. Phys.*, 123(4):044109, 2005. [4](#)
- Jean-Philip Piquemal, G. Andrés Cisneros, Peter Reinhardt, Nohad Gresh, and Thomas A. Darden. Towards a force field based on density fitting. *J. Chem. Phys.*, 124(10):104101, 2006. [4](#)
- Hui Li, Mark S. Gordon, and Jan H. Jensen. Charge transfer interaction in the effective fragment potential method. *J. Chem. Phys.*, 124(21):214108, 2006. [4](#), [13](#)
- Bartosz Błasiak, Hohan Lee, and Minhaeng Cho. Vibrational solvatochromism: Towards systematic approach to modeling solvation phenomena. *J. Chem. Phys.*, 139(4):044111, 2013. [4](#)

- Bartosz Błasiak, Michał Maj, Minhaeng Cho, and Robert W. Góra. Distributed multipolar expansion approach to calculation of excitation energy transfer couplings. *J. Chem. Theory Comput.*, 11(7):3259–3266, 2015. [4](#), [13](#)
- Bartosz Błasiak, Joanna D. Bednarska, Marta Chołuj, Robert W. Góra, and Wojciech Bartkowiak. Ab initio effective one-electron potential operators: Applications for charge-transfer energy in effective fragment potentials. *J. Comput. Chem., Accepted*, 2020a. doi: 10.26434/chemrxiv.13228439.v1. [4](#), [7](#), [9](#)
- Bartosz Błasiak, Wojciech Bartkowiak, and Robert W. Góra. An effective potential for Frenkel excitons. *Submitted*, 2020b. [5](#), [7](#), [154](#)
- Bartosz Błasiak. One-particle density matrix polarization susceptibility tensors. *J. Chem. Phys.*, 149(16):164115, 2018. [5](#)
- Mark S. Gordon, Quentin A. Smith, Peng Xu, and Lyudmila V. Slipchenko. Accurate first principles model potentials for intermolecular interactions. *Annu. Rev. Phys. Chem.*, 64(1):553–578, 2013. [13](#)
- Peng Xu and Mark S. Gordon. Charge transfer interaction using quasiatomic minimal-basis orbitals in the effective fragment potential method. *J. Chem. Phys.*, 139(19):194104, 2013. [13](#)
- John Norman Murrell, M. Randić, D. R. Williams, and Hugh Christopher Longuet-Higgins. The theory of intermolecular forces in the region of small orbital overlap. *Proc. R. Soc. Lond. A*, 284(1399):566–581, 1965. [13](#)
- I.C. Hayes and A.J. Stone. An intermolecular perturbation theory for the region of moderate overlap. *Mol. Phys.*, 53(1):83–105, 1984. doi: 10.1080/00268978400102151. [13](#)
- Marcos Mandado and José M. Hermida-Ramón. Electron density based partitioning scheme of interaction energies. *J. Chem. Theory Comput.*, 7(3):633–641, 2011. [13](#)
- Walter J. Stevens and William H. Fink. Frozen fragment reduced variational space analysis of hydrogen bonding interactions. application to the water dimer. *Chem. Phys. Lett.*, 139(1):15 – 22, 1987. [13](#)
- Kazuhiro J. Fujimoto. Transition-density-fragment interaction combined with transfer integral approach for excitation-energy transfer via charge-transfer states. *J. Chem. Phys.*, 137(3): 034101, 2012. [13](#)
- W.Andrzej Sokalski and R.A. Poirier. Cumulative atomic multipole representation of the molecular charge distribution and its basis set dependence. *Chem. Phys. Lett.*, 98(1):86–92, 1983. [155](#)
- Catherine Etchebest, Richard Lavery, and Alberte Pullman. The calculation of molecular electrostatic potential from a multipole expansion based on localized orbitals and developed at their centroids: Accuracy and applicability for macromolecular computations. *Theoret. Chim. Acta*, 62(1):17–28, 1982. [155](#)

- `_calculate_DFI_Vel`
 - The EOPDev Utilities, [78](#)
- `AllAOIntegralsIterator_2`
 - `oepdev::AllAOIntegralsIterator_2`, [97](#), [98](#)
- `AllAOIntegralsIterator_4`
 - `oepdev::AllAOIntegralsIterator_4`, [99](#), [100](#)
- `AllAOShellCombinationsIterator_2`
 - `oepdev::AllAOShellCombinationsIterator_2`, [101](#), [102](#)
- `AllAOShellCombinationsIterator_4`
 - `oepdev::AllAOShellCombinationsIterator_4`, [104](#), [105](#)
- `allocate`
 - `oepdev::GenEffPar`, [196](#)
- `ao_iterator`
 - `oepdev::ShellCombinationsIterator`, [289](#)
- `ao_rotation_matrix`
 - The EOPDev Utilities, [73](#)
- `average_moment`
 - The EOPDev Utilities, [76](#)
- `basisset`
 - `oepdev::GenEffPar`, [198](#)
- `bs_optimize_projection`
 - The EOPDev Utilities, [79](#)
- `build`
 - `oepdev::AOIntegralsIterator`, [107](#), [108](#)
 - `oepdev::CISComputer`, [118](#)
 - `oepdev::DMTPole`, [140](#)
 - `oepdev::Field3D`, [176](#)
 - `oepdev::FragmentedSystem`, [179](#)
 - `oepdev::GenEffParFactory`, [204](#), [205](#)
 - `oepdev::GeneralizedDensityFit`, [208](#), [209](#)
 - `oepdev::OEPDevSolver`, [240](#)
 - `oepdev::OEPotential`, [247](#)
 - `oepdev::Points3DIterator`, [254](#), [255](#)
 - `oepdev::PointsCollection3D`, [258](#)
 - `oepdev::ShellCombinationsIterator`, [288](#)
- `CPHF`
 - `oepdev::CPHF`, [126](#)
- `Ca_subset`
 - `oepdev::WavefunctionUnion`, [340](#)
- `calculate_DFI_Vel_JK`
 - The EOPDev Utilities, [78](#)
- `calculate_DFI_Vel_J`
 - The EOPDev Utilities, [78](#)
- `calculate_JK_r`
 - The EOPDev Utilities, [77](#)
- `calculate_JK`
 - The EOPDev Utilities, [77](#)
- `calculate_OEP_basisopt_V`
 - The EOPDev Utilities, [79](#)
- `Cb_subset`
 - `oepdev::WavefunctionUnion`, [341](#)
- `compute`
 - `oepdev::DIISManager`, [133](#)
 - `oepdev::DMTPole`, [142](#), [143](#)
 - `oepdev::DoubleGeneralizedDensityFit`, [147](#)
 - `oepdev::GeneralizedDensityFit`, [209](#)
 - `oepdev::KabschSuperimposer`, [220](#), [221](#)
 - `oepdev::MultipoleConvergence`, [227](#)
 - `oepdev::OverlapGeneralizedDensityFit`, [251](#)
 - `oepdev::SingleGeneralizedDensityFit`, [291](#)
 - `oepdev::TwoBodyAOInt`, [305](#)
- `compute_benchmark`
 - `oepdev::ChargeTransferEnergySolver`, [114](#)
 - `oepdev::EETCouplingSolver`, [156](#)
 - `oepdev::ElectrostaticEnergySolver`, [163](#)

- oepdev::OEPDevSolver, [241](#)
- oepdev::RepulsionEnergySolver, [283](#)
- compute_density_matrix
 - oepdev::GenEffPar, [200](#), [201](#)
- compute_distance
 - The EOPDev Utilities, [74](#)
- compute_energy
 - oepdev::FragmentedSystem, [181](#)
 - oepdev::GenEffFrag, [187](#)
- compute_energy_term
 - oepdev::FragmentedSystem, [181](#)
 - oepdev::GenEffFrag, [187](#)
- compute_many_body_energy_term
 - oepdev::GenEffFrag, [188](#)
- compute_oep_based
 - oepdev::ChargeTransferEnergySolver, [114](#)
 - oepdev::EETCouplingSolver, [155](#)
 - oepdev::ElectrostaticEnergySolver, [162](#)
 - oepdev::OEPDevSolver, [241](#)
 - oepdev::RepulsionEnergySolver, [282](#)
- compute_shell
 - oepdev::AllAOShellCombinationsIterator_2, [103](#)
 - oepdev::AllAOShellCombinationsIterator_4, [105](#), [106](#)
 - oepdev::ShellCombinationsIterator, [289](#)
 - oepdev::TwoElectronInt, [308](#)
- ConvergenceLevel
 - oepdev::MultipoleConvergence, [226](#)
- coupling_direct
 - oepdev::TIData, [299](#)
- coupling_direct_coul
 - oepdev::TIData, [300](#)
- coupling_direct_exch
 - oepdev::TIData, [300](#)
- coupling_indirect
 - oepdev::TIData, [300](#)
- coupling_indirect_ti2
 - oepdev::TIData, [300](#)
- coupling_indirect_ti3
 - oepdev::TIData, [301](#)
- coupling_total
 - oepdev::TIData, [301](#)
- coupling_trcamm
 - oepdev::TIData, [299](#)
- create_atom_basiset_by_copy
 - The EOPDev Utilities, [74](#)
- create_basiset_by_copy
 - The EOPDev Utilities, [73](#)
- create_superfunctional
 - The EOPDev Utilities, [73](#)
- d_N_n1_n2
 - The Integral Package Library, [57](#)
- DIISManager
 - oepdev::DIISManager, [133](#)
- DMTPole
 - oepdev::DMTPole, [139](#)
- determine_dmtp_convergence_level
 - oepdev::DMTPole, [140](#)
- dmtp
 - oepdev::GenEffPar, [197](#)
- dpol
 - oepdev::GenEffPar, [198](#)
- ESPSolver
 - oepdev::ESPSolver, [171](#)
- empty
 - oepdev::DMTPole, [140](#)
- energy
 - oepdev::DMTPole, [143](#)
- energy_term
 - oepdev::GenEffFrag, [187](#)
- extract_monomer
 - The EOPDev Utilities, [74](#)
- field
 - oepdev::DMTPole, [144](#)
- Field3D
 - oepdev::Field3D, [175](#)
- GramSchmidt
 - oepdev::GramSchmidt, [217](#)
- include/oepdev_files.h, [343](#)
- include/oepdev_options.h, [343](#)
- index
 - oepdev::AllAOIntegralsIterator_2, [98](#)
 - oepdev::AllAOIntegralsIterator_4, [100](#)
- level
 - oepdev::MultipoleConvergence, [228](#)
- main.cc, [344](#)
- make_mdh_D1_coeff
 - The Integral Package Library, [57](#)
- make_mdh_D2_coeff
 - The Integral Package Library, [58](#)

- make_mdh_D2_coeff_explicit_recursion
 - The Integral Package Library, [59](#)
- make_mdh_D3_coeff
 - The Integral Package Library, [59](#)
- make_mdh_R_coeff
 - The Integral Package Library, [60](#)
- make_oeps3d
 - oepdev::OEPotential, [247](#)
- matrix
 - oepdev::GenEffPar, [197](#)
- MultipoleConvergence
 - oepdev::DMTPole, [145](#)
 - oepdev::MultipoleConvergence, [227](#)
- nmo_
 - oepdev::CISComputer, [121](#)
- OEPDevSolver
 - oepdev::OEPDevSolver, [240](#)
- OEPotential
 - oepdev::OEPotential, [246](#)
- OEPotential3D
 - The Three-Dimensional Vector Fields Library, [63](#)
- ObaraSaikaTwoCenterEFPRecursion_New
 - oepdev::ObaraSaikaTwoCenterEFPRecursion_New, [231](#)
- oep
 - oepdev::GenEffPar, [197](#)
- oepdev, [83](#)
 - psi, [94](#)
- oepdev/lib3d/dmtp.h, [345](#)
- oepdev/lib3d/esp.h, [346](#)
- oepdev/libgefp/gefp.h, [346](#)
- oepdev/libints/eri.h, [348](#)
- oepdev/libints/recurr.h, [349](#)
- oepdev/liboep/oep.h, [350](#)
- oepdev/liboep/oep_gdf.h, [351](#)
- oepdev/libpsi/integral.h, [351](#)
- oepdev/libpsi/osrecur.h, [352](#)
- oepdev/libpsi/potential.h, [353](#)
- oepdev/libsolver/solver.h, [353](#)
- oepdev/libsolver/ti_data.h, [354](#)
- oepdev/libtest/test.h, [355](#)
- oepdev/libutil/basis_rotation.h, [355](#)
- oepdev/libutil/cis.h, [358](#)
- oepdev/libutil/davidson_liu.h, [359](#)
- oepdev/libutil/diis.h, [359](#)
- oepdev/libutil/gram_schmidt.h, [360](#)
- oepdev/libutil/integrals_iter.h, [360](#)
- oepdev/libutil/kabsch_superimposer.h, [361](#)
- oepdev/libutil/quambo.h, [362](#)
- oepdev/libutil/scf_perturb.h, [362](#)
- oepdev/libutil/unitary_optimizer.h, [363](#)
- oepdev/libutil/util.h, [364](#)
- oepdev/libutil/wavefunction_union.h, [366](#)
- oepdev::ABCD, [95](#)
- oepdev::AOIntegralsIterator, [106](#)
 - build, [107](#), [108](#)
- oepdev::AbInitioPolarGEFactory, [95](#)
- oepdev::AllAOIntegralsIterator_2, [97](#)
 - AllAOIntegralsIterator_2, [97](#), [98](#)
 - index, [98](#)
- oepdev::AllAOIntegralsIterator_4, [98](#)
 - AllAOIntegralsIterator_4, [99](#), [100](#)
 - index, [100](#)
- oepdev::AllAOShellCombinationsIterator_2, [100](#)
 - AllAOShellCombinationsIterator_2, [101](#), [102](#)
 - compute_shell, [103](#)
- oepdev::AllAOShellCombinationsIterator_4, [103](#)
 - AllAOShellCombinationsIterator_4, [104](#), [105](#)
 - compute_shell, [105](#), [106](#)
- oepdev::CAMM, [108](#)
- oepdev::CISComputer, [114](#)
 - build, [118](#)
 - nmo_, [121](#)
- oepdev::CISData, [121](#)
- oepdev::CPHF, [122](#)
 - CPHF, [126](#)
- oepdev::ChargeTransferEnergyOEPotential, [110](#)
- oepdev::ChargeTransferEnergySolver, [111](#)
 - compute_benchmark, [114](#)
 - compute_oep_based, [114](#)
- oepdev::CubePoints3DIterator, [126](#)
- oepdev::CubePointsCollection3D, [127](#)
- oepdev::DIISManager, [132](#)
 - compute, [133](#)
 - DIISManager, [133](#)
 - put, [133](#)
 - update, [133](#)
- oepdev::DMTPole, [134](#)
 - build, [140](#)

- compute, [142](#), [143](#)
- DMTPole, [139](#)
- determine_dmtp_convergence_level, [140](#)
- empty, [140](#)
- energy, [143](#)
- field, [144](#)
- MultipoleConvergence, [145](#)
- potential, [143](#)
- recenter, [141](#)
- rotate, [141](#)
- superimpose, [142](#)
- oepdev::DavidsonLiu, [128](#)
- oepdev::DoubleGeneralizedDensityFit, [145](#)
 - compute, [147](#)
- oepdev::EETCouplingOEPotential, [148](#)
- oepdev::EETCouplingSolver, [149](#)
 - compute_benchmark, [156](#)
 - compute_oep_based, [155](#)
- oepdev::EFP2_GEFactory, [156](#)
- oepdev::EFPMultipolePotentialInt, [158](#)
- oepdev::ERI_1_1, [164](#)
- oepdev::ERI_2_2, [166](#)
- oepdev::ERI_3_1, [167](#)
- oepdev::ESPSolver, [169](#)
 - ESPSolver, [171](#)
- oepdev::ElectrostaticEnergyOEPotential, [159](#)
- oepdev::ElectrostaticEnergySolver, [160](#)
 - compute_benchmark, [163](#)
 - compute_oep_based, [162](#)
- oepdev::ElectrostaticPotential3D, [163](#)
- oepdev::FFAbInitioPolarGEFactory, [172](#)
- oepdev::Field3D, [173](#)
 - build, [176](#)
 - Field3D, [175](#)
- oepdev::Fourier5, [177](#)
- oepdev::Fourier9, [177](#)
- oepdev::FragmentedSystem, [178](#)
 - build, [179](#)
 - compute_energy, [181](#)
 - compute_energy_term, [181](#)
 - set_auxiliary, [181](#)
 - set_geometry, [180](#)
 - set_primary, [180](#)
- oepdev::GenEffFrag, [182](#)
 - compute_energy, [187](#)
 - compute_energy_term, [187](#)
 - compute_many_body_energy_term, [188](#)
 - energy_term, [187](#)
 - susceptibility, [186](#)
- oepdev::GenEffPar, [188](#)
 - allocate, [196](#)
 - basisset, [198](#)
 - compute_density_matrix, [200](#), [201](#)
 - dmtp, [197](#)
 - dpol, [198](#)
 - matrix, [197](#)
 - oep, [197](#)
 - rotate, [193](#)
 - set_basisset, [195](#)
 - set_dmtp, [194](#)
 - set_dpol, [195](#)
 - set_matrix, [194](#)
 - set_oep, [194](#)
 - set_susceptibility, [195](#)
 - set_vector, [193](#)
 - superimpose, [193](#)
 - susceptibility, [198](#), [199](#)
 - translate, [193](#)
 - vector, [196](#)
- oepdev::GenEffParFactory, [201](#)
 - build, [204](#), [205](#)
- oepdev::GeneralizedDensityFit, [206](#)
 - build, [208](#), [209](#)
 - compute, [209](#)
- oepdev::GeneralizedPolarGEFactory, [209](#)
- oepdev::GeneralizedPolarGEFactory::StatisticalSet, [291](#)
- oepdev::GramSchmidt, [215](#)
 - GramSchmidt, [217](#)
 - projection, [217](#)
- oepdev::IntegralFactory, [218](#)
- oepdev::KabschSuperimposer, [219](#)
 - compute, [220](#), [221](#)
- oepdev::LinearGradientNonUniformEFieldPolarGEFactory, [221](#)
- oepdev::LinearNonUniformEFieldPolarGEFactory, [222](#)
- oepdev::LinearUniformEFieldPolarGEFactory, [223](#)
- oepdev::MultipoleConvergence, [224](#)
 - compute, [227](#)
 - ConvergenceLevel, [226](#)
 - level, [228](#)
 - MultipoleConvergence, [227](#)
 - Property, [226](#)
- oepdev::NonUniformEFieldPolarGEFactory,

- 228
- oepdev::OEP_EFP2_GEFactory, 231
- oepdev::OEPDevSolver, 232
 - build, 240
 - compute_benchmark, 241
 - compute_oep_based, 241
 - OEPDevSolver, 240
- oepdev::OEPTYPE, 249
- oepdev::OEPotential, 242
 - build, 247
 - make_oeps3d, 247
 - OEPotential, 246
- oepdev::OEPotential3D< T >, 248
- oepdev::ObaraSaikaTwoCenterEFPRecursion_New, 229
 - ObaraSaikaTwoCenterEFPRecursion_New, 231
- oepdev::OverlapGeneralizedDensityFit, 250
 - compute, 251
- oepdev::PerturbCharges, 251
- oepdev::Points3DIterator, 252
 - build, 254, 255
 - Points3DIterator, 254
- oepdev::Points3DIterator::Point, 252
- oepdev::PointsCollection3D, 256
 - build, 258
 - PointsCollection3D, 257
- oepdev::PolarGEFactory, 259
- oepdev::PotentialInt, 260
 - PotentialInt, 261, 262
 - set_charge_field, 262
- oepdev::QUAMBOData, 269
- oepdev::QUAMBO, 266
- oepdev::QuadraticGradientNonUniformEFieldPolarGEFactory, 263
- oepdev::QuadraticNonUniformEFieldPolarGEFactory, 264
- oepdev::QuadraticUniformEFieldPolarGEFactory, 265
- oepdev::R_CISComputer, 270
- oepdev::R_CISComputer_Direct, 270
- oepdev::R_CISComputer_DL, 271
- oepdev::R_CISComputer_Explicit, 273
- oepdev::RHFPerturbed, 283
- oepdev::RandomPoints3DIterator, 274
- oepdev::RandomPointsCollection3D, 275
- oepdev::RepulsionEnergyOEPotential, 276
- oepdev::RepulsionEnergySolver, 277
- compute_benchmark, 283
- compute_oep_based, 282
- oepdev::ShellCombinationsIterator, 285
 - ao_iterator, 289
 - build, 288
 - compute_shell, 289
 - ShellCombinationsIterator, 287
- oepdev::SingleGeneralizedDensityFit, 290
 - compute, 291
- oepdev::TIData, 295
 - coupling_direct, 299
 - coupling_direct_coul, 300
 - coupling_direct_exch, 300
 - coupling_indirect, 300
 - coupling_indirect_ti2, 300
 - coupling_indirect_ti3, 301
 - coupling_total, 301
 - coupling_trcamm, 299
 - overlap_corrected, 301
 - overlap_corrected_direct, 302
 - overlap_corrected_indirect, 303
 - v0, 303
- oepdev::TwoBodyAOInt, 304
 - compute, 305
- oepdev::TwoElectronInt, 306
 - compute_shell, 308
- oepdev::U_CISComputer, 308
- oepdev::U_CISComputer_DL, 309
- oepdev::U_CISComputer_Explicit, 310
- oepdev::UniformEFieldPolarGEFactory, 311
- oepdev::UnitaryOptimizer, 312
 - UnitaryOptimizer, 317, 318
- oepdev::UnitaryOptimizer_2, 318
 - UnitaryOptimizer_2, 322, 323
 - oepdev::UnitaryOptimizer_2.1, 323
 - oepdev::UnitaryOptimizer_2.1, 326, 327
 - oepdev::UnitaryOptimizer_2.1, 326, 327
 - UnitaryOptimizer_4.2, 331, 332
- oepdev::UnitaryOptimizer_2.1, 323
- oepdev::UnitaryOptimizer_2.1, 326, 327
- oepdev::UnitaryOptimizer_2.1, 326, 327
- oepdev::UnitaryOptimizer_4.2, 327
- UnitaryOptimizer_4.2, 331, 332
- oepdev::UnitaryTransformedMOPolarGEFactory, 332
- oepdev::WavefunctionUnion, 333
 - Ca_subset, 340
 - Cb_subset, 341
 - WavefunctionUnion, 339
- oepdev::test::Test, 292
- overlap_corrected
 - oepdev::TIData, 301
- overlap_corrected_direct

- oepdev::TIDData, [302](#)
- overlap_corrected_indirect
 - oepdev::TIDData, [303](#)
- Points3DIterator
 - oepdev::Points3DIterator, [254](#)
- PointsCollection3D
 - oepdev::PointsCollection3D, [257](#)
- populate
 - The EOPDev Utilities, [72](#)
- potential
 - oepdev::DMTPole, [143](#)
- PotentialInt
 - oepdev::PotentialInt, [261](#), [262](#)
- projection
 - oepdev::GramSchmidt, [217](#)
- Property
 - oepdev::MultipoleConvergence, [226](#)
- psi, [93](#)
 - oepdev, [94](#)
 - read_options, [93](#)
- put
 - oepdev::DIISManager, [133](#)
- r6
 - The EOPDev Utilities, [71](#)
- read_options
 - psi, [93](#)
- recenter
 - oepdev::DMTPole, [141](#)
- rotate
 - oepdev::DMTPole, [141](#)
 - oepdev::GenEffPar, [193](#)
- set_auxiliary
 - oepdev::FragmentedSystem, [181](#)
- set_basisset
 - oepdev::GenEffPar, [195](#)
- set_charge_field
 - oepdev::PotentialInt, [262](#)
- set_dmtip
 - oepdev::GenEffPar, [194](#)
- set_dpole
 - oepdev::GenEffPar, [195](#)
- set_geometry
 - oepdev::FragmentedSystem, [180](#)
- set_matrix
 - oepdev::GenEffPar, [194](#)
- set_oep
 - oepdev::GenEffPar, [194](#)
- set_primary
 - oepdev::FragmentedSystem, [180](#)
- set_susceptibility
 - oepdev::GenEffPar, [195](#)
- set_vector
 - oepdev::GenEffPar, [193](#)
- ShellCombinationsIterator
 - oepdev::ShellCombinationsIterator, [287](#)
- solve_scf
 - The EOPDev Utilities, [75](#)
- solve_scf_sad
 - The EOPDev Utilities, [75](#)
- superimpose
 - oepdev::DMTPole, [142](#)
 - oepdev::GenEffPar, [193](#)
- susceptibility
 - oepdev::GenEffFrag, [186](#)
 - oepdev::GenEffPar, [198](#), [199](#)
- The Density Functional Theory Library, [65](#)
- The EOPDev Solver Library, [49](#)
- The EOPDev Testing Platform Library, [81](#)
- The EOPDev Utilities, [66](#)
 - _calculate_DFI_Vel, [78](#)
 - ao_rotation_matrix, [73](#)
 - average_moment, [76](#)
 - bs_optimize_projection, [79](#)
 - calculate_DFI_Vel_JK, [78](#)
 - calculate_DFI_Vel_J, [78](#)
 - calculate_JK_r, [77](#)
 - calculate_JK, [77](#)
 - calculate_OEP_basisopt_V, [79](#)
 - compute_distance, [74](#)
 - create_atom_basisset_by_copy, [74](#)
 - create_basisset_by_copy, [73](#)
 - create_superfunctional, [73](#)
 - extract_monomer, [74](#)
 - populate, [72](#)
 - r6, [71](#)
 - solve_scf, [75](#)
 - solve_scf_sad, [75](#)
- The Generalized Effective Fragment Potentials Library, [50](#)
- The Generalized One-Electron Potentials Library, [47](#)
- The Integral Package Library, [52](#)
 - d_N_n1_n2, [57](#)
 - make_mdh_D1_coeff, [57](#)

- make_mdh_D2_coeff, [58](#)
- make_mdh_D2_coeff_explicit_recursion, [59](#)
- make_mdh_D3_coeff, [59](#)
- make_mdh_R_coeff, [60](#)
- The Three-Dimensional Vector Fields Library,
[62](#)
- OEPotential3D, [63](#)
- translate
 - oepdev::GenEffPar, [193](#)
- UnitaryOptimizer
 - oepdev::UnitaryOptimizer, [317](#), [318](#)
- UnitaryOptimizer_2
 - oepdev::UnitaryOptimizer_2, [322](#), [323](#)
- UnitaryOptimizer_2_1
 - oepdev::UnitaryOptimizer_2_1, [326](#), [327](#)
- UnitaryOptimizer_4_2
 - oepdev::UnitaryOptimizer_4_2, [331](#), [332](#)
- update
 - oepdev::DIISManager, [133](#)
- v0
 - oepdev::TIData, [303](#)
- vector
 - oepdev::GenEffPar, [196](#)
- WavefunctionUnion
 - oepdev::WavefunctionUnion, [339](#)