GHF

Release 0.1

Xeno De Vriendt

Apr 26, 2020

CONTENTS:

CHAPTER

ONE

RESTRICTED HARTREE FOCK, BY MEANS OF SCF PROCEDURE

This class is used to calculate the RHF energy of a given molecule and the number of electrons. The molecule has to be created in pySCF: molecule = gto.M(atom = geometry, spin = diff. in alpha and beta electrons, basis = basis set)

class hf.HartreeFock.RHF.**MF**(*molecule*, *number_of_electrons*, *int_method='pyscf'*)

Input is a molecule and the number of electrons.

Molecules are made in pySCF and calculations are performed as follows, eg.: The following snippet prints and returns RHF energy of h_2 and the number of iterations needed to get this value.

```
>>> from hf.HartreeFock import *
>>> h_2 = gto.M(atom = 'h 0 0 0; h 0 0 1', spin = 0, basis = 'sto-3g')
>>> x = RHF.MF(h_2, 2)
>>> x.get_scf_solution()
```

diis(*convergence=1e-12*, *complex_method=False*)

When needed, DIIS can be used to speed up the RHF calculations by reducing the needed iterations.

Parameters

- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns scf energy, number of iterations, mo coefficients, last density matrix, last fock matrix

get_dens(*i=-1*)

Returns the (last) density matrix.

Returns The last density matrix.

get_fock(*i=-1*)

Returns the (last) fock matrix.

Returns The last Fock matrix.

get_fock_orth(*i=-1*)

Returns the (last) fock matrix in the orthonormal basis.

Returns The last Fock matrix.

get_mo_coeff(*i=-1*)

Returns mo coefficients.

Returns The mo coefficients

get_mo_energy()

Returns the MO energies. :return: an array of MO energies.

get_one_e()

Returns The one electron integral matrix: T + V

get_ovlp()

Returns The overlap matrix

get_scf_solution(*convergence=1e-12*, *complex_method=False*)
Prints the number of iterations and the converged scf energy.

Parameters

- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns the converged energy

get_scf_solution_diis(*convergence=1e-12*, *complex_method=False*)
Prints the number of iterations and the converged DIIS energy. The number of iterations will be lower than with a normal scf, but the energy value will be the same. Example:

```
>>> from hf.HartreeFock import *
>>> h2 = gto.M(atom = 'h 0 0 0; h 1 0 0', basis = 'cc-pvdz')
>>> x = RHF.MF(h2, 2)
>>> x.get_scf_solution_diis()
```

Parameters

- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns The converged scf energy, using DIIS.

get_two_e()

Returns The electron repulsion interaction tensor

nuc_rep()

Returns The nuclear repulsion value

scf(*convergence=1e-12*, *complex_method=False*)
Performs a self consistent field calculation to find the lowest RHF energy.

Parameters

- **convergence** – Convergence criterion. If none is specified, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns number of iterations, scf energy, mo coefficients, last density matrix, last fock matrix

stability_analysis(*method*)
Internal stability analysis to verify whether the wave function is stable within the space of the used method. :param method: Indicate whether you want to check the internal or external stability of the wave function. Can be internal or external. :param step_size: Step size for orbital rotation. standard is 1e-4. :return: In case of internal stability analysis, it returns a new set of coefficients.

CHAPTER

TWO

UNRESTRICTED HARTREE FOCK, BY MEANS OF SCF PROCEDURE

This class is used to calculate the UHF energy for a given molecule and the number of electrons of that molecule. Several options are available to make sure you get the lowest energy from your calculation, as well as some useful functions to get intermediate values such as MO coefficients, density and fock matrices.

class hf.HartreeFock.UHF.**MF**(*molecule*, *number_of_electrons*, *int_method='pyscf'*)

Input is a molecule and the number of electrons.

Molecules are made in pySCF and calculations are performed as follows, eg.: The following snippet prints and returns UHF energy of h_3 and the number of iterations needed to get this value.

For a normal scf calculation your input looks like the following example:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1, basis␣
↪= 'cc-pvdz')
>>> x = UHF.MF(h3, 3)
>>> x.get_scf_solution()
```

diis(*initial_guess=None*, *convergence=1e-12*, *complex_method=False*)

When needed, DIIS can be used to speed up the UHF calculations by reducing the needed iterations.

Parameters

- **initial_guess** – Initial guess for the scf procedure. None specified: core Hamiltonian.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns scf energy, number of iterations, mo coefficients, last density matrix, last fock matrix

extra_electron_guess()

This method adds two electrons to the system in order to get coefficients that can be used as a better guess for the scf procedure. This essentially forces the system into it's <S_z> = 0 state.

!!!IMPORTANT!!! Only supported with pyscf.

To perform a calculation with this method, you will have to work as follows:

```
>>> from hf.HartreeFock import *
>>> h4 = gto.M(atom = 'h 0 0 0; h 1 0 0; h 0 1 0; h 1 1 0' , spin = 2, basis␣
↪= 'cc-pvdz')
>>> x = UHF.MF(h4, 4)
>>> guess = x.extra_electron_guess()
>>> x.get_scf_solution(guess)
```

Returns A new guess matrix to use for the scf procedure.

get_dens(*i=-1*)
Gets the (last) density matrix of the converged solution. Alpha density in the first matrix, beta density in the second.

Returns The last density matrix.

get_fock(*i=-1*)
Gets the (last) fock matrix of the converged solution. Alpha Fock matrix first, beta Fock matrix second.

Returns The requested Fock matrices.

get_fock_orth(*i=-1*)
Get the fock matrices in the orthonormal basis. Defaults to the last one :param i: index of the matrix you want. :return: a and b orthonormal fock matrix

get_hessian(*prime*)
Get the Hessian matrix after performing a stability analysis. :param: specify whether you want to look at H' or H" by putting 1 or 2 :return: The hessian matrix

get_mo_coeff(*i=-1*)
Gets the mo coefficients of the converged solution. Alpha coefficients in the first matrix, beta coefficients in the second.

Returns The mo coefficients

get_mo_energy()
Returns the MO energies of the converged solution. :return: an array of MO energies.

get_one_e()

Returns The one electron integral matrix: T + V

get_ovlp()

Returns The overlap matrix

get_scf_solution(*guess=None*, *convergence=1e-12*, *complex_method=False*)
Prints the number of iterations and the converged scf energy. Also prints the expectation value of S_z, S^2 and the multiplicity.

Parameters

- **guess** – The initial guess for the scf procedure. If none is given: core Hamiltonian.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns The converged scf energy.

get_scf_solution_diis(*guess=None*, *convergence=1e-12*, *complex_method=False*)
Prints the number of iterations and the converged diis energy. Also prints the expectation value of S_z, S^2 and the multiplicity.

Parameters

- **guess** – The initial guess. If none is specified, core Hamiltonian.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns The converged diis energy.

get_two_e()

Returns The electron repulsion interaction tensor

nuc_rep()

Returns The nuclear repulsion value

scf(*initial_guess=None*, *convergence=1e-12*, *complex_method=False*)

Performs a self consistent field calculation to find the lowest UHF energy.

Parameters

- **initial_guess** – A tuple of an alpha and beta guess matrix. If none, the core hamiltonian will be used.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns The scf energy, number of iterations, the mo coefficients, the last density and the last fock matrices

stability_analysis(*method*)

Internal stability analysis to verify whether the wave function is stable within the space of the used method. :param method: Indicate whether you want to check the internal or external stability of the wave function. Can be internal or external. :return: In case of internal stability analysis, it returns a new set of coefficients.

CHAPTER

THREE

CONSTRAINED UNRESTRICTED HARTREE FOCK BY SCUSERIA

This class is used to calculate the ROHF energy for a given molecule and the number of electrons of that molecule, using a constrained version of unrestricted Hartree Fock, according to Scuseria.. Several options are available to make sure you get the lowest energy from your calculation, as well as some useful functions to get intermediate values such as MO coefficients, density and fock matrices.

class `hf.HartreeFock.cUHF_s.`**`MF`**(*molecule*, *number_of_electrons*, *int_method='pyscf'*)

Input is a molecule and the number of electrons.

Molecules are made in pySCF/psi4 and calculations are performed as follows, eg.: The following snippet prints and returns UHF energy of h_3 and the number of iterations needed to get this value.

For a normal scf calculation your input looks like the following example:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1, basis␣
↪= 'cc-pvdz')
>>> x = cUHF_s.MF(h3, 3)
>>> x.get_scf_solution()
```

`get_dens`(*i=-1*)

Gets the last density matrix of the converged solution. Alpha density in the first matrix, beta density in the second.

Returns The last density matrix.

`get_fock`(*i=-1*)

Gets the last fock matrix of the converged solution. Alpha Fock matrix first, beta Fock matrix second.

Returns The last Fock matrix.

`get_mo_coeff`(*i=-1*)

Gets the mo coefficients of the converged solution. Alpha coefficients in the first matrix, beta coefficients in the second.

Returns The mo coefficients

`get_one_e`()

Returns The one electron integral matrix: T + V

`get_ovlp`()

Returns The overlap matrix

`get_scf_solution`(*guess=None*, *convergence=1e-12*, *diis=True*)

Prints the number of iterations and the converged scf energy. Also prints the expectation value of S_z, S^2 and the multiplicity.

Parameters

- **guess** – Initial scf guess
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **diis** – Accelerates the convergence, default is true.

Returns The converged scf energy.

get_two_e()

Returns The electron repulsion interaction tensor

nuc_rep()

Returns The nuclear repulsion value

random_guess()

A function that creates a matrix with random values that can be used as an initial guess for the SCF calculations.

To use this guess:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = cUHF_s.MF(h3, 3)
>>> guess = x.random_guess()
>>> x.get_scf_solution(guess)
```

Returns A random unitary matrix.

scf(*initial_guess=None*, *convergence=1e-12*, *diis=True*)

Performs a self consistent field calculation to find the lowest UHF energy.

Parameters

- **initial_guess** – Random initial guess, if none is given the Core Hamiltonian is used.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **diis** – Accelerates the convergence, default is true.

Returns The scf energy, number of iterations, the mo coefficients, the last density and the last fock matrices

CHAPTER

FOUR

CONSTRAINED UNRESTRICTED HARTREE FOCK BY P. BULTINCK

This class is used to calculate the ROHF energy for a given molecule and the number of electrons of that molecule, using a constrained version of unrestricted Hartree Fock. This constraint is an idea from professor P. Bultinck, where the alpha and beta MO's are made equal for the closed shell part of the system. Several options are available to make sure you get the lowest energy from your calculation, as well as some useful functions to get intermediate values such as MO coefficients, density and fock matrices.

class hf.HartreeFock.cUHF_b.**MF**(*molecule*, *number_of_electrons*, *int_method='pyscf'*)

Input is a molecule and the number of electrons.

Molecules are made in pySCF/psi4 and calculations are performed as follows, eg.: The following snippet prints and returns UHF energy of h_3 and the number of iterations needed to get this value.

For a normal scf calculation your input looks like the following example:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1, basis␣
↪= 'cc-pvdz')
>>> x = cUHF_b.MF(h3, 3)
>>> x.get_scf_solution()
```

diis(*initial_guess=None*, *convergence=1e-12*)

When needed, DIIS can be used to speed up the UHF calculations by reducing the needed iterations.

Parameters

- **initial_guess** – Initial guess for the scf procedure. None specified: core Hamiltonian.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns scf energy, number of iterations, mo coefficients, last density matrix, last fock matrix

get_constrained_mo(*i=-1*)

Gets the constrained mo coefficients of the converged solution. Alpha coefficients in the first matrix, beta coefficients in the second.

Parameters i – Iteration of which the mo coefficients are given. If None, the ones of the last iteration are given.

Returns The (last) mo coefficients

get_dens(*i=-1*)

Gets the last density matrix of the converged solution. Alpha density in the first matrix, beta density in the second.

Parameters i – Iteration of which the densities are given. If None, the ones of the last iteration are given.

Returns The (last) density matrix.

get_fock(*i=-1*)
Gets the fock matrix of the converged solution. Alpha Fock matrix first, beta Fock matrix second.

Parameters **i** – Iteration of which the focks are given. If None, the ones of the last iteration are given.

Returns The (last) Fock matrix.

get_last_dens()
Gets the last density matrix of the converged solution. Alpha density in the first matrix, beta density in the second.

Returns The last density matrix.

get_last_fock()
Gets the last fock matrix of the converged solution. Alpha Fock matrix first, beta Fock matrix second.

Returns The last Fock matrix.

get_mo(*i=-1*)
Gets the mo coefficients of the converged solution. Alpha coefficients in the first matrix, beta coefficients in the second.

Parameters **i** – Iteration of which the mo coefficients are given. If None, the ones of the last iteration are given.

Returns The (last) mo coefficients

get_mo_coeff()
Gets the mo coefficients of the converged solution. Alpha coefficients in the first matrix, beta coefficients in the second.

Returns The mo coefficients

get_one_e()

Returns The one electron integral matrix: T + V

get_orth_fock(*i=-1*)
Gets the orthonormal fock matrix of the converged solution. Alpha Fock matrix first, beta Fock matrix second.

Parameters **i** – Iteration of which the focks are given. If None, the ones of the last iteration are given.

Returns The (last) orthonormal Fock matrix.

get_ovlp()

Returns The overlap matrix

get_scf_solution(*guess=None*, *convergence=1e-12*)
Prints the number of iterations and the converged scf energy. Also prints the expectation value of S_z, S^2 and the multiplicity.

Parameters

- **guess** – Initial scf guess
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns The converged scf energy.

get_scf_solution_diis(*guess=None*, *convergence=1e-12*)

Prints the number of iterations and the converged diis energy. Also prints the expectation value of S_z, S^2 and the multiplicity.

Parameters

- **guess** – The initial guess. If none is specified, core Hamiltonian.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns The converged diis energy.

get_two_e()

Returns The electron repulsion interaction tensor

nuc_rep()

Returns The nuclear repulsion value

random_guess()

A function that creates a matrix with random values that can be used as an initial guess for the SCF calculations.

To use this guess:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = cUHF_b(h3, 3)
>>> guess = x.random_guess()
>>> x.get_scf_solution(guess)
```

Returns A random hermitian matrix.

scf(*initial_guess=None*, *convergence=1e-12*)

Performs a self consistent field calculation to find the lowest UHF energy.

Parameters

- **initial_guess** – Set the convergence criterion. If none is given, 1e-12 is used.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns The scf energy, number of iterations, the mo coefficients, the last density and the last fock matrices

CHAPTER

FIVE

GENERALISED HARTREE FOCK, BY MEANS OF SCF PROCEDURE

This class creates a generalised Hartree-Fock object which can be used for scf calculations. Different initial guesses are provided as well as the option to perform a stability analysis. The molecule has to be created in pySCF: molecule = gto.M(atom = geometry, spin = diff. in alpha and beta electrons, basis = basis set)

class hf.HartreeFock.GHF.MF(*molecule*, *number_of_electrons*, *int_method='pyscf'*)

Input is a molecule and the number of electrons.

Molecules are made in pySCF and calculations are performed as follows, eg.: The following snippet prints and returns UHF energy of h3 and the number of iterations needed to get this value.

For a normal scf calculation your input looks like the following example:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1, basis␣
↪= 'cc-pvdz')
>>> x = GHF.MF(h3, 3)
>>> x. get_scf_solution()
```

diis(*guess=None*, *convergence=1e-12*, *complex_method=False*)

The DIIS method is an alternative to the standard scf procedure. It reduces the number of iterations needed to find a solution. The same guesses can be used as for a standard scf calculation. Stability analysis can be done as well.

Parameters

- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **guess** – The initial guess matrix, if none is specified, the spin blocked core Hamiltonian is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns scf_energy, iterations, mo coefficients, last density matrix & last Fock matrix

get_hessian()

After stability analysis is performed, the hessian is stored and can be used for further studying. :return: The Hessian matrix

get_last_dens()

Gets the last density matrix of the converged solution.

Returns The last density matrix.

get_last_fock()

Gets the last fock matrix of the converged solution.

Returns The last Fock matrix.

get_mo_coeff()
Gets the mo coefficients of the converged solution.

Returns The mo coefficients

get_one_e()

Returns The one electron integral matrix: T + V

get_ovlp()

Returns The overlap matrix

get_scf_solution(*guess=None*, *convergence=1e-12*, *complex_method=False*)
Prints the number of iterations and the converged scf energy.

Parameters

- **guess** – Initial guess for scf. If none is specified: expanded core Hamiltonian.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns The converged scf energy.

get_scf_solution_diis(*guess=None*, *convergence=1e-12*, *complex_method=False*)
Prints the number of iterations and the converged energy after a diis calculation. Guesses can also be specified just like with a normal scf calculation.

Example:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = GHF.MF(h3, 3)
>>> guess = x.random_guess()
>>> x.get_scf_solution_diis(guess)
```

Parameters

- **guess** – Initial guess for scf. None specified: expanded core Hamiltonian
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.cd

Returns The converged scf energy.

get_two_e()

Returns The electron repulsion interaction tensor

nuc_rep()

Returns The nuclear repulsion value

random_guess()
A function that creates a matrix with random values that can be used as an initial guess for the SCF calculations.

To use this guess:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = GHF.MF(h3, 3)
>>> guess = x.random_guess()
>>> x.get_scf_solution(guess)
```

Returns A random hermitian matrix.

scf(*guess=None*, *convergence=1e-12*, *complex_method=False*)

This function performs the SCF calculation by using the generalised Hartree-Fock formulas. Since we're working in the real class, all values throughout are real. For complex, see the "complex_GHF" class.

Parameters

- **guess** – Initial guess to start SCF. If none is given, core hamiltonian will be used.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **complex_method** – Specify whether or not you want to work in the complex space. Default is real.

Returns scf_energy, iterations, mo coefficients, last density matrix & last Fock matrix

stability_analysis(*method*, *step_size=0.0001*)

Internal stability analysis to verify whether the wave function is stable within the space of the used method. :param method: Indicate whether you want to check the internal or external stability of the wave function. Can be internal or external. :param step_size: Step size for orbital rotation. standard is 1e-4. :return: In case of internal stability analysis, it returns a new set of coefficients.

unitary_rotation_guess(*init=None*)

A function that creates an initial guess matrix by performing a unitary transformation on the core Hamiltonian matrix.

To use this guess:

```
>>> from hf.HartreeFock import *
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = GHF.MF(h3, 3)
>>> guess = x.unitary_rotation_guess()
>>> x.get_scf_solution(guess)
```

Returns A rotated guess matrix.

CHAPTER

SIX

USEFUL FUNCTIONS FOR SCF PROCEDURE

A number of functions used throughout the UHF and RHF calculations are summarised here.

`hf.utilities.SCF_functions.`**`calc_mo`**(*f_o*, *t*)

Calculate the mo coefficients. :param f_o: Fock matrix in orthonormal basis. :param t: Transformation matrix. :return: mo coefficients

`hf.utilities.SCF_functions.`**`calc_mo_e`**(*f_o*)

Calculate mo energies. :param f_o: Fock matrix in orthonormal basis. :return: mo energies

`hf.utilities.SCF_functions.`**`density_matrix`**(*f_matrix*, *occ*, *trans*)

- density() creates a density matrix from a fock matrix and the number of occupied orbitals.
- Input is a fock matrix, the number of occupied orbitals, which can be separate for alpha and beta in case of UHF. And a transformation matrix X.

`hf.utilities.SCF_functions.`**`get_integrals_psi4`**(*mol*)

A function to calculate your integrals & nuclear repulsion with psi4.

Parameters **`mol`** – Psi4 instance

Returns overlap, core hamiltonian, eri tensor and nuclear repulsion

`hf.utilities.SCF_functions.`**`get_integrals_pyscf`**(*molecule*)

A function to calculate your integrals & nuclear repulsion with pyscf.

`hf.utilities.SCF_functions.`**`trans_matrix`**(*overlap*)

- Define a transformation matrix X, used to orthogonalize different matrices throughout the calculation.
- Input should be an overlap matrix.

`hf.utilities.SCF_functions.`**`uhf_fock_matrix`**(*density_matrix_1*, *density_matrix_2*, *one_electron*, *two_electron*)

- calculate a fock matrix from a given alpha and beta density matrix
- fock alpha if 1 = alpha and 2 = beta and vice versa
- input is the density matrix for alpha and beta, a one electron matrix and a two electron tensor.

`hf.utilities.SCF_functions.`**`uhf_scf_energy`**(*density_matrix_a*, *density_matrix_b*, *fock_a*, *fock_b*, *one_electron*)

- calculate the scf energy value from a given density matrix and a given fock matrix for both alpha and beta, so 4 matrices in total.
- then calculate the initial electronic energy and put it into an array
- input is the density matrices for alpha and beta, the fock matrices for alpha and beta and lastly a one electron matrix.

CHAPTER

SEVEN

FUNCTIONS TO CALCULATE SPIN EXPECTATION VALUES

This file contains functions that calculate the expectation values of the different spin operators.

`hf.properties.spin.ghf(coeff, n_e, trans)`

A function used to calculate the spin expectation values in the generalised hartree fock formalism.

Parameters

- **`coeff`** – The generalised MO coefficients
- **`n_e`** – number of electrons
- **`trans`** – transformation matrix, eg.: S^(-1/2)

Returns The expectation values of S_z, S^2 and the multiplicity (2S+1)

`hf.properties.spin.uhf(occ_a, occ_b, coeff_a, coeff_b, overlap)`

A function used to calculate the spin expectation values in the unrestricted hartree fock formalism.

Parameters

- **`occ_a`** – number of occupied alpha orbitals
- **`occ_b`** – number of occupied beta orbitals
- **`coeff_a`** – MO coefficients of alpha orbitals
- **`coeff_b`** – MO coefficients of beta orbitals
- **`overlap`** – overlap matrix of the molecule

Returns S^2, S_z and spin multiplicity

CHAPTER

EIGHT

FUNCTIONS TO DEAL WITH MATRIX AND TENSOR TRANSFORMATIONS.

This file contains functions that calculate the expectation values of the different spin operators.

`hf.utilities.transform.`**`expand_matrix`**`(`*matrix*`)`

Expand a matrix to spinor basis.

Parameters **`matrix`** – a matrix.

Returns a matrix double the size, where blocks of zero's are added top right and bottom left.

`hf.utilities.transform.`**`expand_tensor`**`(`*tensor*, *complexity=False*`)`

Expand a given tensor to spinor basis representation.

Parameters

- **`tensor`** – The tensor, usually eri, that you wish to expand.
- **`complexity`** – Is your tensor complex or not? Default is false.

Returns a tensor where each dimension is doubled.

`hf.utilities.transform.`**`mix_tensor_to_basis_transform`**`(`*tensor*, *matrix1*, *matrix2*, *matrix3*, *matrix4*`)`

Transform a tensor to a mixed basis. Each index can have a separate transformation matrix. Mostly useful in UHF calculations where tensors must be transformed to an alpha-beta basis. :param tensor: The tensor you wish to transform :param matrix1: matrix to transform index 1 :param matrix2: matrix to transform index 2 :param matrix3: matrix to transform index 3 :param matrix4: matrix to transform index 4 :return: The transformed tensor in the mixed basis.

`hf.utilities.transform.`**`rotate_to_eigenvec`**`(`*eigenvec*, *mo_coeff*, *occ*, *number_of_orbitals*`)`

A function used to rotate a given set of coefficients to a given eigenvector. This is done by making an irreducible representation of the eigenvector and creating the exponential matrix of the result. :param eigenvec: The eigenvector to which you wish to rotate :param mo_coeff: The MO coefficients you whish to rotate :param occ: the number of occupied orbitals :param number_of_orbitals: the total number of orbitals :return: A rotated set of coefficients.

`hf.utilities.transform.`**`spin_blocked`**`(`*block_1*, *block_2*, *block_3*, *block_4*`)`

When creating the blocks of the density or fock matrix separately, this function is used to add them together, and create the total density or Fock matrix in spin Blocked notation. Transforms four separate matrices to 1 big matrix in spin-blocked notation.

Returns a density matrix in the spin-blocked notation

`hf.utilities.transform.`**`tensor_basis_transform`**`(`*tensor*, *matrix*`)`

Transform the two electron tensor to MO basis. Scales as N^5.

Parameters

- **`tensor`** – A tensor in it’s initial basis.
- **`matrix`** – the transformation matrix.

Returns The tensor in the basis of the transformation matrix.

PYTHON MODULE INDEX

INDEX