GHF

Release 0.1

Xeno De Vriendt

Dec 05, 2019

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 ghf.RHF.**RHF**(*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.

```
>>> h_2 = gto.M(atom = 'h 0 0 0; h 0 0 1', spin = 0, basis = 'sto-3g')
>>> x = RHF(h_2, 2)
>>> x.get_scf_solution()
Number of iterations: 2
Converged SCF energy in Hartree: -1.0661086493179357 (RHF)
```

diis(*convergence=1e-12*)

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.

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

get_last_dens()

Returns the last density matrix of the converged solution.

Returns The last density matrix.

get_last_fock()

Returns the last fock matrix of the converged solution.

Returns The last Fock matrix.

get_mo_coeff()

Returns 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(*convergence=1e-12*)

Prints the number of iterations and the converged scf energy.

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

Returns the converged energy

get_scf_solution_diis(*convergence=1e-12*)

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:

```
>>> h2 = gto.M(atom = 'h 0 0 0; h 1 0 0', basis = 'cc-pvdz')
>>> x = RHF(h2, 2)
>>> x.get_scf_solution_diis()
Number of iterations: 9
Converged SCF energy in Hartree: -1.100153764878446 (RHF)
```

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

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*)

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

Parameters **convergence** – Convergence criterion. If none is specified, 1e-12 is used.

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

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 usefull functions to get intermediate values such as MO coefficients, density and fock matrices.

class ghf.UHF.**UHF**(*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:

```
>>> 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(h3, 3)
>>> x.get_scf_solution()
Number of iterations: 47
Converged SCF energy in Hartree: -1.506274320261134 (UHF)
<S^2> = 0.7735672504295973, <S_z> = 0.5, Multiplicity = 2.023430009098014
```

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

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:

```
>>> 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(h4, 4)
>>> guess = x.extra_electron_guess()
>>> x.get_scf_solution(guess)
Number of iterations: 60
```

(continues on next page)

(continued from previous page)

```
Converged SCF energy in Hartree: -2.0210882477030547 (UHF)
<S^2> = 1.0565277001056579, <S_z> = 0.0, Multiplicity = 2.2860688529487976
```

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

get_hessian()
Get the Hessian matrix after performing a stability analysis. :return: The hessian 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_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_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** – 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.

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

internal_stability_analysis(*step_size=1e-05*)
Perform an internal stability analysis on the UHF wave function. This will determine whether or not the wave function is stable within the real UHF method. If there is an instability, the MO coefficients will be

rotated in the direction of the lowest eigenvector of the (A' + B') part of the Hessian, resulting in improved MO's and hopefully a stable wave function.

Returns New and improved MO's

nuc_rep()

Returns The nuclear repulsion value

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

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

stability()
Performing a stability analysis checks whether or not the wave function is stable, by checking the lowest eigen- value of the Hessian matrix. If there's an instability, the MO's will be rotated in the direction of the lowest eigenvalue. These new MO's can then be used to start a new scf procedure.

To perform a stability analysis, use the following syntax:

```
>>> 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(h4, 4)
>>> guess = x.stability()
>>> x.get_scf_solution(guess)
There is an internal instability in the UHF wave function.
Number of iterations: 66
Converged SCF energy in Hartree: -2.0210882477030716 (UHF)
<S^2> = 1.056527700105677, <S_z> = 0.0, Multiplicity = 2.2860688529488145
```

Returns New and improved MO's.

CHAPTER

THREE

REAL 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 ghf.real_GHF.**RealGHF**(*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:

```
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1, basis␣
↪= 'cc-pvdz')
>>> x = RealGHF(h3, 3)
>>> x. get_scf_solution()
Number of iterations: 81
Converged SCF energy in Hartree: -1.5062743202607725 (Real GHF)
```

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

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.

Returns scf_energy, iterations, mo coefficients, last density matrix & last Fock 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*)
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.

Returns The converged scf energy.

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

```
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = RealGHF(h3, 3)
>>> guess = x.random_guess()
>>> x.get_scf_solution_diis(guess)
Number of iterations: 23
Converged SCF energy in Hartree: -1.5062743202915496 (Real GHF)
```

Without DIIS, 81 iterations are needed to find this solution.

Parameters

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

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:

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

Returns A random hermitian matrix.

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

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.

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

stability()

Performing a stability analysis checks whether or not the wave function is stable, by checking the lowest eigenvalue of the Hessian matrix. If there's an instability, the MO's will be rotated in the direction of the lowest eigenvalue. These new MO's can then be used to start a new scf procedure.

To perform a stability analysis, use the following syntax, this will continue the analysis until there is no more instability:

```
>>> 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 = RealGHF(h4, 4)
>>> x.scf()
>>> guess = x.stability()
>>> while x.instability:
>>>     new_guess = x.stability()
>>>     x.get_scf_solution(new_guess)
```

Returns New and improved MO's.

unitary_rotation_guess()

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

To use this guess:

```
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = RealGHF(h3, 3)
>>> guess = x.unitary_rotation_guess()
>>> x.get_scf_solution(guess)
```

Returns A rotated guess matrix.

CHAPTER

FOUR

COMPLEX 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 ghf.complex_GHF.**ComplexGHF**(*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:

```
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1, basis␣
↪= 'cc-pvdz')
>>> x = ComplexGHF(h3, 3)
>>> x.loop_calculations()
```

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

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

- **guess** – The initial guess matrix, if none is specified: expanded core Hamiltonian unitarily rotated.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns scf_energy, iterations, mo coefficients, last density matrix & last Fock 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*)
Prints the number of iterations and the converged scf energy.

Parameters

- **guess** – The initial scf guess. None specified: core Hamiltonian unitarily rotated with complex U matrix.
- **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 energy after a diis calculation. Guesses can also be specified just like with a normal scf calculation.

Example:

```
>>> h3 = gto.M(atom = 'h 0 0 0; h 0 0.86602540378 0.5; h 0 0 1', spin = 1,␣
↪basis = 'cc-pvdz')
>>> x = ComplexGHF(h3, 3)
>>> guess = x.random_guess()
>>> x.get_scf_solution_diis(guess)
```

Parameters

- **guess** – Initial scf guess. None specified: core HAmiltonian unitarily rotated with complex U matrix.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns The converged scf energy.

get_two_e()

Returns The electron repulsion interaction tensor

loop_calculations(*number_of_loops*, *guess=None*, *convergence=1e-12*)
This function is specifically catered to the random guess method. Since it is hard to predict the seed of the correct random matrix, a simple solution is to repeat the scf calculation a certain number of times, starting from different random guesses and returning the lowest value of all the different calculations. The loops will automatically perform a stability analysis until there is no more instability in the wave function.

Parameters

- **number_of_loops** – The amount of times you want to repeat the scf + stability procedure.
- **guess** – The guess used for the scf procedure.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns The scf energy after the loops.

loop_calculations_diis(*number_of_loops*, *guess=None*, *convergence=1e-12*)
This function is specifically catered to the random guess method. Since it is hard to predict the seed of the correct random matrix, a simple solution is to repeat the scf calculation a certain number of times, starting from different random guesses and returning the lowest value of all the different calculations. The loops

will automatically perform a stability analysis until there is no more instability in the wave function. This option uses the DIIS iteration so that convergence is generally reached faster.

Parameters

- **number_of_loops** – The amount of times you want to repeat the DIIS + stability procedure.
- **guess** – The guess used for the DIIS procedure.
- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.

Returns The energy after the loops.

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.

IMPORTANT: It is recommended to use a random guess since the results are significantly better than those found when using the standard guess.

To use this guess:

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

Returns A random hermitian matrix.

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

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

Parameters

- **convergence** – Set the convergence criterion. If none is given, 1e-12 is used.
- **guess** – Initial guess for scf. If none is given, a unitary rotation on the core Hamiltonian is used.

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

stability()

Performing a stability analysis checks whether or not the wave function is stable, by checking the lowest eigenvalue of the Hessian matrix. If there's an instability, the MO's will be rotated in the direction of the lowest eigenvalue. These new MO's can then be used to start a new scf procedure.

To perform a stability analysis, use the following syntax, this will continue the analysis until there is no more instability:

```
>>> 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 = ComplexGHF(h4, 4)
>>> x.scf()
>>> guess = x.stability()
>>> while x.instability:
```

(continues on next page)

(continued from previous page)

```
>>>     new_guess = x.stability()
>>>     x.get_scf_solution(new_guess)
```

Returns New and improved MO's.

CHAPTER

FIVE

USEFUL FUNCTIONS FOR SCF PROCEDURE

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

ghf.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.

ghf.SCF_functions.**expand_matrix**(*matrix*)

Parameters **matrix** –

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

ghf.SCF_functions.**get_integrals_psi4**(*mol*)
A function to calculate your integrals & nuclear repulsion with psi4. :param mol: Psi4 instance :return: overlap, core hamiltonian, eri tensor and nuclear repulsion

ghf.SCF_functions.**get_integrals_pyscf**(*molecule*)
A function to calculate your integrals & nuclear repulsion with pyscf.

ghf.SCF_functions.**spin**(*occ_a*, *occ_b*, *coeff_a*, *coeff_b*, *overlap*)

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

ghf.SCF_functions.**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. :return: a density matrix in the spin-blocked notation

ghf.SCF_functions.**trans_matrix**(*overlap*)

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

ghf.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.

ghf.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

SIX

TESTING THE RHF AND UHF METHODS

Simple tests to check whether or not the functions return the correct value.

ghf.tests.test_auth.**test_RHF**()
: test_RHF will test whether or not the RHF method returns the wanted result. The accuracy is 10^-11.

ghf.tests.test_auth.**test_UHF**()
: test_UHF will test the regular UHF method, by checking whether or not it returns the expected result. The accuracy is 10^-6.

ghf.tests.test_auth.**test_extra_e**()
: test_extra_e will test the UHF method, with the added option of first adding 2 electrons to the system and using those coefficients for the actual system, by checking whether or not it returns the expected result. The accuracy is 10^-6.

ghf.tests.test_auth.**test_stability**()
: test_stability will test the UHF method, with stability analysis, by checking whether or not it returns the expected result. The accuracy is 10^-6.

PYTHON MODULE INDEX

g

INDEX

N

R

S

T

U