First project fall 2016 FYS-KJM4480/9480

Date given: Tuesday September 12, 2015.
Deadline: Thursday October 5 at 12pm. It counts 30% of the final mark.

All deliverables must be electronic, and code must be properly commented. Uncommented codes will automatically fail.

It is suggested that you use Python/Jupyter Notebooks for this project, but you are free to choose.

Introduction

In this project we will apply restricted Hartree–Fock (RHF) and unrestricted Hartree–Fock (UHF) for simple molecules. You will implement basic solvers for these methods, in principle applicable to any electronic system. We will formulate the methods in terms of matrix elements of the Hamiltonian that we assume are available from external libraries – in this case we will use the quantum chemical code Psi4 to generate these. As almost universally the case with such codes, the single-particle basis functions employed are linear combinations of Gaussian functions.

In RHF, it is assumed that all HF single-particle functions are of the form of spin-orbitals,

$$\phi_{p\sigma}(x) = \varphi_p(\vec{r})\chi_\sigma(s),$$

i.e., that they separate into space and spin parts, and that each *orbital* φ_i is doubly occupied,

$$|\Phi_{\text{RHF}}\rangle = \left|\phi_{1\uparrow}\phi_{1\downarrow}\cdots\phi_{N\uparrow}\phi_{N\downarrow}\right\rangle = |\varphi_1\cdots\varphi_N\bar{\varphi}_1\cdots\bar{\varphi}_N\rangle \tag{1}$$

where the total number of electrons is $2N$, i.e., even. Here, the $\bar{\varphi}$ indicates a spin-$\downarrow$ spin-orbital.

In contrast, in UHF, we have $N_\uparrow$ and $N_\downarrow$ electrons of spin $\uparrow$ and $\downarrow$, respectively. The $\uparrow$-electrons have orbitals $\varphi_i^\uparrow(\vec{r})$, while the $\downarrow$-electrons have orbitals $\varphi_i^\downarrow(\vec{r})$, giving

$$\phi_{p\sigma}(x) = \varphi_p^\sigma(\vec{r})\chi_\sigma(s), \tag{2}$$

and a total Slater determinant

$$|\Phi_{\text{UHF}}\rangle = \left|\varphi_1^\uparrow\cdots\varphi_{N_\uparrow}^\uparrow\bar{\varphi}_1^\downarrow\cdots\bar{\varphi}_{N_\downarrow}^\downarrow\right\rangle. \tag{3}$$

Thus, the $\uparrow$ and $\downarrow$-electrons have different spatial parts of the orbitals.

The UHF ansatz is more flexible than the RHF ansatz, and will produce lower ground-state energy estimates at the cost of more variables to store and compute. On the other hand, the RHF wavefunction is an eigenfunction of the total electron spin (with eigenvalue $S = 0$), while this is not true for the UHF wavefunction, unless it happens to be that $\varphi_i^\uparrow = \varphi_i^\downarrow$ at the solution. Considering that the exact ground-state wavefunction of a molecule is a spin-eigenfunction, we have the so-called *symmetry dilemma*: should we prefer lower energy and breaking a symmetry in the wavefunction over a higher energy with more symmetry?

The Hamiltonian of the system is assumed to be on the form

$$\hat{H} = \hat{H}_0 + \hat{W} = \sum_{i=1}^{N_{\text{el}}}\hat{h}(\vec{r}_i) + \sum_{i<j=1}^{N_{\text{el}}}\hat{w}(\vec{r}_i - \vec{r}_j). \tag{4}$$

Notably, the Hamiltonian is not dependent on spin degrees of freedom. In the case of a molecule, we have an additional energy term from the Coulomb repulsion between the nuclei.

As is common in quantum chemistry, we choose choose atomic units[1] for our calculation. This is also convenient in Psi4. The unit for length is the Bohr radius (approximately equal to 0.53 Å), and the energy unit is the Hartree (approximately equal to 27 eV). The one-body part of the Hamiltonian in atomic units reads

$$\hat{h}(\vec{r}) = -\frac{1}{2}\nabla^2 + \sum_\alpha \frac{-Z_\alpha}{|\vec{r} - \vec{R}_\alpha|} \tag{5}$$

where $\vec{R}_\alpha$ is the location of nucleus α, with charge Z_α. The the two-body operator is

$$\hat{w}(\vec{r}_1 - \vec{r}_2) = \frac{1}{|\vec{r}_1 - \vec{r}_2|}. \tag{6}$$

Additionally, the total energy has a classical contribution from the repulsion among the nuclei,

$$E_{\text{nuc}} = \sum_{\alpha\beta} \frac{Z_\alpha Z_\beta}{|\vec{R}_\alpha - \vec{R}_\beta|}. \tag{7}$$

In this project, a matrix element involving space-functions only are denoted with round brackets, i.e.,

$$(\varphi|\hat{h}|\psi) = \int \varphi(\vec{r})^*[\hat{h}\psi(\vec{r})]\; d^3r, \tag{8}$$

and

$$(\varphi_1\varphi_2|\hat{w}|\psi_1\psi_2) = \iint \varphi_1(\vec{r}_1)^*\varphi_2(\vec{r}_2)^* w(r_{12})\psi_1(\vec{r}_1)\psi_2(\vec{r}_2)\; d^3r_1 d^3r_2, \tag{9}$$

where we also indicate that $\hat{w}$ is actually a local potential, i.e., the Coulomb potential. This small fact is not needed for the derivations or implementations, however. The distinction between integrals over both spin and space is important. For example, for two spin-orbitals $\phi_{p\sigma} = \varphi_p\chi_\sigma$ and $\phi_{q\tau} = \varphi_q\chi_\tau$, we have

$$\langle\phi_{p\sigma}|\hat{h}|\phi_{q\tau}\rangle = \delta_{\sigma\tau}(\varphi_p|\hat{h}|\varphi_q), \tag{10}$$

and similarly for the two-electron integrals.

Theory

In these exercises, we derive the UHF equations, and then the matrix representation of these equations, the Roothan–Hall equation. (Some reserve the term "Roothan–Hall equation" for RHF, and prefer the term Pople–Nesbet–Berthier equations for UHF.) We proceed in several steps.

1. Energy expression

a) For a Slater determinant $|\Phi\rangle = |\phi_1\phi_2\cdots\phi_N\rangle$, show that

$$\langle\Phi|\hat{H}|\Phi\rangle = \sum_I \langle\phi_I|\hat{h}|\phi_I\rangle + \frac{1}{2}\sum_{IJ}\langle\phi_I\phi_J|\hat{W}|\phi_I\phi_J - \phi_J\phi_I\rangle, \tag{11}$$

where the capital indices imply summation over all occupied single-particle functions. Use the second-quantized form of the Hamiltonian.

[1] `https://en.wikipedia.org/wiki/Atomic_units`

b) The UHF energy expression is

$$E_{\text{UHF}} = \langle \Phi_{\text{UHF}} | \hat{H} | \Phi_{\text{UHF}} \rangle. \tag{12}$$

Using the result from 1a), show that

$$E_{\text{UHF}} = \sum_{i\sigma} (\varphi_i^\sigma | \hat{h} | \varphi_i^\sigma) + E_{\text{H}} - E_{\text{ex}}, \tag{13}$$

with the *Hartree energy* (or direct interaction energy)

$$E_{\text{H}} = \frac{1}{2} \sum_{ij\sigma\tau} (\varphi_i^\sigma \varphi_j^\tau | \hat{w} | \varphi_i^\sigma \varphi_j^\tau) \tag{14}$$

and the *exchange energy*

$$E_{\text{ex}} = \frac{1}{2} \sum_{ij\sigma} (\varphi_i^\sigma \varphi_j^\sigma | \hat{w} | \varphi_j^\sigma \varphi_i^\sigma). \tag{15}$$

The Hartree energy is a classical term describing the energy of an electron in the mean field of all the other electrons. The excange term is non-classical.

2. UHF equations

In this exercise we derive the UHF equations: E_{UHF} is a stationary point with respect to all infinitesimal variations in the φ_i^σ, subject to the orthonormality constraint, if and only if there are constants λ_{ji}^σ such that, for all σ and all $i = 1, 2, \cdots, N_\sigma$,

$$\left[\hat{h} + \sum_\tau \hat{v}_{\text{H}}^\tau - \hat{v}_{\text{ex}}^\sigma \right] \varphi_i^\sigma = \sum_j \lambda_{ji}^\sigma \varphi_j^\sigma. \tag{16}$$

Here,

$$\hat{v}_{\text{H}}^\tau \psi = \sum_j (\cdot \, \varphi_j^\tau | \hat{w} | \psi \varphi_j^\tau), \tag{17}$$

and

$$\hat{v}_{\text{ex}}^\sigma \psi = \sum_j (\cdot \, \varphi_j^\sigma | \hat{w} | \varphi_j^\sigma \psi). \tag{18}$$

Thus, the UHF equation (16) is a coupled set of nonlinear integro-partial differential equations. We note that the equation for $\sigma =\uparrow$ is coupled to that of $\sigma =\downarrow$ via the Hartree potential $\hat{v}_{\text{H}}^\tau$. We define the Fock operator for each spin direction as

$$\hat{f}^\sigma = \hat{h} + \sum_\tau \hat{v}_{\text{H}}^\tau - \hat{v}_{\text{ex}}^\sigma. \tag{19}$$

The N_σ occupied orbitals φ_i^σ can be extended to a basis for the square-integrable space functions, i.e.,

$$\{\varphi_p^\sigma\} = \{\varphi_i^\sigma\} \cup \{\varphi_a^\sigma\}$$

is a basis for the square-integrable functions over space, for each $\sigma =\uparrow$ or $\downarrow$. Thus, any square-integrable function ψ can be written

$$\psi(\vec{r}) = \sum_p (\varphi_p^\sigma | \psi) \varphi_p^\sigma(\vec{r}),$$

where notably the summation does not include σ.

a) We prove Eq. (16) by performing all independent variations in the orbitals. According to the previous paragraph, these variations can be expanded in our basis,

$$\delta\varphi_i^\sigma(\vec{r}) = \sum_p \varphi_p^\sigma(\vec{r})(\varphi_p^\sigma|\delta\varphi_i^\sigma) \equiv \sum_p \varphi_p^\sigma(\vec{r})\eta_{pi}^\sigma \tag{20}$$

Explain why we can assume that $(\varphi_j^\sigma|\delta\varphi_i^\sigma) = 0$. Explain why it is sufficient to vary only the bra wavefunction $\langle\Phi_{\text{UHF}}|$ in the UHF energy expression.

b) The numbers η_{bj}^σ are all independent variation parameters. Thus, set all to zero, except for a single index (i, a, σ). Show that we must in this case have

$$\delta E_{\text{UHF}} = \langle\delta\Phi_{\text{UHF}}|\hat{H}|\Phi_{\text{UHF}}\rangle = (\varphi_a^\sigma|\hat{h}|\varphi_i^\sigma) + \sum_{j\tau}(\varphi_a^\sigma\varphi_j^\tau|\hat{w}|\varphi_i^\sigma\varphi_j^\tau) - \sum_j(\varphi_a^\sigma\varphi_j^\sigma|\hat{w}|\varphi_j^\sigma\varphi_i^\sigma). \tag{21}$$

Show that equating (21) to zero becomes

$$(\varphi_a^\sigma|\hat{f}^\sigma|\varphi_i^\sigma) = 0. \tag{22}$$

c) Now comes the very final step in the proof of Eq. (16). Given (i, σ), Eq. (22) must hold for all a. Explain why it then must hold that there exists numbers λ_{ij}^σ such that

$$\hat{f}^\sigma\varphi_i^\sigma(\vec{r}) = \sum_j \lambda_{ji}^\sigma\varphi_j^\sigma(\vec{r}). \tag{23}$$

3. Roothan–Hall equations

The UHF equatuions (24) are of *non-canonical* form. Similar as to general HF theory, $|\Phi_{\text{UHF}}\rangle$ is invariant under unitary mixings of the $\uparrow$ and $\downarrow$ orbitals, separately. Thus, there is a particular choice of unitary mixing that diagonalizes λ^σ, i.e., we can replace the noncanonical equations with the canonical equations, which are of eigenvalue form,

$$\hat{f}^\sigma\varphi_i^\sigma = \epsilon_i^\sigma\varphi_i^\sigma. \tag{24}$$

Note carefully, that we have *two* coupled eigenvalue problems, one for each spin direction σ.

For a numerical treatment, we introduce *atomic orbital basis functions* ψ_p, $p = 1, 2, \cdots, K$ in which we expand our orbitals,

$$\varphi_q^\sigma = \sum_p \psi_p U_{pq}^\sigma. \tag{25}$$

These are often called *molecular orbitals*.

Note carefully that the basis is the same for each spin direction. The atomic orbitals are not necessarily orthonormal; instead we assume a general non-diagonal overlap matrix

$$S_{pq} = (\psi_p|\psi_q). \tag{26}$$

When the basis functions are linearly independent, S is invertible. Indeed, S is symmetric and positive definite (SPD).

We define the *atomic integrals*

$$h_{qp} = (\psi_q|\hat{h}|\psi_p), \quad (pr|qs) = (\psi_p\psi_q|\hat{w}|\psi_r\psi_s), \tag{27}$$

and note carefully the index placements in the two-electron integrals. This is the *Mulliken notation* for two-electron integrals, and the index convention is used by Psi4 and many other quantum chemical codes.

Moreover, we define the *density matrices*

$$D^\sigma_{rs} = \sum_j U^\sigma_{rj}(U^\sigma_{sj})^* = U^\sigma_{\mathrm{occ}}(U^\sigma_{\mathrm{occ}})^H, \tag{28}$$

where U^σ_{occ} is the $K \times N_\sigma$ submatrix of U^σ formed by the first N_σ columns.

a) The Roothan–Hall equations are obtained by projecting the UHF equations onto the atomic orbital basis. Show the following relations:

$$(\psi_p|\hat{h}|\varphi^\sigma_i) = \sum_q h_{pq}U^\sigma_{qi} \tag{29a}$$

$$(\psi_p|\hat{v}^\tau_{\mathrm{H}}|\varphi^\sigma_i) = \sum_q \left(\sum_{rs}(pq|rs)D^\tau_{sr}\right) U^\sigma_{qi} \tag{29b}$$

$$(\psi_p|\hat{v}^\sigma_{\mathrm{ex}}|\varphi^\sigma_i) = \sum_q \left(\sum_{rs}(ps|rq)D^\sigma_{sr}\right) U^\sigma_{qi} \tag{29c}$$

$$(\psi_p|\varphi^\sigma_i) = \sum_q S_{pq}U^\sigma_{qi}. \tag{29d}$$

We define the Fock matrix

$$F^\sigma = h + J(D^\uparrow + D^\downarrow) - K(D^\sigma), \tag{30}$$

where

$$J(D)_{pq} = \sum_{rs}(pq|rs)D_{sr}, \quad K(D)_{pq} = \sum_{rs}(ps|rq)D_{sr}. \tag{31}$$

b) Explain why the projected UHF equation (the Roothan–Hall equation) can be written

$$F^\sigma(D^\uparrow, D^\downarrow)U^\sigma = SU^\sigma\epsilon^\sigma, \quad \epsilon^\sigma = \mathrm{diag}(\epsilon^\sigma_1, \epsilon^\sigma_2, \cdots, \epsilon^\sigma_K). \tag{32}$$

For a given spin direction, discuss the dependence in this equation of of the various molecular orbitals.

4. Restricted Hartree–Fock

This is a short exercise, as we will not derive the RHF equations in this project. Basically, the UHF formulas ar still valid if we introduce the restructions $U = U^\uparrow = U^\downarrow$ and $N = N_\uparrow = N_\downarrow$. We obtain the Roothan–Hall equation for RHF,

$$F(D)U = SU\epsilon, \quad \epsilon = \mathrm{diag}(\epsilon_1, \epsilon_2, \cdots, \epsilon_K). \tag{33}$$

The Fock operator reads

$$F = h + J(D) - \frac{1}{2}K(D), \quad D = 2U_{\mathrm{occ}}U^H_{\mathrm{occ}}. \tag{34}$$

Here, the molecular orbitals are expanded as

$$\varphi_p(\vec{r}) = \sum_q \psi_q(\vec{r})U_{qp}. \tag{35}$$

a) Show that if a solution of the RHF Roothan–Hall equations are found, then we also have a solution of the UHF Roothan–Hall equations. (As solution of the latter, it may be numerically unstable. There may be solutions of the UHF Roothan–Hall equations nearby that break the symmetry $U^\uparrow = U^\downarrow$ and that are stable.)

Implementation exercises

In this exercise, you are going to write a simple implementation of the Roothan–Hall equations for RHF and UHF. You may use any programming environment, but it is suggested that you use Anaconda Python 3, avaiable for all major operating systems, as it includes NumPy, SciPy, Psi4, etc.[2] Anaconda comes with a convenient package manager. You can install Psi4 from the command line using

```
conda install -c psi4 psi4
```

Notably, Anaconda does not meddle with any other Python installation, as it is stored in a local folder in the user's home directory.

The example snippet below uses Jupyter Notebooks. It is a convenient way to do the coding for this project.

You may also try Jupyterhub at UiO, `http://www.uio.no/tjenester/it/forskning/beregning/jupyter/`. In that way you can avoid any installation of Python at all. It has Psi4 installed.

A good source of further information on Psi4 and Python is Psi4numpy[3]. It contains notebooks with valuable information and tutorials. The information contained in the present document should be sufficient, however.

Python code for getting started

In the appendix a Jupyter Notebook is included, so you can see how to use Psi4 in python. The file `Project 1 - Setup.ipynb` is also provided from the Google Drive folder of the course. In the notebook, we define a molecule and extract atomic integrals from Psi4. Of course, Psi4 has both UHF and RHF implemented, and you can use this to check that your implementation gives the right result. Study the code snippet, and note in particular that to compare our HF energy with the Psi4 HF energy, we must add the nuclear interaction contribution. The notebook also contains a routine to solve the generalized eigenvalue problem with numpy.

Psi4 comes with many different basis sets. It will take us too far to go into the details in this project, so wi stick with the basis set called cc-pVDZ.[4] The basis sets are constructed atom-by-atom, adding basis functions per electron. As a molecule is drawn apart (dissociated) the basis functions for each atom decouple.

SCF iterations for UHF and RHF

Self-consistent field (SCF) iterations for UHF can now be set up by choosing initial guesses for $U^\uparrow$, $U^\downarrow$, and ϵ, and iterating the Roothan–Hall equation (33) until convergence,

$$F^\sigma(D^{\uparrow,(k)}, D^{\downarrow,(k)})U^{\sigma,(k+1)} = SU^{\sigma,(k+1)}\epsilon^{\sigma,(k+1)}. \tag{36}$$

Thus, each iteration requires the diagonalization of two matrices, $F^\uparrow$ and $F^\downarrow$. The eigenvalues $\epsilon_p^{\sigma,(k)}$ should be sorted in ascending order. A convenient convergence criterion could be that $\Delta^{(k)} = \max_{p,\sigma} |\epsilon_p^{\sigma,(k+1)} - \epsilon_p^{\sigma,(k)}|$ is smaller than some fixed threshold.

The initial guess in UHF must be chosen with some care. Compute the density matrix from H as your initial guess for a density matrix $U^{\sigma,(0)}$ for both spin directions, but add *a small component of random Hermitian noise*, pseudocode,

[2]`https://anaconda.org/`

[3]`https://github.com/psi4/psi4numpy`

[4]See for example the Psi4 manual `http://www.psicode.org/psi4manual/master/basissets.html`, or `http://vergil.chemistry.gatech.edu/courses/chem6485/pdf/basis-sets.pdf`.

```
X = np.random.rand(nbf,nbf)
X = X + np.transpose(X)
Dsigma = Dsigma + 0.001*X
```

You are then guaranteed that you will not be trapped in a RHF-type state: If the initial guess for $U^\uparrow$ and $U^\downarrow$ are identical, each iteration will also have identical up and down parts, and you will end up with solving the RHF equations instead of the RHF equations. When solving the RHF equations you cannot rely on this – you should use the formulas from Exercise 4 and write a separate SCF iteration loop. Otherwise the loop is similar.

Simple SCF iterations often have convergence problems, and may end up oscillating, for example. One simple technique to try if this happens is to mix the current density matrix with a small part of the previous density matrix, i.e., replace $D^{\sigma,(k)}$ in the left-hand-side of Eq. (36) by the convex combination

$$\tilde{D}^{\sigma,(k)} = (1-\theta)D^{\sigma,(k)} + \theta D^{\sigma,(k-1)}, \tag{37}$$

where $\theta \in [0, 1]$.

The SCF iterations for RHF are of course similar, but simpler, as you set $D^{\uparrow,(k)} = D^{\downarrow,(k)}$.

5 Write SCF solvers

Write a function that solves the UHF SCF iteration problem for a given set of matrix input data. Write a similar function for the RHF SCF iteration. Implement the convex combination hack from above, and let θ be a parameter to your function. Test the routines on the H_2O molecule for a bond angle of 104° and distances $R = 1.84$, which is close to the equilibrium. The UHF and RHF numbers should be the same. See the appendix for the Psi4 definition of the H_2O molecule. Compare with the Psi4 result. Also test on the H_2 molecule with bond distance $R = 5$, for which UHF and RHF should give different answers.

Exploration

6 Dissociation of H_2

Compute UHF and RHF energies for the H_2 molecule for interatomic distance ranging in the interval $R \in [0.7, 7.0]$. Discuss the result. You should observe a critical R_c where the UHF and RHF solutions are not the same. Note that the exact ground-state energy of the H atom is -0.5 Hartrees.

Discuss the limit of the RHF and UHF wavefunctions as R increases. Hint: consider the limits of the matrix elements of $\hat{h}$ and $\hat{w}$ in these limits. Half of the basis functions ψ_p are centered at each atom in the limit.

Project 1 H2017 - Setup

September 12, 2017

1 Getting integrals from Psi4

This is a notebook showing how to call Psi4 and obtain integrals, etc. We also show how to call Psi4 to get benchmark energies for RHF and UHF calculations.

Here, we choose a fixed basis with acronym cc-pVDZ; a common basis set. You may experiment with other basis sets in your calculations, such as cc-pVTQ, cc-pVQZ, sto-3G.

```
In [1]: import psi4
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [2]: def call_psi4(mol_spec, extra_opts = {}):
            mol = psi4.geometry(mol_spec)

            # Set options to Psi4.
            opts = {'basis': 'cc-pVDZ',
                             'reference' : 'uhf',
                             'scf_type' :'direct',
                             'guess' : 'core',
                             'guess_mix' : 'true',
                             'e_convergence': 1e-7}

            opts.update(extra_opts)

            # If you uncomment this line, Psi4 will *not* write to an output file.
            #psi4.core.set_output_file('output.dat', False)

            psi4.set_options(opts)

            # Set up Psi4's wavefunction. Psi4 parses the molecule specification,
            # creates the basis based on the option 'basis' and the geometry,
            # and computes all necessary integrals.
            wfn = psi4.core.Wavefunction.build(mol, psi4.core.get_global_option('BASIS'))

            # Get access to integral objects
            mints = psi4.core.MintsHelper(wfn.basisset())

            # Get the integrals, and store as numpy arrays
```

```
    T = np.asarray(mints.ao_kinetic())
    V = np.asarray(mints.ao_potential())
    H = T + V
    W = np.asarray(mints.ao_eri())
    S = np.asarray(mints.ao_overlap())

    # Get number of basis functions and number of occupied orbitals of each type
    nbf = wfn.nso()
    nalpha = wfn.nalpha()
    nbeta = wfn.nbeta()


    # Perform a SCF calculation based on the options.
    SCF_E_psi4 = psi4.energy('SCF')
    Enuc = mol.nuclear_repulsion_energy()

    # We're done, return.
    return SCF_E_psi4, Enuc, H,W,S, nbf, nalpha, nbeta
```

2 Do a test calculation

We define two molecule geometries and perform UHF and RHF calculations using Psi4.

```
In [3]: r = 1.84

        h2o ="""
            O
            H 1 r
            H 1 r 2 104
            symmetry c1
            r = %f
            units bohr
        """ % (r)

        E_UHF_psi4, Enuc, H, W, S, norb, nocc_up, nocc_dn = call_psi4(h2o, {'reference' : 'uhf

        r = 2.0


        h2 = """
            0 1
            H
            H 1 %f
            symmetry c1
            units bohr
            """ % (r)

        E_RHF_psi4, Enuc, H, W, S, norb, nocc_up, nocc_dn = call_psi4(h2, {'reference' : 'rhf'}
        print("RHF energy of H2 molecule at interatomic distance R = %f is E_RHF = %f" % (r, E
        print("The nuclear term is Enuc = ", Enuc)
        print("Basis size = %d, N_up = %d, N_down = %d" % (norb, nocc_dn, nocc_up))

        E_UHF_psi4, Enuc, H, W, S, norb, nocc_up, nocc_dn = call_psi4(h2o, {'reference' : 'uhf

        print("RHF energy of H2O molecule at interatomic distance R = %f is E_RHF = %f" % (r, 
        print("The nuclear term is Enuc = ", Enuc)
        print("Basis size = %d, N_up = %d, N_down = %d" % (norb, nocc_dn, nocc_up))


        #print(np.allclose(np.dot(S12,np.dot(S,S12)),np.eye(norb)))
```

```
RHF energy of H2 molecule at interatomic distance R = 2.000000 is E_RHF = -1.089283
The nuclear term is Enuc =  0.5
Basis size = 10, N_up = 1, N_down = 1
RHF energy of H2O molecule at interatomic distance R = 2.000000 is E_RHF = -76.025518
The nuclear term is Enuc =  9.040494080182766
Basis size = 24, N_up = 5, N_down = 5
```

3 Solving the generalized eigenvalue problem with numpy

For unknown reasons, the numpy team has removed the direct support for solution of the generalized eigenvalue prolem: for Hermitian matrices A and B, with B positive definite, solve

$$AU = BU\lambda, \quad \lambda \text{ diagonal.}$$

This generalized eigenvalue problem is at the core of the SCF iterations, so we need to be able to solve them. Here is a small routine that transforms the generalized eigenvalue problem to standard form by computing the Cholesky decomposition $B = LL^T$, with L lower triangular. A solution to the generalized EVP can then be found by solving the symmetric EVP

$$(L^{-1}AL^{-T})V = V\lambda,$$

and compute $U = L^{-T}V$.

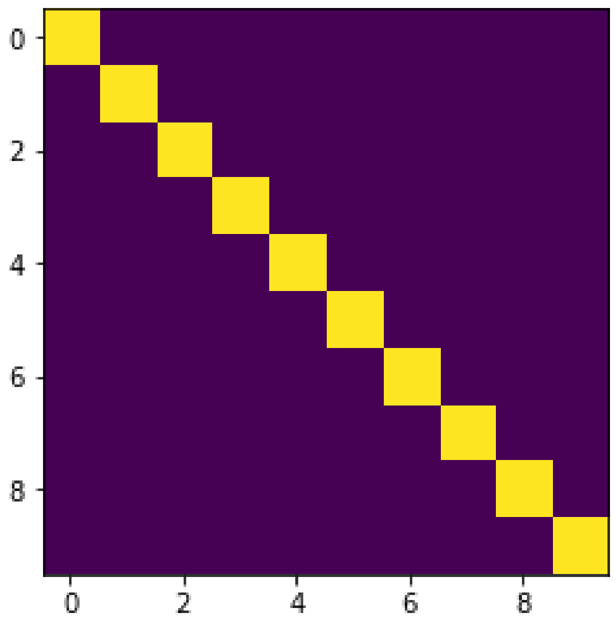
The routine returns the eigenvalues in ascending order (a feature of LAPACK).

```
In [4]: def geig(A,B):
            """
            Solve the generalized eigenvalue problem A*U = B*U*lambda,
            where B is symmetric, positive definite, and where A is symmetric.

            Turns out numpy has removed the generalize eigenvalue call to LAPACK ...
            Why? Beats me.

            Returns lamb, U.
            lamb = vector of eigenvalues in ascending order
            U = matrix of eigenvectors.
            """
            # Compute Cholesky decomp B = L * L^T
            L = np.linalg.cholesky(B)

            # Compute new matrix C = L^{-1} A L^{-T}
            Linv = np.linalg.inv(L)
            LinvT = np.transpose(Linv)
            C = np.linalg.multi_dot([Linv, A, LinvT])

            lamb, V = np.linalg.eigh(C)
            U = np.dot(LinvT, V)
            return lamb, U
```

```
In [5]: E_RHF_psi4, Enuc, H, W, S, norb, nocc_up, nocc_dn = call_psi4(h2, {'reference' : 'rhf'
        lamb, U = geig(H,S)
```

```
In [6]: print("Eigenvalues of H:", lamb)
        print("Eigenvectors:", U)
        plt.imshow(np.linalg.multi_dot([np.transpose(U),S,U]))
        plt.show()
```

```
Eigenvalues of H: [-1.10026467 -0.66459207 -0.22970275 -0.06169326  0.24348876  0.24348876
  0.54773078  0.79117215  0.79117215  1.56633065]
Eigenvectors: [[ -4.80361032e-01   5.66026402e-01   6.93239252e-01   8.70606993e-01
    4.94103810e-17  -3.87012485e-17  -4.52709658e-01  -2.54221242e-16
   -4.25523637e-17  -6.51665963e-01]
 [ -1.25456465e-01   6.80678739e-01  -7.51915609e-01  -1.76769067e+00
   -3.29192486e-17  -1.38729074e-16   2.43196575e-01   1.64965128e-16
    1.29289156e-16  -4.49691060e-01]
 [ -5.03728667e-02  -2.25580810e-02   1.09426703e-01  -6.55225220e-02
   -1.02819037e-16   1.72570205e-16   6.08045718e-01   6.78057773e-17
   -4.37180877e-17  -1.29762105e+00]
 [ -3.43313454e-19   5.38867286e-17   1.63977196e-16  -1.07297867e-16
   -1.88581426e-01   6.08065525e-01  -4.73449048e-17  -7.88727153e-01
   -1.74181387e-01   2.32232635e-16]
 [ -7.26403456e-20   2.70370170e-17  -1.54106682e-16   5.23509410e-17
    6.08065525e-01   1.88581426e-01  -1.99324344e-17   1.74181387e-01
   -7.88727153e-01  -9.84226545e-19]
 [ -4.80361032e-01  -5.66026402e-01   6.93239252e-01  -8.70606993e-01
    3.13387209e-17  -1.26374418e-16  -4.52709658e-01  -2.64452198e-16
    4.59157241e-17   6.51665963e-01]
 [ -1.25456465e-01  -6.80678739e-01  -7.51915609e-01   1.76769067e+00
   -9.17035538e-19   2.26706972e-16   2.43196575e-01   2.28950985e-16
   -1.23400868e-16   4.49691060e-01]
 [  5.03728667e-02  -2.25580810e-02  -1.09426703e-01  -6.55225220e-02
    5.24473595e-17  -1.24841149e-16  -6.08045718e-01  -2.56403479e-16
    1.39247526e-17  -1.29762105e+00]
 [  1.46945063e-18   6.95285550e-18   1.09412974e-17  -1.59413339e-17
   -1.88581426e-01   6.08065525e-01  -3.91351515e-16   7.88727153e-01
    1.74181387e-01  -4.36068184e-17]
 [  3.10915289e-19   3.07557539e-18  -5.31890840e-17   1.35263356e-17
    6.08065525e-01   1.88581426e-01   8.53148282e-17  -1.74181387e-01
    7.88727153e-01   4.21268757e-18]]
```



```
In [ ]:
```