Assignment #2 Hartree-Fock Self-Consistent Field Program

This assignment is meant to give you some experience using NumPy and SciPy in a scientific code.

Introduction

This assignment has been adapted from the "Programming Tutorial in Chemistry by Python" originally developed by Daniel Crawford

The Hartree-Fock (HF) method is an approximation that allows one to solve the molecular Schrodinger equation by assuming that the potential felt by an electron is determined as a mean-field of all of the other electrons in the system. Essentially this allows us to represent the wavefunction of a molecule as a Slater determinant. We can then use the variational method to show that solving equations is a minimization process that can be implemented iteratively, which is termed the Self-Consistent Procedure (SCF)

To learn more about HF theory, it is recommended that you start with the notes from C. David Sherrill at the Georgia Institute of Technology or Chapter 3 of Szabo and Ostlund's Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory

This project uses the PySCF Module, see References at the bottom for citation.

The Assignment

Assumptions: You have Python installed on your system

To complete the assignment:

1. Fork this repository into your own GitHub account.
2. Clone your forked repository
3. Go into your new directory and create a virtualenv with he command

```
python -m venv ./venv
```

4. Activate your new virtual environment

```
source ./venv/bin/activate`
```

5. Install the necessary modules

```
pip install -r requirements.txt
```

6. Now you are ready to start programming. There are two python files.

- main.py - this is the main controller program, it has the basic SCF procedure outlined with all of the function calls that will be needed. You should not need to edit this file, but you should review it as it will be needed to run the final program.
- SCF.py - this is a file with stubs for all of the functions that you will need to implement. The goal will be to fill in each function. Notes are given below about each step of the procedure.

7. Submit your assignment by creating a pull request for your fork to the original repository and assign review to Shawn Brown.

Note: This process will allow you to see others assignments, while we encourage you to share thoughts and work together, but the purpose of this is not to get a grade, but to learn programming, so we are trusting you to do the assignment without cheating.

The SCF Procedure

The outline of the main SCF procedure is:

Main SCF Procedure

Each step is outlined in the Github Repo Readme Your job will be to fill in the stubbed functions in SCF.py

- Step 1. Calculate the Electron Nuclear Repulsion Energy
- Step 2. Calculate the Initial Hcore Matrix
- Step 3. Calculate the Initial Density Matrix
- Step 4. Start the SCF Procedure
  - Step 4a. Calculate the Fock Matrix
  - Step 4b. Solve Eigenvalues and Eigenvectors of Roothan Equations
  - Step 4c. Calculate the Total Energy of the Current Iteration
  - Step 4d. Calculate the new Density Matrix
  - Step 4e. Calculate the Energy Difference and RMS Difference of Density
  - Step 4f. Check for Convergence, if Converged, Exit
  - Step 4g. If not Converged, update Density Matrix and Energy and do another iteration
- Step 5. Print out Final Total Energy for User

The Input Parameters and the integrals

In main.py, you will see a number of variables that are defined including:

Molecular properties

- the `mol_h2o` variable which defines all of the properties of a water molecule. This is created from PySCF and will contain all of the necessary properties needed to execute the functions.

One-electron Integrals

All three of the matrices below are two-dimensional matrices with the dimensions of number of atomic orbitals x number of atomic orbitals.

- S_{uv}: The Overlap matrix

$$S_{\mu\nu} = \int \phi_\mu(\boldsymbol{r})\phi_\nu(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r}$$

- T_{uv}: The Kinetic Integral Matrix

$$T_{\mu\nu} = \int \phi_\mu(\boldsymbol{r})\left(-\frac{1}{2}\nabla_{\boldsymbol{r}}^2\right)\phi_\nu(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r}$$

- V_{uv}: The Nuclear-Attraction Integral Matrix

$$V_{\mu\nu} = \int \phi_\mu(\boldsymbol{r})\left(-\sum_A \frac{Z_A}{|\boldsymbol{r}-\boldsymbol{R}_A|}\right)\phi_\nu(\boldsymbol{r})\,\mathrm{d}\boldsymbol{r}$$

Two-electron Integrals

This is a 4-D matrix that has four equivalent dimensions of number of atomic orbitals.

- eri: The Electron Repulsion Integrals

$$(\mu\nu|\kappa\lambda) = \int \phi_\mu(\boldsymbol{r}_1)\phi_\nu(\boldsymbol{r}_1)\frac{1}{|\boldsymbol{r}_1-\boldsymbol{r}_2|}\phi_\kappa(\boldsymbol{r}_2)\phi_\lambda(\boldsymbol{r}_2)\,\mathrm{d}\boldsymbol{r}_1\,\mathrm{d}\boldsymbol{r}_2$$

Convergence Criteria

As we perform the iterative SCF procedure, we need criteria to indicate that we have converged the calculation. Both of the below criteria should be met before declaring convergence. As the algorithm moves forward, these should trend lower until they are both below the thresholds.

- E_conv_threshold: The threshold for the difference in the total energy between SCF iterations is below to indicate convergence.

$$\Delta_E = |E_{\text{tot}}^t - E_{\text{tot}}^{t-1}| < \delta_1$$

- D_conv_threshold: The threshold for the RMS of the Density Matrix between SCF iterations is below to indicate convergence.

$$\Delta_E = |E_{\text{tot}}^t - E_{\text{tot}}^{t-1}| < \delta_1$$

Step 1. Calculate the Electron Nuclear Repulsion Energy

Function Stub: `calc_nuclear_repulsion_energy`

The Nuclear Repulsion energy is function of the charge and positions of the atoms in the molecule and defined by the formula:

$$E_{\text{nuc}} = \sum_{B>A} \sum_{A} \frac{Z_A Z_B}{R_{AB}}$$

where A and B are atoms in the molecule. First compute the distance between two points with the NumPy function `np.linalg.norm` (api doc here), and then perform the summation loop to compute the E_{nuc}.

Hint: Your function should return `8.00236706181077`

Step 2. Calculate the Initial H_{core} Matrix

Function Stub: `calc_hcore_matrix`

H_{core} is termed the *Core Hamiltonian* matrix and it is the simple sum of the 1-electron integrals:

$$H^{\text{core}}_{\mu\nu} = T_{\mu\nu} + V_{\mu\nu}$$

Hint:

```
Huv[0,0] = -32.57739541261037
Huv[3,4] = Huv[4,3] = 0.0
```

Step 3. Calculate the Initial Density Matrix

Function Stub: `calc_initial_density`

There are many ways to make an initial guess at the density matrix for the HF-SCF procedure. In our case, we will start with a density matrix that is just filled with zeros which will essentially use the Core Hamiltonian as the initial guess.

This function is provided for later enhancements where you may want to try other density matrix guesses, such as random. For now, this should return a matrix with the appropriate dimentions (number of atomic orbitals X number of atomic orbitals) filled with double precision 0s.

Step 4. Start the SCF Procedure

Here we begin the iterative process of solving the HF equations, calculating the total energy of the molecule, and forming an updated density matrix.

The HF Equations take the form of the eigenvector matrix Roothaan Equations:

$$\mathbf{FC} = \mathbf{SC}\varepsilon$$

Where **F** is known as the Fock Matrix.

Step 4a. Calculate the Fock Matrix

Function Stub: `calc_fock_matrix`

The Fock Matrix is formed through the following equation:

$$F_{\mu\nu} = H_{\mu\nu}^{\text{core}} + \sum_{\kappa\lambda} D_{\kappa\lambda}\left[(\mu\nu|\kappa\lambda) - \frac{1}{2}(\mu\kappa|\nu\lambda)\right]$$

F_{uv} is the Fock Matrix

H^{core}_{uv} is the Core Hamiltonian Matrix from Step 2.

D_{uv} is the Density Matrix from Step 3

The terms that include $(\mu\nu|\kappa\lambda)$, these are the electron repulsion integrals, `eri`. This integral would be equivalent to eri[μ,ν,κ,λ].

Now let's look at the two terms within the square brackets.

The first term: $\sum_{\kappa\lambda} D_{\kappa\lambda}(\mu\nu|\kappa\lambda)$ is the Coulomb term and represents the classical analogue to the Coulomb force between two charged particles.

This term could be implemented with a four-fold loop, but you should look at how you can use the `sum()` aggregator to do it with just a loop over μ and ν For example, if I wanted to calculate this term's contribution to the Fock Matrix element [0,0], you could use:

```
(Duv*eri[0,0]).sum()
```

which would be equivalent to:

```
for k in range(nao):
  for l in range(nao):
    Fuv[0,0] += Duv[k,l]*eri[0,0,k,l]}
```

The second term: $\sum_{\kappa\lambda} D_{\kappa\lambda}(\mu\kappa|\nu\lambda)$, is the Exchange Term and has no classical analogue. It is a result of the Pauli principle in electrons of the same spin avoid each other.

Similarly to the Coulomb Term, this can be calculated as a four-fold loop over atomic orbitals, but could also be calculated through aggregation. For the contribution of this term to the Fock Matrix element [0,0]:

```
  -(Duv*eri[0,:,0]).sum()
```

would be equivalent to:

```
  for k in range(nao):
    for l in range(nao):
      Fuv[0,0] -= Duv[k,l]*eri(0,k,0,l)
```

Hint: For the first iteration:

```
    Fuv[0,0] = -32.57739541261037
    Fuv[2,5] = Fuv[5,2] = -1.6751501447185015
```

For the second iteration:

```
    Fuv[0,0] = -18.81326949992384
    Fuv[2,5] = Fuv[5,2] = -0.1708886336992761
```

Step 4b. Solve Eigenvalues and Eigenvectors of Roothaan Equations

Function Stub: `solve_Roothan_equations`

Now that we have formed a Fock Matrix, we have all we need to solve the Roothan equations. Recall the Roothaan equations take the following form:

$$\mathbf{FC} = \mathbf{SC}\boldsymbol{\varepsilon}$$

This is an eigenvalue problem, where the solutions translate to:

F is the Fock Matrix

S is the Overlap Matrix

ε: will be the eigenvalues and will be an array of the molecular orbital energies.

C: will be the eigenvectors and represent the translation of the atomic orbitals to the molecular orbitals.

To solve the Roothaan Equations through SciPy's `linalg.eigh` function (API doc here). You should be able to implment this in one line.

Hint

For the first iteration

```
  mol_e = [-32.57830292  -8.08153571  -7.55008599
           -7.36396923   7.34714487  -4.00229867
```

```
         -3.98111115]

  mol_c[0,:] = [-1.00154358e+00  2.33624458e-01  4.97111543e-16
                -8.56842145e-02  2.02299681e-29  4.82226067e-02
                -4.99600361e-16]
```

Step 4c. Calculate the Total Energy of the Current Iteration

Function Stub: `calc_tot_energy`

Now we have everything we need to calculate the current total Hartree-Fock energy of the molecule.

The formula for computing the total energy is:

$$E_{\text{tot}} = \frac{1}{2}\sum_{\mu\nu} D_{\mu\nu}(H_{\mu\nu}^{\text{core}} + F_{\mu\nu}) + E_{nuc}$$

You should be able to implement this equation in one line as all of these terms are either scalars or NumPy Matrices.

Hint:

For the first iteration:

```
  Etot =  8.0023670618
```

For the second iteration:

```
  Etot = -73.2857964211
```

Step 4d. Calculate the new Density Matrix

Function Stub: `form_density_matrix`

Next step is to derive the new updated Density Matrix from the molecular orbital coefficients we determined in Step 5b. The formula for computing the Density Matrix is as follows:

$$D_{\mu\nu} = 2\sum_{i} C_{\mu i}C_{\nu i}$$

Note: The summation here is over **i**, which indicates that is over the number of orbitals occupied by electrons, not the total number of orbitals. In water there are 10 electrons, and each orbital can take 2 electrons (one spin up and one spin down), so this number ends up being 5.

This variable is set in the stub function as `nelec`.

Hint:

For the first iteration:

```
Duv_new[0,0] = 2.130023428655504
Duv_new[2,5] = Duv_new[5,2] = -0.29226330209653156
```

Step 4e. Calculate the Energy Difference and RMS Difference of Density

For each iteration, we will need to check the difference in the total energy and the RMS of the density with the previous iteration. As stated above, the formulas to compute are as follows:

$$\Delta_E = |E_{\text{tot}}^t - E_{\text{tot}}^{t-1}| < \delta_1$$

$$\text{rms}_{\mathbf{D}} = \left[\sum_{\mu\nu} (D_{\mu\nu}^t - D_{\mu\nu}^{t-1})^2 \right]^{-1/2} < \delta_2$$

These operations are already computed in `main.py` so there is no need for you to implement.

Step 4f. Check for Convergence, if Converged, Exit

In this step, we compare the difference to the convergence criteria to see if we are finished. If both values are below the thresholds, then we end the loop with a `break` and print the final result.

This operation is already done in `main.py` and you do not need to implement.

Step 4g. If not Converged, update Density Matrix and Energy and do another iteration

If the SCF procedure has not met convergence, then we need to update the Density Matrix with the newly calculated one and repeat another iteration.

This operation is already done in `main.py` and you do not need to implement.

Final Answer

The final total energy should be: `-74.9420799282` after 25 iterations.

Getting Help

If you have any issues or questions, please submit to the Issues page of the main repository.

References

1. Programming Tutorial in Chemistry by Python, Daniel Crawford.
2. An Introduction to Hartree-Fock Molecular Orbital Theory, C. David Sherrill.

3. Szabo, A., Ostlund, N. S. (1996). Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory. Mineola: Dover Publications, Inc., available at CMU Library.

4. Recent developments in the PySCF program package, Q. Sun, X. Zhang, S. Banerjee, P. Bao, M. Barbry, N. S. Blunt, N. A. Bogdanov, G. H. Booth, J. Chen, Z.-H. Cui, J. J. Eriksen, Y. Gao, S. Guo, J. Hermann, M. R. Hermes, K. Koh, P. Koval, S. Lehtola, Z. Li, J. Liu, N. Mardirossian, J. D. McClain, M. Motta, B. Mussard, H. Q. Pham, A. Pulkin, W. Purwanto, P. J. Robinson, E. Ronca, E. R. Sayfutyarova, M. Scheurer, H. F. Schurkus, J. E. T. Smith, C. Sun, S.-N. Sun, S. Upadhyay, L. K. Wagner, X. Wang, A. White, J. Daniel Whitfield, M. J. Williamson, S. Wouters, J. Yang, J. M. Yu, T. Zhu, T. C. Berkelbach, S. Sharma, A. Yu. Sokolov, and G. K.-L. Chan, J. Chem. Phys. 153, 024109 (2020)

5. PySCF: the Python-based simulations of chemistry framework, Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. McClain, S. Sharma, S. Wouters, and G. K.-L. Chan, WIREs Comput. Mol. Sci. 8, e1340 (2018).

6. Libcint: An efficient general integral library for Gaussian basis functions, Q. Sun, J. Comp. Chem. 36, 1664 (2015).