Programming Projects on Basic Quantum Chemistry Methods

Hong-Zhou Ye*[1]

[1]*Department of Chemistry and Biochemistry and Institute for Physical Science and Technology*
University of Maryland, College Park, MD, 20742

Aug 2024

This document collects a few programming projects that help you build basic skills for quantum chemistry programming by turning the methods you learned in Szabo & Ostlund into computer programs. In addition to S & O, you may also find relevant chapters in the book *Molecular Electronic-Structure Theory* by Helgaker, Jørgensen, and Olsen (HJO) helpful. There are also excellent notes/tutorials available on similar topics:

- Notes by Prof. David Sherrill: link
- Programming tutorials by Prof. Eugene DePrince: link
- Programming tutorials by Prof. Daniel Crawford: link
- Programming tutorials by ajz34: link

In what follows, I will refer to relevant materials in these books/tutorials/notes from time to time. My personal experience is that seeing the same materials presented in a slightly different language/notations/order helps understand things.

1 Hartree-Fock (HF)

1.1 Theory

Here's a concise recap of the SCF algorithm for RHF from S&O:

1. Input:
   - Nuclear repulsion energy E_{nuc}
   - Number of occupied orbitals $N_{\text{occ}} = N_{\text{elec}}/2$
   - AO overlap matrix $S_{\mu\nu}$
   - 1e integrals (aka H^{core}) $h_{\mu\nu}$
   - 2e integrals (aka ERIs) $(\mu\nu|\lambda\sigma)$
2. Initial guess:
   (a) Initial MOs from core guess
   $$\mathbf{hC}^{(0)} = \mathbf{SC}^{(0)}\boldsymbol{\varepsilon}^{(0)} \tag{1}$$
   (b) Initial density matrix
   $$D^{(0)}_{\mu\nu} = 2\sum_{i}^{\text{occ}} C^{(0)}_{\mu i} C^{(0)*}_{\nu i} \tag{2}$$

*hzyechem@gmail.com

(c) Initial Fock matrix

$$F_{\mu\nu}^{(0)} = h_{\mu\nu} + \frac{1}{2}\sum_{\lambda\sigma}[2(\mu\nu|\lambda\sigma) - (\mu\sigma|\lambda\nu)]D_{\sigma\lambda}^{(0)} \tag{3}$$

(d) Initial SCF energy

$$E_{\text{tot}}^{(0)} = E_{\text{nuc}} + \frac{1}{2}\text{Tr }\mathbf{D}^{(0)}(\mathbf{h} + \mathbf{F}^{(0)}) \tag{4}$$

3. Entering SCF cycles. In cycle n, do

(a) Update MOs by diagonalizing Fock matrix from last cycle

$$\mathbf{F}^{(n-1)}\mathbf{C}^{(n)} = \mathbf{S}\mathbf{C}^{(n)}\boldsymbol{\varepsilon}^{(n)} \tag{5}$$

(b) Update density matrix

$$D_{\mu\nu}^{(n)} = 2\sum_{i}^{\text{occ}} C_{\mu i}^{(n)} C_{\nu i}^{(n)*} \tag{6}$$

(c) Update SCF energy

$$E_{\text{tot}}^{(n)} = E_{\text{nuc}} + \frac{1}{2}\text{Tr }\mathbf{D}^{(n)}(\mathbf{h} + \mathbf{F}^{(n)}) \tag{7}$$

(d) Check convergence by one or more of the following criteria:

- Energy criterion: $|E^{(n)} - E^{(n-1)}| < \epsilon_{\text{ene}}$, often chosen to be 10^{-8} Ha.
- Density matrix criterion: $\|\mathbf{D}^{(n)} - \mathbf{D}^{(n-1)}\|_{\text{F}} < \epsilon_{\text{dm}}$, often chosen to be 10^{-4} a.u., where $\|\cdot\|_{\text{F}}$ stands for the Frobenius norm.
- SCF commutator criterion: $\|\mathbf{F}^{(n)}\mathbf{D}^{(n)}\mathbf{S} - \mathbf{S}\mathbf{D}^{(n)}\mathbf{F}^{(n)}\|_{\text{F}} < \epsilon_{\text{comm}}$, often chosen to be 10^{-4} a.u. [Exercise: prove that this commutator does vanish at convergence, i.e., when the Roothaan equation (5) is satisfied.]

(e) If convergence is not achieved, update Fock matrix

$$F_{\mu\nu}^{(n)} = h_{\mu\nu} + \frac{1}{2}\sum_{\lambda\sigma}[2(\mu\nu|\lambda\sigma) - (\mu\sigma|\lambda\nu)]D_{\sigma\lambda}^{(n)} \tag{8}$$

and proceed to the next cycle.

1.2 Simple SCF code

Implement the SCF algorithm outlined in the previous section for RHF. You may want to first implement it by simply transforming the summation into for loops. Once you confirm the correctness of this naïve version, refactor it by replacing for loops with `numpy` matrix/tensor operations. You may find the following habits make your life much easier:

1. Make changes one block/equation at a time and always make sure that the code is still correct after every change.
2. Instead of overwriting the old code with new code in place, work on a new copy and keep the old code. In this way you can always go back for sanity checks if you need to.

To test your code, in directory `reference/ints`, you can find the necessary AO integrals $\mathbf{S}$, $\mathbf{h}$, and $\mathbf{V}$ and nuclear repulsion energy E_{nuc} for a water molecule with equilibrium bond length in the STO-3G basis set (stored as `npy` files, which can be loaded using `numpy.load`). Use your SCF code to find the RHF ground state for this water molecule. The reference RHF total energy from PySCF is -74.94502101 Ha.

1.3 Couple your code to PySCF

Study the script `reference/generate_reference.py` for how to generate AO integrals for an arbitrary molecule and basis set. Make your code take integrals and other necessary input parameters directly from PySCF in order to avoid the intermediate steps such as saving integrals to disk and counting electrons manually. Now your code can basically do RHF calculations for any closed-shell molecules! Try it out on the molecular structures provided in `reference/geom` using different basis sets such as cc-pVDZ, cc-pVTZ, def2-SVP, def2-TZVP in addition to the minimal STO-3G basis set used above. Confirm the correctness of your code against PySCF.

1. Do RHF calculations for water with equilibrium bond length (`h2o_eq.xyz`) using STO-3G, cc-pVDZ, and cc-pVTZ basis sets with both your code and PySCF. Plot the SCF energy error (or either of the other two convergence criteria) as a function of the SCF cycle. Do you observe that (i) your code in general requires many more cycles to convergence than PySCF and (ii) the number of SCF cycles needed for convergence increases more quickly when using a larger basis set? In section 1.5, we will learn a technique to accelerate the SCF convergence.
2. Do RHF calculations using STO-3G for `h2o_2eq.xyz` where both O−H bonds in water are stretched to twice of their equilibrium value (use `avogadro` or `VESTA` to visually verify this). You may observe that your SCF code fail to converge in this case. Plot the SCF energy as a function of the SCF cycle. Do you see that your SCF energy oscillates between two values? In section 1.4, we will learn a technique to stabilize SCF convergence in cases like this.

1.4 SCF with damping: help convergence in challenging cases

When molecules move away from their equilibrium structure such as in bond stretching, one may encounter SCF convergence problems. An example is `h2o_2eq.xyz`, where the two O−H bonds in a water molecule are stretched to twice their equilibrium value. If you plot the SCF energy as a function of the SCF cycle, you will see an oscillation between two energies. This suggests that each SCF update is too big and overshoots, causing an oscillation between two competing solutions. The convergence can be improved by simply "damping" the SCF update on the density matrix, i.e.,

$$\tilde{\mathbf{D}}^{(n)}(\beta) = (1-\beta)\mathbf{D}^{(n)} + \beta\mathbf{D}^{(n-1)} \tag{9}$$

where $\beta \in [0, 1]$ is an adjustable parameter that controls how strongly we damp the SCF update: $\beta = 0$ means no damping at all and reduces to the normal SCF algorithm, while $\beta = 1$ means completely damped, in which case the density matrix is not updating at all.

Implement damping in your SCF code. Note that the damping (9) should happen right before the Fock build (8), i.e., the energy evaluation (7) and convergence check should still be done using the unmodified density matrix.

1. Repeat the RHF calculation for the stretched water molecule with a few different values of β. You should see SCF convergence is now achievable with a large enough β. Is a large value for β (closer to 1) always better?
2. Try your damped SCF code on water molecule at equilibrium geometry. You should see that convergence is actually slowed down by damping for cases where SCF convergence is easy. So damping is better turned off by default and turned on whenever you encounter a challenging case where the regular SCF fails.

1.5 SCF with DIIS

The SCF algorithm implemented above is the so-called fixed-point method (i.e., essentially we are solving $f(x) = x$ for x where x is the density matrix), which is the simplest algorithm for converging SCF. The simple damping discussed above unfortunately does not accelerate the convergence for molecules at equilibrium geometry – it actually does the opposite and slows down the convergence.

One popular method to accelerate SCF convergence is the direct inversion of the iterative subspace (DIIS) method, first introduced by Peter Pulay. Read one of the tutorials on DIIS (e.g., DePrince `Hartree-Fock 3`; Crawford `Project #8`; HJO 10.6.2) and implement DIIS for your SCF code. You should observe that the number of SCF cycles required for convergence is significantly reduced and close to the PySCF results.

Hint for implementation: in each SCF cycle, you should still build the Fock matrix $\mathbf{F}^{(n)}$ normally using eq. (8). But instead of proceeding to the next cycle which will diagonalize $\mathbf{F}^{(n)}$, you update $\mathbf{F}^{(n)}$ to $\tilde{\mathbf{F}}^{(n)}$ according to DIIS, i.e.,

$$\tilde{\mathbf{F}}^{(n)} = \sum_{m=0}^{h-1} c_m \mathbf{F}^{(n-m)} \tag{10}$$

where h is the length of history, usually chosen to be 8 – 16.

2 Second-order Møller-Plesset perturbation theory (MP2)

MP2 is the simplest electron correlation theory. In spin-orbital, the MP2 correlation energy reads

$$E_{\text{corr}}^{\text{MP2}} = \frac{1}{4} \sum_{ijab} \frac{|\langle ij||ab\rangle|^2}{\Delta_{iajb}} \tag{11}$$

where $\Delta_{iajb} = \varepsilon_i + \varepsilon_j - \varepsilon_a - \varepsilon_b$ and $\langle ij||ab\rangle = \langle ij|ab\rangle - \langle ij|ba\rangle$. For a RHF reference, it's more convenient to use the spin-summed formula (we also switched to the chemists' notation for ERIs)

$$E_{\text{corr}}^{\text{MP2}} = \sum_{ijab} \frac{(ia|jb)^*[2(ia|jb) - (ib|ja)]}{\Delta_{iajb}} \tag{12}$$

where the ERIs are now in spatial orbitals

$$(ia|jb) = \sum_{\mu\nu\lambda\sigma} (\mu\nu|\lambda\sigma) C_{\mu i}^* C_{\mu a} C_{\lambda j}^* C_{\sigma k} \tag{13}$$

Derive the spin-summed expression (12) from the spin-orbital formula (11) and convince yourself that they are equivalent to Eqns. 6.74 and 6.72 in S&O (note that we use modern notations here where i, j denote occupieds and a, b denote virtuals).

Implementing a function for calculating the MP2 correlation energy using the RHF solutions generated by either your SCF code or PySCF. Test your implementation using the water/STO-3G system. The reference MP2 energy from PySCF is -0.03108253 Ha.

Hint: (i) You may find Crawford `Project #4` helpful. (ii) In our cases all orbitals are real-valued, and you can safely ignore the complex conjugate.

Notes on integral transform: If one performs the integral transform using eq. (13), the computational cost scales $N_{\text{occ}}^2 N_{\text{vir}}^2 N_{\text{ao}}^2 \sim \mathcal{O}(N^8)$. A smarter way to do this is to break it down into four $\mathcal{O}(N^5)$ steps:

$$(i\nu|\lambda\sigma) = \sum_{\mu} (\mu\nu|\lambda\sigma) C_{\mu i}^* \qquad N_{\text{occ}} N_{\text{ao}}^4 \sim \mathcal{O}(N^5) \tag{14a}$$

$$(i\nu|j\sigma) = \sum_{\lambda} (i\nu|\lambda\sigma) C_{\lambda j}^* \qquad N_{\text{occ}}^2 N_{\text{ao}}^3 \sim \mathcal{O}(N^5) \tag{14b}$$

$$(ia|j\sigma) = \sum_{\nu} (i\nu|j\sigma) C_{\nu a} \qquad N_{\text{occ}}^2 N_{\text{vir}} N_{\text{ao}}^2 \sim \mathcal{O}(N^5) \tag{14c}$$

$$(ia|jb) = \sum_{\sigma} (ia|j\sigma) C_{\sigma b} \qquad N_{\text{occ}}^2 N_{\text{vir}}^2 N_{\text{ao}} \sim \mathcal{O}(N^5) \tag{14d}$$

Once the $(ov|ov)$-type integrals are obtained, evaluating the MP2 correlation energy using eq. (12) requires only $N_{\text{occ}}^2 N_{\text{vir}}^2 \sim \mathcal{O}(N^4)$ cost, which is lower than the integral transform step. Therefore, the cost of MP2 is dominated by integral transformation, which scales as $\mathcal{O}(N^5)$.