A tutorial on writing a Hartree Fock program in C++

Rui Li

February 25, 2023

1 Introduction

Hartree Fock has always been the everyday keyword in electronic structure - the first step of understanding quantum chemistry, the basis of post-HF methods, the sandbag of DFTs and semi-empirical methods, and so on.

Hartree-Fock method is known as a mean-field theory, i.e. the interaction among electrons is treated as a potential function over the real space for a single electron being considered, and the movement of this electron will be approximated as another potential that affects other electrons. Eventually we get the orbitals that give self-consistent results in terms of these 'mean-field' potentials.

Well, for sure we will not directly calculate these potentials, and honestly I do not like understanding Hartree-Fock in this way. I prefer ScholarPedia's way[1] of telling Hartree-Fock theory, which starts from a second quantization description of the energy, and the Fock matrix is defined as the derivative of the energy w.r.t. the density matrix of the system, and then we can solve the self-consistent equation

$$FC = SCE \tag{1.1}$$

and get the molecular orbitals. I am not super good at telling the mathematical background, plus I believe there are lots of textbooks that have better explanations, so I will just skip the part of explaining Hartree Fock.

What I will be trying to elaborate here is how to write a Hartree-Fock program with mathematical details behind each step of it.

C++ has a lot more tools than C, which helps making the program pointer-free. I recall the days and months dealing with memory leaks and segmentation faults, yet the code being so slow- maybe this time with C++, I can finally claim that I write good codes with awesome quality. Well, maybe not.

[1] http://www.scholarpedia.org/article/The_Hartree-Fock_method

Before going into introducing C++ features that C does not have - which might be needed for C-only learners - I would like to recommend Sourcetrail[2] that sorts out the code tree of a repository. I believe this will be extremely helpful when tracking the workflows and function calls. CLion is also a good IDE for C++ - even when only reading is required for a repository. You can use function jump (ctrl + left-click) and find usage to track the function calls, and the highlights of variables help reveal the logic of a function.

2 C++ features

We will only need to cover member functions in structs and then templates that can add tons of simplicity and functionality.

2.1 Member functions

Structs are originally intended, in my opinion, to pack the data, so that we can avoid having super long list of arguments in a function. For example, instead of

```
double atom_distance (double A_x,
                      double A_y,
                      double A_z,
                      double B_x,
                      double B_y,
                      double B_z) {...}
```

we can have

```
double atom_distance (XYZ A_coordinates,
                      XYZ B_coordinates) {...}
```

with struct definition

```
struct XYZ {
    double x;
    double y;
    double z;
}
```

and the members can be called, for example, `A_coordinates.x`.

Further a concise struct name can also help people have a clear idea about what this argument is about and thus good for type checking.

There are some information that can be directly derived from the members, for example, a molecule with xyz coordinates and atomic numbers defined can generate its atomic repulsion energy. Instead of writing

[2] https://github.com/CoatiSoftware/Sourcetrail

```
struct Atom {...}

double atomic_repulsion_energy(const Atom atom) {...}
```

we can do

```
struct Atom {
    ... geometry;
    ... atomic_numbers;

    double atomic_repulsion_energy() const {...}
}
```

so that the function can be called more easily by `atom.atomic_repulsion_energy()`. Mind the keyword `const` that helps avoid unintended change of the structs and allow optimization by the compiler.

2.2 Templates

Mathematically the numbers to be added do not require the types of the numbers - but in static programming they will. So it does not work if we write

```
double add (double a, double b) {...}

... {
 float a;
 float b;
 float c = add(a,b);
}
```

Then we can use templates to make it more generic:

```
template<typename T>
T add (T a, T b) {...}

... {
 float a;
 float b;
 float c = add<float>(a, b);
}
```

So that compiler will help generate function that puts `float` type in the place of `T`.

The templates also work for structs, for example

```
template<typename T>
struct Vector {
 T * data;
 unsigned long long n_elem;
```

```
 Vector<T> transpose() const {...}
};
```

This way the vector can hold any arbitrary types, but with generic operations of a vector. This will be similar to the vector in standard library `std::vector`, and the vector in Armadillo library `arma::Col` which is mathematically more convenient.

3 Some Mathematics

3.1 Overlap matrix

The definition goes like this:

$$S_{ij} = \langle \phi_i | \phi_j \rangle = \int \phi_i(\boldsymbol{r}) \phi_j(\boldsymbol{r}) d\boldsymbol{r}. \tag{3.1}$$

As we are using Gaussian functions- or Gaussian functions multiplied by some polynomials- for the orbitals, we actually have an analytical expression for the integral.

$$\int_{\mathrm{R^n}} dX\, f(X)\, e^{-\frac{1}{2}X^T A X + B^T X} = \sqrt{\frac{(2\pi)^n}{\det A}} e^{\frac{1}{2}B^T A^{-1} B} \exp\left(\frac{1}{2}(A^{-1})_{ij} \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j}\right) f(X)\Big|_{X=B} \tag{3.2}$$

where X is a vector of arbitrary number of variables, and f being a polynomial function of X. The exponential function on the right hand side should be understood as

$$\exp\{\hat{g}\} = \sum_n \frac{\hat{g}^n}{n!} \tag{3.3}$$

It will be super hard to directly implement such expression in static programming languages, so we turn to some recursion relations of the integral, thanks to the various properties offered by Gaussian functions.

To define a Gaussian function,

$$g(\vec{r}, \alpha, \boldsymbol{a}, \vec{A}) = (x - A_x)^{a_x} (y - A_y)^{a_y} (z - A_z)^{a_z} \exp\left(-\alpha |\vec{r} - \vec{A}|^2\right) \tag{3.4}$$

And there is a Gaussian Product Theorem:

Theorem 3.1. *Multiplication of two Gaussian functions can be expressed as a sum of Gaussian functions,*

$$g(\vec{r}, \alpha, \mathbf{0}, \vec{A}) g(\vec{r}, \beta, \mathbf{0}, \vec{B}) = \exp\left(-\xi |\overline{AB}|^2\right) g(\vec{r}, \zeta, \mathbf{0}, \vec{P}), \tag{3.5}$$

where$\zeta = \alpha + \beta$, $\xi = \frac{\alpha\beta}{\zeta}$, $\vec{P} = \frac{\alpha\vec{A} + \beta\vec{B}}{\zeta}$, $\overline{AB} = \vec{A} - \vec{B}$. *For Gaussian functions with polynomials, we also have*

$$(x - A_x)^{a_x} = [(x - P_x) + \overline{PA}_x]^{a_x} = \sum_{i=0}^{a_x} C_{a_x}^i (x - P_x)^i P A_x^{a_x - i} \tag{3.6}$$

so that multiplication of two Gaussian functions can be expressed as

$$g_1 g_2 = \exp\big(-\xi|AB|^2\big)\exp\Big(-\zeta|\vec{r}-\vec{P}|^2\Big) \times \prod_{k=x,y,z}\sum_{i=0}^{i=a_k+b_k}(k-P_k)^i f_i(a_k,b_k,\overline{PA}_k,\overline{PB}_k), \tag{3.7}$$

where

$$f_i(a_k,b_k,\overline{PA}_k,\overline{PB}_k) = \sum_{n=0}^{n=i} C_{a_k}^n C_{b_k}^{i-n}\overline{PA}_k^{a_x-n}\overline{PB}_k^{b_x-n+k} \tag{3.8}$$

Well it is slightly off the topic of recursion relations, but it is important to know that the multiplication of Gaussian orbitals in those molecular integrals can be parsed as some other Gaussian functions, so that we only need to care about Gaussian functions.

OK, now back to the recursion relation for the overlap integral- first thing first, the exponential part can be decomposed into three dimension,

$$e^{-\alpha(r-r_0)^2} \Rightarrow e^{-\alpha(x-x_0)^2}e^{-\alpha(y-y_0)^2}e^{-\alpha(z-z_0)^2} \tag{3.9}$$

$\rightarrow$ This allows decomposition of integrals into three independent dimensions. Next thing is the derivative of Gaussian function,

$$\frac{\partial}{\partial x}[x^n e^{-\alpha x^2}] = nx^{n-1}e^{-\alpha x^2} - 2\alpha x^{n+1}e^{-\alpha x^2} \tag{3.10}$$

where we can see that the derivative is also converted to a sum of two Gaussian functions, or in other way, a Gaussian function with higher-order polynomial can be decomposed into a sum of a derivative of Gaussian function and another Gaussian function with lower-order polynomial. This gives a good hint of a possible recursion relation for the overlap integral. Focusing only on the x component of the overlap integral,

$$(\phi_1|\phi_2) \rightarrow \int dx (x-x_1)^a e^{-\alpha(x-x_1)^2}(x-x_2)^b e^{-\beta(x-x_2)^2} \rightarrow \langle a|b\rangle \tag{3.11}$$

where a and b are used to directly represent the orbitals with different angular momentum. Using the property of integral-derivative pair,

$$\int dx\, \partial_x(\phi_1|\phi_2) = 0 \tag{3.12}$$

we have

$$a(a-1|b) - 2\alpha(a+1|b) + b(a|b-1) - 2\beta(a|b+1) = 0 \tag{3.13}$$

To eliminate $(a|b+1)$, as it is not reducing the total angular momentum number of the integral, we can utilize the fact

$$(x-x_2) \rightarrow (x-x_1) + (x_1-x_2) \tag{3.14}$$

This gives

$$2(\alpha+\beta)(a+1|b) = a(a-1|b) + b(a|b-1) - 2\beta(x_1 - x_2)(a|b) \tag{3.15}$$

And this also applies to y and z component. Now that we have a recursion relation, we can decompose the original over integral into a linear combination of 0-0 type overlap integral

$$\langle\phi_1|\phi_2\rangle = c_1\langle 0|0\rangle + c_2\langle 0|0\rangle + ... \tag{3.16}$$

whose value can be easily calculated, as they are only a simple multiplication of Gaussian functions.

As a summary, a recursion relation of the integrals can help us decompose the complicated ones to a linear combination of simple Gaussian integrals.

3.2 Kinetic Integrals

In quantum mechanics, the momentum operator is a derivative operator - and this is always easy to do, as the derivative operator on a Gaussian function will only convert it to the Gaussian function multiplied by some polynomials, and the polynomials represent the angular momentum. It can also have some recursive algorithm, which is written in Obara-Saika's paper, but it also works having a Laplace operator performed on one of the orbital pairs, and then calculate the overlap integral, mathematically written as

$$\langle a| -\frac{\nabla^2}{2}|b\rangle \rightarrow \langle a|(-\frac{\nabla^2}{2}|b\rangle) \tag{3.17}$$

3.3 Nuclear Attraction Integral

The definition is

$$Z_{ij} = \langle\phi_i|\frac{1}{|\boldsymbol{r}-\boldsymbol{r}_l|}|\phi_j\rangle = \int \frac{\phi_i(\boldsymbol{r})\phi_j(\boldsymbol{r})}{|\boldsymbol{r}-\boldsymbol{r}_l|}d\boldsymbol{r} \tag{3.18}$$

where $\boldsymbol{r}_l$ is the coordinates of the nucleus.

Obviously, the new term $\frac{1}{|\boldsymbol{r}-\boldsymbol{r}_l|}$ messes everything up, especially regarding the painful truth that

$$\frac{1}{|\boldsymbol{r}-\boldsymbol{r}_l|} = \frac{1}{\sqrt{(r_x - r_{l,x})^2 + (r_y - r_{l,y})^2 + (r_z - r_{l,z})^2}} \tag{3.19}$$

Here comes our savior, Gaussian function,

$$|\boldsymbol{r}_1 - \boldsymbol{r}_2|^{-1} = \frac{2}{\pi^{1/2}}\int_0^\infty du \exp\left[-(\boldsymbol{r}_1 - \boldsymbol{r}_2)^2 u^2\right]. \tag{3.20}$$

Using our previous notation,

$$\exp\left[-(\boldsymbol{r}-\boldsymbol{r}_l)^2 u^2\right] = g(\boldsymbol{r}; u^2, \boldsymbol{0}, \boldsymbol{r}_l) \tag{3.21}$$

Then the problem reduces to a three-center overlap integral,

$$\langle\phi_i|\frac{1}{|\boldsymbol{r}-\boldsymbol{r}_l|}|\phi_j\rangle = \frac{2}{\pi^{1/2}}\int_0^\infty du\,(\phi_i|\mathbf{0}_{\boldsymbol{r}_2}^{u^2}|\phi_j) \tag{3.22}$$

Just like the previous overlap integral, we also have a recursion relation for the three-center version,

$$\begin{aligned}(\boldsymbol{a}+\mathbf{1}_i|\boldsymbol{c}|\boldsymbol{b}) =& (G_i - A_i)\,(\boldsymbol{a}|\boldsymbol{c}|\boldsymbol{b}) + \frac{a_i}{2\,(\zeta+\zeta_c)} \\ &\times (\boldsymbol{a}-\mathbf{1}_i|\boldsymbol{c}|\boldsymbol{b}) + \frac{b_i}{2\,(\zeta+\zeta_c)}\,(\boldsymbol{a}|\boldsymbol{c}|\boldsymbol{b}-\mathbf{1}_i) \\ &+ \frac{c_i}{2\,(\zeta+\zeta_c)}\,(\boldsymbol{a}\,|\boldsymbol{c}-\mathbf{1}_i|\,\boldsymbol{b})\end{aligned} \tag{3.23}$$

where $\boldsymbol{G} = \frac{\zeta\boldsymbol{P}+\zeta_c\boldsymbol{C}}{\zeta+\zeta_c}$. Because we do not have any angular momentum for the 'Gaussian orbital' transformed from $1/\sqrt{\boldsymbol{r}-\boldsymbol{r}_l}$,

$$\begin{aligned}\left(\boldsymbol{a}+\mathbf{1}_i|\mathbf{0}_{\boldsymbol{r}_l}^{u^2}|\boldsymbol{b}\right) =& (G_i - A_i)\,(\boldsymbol{a}|\mathbf{0}_{\boldsymbol{r}_l}^{u^2}|\boldsymbol{b}) + \frac{a_i}{2\,(\zeta+u^2)} \\ &\times \left(\boldsymbol{a}-\mathbf{1}_i|\mathbf{0}_{\boldsymbol{r}_l}^{u^2}|\boldsymbol{b}\right) + \frac{b_i}{2\,(\zeta+u^2)}\left(\boldsymbol{a}|\mathbf{0}_{\boldsymbol{r}_l}^{u^2}|\boldsymbol{b}-\mathbf{1}_i\right)\end{aligned} \tag{3.24}$$

where $G_i - A_i$ is

$$G_i - A_i = \frac{\beta(B_i - A_i) + u^2(C_i - A_i)}{\zeta+u^2}, \quad \boldsymbol{C} = \boldsymbol{r}_l \tag{3.25}$$

This means when unraveling recursion relations, $\frac{u^2}{\zeta+u^2}$ and $\frac{u^2}{\zeta+u^2}$ will contribute.

Fortunately, this is not the end of the world. We can perform a transformation

$$\frac{1}{\zeta+u^2} = \frac{1}{\zeta}\left(1 - \frac{u^2}{\zeta+u^2}\right) \tag{3.26}$$

This can convert $\frac{u^2}{\zeta+u^2}$ to $\frac{u^2}{\zeta+u^2}$. This indicates that there will only be some power series of $(\frac{u^2}{\zeta+u^2})^m$ that get multiplied to the final result. Therefore, we can define

$$(\boldsymbol{a}|\frac{1}{|\boldsymbol{r}-\boldsymbol{r}_l|}|\boldsymbol{b})^{(m)} = \frac{2}{\pi^{1/2}}\int_0^\infty \left(\frac{u^2}{\zeta+u^2}\right)^m du\,(\boldsymbol{a}|\mathbf{0}_{\boldsymbol{r}_2}^{u^2}|\boldsymbol{b}) \tag{3.27}$$

and the $m = 0$ case represents the original version. The term $(\boldsymbol{a}|\mathbf{0}_{\boldsymbol{r}_2}^{u^2}|\boldsymbol{b})$ will then be simplified to $(\mathbf{0}_A|\mathbf{0}_C|\mathbf{0}_B)$ and performing standard Gaussian function integral we have

$$(\mathbf{0}_A\,|\mathbf{0}_C|\,\mathbf{0}_B) = \left(\frac{\pi}{\zeta_a+\zeta_b+\zeta_c}\right)^{3/2}\kappa_{abc}, \tag{3.28}$$

where

$$\kappa_{abc} = \exp\left[-\xi(\boldsymbol{A}-\boldsymbol{B})^2\right]\exp\left[-\frac{\zeta\zeta_c}{\zeta+\zeta_c}(\boldsymbol{P}-\boldsymbol{C})^2\right]. \tag{3.29}$$

For $\left(\frac{\pi}{\zeta_a+\zeta_b+\zeta_c}\right)^{3/2}$, we can use the previous transformation $\frac{1}{\zeta+u^2}$ and then convert

$$\int_0^\infty \left(\frac{u^2}{\zeta+u^2}\right)^{(m+\frac{3}{2})} \exp\left[-\frac{u^2}{\zeta+u^2}\zeta(\boldsymbol{P}-\boldsymbol{r}_2)^2\right]\, du.$$

We replace the variables

$$t^2 = \frac{u^2}{\zeta+u^2}. \tag{3.30}$$

Then

$$\frac{1}{t^2} = 1 + \frac{\zeta}{u^2}. \tag{3.31}$$

This means

$$u = \sqrt{\frac{\zeta t^2}{1-t^2}}. \tag{3.32}$$

This will transform the original integral to

$$\int_0^1 t^{2m} \exp\left[-\zeta(\boldsymbol{P}-\boldsymbol{r}_2)^2 t^2\right]\, dt.$$

It is hard to get the analytical value, but this type of function, called Boys function, has an equivalent form

$$\int_0^1 t^{2m} \exp\left(-Tt^2\right) dt = \frac{1}{2}\, T^{-\frac{2m+1}{2}} \left[\Gamma\left(\frac{2m+1}{2}\right) - \Gamma\left(\frac{2m+1}{2}, T\right)\right], \tag{3.33}$$

where $\Gamma(n,x)$ is the incomplete Gamma function. The Gamma functions can be calculated using some special function libraries, e.g. one in Boost library. Nevertheless, there is a Rys quadrature method that transforms the original polynomial of t's to a linear combination of Rys polynomials whose coefficients are calculated using a generated set of grid points and weights specified by the value of T.

3.4 Electron Repulsion Integral

Similar to the nuclear attraction integral, it also introduces the $\frac{1}{r}$ term that will eventually convert into Boys functions. The definition goes as

$$[ij|kl] = \int_{\mathbb{R}^6} d\boldsymbol{r}_1 d\boldsymbol{r}_2\, \phi_i(\boldsymbol{r}_1)\phi_j(\boldsymbol{r}_1)\frac{1}{|\boldsymbol{r}_1-\boldsymbol{r}_2|}\phi_k(\boldsymbol{r}_2)\phi_l(\boldsymbol{r}_2) \tag{3.34}$$

where we can see that, exchanging ϕ_i and ϕ_j do not change the value of the integral, neither for exchanging ϕ_k and ϕ_l, neither for swapping $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$. Therefore, the ERI has 8-fold symmetry, which can be utilized to reduce computation time.

Reducing this integral to a recursive form will be similar to Nuclear attraction integrals, by transforming $\frac{1}{|\boldsymbol{r}_1-\boldsymbol{r}_2|}$ to Gaussian function, and performing recursive algorithms to a linear combination of integrals with no angular momentums. Detailed math can be found in this article[3].

[3] *Efficient Electronic Integrals and their Generalized Derivatives for Object Oriented Implementations of Electronic Structure Calculations*, by N. FLOCKE, V. LOTRICH, with DOI 10.1002/jcc.21018.

3.5 Energy expression, Fock matrix

The energy of Hartree Fock is expressed as

$$E = H^{ij} a_i^\dagger a_j + \frac{1}{2} v^{ijkl} a_i^\dagger a_j a_k^\dagger a_l + \text{const} \tag{3.35}$$

where I used Einstein's summation convention- the same alphabet that appears in upper index and the lower index represent summation in that dimension. H^{ij} is one-electron, core hamiltonian including the kinetic energy and nuclear attraction integrals, and v^{ijkl} is the two-electron part,

$$v^{ijkl} \equiv \langle jl|ik\rangle \equiv [ij|kl] \tag{3.36}$$

The constant term mainly contains the electric repulsion from atomic cores, which can be directly calculated from the molecular geometry.

It can be converted to a density matrix form

$$E = H^{ij} D_{ji} + \frac{1}{2} \bar{v}^{ijkl} D_{ji} D_{lk} \tag{3.37}$$

Where it becomes $\bar{v} \equiv v^{ijkl} - v^{ikjl}$ due to an anti-commutation rule introduced by the fermionic creation/annihilation operators. Here the $ijkl$ are usually indices for spin orbitals, but if we are dealing with Restricted Hartree-Fock, we can have

$$\bar{v}^{ijkl} = v^{ijkl} - \frac{1}{2} v^{ikjl} \tag{3.38}$$

due to a special symmetry between alpha electrons (spin up) and beta electrons (spin down). In this case, the occupation of electrons per orbital will become 2, and $ijkl$ will be representing molecular orbitals. The first term is called coulomb integral, and the second term exchange integral (as we exchanged the order of j and k).

The Fock matrix is the derivative of energy w.r.t. the density matrix, so

$$F^{ij} = \frac{\partial E}{\partial D_{ij}} = H^{ij} + \bar{v}^{ijkl} D_{lk} \tag{3.39}$$

To restore the energy, we can have

$$E = \frac{1}{2} F^{ij} D_{ji} + \frac{1}{2} H^{ij} D_{ji} \tag{3.40}$$

instead of performing the contraction among the two-electron tensor and density matrices.

And from this we can see that the Fock matrix is dependent on the density matrix, which is unknown when we are solving the Hartree-Fock equation, and this becomes the reason for the necessity of SCF routine.

The Fock matrix should follow

$$FC = SCE \tag{3.41}$$

where C is a set of molecular orbital coefficient vectors. The converged result should give consistent set of molecular orbitals that are put into Fock matrix and that pop out from the eigensystem.

3.6 SCF routine

To start an SCF iteration, we need to generate an initial set of molecular orbitals, so that the Fock matrix can be calculated, and perform a diagonalization to generate a new set of molecular orbitals. Such an initial set is called initial guess, and there are multiple ways of generating initial guess. The easiest one should be an 'identity' matrix - but not exactly an identity matrix, as we need to obey the normalization offered by the overlap matrix. It can also be the eigenstates of core hamiltonian, which does not require SCF iteration. Usually this is good enough, especially when the two-electron-integral part is not super important in the system.

After we obtain an initial guess, we can try solving the eigensystem

$$F(C^{\text{in}})C^{\text{out}} = SC^{\text{out}}E \tag{3.42}$$

where C^{in} represents an initial guess, or some intermediate set of molecular orbitals that are not yet converged, and C^{out} is the molecular orbital solution that fits in this eigensystem problem.

Usually if SCF is not converged, we will see a huge difference in C^{in} and C^{out}, so that we need to generate a new set of molecular orbitals and perform another iteration of SCF. Most likely Cout will be closer to the final solution - otherwise the optimization (of molecular orbitals) will never come to an end. Therefore if we input C^{out} into the next iteration of SCF, we can see a reduced eigenvalues and energy. To have faster convergence, we can mix the two set of molecular orbitals and generate a new one that will go into next iteration. The mixing method can also be variant, either a fixed ratio of mixing, or with some update methods like DIIS.

Eventually we will have a converged solution, where the eigenvalues and energy only change within a very small value, for example 10^{-5} Hartree. Note that it can not necessarily be

$$|C^{\text{in}} - C^{\text{out}}| < 10^{-5} \tag{3.43}$$

or some other small numbers as tolerance, because the molecular orbitals can have arbitrary phase factor, i.e. it can be with or without a negative sign and still fit the eigensystem.

4 Introduction to the code structure

Level-wise the code is split as

1. `main.cpp` - where the program starts
2. `run.cpp` - handles the commands specified in input file (perform single SCF calculation, calculate the gradient, etc.)

3. `hf/rhf.cpp` - where the preparation of restricted Hartree-Fock method is performed

4. `scf/scf.h` - where a templated scf calculation is performed

Timeline-wise the code follows

1. `geometry/resolve.h` Parse the atoms specified in input file

2. `basis/basis.cpp` Construct the basis according to the geometry and given basis name

3. `hf/rhf.cpp` Construct hartree-fock related objects, including fock matrix builder, energy builder (generator of function that returns the targeting objects), and all the necessary integrals.

4. `integral/obara_saika.cpp`, `integral/rys_quadrature.cpp` where the integrals are generated.

5. `scf/scf.h` Generic SCF routine.