Hartree-Fock理论

Sun Xinyu
sunxinyu347@gmail.com

2024 年 1 月 10 日

目录

1 前言

Hartree-Fock（后文简称HF）方法是量子化学最经典的波函数方法，如今常用来为后续高级别算法提供初猜、选择活性空间等。我们仍有必要学习HF代码，其中的思想和专有名词是应用量子化学计算的必要知识储备，让初学者对量化软件的逻辑有一个基础且必须的了解，避免糊算乱算，并且在自己的计算过程中遇到的错误能有合理的解释和解决办法。

这是南开大学彭谦课题组新人入组手册系列之一，gitlab地址为$https://github.com/Yxwxwx/Penglab_tutorial$

2 Hartree-Fock理论

本章要求熟练掌握线性代数相关知识，熟悉《结构化学》中，量子力学基础部分。

2.1 高斯型基函数（GTOs）

我们使用的基组函数一般都是GTO，其数学表达形式为

$$|\phi^{GTO}\rangle = \left(\frac{2a}{\pi}\right)^{3/4} e^{-a\boldsymbol{r}^2}$$

我们成为原始的Gaussian函数（primitive）；比如STO-3G对于H原子，使用三个GTO拟合一个STO，那么Gaussian函数的线性组合（CGTO）可以写为

$$\begin{aligned}|\phi^{CGTO}(r)\rangle =& d_1 \times \phi^{GTO}(a_1, \boldsymbol{r}) \\ &+ d_2 \times \phi^{GTO}(a_2, \boldsymbol{r}) \\ &+ d_3 \times \phi^{GTO}(a_3, \boldsymbol{r})\end{aligned}$$

其中的系数d和指数a通过读取基组文件得到。

```
BASIS "ao basis" PRINT
#BASIS SET: (3s) -> [1s]
H    S
      3.42525091              0.15432897
      0.62391373              0.53532814
      0.16885540              0.44463454
```

2.2 多电子Hartree-Fock近似

2.2.1 Fock算符

根据变分原理，对于Slater行列式形式的波函数中，最优的波函数对应的是最低的能量：

$$E_0 = \langle \Psi_0 | \mathbf{H} | \Psi_0 \rangle \tag{1}$$

在单电子近似下，我们假设一个有效的单电子算符$f(i)$，被称为Fock算符，其形式为：

$$f(i) = -\frac{1}{2}\Delta_i^2 - \sum_{A-1}^{M} \frac{Z_A}{r_{iA}} + v^{HF}(i) \tag{2}$$

$v^{HF}(i)$是i电子和其他电子的相互作用产生的平均势。所谓“自洽场”中的场，可以简单理解为这个平均势。Hartree-Fock近似的精髓在于将复杂的多

电子问题转化为单电子问题，单电子问题使用平均的方式处理复杂的电子排斥。
Fock算符是轨道的本征函数：

$$f|\chi_a\rangle = \epsilon_a|\chi_a\rangle \tag{3}$$

χ_a是自旋轨道波函数，后文会经常用到。

2.2.2 自旋轨道下的Hartree-Fock能量方程

基态波函数的Slater行列式形式为：

$$\Psi_0 = \frac{1}{2^{1/2}}\begin{vmatrix}\chi_1(x_1) & \chi_2(x_1)\\ \chi_1(x_2) & \chi_2(x_2)\end{vmatrix} = |\chi_1\chi_2\rangle \tag{4}$$

以极小基H_2模型考虑。在Born-Oppenheimer近似下，双电子体系的哈密顿量为：

$$\begin{aligned}\mathbf{H} &= \left(-\frac{1}{2}\Delta_1^2 - \sum_A \frac{Z_A}{r_{1A}}\right) + \left(-\frac{1}{2}\Delta_2^2 - \sum_A \frac{Z_A}{r_{2A}}\right) + \frac{1}{r_{12}}\\ &= h(1) + h(2) + \frac{1}{r_{12}}\end{aligned} \tag{5}$$

其中$h(1)$就是电子1的“核哈密顿量”，表示电子在核的势场中的动能和势能，对应后文的“单电子积分”，$\frac{1}{r_{12}}$对应双电子部分。为方便，我们将总哈密顿分成单电子和双电子部分:

$$\begin{aligned}\mathbf{O}_1 &= h(1) + h(2)\\ \mathbf{O}_2 &= r_{12}^{-1}\end{aligned} \tag{6}$$

矩阵元$\langle\Psi_0|\mathbf{H}|\Psi_0\rangle = \langle\Psi_0|\mathbf{O_1}|\Psi_0\rangle + \langle\Psi_0|\mathbf{O_2}|\Psi_0\rangle$

$$\begin{aligned}\langle\Psi_0|h(1)|\Psi_0\rangle = \int dx_1 dx_2 [2^{\frac{-1}{2}}(\chi_1(x_1)\chi_2(x_2) - \chi_2(x_1)\chi_1(x_2))]^*\\ \times h(r_1)[2^{\frac{-1}{2}}(\chi_1(x_1)\chi_2(x_2) - \chi_2(x_1)\chi_1(x_2))]\end{aligned} \tag{7}$$

$$\begin{aligned}= \frac{1}{2}\int dx_1 dx_2 [\chi_1^*(x_1)\chi_2^*(x_2)h(r_1)\chi_1(x_1)\chi_2(x_2) - \chi_1^*(x_1)\chi_2^*(x_2)h(r_1)\chi_2(x_1)\chi_1(x_2)\\ - \chi_2^*(x_1)\chi_1^*(x_2)h(r_1)\chi_1(x_1)\chi_2(x_2) + \chi_2^*(x_1)\chi_1^*(x_2)h(r_1)\chi_2(x_1)\chi_1(x_2)\end{aligned} \tag{8}$$

根据自旋轨道的正交归一性，将x_2积掉，公式(6)只有两项：

$$\langle\Psi_0|h(1)|\Psi_0\rangle = \frac{1}{2}\int dx_1 \chi_1^*(x_1)h(r_1)\chi_1(x_1) + \frac{1}{2}\int dx_1 \chi_2^*(x_1)h(r_1)\chi_2(x_1) \tag{9}$$

对于重复操作，可得：$\langle\Psi_0|h(1)|\Psi_0\rangle = \langle\Psi_0|h(2)|\Psi_0\rangle$ 则：

$$\langle\Psi_0|\mathbf{O_1}|\Psi_0\rangle = \int dx_1\chi_1^*(x_1)h(r_1)\chi_1(x_1) + \int dx_1\chi_2^*(x_1)h(r_1)\chi_2(x_1) \tag{10}$$

上式被称为**单电子积分**，使用$\langle i|h|j\rangle = \langle\chi_i|h|\chi_j\rangle = \int dx_1\chi_1^*(x_1)h(r_1)\chi_1(x_1)$简化表示，则有：

$$\langle\Psi_0|\mathbf{O_1}|\Psi_0\rangle = \langle 1|h|1\rangle + \langle 2|h|2\rangle \tag{11}$$

现在计算O_2矩阵元：

$$\begin{aligned}\langle\Psi_0|\mathbf{O_2}|\Psi_0\rangle = \int dx_1dx_2[2^{\frac{-1}{2}}(\chi_1(x_1)\chi_2(x_2) - \chi_2(x_1)\chi_1(x_2))]^* \\ \times r_{12}^{-1}[2^{\frac{-1}{2}}(\chi_1(x_1)\chi_2(x_2) - \chi_2(x_1)\chi_1(x_2))]\end{aligned} \tag{12}$$

$$\begin{aligned}= \frac{1}{2}\int dx_1dx_2[\chi_1^*(x_1)\chi_2^*(x_2)r_{12}^{-1}\chi_1(x_1)\chi_2(x_2) - \chi_1^*(x_1)\chi_2^*(x_2)r_{12}^{-1}\chi_2(x_1)\chi_1(x_2) \\ - \chi_2^*(x_1)\chi_1^*(x_2)r_{12}^{-1}\chi_1(x_1)\chi_2(x_2) + \chi_2^*(x_1)\chi_1^*(x_2)r_{12}^{-1}\chi_2(x_1)\chi_1(x_2)\end{aligned} \tag{13}$$

显然$r_{12}^{-1} = r_{21}^{-1}$，上式中的积分变量可以交换，即第一项和第四项相同，第二项与第三项相同，因此：

$$\begin{aligned}\langle\Psi_0|\mathbf{O_2}|\Psi_0\rangle = \int dx_1dx_2\chi_1^*(x_1)\chi_2^*(x_2)r_{12}^{-1}\chi_1(x_1)\chi_2(x_2) \\ - \int dx_1dx_2\chi_1^*(x_1)\chi_2^*(x_2)r_{12}^{-1}\chi_2(x_1)\chi_1(x_2)\end{aligned} \tag{14}$$

与单电子积分类似，我们同样以

$$\langle ij|kl\rangle = \langle\chi_i\chi_j|\chi_k\chi_l\rangle = \int dx_1dx_2\chi_i^*(x_1)\chi_j^*(x_2)r_{12}^{-1}\chi_k(x_1)\chi_l(x_2)$$

简化表示。那么就有:

$$\langle\Psi_0|\mathbf{O_2}|\Psi_0\rangle = \langle 12|12\rangle - \langle 12|21\rangle \tag{15}$$

所以，Hartree-Fock基态能量为：

$$\begin{aligned}E_0 = \langle\Psi_0|\mathbf{H}|\Psi_0\rangle &= \langle\Psi_0|\mathbf{O_1} + \mathbf{O_2}|\Psi_0\rangle \\ &= \langle 1|h|1\rangle + \langle 2|h|2\rangle + \langle 12|12\rangle - \langle 12|21\rangle\end{aligned} \tag{16}$$

有时也常用**反对称双电子积分**$\langle ij||kl\rangle ij|kl\rangle - \langle ij|lk\rangle$ 上述使用的积分符号也被称为**物理学家符号**，为方便，本文所有公式推导均是在物理学家符号下。双电子积分具有以下对称性：

$$\begin{aligned}\langle ij|kl\rangle &= \langle ji|lk\rangle \\ \langle ij|kl\rangle &= \langle kl|ij\rangle^*\end{aligned} \tag{17}$$

对于孤立体系、不考虑相对论效应的情况下，双电子积分的矩阵元均为实数，所以双电子积分具有**八重对称性**。
还有一种符号计法，被称为**化学家符号**，自旋轨道中使用[]表示，和物理学家符号的关系是：

$$\langle ij|kl\rangle = \langle \chi_i\chi_j|\chi_k\chi_l\rangle = [ik|jl] \tag{18}$$

对于空间轨道，化学家符号使用()表示。事实上，常用的量子化学电子积分库Libint2，Libcint都是在化学家符号下计算的。

2.2.3 空间轨道下的Hartree-Fock能量方程

使用自旋轨道可以简化公式推导，但实际计算中自旋函数α,β必须呗积分掉才能将其约化为可数值计算的空间轨道和积分。（实际上，对于闭壳层，自旋轨道比自旋轨道多浪费一个维度的资源）。

自旋轨道下Hartree-Fock基态能量为：

$$E_0 = \langle\chi_1|h|\chi_1\rangle + \langle\chi_2|h|\chi_2\rangle + \langle\chi_1\chi_2|\chi_1\chi_2\rangle - \langle\chi_1\chi_2|\chi_2\chi_1\rangle \tag{19}$$

我们知道，空间轨道和自旋轨道的关系为：

$$\begin{aligned}\chi_1(x) &= \psi_1(r)\alpha(w) = \psi_1\\ \chi_2(x) &= \psi_1(r)\beta(w) = \overline{\psi_1}\end{aligned} \tag{20}$$

联立上述方程，可以得到空间轨道下的Hartree-Fock能量：

$$E_0 = \langle\psi_1|h|\psi_1\rangle + \langle\overline{\psi_1}|h|\overline{\psi_1}\rangle + \langle\psi_1\overline{\psi_1}|\psi_1\overline{\psi_1}\rangle - \langle\psi_1\overline{\psi_1}|\overline{\psi_1}\psi_1\rangle \tag{21}$$

对于单电子积分：

$$\langle\overline{\psi_1}|h|\overline{\psi_1}\rangle = \int dr\psi_1^*(r_1)\beta^* h(r_1)\psi_1(r_1)\beta \tag{22}$$

对于非相对论下，单电子算符是不依赖自旋的，根据正交归一性，$\langle\beta|\beta\rangle = 1$，所以显然：

$$\langle\overline{\psi_1}|h|\overline{\psi_1}\rangle = \langle\psi_1|h|\psi_1\rangle \tag{23}$$

对于双电子积分的第一项：

$$\begin{aligned}\langle\psi_1\overline{\psi_1}|\psi_1\overline{\psi_1}\rangle = \int & dr_1dr_2dw_1dw_2\psi_1^*(r_1)\alpha^*(w_1)\psi_1(r_2)\alpha(w_2)r_{12}^{-1}\\ & \times\psi_1^*(r_1)\beta^*(w_1)\psi_1(r_2)\beta(w_2)\end{aligned} \tag{24}$$

同样把α,β积掉，得到：

$$\begin{aligned}\langle\psi_1\overline{\psi_1}|\psi_1\overline{\psi_1}\rangle &= \int dr_1dr_2\psi_1^*(r_1)\psi_1(r_2)r_{12}^{-1}\psi_1^*(r_1)\psi_1(r_2)\\ &= \langle\psi_1\psi_1|\psi_1\psi_1\rangle\end{aligned} \tag{25}$$

对于双电子积分的第二项：

$$\begin{aligned}\langle\psi_1\overline{\psi_1}|\overline{\psi_1}\psi_1\rangle = \int dr_1 dr_2 \psi_1^*(r_1)\alpha^*(w_1)\psi_1^*(r_2)\beta^*(w_2)r_{12}^{-1} \\ \times \psi_1(r_1)\beta(w_1)\psi_1(r_2)\alpha(w_2) = 0\end{aligned} \tag{26}$$

这是因为正交性$\langle\alpha|\beta\rangle = \langle\beta|\alpha\rangle = 0$
所以，空间轨道下的极小基H2的Hartree-Fock基态能量为：

$$\begin{aligned}E_0 &= 2\langle\psi_1|h|\psi_1\rangle + \langle\psi_1\psi_1|\psi_1\psi_1\rangle \\ &= 2\langle 1|h|1\rangle + \langle 11|11\rangle\end{aligned} \tag{27}$$

通常下，空间轨道的公式会比自旋轨道下的更紧凑，更好编程。

2.2.4 N电子体系闭壳层限制性Hartree-Fock能量方程

类似H2极小基的莫函数，N电子系统的Hartree-Fock波函数为：

$$\begin{aligned}|\Psi_0\rangle &= |\chi_1\chi_2\chi_3\chi_4\cdots\chi_{N-1}\chi_N\rangle \\ &= |\psi_1\overline{\psi_1}\psi_2\overline{\psi_2}\ldots\psi_{N/2}\overline{\psi_{N/2}}\rangle\end{aligned} \tag{28}$$

对于自旋轨道，Hartree-Fock能量方程为：

$$E_0 = \sum_i^N \langle i|h|i\rangle + \frac{1}{2}\sum_i^N\sum_j^N \langle ij|ij\rangle - \langle ij|ji\rangle \tag{29}$$

其中的双电子部分，共N/2对电子，故有$\frac{1}{2}$
自旋轨道波函数包括N/2个α自旋轨道和N/2个β自旋轨道，我们将自旋轨道的求和分为两部分：

$$\sum_i^N \chi_i = \sum_i^{N/2}\psi_i + \sum_i^{N/2}\overline{\psi_i} \tag{30}$$

对于双求和：

$$\begin{aligned}\sum_i^N\sum_j^N \chi_i\chi_j &= \sum_i^N \chi_i \sum_j^N \chi_j \\ &= \sum_i^{N/2}(\psi_i + \overline{\psi_i})\sum_j^{N/2}(\psi_j + \overline{\psi_j}) \\ &= \sum_i^{N/2}\sum_j^{N/2}\psi_i\psi_j + \psi_i\overline{\psi_j} + \overline{\psi_i}\psi_j + \overline{\psi_i\psi_j}\end{aligned} \tag{31}$$

我们将其转化为空间轨道方程。对于单电子积分：

$$\sum_i^N \langle i|h|i\rangle = \sum_i^{N/2}\langle i|h|i\rangle + \sum_i^{N/2}\langle \bar{i}|h|\bar{i}\rangle = 2\sum_i^{N/2}\langle \psi_i|h|\psi_i\rangle \tag{32}$$

双电子积分项：

$$\begin{aligned}
&\frac{1}{2}\sum_i^N\sum_j^N \langle ij|ij\rangle - \langle ij|ji\rangle \\
&\quad= \frac{1}{2}\sum_i^{N/2}\sum_j^{N/2}\langle ij|ij\rangle - \langle ij|ji\rangle + \frac{1}{2}\sum_i^{N/2}\sum_j^{N/2}\langle i\bar{j}|i\bar{j}\rangle - \langle i\bar{j}|\bar{j}i\rangle \\
&\quad+ \frac{1}{2}\sum_i^{N/2}\sum_j^{N/2}\langle \bar{i}j|\bar{i}j\rangle - \langle \bar{i}j|j\bar{i}\rangle + \frac{1}{2}\sum_i^{N/2}\sum_j^{N/2}\langle \bar{i}\bar{j}|\bar{i}\bar{j}\rangle - \langle \bar{i}\bar{j}|\bar{j}\bar{i}\rangle \\
&\quad= \sum_i^{N/2}\sum_j^{N/2} 2\langle \psi_i\psi_j|\psi_i\psi_j\rangle - \langle \psi_i\psi_j|\psi_j\psi_i\rangle
\end{aligned} \tag{33}$$

因此，空间轨道下的闭壳层限制性Hartree-Fock能量方程是：

$$E_0 = 2\sum_i^{N/2}\langle \psi_i|h|\psi_i\rangle + \sum_i^{N/2}\sum_j^{N/2} 2\langle \psi_i\psi_j|\psi_i\psi_j\rangle - \langle \psi_i\psi_j|\psi_j\psi_i\rangle \tag{34}$$

2.2.5 电子积分

这里只介绍HF方法中使用到的电子积分，并不展开其计算的解析式。

- 单电子积分
  - 重叠积分S：$S_{pq} = \langle \psi_p|\psi_q\rangle$
  - 动能积分T：$T_{pq} = \langle \psi_p| - \frac{1}{2}\nabla^2|\psi_q\rangle$
  - 核-电子势能积分V：$V_{pq} = \langle \psi_p|\frac{1}{r_C}|\psi_q\rangle$
  - 核-哈密顿矩阵H^{core}：$H^{core} = T_{pq} + V_{pq}$
- 双电子积分
  - I：$I_{pqrs} = \int d\mathbf{r_1}d\mathbf{r_2}\phi_p^*(\mathbf{r_1})\phi_q(\mathbf{r_1})\phi_r^*(\mathbf{r_2})\phi_s(\mathbf{r_2})$
  - 库伦积分J：$J_{ij} = \langle \psi_i\psi_j|\psi_i\psi_j\rangle$
  - 交换积分K：$K_{ij} = \langle \psi_i\psi_j|\psi_j\psi_i\rangle$

对于水分子在STO-3G下，共有10个电子、7个分子轨道，所以单电子积分为7*7的对称阵，双电子积分为7*7*7*7的四维矩阵，显然，双电子积分是自洽场计算中最耗时的部分，此处留个疑问：**存储四维矩阵显然浪费，且处理四维矩阵更耗时，计算程序采用那些策略优化呢？**

2.3 Roothan方程

Roothan方程一般是指限制性闭壳层HF方程，也称为Hartree-Fock-Roothan方程。以他为标题是因为下文的代码实现部分是如此。我们先推导一般性的HF方程。假设读者已经了解线性变分法（应该是结构化学的的知识）。

首先，给定一个线性变分尝试波函数：

$$|\Phi\rangle = \sum_{i=0}^{N} c_i |\Psi_i\rangle \tag{35}$$

其需要能量极小化的表达式为：

$$E = \langle\Phi|\mathbf{H}|\Phi\rangle = \sum_{ij} c_i^* c_j \langle\Psi_i|\mathbf{H}|\Psi_j\rangle \tag{36}$$

注意，尝试波函数需要满足归一化：

$$\langle\Phi\Phi\rangle - 1 = \sum_{ij} c_i^* c_j \langle\Phi\Phi\rangle - 1 = 0 \tag{37}$$

使用Lagrange不定乘子发，定义方程L和Lagrange乘子ϵ：

$$\begin{aligned} L &= \langle\Phi|\mathbf{H}|\Phi\rangle - \epsilon(\langle\Phi\Phi\rangle - 1) \\ &= \sum_{ij} c_i^* c_j \langle\Psi_i|\mathbf{H}|\Psi_j\rangle - \epsilon\left(\sum_{ij} c_i^* c_j \langle\Phi|\Phi\rangle - 1\right) \end{aligned} \tag{38}$$

进行线性变分：

$$\begin{aligned} \delta L &= \sum_{ij} \delta c_i^* c_j \langle\Psi_i|\mathbf{H}|\Psi_j\rangle - \epsilon \sum_{ij} \delta c_i^* c_j \langle\Phi|\Phi\rangle \\ &+ \sum_{ij} c_i^* \delta c_j \langle\Psi_i|\mathbf{H}|\Psi_j\rangle - \epsilon \sum_{ij} c_i^* \delta c_j \langle\Phi|\Phi\rangle \\ &= 2\sum_{i} \delta c_i^* \left(\sum_{j} \langle\Psi_i|\mathbf{H}|\Psi_j\rangle c_j - \epsilon\langle\Psi_i|\Psi_j\rangle c_j\right) = 0 \end{aligned} \tag{39}$$

我简单将复共轭合并，注意，索引i,j是等价的。
我们使用上文定义的$H_{ij} = \langle\Psi_i|\mathbf{H}|\Psi_j\rangle$和重叠积分$S_{ij} = \langle\Psi_i|\Psi_j\rangle$，HF方程可以写为：

$$\sum_j H_{ij}c_j = \epsilon\sum_j S_{ij}c_j \tag{40}$$

当然，一般写为：

$$\mathbf{Hc} = \epsilon\mathbf{Sc} \tag{41}$$

聪明的你显然注意到了，虽然我前文没有着墨声明：算符=矩阵，积分=张量，大家在符号中也能体会到了。
下面引入基函数，假设有K个已知的基函数（显然是我们上文提到的Gaussian形基函数），那么位置的分子轨道可以表示为基函数的线性展开：

$$\psi_i = \sum_\mu^K C_{\mu i}\phi_\mu \tag{42}$$

带入Fock算符，HF方程可写为

$$f\sum_v C_{vi}\phi_v = \epsilon_i\sum_v C_{vi}\phi_v \tag{43}$$

左乘波函数的复共轭得到：

$$\sum_v C_{vi}\langle\phi_v^*|f|\phi_v\rangle = \epsilon_i\sum_v C_{vi}\langle\phi_v^*|\phi_v\rangle \tag{44}$$

我们定义Fock矩阵$F_{\mu v} = \langle\phi_v^*|f|\phi_v\rangle$，还有上文提到的重叠积分。那么，Roothan方程可以表示为：

$$\sum_v F_{\mu v}C_{vi} = \epsilon_i\sum_v S_{\mu v}C_{vi} \tag{45}$$

或者写成矩阵形式：

$$\mathbf{FC} = \mathbf{SC}\epsilon \tag{46}$$

那么聪明的你，猜猜这些矩阵的形状是什么呢？
我们定义总电荷密度：

$$\rho(r) = 2\sum_a^{N/2}|\psi_a(r)|^2 \tag{47}$$

将波函数的线性展开带入其中得到：

$$\begin{aligned}\rho(r) &= 2\sum_{a}^{N/2}\psi_a^*(r)\psi_a(r) \\ &= 2\sum_{a}^{N/2}\sum_{v}C_{va}^*\phi_v^*(r)\sum_{\mu}C_{\mu a}\phi_\mu(r) \\ &= \sum_{\mu v}\left(2\sum_{a}^{N/2}C_{va}^*C_{\mu a}\right)\phi_v^*(r)\phi_\mu(r) \\ &= \sum_{\mu v}P_{\mu v}\phi_v^*(r)\phi_\mu(r)\end{aligned} \tag{48}$$

我们得到了一个重要的概念：密度矩阵$P_{\mu v}=2\sum_a^{N/2}C_{va}^*C_{\mu a}$可以证明Fock算符是波函数的本正算符，结合上节的推导可得到其表达式：

$$f = h + \sum_{a}^{N/2}2J_a - K_a \tag{49}$$

将波函数的线性展开带入Fock算符得到，并且写为矩阵形式（主要是好写）：

$$\begin{aligned}F_{\mu v} &= H_{\mu v}^{core} + \sum_{a}^{N/2}\sum_{ij}C_{ia}C_{ja}^*[2\langle ij|ij\rangle - \langle ij|ji\rangle] \\ &= H_{\mu v}^{core} + \sum_{ij}P_{ij}[\langle ij|ij\rangle - \frac{1}{2}\langle ij|ji\rangle]\end{aligned} \tag{50}$$

我们发现，Fock矩阵只与密度矩阵有关，也就是Fock矩阵依赖于展开系数。展开系数又可以通过求解Roothan方程得到，如此迭代直到能量最低，这就是所谓“自洽（self-consistent）”。

3 程序编写

3.1 环境准备

对于开发来说，尤其是计算化学程序，Linux绝对是最好的选择，相比Windows下需要考虑的更少，开发效率更高、更灵活。显然大多数电脑系统还是Windows,所以建议大家安装WSL2（安装教程），安装好gcc和gfortran，建议使用Anaconda简化python库文件的管理，我们后续代码需要Numpy。其次建议使用MacOS系统。本文所有代码均在WSL2中运行，Python版本为3.10

教程使用Python，当然使用什么语言无所谓，C/C++、Fortran都可以，他们的效率可能更高，但需要一定熟练度，否则不一定快于Numpy（因为其底层还是用C/Fortran写的），使用Python我认为对新手更Friendly。我会提供单、双电子积分，可以直接load，如果想跑其他结构，建议安装Pyscf。

对于Python语法，我不会用很复杂，因为只涉及矩阵处理，但也没精力再写一遍Numpy教程。这里假设读者水平为大学上过编程课程，会使用Chat-GPT等AI大模型问问题，基本上课余时间一周肯定能写出来!

3.2 程序流程

根据Szabo和Ostlund书中146页描述，自洽场迭代步骤为：

- 步骤
  1. 得到一个密度矩阵初猜
  2. 根据电子积分和初猜构建Fock矩阵
  3. 对角化Fock矩阵
  4. 选择占据轨道并计算新的密度矩阵
  5. 计算HF能量
  6. 计算误差判断是否收敛
     - 如果不收敛，使用新的密度矩阵继续
     - 如果收敛，输出能量

3.3 代码实现

我们的任务目标是：根据提供的单、双电子积分，计算H_2O分子在$STO-3G$基组下的HF电子能量，参考值:-84.1513215474753
载入环境

```python
import numpy as np
import scipy
```

将.npy二进制文件和python脚本文件放在同一文件夹中，载入单电子积分和双电子积分,同时获取轨道数nao。我们使用一个单位阵作为初猜，对应步骤1.值得注意，这是一种很粗糙的制作初猜的方式，不同的计算化学程序又不同的制作初猜的方法，初猜越准确，自洽场收敛越快。

```python
overlap_matrix = np.load("overlap.npy")
H = np.load("core_hamiltonian.npy")
int2e = np.load("int2e.npy")
nao =  len(overlap_matrix[0])
assert nao ==  len(int2e[0])
dm = np.eye(nao)
```

下面我们需要些两个函数，用于构建库伦积分**J**和交换积分**K**，传入函数的是密度矩阵dm，对应步骤2.因为双电子积分是四维矩阵，所以应该是四重循环。根据上文的公式，遍历壳层p, q, r, s

```python
# Calculate the coulomb matrices from density matrix
def get_j(dm):
    J = np.zeros((nao, nao))  # Initialize the Coulomb matrix

    # Loop over all indices of the Coulomb matrix
    for p in  range(nao):
        for q in  range(nao):
            # Calculate the Coulomb integral for indices (p,q)
            for r in  range(nao):
                for s in  range(nao):
                    J[p, q] += dm[r, s] * int2e[p, q, r, s]

    return J

# Calculate the exchange  matrices from density matrix
def get_k(dm):
    K = np.zeros((nao, nao))  # Initialize the K matrix

    # Loop over all indices of the K matrix
    for p in  range(nao):
        for q in  range(nao):
            # Calculate the K integral for indices (p,q)
            for r in  range(nao):
                for s in  range(nao):
                    K[p, q] += dm[r, s] * int2e[p, r, q, s]

    return K
```

我们还需要一个函数用于构建新的密度矩阵dm，将Fock矩阵传入函数。根据密度矩阵的公式，我们要遍历所有占据轨道的系数矩阵。对于本体系共10个电子，也就是5个占据轨道和2个空轨道。在这个函数中，我们要先完成步骤3，将Fock矩阵对角化，得到系数矩阵**C**。我们需要将所有原子轨

道基函数正交化，一个可以的方法是对称正交化。例如定义一个矩阵$\mathbf{A}$，满足：

$$\mathbf{A}^{\dagger}\mathbf{S}\mathbf{A} = 1$$

令$\mathbf{A} = \mathbf{S}^{-1/2}$，于是有：

$$\mathbf{A}^{\dagger}\mathbf{S}\mathbf{A} = \mathbf{S}^{1/2}\mathbf{S}\mathbf{S}^{-1/2} = \mathbf{S}^{-1/2}\mathbf{S}^{1/2} = \mathbf{S}^{0} = 1$$

我们借助矩阵$\mathbf{A}$将Roothan方程转换为本征方程（canonical eigenvalue equation），令$\mathbf{C} = \mathbf{A}\mathbf{C}'$

$$\mathbf{F}\mathbf{A}\mathbf{C}' = \mathbf{S}\mathbf{A}\mathbf{C}'\epsilon$$

$$\mathbf{A}^{\dagger}(\mathbf{F}\mathbf{A}\mathbf{C}') = \mathbf{A}^{\dagger}(\mathbf{S}\mathbf{A}\mathbf{C}')\epsilon$$

$$(\mathbf{A}^{\dagger}\mathbf{F}\mathbf{A})\mathbf{C}' = (\mathbf{A}^{\dagger}(\mathbf{S}\mathbf{A})\mathbf{C}'\epsilon$$

$$\mathbf{F}'\mathbf{C}' = \mathbf{C}'\epsilon$$

于是我们可以通过将$\mathbf{F}'$矩阵对角化得到$\mathbf{C}'$，再将其转换回$\mathbf{C}$通过$\mathbf{C} = \mathbf{A}\mathbf{C}'$可以通过Scipy库实现，我们使用了scipy.linalg.fractional_matrix_power()函数和np.linalg.eigh()函数

```
# Calculate the density matrix
def get_dm(fock, nocc):
    dm = np.zeros((nao, nao))
    S = overlap_matrix
    A = scipy.linalg.fractional_matrix_power(S, -0.5)
    F_p = A.T @ fock @ A
    eigs, coeffsm = np.linalg.eigh(F_p)

    c_occ = A @ coeffsm
    c_occ = c_occ[:, :nocc]
    for i in  range(nocc):
        for p in  range(nao):
            for q in  range(nao):
                dm[p, q] += c_occ[p, i] * c_occ[q, i]
    return dm
```

OK!准备工作已经充足，可以开始自洽场迭代了！对应步骤5、6
我们首先要确定收敛限和最大迭代次数。收敛限即最后收敛能量与上一次循环的HF能量变化小于一个值，我们设置为$1.0e-10$，最大迭代次数为40次同时将能量初始化

```
# Maximum SCF iterations
max_iter = 100
E_conv = 1.0e-10
# SCF & Previous Energy
SCF_E = 0.0
E_old = 0.0
```

根据步骤6书写循环

```
for scf_iter in  range(1, max_iter + 1):
    # GET Fock martix
    F = H + 2 * get_j(dm) - get_k(dm)
    assert F.shape == (nao, nao)

    SCF_E = np. sum(np.multiply((H + F), dm))
    dE = SCF_E - E_old
    print('SCF Iteration %3d: Energy = %4.16f dE = % 1.5E' % (scf_iter, SCF_E, dE)
        )

    if ( abs(dE) < E_conv):
        print("SCF convergence! Congrats")
        break
    E_old = SCF_E

    dm = get_dm(F, 5)

assert(np. abs(SCF_E + 84.1513215474753) < 1.0e-10)
```

4 后记

完整的代码我上传在gitlab上，从上面可以下载二进制积分文件,npy和源代码。本问的代码非常简单，只有几十行，而且有非常多可以改良的地方，比如**get_j**函数中使用了四重循环，这在python中是灾难性的代码；而且也并未考虑积分对称性。我列出几个可以继续思考的地方：

1. 结合参考书，实现UHF代码
2. 使用np.einsum代替循环和一些内置函数以提高效率
3. 考虑双电子积分的八重对称性加速构建Fock矩阵
4. 使用DIIS技术加速SCF收敛
5. 使用Cython，C/C++，Fortran等静态编程语言重写代码以加速
6. 安装并学习Pyscf，尝试计算更多电子结构
7. ...

能自己写一个计算化学代码并且和自己常用的计算化学软件对应上，其实是一个很有成就感的过程。Keep coding！Keep thinking!

参考文献

[1] Szabo and Ostlund. *Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*[M]. New York:Dover Publications,1996.

A 完整代码（简易版）

```python
import numpy as np
import scipy

overlap_matrix = np.load("overlap.npy")
H = np.load("core_hamiltonian.npy")
int2e = np.load("int2e.npy")
nao =  len(overlap_matrix[0])
assert nao ==  len(int2e[0])
assert int2e.shape == (nao, nao, nao, nao)
dm = np.eye(nao)

# Calculate the coulomb matrices from density matrix
def get_j(dm):
    J = np.zeros((nao, nao))  # Initialize the Coulomb matrix

    # Loop over all indices of the Coulomb matrix
    for p in  range(nao):
        for q in  range(nao):
            # Calculate the Coulomb integral for indices (p,q)
            for r in  range(nao):
                for s in  range(nao):
                    J[p, q] += dm[r, s] * int2e[p, q, r, s]

    return J

# Calculate the exchange  matrices from density matrix
def get_k(dm):
    K = np.zeros((nao, nao))  # Initialize the K matrix

    # Loop over all indices of the K matrix
    for p in  range(nao):
        for q in  range(nao):
            # Calculate the K integral for indices (p,q)
            for r in  range(nao):
                for s in  range(nao):
                    K[p, q] += dm[r, s] * int2e[p, r, q, s]

    return K

# Calculate the density matrix
def get_dm(fock, nocc):
    dm = np.zeros((nao, nao))
    S = overlap_matrix
    A = scipy.linalg.fractional_matrix_power(S, -0.5)
    F_p = A.T @ fock @ A
    eigs, coeffsm = np.linalg.eigh(F_p)

    c_occ = A @ coeffsm
    c_occ = c_occ[:, :nocc]
    for i in  range(nocc):
        for p in  range(nao):
```

```
52              for q in  range(nao):
53                  dm[p, q] += c_occ[p, i] * c_occ[q, i]
54      return dm
55  # Maximum SCF iterations
56  max_iter = 100
57  E_conv = 1.0e-10
58  # SCF & Previous Energy
59  SCF_E = 0.0
60  E_old = 0.0
61  for scf_iter in  range(1, max_iter + 1):
62      # GET Fock martix
63      F = H + 2 * get_j(dm) - get_k(dm)
64      assert F.shape == (nao, nao)
65
66      SCF_E = np. sum(np.multiply((H + F), dm))
67      dE = SCF_E - E_old
68      print('SCF Iteration %3d: Energy = %4.16f dE = % 1.5E' % (scf_iter, SCF_E, dE)
            )
69
70      if ( abs(dE) < E_conv):
71          print("SCF convergence! Congrats")
72          break
73      E_old = SCF_E
74
75      dm = get_dm(F, 5)
76
77  assert(np. abs(SCF_E + 84.1513215474753) < 1.0e-10)
```

B 完整代码（Libcint版）

```python
import numpy as np
import scipy
import ctypes
from pyscf import gto
import time

# slots of atm
CHARGE_OF       = 0
PTR_COORD       = 1
NUC_MOD_OF      = 2
PTR_ZETA        = 3
PTR_FRAC_CHARGE = 3
RESERVE_ATMLOT1 = 4
RESERVE_ATMLOT2 = 5
ATM_SLOTS       = 6


# slots of bas
ATOM_OF         = 0
ANG_OF          = 1
NPRIM_OF        = 2
NCTR_OF         = 3
KAPPA_OF        = 4
PTR_EXP         = 5
PTR_COEFF       = 6
RESERVE_BASLOT  = 7
BAS_SLOTS       = 8

# Create a molecular object for H2O molecule
mol = gto.M(atom='O 0 0 0; H 0 -0.757 0.587; H 0 0.757 0.587', basis='6-31g')

# necessary parameters for Libcint
atm = mol._atm.astype(np.intc)
bas = mol._bas.astype(np.intc)
env = mol._env.astype(np.double)
nao = mol.nao_nr().astype(np.intc)
nshls =  len(bas)
natm =  len(atm)

_cint = ctypes.cdll.LoadLibrary('/home/yx/cint_and_xc/lib/libcint.so')

def get_ovlp_matrix():

    ovlp_matrix = np.zeros((nao, nao), order='F')

    _cint.cint1e_ovlp_sph.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.double, ndim=2),
    (ctypes.c_int * 2),
    np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
    ctypes.c_int,
    np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
```

```
    ctypes.c_int,
    np.ctypeslib.ndpointer(dtype=np.double, ndim=1)
]


    _cint.CINTcgto_spheric.restype = ctypes.c_int
    _cint.CINTcgto_spheric.argtypes = [ctypes.c_int, np.ctypeslib.ndpointer(dtype=
        np.intc, ndim=2)]

    for ipr in  range(nshls):
        di = _cint.CINTcgto_spheric(ipr, bas)
        x = 0
        for i in  range(ipr):
            x += _cint.CINTcgto_spheric(i, bas)

        for jpr in  range(nshls):
            dj = _cint.CINTcgto_spheric(jpr, bas)
            y = 0
            for j in  range(jpr):
                y += _cint.CINTcgto_spheric(j, bas)

            buf = np.empty((di, dj), order='F')
            _cint.cint1e_ovlp_sph(buf, (ctypes.c_int * 2)(ipr, jpr), atm, natm,
                bas, nshls, env)

            # Update the overlap matrix with the values from buf
            ovlp_matrix[x: x + di, y : y + dj] = buf

    return ovlp_matrix
def get_core_hamiltonian():

    core_h = np.zeros((nao, nao), order='F')

    _cint.cint1e_nuc_sph.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.double, ndim=2),
    (ctypes.c_int * 2),
    np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
    ctypes.c_int,
    np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
    ctypes.c_int,
    np.ctypeslib.ndpointer(dtype=np.double, ndim=1)
]
    _cint.cint1e_kin_sph.argtypes = [
    np.ctypeslib.ndpointer(dtype=np.double, ndim=2),
    (ctypes.c_int * 2),
    np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
    ctypes.c_int,
    np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
    ctypes.c_int,
    np.ctypeslib.ndpointer(dtype=np.double, ndim=1)
]


    _cint.CINTcgto_spheric.restype = ctypes.c_int
```

```
104     _cint.CINTcgto_spheric.argtypes = [ctypes.c_int, np.ctypeslib.ndpointer(dtype=
            np.intc, ndim=2)]
105
106     for ipr in  range(nshls):
107         di = _cint.CINTcgto_spheric(ipr, bas)
108         x = 0
109         for i in  range(ipr):
110             x += _cint.CINTcgto_spheric(i, bas)
111
112         for jpr in  range(nshls):
113             dj = _cint.CINTcgto_spheric(jpr, bas)
114             y = 0
115             for j in  range(jpr):
116                 y += _cint.CINTcgto_spheric(j, bas)
117
118             buf1 = np.empty((di, dj), order='F')
119             buf2 = np.empty((di, dj), order='F')
120
121             _cint.cint1e_nuc_sph(buf1, (ctypes.c_int * 2)(ipr, jpr), atm, natm,
                    bas, nshls, env)
122             _cint.cint1e_kin_sph(buf2, (ctypes.c_int * 2)(ipr, jpr), atm, natm,
                    bas, nshls, env)
123
124             # Update the overlap matrix with the values from buf
125             core_h[x: x + di, y : y + dj] += buf1
126             core_h[x: x + di, y : y + dj] += buf2
127
128     return core_h
129 def get_int2e():
130     int2e = np.zeros((nao, nao, nao, nao), order='F')
131
132     _cint.cint2e_sph.argtypes = [
133     np.ctypeslib.ndpointer(dtype=np.double, ndim=4),
134     (ctypes.c_int * 4),
135     np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
136     ctypes.c_int,
137     np.ctypeslib.ndpointer(dtype=np.intc, ndim=2),
138     ctypes.c_int,
139     np.ctypeslib.ndpointer(dtype=np.double, ndim=1),
140     ctypes.POINTER(ctypes.c_void_p)
141 ]
142     _cint.CINTcgto_spheric.restype = ctypes.c_int
143     _cint.CINTcgto_spheric.argtypes = [ctypes.c_int, np.ctypeslib.ndpointer(dtype=
            np.intc, ndim=2)]
144
145     for ipr in  range(nshls):
146         di = _cint.CINTcgto_spheric(ipr, bas)
147         x = 0
148         for i in  range(ipr):
149             x += _cint.CINTcgto_spheric(i, bas)
150
151         for jpr in  range(nshls):
152             dj = _cint.CINTcgto_spheric(jpr, bas)
153             y = 0
```

```
154             for j in  range(jpr):
155                 y += _cint.CINTcgto_spheric(j, bas)
156 
157             for kpr in  range(nshls):
158                 dk = _cint.CINTcgto_spheric(kpr, bas)
159                 z = 0
160                 for k in  range(kpr):
161                     z += _cint.CINTcgto_spheric(k, bas)
162 
163                 for lpr in  range(nshls):
164                     dl = _cint.CINTcgto_spheric(lpr, bas)
165                     w = 0
166                     for l in  range(lpr):
167                         w += _cint.CINTcgto_spheric(l, bas)
168 
169                     buf = np.empty((di, dj, dk, dl), order='F')
170                     _cint.cint2e_sph(buf, (ctypes.c_int * 4)(ipr, jpr, kpr, lpr),
                            atm, natm, bas, nshls, env, ctypes.POINTER(ctypes.c_void_p
                            )())
171 
172                     # Update the overlap matrix with the values from buf
173                     int2e[x: x + di, y : y + dj, z : z + dk, w : w + dl] = buf
174         int2e.reshape([nao, nao, nao, nao])
175 
176     return int2e
177 def make_j(D):
178     return np.einsum('pqrs,rs->pq', I, D, optimize=True)
179 def make_k(D):
180     return np.einsum('prqs,rs->pq', I, D, optimize=True)
181 def make_d(fock, norb):
182     eigs, coeffs = scipy.linalg.eigh(fock, S)
183     c_occ = coeffs[:, :norb]
184     return np.einsum('pi,qi->pq', c_occ, c_occ, optimize=True)
185 
186 if __name__ == '__main__':
187     start = time.time()
188     S = get_ovlp_matrix()
189     H = get_core_hamiltonian()
190     I = get_int2e()
191     # SCF & Previous Energy
192     SCF_E = 0.0
193     E_old = 0.0
194     # start DM
195     D = make_d(H, 5)
196     # ==> RHF-SCF Iterations <==
197     for scf_iter in  range(1, 100 + 1):
198 
199         # GET Fock martix
200         F = H + 2 * make_j(D) - make_k(D)
201         '''error vector = FDS - SDF '''
202         diis_r = F.dot(D).dot(S) - S.dot(D).dot(F)
203         SCF_E = np.einsum('pq,pq->', (H + F), D, optimize=True)
204         dE = SCF_E - E_old
205         dRMS = 0.5 * np.mean(diis_r ** 2) ** 0.5
```

```
206         print('SCF Iteration %3d: Energy = %4.16f dE = % 1.5E dRMS = %1.5E' % (
                scf_iter, SCF_E, dE, dRMS))
207
208         if ( abs(dE) < 1e-10) and (dRMS < 1e-10):
209             end = time.time()
210             print("SCF convergence! Congrats")
211             print("Time used: ", end - start)
212             break
213         E_old = SCF_E
214         D = make_d(F, 5)
```