一个简单的Hartree-Fock代码基于Pyscf

Sun Xinyu
sunxinyu347@gmail.com

2023 年 10 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

在这里我假设大家已经系统上过“线性代数”这门课，阅读过了Szabo的《Modern Quantum Chemistry》，至少也是阅读了Levine的《Quantum Chemistry》，能看懂我后续的公式。如果看不懂，就需要再仔细看书了。

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 电子积分

这里只介绍HF方法中使用到的电子积分，并不展开其计算公式。

- 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：H=
- 双电子积分I：$I_{pqrs} = \int\int d\mathbf{r_1} d\mathbf{r_2} \phi_p^*(\mathbf{r_1})\phi_q(\mathbf{r_1})\phi_r^*(\mathbf{r_2})\phi_s(\mathbf{r_2})$

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

HF方法，又称为自洽场方法，目标为解一个伪特征值矩阵方程（pseudo-eigenvalue matrix equation）：

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

通过迭代的方法求解稀疏矩阵**C**得到能量本征值ϵ_i，其中**S**为重叠积分，**F**为Fock矩阵，其通过电子积分得到：

$$F_{pq} = \mathbf{H} + 2(pq|rs)D_{rs} - (pr|qs)D_{rs}$$

其中$2(pq|rs)D_{rs}$被称为库伦积分（**J**）,$-(pq|rs)D_{rs}$被称为交换积分（**K**），矩阵**D**为密度矩阵（density matrix）：

$$D_{pq} = \sum_i C_{pi}C_{qi}$$

Hartree-Fock能量为：

$$E_{elec} = (F_{pq} + H_{pq})\, D_{pq}$$

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

```python
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

```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)
```