Hartree-Fock Self Consistent Field Procedure

24 Apr 2013

Quick links to the program and files you'll need

SCF program: `scf.py`

Overlap integrals: `s.dat` ,

Kinetic Energy integrals: `t.dat` ,

Potential Energy integrals: `v.dat` ,

Nuclear Repulsion: `enuc.dat` ,

Electron Repulsion integrals: `eri.dat`

I do quite a bit of development work as a graduate student, and one of the most helpful parts of my education was writing a Hartree-Fock SCF (self-consistent field) program from scratch. The SCF procedure is the workhorse of most computational chemistry software packages, as it is generally the first step before doing more advanced calculations (such as MP2, etc). It is also the conceptual basis for molecular orbital theory. I've shared my code below, in case you want to try it out for yourself. I wrote it in Python 2.7 (it won't work in Python 3), utilizing the NumPy package. I find it to be very readable, which makes testing out ideas fast and (relatively) intuitive. The program is written to give an idea how SCF calculations work, so it isn't that efficient – but I only use it for two electron systems anyway so who cares!

```
#!/usr/bin/python

#####################################
#
# SELF CONSISTENT FIELD METHOD
#
```

```
###################################

from __future__ import division
import sys
import math
import numpy as np
from numpy import genfromtxt
import csv

###################################
#
# FUNCTIONS
#
###################################

# Symmetrize a matrix given a triangular one
def symmetrize(a):
    return a + a.T - np.diag(a.diagonal())

# Return compund index given four indices
def eint(a,b,c,d):
    if a > b: ab = a*(a+1)/2 + b
    else: ab = b*(b+1)/2 + a
    if c > d: cd = c*(c+1)/2 + d
    else: cd = d*(d+1)/2 + c
    if ab > cd: abcd = ab*(ab+1)/2 + cd
    else: abcd = cd*(cd+1)/2 + ab
    return abcd

# Return Value of two electron integral
# Example: (12\vert 34) = tei(1,2,3,4)
# I use chemists notation for the SCF procedure.
def tei(a,b,c,d):
    return twoe.get(eint(a,b,c,d),0.0)

# Put Fock matrix in Orthonormal AO basis
def fprime(X,F):
    return np.dot(np.transpose(X),np.dot(F,X))

# Diagonalize a matrix. Return Eigenvalues
# and non orthogonal Eigenvectors in separate 2D arra
def diagonalize(M):
    e,Cprime = np.linalg.eigh(M)
    #e=np.diag(e)
    C = np.dot(S_minhalf,Cprime)
    return e,C

# Make Density Matrix
# and store old one to test for convergence
def makedensity(C,P,dim,Nelec):
    OLDP = np.zeros((dim,dim))
    for mu in range(0,dim):
        for nu in range(0,dim):
            OLDP[mu,nu] = P[mu,nu]
            P[mu,nu] = 0.0e0
            for m in range(0,Nelec//2):
                P[mu,nu] = P[mu,nu] + 2*C[mu,m]*C[nu,
    return P, OLDP
# Make Fock Matrix
def makefock(Hcore,P,dim):
    F = np.zeros((dim,dim))
```

```
    for i in range(0,dim):
        for j in range(0,dim):
            F[i,j] = Hcore[i,j]
                for k in range(0,dim):
                    for l in range(0,dim):
                        F[i,j] = F[i,j] + P[k,l]*(tei
    return F

# Calculate change in density matrix
def deltap(P,OLDP):
    DELTA = 0.0e0
    for i in range(0,dim):
        for j in range(0,dim):
            DELTA = DELTA+((P[i,j]-OLDP[i,j])**2)
    DELTA = (DELTA/4)**(0.5)
    return DELTA

# Calculate energy at iteration
def currentenergy(P,Hcore,F,dim):
    EN = 0.0e0
    for mu in range(0,dim):
        for nu in range(0,dim):
            EN = EN + 0.5*P[mu,nu]*(Hcore[mu,nu] + F[
    return EN

###################################
#
# FORM CORE HAMILTONIAN
#
###################################
Nelec = 2 # The number of electrons in our system

# ENUC = nuclear repulsion, Sraw is overlap matrix, T
# Vraw is potential energy matrix

ENUC = genfromtxt('./enuc.dat',dtype=float,delimiter=
Sraw = genfromtxt('./s.dat',dtype=None)
Traw = genfromtxt('./t.dat',dtype=None)
Vraw = genfromtxt('./v.dat',dtype=None)

# dim is the number of basis functions
dim = int((np.sqrt(8*len(Sraw)+1)-1)/2)

# Initialize integrals, and put them in convenient Nu
S = np.zeros((dim,dim))
T = np.zeros((dim,dim))
V = np.zeros((dim,dim))

for i in Sraw: S[i[0]-1,i[1]-1] = i[2]
for i in Traw: T[i[0]-1,i[1]-1] = i[2]
for i in Vraw: V[i[0]-1,i[1]-1] = i[2]

# The matrices are stored triangularly. For convenier
# the whole matrix. The function is defined above.
S = symmetrize(S)
V = symmetrize(V)
T = symmetrize(T)

Hcore = T + V
#print Hcore
```

```
####################################
#
# TWO ELECTRON INTEGRALS
#
####################################

# Like the core hamiltonian, we need to grab the inte
# separate file, and put into ERIraw (ERI = electron
# I chose to store the two electron integrals in a py
# The function 'eint' generates a unique compund inde
# electron integral, and maps this index to the corre
# 'twoe' is the name of the dictionary containing the

ERIraw = genfromtxt('./eri.dat',dtype=None)
twoe = {eint(row[0],row[1],row[2],row[3]) : row[4] fo

####################################
#
# SCF PROCEDURE
#
####################################

# Step 1: orthogonalize the basis (I used symmetric o
# which uses the S^(-1/2) as the transformation matri
# p 143 for more details.

SVAL, SVEC = np.linalg.eig(S)
SVAL_minhalf = (np.diag(SVAL**(-0.5)))
S_minhalf = np.dot(SVEC,np.dot(SVAL_minhalf,np.transp
# This is the main loop. See Szabo and Ostlund p 146.
# Try uncommenting the print lines to see step by ste
# output. Look to see the functions defined above for
# understanding of the process.

P = np.zeros((dim,dim)) # P is density matrix, set ir
DELTA = 1.0
convergence = 0.00000001
G = np.zeros((dim,dim)) # The G matrix is used to mak
while DELTA > convergence:
    F = makefock(Hcore,P,dim)
   # print "F = \n", F
    Fprime = fprime(S_minhalf,F)
   # print "Fprime = \n", Fprime

    E,Cprime = np.linalg.eigh(Fprime)
    C = np.dot(S_minhalf,Cprime)
    E,C = diagonalize(Fprime)
   # print "C = \n", C

    P,OLDP = makedensity(C,P,dim,Nelec)

   # print "P = \n", P

# test for convergence. if meets criteria, exit loop

    DELTA = deltap(P,OLDP)
   #print "E= ",currentenergy(P,Hcore,F,dim)+ENUC
   #print "Delta = ", DELTA,"\n"

    EN = currentenergy(P,Hcore,F,dim)
```

```
print "TOTAL E(SCF) = \n", EN + ENUC
#print "C = \n", C
```

To use it, download the `scf.py` file to your computer, and put it in a folder along with `s.dat`, `t.dat`, `v.dat`, and `enuc.dat`. These are the integral values it needs to run: `s.dat` is the overlap matrix values, `t.dat` is the kinetic energy matrix, `v.dat` is the potential energy matrix, and `enuc.dat` contains the nuclear repulsion energy. You'll also need the `eri.dat`, which contains the two electron integrals. All values were taken from `Gaussian09` for HeH+ at a bond length of 0.9295 Angstrom with an STO-3G basis set. Once you have those downloaded, fire up your terminal and give it a try! See the example of my terminal below for usage. Just navigate to the folder you have everything in, and type `python scf.py` then hit Enter!

```
meow:heh+_xsli jjgoings$ ls
enuc.dat eri.dat  s.dat    scf.py   t.dat    v.dat
meow:heh+_xsli jjgoings$ python scf.py
TOTAL E(SCF) =
-2.62613304202
meow:heh+_xsli jjgoings$
```

Related Posts

The Magnus Expansion 15 Jun 2017

A (hopefully) gentle guide to the computer implementation of molecular integrals over Gaussian basis functions. 28 Apr 2017

Quantum Dynamics Movies 20 Apr 2016