ELECTRONIC STRUCTURE - PRACTICAL SESSION 3

Self Consistent Field Calculations:
Restricted Hartree-Fock method for two-electron diatomic molecules (Part I)

Gisela Martí Guerrero

January 2023

1 Problem 1

Let us consider first a H_2 molecule with an internuclear distance R = 1.4 bohr. In our basis set we will include two Slater type 1s atomic orbitals, each centered on one of the two atoms in the molecule with an exponent $\zeta = 1.24$. Although at the final stage we will write a complete program, including the calculation of all necessary integrals, let us write first a program performing the whole SCF procedure taking the values of the molecular integrals as input parameters.

For the H2 molecule with R = 1.4 bohr and two 1s AOs with $\zeta = 1.24$ we find (using a STO-3G representation of the 1s STO):

$$\begin{array}{ll} S_{12} = 0.6593 & (11|11) = 0.7746 \\ t_{11} = 0.7600 & (11|22) = 0.5697 \\ t_{12} = 0.2365 & (21|11) = 0.4441 \\ V^1_{11} = -1.2266 & (21|21) = 0.2970 \\ V^1_{12} = -0.5974 & \\ V^1_{22} = -0.6538 & \end{array}$$

where all values are give in atomic units.

Using these values, write a program to perform an RHF calculation for the H_2 molecule. Your output should contain the set of coefficients for the two MOs, their energies, the total energy for the ground state and the number of iterations needed to obtain a converged solution. Calculate also the nucleus-nucleus repulsion energy and add it to the electronic energy to get the system's total energy.

To start the SCF process build the trial density matrix using the expression for the bonding orbital for the H_2^+ molecular ion:

$$1\sigma_g = \frac{1}{\sqrt{2(1+S_{12})}}\,(1s_1 + 1s_2) \tag{1}$$

considering it to be doubly occupied by the two electrons in the H_2 system.

Use the following unitary matrix to build the transformation matrix $\mathbf{X}$:

$$\mathbf{U} = \begin{pmatrix} 2^{-1/2} & 2^{-1/2} \\ 2^{-1/2} & -2^{-1/2} \end{pmatrix} \tag{2}$$

If your program is working properly, you should obtain $\epsilon_1 = -0.5782 a.u.$ and $\epsilon_2 = +0.6703 a.u.$ and a total electronic energy of $-1.8310 a.u.$ If you add the nuclear repulsion term the total energy should be $U_0(R = 1.4\,a.u.) = -1.1167\,a.u.$ Check that your program converges to the same solution in just one iteration irrespective of the density matrix that you choose. Try for instance a null density matrix. This is equivalent to completely neglecting the bielectronic term in the first guess of the Fock matrix so that $\mathbf{F^{(0)}} = \mathbf{h}$.

```
import numpy as np
import sys

# Step 1: specifying molecular geometry, basis set and other variables
Nelec = 2
R = 1.4 #bohr
dim = 2 #dim is the number of basis functions
```

```
# Molecular integrals are read from input files
S_int = np.genfromtxt('./s.dat',dtype=None) # overlap matrix
T_int = np.genfromtxt('./t.dat',dtype=None) # kinetic energy matrix
V_int = np.genfromtxt('./v.dat',dtype=None) # potential energy matrix
te_int = np.genfromtxt('./elec.dat') # two electron integrals

# Step 2: Calculate all the required molecular integrals S,T,V and 2-electron
S = np.zeros((dim, dim))
T = np.zeros((dim, dim))
V = np.zeros((dim, dim))

# Put the integrals into a matrix
for i in S_int:
    S[i[0]-1, i[1]-1] = i[2]
for i in T_int:
    T[i[0]-1, i[1]-1] = i[2]
for i in V_int:
    V[i[0]-1, i[1]-1] = i[2]

# Step 3: Diagonalize S and use its eigenvalues to obtain a transformation matrix X

def symmetrise(matrix):
    """
    Function to symmetrize a matrix given a triangular one
    """
    return matrix + matrix.T - np.diag(matrix.diagonal())

# Flip the triangular matrix in the diagonal
S = symmetrise(S)
V = symmetrise(V)
T = symmetrise(T)
# Form core Hamiltonian matrix as sum of the T and V matrices
Hcore = T + V

# Diagonalize overlap matrix. S_val are the eigenvalues and S_vec the eigenvectors
S_val, S_vec = np.linalg.eigh(S)
# Find inverse square root of eigenvalues
s_half = (np.diag(S_val**(-0.5)))
# Form the transformation matrix X. The unitary matrix are the eigenvectors
X_matrix = -np.dot(S_vec, np.dot(s_half, np.transpose(S_vec)))

# Step 4: Construct a guess denisty matrix P(0)
P = np.zeros((dim, dim))

# Step 5: Calculate the bielectronic term G(0) using the guess denisty matrix and the
two electron integrals
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
```

```
# two-electron integrals are stored in a dictionary
twoe = {eint(row[0], row[1], row[2], row[3]) : row[4] for row in te_int}

def two_elec_int(a, b, c, d): # Return value of two electron integral
    """
    Return value of two electron integral
    """
    return twoe.get(eint(a, b, c, d), 0)

# Step 6: Add G(0) to the one-electron term h to get a first guess of the Fock matrix
F(0) = h + G(0).
def fock_matrix(Hcore, P, dim):
    """
    Function to build the Fock Matrix
    """
    F = np.zeros((dim, dim)) # zero array
    for i in range(0, dim):
        for j in range(0, dim):
            F[i,j] = Hcore[i,j] # initial Fock matrix
            for k in range(0, dim):
                for l in range(0, dim):
                    # Form the Fock matrix using the product of the density matrix and
                      G matrix
                    F[i,j] = F[i,j] + P[k,l]*(two_elec_int(i+1,j+1,k+1,l+1)-
                             0.5*two_elec_int(i+1,k+1,j+1,l+1))
    return F

# Step 7: Transform the Fock matrix
def f_transform(X, F):
    """
    Transform Fock matrix with the transformation matrix X
    """
    return np.dot(np.transpose(X), np.dot(F, X))

# Step 10: Form a new density matrix P(1) using C(1)
def density_matrix(C, D, dim, Nelec): # Make density matrix and store old one to test for
convergence
    """
    Make new density matrix and store old one to test for convergence

    Returns:
        D: new density matrix
        Dold: old denisty matrix
    """
    Dold = np.zeros((dim, dim)) # Initiate zero array
    for mu in range(0, dim):
        for nu in range(0, dim):
            Dold[mu,nu] = D[mu, nu] # Set old density matrix to the density matrix, D,
            input into the function
            D[mu,nu] = 0
            for m in range(0, int(Nelec/2)):
                # Form new density matrix
                D[mu,nu] = D[mu,nu] + 2*C[mu,m]*C[nu,m]
    return D, Dold

# Step 11: Determine if the process has converged by comparing P(1) with P(0).
def threshold(D, Dold):
    """
    Calculate change in density matrix using Root Mean Square Deviation (RMSD)
    """
```

```
    DELTA = 0.0
    for i in range(0, dim):
        for j in range(0, dim):
            DELTA = DELTA + ((D[i,j] - Dold[i,j])**2)

    return (DELTA/4.0)**(0.5)

# Step 12: If the process has converged, use the resultant solution, represented by C(k),
  P(k), and F(k)
def energy_iteration(D, Hcore, F, dim):
    """
    Function that calculates the energy at each iteration
    """
    EN = 0
    for mu in range(0, dim):
        for nu in range(0, dim):
            EN += 0.5*D[mu,nu]*(Hcore[mu,nu] + F[mu,nu])
    return EN

# Finally we make the iteration loop
DELTA = 1
count = 0 # cycles counter
nuclear_repulsion = 1/R
while DELTA > 1e-4:
    count += 1
    F = fock_matrix(Hcore, P, dim) # Calculate Fock matrix (step 6)
    Fprime = f_transform(X_matrix, F) # Calculate transformed Fock matrix (step 7)
    E, Cprime = np.linalg.eigh(Fprime) # Diagonalize F' matrix (step 8)
    C = np.dot(X_matrix, Cprime) # transform the coefficients into original basis using
                                 transformation matrix (step 9)
    P, OLDP = density_matrix(C, P, dim, Nelec) # Make density matrix (step 10)
    DELTA = threshold(P, OLDP) # Test for convergence (step 11)
    print("E = {:.6f}, N(SCF) = {}".format(energy_iteration(P, Hcore, F, dim), count))

print("SCF procedure complete, TOTAL E(SCF) = {} a.u.".format(energy_iteration(P, Hcore, F, dim)))
print("------------------------")
print("The total energy is: ", energy_iteration(P, Hcore, F, dim) + nuclear_repulsion, "a.u.")
print("------------------------")
print("The orbital energies matrix is: ","\n", np.diag(E))
print("------------------------")
print("The final delta value is: ",DELTA)
```

The output of the code is:

E = -2.714117, N(SCF) = 1
E = -1.942439, N(SCF) = 2
E = -1.839035, N(SCF) = 3
E = -1.832375, N(SCF) = 4
E = -1.831028, N(SCF) = 5
SCF procedure complete, TOTAL E(SCF) = -1.8310283465028736 a.u.

The total energy is: -1.1167571937431286 a.u.

The orbital energies matrix is:
[[-0.5782593 0.]
[0. 0.67035598]]

The final delta value is: 6.430147411898742e-06

The input files used in this program have the molecular orbitals' integrals in the form of the following matrices:

$$\mathbf{S} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 0.6593 \\ 2 & 2 & 1 \end{pmatrix} \mathbf{T} = \begin{pmatrix} 1 & 1 & 0.7600 \\ 2 & 1 & 0.2365 \\ 2 & 2 & 1 \end{pmatrix} \mathbf{V} = \begin{pmatrix} 1 & 1 & -2.4532 \\ 2 & 1 & -1.1948 \\ 2 & 2 & -1.3076 \end{pmatrix} (\mu\upsilon|\sigma\lambda) = \begin{pmatrix} 2 & 1 & 2 & 1 & 0.2970 \\ 2 & 1 & 1 & 1 & 0.4441 \\ 1 & 1 & 2 & 2 & 0.5697 \\ 1 & 1 & 1 & 1 & 0.7746 \end{pmatrix} \quad (3)$$

The potential energy integrals (V) are the same for both atoms of hydrogen, so the total potential energy is the sum of both of them.

2 Problem 2

Rewrite your program to consider the case of a HHe+ diatomic molecule with an internuclear distance $R = 1.4632\,a.u.$ We will use the molecular integrals calculated for 1s type STO-3G orbitals with $\zeta(He) = 2.0925$ and $\zeta(H) = 1.24$. The molecular integrals (in a.u.) for this case, considering the He atom at position 1, are:

$$\begin{array}{ll} S_{12} = 0.4508 & (11|11) = 1.3072 \\ t_{11} = 2.1643 & (22|22) = 0.7746 \\ t_{22} = 0.7600 & (22|11) = 0.6057 \\ t_{12} = 0.16170 & (21|11) = 0.4373 \\ V^1_{11} = -4.1398 & (22|21) = 0.3118 \\ V^1_{12} = -1.1029 & (21|21) = 0.1773 \\ V^1_{22} = -1.2652 & \\ V^2_{11} = -0.6772 & \\ V^2_{12} = -0.4113 & \\ V^2_{22} = -1.2266 & \end{array}$$

Compare the values of these integrals with those of the H2 molecule and explain why they are larger / smaller in this case.

To start the iteration process you can use a null density matrix.

If your program is working well your final MOs should correspond to:

$$C = \begin{pmatrix} 0.8019 & -0.7823 \\ 0.3368 & 1.0684 \end{pmatrix} \quad (4)$$

$$\mathbf{e} = \begin{pmatrix} -1.5975 & 0.0 \\ 0.0 & -0.0617 \end{pmatrix} \quad (5)$$

The total energy for this interatomic distance is $U_0(R = 1.4632\,a.u.) = -2.860662\,a.u.$

```
import numpy as np
import sys

# Step 1: specifying molecular geometry, basis set and other variables
Nelec = 2
R = 1.4632 #bohr
dim = 2 #dim is the number of basis functions
# Molecular integrals are read from input files
S_int = np.genfromtxt('./s_2.dat',dtype=None) # overlap matrix
T_int = np.genfromtxt('./t_2.dat',dtype=None) # kinetic energy matrix
V_int = np.genfromtxt('./v_2.dat',dtype=None) # potential energy matrix
te_int = np.genfromtxt('./elec_2.dat') # two electron integrals

# Step 2: Calculate all the required molecular integrals S,T,V and 2-electron
S = np.zeros((dim, dim))
T = np.zeros((dim, dim))
V = np.zeros((dim, dim))
# Put the integrals into a matrix
for i in S_int:
    S[i[0]-1, i[1]-1] = i[2]
for i in T_int:
```

```
    T[i[0]-1, i[1]-1] = i[2]
for i in V_int:
    V[i[0]-1, i[1]-1] = i[2]

# Step 3: Diagonalize S and use its eigenvalues to obtain a transformation matrix X
def symmetrise(matrix):
    """
    Function to symmetrize a matrix given a triangular one
    """
    return matrix + matrix.T - np.diag(matrix.diagonal())

# Flip the triangular matrix in the diagonal
S = symmetrise(S)
V = symmetrise(V)
T = symmetrise(T)
# Form core Hamiltonian matrix as sum of the T and V matrices
Hcore = T + V

# Diagonalize overlap matrix. S_val are the eigenvalues and S_vec the eigenvectors
S_val, S_vec = np.linalg.eigh(S)
# Find inverse square root of eigenvalues
s_half = (np.diag(S_val**(-0.5)))
# Form the transformation matrix X. The unitary matrix are the eigenvectors
X_matrix = -np.dot(S_vec, np.dot(s_half, np.transpose(S_vec)))

# Step 4: Construct a guess denisty matrix P(0) (null)
P = np.zeros((dim, dim))

# Step 5: Calculate the bielectronic term G(0) using the guess denisty matrix and the
two electron integrals
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

# two-electron integrals are stored in a dictionary
twoe = {eint(row[0], row[1], row[2], row[3]) : row[4] for row in te_int}

def two_elec_int(a, b, c, d): # Return value of two electron integral
    """
    Return value of two electron integral
    """
    return twoe.get(eint(a, b, c, d), 0)

# Step 6: Add G(0) to the one-electron term h to get a first guess of the Fock matrix
F(0) = h + G(0).
def fock_matrix(Hcore, P, dim):
    """
    Function to build the Fock Matrix
    """
    F = np.zeros((dim, dim)) # zero array
```

```
    for i in range(0, dim):
        for j in range(0, dim):
            F[i,j] = Hcore[i,j] # initial Fock matrix
            for k in range(0, dim):
                for l in range(0, dim):
                    # Form the Fock matrix using the product of the density matrix and G matrix
                    F[i,j] = F[i,j] + P[k,l]*(two_elec_int(i+1,j+1,k+1,l+1)-
                             0.5*two_elec_int(i+1,k+1,j+1,l+1))
    return F

# Step 7: Transform the Fock matrix
def f_transform(X, F):
    """
    Transform Fock matrix with the transformation matrix X
    """
    return np.dot(np.transpose(X), np.dot(F, X))

# Step 10: Form a new density matrix P(1) using C(1)
def density_matrix(C, D, dim, Nelec): # Make density matrix and store old one to test
for convergence
    """
    Make new density matrix and store old one to test for convergence

    Returns:
        D: new density matrix
        Dold: old denisty matrix
    """
    Dold = np.zeros((dim, dim)) # Initiate zero array
    for mu in range(0, dim):
        for nu in range(0, dim):
            Dold[mu,nu] = D[mu, nu] # Set old density matrix to the density matrix, D,
                                    input into the function
            D[mu,nu] = 0
            for m in range(0, int(Nelec/2)):
                # Form new density matrix
                D[mu,nu] = D[mu,nu] + 2*C[mu,m]*C[nu,m]
    return D, Dold

# Step 11: Determine if the process has converged by comparing P(1) with P(0).
def threshold(D, Dold):
    """
    Calculate change in density matrix
    """
    DELTA = 0.0
    for i in range(0, dim):
        for j in range(0, dim):
            DELTA = DELTA + ((D[i,j] - Dold[i,j])**2)

    return (DELTA/4.0)**(0.5) #1/m**2, m=dim

# Step 12: If the process has converged, use the resultant solution, represented by C(k),
P(k), and F(k)
def energy_iteration(D, Hcore, F, dim):
    """
    Function that calculates the energy at each iteration
    """
    EN = 0
    for mu in range(0, dim):
        for nu in range(0, dim):
            EN += 0.5*D[mu,nu]*(Hcore[mu,nu] + F[mu,nu])
    return EN
```

```
# Finally we make the iteration loop
DELTA = 1
count = 0 # cycles counter
nuclear_repulsion = 2/R # Z(H)*Z(He)=2
while DELTA > 1e-4:
    count += 1
    F = fock_matrix(Hcore, P, dim) # Calculate Fock matrix (step 6)
    Fprime = f_transform(X_matrix, F) # Calculate transformed Fock matrix (step 7)
    E, Cprime = np.linalg.eigh(Fprime) # Diagonalize F' matrix (step 8)
    C = np.dot(X_matrix, Cprime) # transform the coefficients into original basis using
                                 transformation matrix (step 9)
    P, OLDP = density_matrix(C, P, dim, Nelec) # Make density matrix (step 10)
    DELTA = threshold(P, OLDP) # Test for convergence (step 11)
    print("E = {:.6f}, N(SCF) = {}".format(energy_iteration(P, Hcore, F, dim)+
    nuclear_repulsion, count))

print("SCF procedure complete, TOTAL E(SCF) = {} a.u.".format(energy_iteration(P, Hcore,
  F, dim) + nuclear_repulsion))
print("------------------------")
print("The expansion coefficients matrix is: ","\n", C)
print("------------------------")
print("The orbital energies matrix is: ","\n", np.diag(E))
print("------------------------")
print("The final delta value is: ",DELTA)
```

The output of the code is:

E = -3.983973, N(SCF) = 1
E = -2.782644, N(SCF) = 2
E = -2.857891, N(SCF) = 3
E = -2.865606, N(SCF) = 4
E = -2.866191, N(SCF) = 5
E = -2.866234, N(SCF) = 6
SCF procedure complete, TOTAL E(SCF) = -2.866233596471182 a.u.

The expansion coefficients matrix is:
[[0.80019851 -0.78405034]
[0.33913343 1.06772708]]

The orbital energies matrix is:
[[-1.60128635 0.]
[0. -0.05293131]]

The final delta value is: 1.1695797349974723e-05

The same way as the previous problem, the input files used in this program have molecular orbitals' integrals in the form of the following matrices:

$$\mathbf{S} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 1 & 0.4508 \\ 2 & 2 & 1 \end{pmatrix} \mathbf{T} = \begin{pmatrix} 1 & 1 & 2.1643 \\ 2 & 1 & 0.1617 \\ 2 & 2 & 0.7600 \end{pmatrix} \mathbf{V} = \begin{pmatrix} 1 & 1 & -4.8170 \\ 2 & 1 & -1.5142 \\ 2 & 2 & -2.4918 \end{pmatrix} (\mu\upsilon|\sigma\lambda) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1.3072 \\ 2 & 2 & 2 & 2 & 0.7746 \\ 2 & 2 & 1 & 1 & 0.6057 \\ 2 & 1 & 1 & 1 & 0.4373 \\ 2 & 2 & 2 & 1 & 0.3118 \\ 2 & 1 & 2 & 1 & 0.1773 \end{pmatrix} \quad (6)$$

The overlap integral (S) is smaller in the HHe^+ molecule because the molecular orbitals of the hydrogen atom and the helium ion are more overlapped than in the H_2 molecule; meaning that the nuclei are closer in the HHe^+ molecule. For the same reason, nuclear attraction integrals (V) also have higher values in the HHe^+ molecule.