

MARCH 10 2021

# PROGRAMMING PROJECT 3: RESTRICTED HARTREE-FOCK

Van der Stichelen Ruben, Wieme Xander

# CONTENTS

1 WHY?

2 How?

# WHY?

# WHY?

- Schrödinger Equation

$$\hat{H}\Psi = E\Psi$$

- True Hamiltonian

$$\hat{H} = \sum_i^N \hat{h}(i) + \frac{1}{2} \sum_i^N \sum_j^N \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|}$$

# WHAT DO WE NEED?

## 1 Slater Determinant

$$|\Psi\rangle = \frac{1}{\sqrt{N!}} \begin{vmatrix} \chi_1(\mathbf{x}_1) & \cdots & \chi_n(\mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ \chi_1(\mathbf{x}_n) & \cdots & \chi_n(\mathbf{x}_n) \end{vmatrix}$$

## 2 spin-orbitals

$$\chi_i(\mathbf{x}_j) = \psi_i(\mathbf{r}_j)\gamma(\omega_j)$$

■ Expectation value

$$\begin{aligned}\langle \Psi | \hat{H} | \Psi \rangle &= \sum_i^N \langle \chi_i | \hat{h}(1) | \chi_i \rangle \\ &+ \frac{1}{2} \sum_i^N \sum_j^N \left( \int \chi_i^*(1) \chi_i(1) \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \chi_j^*(2) \chi_j(2) d1 d2 \right. \\ &\quad \left. - \int \chi_i^*(1) \chi_j(1) \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \chi_j^*(2) \chi_i(2) d1 d2 \right)\end{aligned}$$

- Energy is stationary

$$E(\chi_i + \delta\chi_i) - E(\chi_i) = \delta E = 0$$

- Correct for non-orthonormality

$$L = E - \sum_{ij} \epsilon_{ij} (\langle \chi_i | \chi_j \rangle - \delta_{ij})$$

$$\delta L = \sum_i \langle \delta\chi_i | \hat{h}(1) + \sum_j (\hat{J}_j - \hat{K}_j) | \chi_i \rangle - \sum_{ij} \epsilon_{ij} \langle \delta\chi_i | \chi_j \rangle + CC = 0$$

## SOME HELPFULL OPERATORS

- Coulomb operator

$$\langle \chi_i | \hat{J}_j | \chi_i \rangle = \int \chi_i^*(1) \chi_j^*(2) \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \chi_i(1) \chi_j(2) d1 d2$$

- exchange operator

$$\langle \chi_i | \hat{K}_j | \chi_i \rangle = \int \chi_i^*(1) \chi_j(1) \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \chi_j^*(2) \chi_i(2) d1 d2$$



■ Condition

$$\left( \hat{h}(1) + \sum_j (\hat{J}_j - \hat{K}_j) \right) \chi_i = \sum_j \epsilon_{ij} \chi_j$$

■ Hartree-Fock Equation

$$\hat{f}(1) \chi_i = \epsilon_i \chi_i$$

# SPIN ELIMINATION

- No interaction with spin  $\rightarrow$  remove it
- Example:  $\alpha$ -Spin

$$\langle \alpha | \hat{f}(1) | \psi_i(\mathbf{r}_1) \alpha \rangle = \epsilon_i \psi_i(\mathbf{r}_1)$$

- Exchange operator?

$$\hat{K}_j | \psi_i(1) \alpha(1) \rangle = \int \chi_j(1) \frac{1}{|\mathbf{r}_i - \mathbf{r}_j|} \chi_j^*(2) \psi_i(2) \alpha(2) d1 d2$$

# RESTRICTED HARTREE FOCK

- Only paired electrons (Pauli principle)
- In the same orbitals

$$\hat{f}(1) = \hat{h}(1) + 2 \sum_i^{N/2} \hat{J}_i - \sum_i^{N/2} \hat{K}_i$$

# SOLVING EQUATIONS

- Begin iterations

$$\mathbf{H}^{core} \mathbf{C} = \mathbf{SCc}$$

- Roothaan-Hall Equations

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

# THE DENSITY MATRIX

- Expansion in the basis

$$\hat{J}_i = \int \sum_{\sigma} c_{\sigma}^* \phi_{\sigma}^*(x_j) \frac{1}{|\mathbf{r}_1 - \mathbf{r}_j|} \sum_{\nu} c_{\nu} \phi_{\nu}(x_j) dx_j$$

- Density Matrix

$$D_{\sigma\nu} = 2 \sum_j^{N/2} c_{\sigma j}^* c_{\nu j}$$

How?

## How?

```
class Molecule:
    def __init__(self, geom_file):
        if "pubchem" in geom_file:
            self.id = psi4.geometry(geom_file)
        else:
            input = open(filename, 'r').readlines()
            data = ""
            for i, row in enumerate(input[1:]):
                Z, x, y, z = row.split()
                data += f"{int(float(Z))} {x} {y} {z}\n"
            ...
```

```
...  
    data += "units bohr"  
    self.id = psi4.geometry(data)  
self.id.update_geometry()  
self.wfn = psi4.core.Wavefunction.build(self.id, psi4.core.  
    ↪ get_global_option('basis'))  
self.basis = self.wfn.basisset()  
self.integrals = psi4.core.MintsHelper(self.basis)  
self.occupied = self.wfn.nalpha()
```



## NUCLEAR REPUSLTION ENERGY

```
def displayNucRep(self):  
    return self.id.nuclear_repulsion_energy()
```

# ONE ELECTRON INTEGRALS

```
def one_electron_integrals(self):  
    self.S = self.integrals.ao_overlap().np  
    self.T = self.integrals.ao_kinetic().np  
    self.V = self.integrals.ao_potential().np  
    self.H_core = self.T + self.V
```

## TWO ELECTRON INTEGRALS

```
def ElectronRepulsion(self):  
    self.tei = self.integrals.ao_eri().np
```

## THE INITIAL (GUESS) MATRIX

```
def density_matrix(self, initial=False):  
    # First time guess  
    if initial:  
        self.C = eigh(self.H_core, self.S)[1]  
  
    # Return the density matrix  
    self.D = 2 * np.einsum('ij,jk->ik', self.C[:, :self.occupied], self.C.  
        ↪ T[:self.occupied, :], optimize=True)
```

# THE FOCK MATRIX

```
def fock_matrix(self):  
    # tei is a 4D numpy array where the indices are given according to the  
    ↪ integral in step 2.3  
    # The coulomb integral is the tei matrix, the exchange matrix is the  
    ↪ tei matrix where the second and third  
    # indices are swapped  
    self.F = self.H_core + np.einsum('ij,uvij→uv', self.D, self.tei -  
    ↪ 1/2 * np.swapaxes(self.tei, 1, 2))
```

# THE SCF ENERGY

```
def electronic_energy(self):  
    self.E_elec = 1/2*np.einsum('ij,ij', self.D, self.H_core + self.F)  
  
def total_energy(self):  
    self.E_tot = self.id.nuclear_repulsion_energy() + self.  
        ↪ electronic_energy()
```

# THE SCF ENERGY

```
def SCF(self):  
    iteration = 0  
    iterations = [0]  
    energies = list()
```

```
# Init
self.one_electron_integrals()
self.ElectronRepulsion()
self.density_matrix(initial=True)
self.fock_matrix()
self.electronic_energy()
self.total_energy()

prev_D = self.D
prev_E_elec = self.E_elec

energies.append(prev_E_elec)
```



```
# Convergence variables
delta_E_elec = 1
delta_D = 1

# Print header
print("{} {:>15} {:>15} {:>15}".format("i", "E(elec)" , "Delta(E)"
    ↪ , "RMS(D)"))
```

```
# Repeat until the density and enery differences are in there
    ↪ thresholds or max iteration is reached
while np.abs(delta_D) > 10**(-4) and iteration < 50:
    # Update
    self.C = eigh(self.F, self.S)[1]
    self.density_matrix()
    self.fock_matrix()
    self.electronic_energy()
```

```
# Tets convergence
delta_E_elec = self.E_elec - prev_E_elec
delta_D = np.sqrt(np.einsum('ij->', np.square(self.D - prev_D)
    ↪ ))
```

```
# Print progress
iteration += 1
iterations.append(iteration)
energies.append(self.E_elec)
print("{} {:>15.10f} {:>15.10f} {:>15.10f}".format(iteration,
    ↪ self.E_elec , delta_E_elec, delta_D))

prev_D = self.D
prev_E_elec = self.E_elec
print(self.total_energy)
```

```
fig = plt.figure()
ax = fig.add_subplot()
ax.set_xlabel("iterations")
ax.set_ylabel("energy")
ax.plot(iterations, energies)
plt.show()

water = Molecule("""pubchem:water""")
water.SCF()
```

# WATER

i	E( elec)	Delta (E)	RMS(D)
1	-82.8304924416	-1.5423289586	3.6533461690
2	-82.9378550598	-0.1073626183	0.9591407297
3	-82.9438448043	-0.0059897445	0.1736633778
4	-82.9443390579	-0.0004942536	0.0620522727
5	-82.9444231232	-0.0000840653	0.0215985664
6	-82.9444414809	-0.0000183577	0.0105096537
7	-82.9444457095	-0.0000042285	0.0048771593
8	-82.9444466920	-0.0000009825	0.0023545591
9	-82.9444469206	-0.0000002287	0.0011290864
10	-82.9444469738	-0.0000000532	0.0005444100
11	-82.9444469862	-0.0000000124	0.0002623606
12	-82.9444469891	-0.0000000029	0.0001265513
13	-82.9444469898	-0.0000000007	0.0000610451
	-74.94207992798844		

