FACULTY
OF SCIENCES

Report restricted Hartree Fock

Ruben Van der Stichelen

$\langle \mathbf{G}|\mathbf{QC}|\mathbf{G}\rangle$

Ghent University
April 16, 2021

GHENT
UNIVERSITY

1 Introduction

This report will discuss the notebook we made on restricted Hartree-Fock theory. We will walk trough the notebook step by step and discuss what we see there. The same titles as in the notebook will be used, as to make sure the report can be easily followed. Relevant parts of the code will be displayed.

2 Theoretical Approach

2.1 Introduction to Theory

Before we actually dive into the code, we will discuss the theory behind it. The actual goal of this experiment is to calculate the energy of a molecule, mainly to solve the Schrödinger equation for that molecule. This, unfortunately, is impossible. The interelectronic interactions prohibit us from separating the variables. Intuitvely, we would need the coordinates of the second electrons to determine the coordinates of the second electron. But for those, we would need the coordinates of the first electron again. So we have to find an alternative. This alternative would be Hartree-Fock theory. Bluntly put we can define another operator, the Fock operator, that consists of one-electron operators and two-electron operators. This is displayed in Equation 1.

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

$\hat{h}$ is a one electron Hamiltonian, which accounts for the kinetic energy of the electron and the potential energy with respect to the nucleus. $\hat{J}$ is the Coulomb operator, it accounts for the interelectronic repulsion. $\hat{K}$ is the exchange operator, that accounts for the stabilizing interaction between electrons that have the same spin.

2.2 Deriving the Fock Operator

Let us step back a bit here to see where this operator comes from. To do that, we need to begin with the true Hamiltonian for the system, which is given in Equation 2

$$\hat{\boldsymbol{H}} = \sum_{i}^{N} \hat{h}(i) + \frac{1}{2}\sum_{i}^{N}\sum_{j}^{N} \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \tag{2}$$

$\hat{h}(i)$ is a one electron operator, as said before. The second term accounts for interelectronic interactions. $|\boldsymbol{r}_i - \boldsymbol{r}_j|$ is merely the distance between electrons i and j. Now, we also need a wavefunction Ψ, which is a Slater determinant. This determinant looks like Equation 3.

$$|\Psi\rangle = 1/\sqrt{N!}\begin{vmatrix} \chi_1(\boldsymbol{x}_1) & \cdots & \chi_n(\boldsymbol{x}_1) \\ \cdots & \cdots & \cdots \\ \chi_1(\boldsymbol{x}_n) & \cdots & \chi_n(\boldsymbol{x}_n) \end{vmatrix} \tag{3}$$

When we look at the individual functions $\chi_i(\boldsymbol{x}_j)$ we can state Equation 4

$$\chi_i(\boldsymbol{x_j}) = \psi_i(\boldsymbol{r}_j)\gamma(\boldsymbol{\omega}_j) \tag{4}$$

Here, ψ is a spatial orbital, γ is a spin function, which can be either α or β. Now we have all the ingredients, we can start building the Fock matrix. The first step would be to calculate the expectation value of the Hamiltonian over the wave function Ψ. First of all we assert that all spin-orbitals are normalised, so that $\langle\chi_i|\chi_i\rangle = 1$. We will not give the exact derivation here, but jump straight to the end result. This is given in Equation 5.

$$\langle\Psi|\,\hat{\boldsymbol{H}}\,|\Psi\rangle = \sum_i^N \langle\chi_i|\,\hat{h}(1)\,|\chi_i\rangle + \frac{1}{2}\sum_i^N\sum_j^N \left(\int \chi_i^*(1)\chi_i(1)\frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|}\chi_j^*(2)\chi_j(2)d1d2 - \int \chi_i^*(1)\chi_j(1)\frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|}\chi_j^*(2)\chi_i(2)d1d2\right) \tag{5}$$

This expectation value can always be calculated, even if Ψ is not an eigenfucntion of the Hamiltonian. The only problem we have here is the choice of the spin-orbitals. We need to choose them in such a way that the energy is minimal. This would mean that the energy is a stationary point, so we enforce Equation 6.

$$E(\chi_i + \delta\chi_i) - E(\chi_i) = \delta E = 0 \tag{6}$$

E is off course nothing else then the expectation value of the wave function Ψ over the Hamiltonian. We could write $\langle\Psi|\,\hat{\boldsymbol{H}}\,|\Psi\rangle = E(\chi_i)$. Now we will enforce orthonormality of the spin-orbitals via a seperate function, as can be seen in Equation 7.

$$L = E - \sum_{ij}\epsilon_{ij}(\langle\chi_i|\chi_j\rangle - \delta_{ij}) \tag{7}$$

This is another constraint we need to enforce, since whatever fucntions we are looking for, they need to be orthonormal. If they are not, we need to make a correction. This is done by introducing the fucntion L. It is easy to see that in the case of orthonormal spin-orbitals L reduces to E. The ϵ_{ij} are the Lagrange multipliers. Now we will calculate δL, this is possible

with the information that is given. We will once again skip to the end. The final equation is given in Equation 8.

$$\delta L = \sum_i \langle \delta\chi_i | \hat{h}(1) + \sum_j (\hat{J}_j - \hat{K}_j) | \chi_i \rangle - \sum_{ij} \epsilon_{ij} \langle \delta\chi_i | \chi_j \rangle + complex\ conjugate = 0 \quad (8)$$

We have added the Coulomb and exchange operators. When we define the expectation values as Equations 9 and 10, we see that we can indeed add them in this way.

$$\langle \chi_i | \hat{J}_j | \chi_i \rangle = \int \chi_i^*(1)\chi_j^*(2) \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \chi_i(1)\chi_j(2) d1d2 \quad (9)$$

$$\langle \chi_i | \hat{K}_j | \chi_i \rangle = \int \chi_i^*(1)\chi_j(1) \frac{1}{|\boldsymbol{r}_i - \boldsymbol{r}_j|} \chi_j^*(2)\chi_i(2) d1d2) \quad (10)$$

When we check Equation 8, we see that it can only hold if Equation 11 and its counterpart from the complex conjugate both hold.

$$\left(\hat{h}(1) + \sum_j (\hat{J}_j - \hat{K}_j) \right) \chi_i = \sum_j \epsilon_{ij} \chi_j \quad (11)$$

These two equations allow us to conclude that $\epsilon_{ij}^* = \epsilon_{ji}$. These factors are called the Lagrange multipliers and form a Hermitian matrix. When we take a closer look at the operator in Equation 11, we can recognise the Fock operator from Equation 1, although we are not quite there yet. However we can already write some interesting equations with the formulas we have. We can write Equation 12.

$$\hat{f}(1)\chi_i = \epsilon_i \chi_i \quad (12)$$

These equations are called the Hartree Fock equations.

2.3 Restricted Hartree-Fock

As we said in the last subsection, we are not quite where we want to be just yet. We need to get to the Fock matrix from Equation 1, but we are only at the one in 11. This is where we will make the jump to restricted Hartree Fock theory. To start, we will take a look at Equation 4. Here we saw that the spin-orbital is actually a product of a spatial orbital and a spin function. When we take a look at our operators however, we notice that none of them will actually interact with this spin function. We would then like to get rid of it. Doing this is actually not that difficult. If we would collect all functions that have α as their spin component, we can write a Hartree-Fock equation and then simply integrate the spin component out by adding

$\langle\alpha|$ on every side of the equation. For the one electron part and the Coulomb operators, there is no problem at all, since all electrons stay in their respective orbitals. However, the exchange operator will move electrons around to different spin-orbitals.(see Equation 10) When the electron from an $alpha$ spin-orbital would be moved to a $beta$ spin-orbital, this would result in a zero, since $\langle\alpha|\beta\rangle = 0$ So here it is clear that the exchange operator will only remain active after spin elimination between electrons that have the same spin function. So now that we have gotten rid of spin, all that is left is Equation 13.

$$\hat{f}^{\alpha}(1)\psi_i^{\alpha} = \epsilon_i^{\alpha}\psi_i^{\alpha} \tag{13}$$

Off course the same argument stands for the β-spins.

Now we have come to a point where we can define a restricted closed shell system. Imagine what would happen if all orbitals were doubly occupied, with an α and a β electron, all according to the Pauli princple. This has some consequences. The first one is that the amount of α-electrons would equal the amount of β-electrons. Secondly, this would mean that the difference between these electrons is only their spin function. They reside in the same orbitals. This means that we only need to account for half the electrons, since the other half is in the same orbitals, just with a different spin. Now think about the Fock operator. Let us take a look at a single electron. It has a one election hamiltonian, electronic repulsion for all other electrons in the system and it only "exchanges" with half of the other electrons, only those with the same spin. This means we have a double the amount of Coulomb operators to exchange operators. When we now look back at Equation 1, we see that we are finally there.

2.4 Application of the Hartree-Fock Theory

At this point, we might think back to the point where we defined the Coulomb and exchange operators. These operators themselves depend on the orbitals. So we would need to know the orbitals to define our operator. It seems we are stuck again. However we can apply some mathemathical tricks to get us out of this impasse. First we can assert that every function can be expressed as a linear combination of basis functions. We also know that the eigenfunctions of a Hermitian operator form such a set of basis functions. We begin by selecting a convenient basis set. This could be a set of atomic orbitals. Then we would solve the eigenproblem in Equation 14.

$$\hat{\boldsymbol{H}}^{core}\boldsymbol{C} = \boldsymbol{S}\boldsymbol{C}c \tag{14}$$

C is a coeficcient matrix. It holds the coeficcients of the eigenfunctions of the Hamilonian (or later the fock operator) in the basis. Calculating the eigenvalues and eigenfunctions for

the core Hamiltonian is perfectly possible, since it is a one-electron operator. We now have a set of orbitals which we can use to build our Fock operator. For this Fock operator, we can solve the generalised eigenproblem in the form of the Roothaan-Hall equations (see Equation 15).

$$\boldsymbol{FC} = \boldsymbol{SC}\epsilon \tag{15}$$

Solving this equation will give us a set of eigenfunctions of the Fock matrix. These functions can then be used to build a new Fock matrix. This is where we start the iterations. We can continue untill we reach a point where the energy does not change any more. At that point, we have found the energy of the molecule.

Before we move on, some clarifications are in order.

1. The basis set we choose is not the same as the set that is derived from the eigenfunctions of the Hamiltonian. These eigenfunctions are in fact our first guess of the actual molecular orbitals. However, this guess will not be very good yet, since of course the core Hamiltonian is not the same as the molecular Hamiltonian.

2. A basis set is often chosen for the ease of integration. Often, these are Gaussian functions. So we can see that the chosen basis set does not need to have the same properties as the eigenfunctions of the Hamiltonian. Of course the functions do have to be "well-behaved". The size however, will always be the same. If you choose a basis set with only two functions, your Hamiltonian will have only two eigenfunctions. The bigger the basis set, the more eigenfunctions you will have.

3. We are using an expansion of a function in a basis. That is always possible. However, there is always a catch, namely that the basis set used has to be complete. In most cases this would actually mean that it is infinitely large, so of no real use to us. However, if we choose an incomplete set, we will only be able to make approximations.

4. If we want to do the expansion, we need to integrate the spin out first. So we need to end up with an expansion that is only dependent on the spacial orbital, like in Equation 16.

$$\psi_i = \sum_j C_{ji}\phi_j \tag{16}$$

2.5 The Density Matrix

At this point we still need to adress one topic, the density matrix. This will be very important since it will be used to calculate the Fock matrix in our program. To demonstrate what it is, we will look at the expression for the Coulomb operator. When we take Equation 9 and then expand the functions χ_i in a basis after eliminating the spin, we find Equation 17.

$$\hat{J}_i = \int \sum_{\sigma} C_{\sigma}^{*} \phi_{\sigma}^{*}(x_j) \frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_j|} \sum_{\nu} C_{\nu} \phi_{\nu}(x_j) dx_j \tag{17}$$

The C factors in this equation are merely the exapansion coeficcients of the function ψ_i.(Rememmber that this is the spatial orbital, see Equation 4) These are constants, so we can safely extract them from the integrals to form Equation 18.

$$\hat{J}_i = \sum_{\sigma} \sum_{\nu} C_{\nu j} C_{\sigma j}^{*} \int \phi_{\sigma}^{*}(x_j) \frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_j|} \phi_{\nu}(x_j) dx_j \tag{18}$$

We can apply this same strategy to the exchange operator. In the end we will be able to factor out the products of the expansion coëficcients and afer summation over all basis functions we can define the density matrix as Equation 19.

$$D_{\sigma\nu} = 2 \sum_{j}^{N/2} C_{\sigma j}^{*} C_{\nu j} \tag{19}$$

This will be used to calculate the Fock matrix. (see below). However, before we move on, there is something noteworthy here. We only have to account for the occupied orbitals j. Let us take the example where there are seven eigenfucntions and only ten electrons. In restricted closed shell Hartree Fock that would mean five doubly occupied orbitals. We would only have to consider the five occupied orbitals here. You would still end up with a seven by seven density matrix. This can be seen more clearly in Section 5.

3 Identifying the Molecule

In this first step we initiated the class molecule, which will be equipped with the necessary methods to do all the calculations we will be required to do. A molecule object has several properties that need to be defined first.

Listing 1: intitialising the molecule object

```
class molecule:
```

```
        def __init__(self, geom_file):
            if """pubchem""" in geom_file:
                self.id = psi4.geometry(geom_file)
            else:
                self.id = psi4.geometry(f"""
                {geom_file}

                units bohr
                """)
            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)
            # only works for closed shell systems
            self.occupied = self.wfn.nalpha()
            self.guessMatrix = "empty"
```

In Listing 1 we define the **__init__** method of the molecule class. It sets us up with some of the information we will need later, namely the `psi4.core.Molecule` representation as **self.id**. We also see the wave function, basis set, integrals, occupied orbitals and a guess matrix. Since we will be doing Hartree-Fock calculations, which involve a lot of iterations, the molecule object will need a way to store the Fock matrix from the last iteration. From there we can then start for the next iteration. Hence, the first method we actually have to define is a method that allows us to change this parameter.

Listing 2: setting the guessMatrix

```
    def setGuess(self, new_guess):
        """
        sets the guessMatrix to a new value

        input:
        new_guess: numpy array that represents a new fock matrix
        """
        self.guessMatrix = new_guess
```

This is shown in Listing 2.

4 Prerequisite Calculations

In this section, we do some prerequisite calculations. These are relatively straightforward. In this step, we added some methods to the class that allow us to calculate important properties like the nuclear repulsion, kinetic energy and so on. The commands used are directly implemented from the psi4 package. They are listed in Table 1

command	property
`self.id.nuclear_repulsion_energy()`	nuclear repulsion energy
`self.integrals.ao_overlap().np`	overlap matrix
`self.integrals.ao_kinetic().np`	kinetic energy
`self.integrals.ao_potential().np`	potential energy
`self.displayE_kin() + self.displayE_pot()`	the core hamiltonian
`self.integrals.ao_eri().np`	repulsion between electrons

Table 1: Commands useed to calculate various properties

We will not list the code for the various methods here, however when they appear in later blocks of code we will mention them.

5 The inital (Guess) Density Matrix

We will skip ahead to the function that gives us the density matrix and start from there.

Listing 3: calculating the density matrix

```
class molecule(molecule):
    def getDensityMatrix(self):
        """
        generates the densitiy matrix on the AO level
        """
        C = self.getEigenStuff()[1]
        A = 2*np.einsum("pq, qr->pr", C[:, :self.occupied],
                        C[:, :self.occupied].T, optimize=True)
        return A
```

This function uses the method `getEigenStuff`, which just calls the scipy function `linalg.eigh`. This solves the generalised eigenproblem posed by the Roothaan-Hall equations, Equation 15, for whatever matrix that is currently in the `self.guessMatrix` parameter. C in Listing 3 then refers to **C** in Equation 15. The first matrix for which we calculate the eigenvalues and

eigenvectors is the core Hamiltonian, so this will have to be stored as the **self.guessMatrix** before proceeding. Now we are set to build a Fock matrix.

6 Updating the Fock Matrix

Listing 4: calculating the Fock matrix

```
class molecule(molecule):
    def displayFockMatrix(self):
        """Will display the Fock matrix"""
        coulomb = np.einsum("nopq,pq->no",
                self.displayElectronRepulsion(),
                self.getDensityMatrix(), optimize=True)
        exchange = np.einsum("npoq,pq->no",
                self.displayElectronRepulsion(),
                self.getDensityMatrix(), optimize=True)
        self.fockMatrix = self.displayHamiltonian()
                        + coulomb - 0.5*exchange
        return self.fockMatrix
```

We see that the Fock-matrix uses the density matrix and the electronic repulsion matrix and the Hamiltonian , which the molecule object calls with the methods seen in Section 4. Since the density matrix depends on the current **self.guessMatrix**, the Fock matrix will also depend on it. We will discuss the importance of this in Subsection 8.1.

7 The SCF Energy

From the Fock matrix we derived in the previous section, we can now calculate the electronic energy according to Equation (20).

$$E_{elek} = \frac{1}{2} \sum_{\mu\nu} D_{\mu\nu} \cdot (H_{\mu\nu} + F_{\mu\nu}) \tag{20}$$

In Python this looks like Listing 5.

Listing 5: calculating the energy

```
class molecule(molecule):
    def getElectronicEnergy(self):
        """
        calculates the energy with the current fock matrix
```

```
        """
        sumMatrix = self.displayHamiltonian()
                + self.displayFockMatrix()
        return 0.5*np.einsum("pq,pq->", sumMatrix,
                             self.getDensityMatrix())
```

The total energy is merely the sum of this electronic energy and the nuclear repulsion energy. We have a method for the latter defined in Section 4.

8 Test for Convergence

Now we only need to bring this all together to get the final energy. This is done in the function `iterator` as seen in Listing 6.

Listing 6: iteration sequence

```
def iterator(target_molecule):
    """
    Function that performs the Hartree-Fock iterative calculations
    for the given molecule.

    input:
    target_molecule: a molecule object from the class molecule
    """
    # setting up entry parameters for the while loop
    E_new = 0
    E_old = 0
    d_old = target_molecule.getDensityMatrix()
    convergence = False
    E_list = []

    # step 2: start iterating
    itercount = 0
    while not convergence and itercount < 50:

        # calculating block: calculates energies
        E_new = target_molecule.getElectronicEnergy()
        E_total = target_molecule.getTotalEnergy()

        # generating block: generates new matrices
        F_n =  target_molecule.displayFockMatrix()
```

```
            target_molecule.setGuess(F_n)
            d_new = target_molecule.getDensityMatrix()

            # comparing block: "Are we there yet?"
            rms_D = np.einsum("pq->", np.sqrt((d_old - d_new)**2))
            if abs(E_old - E_new) < 1e-6 and rms_D < 1e-4:
                convergence = True


            # maintenance block: keeps everything going
            print(f"""iteration: {itercount}, E_tot: {E_total: .8f},
                        E_elek: {E_new: .8f},
                        deltaE: {E_new - E_old: .8f},
                        rmsD: {rms_D: .8f}""")
            E_old = E_new
            d_old = d_new
            E_list.append(E_new)
            itercount += 1

    return E_list
```

First we need to set up the parameters before we start iterating. That is done in the first block of code. Then we start a while loop, that will stop iterating once we have reached convergence, or when we have reached the maximum amount of iterations. Inside the loop, we see four blocks of code. The calculating block provides us with the energy of the molecule for the given `self.guessMatrix`. The next block will generate new matrices for the next iterative step. In the comparing block we check the conditions for convergence. For this we need four values, the old and new energies and the old and new distance matrices. For more information, see Subsection 8.2. For now we will continue to walk trough the code. After the comparing block we move to the final block, which sets us up for the next iteration. This iterations energy is stored, as well as the density matrix. We print out a line that summarises all the relevant values for this iteration. We can then move on to the next iterative step until the while loop finds one of its conditions is no longer met.

8.1 On Iterations and the Molecule Object

In this subsection we will discuss what the iteration actually does to the molecule object along the way. We call a lot of methods on the object, but those do not change any of the fundamental properties of the object. Now pay special attention to the generating block, where we change the `self.guessMatrix` to a new value. After this is done, all new values

that are calculated will be based on the new matrix. This is the only value that actually changes, but is has far reaching consequences on all the methods.

8.2 On Comparing Values

In Listing 6 we can see that there is a difference in the citeria for the density matrix and the energy. However when we look at the equation we see that the energy uses the density matrix. Indeed we see in Equation 20 that the energy is calculated using a product of the Fock matrix with the density matrix, which already uses the density matrix as can be seen in Equation 21.

$$F_{\mu\nu} = H_{\mu\nu} + \sum_{\lambda\sigma}^{AO} D_{\lambda\sigma}[(\mu\nu|\lambda\sigma) - \frac{1}{2}(\mu\lambda|\nu\sigma)] \tag{21}$$

Considering that the root-mean-square of the density matrix uses all elements of this matrix, we could say that the criterion for the density matrix is inherently the strictest, since the energy criterion only uses one single value and the density matrix criterion uses a multitude of values that have to behave accordingly. However, this expalanation is to simple. Since the energy uses the density matrix, these conditions are in fact connected. In this case that would mean that enforcing one condition is enough to enforce the other one. We can indeed see that this is the case in Section 9.

9 Some examples

In this section, we will display some of the results from our calculations for two example systems, water and methane. We will give a display of the output as generated by the functions we discussed above.

9.1 water

Listing 7: iterations for water

```
0, E_tot: −73.28579642, E_elek: −81.28816348, deltaE: −81.2881634, rmsD:  14.05298222
1, E_tot: −74.82812538, E_elek: −82.83049244, deltaE: −1.54232896, rmsD:  3.17285816
2, E_tot: −74.93548800, E_elek: −82.93785506, deltaE: −0.10736262, rmsD:  0.65858574
3, E_tot: −74.94147774, E_elek: −82.94384480, deltaE: −0.00598974, rmsD:  0.24093051
4, E_tot: −74.94197200, E_elek: −82.94433906, deltaE: −0.00049425, rmsD:  0.08612099
5, E_tot: −74.94205606, E_elek: −82.94442312, deltaE: −0.00008407, rmsD:  0.04061885
6, E_tot: −74.94207442, E_elek: −82.94444148, deltaE: −0.00001836, rmsD:  0.01827516
7, E_tot: −74.94207865, E_elek: −82.94444571, deltaE: −0.00000423, rmsD:  0.00892060
8, E_tot: −74.94207963, E_elek: −82.94444669, deltaE: −0.00000098, rmsD:  0.00425406
9, E_tot: −74.94207986, E_elek: −82.94444692, deltaE: −0.00000023, rmsD:  0.00205358
```

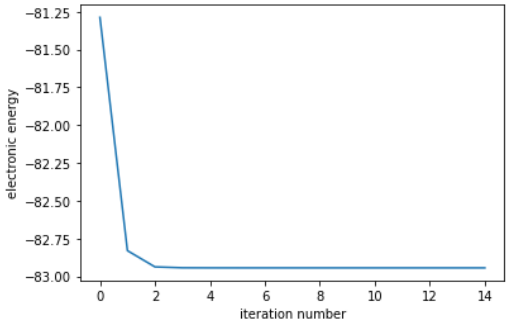

Figure 1: convergence for water

```
10, E_tot: −74.94207991, E_elek: −82.94444697, deltaE: −0.00000005, rmsD:  0.00098889
11, E_tot: −74.94207992, E_elek: −82.94444699, deltaE: −0.00000001, rmsD:  0.00047710
12, E_tot: −74.94207993, E_elek: −82.94444699, deltaE: −0.00000000, rmsD:  0.00023011
13, E_tot: −74.94207993, E_elek: −82.94444699, deltaE: −0.00000000, rmsD:  0.00011102
14, E_tot: −74.94207993, E_elek: −82.94444699, deltaE: −0.00000000, rmsD:  0.00005356
```

Here we see the the amount of iterations before the convergence is reached and all parameters calculated during that iteration. Furtermore we can use the `oeprop` method to get the dipole and nuclear charges. The dipole moment is given in eÅ, the charge is given in e, where e is the elemental charge.

total dipole moment	0.6034
nuclear charges	
O	-0.25302
H	0.12651
H	0.12651

Table 2: Some properties of water

9.2 methane

Listing 8: iterations for methane

```
0, E_tot: −36.08344857, E_elek: −49.58075304, deltaE: −49.58075304, rmsD:  35.9980315
1, E_tot: −39.56451342, E_elek: −53.06181788, deltaE: −3.48106485, rmsD:  4.73662689
2, E_tot: −39.72183632, E_elek: −53.21914079, deltaE: −0.15732290, rmsD:  0.79654919
3, E_tot: −39.72669300, E_elek: −53.22399746, deltaE: −0.00485667, rmsD:  0.14041648
4, E_tot: −39.72684535, E_elek: −53.22414982, deltaE: −0.00015236, rmsD:  0.02394664
5, E_tot: −39.72685016, E_elek: −53.22415462, deltaE: −0.00000480, rmsD:  0.00443984
6, E_tot: −39.72685031, E_elek: −53.22415477, deltaE: −0.00000015, rmsD:  0.00072904
7, E_tot: −39.72685032, E_elek: −53.22415478, deltaE: −0.00000001, rmsD:  0.00014205
8, E_tot: −39.72685032, E_elek: −53.22415478, deltaE: −0.00000000, rmsD:  0.00002652
```

total dipole moment	0.0000
nuclear charges	
C	-0.26031
H	0.06508
H	0.06508
H	0.06508
H	0.06508

Table 3: Some properties of methane