Han-sur-Lesse 2017
Course: Build your own *ab initio* electronic structure program
Ivo Filot <i.a.w.filot@tue.nl>

The SCF algorithm in Hartree-Fock calculations

Exercise 1: The hydrogen atom

The hydrogen atom is perhaps the simplest system for which we can calculate the electronic energy. The electronic energy can be calculated exactly for this system, which is E = -0.5 HT. In this exercise, you are going to calculate the energy for a hydrogen atom using the STO-3G and STO-6G basis function.

The coefficients for the STO-3G basis functions for the H-atom are as follows:

i	d	α
1	3.4252509	0.15432897
2	0.6239137	0.53532814
3	0.1688554	0.44463454

In hfcxx.cpp you can see that the source code for this Contracted Gaussian Function (CGF) is:

```
const vec3 pos1(0.0, 0.0, 0.0);     // set position
CGF cgf1(pos1);                     // construct cgf

// add gtos to the cgf
cgf1.add_gto(CGF::GTO_S, 3.4252509099999999,  0.15432897000000001, pos1);
cgf1.add_gto(CGF::GTO_S, 0.62391373000000006, 0.53532813999999995, pos1);
cgf1.add_gto(CGF::GTO_S, 0.16885539999999999, 0.44463454000000002, pos1);
```

Question 1.1: Show that the overlap integral for this CGF is unity (i.e. 1.0). Construct an *Integrator* object and execute the *overlap* function to calculate the value for the overlap integral.

```
double s = integrator.overlap(cgf1, cgf1);
```

Output the result using the std::cout functionality.

```
std::cout << s << std::endl;
```

Question 1.2: Calculate the values for the kinetic and nuclear attraction integrals. The corresponding functions are *kinetic* and *nuclear*. Note that the *nuclear* function requires a third and fourth parameter, which are the position of the nucleus and the charge.

```
v = integrator.nuclear(cgf1, cgf1, pos1, 1.0);
```

Question 1.3: Calculate the total electronic energy for the STO-3G basis function and output the result. It should be close to -0.5. Which type of integral does not have to be evaluated in this simple system?

Question 1.4: Do the same (i.e. calculate the overlap, kinetic and nuclear integrals and evaluate the total electronic energy) using the STO-6G basis function for H. The coefficients for this basis function are:

i	d	α
1	35.523221	0.009164
2	6.513144	0.049361
3	1.822143	0.168538
4	0.625955	0.370563
5	0.243077	0.416492
6	0.100112	0.130334

Note that the final electronic energy is closer to the exact solution (-0.5HT) than the final electronic energy using a STO-3G basis function. Provide a reasoning why.

Exercise 2: The Helium atom

The second most simple system is the helium atom. It can be modelled using a single basis function, but in contrast to the H atom case, we have to evaluate a two-electron integral here.

Question 2.1: Show that the overlap integral for the STO-3G integral for He is equal to 1.0.

Question 2.2: Evaluate the two-electron integral for He and show that the value is 1.05571 HT. Why is this value positive? Use the following code snippet for the two-electron integral.

```
double te = integrator.repulsion(cgf1, cgf1, cgf1, cgf1);
```

Question 2.3: Write out the Hamiltonian for He and show that the ground state energy is equal to

$$E = 2\langle\varphi|\hat{T}|\varphi\rangle + 2\langle\varphi|\hat{V}|\varphi\rangle + (\varphi\varphi|\varphi\varphi)$$

Calculate the ground state energy for He and show that its value is -2.80778 HT. The experimental value is −2.90338583(13) HT.[1]

Exercise 3: The integrals for the H_2 molecule

The next step to increase the complexity of our toy system is a simple molecule. The simplest molecule is the H_2 molecule, which has two nuclei and two electrons. Because we have two nuclei, the minimal basis set we must use is two STO-3G CGFs for each of the H atoms. The two CGFs are constructed and these objects are placed inside a *vector*. A *vector* is a handy computational object to store data. You can simply refer to one of its elements using an index. It is basically a one-dimensional matrix. Remember

[1] Kramida, A., Ralchenko, Yu., Reader, J., and NIST ASD Team. "NIST Atomic Spectra Database Ionization Energies Data". Gaithersburg, MD: NIST.

that in C++, these vectors are zero-based. That means that the first element has the index 0. In the code, you can see that we have placed the two CGFs in the vector using the following instructions.

```
std::vector<CGF> cgfs;
cgfs.push_back(cgf1);
cgfs.push_back(cgf2);
```

If we want to use *cgf1* anywhere in the code, we can simply refer to it by cgfs[0]. In this exercise, it will become clear how we can efficiently construct all data objects in the program by referring to them using indices.

Question 3.1: Why would a single CGF on either of the H atoms not suffice?

Question 3.2: We wish to store all the integrals in matrices. This is for computers the most efficient way to store the values. In the source code, the 2x2 matrices are already initialized for you.

```
// Construct 2x2 matrices to hold values for the overlap,
// kinetic and two nuclear integral values, respectively.
Eigen::Matrix2d S, T, V1, V2;
```

Your task is to set the right values in the right elements. For a 2x2 matrix, you could in principle write the following for the overlap matrix S:

```
S(0,0) = integrator.overlap(cgfs[0], cgfs[0]);
S(1,0) = integrator.overlap(cgfs[1], cgfs[0]);
S(0,1) = integrator.overlap(cgfs[0], cgfs[1]);
S(1,1) = integrator.overlap(cgfs[1], cgfs[1]);
```

Yet we want to program it a bit more systematic and use a *for loop*. This (nested) *for* loop has already been constructed for you as well as the allocation for the overlap matrix elements. Put after this line the corresponding instructions to allocate the elements for the nuclear attraction (there are <u>two</u> of these!) and kinetic energy matrices. Show that you obtain the following values for the S, T, V_1 and V_2 matrices.

$$\mathbf{S} = \begin{pmatrix} 1 & 0.659318 \\ 0.659318 & 1 \end{pmatrix} \quad \mathbf{T} = \begin{pmatrix} 0.760032 & 0.236455 \\ 0.236455 & 0.760032 \end{pmatrix}$$

$$\mathbf{V}_1 = \begin{pmatrix} -1.22661 & -0.597417 \\ -0.597417 & -0.653827 \end{pmatrix} \qquad \mathbf{V}_2 = \begin{pmatrix} -0.653827 & -0.597417 \\ -0.597417 & -1.22661 \end{pmatrix}$$

What kind of symmetries do you note for the values in these matrices? Can you rationalize these symmetries?

Question 3.3: Next, we wish to evaluate all the two-electron integrals. A lot of these integrals have the same values and hence it is not very efficient to calculate all N^4 of these integrals. As such, we use a relatively complicated set of nested for loops to avoid some of the double counting as given below.

```
for(unsigned int i=0; i<2; i++) {
       for(unsigned int j=0; j<2; j++) {
           unsigned int ij = i*(i+1)/2 + j;
           for(unsigned int k=0; k<2; k++) {
               for(unsigned int l=0; l<2; l++) {
                   unsigned int kl = k * (k+1)/2 + l;
                   if(ij <= kl) {
```

The number of unique two-electron integrals (not taking any symmetries of the molecule into account) is

$$n = \frac{1}{8} n\left(n+1\right)\left(n^2+n+2\right)$$

Despite the complicated set of nested for loops, we still have some double counting which we can avoid. In our program, we adopt an indexing method to keep track of all integrals. Given four indices where each index identifies a CGF, we can generate a unique index for the two-electron integral as follows

```
unsigned int idx = integrator.teindex(i,j,k,l);
```

Using this index, we can check whether a value has been calculated for this index. If this is not the case, we evaluate the two-electron integral for the four CGFs under consideration. Write the line of code that evaluates the two-electron integral.

Question 3.4: Using the above formula for the number of integrals, we can see that there should be six unique two-electron integrals. Careful inspection shows that there are only four distinct values. Which two-electron integrals have a different index, but the same value? (if you find this question too difficult or don't like this kind of a math puzzle, just skip it)

Exercise 4: Constructing the transformation matrix

In this question, we will explore how to construct the transformation matrix X and explore the properties of orthonormal and unitary matrices. Furthermore, we explore the matrix diagonalization procedure to obtain eigenvalues and eigenvectors.

Question 4.1: In the source code, the overlap matrix S is diagonalized, resulting in a set of eigenvalues and eigenvectors. Show that the matrix is a diagonal matrix (i.e. that it has only nonzero values on its diagonal) and that it contains the eigenvalues of S with corresponding eigenvalues as given in the unitary matrix U. In other words, show that the equality below holds:

$$\mathbf{S} = \mathbf{U}\mathbf{D}\mathbf{U}^{\dagger}$$

Question 4.2: Show that the matrix U is unitary by showing that the following equality holds:

$$\mathbf{U}^{-1} = \mathbf{U}^{\dagger}$$

Question 4.3: Construct the transformation matrix X using canonical orthogonalization:

$$\mathbf{X} = \mathbf{U}\mathbf{D}^{-1/2}$$

where

$$\mathbf{D}^{-1/2} = \begin{pmatrix} \frac{1}{\sqrt{D_{1,1}}} & 0 \\ 0 & \frac{1}{\sqrt{D_{2,2}}} \end{pmatrix}$$

Show that

$$\mathbf{X} = \begin{pmatrix} 0.548934 & -1.21146 \\ 0.548934 & 1.21146 \end{pmatrix}$$

Question 4.4: Using the transformation matrix X, we can obtain an orthonormalized overlap matrix by the following formula

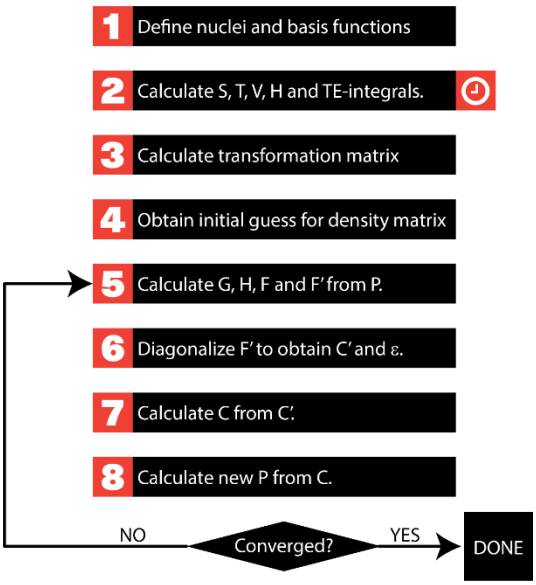
$$\mathbf{S}^{\Diamond} = \mathbf{X}^{\dagger}\mathbf{S}\mathbf{X}$$

Show that $\mathbf{S}$ is **not** orthonormal and that $\mathbf{S}^{\Diamond}$ is orthonormal. Remember that for orthonormal matrices, the following properties hold:

$$\sum_a M_{ai} \cdot M_{aj} = \sum_a M_{ia} \cdot M_{ja} = \delta_{ij} \text{ and}$$

$$\mathbf{M}^{\dagger}\mathbf{M} = \mathbf{M}\mathbf{M}^{\dagger} = \mathbf{1}$$

Exercise 5: Self-consistent field calculation for the H_2 molecule



In this exercise, we are going to explore the SCF algorithm. We use H_2 as our test system.

Question 5.1: Open the source code and explore the algorithm. Compare all the steps with the scheme as shown here on the left. Add comments to the script to show where each of the steps on the left hand side start. (in other words, you are here providing additional documentation to the script)

Question 5.2: Calculate the (converged) energies for the molecular orbitals and show that these are equal to

$$\varepsilon_1 = -0.578208$$
$$\varepsilon_2 = +0.670259$$
$$\mathrm{E}_{\mathrm{total}} = -1.11671$$

Remember: the energies for the MOs are equal to the eigenvalues of the Fock-matrix.

Question 5.3: Show that the corresponding eigenvectors (coefficient vectors) are:

$$\vec{c}_1 = \left(0.548934, 0.548934\right)^{\dagger}$$
$$\vec{c}_2 = \left(-1.21146, 1.21146\right)^{\dagger}$$

Remember that these are the columns in the C matrix.

Bonus subquestion: Can you identify the *gerade* and *ungerade* MOs?

Question 5.4: Build an MO diagram based on this solution.

Question 5.5: The STO-3G basis set is a rather tiny basis set. Let us use the somewhat bigger (but still relatively small) 3-21G basis set. This basis set uses two CGFs for each H atom. The coefficients are as follows:

$$\left|\varphi_{1S}\right\rangle$$

i	d	α
1	5.447178	0.156285
2	0.824547	0.904691

$$\left|\varphi_{1S}{}'\right\rangle$$

i	d	α
1	0.183192	1.000000

Show that:

- The basis functions in these basis set are normalized (yet not orthogonal with respect to each other).
- The energy for the HOMO and LUMO are -0.592368 HT and 0.263944 HT, respectively.
- The total energy is -1.12293. (bonus question: Why is this more accurate?)

Bonus question

Modify the solution of Exercise 5 to calculate the CO molecule (with a bond distance of 2.116 a.u.). Do not forget to change the charges and the number of electrons in the algorithm!

Use the following coefficients for the STO-3G basis set of C and O:

```
std::vector<CGF> cgfs;
// construct the CGFs for CO
// C
const vec3 pos1(0.0, 0.0, 0.0);
// 1S
cgfs.emplace_back(pos1);
cgfs.back().add_gto(CGF::GTO_S, 71.616837,  0.154329, pos1);
cgfs.back().add_gto(CGF::GTO_S, 13.045096, 0.535328, pos1);
cgfs.back().add_gto(CGF::GTO_S, 3.530512, 0.444635, pos1);
// 2S
cgfs.emplace_back(pos1);
cgfs.back().add_gto(CGF::GTO_S, 2.941249,  -0.099967, pos1);
cgfs.back().add_gto(CGF::GTO_S, 0.683483, 0.399513, pos1);
cgfs.back().add_gto(CGF::GTO_S, 0.222290, 0.700115, pos1);
// 2Px
cgfs.emplace_back(pos1);
cgfs.back().add_gto(CGF::GTO_PX, 2.941249,  0.155916, pos1);
cgfs.back().add_gto(CGF::GTO_PX, 0.683483, 0.607684, pos1);
cgfs.back().add_gto(CGF::GTO_PX, 0.222290, 0.391957, pos1);
// 2Py
cgfs.emplace_back(pos1);
cgfs.back().add_gto(CGF::GTO_PY, 2.941249,  0.155916, pos1);
cgfs.back().add_gto(CGF::GTO_PY, 0.683483, 0.607684, pos1);
cgfs.back().add_gto(CGF::GTO_PY, 0.222290, 0.391957, pos1);
// 2Pz
cgfs.emplace_back(pos1);
cgfs.back().add_gto(CGF::GTO_PZ, 2.941249,  0.155916, pos1);
cgfs.back().add_gto(CGF::GTO_PZ, 0.683483, 0.607684, pos1);
cgfs.back().add_gto(CGF::GTO_PZ, 0.222290, 0.391957, pos1);

// O
const vec3 pos2(0.0, 0.0, 2.116);
// 1S
cgfs.emplace_back(pos2);
cgfs.back().add_gto(CGF::GTO_S, 130.709320,  0.154329, pos2);
cgfs.back().add_gto(CGF::GTO_S, 23.808861, 0.535328, pos2);
```

```
cgfs.back().add_gto(CGF::GTO_S, 6.443608, 0.444635, pos2);
// 2S
cgfs.emplace_back(pos2);
cgfs.back().add_gto(CGF::GTO_S, 5.033151,  -0.099967, pos2);
cgfs.back().add_gto(CGF::GTO_S, 1.169596, 0.399513, pos2);
cgfs.back().add_gto(CGF::GTO_S, 0.380389, 0.700115, pos2);
// 2Px
cgfs.emplace_back(pos2);
cgfs.back().add_gto(CGF::GTO_PX, 5.033151,  0.155916, pos2);
cgfs.back().add_gto(CGF::GTO_PX, 1.169596, 0.607684, pos2);
cgfs.back().add_gto(CGF::GTO_PX, 0.380389, 0.391957, pos2);
// 2Py
cgfs.emplace_back(pos2);
cgfs.back().add_gto(CGF::GTO_PY, 5.033151,  0.155916, pos2);
cgfs.back().add_gto(CGF::GTO_PY, 1.169596, 0.607684, pos2);
cgfs.back().add_gto(CGF::GTO_PY, 0.380389, 0.391957, pos2);
// 2Pz
cgfs.emplace_back(pos2);
cgfs.back().add_gto(CGF::GTO_PZ, 5.033151,  0.155916, pos2);
cgfs.back().add_gto(CGF::GTO_PZ, 1.169596, 0.607684, pos2);
cgfs.back().add_gto(CGF::GTO_PZ, 0.380389, 0.391957, pos2);
```

Show that the energy is -111.223 HT and create an MO diagram using the coefficient matrix C.