Module code - Lecture 21
Week 8

Scribe(s): Emile
Instructor: M. Melgaard

Construction of an Electronic Structure Program

Intro Programming for Electronic Structure
The objectives are to :

- Understand what happens in a real electronic structure calculation, especially when working on a more complex molecule (e.g. H_2)
- Give a general idea or impression of how some of these calculations can be programmed (It is easy to consult a book and look up the formulas, but it is not quite obvious especially to the chemist and who has never seen programming to implement them in a program).

Why Matlab?

- MATLAB is free for many universities
- Very easy to implement numerical functions.

All we are going to do is

- A direct implementation of the formula found in a book on electronic structure calculations or electronic structure theory.
- Use some common programming elements, such as functions, arrays, sub-program, structures and control loops, which can be applied to future projects.

Statement of Problem
It turns out that we can write the Hamiltonian of a molecule in a relatively simple way. We know all the interactions.

The problem is that the electron repulsion term makes it very difficult to solve the eigenvalues of the Hamiltonian in a realistic sense, as on paper. So what we do is we simplify the problem.

It is simplified in two ways.

- Hatree-Fock approximation, i.e. instead of dealing with the full electron repulsion, we see for each electron that it feels an average electron repulsion or average field energy due to the presence of the other electrons. This means that we lose the correlation information, so that one electron cannot actually avoid another. It moves around the mean potential. So we lose some energy or accuracy, which can be taken into account later.
- The other thing we do is that rather than trying to integrate the Schrodinger equation and get the exact solution. We will solve the Schrodinger equation and the molecular Hamiltonian using a finite basis on that basis.

  So it's going to consist of Gaussian functions. Maybe they're multiplied by polynomials that are centred on each atom and these combinations these Gaussian functions are going to be what we use to create our actual independent individual electronic orbitals.

And because we solve it on a finite basis. This means that we will evaluate a number of operators and build them explicitly in our database.

So if we use 20 basis functions, each of our operators will be a 20 by 20 matrix where each element of the matrix, most of them, will be the global integral space of the basis.

$$O_{mn} = \langle m|O|n\rangle = \int \mathrm{d}\tau \phi_m O \phi_n$$

O: Operator acts on the base.

We will be mainly concerned with the operators that appear in our Hamiltonian ($\hat{H}$). And our Hamiltonian ($\hat{H}$) can be divided into four basic operators.

- The first is the nuclear repulsion energy (V_{NM}). It does not depend on the electrons at all. We'll assume that the nuclear ones are fixed in space, which is basically the Born- Oppenheimer approximation. And so it's a very simple sum. If you tell the program where the nuclei are and what the atomic number of each nucleus is, you can calculate it very quickly.
- There are the one-electron operators which will be the kinetic energy of the electrons ($\hat{T_e}$) in the nuclear attraction energy of the electrons (V_{eN}). These operations are relatively simple to solve, to evaluate these matrix elements.
- And the last and most important one is the electron repulsion matrix (V_{ee}). And to calculate it, we will build a kind of help matrix, which will be a four-dimensional array. It will be number $\phi_i(1)\phi_j(1)\dfrac{1}{r_{12}}\phi_k(2)\phi_l(2)$.

  And then we will select elements from this matrix to create our actual electronic repulsion.

Strategy:
As you might expect, this will involve a substantial amount of code. And so we're going to do it in three steps.

- The first step is to write a program capable of calculating the different elements of the matrix that we will need, and we will limit ourselves to the use of s orbitals. The reason we use these orbitals is that we need a computational program and the mathematics and programming is simple. We don't have to worry about the angular momentum components and we get all the basic physics anyway.

  But once we have our matrix elements, we're going to write another program or another sub-program that's going to take those matrix elements. The total number of electrons in which our basic functions will do the self-consistent field method to find the lowest energy states, and then we'll populate those two electrons per orbital. We'll add them up to get our final total energy.

  Now, it turns out that once you build this program, this program doesn't care about your momentum. All the information about angular momentum is in the matrix elements. And so once we know that we've built this sub-program can calculate things for us. Orbital will expand the program and look at the matrix on this program to include different angular momentum values.

Build HF Matrix Elements
So the general structure for the actual program is that we're going to break it down into a variety of smaller programs. We're not going to try to write it all in one giant document where MATLAB goes through and executes each function one by one. Instead, we're going to have one big master program, which we'll call our driver, that will drive the program. And the only thing we're going to give it is our atomic numbers, the locations in the nuclei, and then we're going to use a specific basis set, which will depend on the details of what will depend on the atomic numbers.

This large driver program will call smaller programs that will be stored in separate files. And it's these smaller programs that will calculate the elements of the matrix. And in order to calculate the elements of the matrix, we're going to need to know how to do these integrals. And we're going to have another set of very small programs that do very specific things that we're going to use over and over again in the different matrix element filters.

HF_Driver.m
And so for our pilot, we're going to go ahead and build it now. And then in the next few videos, we'll create the programs that will help us build our base sets and calculate the actual matrix elements for each of them.

The only thing we need is a list of atomic numbers. So if I have a methane, we need to tell it that we have one carbon atom and four hydrogen atoms. So it could be a list of 41111. Now I have to say what the coordinates of each of the individual nuclei are. So it's going to be the valence number of atoms by three tables. And once we have that, we'll just declare a number of functions that we'll call up later.

So our pilot program will need to build a set of basic functions, based on the nuclei we give it from nuclear.

It has to calculate the nuclear repulsion energy. And then we have to start dealing with the electrons. Given our basis set, these are Gaussian functions that are going to be normal. And so we have to calculate how much each basis function overlaps with all the other basis functions. Then we calculate our one-electron operator.

The kinetic energy matrix for the basis set is calculated and then the nuclear attraction energy is calculated, i.e. how much each electron and basis function would be attracted to each of the nuclei.

And then we will do the most difficult part, really, which is to build the electron repulsion matrix.

And once we've got all that, we'll feed it into another program that will do the self-consistent field method and return a number. This number will be the final energy.

MATLAB
Open MATLAB and we'll start building.
Open an **HF_driver.m** file.

Next, we will start building our program.

```
Function Min_Energy=HF_Driver(Z.AL)
```

This will be a function that returns a minimum energy, and we will have to give it our list of atomic numbers ($Z.AL$) in our atomic coordinates.

Once we've given it that, then we'll start building each of these individual sub-programs.

You can actually just from the nuclear atomic number and the coordinates. You can construct a nuclear repulsion.

```
Nuclear_Repulsion=Build_Nuclear_Repulsion(Z.AL)
```

And it will be a function that takes the atomic number.
Let's move on to the next function we will need: we need to build our base asset.

```
[basis,N]=Build_Basis(Z,AL)
```

We will get several values back. Our atomic numbers (Z) and coordinates (AL) will now return a cell structure of basic functions and define what this represents in terms of total number of electrons (N) in our system.

Once we have this, we can start to build our matrices which we will need to calculate our total energy. So we'll say that S equals the overlap below. This is the common linear use for an overlap matrix and all we need to give it all our basic functions.

```
S=Build_Overlap(basis) ;
```

The same applies to kinetic energy.

```
T=Build_Kinetic(basis) ;
```

We only need to give it a basic function.

And then we need the nuclear attraction can be our matrix or the nuclear attraction of the electrons, build the nuclear attraction and we're going to have to give it our atomic numbers (Z), our common coordinates (AL), and then our basis functions.

```
Nuclear_Attraction=Build_Nuclear_Attraction(Z,AL,basis)
```

And then finally, we have to build our four-dimensional lattice and the electron-electron repulsion. And so we go $Gabcd$, where the new abc's refer to the individual orbitals.