A Brief Introduction to Computational Chemistry with FHIaims

Adrian F. Rumson
Authored May, 2025

N.B.) This tutorial assumes a basic working knowledge of the Linux terminal. Although computational chemistry is not synonymous with the command line, the latter is an critically important skill. The Johnson Group has previously produced a manual for basic Linux commands, and ACENET offers training sessions on shell scripting and commands. Google and ChatGPT are very helpful resources, but remember that they are not substitutes for knowledge; only supplements. I also recommend having `critic2` installed on your system. Linux or linux-like command line interfaces can be found natively on Linux and Mac computers. On Windows, you can use software like MobaXTerm to emulate the terminal, or you can install the Windows Sub-system for Linux (WSL). The content of this tutorial is subject to change as further developments are made in FHIaims.

Input files can be found in the following repository:
`https://github.com/arumson0/tutorials`

1 Introductory Material

1.1 What is FHIaims?

The Fritz Haber Institute *ab initio* materials simulations (FHIaims or FHI-aims) package is a software toolkit used for computational chemistry. Unlike programs such as Gaussian or Quantum Espresso, which specialize in molecular and solid-state computations, respectively, FHIaims is capable of both. This is accomplished using "numerical atom-centered orbital" (NAO) basis sets which scale linearly in system size ($\mathcal{O}(N)$). If you're just getting started, don't worry about what exactly that means. What you need to know is that FHIaims is relatively fast compared to other popular electronic structure codes. In this tutorial, you will be introduced to some basic chemical systems that will emphasize different aspects of FHIaims.

1.2 What do I need to run a calculation?

Most of the time, we run FHIaims calculations on a supercomputer cluster, such as those operated by ACENET like siku and argo, or the ones operated by the Digital Research Alliance of Canada (DRAC). This tutorial will therefore emphasize how to run FHIaims calculations a "jobs" on a cluster using slurm.

An FHIaims job on a cluster requires three files: `geometry.in`, `control.in`, and a slurm submission script. `geometry.in` is the easiest to understand. It is simply a file containing a list of atoms, and their positions. If you are studying a periodic (or solid-state) system, it may also include a list of three "lattice vectors." We'll come back to these. `control.in` contains the specifications for how the calculation is to be run. This includes the exchange-corrlation treatment, spin, charge, basis set, *et cetera*. Finally, the slurm submission script adds your calculation to a queue of jobs on the cluster. When your job is selected, the node will execute the shell commands included in your submission script. This will include the call to FHIaims, but may also include other post-processing measures.

2 Molecular Systems

2.1 Dihydrogren

One of the simplest chemical systems is dihydrogen, or molecular hydrogen. The `geometry.in` file for this molecule might look like:

```
atom 0.00 0.00 -0.5 H
atom 0.00 0.00  0.5 H
```

As you might guess, the first entry in each line declares that this line will define the position of an atom. The next three entries are the xyz coordinates for the position of the atom in Ångstroms. Note that this does not inherently allow us to declare bond order. Most electronic structure codes are "smart" enough to figure out what the order of a bond should be from the surrounding chemical environment and predict the electronic structure accordingly.

Now we have to set up our `control.in` file. These files tend to be complicated, so it's best to start with a template that you store somewhere memorable (like your home directory). Let's walk through everything that we need in the `control.in` file. First, we must choose a functional (this is the F in DFT). Knowing what functional to use based on what system you're studying is a fine art. `B3LYP` is a very popular choice for treatment of the exchange and correlation effects in molecular systems. The first line of `control.in` should be:

```
xc b3lyp
```

Next, is our system magnetic or spin polarized? No. Dihyrdogen is diamagnetic (all electrons are paired). So the next line is:

```
spin none
```

The other option is `spin collinear`, where the electronic spin-magnetic moment is aligned with the z-axis. That's useful in molecular radicals, or diradicals like O_2. Next, is our system charged? No. It's neutral. So:

```
charge 0
```

Awesome! Those are the tags in `control.in` that you'll need to change the most when using FHIaims to study molecules. You will also need the following lines, but these can stay constant in your template. We also need the line:

```
output_level MD_light
```

to control the amount of information printed to the output file (`output_level full` prints an outrageously excessive amount of text). We must also ask if our system will experience relativistic effects. These are especially pronounced in heavy elements, which hydrogen is not. Therefore, include the line:

```
relativistic none
```

If we were to have heavier elements, we would use the line: `relativistic atomic_zora scalar` to add a simple yet effective relativistic correction.

Finally, we have to declare the basis set for hydrogen. This has already been done for you! Simply run this command:

```
cat ~/projects/def-ejohnson/FHIaims/FHIaims_240920_1/species_defaults/defaults_next/dense/lightdenser/01_H_default \
 >> control.in
```

This prints the entirety of the contents of the file 01_H_default to your control.in file. Rememeber that single `>` after a command means "write the output of this command to a file and erase anything that's already there" and double `>>` after a command means "append the output of this command to a the end of a file, which may or may not already exist."

Once you have a slurm submission script, you're ready to go! Here's a simple one to get you started:

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --tasks-per-node=8
#SBATCH --mem=16GB
#SBATCH --time=1:00:00
#SBATCH --job-name=dihydrogen
#SBATCH --account=def-ejohnson
```

```
module purge
module load StdEnv/2020
module load intel/2020.1.217 intelmpi/2019.7.217 imkl/2020.1.217 libxc/5.1.3

export OMP_NUM_THREADS=1
export MKL_NUM_THREADS=1
export MKL_DYNAMIC=FALSE
ulimit -s unlimited

mpirun ~/projects/def-ejohnson/FHIaims/FHIaims_240920_1/build/aims.240920_1.scalapack.mpi.x </dev/null > h2.out
```

To add this to the queue on a cluster, run the command: `sbatch FHIaims.sub` (or whatever you've named your submission script). To check your queue, run `sq`, and to see the full queue on the cluster, run `squeue`. You can tell that the job is complete when the output file concludes with `Have a nice day.` This is a convenient grep target to test whether a job is done or not.

BASH CHALLENGE
The final electronic energy (eV) of a calculation in FHIaims is associated with the string "Total energy uncorrected"
Use the `grep` and `awk` commands to print the final energy value *in Hartree* (1 Hartree = 27.211 eV). HINT: you may need a "pipe" to combine `grep` and `awk`.

Congratulations! You have now run what's known as a "single point" calculation on H_2.

2.2 Geometry Optimization of m-Fluorophenol

While single point calculations can be very useful, they are limited in their accuracy by the input structure. Fortunately, electronic structure codes can perform geometry optimizations (also known as geometry relaxations) to predict the minimum energy structure *for a specific functional*. I emphasize the functional because no functional is perfect, and can't exactly predict the structure of a molecule or crystal, but it can get really close if you choose your functional wisely.

Let's consider a molecule more complicated than H_2: m-fluorophenol. Here's a possible input structure:

```
atom        0.05357        2.31201        0.00000    O
atom        0.00000        0.93992        0.00000    C
atom        1.21892        0.25280        0.00000    C
atom        1.22134       -1.14204        0.00000    C
atom        0.01953       -1.85995        0.00000    C
atom       -1.19176       -1.16435        0.00000    C
atom       -1.20775        0.23364        0.00000    C
atom        2.14430        0.82112        0.00000    H
atom        2.17064       -1.67189        0.00000    F
atom        0.02861       -2.94620        0.00000    H
atom       -2.13385       -1.70687        0.00000    H
atom       -2.15487        0.77114        0.00000    H
atom       -0.84515        2.67654        0.00000    H
```

Now, as the author of this tutorial, I *know* that this is wrong. I know that because I simply went to Google, looked up "phenol xyz", copied the result, and replaced the meta-hydrogen with a fluorine. I didn't change the C-F bond distance, so it's too small. If I ran a single point on this structure, I'd get an energy that's too high because of the shrunken C-F bond distance. So, to predict the correct geometry, we can run an optimization by adding the following tag to our `control.in` file:

```
relax_geometry bfgs 0.01
```

The number at the end of the line defines the maximum forces allowed on atoms. In this case, we've chosen 0.01 eV/Åas our forces threshold.

We should also think about what functional would be appropriate here. We have a benzene ring, which features delocalization, so we should pick a functional equipped to handle that. Again, `B3LYP` will serve us well. This is also a good opportunity to discuss dispersion. Dispersion is the weakest of the forces that act on molecules, and it arises

from the interaction of instantaneous dipole moments. Although it is small, it can unilaterally govern the interactions in some chemical systems (we will see one later), and can have a noticeable effect on others. For most functionals, a post-SCF dispersion correction is needed to fully capture this effect. Erin's PhD was centered around the development of such a correction that was driven by robust physics–this is the exchange-hole dipole moment (XDM) model. It is available in several electronic structure codes, including FHIaims. We can add the XDM dispersion correction to our calculation with the tag:

```
xdm lightdenser
```

The `lightdenser` keyword defines the basis set used in the calculation. XDM has two parameters that differ depending on the choice of FHIaims basis set. Speaking of which, there are more atomic species present in our system, so we have to add 06_C_default, 08_O_default, and 09_F_default to our control.in file. `cat` those to `control.in` like we did for H.

Our control block should now look like:

```
xc b3lyp
spin none
charge 0
output_level MD_light
relativistic atomic_zora scalar
xdm lightdenser

relax_geometry bfgs 0.01

<BASIS SET STUFF>
```

Reuse the submission script from 2.1 and submit this job to the cluster; it should be fairly fast. If you `ls` the contents of the directory that you're running your job in, you may see a `geometry.in.next_step` file. The structure contained within corresponds to the most recent structure in the output file. If a geometry relaxation calculation does not change the given geometry, no `geometry.in.next_step` will be written.

Over the course of the relaxation, several intermediate geometries will be predicted, along with their electronic energies. This means that grep'ing for "Total energy uncorrected" will give multiple results. The last one in the list is the energy of the final predicted geometry, which is the value you are most often interested in.

Using what you learned in the first BASH CHALLENGE, make a plot of the total energy over the course of the relaxation. The x-axis of your plot should simply be the number of relaxation steps, and the y-axis should be plotted relative to the final energy.

BASH CHALLENGE

Run the command you created in the first BASH CHALLENGE on your relaxation output file. You will see multiple energies printed to the terminal. Add another pipe with the `tail` command to only print the last line. HINT: The `tail` command alone won't be enough. You need an option.

Congratulations! You have successfully optimized m-Fluorophenol!

3 Crystalline Systems

3.1 Unit Cells

Molecules are awesome! But they are not the only frontier of computational chemistry. The other is crystalline systems. The property that separates the two is *periodicity*; simply put, molecules can be treated in isolation, whereas crystals repeat in space nigh infinitely. That sounds intimidating, but it can work in our favour! We just have to find the smallest repeating unit, and apply some cool math. That smallest repeating unit is called a "unit cell" and it's what we do calculations on in solid-state electronic structure.

In 3 dimensions, the unit cell is some parallelepiped that can be defined by 3 "lattice vectors," often labelled: $\mathbf{a}_1$, $\mathbf{a}_2$, and $\mathbf{a}_3$. In a simple-cubic crystal system (like rock salt), the three lattice vectors are orthogonal, and have the same length. Molecular crystals are rarely this simple. Each lattice vector has three cartesian components, so with three lattice vectors, a unit cell is defined by a 3×3 matrix of cartesian lengths. In FHIaims, the unit cell is set at the top of the `geometry.in` file. For the example of rock salt sodium chloride, the first three lines of your geometry file would be:

```
lattice_vector 5.6402 0.0000 0.0000
lattice_vector 0.0000 5.6402 0.0000
latiice_vector 0.0000 0.0000 5.6402
```

Like with atomic positions, these distances are measured in Ångstroms.

One more thing about the `geometry.in` file. Frequently, instead of absolute cartesian coordinates, atomic positions in a crystal are represented with "fractional coordinates." This puts the position of an atom in terms of the lattice vectors, which is a more cohesive system that originates from the vast field of crystallographic symmetry. That said, it is a bit harder to read at a glance. For a concrete example, consider an atom that appears in the exact center of a unit-cell. Its fractional coordinates are (0.5, 0.5, 0.5). You can see the upsides and downside of this system. On the one hand, those fractional coordinates are always the center of the unit cell, no matter the lattice vectors. On the other hand, if I told you to move that atom 1 Ångstrom along the x-direction, that could get pretty tricky. Fractional coordinates are denoted in `geometry.in` with `atom_frac`.

```
lattice_vector 3.359 0.000 0.000
lattice_vector 0.000 3.359 0.000
lattice_vector 0.000 0.000 3.359
atom_frac 0.0 0.0 0.0 Po
```

When you run a periodic calculation in FHIaims, you also have to include 1 tag in your `control.in` file. That is the `k_grid` tag. The exact mathematical motivation of a k-grid is REALLY COMPLICATED and beyond the scope of this tutorial. However, the essence is this: if you have a periodic system, every unit cell will have neighbours. Those neighbours can effectively reach in and influence what's going on in the unit cell. You can account for this in two ways: (1) replicated the unit cell into a supercell, or (2) use a k-grid. Both work, but the latter is much more computationally efficient.

Choosing a k-grid is less artistic than choosing a functional. `critic2` has a command (`kpoints`) that will suggest an appropriate one based on the size and shape of your crystal. For the example of simple cubic polonium that I've provided above:

```
critic2:3> kpoints
%% kpoints

* KPOINTS: calculate dimensions of uniform k-point grid.

# ---  Rk  ---   -- kpts --
  0.1 ->   9.5    1  1  1
  9.6 ->  15.8    2  2  2
 15.9 ->  22.2    3  3  3
 22.3 ->  28.5    4  4  4
 28.6 ->  34.9    5  5  5
 35.0 ->  41.2    6  6  6
 41.3 ->  47.6    7  7  7
 47.7 ->  53.9    8  8  8
 54.0 ->  60.3    9  9  9
 60.4 ->  66.6    10 10 10
 66.7 ->  72.9    11 11 11
 73.0 ->  79.3    12 12 12
 79.4 ->  85.6    13 13 13
 85.7 ->  92.0    14 14 14
 92.1 ->  98.3    15 15 15
 98.4 -> 100.0    16 16 16
```

R_k is a single parameter that describes the density of a k-grid. For molecular crystals, the grid corresponding to $R_k = 50$ is sufficient. For metallic systems, or those featuring inhomogeneity, go larger (e.g. $R_k = 75$). You want your k-grid to be sufficiently dense. A more robust method of selection is to test each suggested k-grid until some property (the total energy for example) stops changing. This method is known as "converging" your k-grid. When you have chosen your k-grid, enter it into your `control.in` with the syntax:

```
k_grid 12 12 12
```

You can also use `critic2` to help you write a `control.in` file during a `write` command. For example:

```
write geometry.in 50
```

Will produce a template `control.in` with a k-grid corresponding to $R_k = 50$.

3.2 Single Point on Crystalline Urea

With all that out of the way, here's a simple task. Provided is the geometry file for crystalline urea. Use FHIaims to run a single point on the structure (just as you did for H_2). Use the k-grid suggested by `critic2` corresponding to $R_k = 65$, and make sure you have all the appropriate basis sets in your `control.in` file. Instead of `B3LYP`, use `B86bPBE`, which performs better for solids and make sure you have XDM turned on.

```
lattice_vector 5.5650000000       0.0000000000       0.0000000000
lattice_vector 0.0000000000       5.5650000000       0.0000000000
lattice_vector 0.0000000000       0.0000000000       4.6840000000
atom_frac  0.0000000000       0.5000000000       0.3260000000       C
atom_frac  0.5000000000       0.0000000000       0.6740000000       C
atom_frac  0.0000000000       0.5000000000       0.5953000000       O
atom_frac  0.5000000000       0.0000000000       0.4047000000       O
atom_frac  0.1459000000       0.6459000000       0.1766000000       N
atom_frac  0.3541000000       0.1459000000       0.8234000000       N
atom_frac  0.8541000000       0.3541000000       0.1766000000       N
atom_frac  0.6459000000       0.8541000000       0.8234000000       N
atom_frac  0.2575000000       0.7575000000       0.2827000000       H
atom_frac  0.2425000000       0.2575000000       0.7173000000       H
atom_frac  0.7425000000       0.2425000000       0.2827000000       H
atom_frac  0.7575000000       0.7425000000       0.7173000000       H
atom_frac  0.1441000000       0.6441000000       0.9620000000       H
atom_frac  0.3559000000       0.1441000000       0.0380000000       H
atom_frac  0.8559000000       0.3559000000       0.9620000000       H
atom_frac  0.6441000000       0.8559000000       0.0380000000       H
```

3.3 Variable-Cell Relaxation on Crystalline Urea

In Section 2, we explored the geometry relaxation of a molecule. The `control.in` file for relaxing a crystal is much the same, just differing by the addition of a k_grid line in the crystal case. The snag is that that case is what's known as a "fixed-cell relaxation." There's another possibility: the variable-cell relaxation. Here, not only are the atomic positions allowed to change, but also the lattice vectors. This can be accessed in FHIaims by the addition of the keyword:

```
relax_unit_cell full
```

There is also the `relax_unit_cell fixed_angles` keyword, which allows the lengths of the lattice vectors to change, but not the angles between them. Relaxations of crystals requires a bit more fine tuning than isolated molecules due to the flatter potential energy surface (PES). For molecular crystals like urea, your geometry optimization tags may look like:

```
relax_geometry bfgs 0.005
sc_accuracy_forces 1e-4
sc_accuracy_etot 1e-6
```

Note the smaller threshold on the `relax_geometry` tag, and the addition of the `sc_accuracy` tags. Including the latter has a debatable impact on the accuracy of your calculation, but they can definitely help where the PES is very flat. For this reason, I always include them.

Starting from the geometry file provided in 3.2, perform a variable-cell relaxation of crystalline urea. Use the same k-grid as you did in 3.2.

Atomic positions are optimized according to the *forces* between atoms. A unit cell is relaxed by minimizing the *stress* acting on it. If you're not familiar with the stress from mechanics, don't worry! It's akin to pressure, which you probably are familiar with. They even have the same units.

This is a good point to address that there is a known issue with XDM related to forces and stresses. This bug can be sidestepped by repeatedly resubmitting the calculation, starting from the final geometry of the last run, until

the `geometry.in.next_step` file ceases to be written. This usually takes only one or two resubmissions. When you're comfortable with this process, I can provide you with a submission script that does this automatically.

Like with an atomic relaxation, several intermediate structures will be printed in your output file. Each of these states the stress acting on that structure, which is associated with the string: "Pressure" in the output file. Create a plot with the stress on the x-axis and the relaxation step on the y-axis.

3.4 Exfoliation of Graphite

As you may know, carbon appears in two primary forms (or allotropes); graphite and diamond. The former is a fantastic chemical system to learn about London dispersion, the weakest of the intermolecular forces. Graphite is what's known as a layered material, composed of stacked sheets of carbon atoms. These sheets are not held together covalently, or through ionic interactions. Instead, they are bound by the dispersion force, which is able to act on unbound, neutral atoms or molecules (thought it may also act upon these).

To highlight the importance of accuractely accounting for dispersion, you will generate two plots of the "exfoliation" of graphite (pulling the layers apart). One will be done with dispersion, the other without. Use the `B86bPBE` XC functional for this task. Here is the geometry file for graphite:

```
lattice_vector 2.4560000000       0.0000000000       0.0000000000
lattice_vector -1.2280000000      2.1269583917       0.0000000000
lattice_vector 0.0000000000       0.0000000000       6.6960000000
atom_frac  0.0000000000       0.0000000000       0.2500000000       C
atom_frac  0.0000000000       0.0000000000       0.7500000000       C
atom_frac  0.3333333300       0.6666666670       0.2500000000       C
atom_frac  0.6666666700       0.3333333330       0.7500000000       C
```

To "exfoliate" graphite, you will have to make the third lattice vector longer. Notice here how fractional coordiates will serve you - you only have to modify the lattice vector. The atomic positions will change accordingly. Make `geometry.in` files with the third lattice vector ranging from 4 Ångstroms to 12 Ångstroms, in steps of 1 Ångstrom. Run a single point calculation on each geometry and record the total electronic energies. Do this will `xdm lightdenser` in `control.in`, and again with that line removed.

> BASH CHALLENGE
> Instead of setting up each geometry.in file manually, make a template.geometry.in file, with a unique string like "SPAM" in the z-component of the third lattice vector. Use a `for` loop and `sed` to make new geometry.in files. The `seq` command may also prove useful.

Your plot should have a "c-parameter (Å)" label on the x-axis, and a "Relative energy (eV)" label on the y-axis. To make the energy measurements "relative" subtract the energy at 12 Å from each energy data point, such that the 12 Å energy value is zero. Plot both the XDM and the "no dispersion" curves on the plot. What is the difference in bonding between them?

3.5 Adsorption of Benzene onto a Graphite Surface

Being able to run calculations on periodic systems is a major accomplishment. Sometimes though, we have to break the periodicity in one or more directions to study a specific system. In this section of the tutorial, we will bring it all together, and study the adsorption of m-fluorophenol onto a graphite surface.

To turn bulk graphite (which we already have the input file for) into a surface (also known as a slab), we need to add vacuum to the direction in which we wish to break periodicity. Here are the steps I'd like you to go through:

- Make a $4 \times 4 \times 1$ supercell of graphite (this can be done with the NEWCELL command in `critic2`)
- Convert your `geometry.in` file from fractional coordinates to cartesian.
- Change the third lattice vector's length from 6.696 Å to 25 Å
- Take note of the largest z-position of the graphite atoms
- Append the geometry for m-fluorophenol (see above) to your graphite geometry. Change the z-positions of m-fluorophenol to be an appropriate height above the graphite (the relaxation will clean this up, but it does need a good initial guess)
- Choose an appropriate k-grid. If you are breaking periodicity, make the k-value for that direction 1.
- Run a fixed-cell relaxation on the structure and note the final energy.

- Delete m-fluorophenol from the geometry and run the calculation again. Note the final energy.

Now, the energy of adsorption can be computed as:

$$E_{\text{ads}} = E^{\text{graphite/mflurophenol}} - E^{\text{graphite}} - E^{\text{mfluorophenol}} \tag{1}$$

Report E_{ads} in eV. You should already have the electronic energy for m-fluorophenol from earlier in this tutorial.