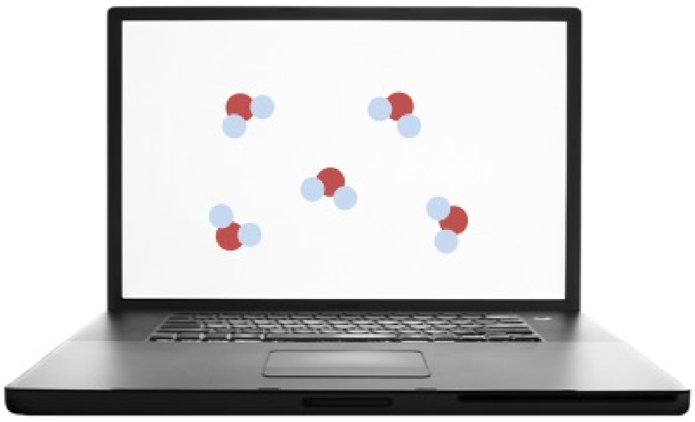
Computational Chemistry Tutorial

Brent Westbrook



Contents

1 Introduction

While computational chemistry can sometimes seem completely detached from reality, in fact it is often a useful tool in the experimental chemist's toolbox. Despite the many approximations necessary for computations on systems larger than a few atoms to finish in our lifetime, the qualitative, and often quantitative, results can point experiment in the right direction. Further, computational results can help to elucidate the theoretical underpinnings of an interesting observation made in lab. An example of this in the work of the Toledo lab is the collaboration with the University of North Texas, which seeks to determine the mechanism underlying the observed reactivity of the synthesized biomimetic nickel complex.

This tutorial assumes a general familiarity with the command line and the tools available therein. Vim or vi is the ubiquitous command line editor, so when editing files on the supercomputer you will usually be using that. Other important tools include ssh for getting to the supercomputer and sftp or scp for getting your files there and back with you. Grep of course will be useful for searching for the important pieces of output files, and some familiarity with shell scripting can help with setting up repetitive calculations, but those are not as necessary.

2 Getting Started

The first step in studying any system computationally is to draw the structure of the molecule in question. The most basic way to draw the structure in a way the computer can understand is by giving the Cartesian coordinates of each atom in the molecule. Since these coordinates correspond to the x, y, and z-axes, a file of this kind is called a .xyz file. The anatomy of a .xyz file is very simple, with the first line containing the number of atoms in the molecule, followed by a blank or comment line, and then the rest of the lines containing the symbol for the atom followed by the x, y, and z coordinates of the atom. For example, a pyridine molecules can be described in this format as shown below.

```
11
Pyridine
C       -0.180226841      0.360945118     -1.120304970
C       -0.180226841      1.559292118     -0.407860970
C       -0.180226841      1.503191118      0.986935030
N       -0.180226841      0.360945118      1.685965030
C       -0.180226841     -0.781300882      0.986935030
C       -0.180226841     -0.837401882     -0.407860970
H       -0.180226841      0.360945118     -2.206546970
H       -0.180226841      2.517950118     -0.917077970
H       -0.180226841      2.421289118      1.572099030
H       -0.180226841     -1.699398882      1.572099030
H       -0.180226841     -1.796059882     -0.917077970
```

Figure 1: Example .xyz file for pyridine

Since the second line is not used by programs that open .xyz files, it is often helpful for you to write the molecule's name on this line to help you keep track of what the coordinates correspond to. As you can probably tell, it would be highly tedious to generate Cartesian coordinates for even a small molecule like water. Instead, coordinates can also be defined in terms of internal coordinates, where each of the atoms is related to other atoms inside the molecule instead of to an external coordinate system. For example, internal coordinates for a methane molecule, starting with the carbon atom, are shown below.

The anatomy here is a little more confusing, but the first column contains the atomic symbols like in the .xyz file, and the second column contains the line number of the atom being referenced. For example, the hydrogen on the second line is located in reference to the carbon on the first line. The third column gives a bond distance between the atom on that row and the reference atom, so the hydrogen on line 2 is 1.089000 angstroms away from the carbon on line 1. On the third row, another two columns are added to give the

```
C
H   1 1.089000
H   1 1.089000  2  109.4710
H   1 1.089000  2  109.4710  3  120.0000
H   1 1.089000  2  109.4710  3 -120.0000
```

Figure 2: Example z-matrix for methane

bond angle between another two atoms. The hydrogen on line three is still 1.089000 angstroms from the carbon on line 1, but it now also forms a bond angle of 109.4710 degrees with the hydrogen on line 2. Finally, for all lines after the third, two additional columns are present, which give the dihedral angle between two atoms. The dihedral angle is probably the trickiest part of the z-matrix, and I am not that well-suited to explaining it myself, so hopefully it is covered somewhat in your organic chemistry class. Anyway, a z-matrix can also contain variables in place of the numbers, so a more helpful version of the z-matrix above would look like that shown below.

```
C
H   1  CH
H   1  CH  2  HCH
H   1  CH  2  HCH  3  D1
H   1  CH  2  HCH  3  D2

CH = 1.0890000
HCH = 109.4710
D1 = 120.0000
D2 = -120.0000
```

Figure 3: Alternative example z-matrix for methane

Defining molecular geometries exactly in these ways can be tedious, but it may be necessary when studying small molecules because small changes to the geometry and the resulting symmetry can have significant impacts on the stability of the system. However, for most of the projects in our lab, the molecules are large enough to make generating a z-matrix or Cartesian coordinates by hand extremely difficult if not impossible. Consequently, I have used a program called Avogadro to build molecules graphically and then to export the Cartesian coordinates for use in the modeling software. Unfortunately, Avogadro will not open z-matrix files, so if you want to play around with those, I recommend downloading MOLDEN, which is a somewhat uglier but more feature-rich visualization and editing tool. Ideally, we would have access to GaussView, which is a very nice editor and visualization tool, but unlike the aforementioned options, it requires a rather expensive license.

3 Using Gaussian

Now that you have the geometry of your molecule, you need to tell the computer what to do with it. One of the most popular ways to run computational chemistry jobs is with a program called Gaussian, which is related to GaussView as mentioned above. Like GaussView, Gaussian requires an expensive license to run, but because it is the software I am most familiar with, it is what I will explain how to use. Other software packages I have used before include CFOUR and Molpro, both of which can accomplish the same things as Gaussian, but each piece of software has its own quirks, so consult the documentation when switching between them. A free option for this type of software that I have heard of is GAMESS, but I have never used it. It may be a good option if you want to put some of this tutorial to work before we get a Gaussian license. CFOUR is technically free too, but you have to print your license agreement and mail it to the developers in

Germany to get a license. A bigger problem you will likely run into is the size of any interesting computations requiring a supercomputer to run. However, if your goal is to play with input files and running calculations at a low level of theory you can probably get something to work on your local machine.

Anyway, on to actually using Gaussian. The Gaussian input file at its most basic is actually quite simple. Complications generally arise when your simple first attempt does not run. An example Gaussian input file I made for the geometry optimization of the ACAC substrate studied in our lab is shown below.

```
%chk=dpen_h2o.chk
%mem=32GB
%nproc=16
!SCRF=(SMD, solvent=acetonitrile)
#BP86/6-31+G(d) opt empiricaldispersion=gd3bj

1-H2O

-1 3
O        -4.23657        3.93579       -0.05242
C        -3.96603        2.73743       -0.02942
C        -2.58625        2.13953       -0.05331
C        -1.39022        2.75358        0.06194
C        -5.10379        1.74343        0.00390
H        -4.74979        0.70961        0.01797
H        -5.72225        1.88225       -0.88741
H        -5.70183        1.91375        0.90335
C        -0.16405        1.85437        0.00498
H         0.06279        1.45987        1.00303
H         0.73186        2.37035       -0.36620
H        -0.32281        1.00249       -0.66805
O        -1.05992        3.95623        0.21590
H        -2.58982        1.05999       -0.15425
```

Going through it line by line, the first line gives the name of the checkpoint file as dpen_h2o.chk. A checkpoint file is where information about the job being run is kept in case of a crash. While you will usually want to restart the calculation from the beginning, you can also restart from the checkpoint file if needed. Sometimes the checkpoint file can also give more information about what went wrong in a calculation. The second line allocates memory for the calculation, in this case 32 gigabytes. The third line allocates the number of processors to be used on the calculation, in this case 16. A brief note about these two lines is that they cannot exceed the memory or number of processors actually available on your machine. For a typical laptop they would likely have to be closer to 8 GB and 4 processors, and that would leave little to no space for even your operating system to run. The fourth line is actually a comment line in Gaussian, as indicated by the exclamation point. Since I wanted to first optimize this system in the default gas phase, I commented out this line that tells Gaussian how to simulate the solvent and which solvent to simulate.

The next line is probably the most important because it specifies the level of theory and the basis set to use in the calculation, as well as what calculation you actually want Gaussian to run. In this case, I included the "opt" keyword to tell Gaussian to carry out a geometry optimization. Other keywords you are likely to encounter are the "freq" keyword for doing a frequency calculation, and the omission of a keyword, leading to a single-point energy calculation without any geometry optimization. There are many other options that can go on this line and on the lines beneath it, but until you know you need to add additional keywords you should be able to keep it as simple as possible.

Below this line must be an empty line, followed by a comment line. In this case, I put the name of the system, 1-H2O, in the comment line. After this is another blank line followed by the charge and spin multiplicity of the system. The charge should be straightforward, while the spin multiplicity is one more than the number of unpaired electrons. Alternatively, you can calculate the spin multiplicity using the formula

$2S + 1$, where S is the total electron spin in the molecule. For this molecule, there are 2 unpaired electrons, giving a total spin of 1, and a multiplicity of 3. Finally, we get to the Cartesian coordinates of the atoms in the system. One important note is that Gaussian requires there to be at least one blank line at the end of the file, so after inputting your coordinates make sure to add an extra line at the bottom.

4 Submitting Jobs

As mentioned before, in almost every case you will not be running Gaussian on your local computer and instead will be submitting the calculation or "job" to a larger computer cluster to be run. Because other people will usually also be submitting their jobs to this same computer, there will be a queuing system in place to hold jobs until space becomes available. Typically you interact with the queuing system by submitting a Portable Batch System or .pbs file like the one shown below.

```
#PBS -q mercury
#PBS -l mem=32gb
#PBS -l nodes=1:ppn=16
#PBS -l walltime=124:00:00
#PBS -j oe
#PBS -e ooc
#PBS -N ooc
#PBS -V

set echo
cd \$PBS_O_WORKDIR

g09 opt.com
```

The anatomy of this file is again pretty straightforward once you know what you are looking at. The first line tells what queue to submit to, in this case it is called mercury. The next line allocates memory for the job; this should be the same as what you put in the Gaussian input file. Next is allocation of the number of nodes and processors per node. Nodes are effectively individual computers that run jobs on the supercomputer. Separate nodes or separate parts of nodes allow multiple jobs to be run at the same time. The walltime directive tells the maximum amount of time the job can run. You can also specify the amount of cputime, which will be the walltime multiplied by the number of processors, but walltime is easier since it is the amount of time we are used to dealing with. The next line joins the log files for the standard output stream and the standard error stream into a single log file. The next line directs this output to a log file with the given name, in this *ooc*. The next line gives the job a name, in this case also *ooc*, and the last line in the #PBS section exports the environment variables in the qsub command environment to the batch job. This just means it passes along your environment variables to the batch script. I am actually not sure what the `set echo` line means, but I assume it says to print information about the job to the command line. The next line changes to the batch working directory using an environment variable, showing why we needed the -V line above, and the last line calls Gaussian 09 on our input file named opt.com. This is the usual name for Gaussian input files, but you can change it if you want to also change this line in your PBS file.

After saving this file, which I usually save as g.pbs, all you have to do is type `qsub g.pbs` to submit it to the queue. On previous supercomputers I have used, I would have to wait a few minutes or a few hours for your job to go from the queue to actually running, but on the Mercury computer Jazmine and I have been using recently it may take weeks. You can check the status of your job by running the `qstat` command, which is short for queue status. If you realize you made a mistake with your input files, run `qstat` to get the job number and then run `qdel` $job - number$ to stop the job or remove it from the queue. You can learn more about all of these commands by using the `man` command as usual, but I think `qstat` is the only one I have ever used any interesting options with.

5 Analyzing Output

The first thing to look for when you get a Gaussian input file is a line near the bottom that says ”Normal termination of Gaussian.” Without this line, something went wrong in the calculation, and you will have to track down the problem and run it again. After that, usually the most interesting information is found by searching for SCF in the output file. This should take you to the energy of the complex’s conformation. However, be sure to grab the last energy line because for each iteration of the calculation an energy will be output. Another often useful piece of information, especially if you need to do another geometry optimization at a higher level of theory, or if you need the optimized geometry for a frequency calculation, is the output geometry. Again, a current geometry is output after each iteration, so make sure to take the last one. The coordinates can be found by searching ”Coordinates” in the output file.

Beyond these straightforward components, the output file can be somewhat confusing, and you probably will not need anything else unless you specifically know what you are looking for. A very common error for geometry optimizations is the optimization failing to converge, which just means that Gaussian could not find a consistently lower energy structure, took as many steps as possible, and then gave up. There are many ways to fix this problem, such as increasing the number of steps you let Gaussian take or giving it a wider tolerance for an acceptable answer, but it will depend on your system and the requirements for your calculation. In some cases you may need to adjust the input geometry.

5.1 Frequency Calculations

The analysis of frequency calculations is slightly different because you are looking for different information. As mentioned above in the running section, you typically want to confirm that your optimized structure is actually a local minimum. You do this by looking at the vibrational frequencies in the frequency output file and seeing if any of them are imaginary. In turn, you can tell if the frequency is imaginary if it is negative in the output file. Zero imaginary frequencies suggests that the structure is at least a local minimum, while one imaginary frequency indicates that the structure is a first-order saddle point or a transition state. Two imaginary frequencies suggests that the structure is a second-order saddle point. Another useful component of the frequency output file is the harmonic vibrational energy correction, which can be used to provide the zero-point vibrational energies for molecules in combination with the energies calculated in the optimization step.