Computer Lab 1c - Python and Data Manipulation

When one performs a molecular simulation the result is numerical data. Much of the effort in interpreting the result of a simulation involves manipulating the raw data. In general, there are four types of data manipulation we will be concerned with:

1. Binning for histogram analysis
2. Function fitting and evaluation
3. Display as plots or graphics
4. Statistical analysis

In this lab, we explore some Python code to perform these types of manipulations and become more proficient with a plotting program to visualize numerical results. You will also be gaining experience with Python programming in anticipation of future labs.

Answer all the yellow-highlighted questions in writing and send the answers to achin14@jhu.edu in the body of the email or as an attached file with your JHEDID in the name (e.g. *JHEDID_lab1c.txt* if created with vim, *JHEDID_lab1c.docx* if created with MSWord).

Exercise 1: Numerically Evaluate a Function

Being able to numerically evaluate an equation is one of the simplest yet most useful skills to have in any programming language. As today's first lab project we will write code to evaluate a function that describes the potential energy of the central carbon-carbon bond torsion angle for butane, CH3-CH2-CH2-CH3, using equation (1).

$$y = 5.0 + 0.5*\cos(\theta) + 0.8*\cos(3.0*\theta) \qquad (1)$$

where θ is in degrees. (Note, this equation does not give the actual potential energy *versus* theta in a quantitative relationship, but it does give us a general shape we can work with).

Open a terminal window on the Mac, change to your flashdrive home directory, create a subdirectory for today's lab and start editing a file called *torsion.py*.

```
cd /Volumnes/[Your_flashdrive]
mkdir lab1c
cd lab1c
vim torsion.py
```

In the file *torsion.py*, write the following Note that you should use blocks of 4 spaces (or a single tab if you must) to indent blocks of code. The exact amount of indentation is not critical but it must be the same for each block of code that you write. This is very important in python. In the code below we end the definition of the torsion function by eliminating the indentation – there is no "end" statement. Note that the function call at the

end of the code is NOT indented. You don't have to include the comments (Lines that begin with #) but it's always a good idea to annotate your code.

```
# Tell Python to import functions from the math module
import math

# Define the function or subroutine
def torsion():
      #Iterate over torsion angle
      for theta in range(0,360):

            # Cosine function expects angles in radians
            # Convert degrees to radians
            x = math.radians(theta)
            # Now write equation [1] in proper syntax
            y  = 5.0 + 0.5*math.cos(x) + 0.8*math.cos(3.0*x)
            # print the result for each theta
            print(theta,y)

# Call the function
torsion()
```

Be sure to save your work by using **:w** and then quit **vim** with **:q**
In object oriented programming you usually call a *function* in a *module* with the syntax: **module.function(argument)**. We are using the cosine function of the math module and supplying the angle value in radians as the argument for the function call.

Run *torsion.py*

```
python3 torsion.py
```

You should see two columns of data appear on the screen.

Now run *torsion.py* and store the result in a file called *torsionOutput.dat*.
This is called *redirecting* the output to a file

```
python3 torsion.py > torsionOutput.dat
```

Look at *torsionOutput.dat* to make sure that the code executed correctly.
Use **xmgrace** to plot potential energy vs θ. Label the plot axes with labels and units where appropriate and add a title (The plot should look like the one in the figure below).

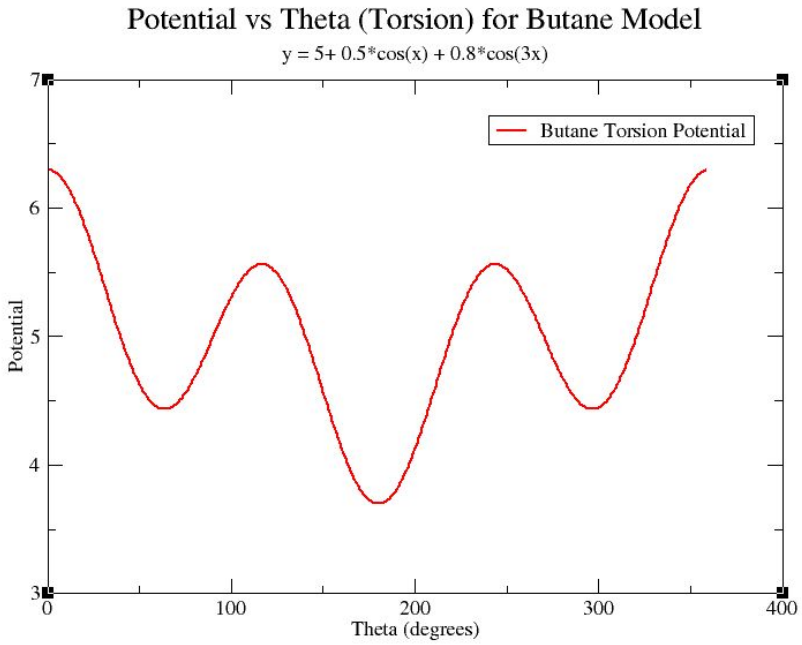
1. Does your plot make physical sense based on what you know about the bond structure and constituents of butane?

2. What causes the activation barriers to rotation?

3. Why would one torsion angle have lower potential energy than the rest?

4. Which energy differences determine whether the molecule shows dynamic or static flexibility?

5. Which energy differences determine the equilibrium proportions of the three most favorable conformations?



Exercise 2: Generating Random Numbers and a Gaussian Distribution:

In order to become familiar with the statistics associated with a physical measurement, or a computational simulation we are going to explore both random numbers and Gaussian distributions. Although the Boltzmann distribution is not exactly a Gaussian, it approximates a Gaussian or normal curve.

Python has a handy random number generator that allows you to produce a random number picked from the range (0,1).

Open a **vim** session in your *lab1c* directory with the command, **`vim myrandom.py`**
Enter the following text into the **vim** window

```
# Import the module called random
import random

# Define the module
def myrandom():
      # print a random number selected from [0 to 1)
      print(random.random())

# Call the module
myrandom()
```

write (**:w**) and quit(**:q**) the **vim** session. The syntax here is that `random()` is a method in the `random` module, so to call the method we use, `random.random()`. We use default arguments for the random function call so we don't need to put anything in the parentheses.

Run *myrandom.py* and observe the output. Run *myrandom.py* a few more times. You're only seeing one value at a time, so how can we tell if we are truly getting random numbers? One way would be to graph many such results and evaluate the graph.

Modify your *myrandom.py* file with a "for loop" to produce 1000 random values in the range [0,1]

```
def myrandom():
      # print 1000 random numbers selected from [0,1]
      for count in range(1,1001):
            print(random.random())
```

Run *myrandom.py* (you should get 1000 random numbers)
Now run myrandom.py and put the output into a file

```
python3 myrandom.py > myrandom.dat
```

Now look at the data in *myrandom.dat* using **xmgrace** to see a histogram of your data

```
xmgrace
Data, Import ASCII
Select File, OK, Cancel
```

Obviously we can't learn much from this plot as it is.
Click **Data** then **Transformations** then **Histograms**. A window will open for specifying what you would like your histogram to look like.

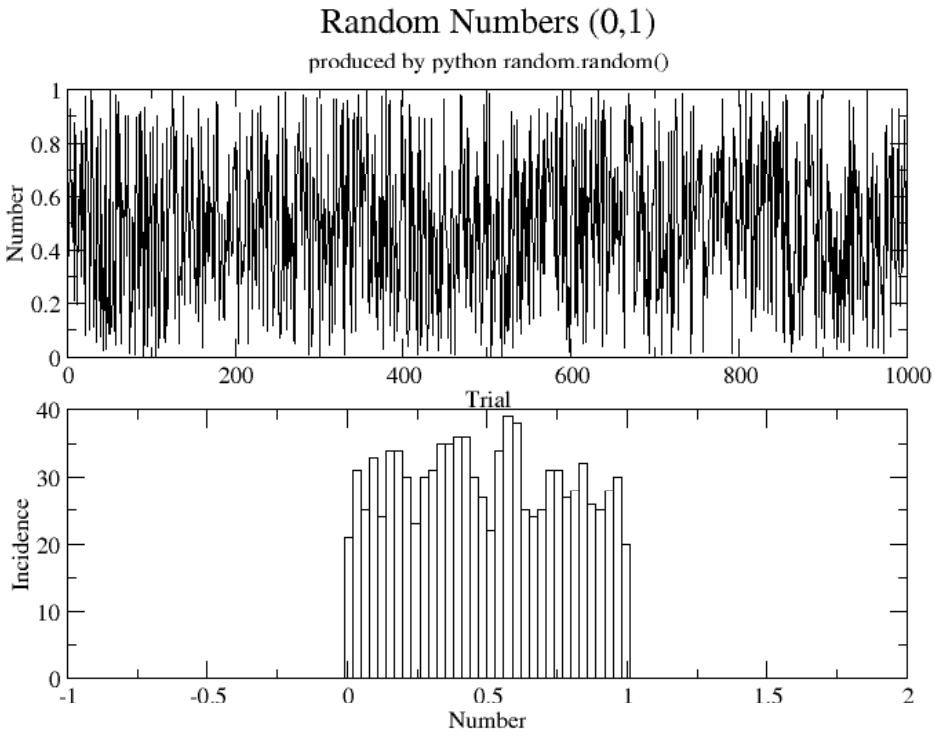
In the upper right hand box named **Destination**, Right click and **Create New**. This creates a new graph **G1** (the original graph was **G0**) where we will put our histogram. Select **G1** in the right hand **Graph** box and then select **G0** in the left hand **Set** box.

The smallest y value looks to be zero and the largest y value is one, we will bracket these values for binning. Make your **Start at: -1** and your **Stop at: 2**, and your **# of bins 100**. Click **Apply** and then **Close**.

Everything is jumbled onto one graph right now. Let's look at our histogram and our original plot as two separate plots.

Click **Edit** then **Arrange Graphs** then make sure both **G0** and **G1** are highlighted (hold down shift while clicking each one) . Number of **columns = 1**, number of **rows = 2**. Click **Apply** and then **Close**.
You should now see two plot boxes, but they are not scaled correctly.

Left click on the bottom plot to make it the active one, then click **Edit** and **Auto Scale Graph**. Select S0...Histogram... and click **Apply** and **Close**.
Your plot should look like the one below; if it doesn't call over Dr. Fleming.
6. Does the histogram indicate that the numbers are random? What is the criterion that indicates random sampling of numbers?
7. How would you increase the confidence level of your conclusion?
Try your idea to increase the confidence level and see if it works.



A **Gaussian** distribution represents the spread of values about a mean (mu, μ), where the width of the spread is characterized by a standard deviation (sigma, σ).

Python also provides a Gaussian value generator as part of the random module. Rather than selecting a random value from between two limits, values for the Gaussian generator are selected based on a Gaussian probability. Values closer to the mean are selected more often, whereas values far from the mean are selected infrequently.

Create a file called *Gaussian.py* using **vim** and enter the following code

```
# Import the random module
import random

# Define the module
def Gaussian():

      # Define the mean and standard deviation parameters
      mu = 3
      sigma = 1

      # Generate 1000 gaussian distributed values
      for j in range(1,1001):
            print(random.gauss(mu,sigma))

# Call the module
Gaussian()
```

Here `gauss()` is a method in the `random` module; `gauss()` requires two arguments and these are the values for `mu` and `sigma`. Run *Gaussian.py* and redirect the output to a file so that you can view a histogram in **xmgrace**

```
python3 Gaussian.py > GaussianOutput.dat
```

Now look at the *GaussianOutput.dat* using **xmgrace** to see a histogram of your data

```
xmgrace
Data, Import ASCII
Select File, OK, Cancel
```

Again, we can't learn much from this plot as it is.

Click **Data** then **Transformations** then **Histograms**. A window will open for specifying what you would like your histogram to look like. In the upper right hand box named Destination, Right Click and **Create New**. This creates a new graph **G1** (the original graph was **G0**) where we will put our histogram. Select **G1.** and then select **G0** in the **Set box.**

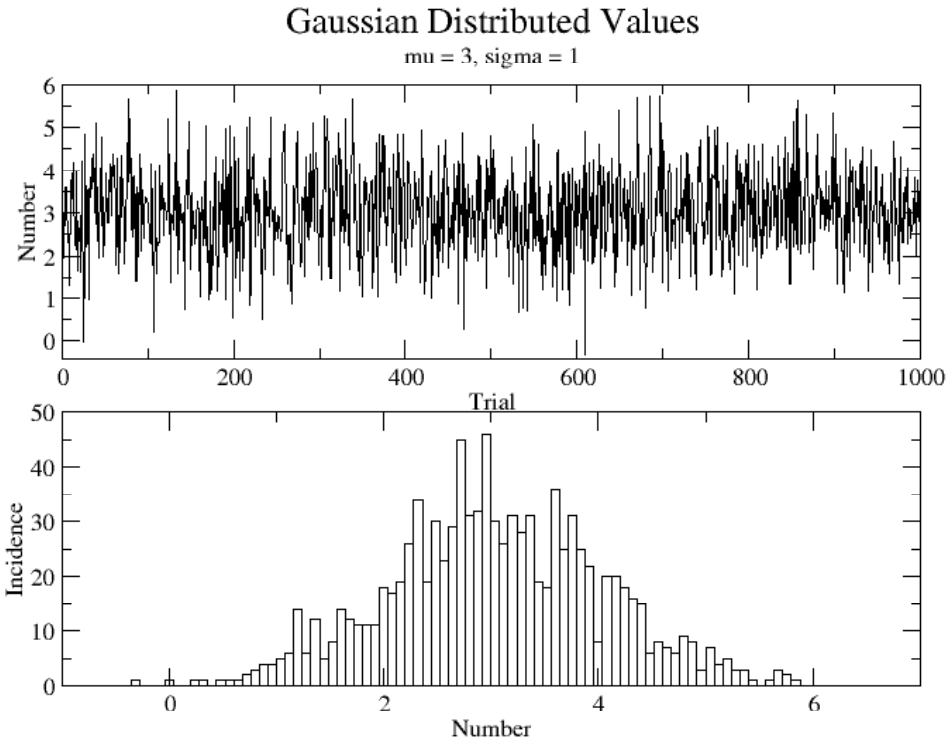
The smallest y value looks to be about zero and the largest about 6 so we will bracket this range for binning. Make your **Start at: -1** and your **Stop at: 7**, and your **# of bins 100**. Click **Apply** and then **Close**.

Let's look at our histogram and our original plot as two separate plots instead of jumbled together on the same plot box.

Edit then **Arrange Graphs** then make sure both **G0** and **G1** are **highlighted** (hold down shift while clicking each one). Number of **columns = 1**, number of **rows = 2**. Click **Apply.**

You should now see two plot boxes, but they are not scaled correctly.

Left click on the bottom plot, then click **Edit** and **Auto Scale Graph**. Select S0...Histogram... and click **Apply** and **Close**.
Your plot should look like the one below; if it doesn't, raise your hand.



8. Is the histogram shaped like a Gaussian distribution?
Here is how can you check it? Make sure the lower plot is active, click **Data** then **Transformations** then **Non-linear curve fitting**. A window will appear that allows you to enter a formula to fit the data with. We will enter a formula describing the Gaussian distribution. This distribution function is frequently defined as,

$$f(x,\mu,\sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(\frac{x-\mu}{2\sigma}\right)^2}$$

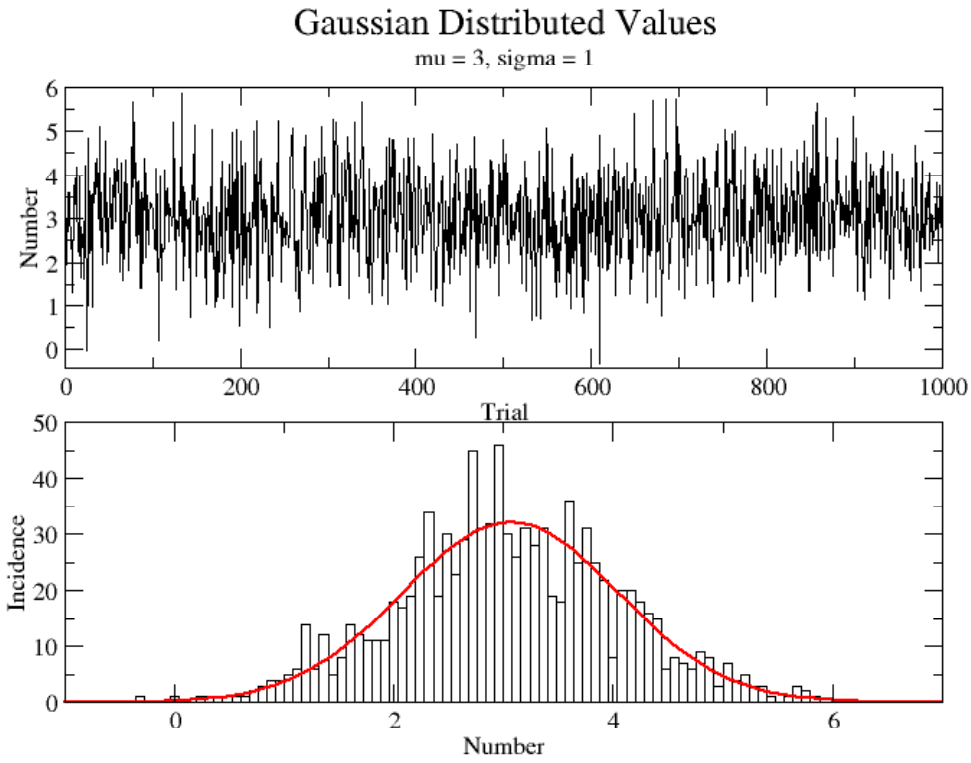
where the $1/\sigma\sqrt{2\pi}$ term can be considered a prefactor that scales the height of the curve and that we will designate as a0 below. Likewise, we will designate the mean μ as a1 and

the standard deviation σ as a2.

Enter the following formula in the **Formula:** box

```
y = a0 * exp(-((x - a1) / (2 * a2))^2)
```

Then click the **Parameters:** button and select **3**. Use the default initial guesses of 1 for a0, a1 and a2 and click **G1.S0** in the left hand **Set** box, click **Apply** and **Close**. You should have a best-fit Gaussian curve appear over your histogram as in the following figure.



9. Does the histogram agree with the shape of the curve?
If the curve does not seem to fit the data click the Apply button again – it may be that the fit did not converge with your initial guesses of 1, 1, 1 for the *prefactor*, *mu*, and *sigma*. If it still won't converge on a reasonable fit, try again with more realistic initial values of 30, 3, 1.
10. How would you improve your confidence in your answer (i.e. how would you decrease the noise in the bin values and increase the correlation coefficient of the fit?
Try your idea.

Remember this lab guide because you may want to refer to the instructions here for future labs and homework assignments.