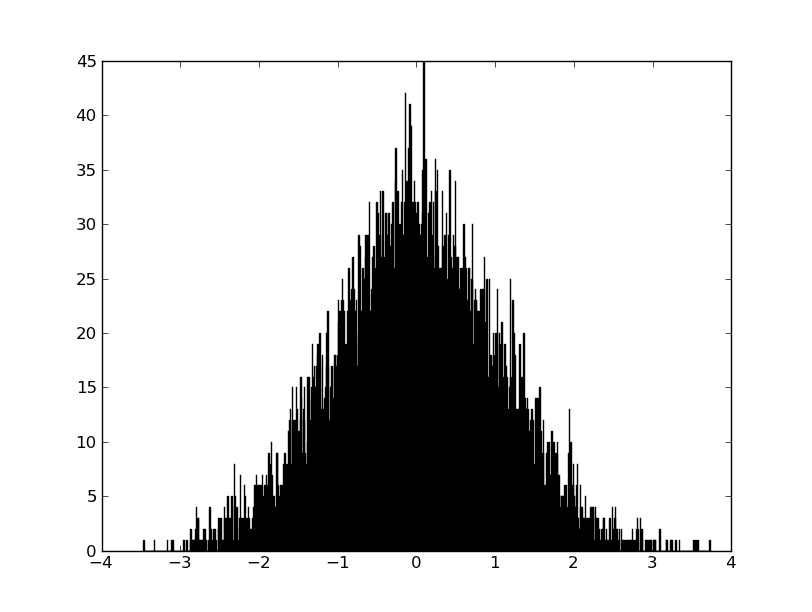
# Lab 0

This Lab is “extra credit”. It is to make sure we have a working system, both in terms of ipython and accessing and submitting labs.

1. Run ipython with pylab. To verify it is working run the command:
   1. hist(randn(10000), 1000)
   2. You should get a histogram on the screen that looks like a ragged Gaussian as shown below. You do not have to show any output from this.



1. The arange command is used to generate a range of numbers (as a numpy array, this fact will be useful in the future but is just a curiosity at this point). Here we will explore this command.
   1. Check the documentation for arange. You should see that start and step are optional arguments (this is why they are in square brackets) and that dtype has a default value (which we will mostly ignore for now).

|  |  |
| --- | --- |
| What are the default values for start and step? | start = 0, step = 1 |

* 1. Play around with arange. Answer for yourself what happens if you enter arange(0), arange(2,0), arange(3,0,-1).

|  |  |
| --- | --- |
| What command would you use to generate a list of numbers from -4 to 4 in steps of 0.1? | arange(-4, 4, 0.1) |
| The command above probably didn’t include 4. Modify the command to include 4 as the last entry. | arange(-4, 4.1, 0.1) |

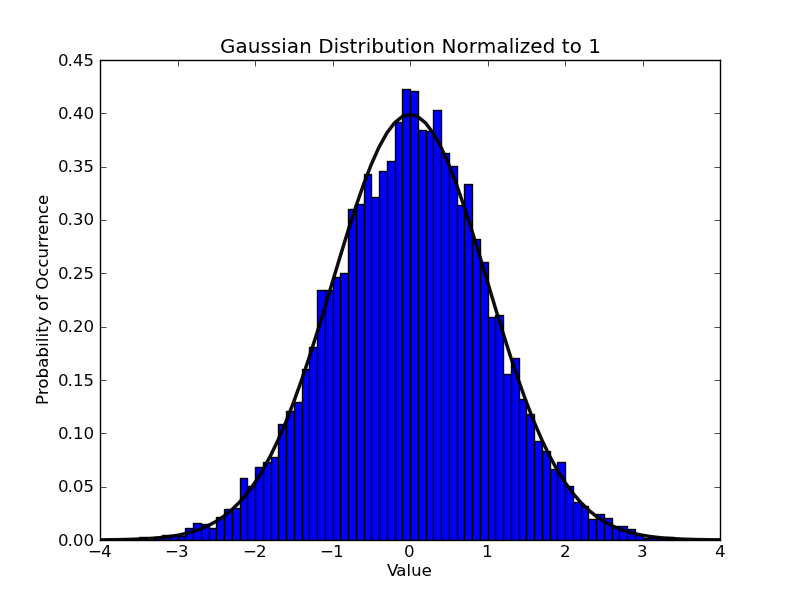
1. The linspace command is also useful. Look up its documentation. Briefly describe the difference between linspace and arange. (Note: Though they can be used somewhat interchangeably there are sometimes reason to use one instead of the other.)

|  |  |
| --- | --- |
| Brief comparison of linspace and arange: | arange’s step size is given directly by the user, where linspace’s range is calculate from the number of partitions given by the user |

1. We will now create a “real” plot. Let us go back to the example from part 1. The randn function is suppose to pick random numbers from a Gaussian distribution with 0 mean and unit variance. To test this we are going to replot the histogram along with the Gaussian distribution it is drawn from.
   1. First generate 10,000 random numbers using randn. Store them in some variable (pick whatever name you like).
   2. Find the minimum and maximum values from your list. Looking through the list by hand is not going to work for this many numbers! Naturally there are functions that will this for you. (Note: There are at least 2 ways to do call these functions. Do you know what they are?)

|  |  |
| --- | --- |
| What are the min and max values for your random numbers? | min = -3.6863335173988365 max = 3.4491374840269131 |

* 1. When we generate a histogram with hist we can specify the binning to use in a number of ways. In part 1 we told it to use 1000 bins, it figured out how big to make them from the data we provided. Alternatively we can specify what bins to use as we will do here. Create an array of values in steps of 0.1 from some value smaller than your minimum value to some value larger than your maximum. It makes sense to use “simple” numbers like 4 or 4.2 instead of arbitrary real numbers. Store this array in some variable. Use this array to set the bins of your histogram using hist. (Note: Check the documentation for hist. You will see there is a keyword call bins.)
  2. We would now like to plot a Gaussian on top of the histogram. We know that a Gaussian is given by . Where is the mean and is the width (variance) of the Gaussian. This distribution is *normalized* but our histogram is not, thus directly plotting this on our figure will not give good agreement. (It is worth doing this and seeing the problem.) There are a number of ways to fix this. The easiest is to normalize the histogram. Fortunately hist will do the work for us! Again check the documentation of hist, you will find a keyword that controls this. Replot a your histogram now normalized. (Note: To erase a plot use clf(). We will use this very often!)
  3. Now we are ready to plot the Gaussian. We can do this using the plot() command. Look up its documentation, it has many options. Use the bins we created in part c for the values (we could use different values if we wanted). Play with the options of plot(). Useful options are the line color and line width. We want to make sure the Gaussian is easily visible in our plot. (Note: It is amusing to plot the Gaussian *first* and then the histogram. What happens? Notice that the order matters. This can be changed by using the zorder keyword.)
  4. We are now ready to make our final version of the plot. Add a title and axis labels to the plot. These should be informative, in particular it should be noted that the Gaussian is normalized somewhere. At this point you can probably guess what functions to use to accomplish this. To get you started look up xlabel(). Include a copy of your final plot below.



Enter the names of all people who worked on this lab as a group. By including a name here you are verifying that said person was actively involved in the work and understands the material:

|  |  |
| --- | --- |
| Names or ids | John Dulin & Lucas Flowers |