Lab 1 - Python, Basic Probability and R

Due: Friday February 10th 11.59PM in PDF form via Websubmit

1 Submission policy.

Every submitted assignment MUST contain the following:

1. your name,
2. the name of any classmates you discussed the assignment with, or the words "no collaborators"
3. a list of sources you used (textbooks, wikipedia, research papers, etc.) to solve the lab, or the words "no sources", and
4. number of late days used for the current assignment, and total number of late days used up thus far in the semester (include on the current assignment).

We will deduct **at least 15% of the points** from submitted assignments that fail to include the four items above.

2 Installing and Configuring Python

In this course we will be using Python for most of the lab assignments. Python is available in the undergraduate lab. If you want to use it on your own computer you'll need to install it.

Start at http://www.python.org/download/releases/2.7.3/ to download the version of Python we will be using in this course. Your next step is to choose a development environment, this is left up to you. You can do some research on your own or use PyDev for Eclipse, an easy to install plugin. It can be found at http://pydev.org/. Configure it to use standard Python (sometimes called CPython) and not JPython or IronPython.

Note: Emacs and VI both have features for Python

Next, take a look at:

- The Python Wiki Guide: http://wiki.python.org/moin/BeginnersGuide
- The official Python tutorial (http://docs.python.org/tutorial/)

Remember, Python is similar to Java in its basic functionality - usually the only difference is syntax. If you have trouble with specific syntax there are hundreds of free, available resources - try looking it up! We highly recommend that you run through a basic tutorial and write some basic Hello, World-esque programs.

One point to notice about Python is its type engine. It is dynamically typed, meaning you dont need to declare the type of most variables. (http://en.wikipedia.org/wiki/Python_(programming_language)#Typing)

Another thing to notice is that while Python is not always interpreted (it can be compiled or interpreted), it follows many of the conventions of an interpreted langauges. This means that Python scripts run from top to bottom. Unlike Java you can have trouble if you call a function above where it is defined.

One of the great things about Python is its huge user base. Lots of users means lots of modules being written to extend Python. We will be relying heavily on three of these modules in this lab, SciPy, NumPy, and matplotlib (which includes pyPlot, the plotting module we will be using). Go ahead and install these modules as well. NumPy and SciPy can be found at http://www.scipy.org/Installing_SciPy - there are downloadable executables which will automatically install in the right place (it should be alongside your Python install).

Getting access to these modules in your Python code is just like Java; an import statement gives you all the functionality you need. The easiest way to plot something is simply

```
from matplotlib import pyplot as plt

plt.figure(1)
plt.plot([x-coordinate array], [y-coordinate array])
plt.show()
```

Instead of using plt.show() to show the window, it is possible to use plt.draw() to draw what has been plotted in the active figure and then plt.show() later - this would be the case where you would like to draw more than one thing in the same figure.

```
from matplotlib import pyplot as plt
plt.figure(1)
plt.plot([1,2],[3,4])
plt.draw()
plt.plot([4,5],[6,7])
plt.draw()
plt.show()
```

Now that you have the basics down, move on to the exercises.

1. Write a function called ex1 to plot points (1,1), (2,2), (3,3), and (4,4) and submit two plots. One should be a "dotted red line". The other should be blue triangles. Note: Look at the pyPlot documentation to learn how to change line color and other features about the line. **Submit your code.**

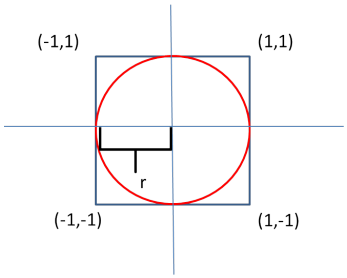
2. Read and understand the function dieRoll(), that estimates the PMF of a random variable Y, where Y represents the number that comes up when you roll a fair die.

```
def dieRoll():
        #Generate an array of size 10,000 with integers 1 through 6
        y_series = randint(1,7,(1,10000))
        #The previous line returns a multi-dimensional array,
        #and we only want the first row of the output
        y_series = y_series[0]
        #Create a figure
        plt.figure(1)
        #Use the pyPlot.hist() function to plot the data.
        plt.hist(y_series,
        #This sets the number of bars in the histogram
        bins=6,
        #This changes the relative width of the bars
        rwidth=.7,
        #This centers the bars
        align='mid',
        #This reweights the data so we see probabilities on the y-axis
        weights=np.zeros_like(y_series) + 1. / y_series.size)
        #Set some features of the graph, so it looks good
        #X-axis size
        plt.xlim(1,6)
        plt.title("Rolling a Die")
        plt.xlabel("Value")
        plt.ylabel("Probability")
        #Show the figure
        plt.show()
```

Modify the function so now, instead of estimating the PMF of Y, it can estimate the PMF of a new random variable Z_m. Z_m is a random variable that represents the sum of the values you obtain when you roll m independent fair dice. (For example, Z_2 is the value you get when you roll two independent dice and take the sum of their outputs. That is, if the first die comes up 4 and the second die comes up 6, then Z_2 takes on the value 10.) Your new function should be called dieroll(m) and take in m as input.

Plot the estimated PMF for Z_1, Z_5, Z_{10} and Z_{100}. Submit these four plots, and your code. Hint: Python has a math module, implementing exponents and all the standard C functionality

3. Write a function pi_Estimate(numPoints) to estimate $\pi = 3.14$ using randomness. To do this, first imagine a unit circle inside a unit box - that is to say a circle with radius 1 inside the box with sides of length 2.



The formula for the area of a circle is πr^2. Now consider the area of the box with corners (1,1), (1,-1), (-1,1) and (-1,-1) - the box that the circle fits in to: $2 * 2 = 4$ or $4r^2$ since $r = 1$.

The ratio of their areas is $\frac{\pi r^2}{4r^2} = \frac{\pi}{4}$ So, the ratio of the area of the circle to the area of the whole is $\frac{\pi}{4}$.

Generate random points in this unit square (i.e. two random numbers between 1 and -1) and determine whether or not they fall in the circle. Keep a count, C for points in the circle and T for total. The ratio C/T $\approx \frac{\pi}{4}$ so $4 * C/T \approx \pi$

About how many points do you need to see 3.14 consistently?

Submit the script you write.

4. Let X be the number of times you need to flip a p-biased coin ($Pr[Head] = p$)until the coin comes up heads (up to and including the final flip). Notice that X is a random variable.

We will estimate the PMF of X using simulation. To do this, write a function *expt(p)* that simulates flipping a p-biased coin until the coin comes up heads, and outputs the number of flips that were required (up to and including the final flip). Now, write a function *Coin_flip_test(n,p)* that runs *expt(p)* n different times, each with independent randomness.

Now write a function that plots a histogram of the output of *Coin_flip_test(n,p)*; this will be our sampled PMF. Scale the y-axis so the histogram shows frequencies, rather than counts, the same way we did in the *dieroll* function.

Submit a picture of the graphs generated with 10,000 samples and probabilities $p = .2, .5, .8$

Does this graph look like what you expected?

Do not use the built in probability mass functions. Rather, come up with a way to use the numPy.random functions to simulate flipping a coin weighted by p.

5. Consider the Coin_flip_test methodology: Continually flip a p-weighted coin until we see one heads. Using math, write a formula for the PMF of this experiment. (i.e. $Pr[X = k] \forall k \in Range(X)$).

Hint:The function is in terms of probability (p) and number of trials (k)

6. Now write a function which uses the formula from the previous question to plot the PMF you computed. Put the plot line on the same plot as the estimated PMF you wrote code for in question 4. Produce one plot for each of $p = .2, .5, .8$ For each plot, your estimated PMF should use $m = 10000$ trials. Hint: Use the figure number (plt.figure(1)) to graph in the same figure. Also, use plt.draw() to draw on the plot before plt.show() to show the figure once and for all.

3 Data analysis with R

In this part of the lab we will learn how to use R to analyze some datasets. R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. If you want it on your computer you will need to install it. You can download it from the following link: http://cran.rstudio.com/. Even though R comes with an integrated development environment (IDE) to write and run commands, my advice is to download RStudio (available at http://www.rstudio.com/products/RStudio/). RStudio is an IDE for R with a lot of nice functionalities. It includes a console, syntax-highlighting editor that supports direct code execution, as well as tools for plotting, history, debugging and workspace management.
A tutorial about R can be found at http://www.cyclismo.org/tutorial/R/ (parts 1,2,3,5 are very useful to solve the following exercises). R also has its own help; when you want to know more about a function just type the symbol ? followed by the name of the function. For example to see the help for the function mean just type **?mean** in the Rstudio console.

7. In this exercise we will learn how to read a .csv (comma separated values) file and compute some basic statistics about it. The function you need to use is **read.csv**. Type **?read.csv** in the Rstudio console to see the function help page. After you've understood how to use it, open the file *grades.csv* (download it from Blackboard under Assignments, Lab 1). The file contains students GPAs for the a class for the past three years and every row is a student GPA. Note that for privacy reasons the student ID is ommitted.
Now that you opened the file, let's take a look at it with the function **View**. To learn how to use it you can consult the functions help page. Among the first 10 students in each class, how many of them have a 4.0?

8. Now let's play with the data. For every class in the dataset compute the summary statistics (you should use the **summary** function). Provide the output of the function and a description of the statistics provided.

9. One statistic not provided by the summary function is the standard deviation. The idea behind the standard deviation is to quantify the spread of a distribution by measuring how far the observations are from their mean. For every class compute the standard deviation of the students GPAs.

10. Now we want to visualize the summary statistics with a boxplot. In descriptive statistics, a boxplot is a convenient way to graphically depict groups of numerical data. Create a figure containing the boxplot of every class GPAs side-by-side. What do the following represent: a)the line inside the box, b)the top of the box and c)the bottom of the box? Which percentage of observations are contained in the values covered by the box?

11. We conclude the analysis of the dataset by plotting the Probability Mass Function (PMF). There are several ways in which we can compute the PMF using R. An easy way is to use the **table** function, which takes as input a vector of values and returns the frequencies of each value in the vector. We can transform frequencies to probabilities by dividing them by the total number of observations in the vector (which is the **length** of the vector or the **sum** of the output of the **table** function). We can plot the result using the function **plot**. Follow the above instruction and plot the PMF of the GPAs for every one of the classes. Provide the commands you used and the plots.