**Probability and Applied Stats**

**Plotter, Salter, Smoother #1**

For this portion of our final in the course Probability and Applied Stats we were asked to create a java program (Plotter.java) that takes a formula of our choosing and plots it into a csv file, for my formula I chose the Pythagorean Theorem. Next, we were tasked with creating a program (Salter.java) that is able to read the data from this csv file and salt it, then save that salted data to a new csv file as well. In other words, it is going to purposefully skew our data by adding and subtracting a random amount from each of the y values in that file. After doing this we would have to attempt to smooth the data again through another program (Smoother.java) that saves that data to a separate csv file. Then, in the csv files we had to manually create charts for each of these programs. For this first portion of plotting, salting, and smoothing we were forced to accomplish these tasks with one hand tied behind our backs. Without the use of any external libraries or ulterior languages such as MATLAB and Octave that can create the charts for you, we took an inefficient approach to these programs. However, while there are other ways to get the same job done faster, in the end the results are similar. For Plotter.java I was able to generate 5 different datasets (data.csv - data5.csv) by changing the parameters of my formula. Specifically, I adjusted the start and end values in order to differ the dataset sizes. Also, I changed the interval and size of the fixed side to vary the values outputted. Doing this allowed me to observe that increasing the interval led to less accurate smoothing. Then, my salter acquires the hypotenuse from data.csv and adds a new parameter called saltedHypotenuse and outputs it to the new saltedData.csv file alongside the original values. The program outputs three variations for the salted data based on the range of the salting using an array called saltRanges to control the randomness. The ranges are as follows, small [-5, 5], medium [-20, 20], and large [-50, 50]. The larger ranges introduced more noise which in turn required more smoothing passes for the improvement to be noticed. From this file, Smoother.java applies a smoothing technique by replacing the saltedHypotenuse with the average of the other saltedHypotenuse values around it. This program takes into account the window size, or the number of neighboring points that are to be averaged, and passes, the number of iterations the program runs (5). This is all output to a final smoothedData.csv file which shows the original, salted, and smoothed data. As mentioned before, I created a chart for each of these programs however another drawback quickly arose during the process. We were required to save the data to a csv file and create the charts manually but every time I would create a chart in excel and save it on the csv file, my work would not be there when opening the file again after closing it. I had to change the format to a .xlsx file in order for my changes to save, this is the reason that there are dataChart.xlsx files to go along with their data.csv counterparts in my submission. These are just little nuances that are avoided when working with MATLAB and external libraries which is highlighted in the parts of this project still to come.