Mohammed Mowla Probability and Applied Statistics

Project 3

**Database Inspired Problems:** The data set that was used to create the following questions used a League of Legends data set that was comprised of [10,000 games of League of Legends ranked Diamond or above games](https://www.kaggle.com/datasets/bobbyscience/league-of-legends-diamond-ranked-games-10-min). To give further context, League of Legends is an extremely popular M.O.B.A. game that has ranked match making. Two teams, red and blue, fight against one another to destroy the enemy nexus. The ranks follow: Iron, Bronze, Silver, Gold, Platinum, Diamond, Master, Grandmaster, and Challenger.

The data set includes only includes ranked games from Diamond to Masters. The data set includes data that correlates to the first 10 minutes of a game. The data includes things such as kills by individual players, gold difference, which side had won (blue vs. red side) among many other things.

Binomial Distribution: Inspired by self.

What is the likelihood of winning four out of seven ranked games on red side given that the probability of success for a single game is 0.507?

Solution:

Geometric Distribution: Inspired by self.

Given any league of legends ranked game from diamond to masters. What is the likelihood of getting dragon at 10 minutes or under at your fifth game of the day knowing that the chance of getting a dragon under 10 minutes is 0.3697?

Solution:

Hypergeometric Distribution: Inspired by self.

Assume you are a League of Legends blue sided player who has placed fourteen wards. Of these fourteen wards placed, four are in enemy territory and ten are in your territory. What is the probability of exactly all four of your wards placed in enemy territory being destroyed if four wards of your are destroyed?

Solution: = = 0.000999001

Poisson Distribution: Inspired by question 3.122 on page 136.

Wards are destroyed on the enemy blue side in a League of Legends game according to a Poisson Distribution at an average of three per hour. During the first 10 mins of a blue players game, what are the possibilities that

a) no more than three wards are destroyed?

b) at least two wards are destroyed?

c) exactly five wards are destroyed?

Solutions:

a) = 0.647

Tchebysheff’s Theorem: Inspired by question \_ on page \_.

Uniform Distribution: Inspired by question 4.44 on page 176.

The change in the number of wards placed by a single person on the blue side of one league of legends game to the next is a random variable with the following density function:

Determine the value of k.

Solution:

k = .

**Stats Library Update:** The only update made to the statistics java library was the uniform distribution. A uniform distribution is a scenario where all outcomes are equally likely to occur.

UniformDistribution.java contains methods that calculate the probability of an event occurring given that it has a uniform distribution, the expected of said uniform distribution, as well as the variance and standard deviation of the uniform distribution. The method are as follow:

- distribution(double lowerBound, double upperBound, double problemLBound, double problemUBound) This method calculates the uniform probability by accepting 4 doubles. The first two are the problems absolute upper and lower bound. The last two are what the problem is asking the solver to find, and this involves the questions upper bound and lower bound given the scenario. Once all four of the requirements are met, the method solves the distribution and returns the answer as a double. The answer is also printed out onto the console as a fraction and decimal answer.

- expected(double lowerBound, double upperBound) accepts two doubles as parameters. It calculates the mean of the uniform distribution problem and returns it as a double. It also prints out the answer as a fraction and decimal onto the console.

- variance(double lowerBound, double upperBound) accepts two doubles as parameters. It calculates the variance of the uniform distribution problem and returns it as a double. It also prints out the answer as a fraction and decimal onto the console.

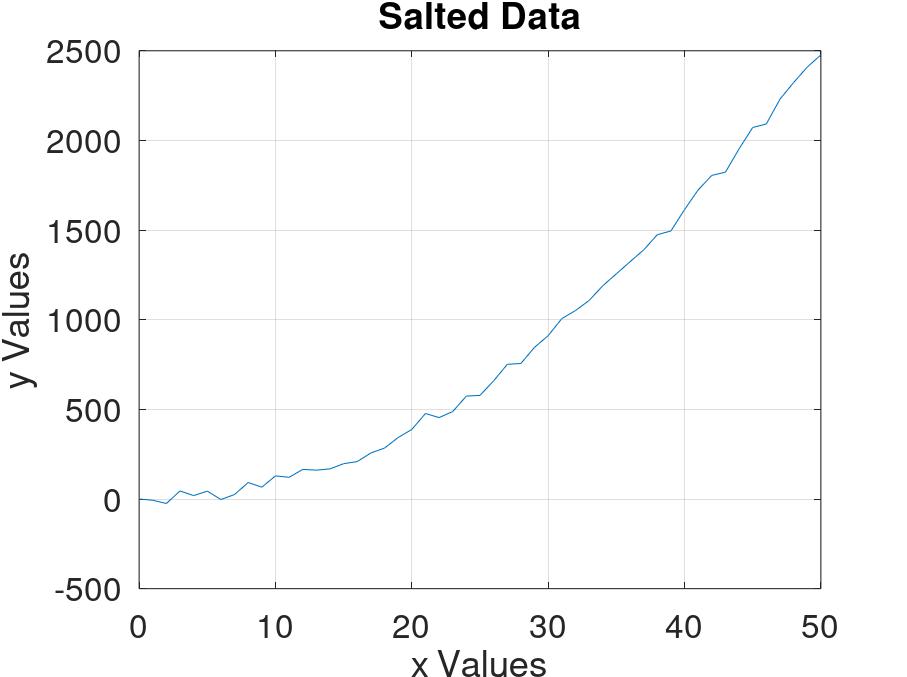
- standardDev(double lowerBound, double upperBound) accepts double as parameters. It calculates the standard deviation of the uniform distribution problem. Since the standard deviation is the square root of the variance, the method repeats the actions of the variance method except it takes the square root of the answer. It returns the answer as a double as well as printing out the answer as a fraction and decimal onto the console.

- runAll(double lowerBound, double upperBound, double problemLBound, double problemUBound) accepts 4 doubles as parameters. It runs all of the previous methods and has no further internal calculations.

The DistributionTester.java class simply tests the uniform distribution.

**Octave Output:** Octave is a programming language that is primarily used for numerical computation. It also allows for graphical output of data. This functionality was used to read data from a salted graph in the form of a .csv (Comma Separated Value) file and output it as a graph. This data then had a rolling average of size 4 applied to the data to smooth the data. This smooth data then was also graphed. The original graph of the salted and smoothed graph is

Salted Graph:

To create this graph, the Octave script was created:

Text

Description automatically generated

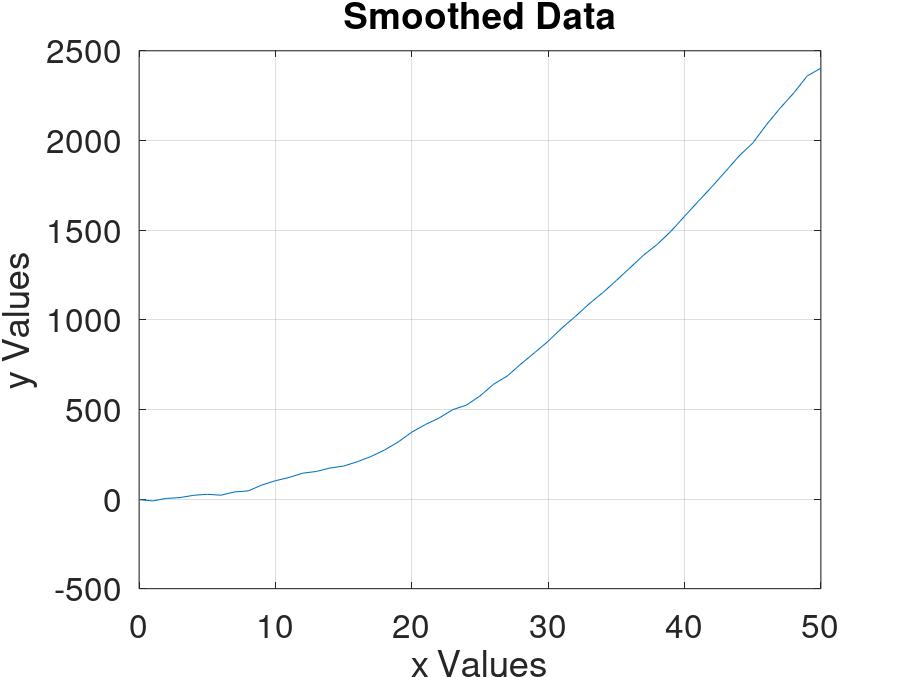
Graph 1: Salted Data at the far left.

Output of Octave Function 2

Octave Function 1: plotSalt.m to the immediate left. Creates Graph 2.

At line 1, clear all, empties all stored data and function calls. This is important because the plotSalt.m and plotSmooth.m scripts use the same variable names and Octave doesn’t determine which script they belong to. At line 2, file sets a variable to the salted data .csv file. Line 3 has the data read from the .csv file and stores it in a matric of size 51 by 2 with the variable name ‘A’. Lines 4 and 5 stores the x and y data in two variables named appropriately ‘x’ and ‘y’. Line 6 actually plots the x and y variables which creates the output. The following lines after are just adjustments to what the output will look like. Such as turning on the gird, increasing the font size, and labeling the axes.

Smooth Graph:

To create this graph the Octave Script was created:

Graph 2: Smoothed Data at the far left.

Output of Octave Function 2

Octave Function 2: plotSmooth.m to the immediate left. Creates Graph 2.

Text

Description automatically generated

The difference between the smoother script and the salter script is at line 6, where a variable yAvg is set to the output of the function movmean(y, 4). The movmean function takes the moving average of a list of a data (first parameter) over an amount of numbers (second parameter). The plot function at line 7, takes ‘x’ and ‘yAvg’ instead of the variable ‘y’ since we want the graphical output of the moving average y values.

**Apache Commons Math and JFreeChart:** Using the Apache Commons Math Library and JFreeChart Library, a graph of the salted data, a smoothed data .csv file, and a graph of the smoothed data were created.

ApacheSmoother.java class took in the data from the “SalterData.csv” file and applied a rolling average to the y values. It did this by using a class that existed in the Apache library. The object of that class was a DescriptiveStatistics object. This class allows for an object of this class to be created with a certain window size. The window size determines how many numbers it can hold before it pushes out the last number in the list out. This allowed the object to function as a rolling avg calculator. By adding numbers to the object and then find the average, and then repeating the process with each y value from the salted data, the salted data was smoothed. This smoothed data was then saved by creating a .csv file named “SmoothData.csv.”

JFreeChartsGraph.java is the class that managed the output for both the salted and smooth data. Since both sets of data are stored in .csv files, the data was read from the files and stored in temporary array lists. From the JFreeChart library, two objects of the DefaultCategoryDataset class were created to store the salt and smooth data. The data from the array lists were added to the DefaultCategoryDataset objects. This was how the data had to be stored for the next class, the ChartFactory class to create a chart from the data. ChartFactory objects then were used as a parameter for the ChartUtilsl.saveChartAsJPEg() static method which output the charts as .jpeg files.

Chart, line chart

Description automatically generated

Graph 3: Smoothed Graph created by combination of Apache Commons Math and JFreeChart

Chart, line chart

Description automatically generated

Graph 4: Salted Graph created by combination of Apache Commons Math and JFreeChart

Works Cited

“Apache Commons Math 3.3 API Overview.” *Apache Commons Math 3.3 API*, 2014, https://commons.apache.org/proper/commons-math/javadocs/api-3.3/index.html?org%2Fapache%2Fcommons%2Fmath3%2Fstat%2Fpackage-summary.html.

“JFreeChart 1.5.3 API.” *Overview (JFreeChart 1.5.3 API)*, 21 Feb. 2021, https://www.jfree.org/jfreechart/javadoc/index.html.

Wackerly, Dennis D., et al. “Chapter 2: Probability.” *Mathematical Statistics with Applications*, Seventh Edition ed., Brooks/Cole, 2008, pp. 157–295.