Report on Java libraries (Apache and JFreeCharts)

**Introduction**

During this time, three versions of the plotter, salter, smoother were created. One being in regular Java, the second being in MATLAB, and the third being created with Java libraries (Apache and JFreeCharts). This report is based off the third mentioned: the Java libraries. The Apache math library can perform statistics computations and JFreeCharts allows users to create graphs within Java without even having to touch Excel. The principle ‘why reinvent the wheel when someone else created it’ is put into stone here. Using the library to do the heavy work instead of importing .csv files into Excel was extremely satisfying. This report will detail the graphs, the process of accessing the libraries and briefly explain the purpose of the program with its methods highlighted. For full code documentation, please visit “Java library documentation” in the documentation folder.

The process of getting it set up was the most difficult part. However, even that was not that bad. Having never used Java libraries before, it was a hassle. Once sat down, it all came easy. All that was done to import these libraries was adding them as dependencies in a Java maven project and calling it a day. In the ‘pom.xml’ file, the dependencies for the libraries can be included in that section of the file. After refreshing/restarting the IDE of choice (VSCode was used for this project), the libraries will be loaded into the project and can be imported.

The process of coding everything and outputting the graphs was very simple once set up. It involved coding in Java still but with the libraries this time. The libraries handled all the graphing and data. After creating the plotter salter smoother, there was a graph class created too which handled all the optimizations of the three graphs like the name, the x and y axis, the color of the line and the size of the window. As both classes worked together, from the Java program itself, the graphs popped up as their own windows without any outside intervention.

**The role of the program**

* The role of the program is to plot, salt and smooth a function. The program does this mainly in the “PlotSaltSmoothData” class. All the classes were split up for organization and personal preference.
* Within the first class is the plotter, salter and smoother. All three methods work off a quadratic function.
* First, the function is plotted in this case, the quadratic function was plotted.
* Second, the function is destroyed and messed up by the salter based off a parameter inserted by the user. It can be salted softly or extremely. In other words, it can look a little bit messed up or a lot messed up.
* Third, the function is smoothed. The function is fixed, and because of the smoother, it looks much more like the original. Based on the value the user chooses; it can do a great job on the function or perform little to nothing. It also depends on the salting value. If it’s extremely salted, it will take more time to smooth.
* The program works off a XYSeries, inserting the plotted, salted and smoothed values all into their own series. This made it easier to keep the values organized to access later to create the graphs.
* After everything is plotted, salted and smoothed, these XYseries are all returned in their own get methods to use in the next class.
* The “GraphMaker” class creates all three of the graphs for the plotted, salted and smoothed quadratic. All their attributes are inserted into the JFreeChart like their line color, title, plot orientation, axis labels and more.
* The program runs with a scanner, allowing the user to press 1, 2, or 3 to run the graph they want to view. It will prompt them, asking which graph they want to see. After they choose, it will pop up in a new window.
* The main method’s purpose is just to run the test() method, which includes the scanner. The scanner handles all three of the graphs that hold the code that will pull from the plotter, salter and smoother.
* Next a quadratic function was plotted, salted and smoothed with Apache and Jfreecharts.

The quadratic function was:

***Fig. 1.1: Plotted Quadratic Function***

A graph with a purple line

Description automatically generated

This was the original quadratic graph. There was an RGB color code used to present the line as purple. The x-values that were plotted were from -50 to 50, keeping the x-value trend throughout all the reports the same. The line was labelled a quadratic function, and the name was “Quadratic Function Plot” to show what it was.

***Fig. 1.2: Salted Quadratic Function***

A graph with a line

Description automatically generated

The salted quadratic was salted by 100. It was not salted intensely, but the salt is visible as it shows a minimum number of jumps in the line. The first graphs in the first two reports tend to be way more salted than this report, but it is still salted. The color chosen here was blue for the line and it was named “salted Quadratic Function Plot” for clarity.

***Fig. 1.3: Smoothed Quadratic Function***

A graph with a line

Description automatically generated

The smoothed quadratic function shows a pretty depiction of the salted function. Instead of having jumps in data and a mess going on, it smoothed it back to the original. It still is not perfect as if looked closely, it is not fully straight, but it is smoothed from the salted. A hot pink color was picked here. The graph was named “Smoothed Quadratic Function Plot” for clarity.

*Note: There is full documentation on all the code and classes in the documentation folder named “Java Libraries PSS.docx”.*

***Method description:***

*The PlotSaltSmooth class is below:*

* plotQuadraticFormula() – the purpose of this method is to plot the quadratic function: It does that by using a for loop and adding x-values -50 to 50, then using the equation to make the y-values.
* SaltFormulas(int bound) – this method uses the XYSeries plot series to get the right range of values to iterate over. It takes both the x and y values using the getX and getY commands. It creates salt by working off a parameter (bound) which the user can enter and subtracts that entered bound by 100. The x value and the salted y value are then added to the salted series.
* smoothFormulas(int range) – This method works off the “range” to average values around each data point. This created the sensation of a ‘fixed’ or smoothed function. It works with the salted series with low and high bounds to fix the salt. It ends up adding the -50 to 50 x values and the smoothed y values into the smoothed XYseries.
* getPlotSeries() – returns the plot series to be used elsewhere.
* getSaltSeries() – returns the salt series to be used elsewhere.
* getSmoothSeries() – returns the smoothed series to be used elsewhere.

*The GraphMaker class is below:*

* Graph() – creates the plotted quadratic graph by working off the plotted Series from the initial class. It creates a JFreeChart with unique attributes.
* Graph2() –creates the salted quadratic graph from the salted Series, using the plotted series values. It takes both from the first class. It creates a JfreeChart with unique attributes.
* Graph3() – creates the smoothed quadratic graph from the plotted and salted values. It smooths the salted values. It takes both from the first class. It creates a JFreeChart with unique attributes.
* run() – utilizes a scanner to prompt the user, asking which graph they would like to view. Based off of the users answer ( 1 – 3 ), the requested graph will show up on the screen in a new window.

There is a main class with a main class which runs the test() method.