**JFreeApache User Manual**

A modified version of PSS using JFreeCharts and Apache Commons Math

Cachary Tolentino

CSCI-3327

**Table of Contents**

**Software Description3**

Detailed Description3.1

System Requirements3.2

**Installation Guide4**

**Class Overview6**

**Result Analysis11**

**Software Description**

A modified version of PSS using JFreeCharts and Apache Commons Math

**Detailed Description**

The JFreeApache program is a variant of the PSS program in which it can plot, salt, and smoothen the values of a quadratic formula. This version focuses on using APIs such as JFreeCharts for simple graphing and Apache for built in math functions.

**System Requirements**

* A working device, primarily a desktop or laptop
* An IDE (ex: VSCode, Eclipse, etc…)
* Java JDK (Ver. 17 & up) & JRE (SE 17 & up)
* Maven (preferred) for JFreeChart and Apache

**Installation Guide**

To begin using JFreeApache you will need to download the following files. “JATest.java” is optional.

A screenshot of a computer

AI-generated content may be incorrect.

After downloading the files, simply move the files to the folder containing your Maven project. Once done, you can open your preferred IDE (for this example we will be using VSCode). Then you can open the folder or the file itself within your IDE.

A screenshot of a computer

AI-generated content may be incorrect.

If you opened the folder containing the files then it should look similar to the image below.

A screenshot of a computer

AI-generated content may be incorrect.

You can immediately start using the files and their functions within. But if you want to see an example, then run the tester file. The result will be displayed on the console and graphs will be displayed on a separate window.

A screenshot of a computer

AI-generated content may be incorrect.

**Class Overview**

TestPlotter Class

**main Function**

The function simply calls on the various functions from the JAPlotter, JASalter, and JASmoother classes. It creates an instance of each object type and calls on their respective function such as plotData(), salterData(), and smoothenData().

JAPlotter Class

**quadraticFunction Function**

The function simply calculates the quadratic function value (Y). The function requires three parameters: x (double), a (double), b(double), c(double). The function will then return the Y value as type double. The function utilizes Apache’s built in PolynomialFunction object to find the value.

**plotData Function**

The function has no parameters but returns a string value. It will perform four main tasks. It will first ask the user for inputs regarding the quadratic function (inputs for x, a, b, and c values). It will then iterate through the given amount x values and add its corresponding y value by calling the quadraticFunction(). These are then added to a data structure (an arraylist of strings). It will then export the data structure into a csv file user the exporter function of the DataHandler class aswell as graph the data. Finally it will return the name of the file.

JASalter Class

**JASalter Constructor**

The salter constructor takes in one parameter: inputFile. inputFile is of type File.

**salter Function**

The salter function has a three parameters: data which is of type ArrayList<String>, lowerBound an int value, and upperBound also an int value. The function will return newData which if of type ArrayList<Double>. The function will salt the data by randomly choosing to either add or subtract a unique salt value to the current Y value. It will purposefully skip all the X values (found at odd positions). Each salted data or X values is then added onto newData, once finished iterating and salting data, then newData is returned. The saltValue and operation uses Apache’s built in RandomDataGenerator to randomly create a value given a lower and upper bound.

**salterData Function**

The salterData function has no parameters but returns a string value. The function is responsible for parsing the inputted data file using the DataHandler’s parser function. Once the data has been parsed, it will then ask the user for the salting ranges which will then be used during the call on the salter() function to salt each Y data value. Finally, the newly salted data is exported using the DataHandler’s exporter function, it will also graph the data, and the function returns the new file name of the salted data.

JASmoother Class

**Smoother Constructor**

The smoother constructor has one parameter value: inputFile. inputFile is of type File.

**smoother Function**

The function has a two parameters: data which is of type ArrayList<String> and windowValue which is of type int. This function is responsible for smoothing the given data set (assumed to be salted). The function does so by using the helper function: rollingAverage(). Once all the data values have been iterated, the function simply returns the ArrayList<Double> containing the newly smoothened data.

**rollingAverage Function**

The function has a two parameters: four parameters. Data, middle, windowValue, and stats. Their types are of: ArrayList<Double>, int, int, and DescriptiveStats. The function uses Apache’s built in getMean() function to average the values provided to the stats object. This function simply finds the appropriate bounds for the current middle position and adds the values to the stats object. It returns the average of all values within the stats object.

**smoothenData Function**

The smoothenData function has no parameters and no return value. The function is responsible for parsing the inputted data file using the DataHandler’s parser function. Once the data has been parsed, it will then ask the user for the salting ranges which will then be used during the call on the smoother() function to smoothen each Y data value. Finally, the newly smoothened data is exported using the DataHandler’s exporter function and the function returns the new file name of the smoothened data.

JADataHandler Class

**parser Function**

The function has a single parameter: dataFile which is of type File. This file contains the none-parsed data. The function will iterate through the file line by line and split each line to their own respective values. The inputs follow the following pattern: “X, Y”. Thus, the parser simply splits at the “,”, and adds the X and Y values from the string[] variable. Once there are no more lines to iterate through, it simply returns the newly created data structure containing the parsed data which is of type ArrayList<Double>.

**writer Function**

The writer function has a single parameter: data which is of type ArrayList<Double>. This function is responsible for adding the X and Y values from the data as string values in the following pattern: “X, Y”. It will iterate through all the data (every pair) and once done it will return the value which is of type ArrayList<String> containing the X and Y values.

**exporter Function**

The exporter function has two parameters: data and fileName which are of type ArrayList<String> and String. The function will export the provided data into a csv file. By using FileWriter, it will add a string in each line in the following pattern: “X, Y”. The function does not return any value.

**grapher Function**

The exporter function has two parameters: fileName and dataset. They are of type: string and XYseriescollection. The function uses JFreeChart’s ability to plot a given dataset. It creates a chart, a chart panel using the chart, and a data frame. It will then display the graph of the values. It can also save the displayed graph as a PNG file. Finally, it returns the data frame used primarily for allowing to exit the program properly on the last call.

**Result Analysis**

The following graphs are example outputs (graphed in Excel from the data points of the program provided) of the program. The quadratic output was configured with its quadratic formula as . Here we can see a typical graph for this type of formula with no problems.

**A graph with a red line

AI-generated content may be incorrect.**

In the following graphs, we can see a more interesting difference mainly due to the salting and smoothing of the graph above.

In the graph below, we can see the salted version of the quadratic output. Here the salting range used was from 10000 to 50000. This causes the data points to either be increased or decreased from their original value, hence, the points starting to deviate from the straight line in comparison to the original graph.

A graph with red lines

AI-generated content may be incorrect.

Finally, here we can see the smoothened graph of the salted graph. This used a window value of 20, as such any points on the graph were replaced with the average value of the values from that focus point’s left and right side. This “smoothens” the salted values thus allowing the graph to appear more like the unsalted version.

A graph with red lines

AI-generated content may be incorrect.