**Java Apache/JFreeCharts CSV Demo & Report – Project 2**

**Anthony D’Alessandro**

**Table of Contents**

Methods……………………………………………………………………………………………1

Plotter…………………………………………………………………………………...…1

Salter………………………………………………………………………………………2

Smoother…………………………………………………………………………………..3

Graph………………………………………………………………………………………4

Small Scale Trial…………………………………………………………………………………..5

Plotter……………………………………………………………………………………...5

Salter………………………………………………………………………………………6

Smoother Run 1…………………………………………………………………………...6

Smoother Run 2…………………………………………………………………………...7

Smoother Run 3…………………………………………………………………………...7

Larger Scale Trial………………………………………………………………………………….8

Plotter……………………………………………………………………………………...8

Salter………………………………………………………………………………………9

Smoother Run 1…………………………………………………………………………...9

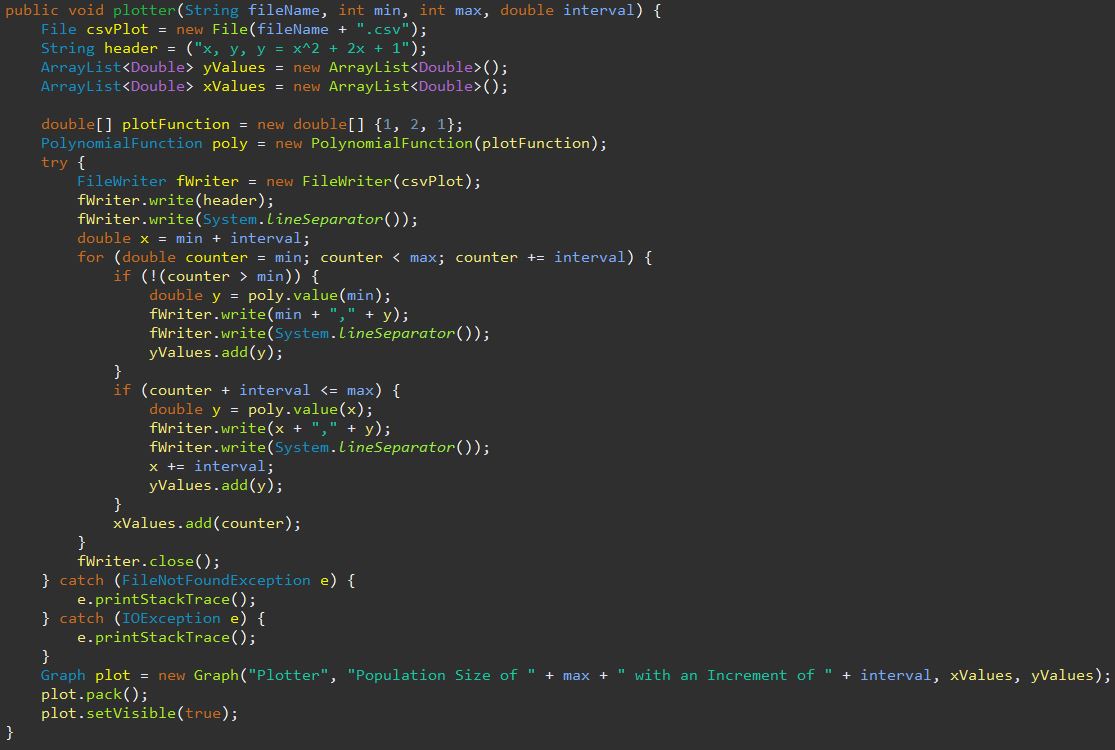
Smoother Run 2………………………………………………………………………….10

Smoother Run 3………………………………………………………………………….10

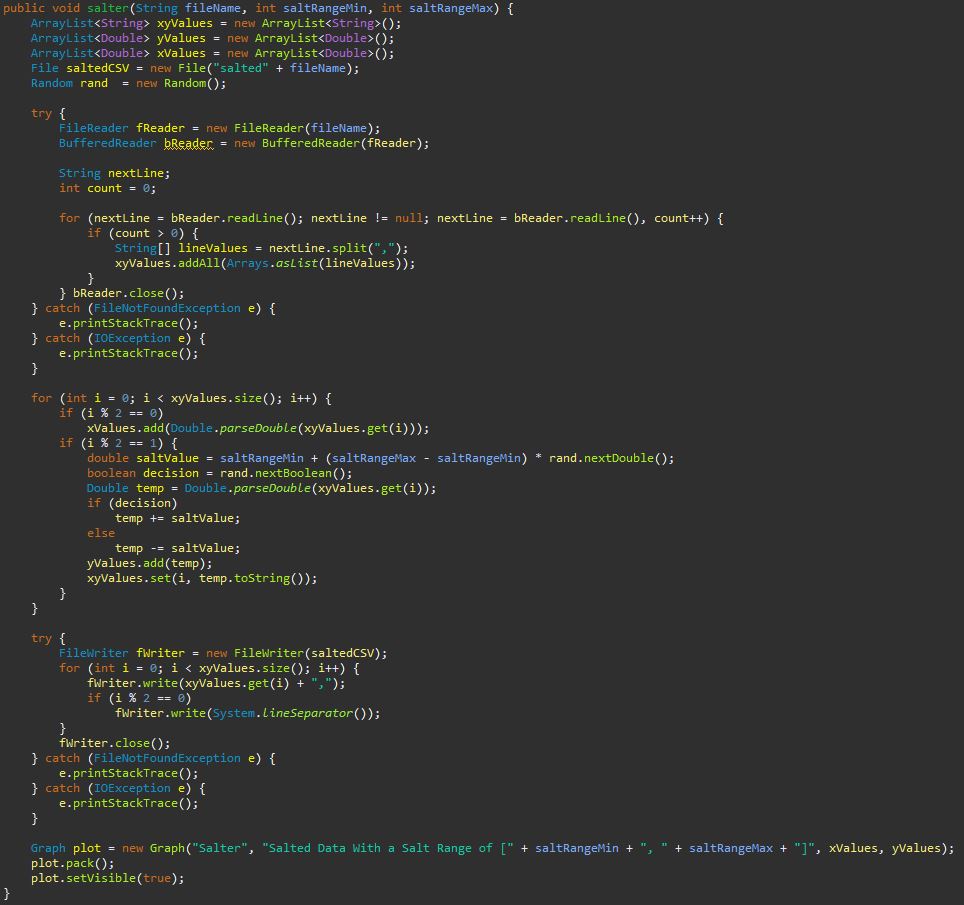
Discussion…………………………………………………………………………………..……11

**Methods**

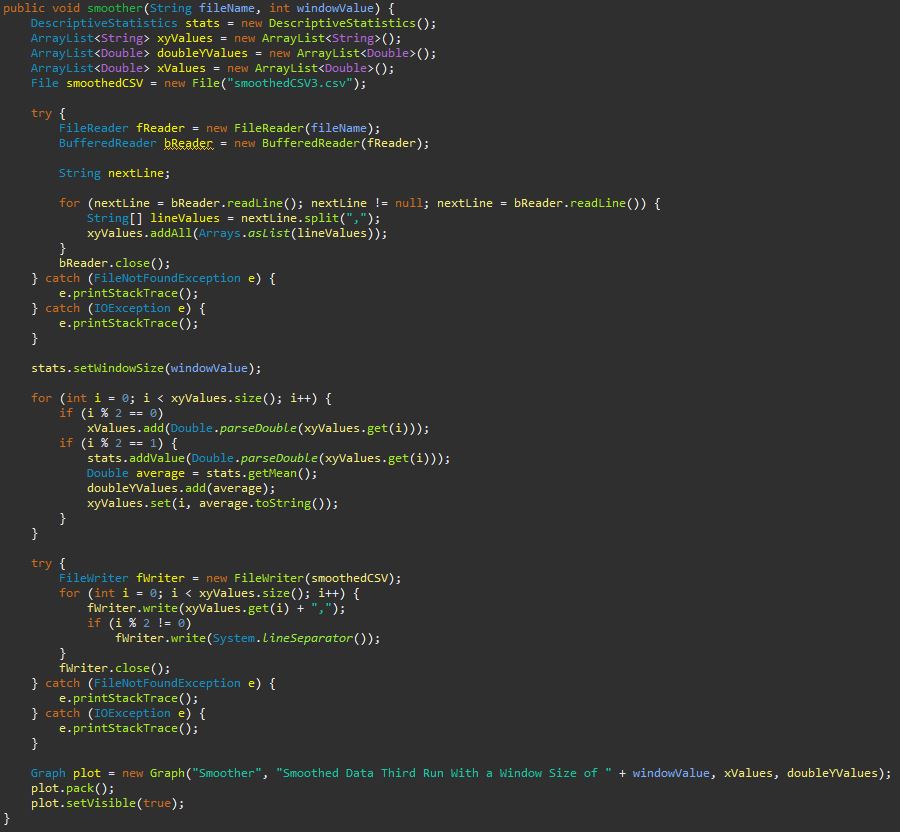
Plotter



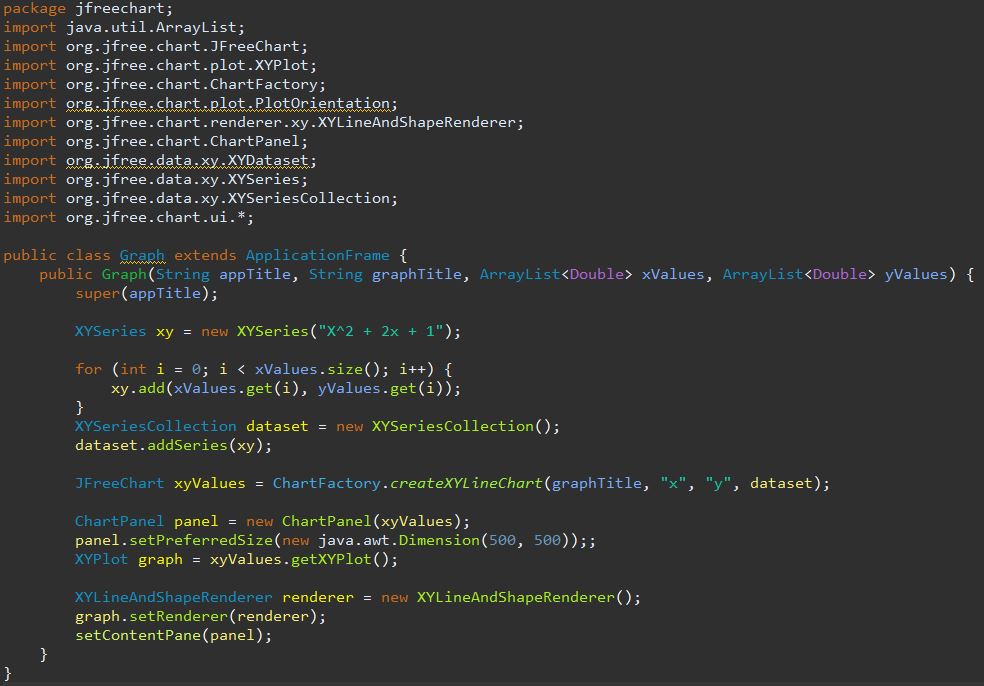
Salter



Smoother



Graph (Class/Method)



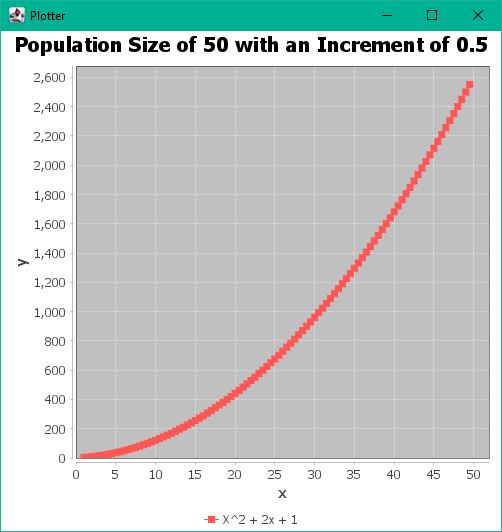
**Small Scale Trial**

Function Used:

Plotter

* Data generated using a population range from [1, 50] with an increment of 0.5

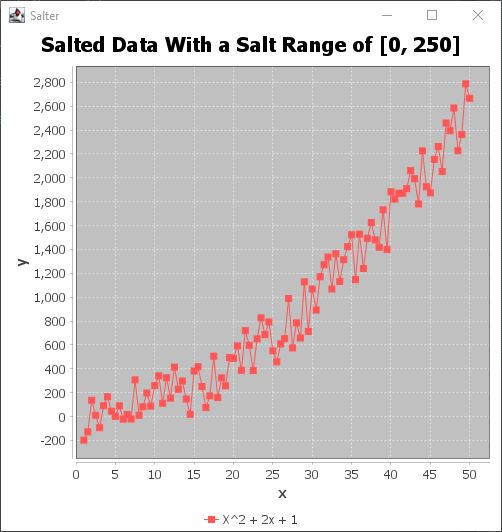




Salter

* Data salted with a salt range from [0, 250]

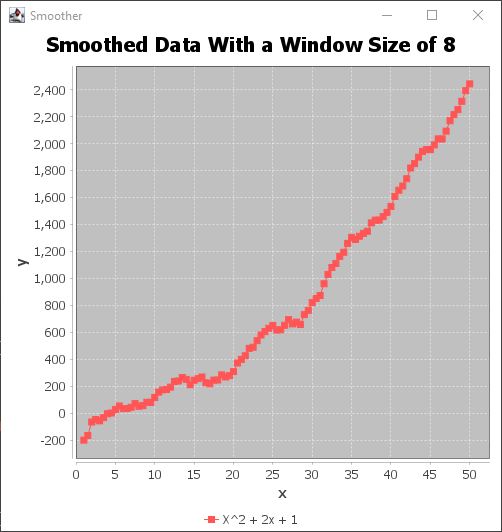




Smoother

* Data smoothed with a window size of 8

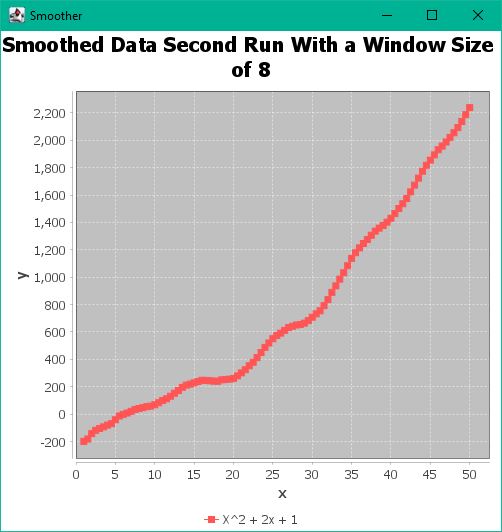




Smoother (Run 2)

* Smoother ran on first set of smoothed data with a window size of 8

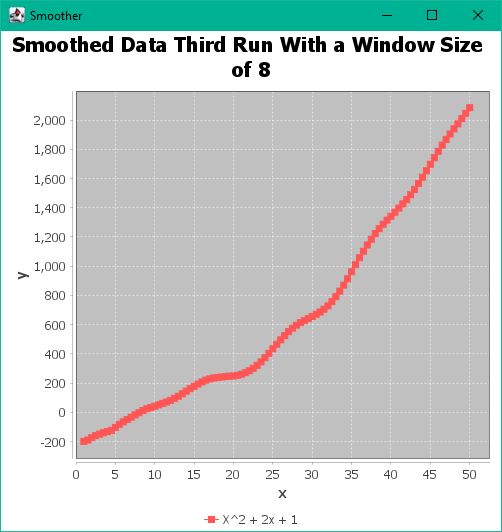




Smoother (Run 3)

* Smoother ran on second set of smoothed data with a window size of 8





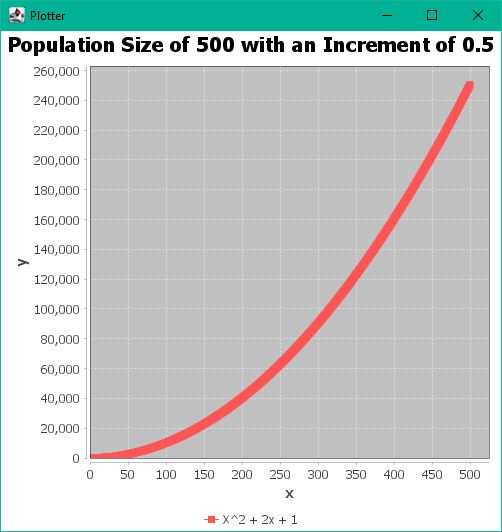
**Larger Scale Trial**

Function Used:

Plotter

* Data generated using a population range from [1, 500] with an increment of 0.5

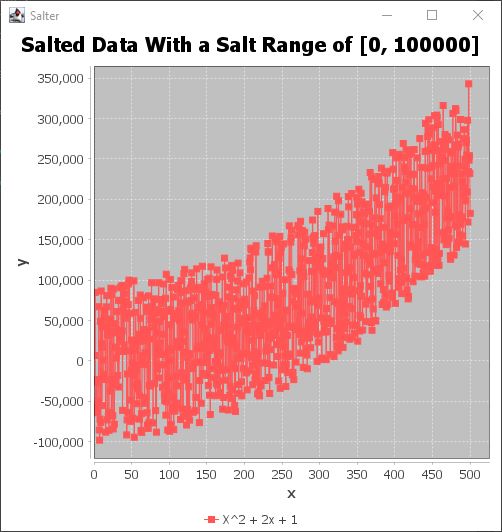




Salter

* Data salted with a salt range from [0, 100000]

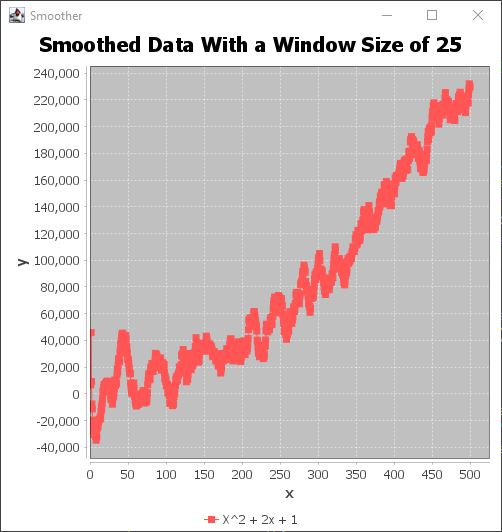




Smoother

* Data smoothed with a window size of 25

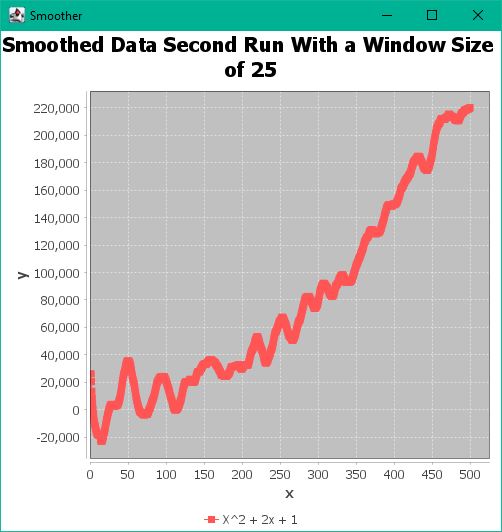




Smoother (Run 2)

* Smoother ran on first set of smoothed data with a window size of 25

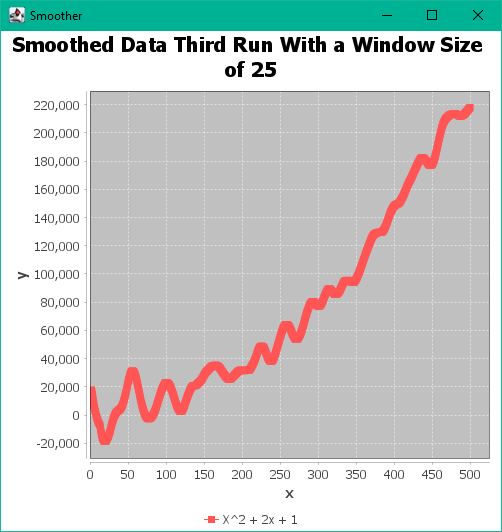




Smoother (Run 3)

* Smoother ran on second set of smoothed data with a window size of 25





**Discussion**

As the third version of the plotting, salting, and smoothing program, the goals of this program were the same as the Java and MATLAB iterations. The function used to generate the data was the same, x2+2x+1, and CSV files were once again used as the medium for the data. Much of the program structure is similar to the original Java version such as the code to read from and write to CSV files as well as the logic for the salting and smoothing methods. The difference in this case however, is that the program was completed using Java, the Apache Commons Math Library API, as well as the JFreeCharts graphing library.

Program Structure

Before getting into the details of the core methods of the program, the Graph class and method deserve some explanation. It takes parameters including an app title, a graph title, an ArrayList of Doubles to represent x-values, and an ArrayList of Doubles to represent y-values. The method starts by creating a new *XYSeries* object with a title that represents the algebraic function being used, x2+2x+1. This is followed by looping through the x and y-values and adding each pair to the XYSeries. Next, an *XYSeriesCollection* object is created and the *XYSeries* is added to the collection. An *XYLineChart* is then created using the graph title parameter, “x” and “y” label for the axes, and the *XYSeriesCollection* as the data set. The method then moves on to make a new *ChartPanel* object passing the *XYLineChart* to it and setting the panel size to 500x500 pixels. The *XYLineChart* is then used to make an *XYPlot* which is followed by the creation of an *XYLineAndShapeRenderer*. Finally, this is used to set the *XYPlot’*s renderer and the *ContentPane* is set to the aforementioned *ChartPanel*.

The program begins with the plotter method that takes parameters such as a file name, minimum and maximum value for the range of the x-values, and an interval to specify how much each x-value should be incremented by. Two ArrayLists are then created to store x-values and y-values for later on in the method when the data is used to create a graph. An array of doubles is created to represent the algebraic function that has been used to generate the data which is then passed to a *PolynomialFunction* object that is part of the Apache library. The program then follows the same structure as the original Java CSV program where it loops from the minimum up to the maximum x-value using the interval to increment each time, evaluates the function with each x-value, and writes the values to a CSV file. The main differences here are that instead of using built-in mathematical operators to carry out the calculations, the *value* method of the *PolynomialFunction* is used instead. The ArrayLists of x and y-values mentioned previously also add each x and y-value from the loop. The final segment of the plotter method is the graphing aspect where a Graph instance is created with the application title of “Plotter”, a graph title that corresponds to the population size, and the ArrayLists of x and y-values. The graph is then packed and the visibility is set to true.

The next method in the program is the salter that takes a file name as well as minimum and maximum values for the salting range. The salter method starts by declaring an ArrayList to store x and y-values read from the CSV file and two separate ArrayLists to store the x-values and salted y-values respectively. Just like the regular Java CSV program, a FileReader and BufferedReader are used to read the data from the CSV file. The program then loops through the x and y-values, adds each x-value to the ArrayList of x-values, and salts each y-value using the provided salting parameters. The salting aspect is identical to the previous Java program in which a random salt value and Boolean are generated and depending on the value of the Boolean, the salt value is either added to or subtracted from the current y-value. Each salted y-value is then added to the ArrayList of y-values and then used to replace the previous y-value in the ArrayList of x and y-values. A FileWriter is then used to write the x and salted y-values to a new CSV file and a Graph instance is created with the application title of “Salter”, a graph title that corresponds with salting range, and the ArrayLists of x and salted y-values.

The final method in this program is the smoother which only takes a file name and a window value as parameters. The smoother starts out by creating a new *DescriptiveStatistics* instance, an ArrayList to store x and y-values that are read from the CSV file, and two separate ArrayLists to store the x-values and smoothed y-values. This is followed by reading the salted data from the CSV file and storing the values in the ArrayList of x and y-values. Before entering the loop to smooth the data, the *DescriptiveStatistics* object’s window size is set using the window value parameter. The smoother then loops through the x and y-values, adds each x-value to the appropriate ArrayList, and adds each y-value to the *DescriptiveStatistics* object. While still in the loop, the rolling average is calculated using the *getMean* method and that average is then used to replace the previous y-value in the ArrayList of x and y-values and it is also added to the ArrayList of smoothed y-values. Just like the salter method, a FileWriter is used to write the x and smoothed y-values to a new CSV file and a Graph instance is created. The Graph is given an application title of “Smoother”, a graph title that corresponds with the order of the smoother run and the window value, as well as the ArrayLists of x and smoothed y-values.

Trials and Observations

To maintain consistency across all of the versions of this program, the Java Apache and JFreeCharts trials used the same polynomial function and data parameters as the previous two CSV programs. The first trial used a population with a range of 1 to 50 with an increment of 0.5 for a total of 100 x-values. The plotter performed as expected and created a graph of the x and y-values in real-time. While there was not a significant decrease in the number of lines of code between this program and the original Java version, it was more convenient to use the *PolynomialFunction* to represent the equation and handle the calculations when plotting the data. Following the plotter, the salter method salted the data with a salt range of 0 to 250. The chart that was generated showed the usual upward trend with the random vertical scattering of the data points. The smoother method was then executed using a window value of 8. Similar to the original Java version and the MATLAB trials, the first graph that was created from the smoother method retained some of the random dispersion of data points but was considerably smoother than the salter graph. To stay consistent with the other versions of this program, the smoother was run two more times. The resulting graphs seemed to resemble a combination of the results from the regular Java program and the MATLAB iteration. For example, the second and third runs produced curves that were definitely smoother but maintained some visible variability like the MATLAB graphs. However, what was interesting is that the curves did not flatten out towards the end like those of the first Java iteration.

The second trial was run using a population with a range of 1 to 500 with an increment of 0.5 for a total of 1000 x-values. Once again, the resulting graph was essentially identical to the plotter graph from the first trial except this was scaled to accommodate the points that increased up to about 250,000. The salter method was executed with a salt range of 0 to 100000. As seen from the previous versions of this program, the salter graph produced a generally upward sloping curve with large random spikes in the data. Next, the smoother was run on the salted data with a window size of 25 and created a graph that reduced the severity of the spikes in the data but appeared not to be nearly as effective as the first smoother run in the larger scale regular Java program. As explained in the MATLAB report, this may be due to a similar reason where the level of detail in the Excel graphs is simply lower than that of the JFreeCharts graphs. After the smoother was run two more times, the final graphs demonstrated similar outcomes when compared to the second and third smoother runs in the larger scale MATLAB trial. However, the main difference in this trial was that the smoother appeared to struggle with smoothing the large data spikes at the beginning of the curve. Nonetheless, the smoothed graphs were still much closer to the original curve than that of the salted graph.

Learning Process and Conclusions

Since this version of the program used external libraries to achieve the specified goals, there was somewhat of a learning curve. For example, the Apache Commons Math Library has an immense amount of different Java classes and associated methods that serve as powerful tools for other areas of research. Some of these include high-level statistical analysis, various applications of calculus, machine learning, among many others. This proved to be relatively overwhelming at first since finding exactly what tools would be useful in this application required a good amount of sorting and reading through Javadocs. Once some of the relevant tools were found such as the *PolynomialFunction* and the *DescriptiveStatistics* it became much easier to learn how to implement them through use of the available Javadocs and various online examples.

The JFreeCharts library required a similar level of dedication to explore what was available, what tools were required for this specific program, and how exactly to implement them. It was slightly easier to find this information since it was obvious that some sort of chart that plotted x and y-values was needed. It was also rather useful that the developers of the JFreeCharts library also made the naming conventions straightforward such as the *XYSeries* and *XYPlot*.

Overall, the programming that was required to complete this version was almost identical to the original Java implementation but integrated some key components from the Apache Commons Math Library and the JFreeCharts Library that not only made it easier, but also more professional. The rolling average functionality helped reduce the number of lines of code as well as the complexity of the smoother method and the JFreeCharts graphing capabilities helped synthesize all of the components of the program. Rather than manually opening the CSV files that contained each data set and creating the graphs in Excel, the charts could be automatically generated in real-time to visualize the performance of the program. While the requirements for this program were relatively simple, the lessons involved transcend into other real-world applications. For example, the ability to use other libraries and APIs is something that is endlessly useful as the years of research and development that have been put into such projects by other people can be leveraged to solve many different problems and innovate new ideas at a much faster rate.