Plotter, Salter, & Smoother Program

Lilith Carpenter

May 5, 2025

CSCI-3327

Byron Hoy

PSS 1: Coding in Java 'CSVFilePlotter'

The first class I created to approach this program was the `Exporter` class. This class contains methods to write 2 columns of numbers to a .csv (comma separated value) file representing x and y values of a function, with the numbers to be determined as arguments of `ArrayLists`.

The Function class would populate the .csv's to be created by the exporter. This class contains a method, `exportQuadratic`, to output a .csv file containing the x and y values of a quadratic function using the export method. The .csv file can then be opened in a program like Excel to visualize the data in a line graph, which is pictured below.

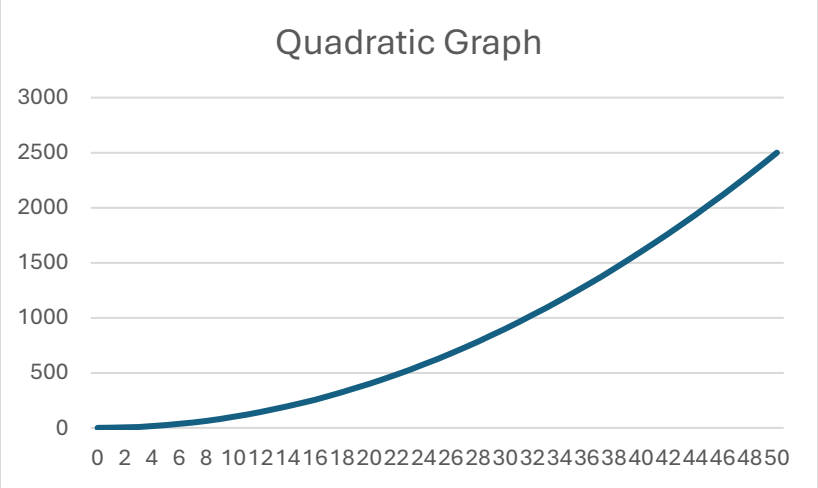

Figure 1: Quadratic Equation Plot

The `Salter` class contains the salt method, which takes a dataset's x -value and 'salts' the y-value data based on bounds determined by arguments. In this context, salting refers to adding random positive or negative values to a previous value to disguise trends of the original data. To achieve this, the salt method takes a .csv file as an argument and looks at each individual row to change only the y-values.

The `Smoother` class instead contains the smoother method, which aims to do the opposite of Salter. This method takes a dataset and creates a simple moving average of each point in the set, accepting an integer argument determining the size of the average's window. It only accepts odd numbered window sizes, making sure the data will be even on both sides, and will throw an `IllegalStateException` otherwise. This effectively smooths the data to become closer to the trendline of the original graph. Using the previously described `Exporter` class I created these 2 graphs in Excel.

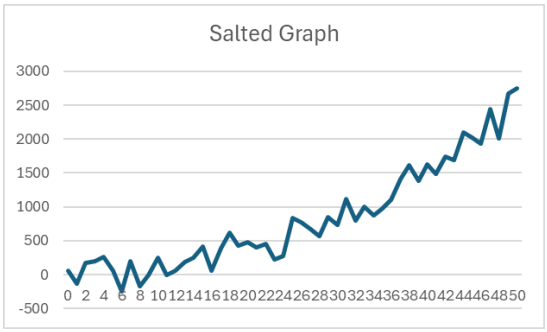

Figure 2: Salted Graph Plot with bound1 = 300

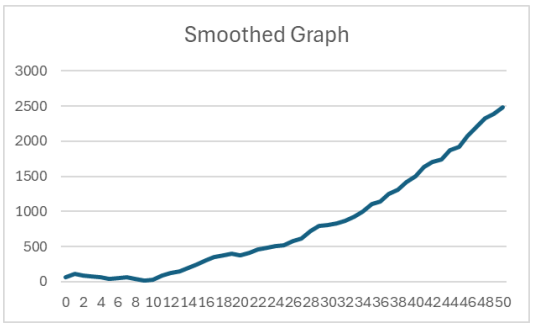

Figure 3: Smoothed Graph Plot with windowValue = 9

I was able to make reasonable changes to the data based on changing my parameters. Reducing the size of the salt method's bounds made less drastic changes to the data, reflecting the fact that a smaller amount of random numbers were used. Using lower numbers for the smooth method's window size created less accurate smoothed results, which reflected that the average trendline would be inaccurate with fewer samples.

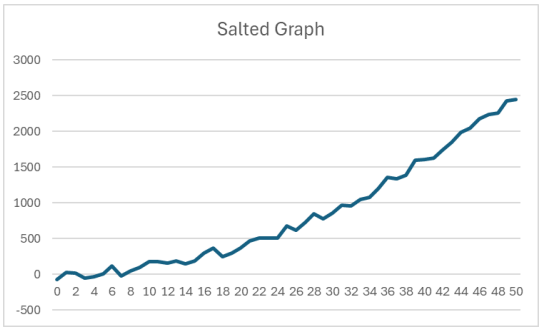

Figure 4: Salted Graph Plot with bound1 = 80

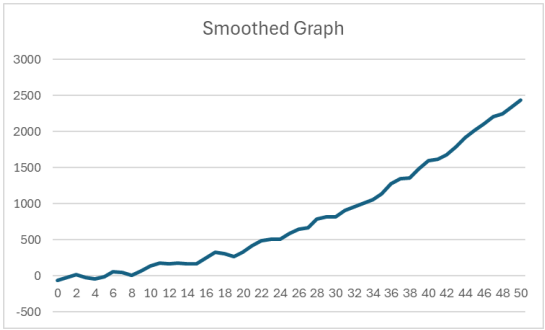

Figure 5: Smoothed Graph Plot with windowValue = 3

PSS 2: Coding in GNU Octave

The first feature in Octave I learned to use was its arithmetic calculation capabilities. The program allows for fast computation of arithmetic calculations as command statements and stores the most recent command's output in a variable called `ans`, for answer.

```
>> 2+4
ans = 6
>>
```

Figure 1: Octave's command prompt format

Then, I began to learn about Octave's plotting functionality. To make use of this capability, first a variable vector must be created that describes the bounds of the vector and by what number it increases by. This can be done using a colon operator between the bounds and inserting the number to be increased by between them. By using the plot function with this vector as the x-values and sin(x) as the y-values, I output the following graph.

```
>> x = -10:0.1:10;
>> plot(x,sin(x));
>>
```

Figure 2: Command prompts for graphing

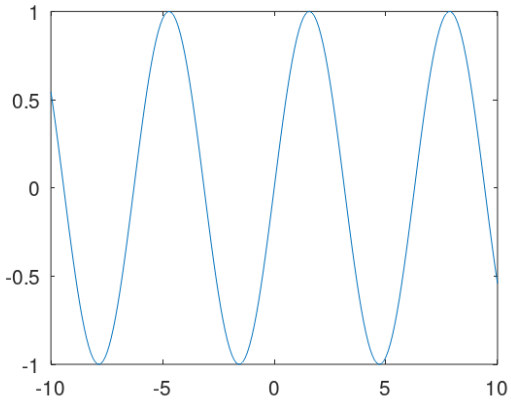

Figure 2: Resulting sine wave

Using these 2 functionalities, I was able to replicate the graphs created by my .csv outputs from my first program, the originally plotted function, the salted graph, and the smoothed graph. Additionally, I learned how to use script files in Octave to create the salting and smoothing elements using more than just command lines. In

script files, it allows you the same functionality of executing operations on large datasets, but with Octave's own syntax.

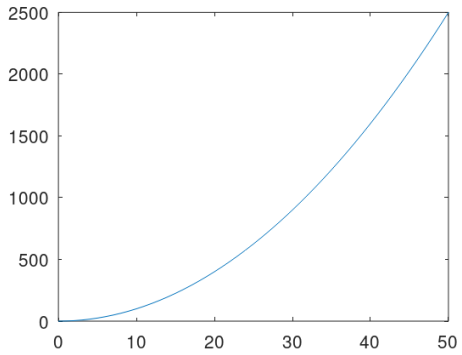

Figure 4: Quadratic Graph in Octave

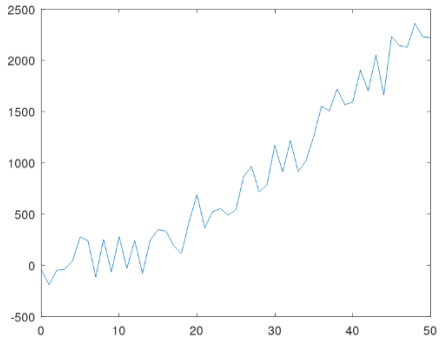

Figure 5: Salted Graph in Octave

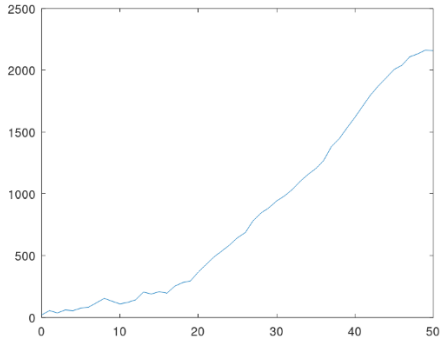

Figure 6: Smoothed Graph in Octave

By creating this program in Octave, I was able to learn how to use the built-in command window provided by Octave and many other languages to create simple, single line, computations to be completed immediately. I also learned how to use script files in Octave, allowing for more complex manipulation of data by creating multiline, more complicated programs. The resulting graphs were much easier to visualize with graphs as opposed to PSS 1's raw .csv file output, which makes checking my plotting function for errors much simpler than when done with Java alone.

PSS 3: Coding in Java using JFreeChart and Apache Statistics Library

Before working on any other classes, I created the `Exporter` class in order visualize each of the datasets to be created by the plotter, salter, and smoother. The `Exporter` class contains only a single method, export, to create a window containing an XY line graph of the given dataset. The export method takes an underlying JFreeChart `XYSeriesCollection` (a collection of series with corresponding x and y points) as the dataset and a `String` as the graph's title. This method formats each output graph as a 600 by 800-pixel window.

Then, I created the `Function` class to populate points in a dataset. This class is home to a method that populates `XYSeriesCollections` with a math function. The methods take a single integer as an argument to determine the domain of the function, then returns an `XYSeriesCollection` with points of the quadratic function. Using this class in conjunction with my `Exporter` class, I created the graph pictured below.

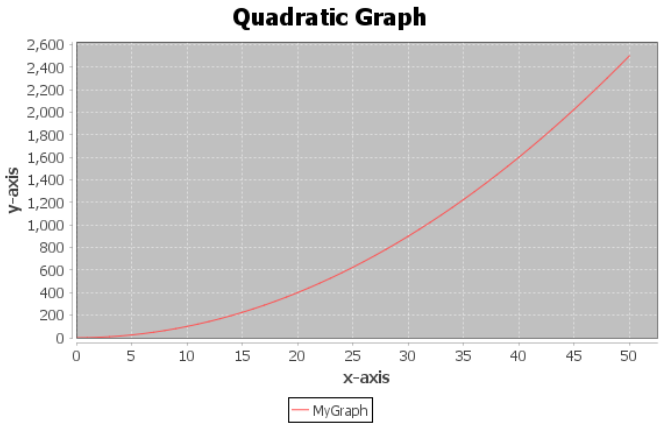

Figure 4: Quadratic Equation Plot

Next, I created my `Salter` and `Smoother` classes. The `Salter` class only contains a salting method, which namely salts the data. The `Smoother` class contains the smoothing method, doing the opposite. Using these two classes with my original output quadratic function, I was able to create the following two graphs, this time in Java Swing windows.

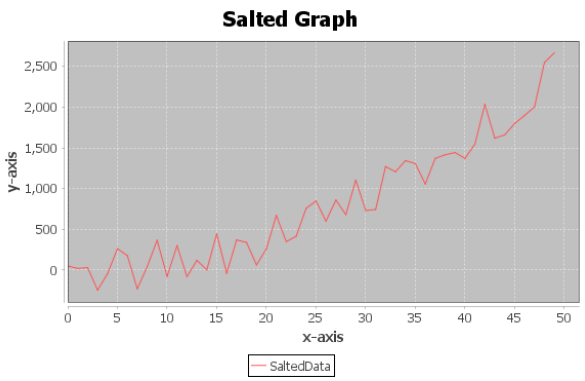

Figure 2: Salted Quadratic Plot

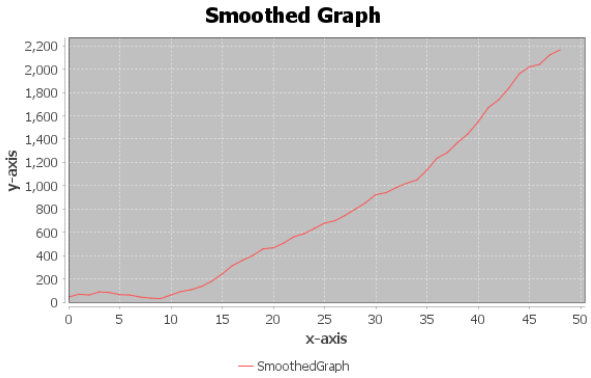

Figure 3: Smoothed Quadratic Plot

Lastly, I created my `Main` class to run the application. It contains the main method of the program, which by default contains code to output the following graphs described here. During the process of creating this program, I ran into several issues regarding dependencies from the external libraries, caused by directly downloading the .jar files in the library. This was remedied by converting my project into a build that used Gradle to handle these external libraries and their dependencies instead. Though learning to use these two external libraries and Gradle was time consuming, it being my first interaction with them, using them to create graphs was also considerably faster than PSS 1's output of raw .csv files and instead using Excel to visualize the data.