Plotter Work Report

By Jake Cubernot

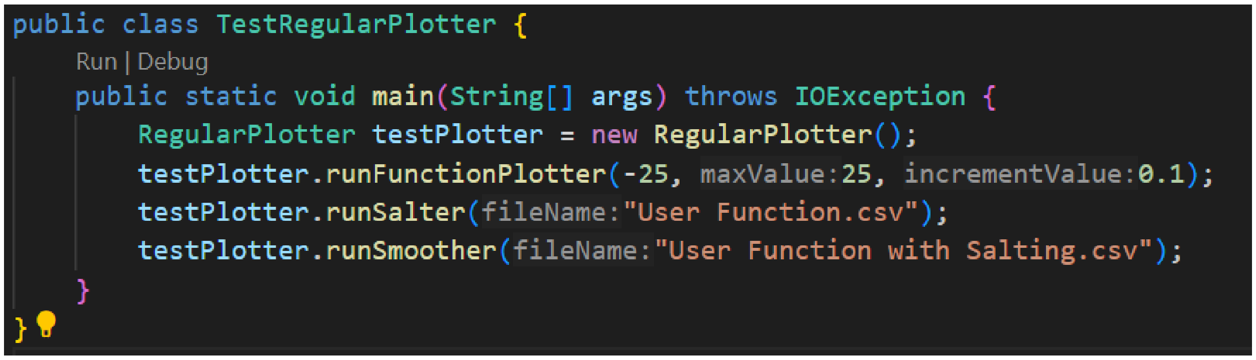
Table of Contents

Regular Plotter

The regular plotter is a simple Java class, *RegularPlotter*, that takes a given function and converts it to a comma-separated values, CSV, file. With this file, one can put it into a spreadsheet software, like Excel, and create a graph from the data collected by the CSV file.

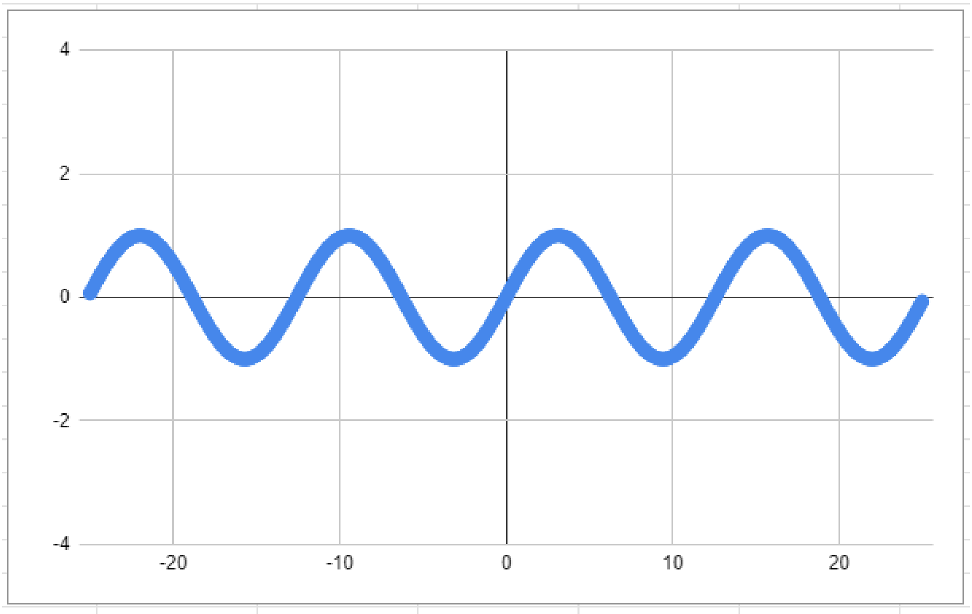
In each plotter, the function $y = sin(\frac{x}{2})$ is used to test all of the plotters' functions.

The regular plotter uses a tester function in which the function has an x range from $x =- 25$ to $x = 25$ with each x-value incrementing by 0.1:

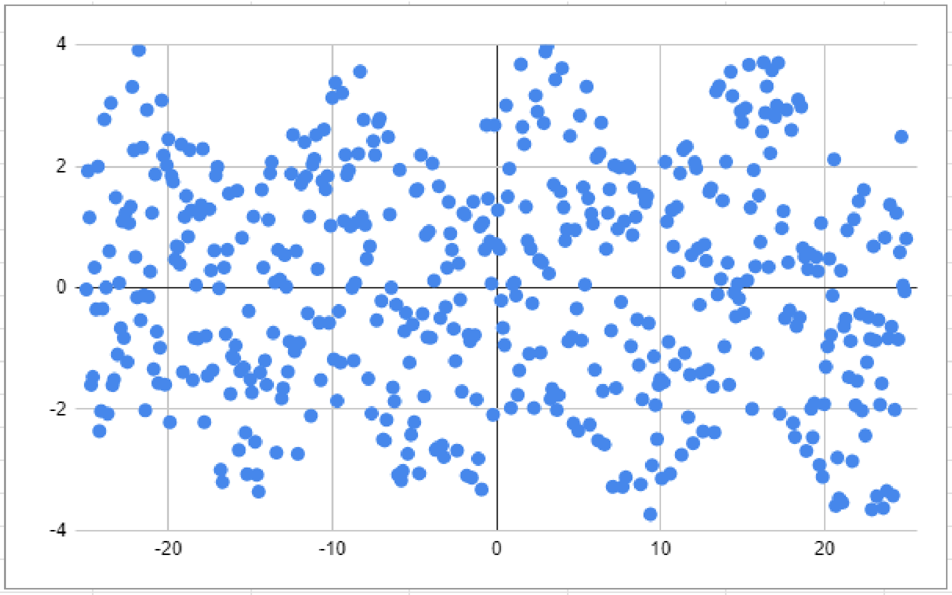


```
public class TestRegularPlotter {
    Run | Debug
    public static void main(String[] args) throws IOException {
        RegularPlotter testPlotter = new RegularPlotter();
        testPlotter.runFunctionPlotter(-25, maxValue:25, incrementValue:0.1);
        testPlotter.runSalter(fileName:"User Function.csv");
        testPlotter.runSmoother(fileName:"User Function with Salting.csv");
    }
}
```

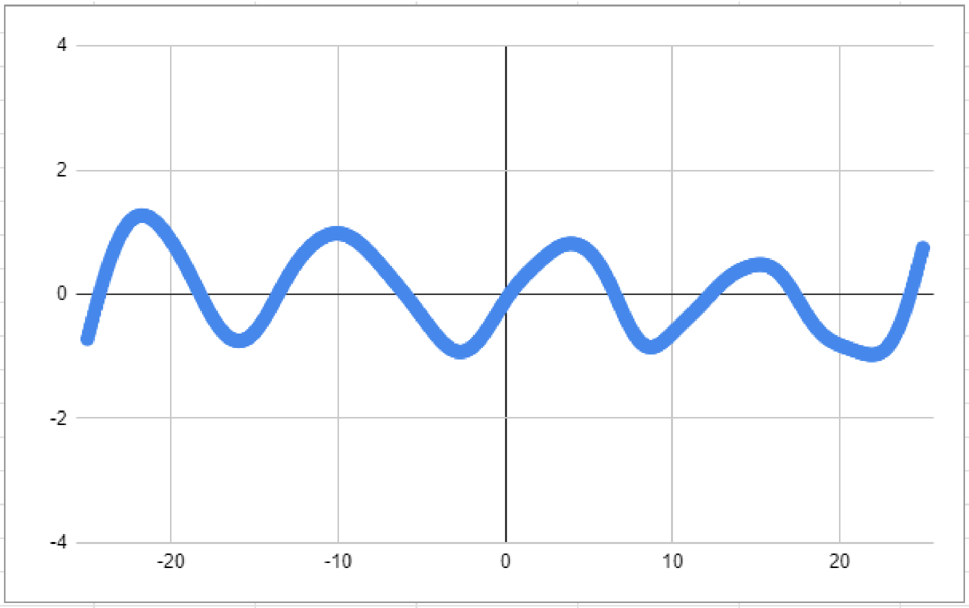
After running the *runFunctionPlotter* function and graphing the CSV file data results into Excel, it gives this graph:



After running the *runSalter* function using the data from the previous function and graphing the CSV file data results into Excel, it gives this graph:



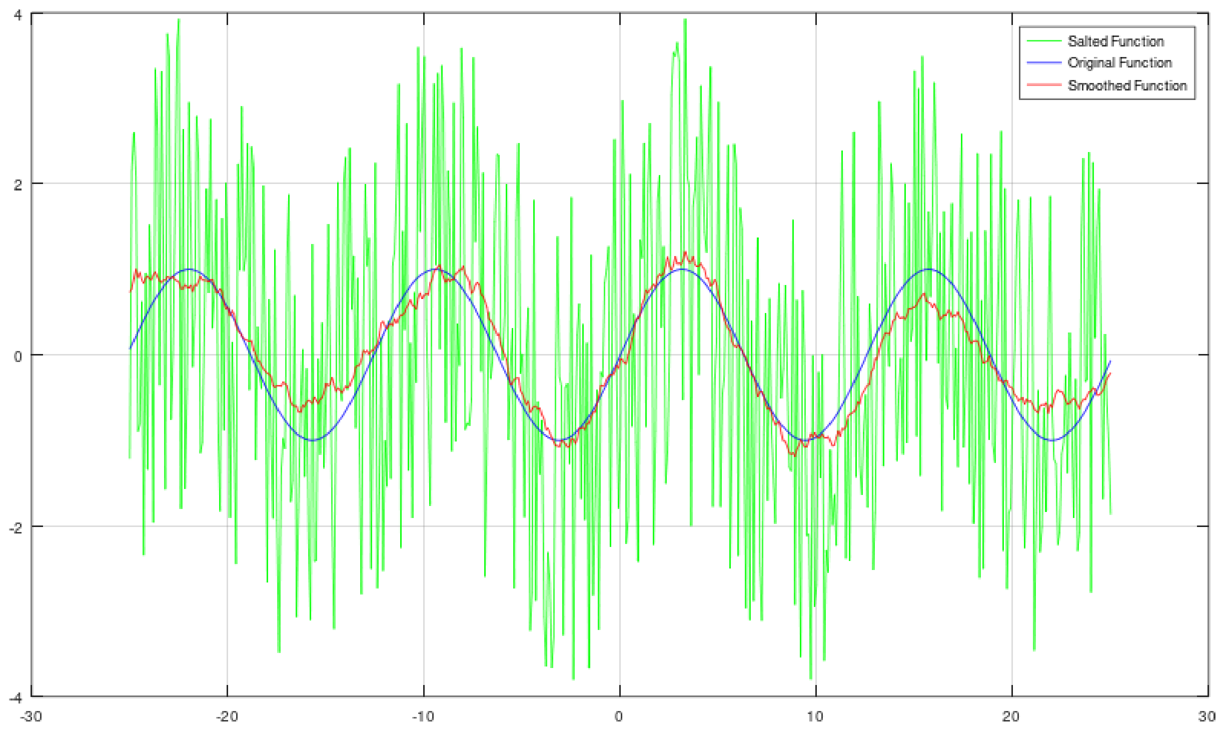
After running the *runSmoother* function using the data from the previous function and graphing the CSV file data results into Excel, it gives this graph:



As this is only a simple smoother function, the function had to be repeated 200 times to get the perfect smoothing on the $y = sin(\frac{x}{2})$ function.

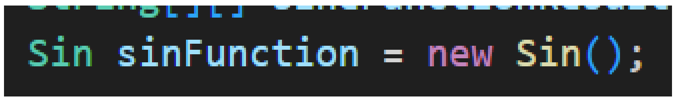
Octave Plotter

The Octave plotter is another approach to the previous work in which the math program, Octave, is used to simplify the work needed, like in the Java version, and it can automatically plot and graph the functions on the spot.
The program is capable of displaying three functions at the same time. Though Octave does the same thing, the smoothed function turned out to be more rough than the previous one using the Java function, but this could probably be improved with a better understanding of the program:

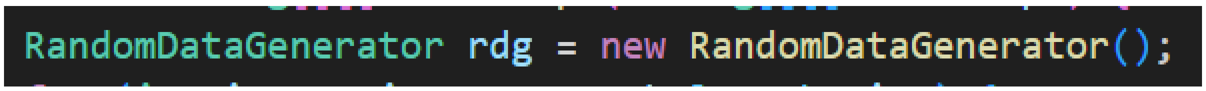


Commons Math Plotter

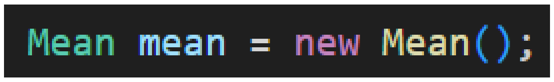
Similar to the regular plotter, Commons Math creates the three functions but using functions within the Commons Math jar file, *jfreechart-1.5.0*.
With the Commons Math jar file, the *Sin* class was implemented to help plot the function, $y = sin(\frac{x}{2})$. The class *RandomDataGenerator* was used to create another approach to salting the function. Finally, Commons Math had a very useful *Mean* class to easily find the averages when plotting the smoothed function:



```
Sin sinFunction = new Sin();
```

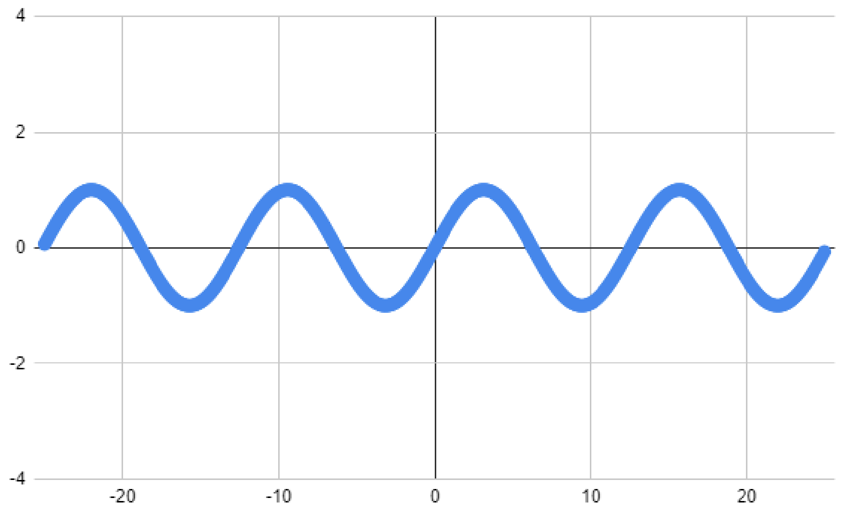


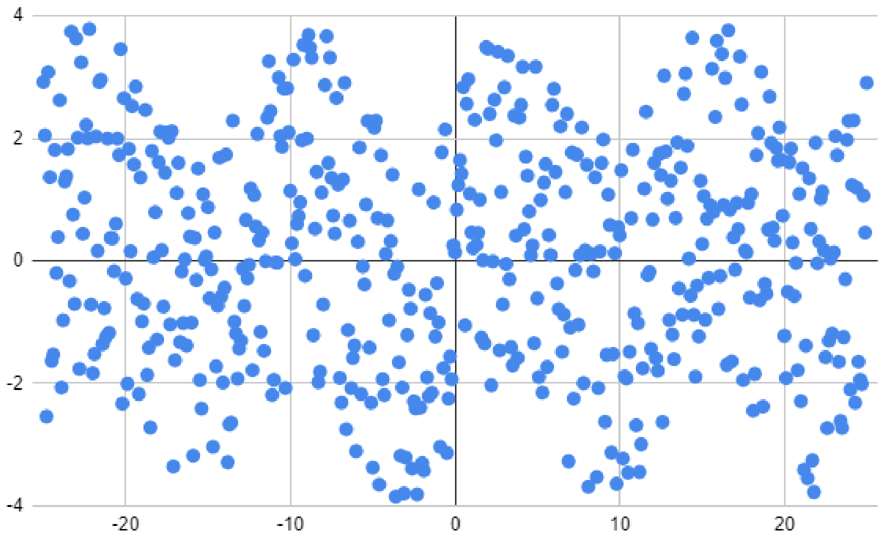
```
RandomDataGenerator rdg = new RandomDataGenerator();
```

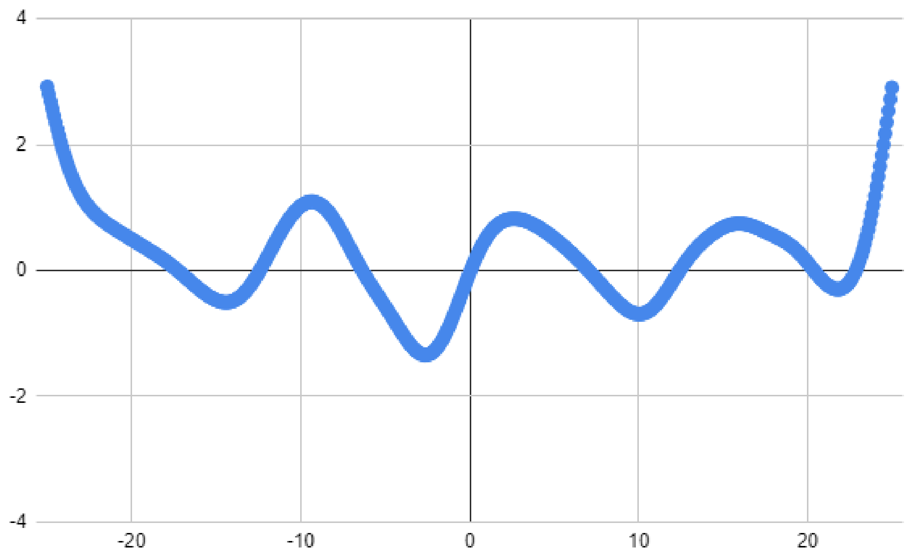


```
Mean mean = new Mean();
```

The results are very similar to the previous Java program, but some of the smoothing may seem a bit different as the random salting can change those final results:

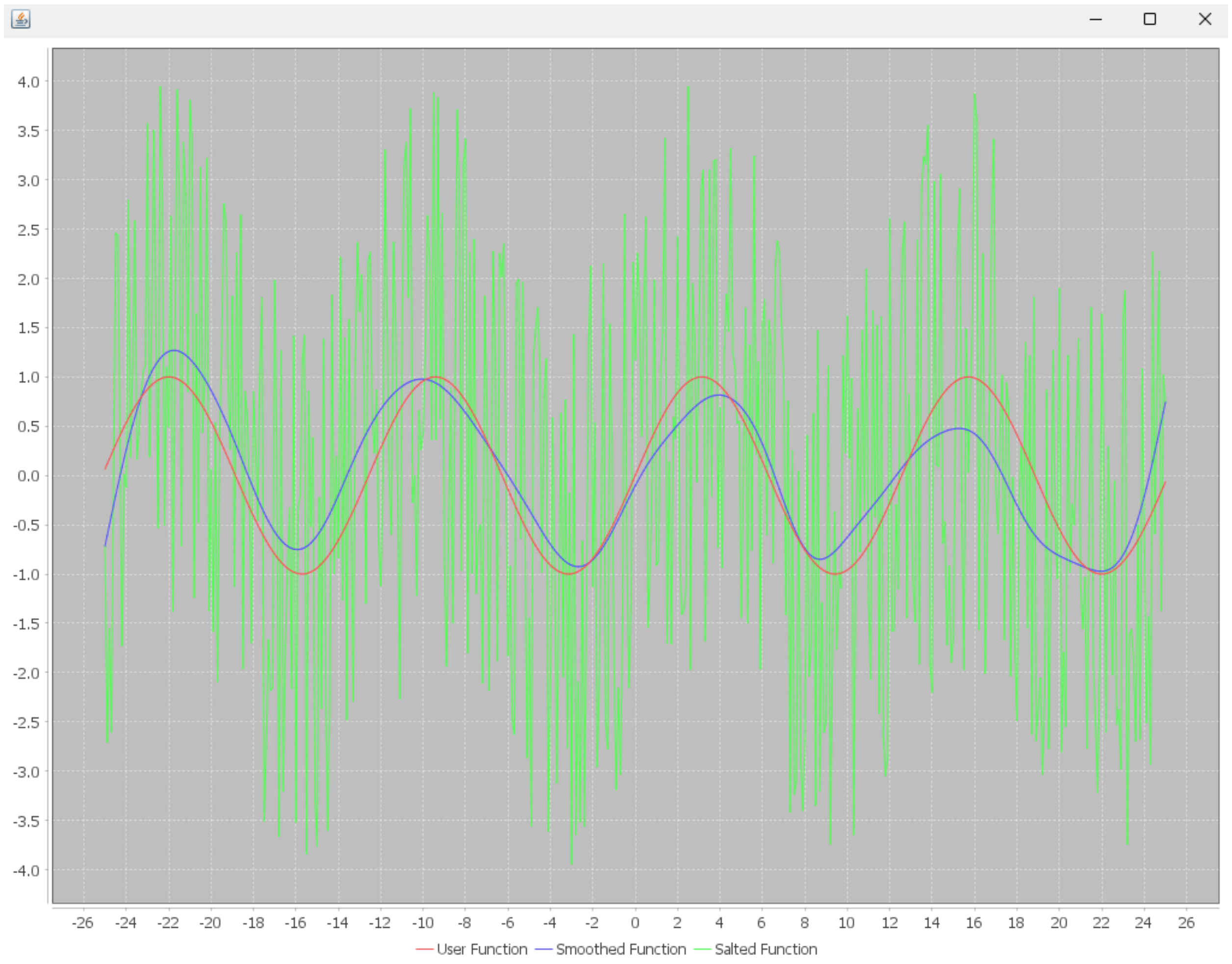

4
2
0
-2
-4
-20
-10
0
10
20


4
2
0
-2
-4
-20
-10
0
10
20


4
2
0
-2
-4
-20
-10
0
10
20

JFreeChart Plotter

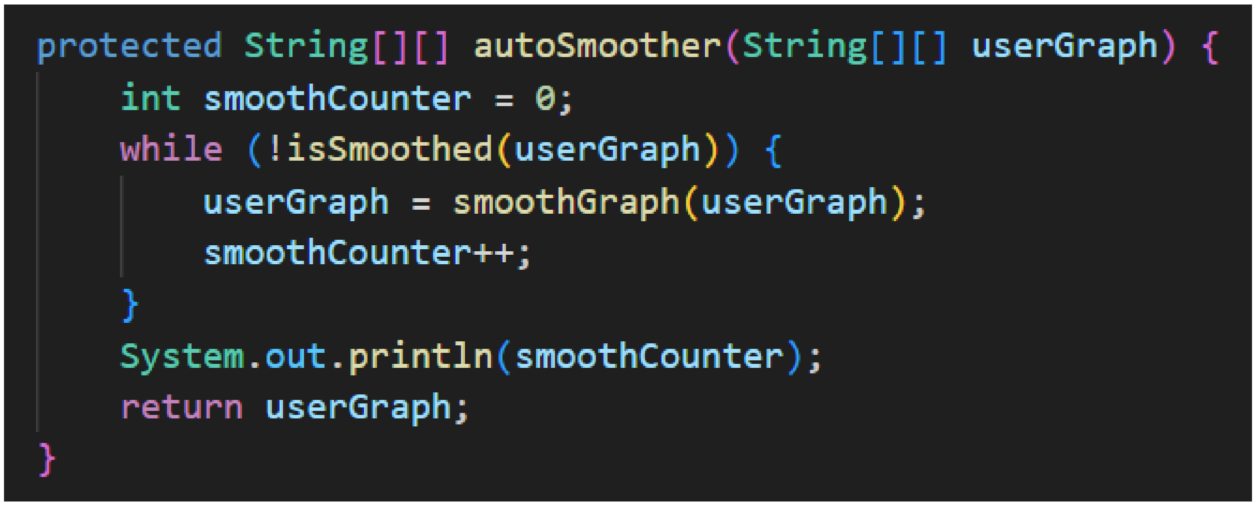
Like Commons Math, this plotter uses the JFreeChart jar file. This class uses functions from the existing regular plotter and puts it into a visually appealing plotter GUI. With the template used, the plotter turned out to be similar to how the Octave plotter is where all functions can be represented on one window. This was a very useful tool to use as it avoids the use of external tools like Excel to graph:



Auto-Smoother Function

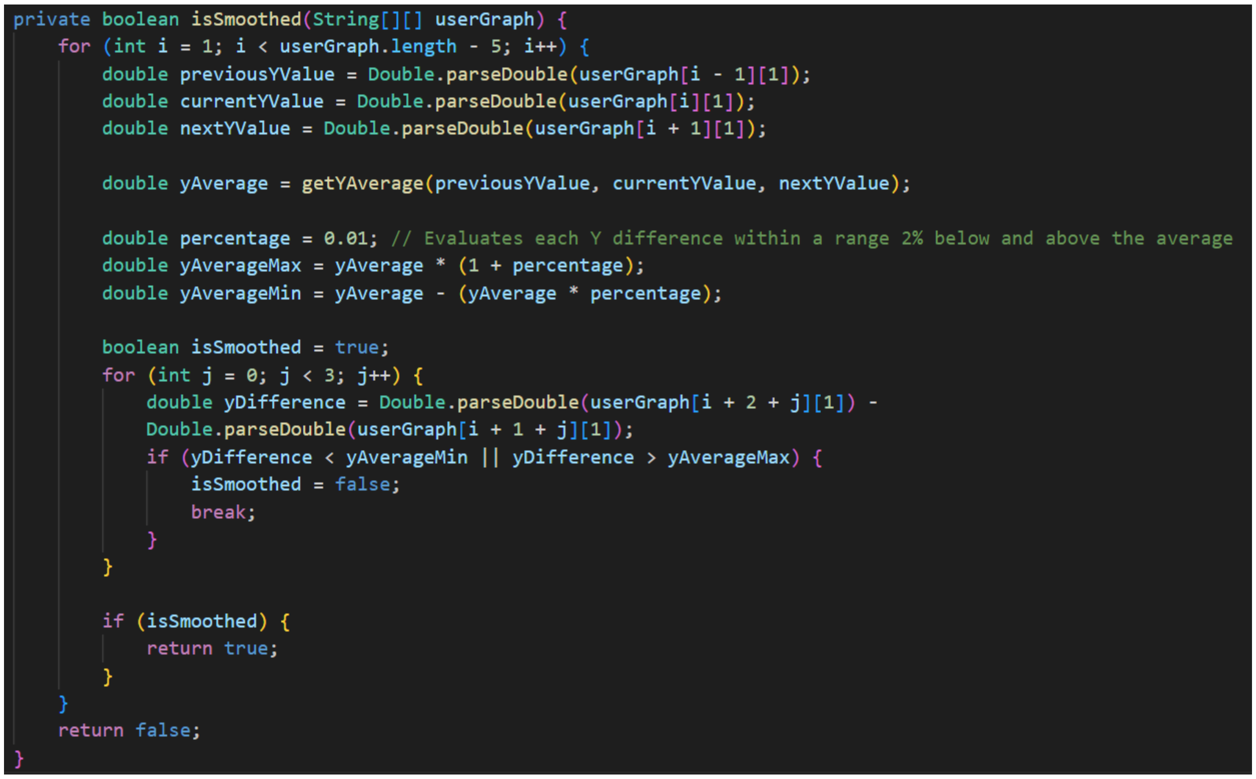
The auto-smoother function is an extension of the plotter function via Java. This function automatically detects when a previously salted function is smoothed enough. Originally, one would have to keep repeating the smoothing function until they thought it was enough, like in the regular plotter where smoothing the graph 200 times seemed to be the perfect amount.

The *autoSmoother* function will simply run through a loop an *N* amount of times until the *isSmoothed* function returns true:



```java
protected String[][] autoSmoother(String[][] userGraph) {
    int smoothCounter = 0;
    while (!isSmoothed(userGraph)) {
        userGraph = smoothGraph(userGraph);
        smoothCounter++;
    }
    System.out.println(smoothCounter);
    return userGraph;
}
```

The *isSmoothed* function first creates a mean average of the Y difference of neighboring values. This mean average Y difference is then compared to the next 3 Y values' difference. If the difference of the compared values are either 2% higher or 2% lower than the mean average Y difference, *isSmoothed* will return false as the graph still needs further smoothing:



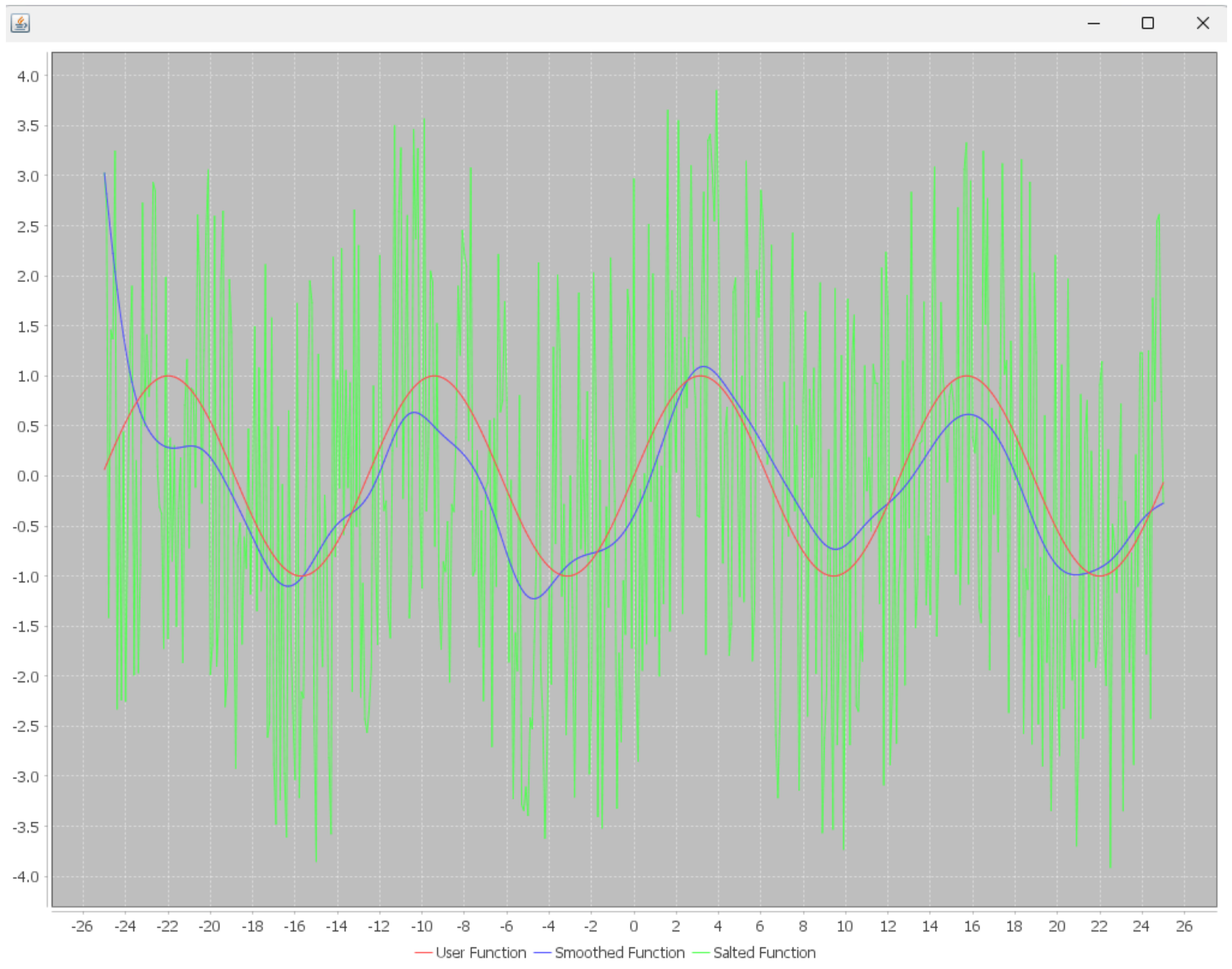
```java
private boolean isSmoothed(String[][] userGraph) {
    for (int i = 1; i < userGraph.length - 5; i++) {
        double previousYValue = Double.parseDouble(userGraph[i - 1][1]);
        double currentYValue = Double.parseDouble(userGraph[i][1]);
        double nextYValue = Double.parseDouble(userGraph[i + 1][1]);

        double yAverage = getYAverage(previousYValue, currentYValue, nextYValue);

        double percentage = 0.01; // Evaluates each Y difference within a range 2% below and above the average
        double yAverageMax = yAverage * (1 + percentage);
        double yAverageMin = yAverage - (yAverage * percentage);

        boolean isSmoothed = true;
        for (int j = 0; j < 3; j++) {
            double yDifference = Double.parseDouble(userGraph[i + 2 + j][1]) -
            Double.parseDouble(userGraph[i + 1 + j][1]);
            if (yDifference < yAverageMin || yDifference > yAverageMax) {
                isSmoothed = false;
                break;
            }
        }

        if (isSmoothed) {
            return true;
        }
    }
    return false;
}
```

Auto-Smoother Function Test Cases

The auto-smoother function was tested using the *JFreeChartPlotter* class as it easily plots the functions and graphs them into a GUI.

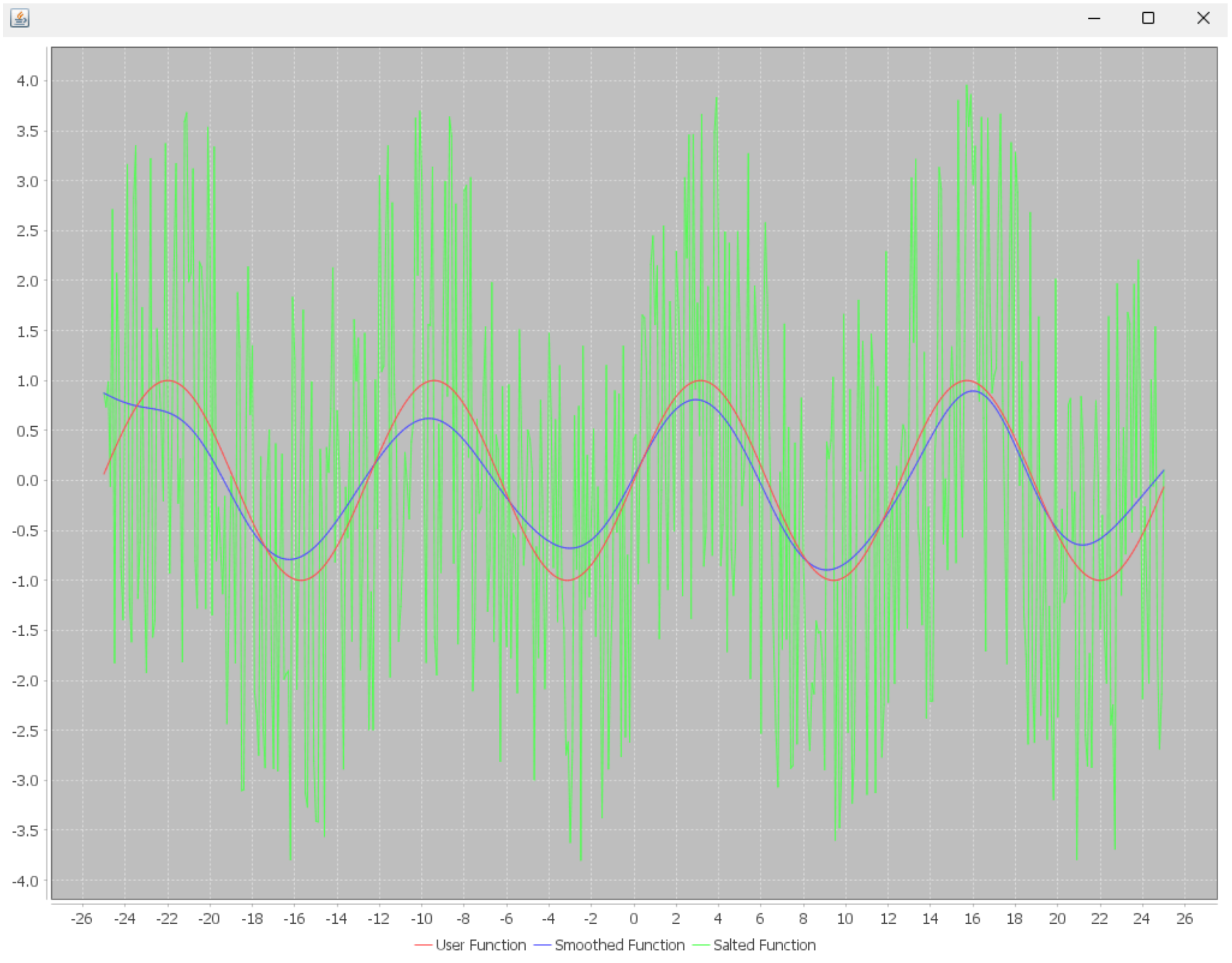
Test Case 1:

In the first test case, the auto-smoother function detected that the graph was smoothed enough after 125 times:



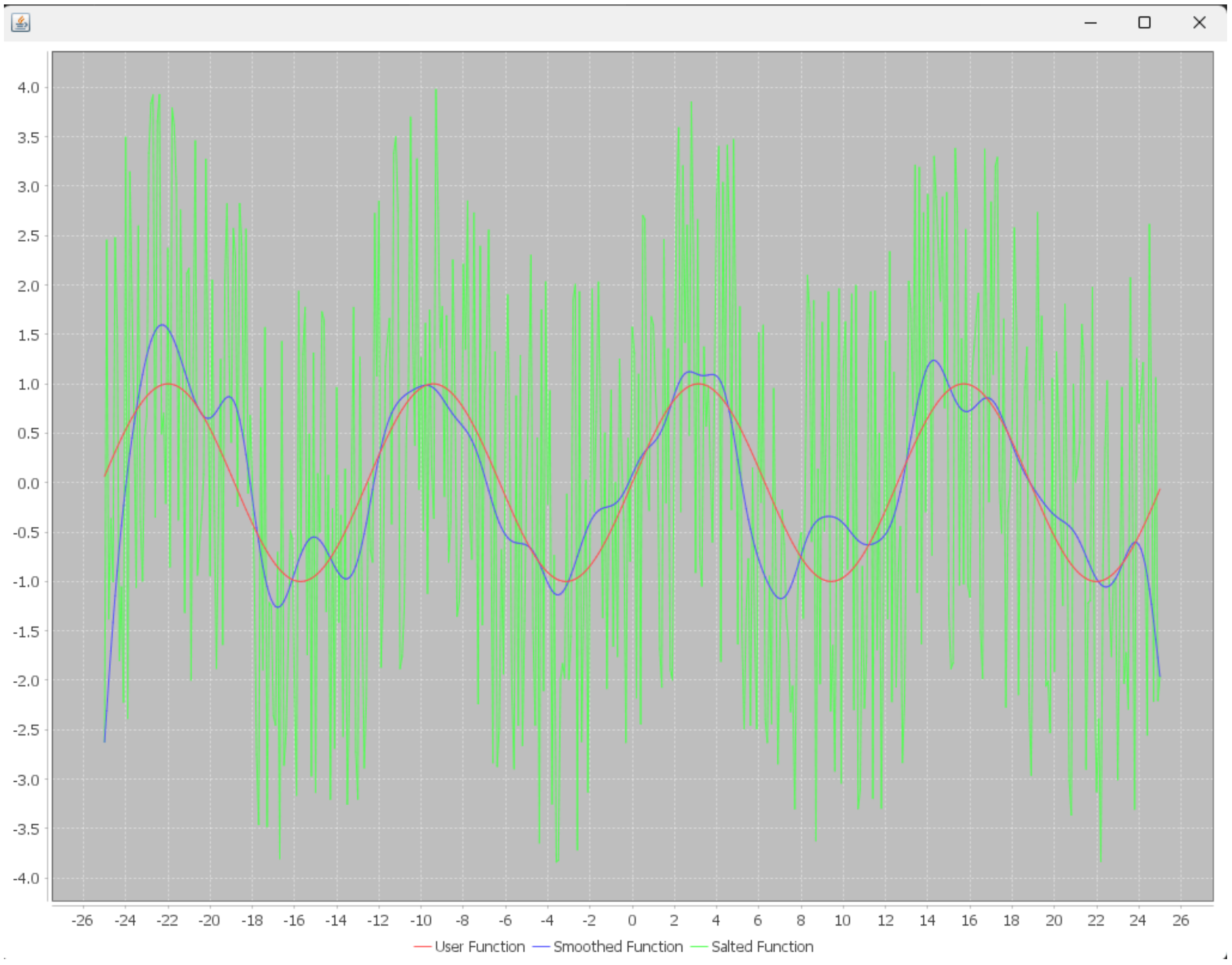
Test Case 2:

In the second test case, the auto-smoother function detected that the graph was smoothed enough after 365 times:



Test Case 3:

In the third test case, the auto-smoother function detected that the graph was smoothed enough after 53 times:



Result:

The auto-smoother function works as expected but because of its design, the detection range can be all over the place. This is because it does not care about repeating the smoothing a similar amount of times in each test case, but as long as there is no large jumps between each Y difference.