

Sophia Milask:

PSS Functions

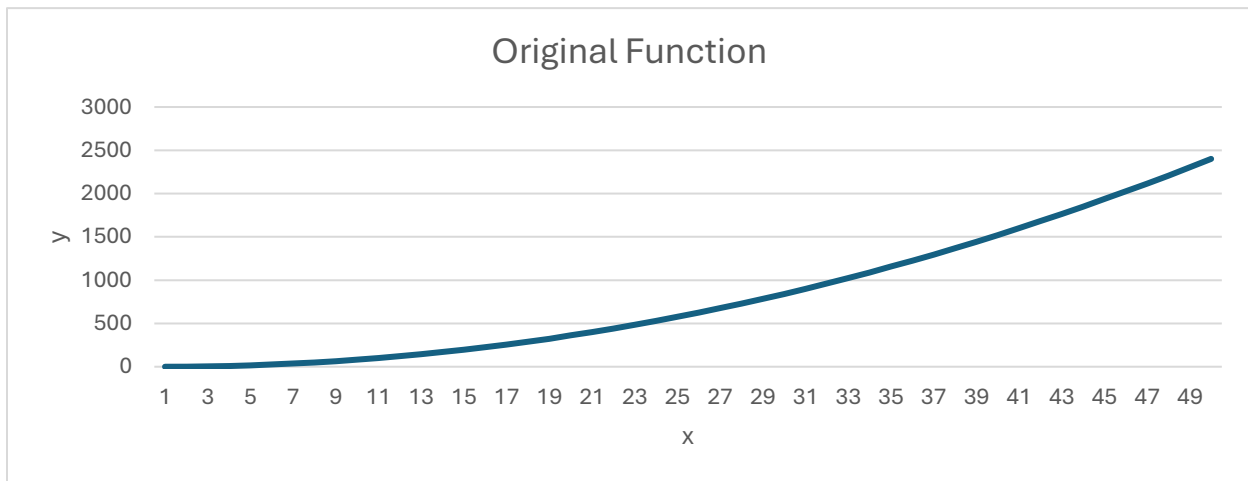
Purpose: To implement plotting, salting, and smoothing functions using Java, Octave, and external JARs. The goal is to compare the efficiency of each method to help determine the assets and drawbacks of each platform. This will aid in deciding the appropriate methods for executing certain tasks.

PSS 1 – Java:

Function: $y = x^2$

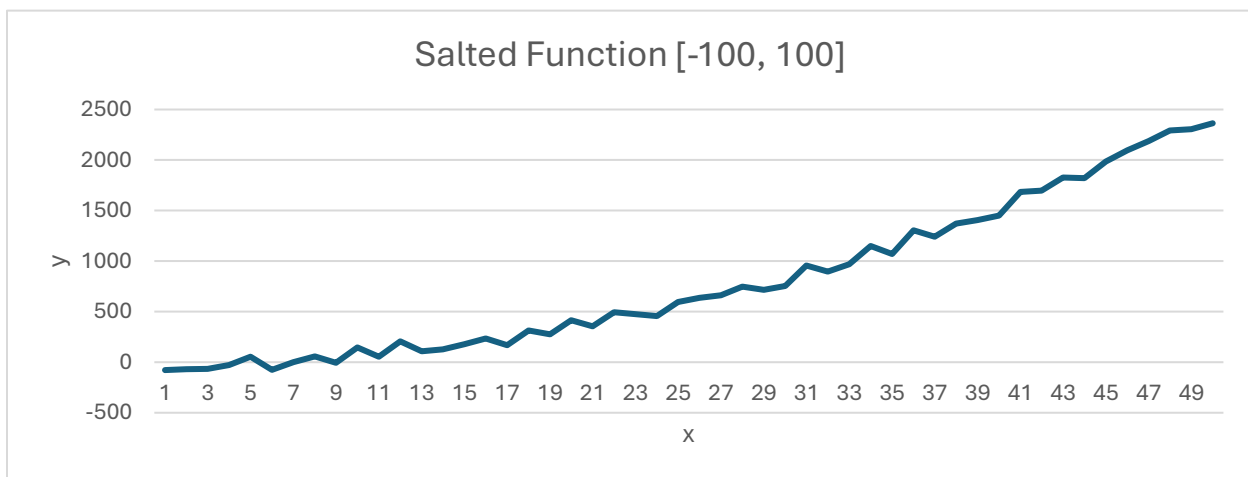
Figure 1: Changing the number of inputs; keeping salting range and smoothing window constant.

Test 1: 50 inputs



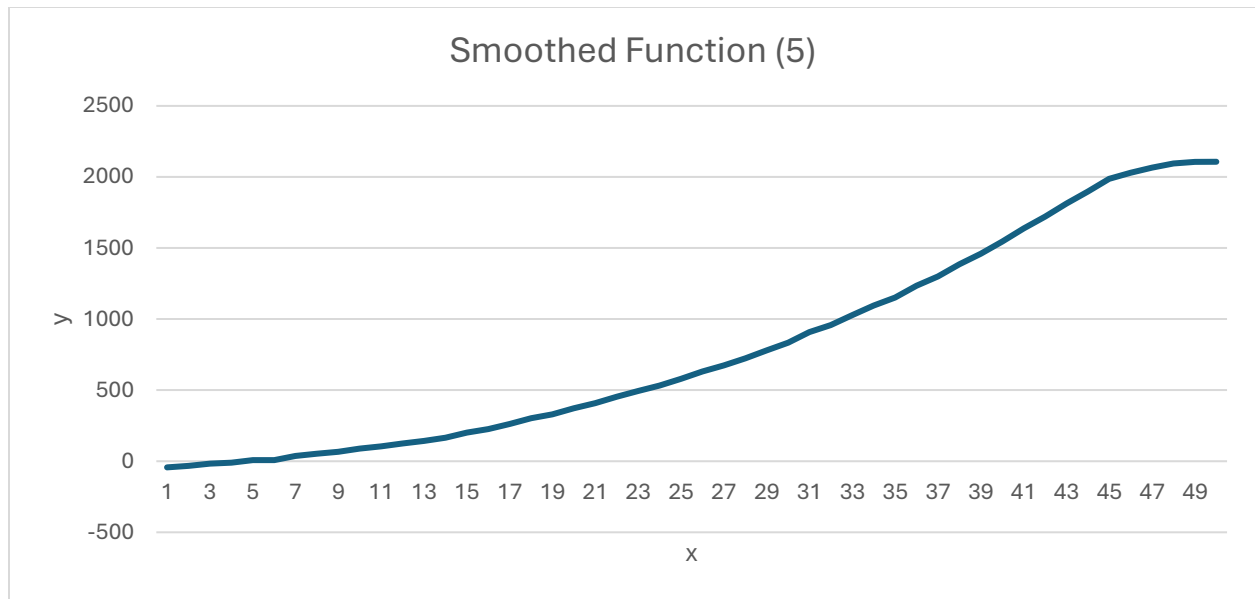
Original Function

The original function with x-values [0, 49].



Salted Function [-100, 100]

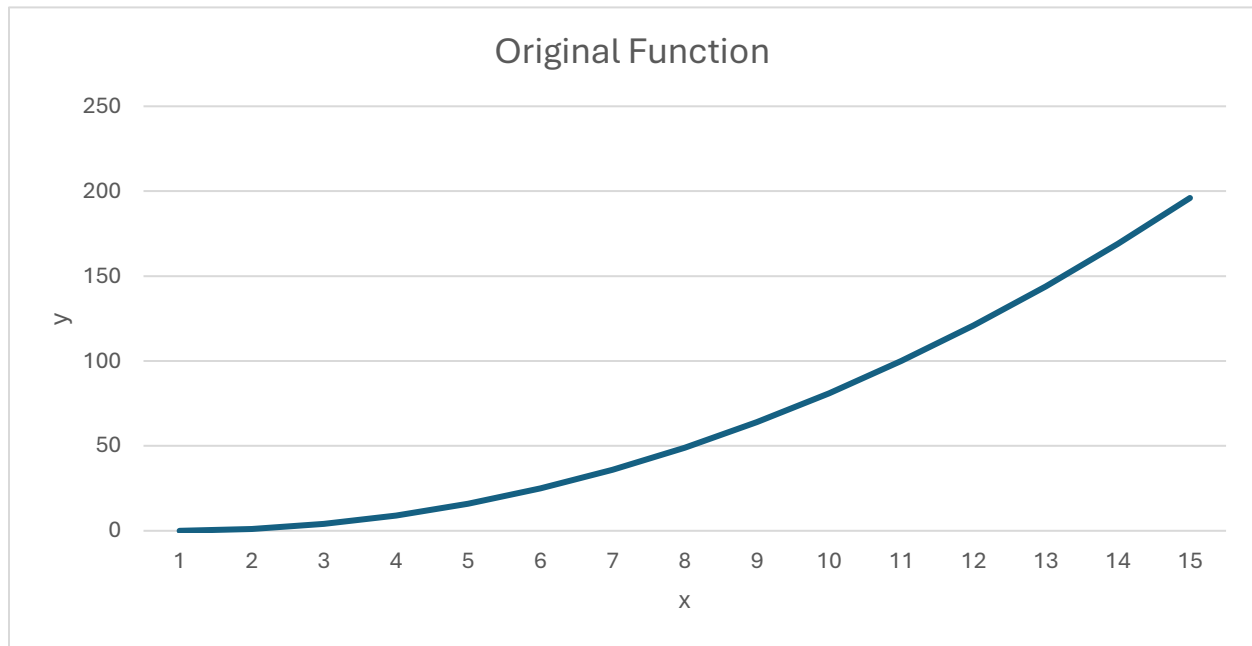
The graph of the original function after being salted with salting range [-100, 100].



Smoothed Function (5)

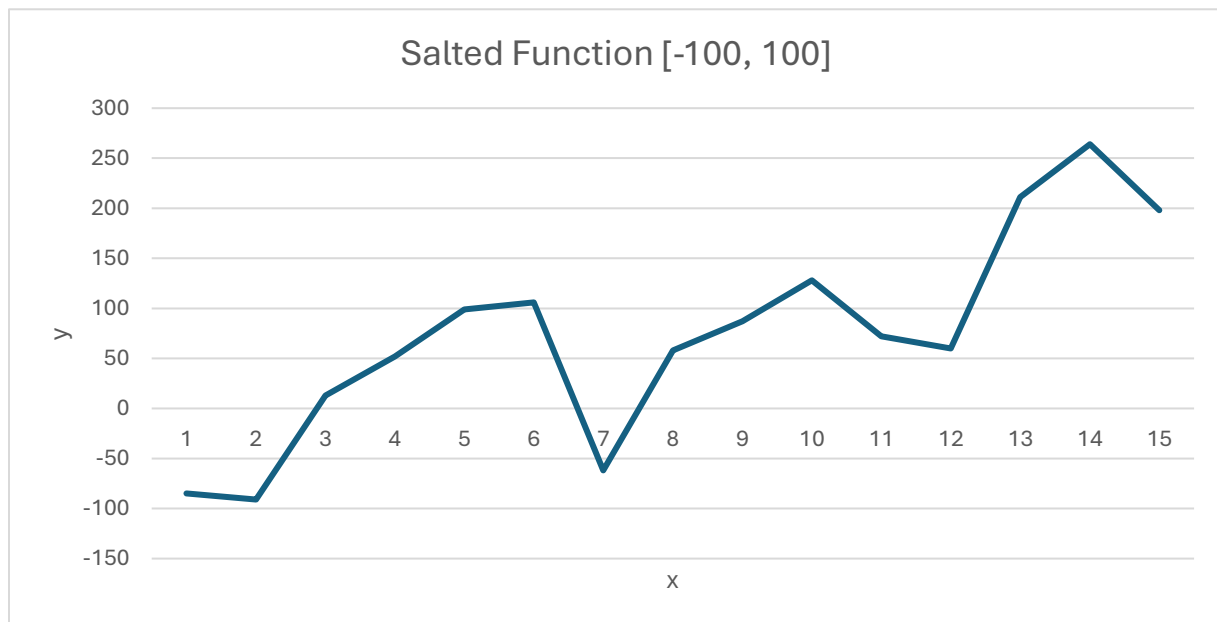
The graph of the function after being smoothed with a window size of 5.

Test 2: 15 inputs



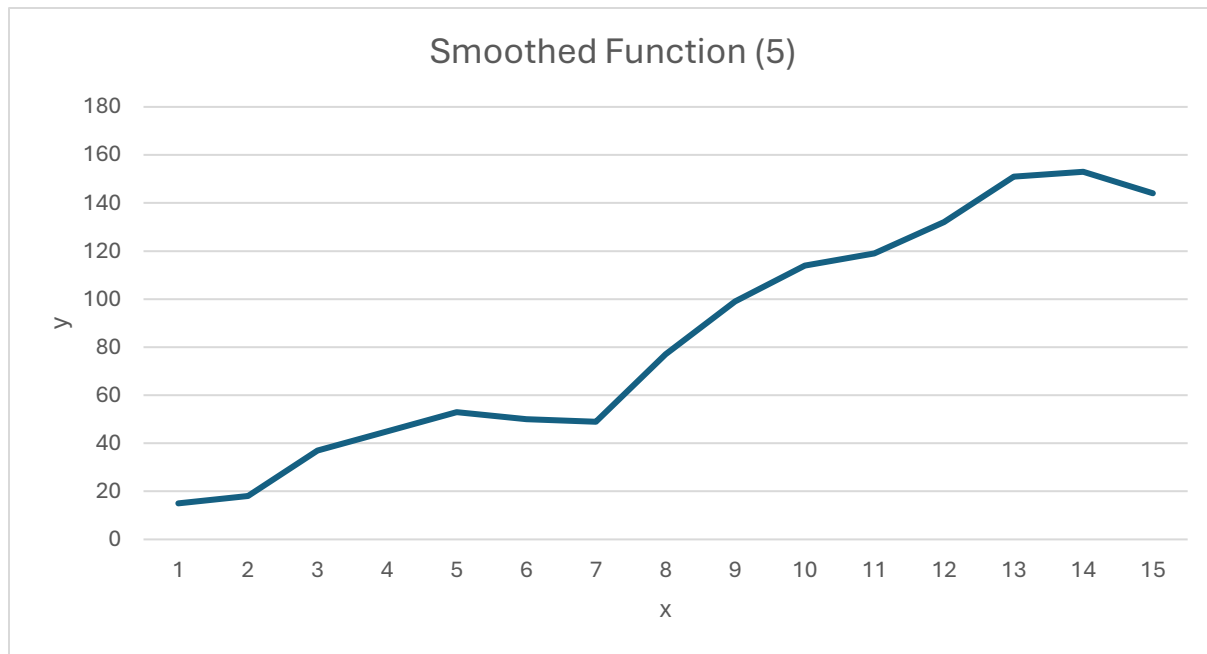
Original Function

The graph with input x-values [0, 14].



Salted Function [-100, 100]

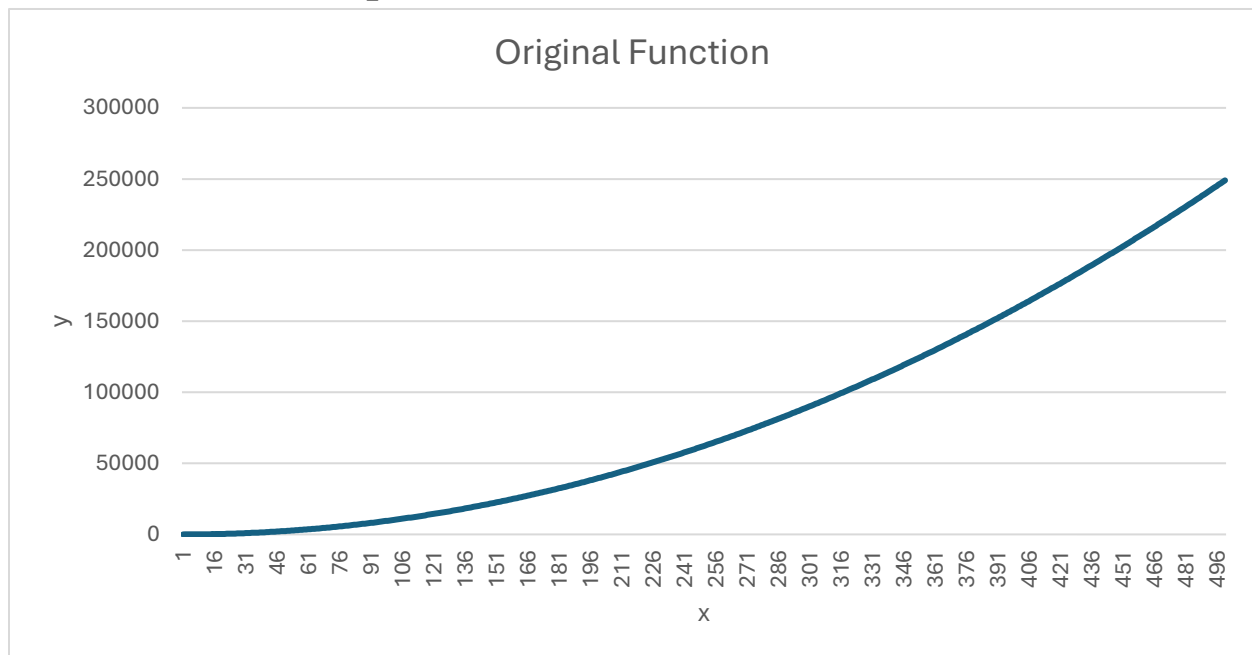
The graph of the original function after being salted with a salting range [-100, 100].



Smoothed Function (5)

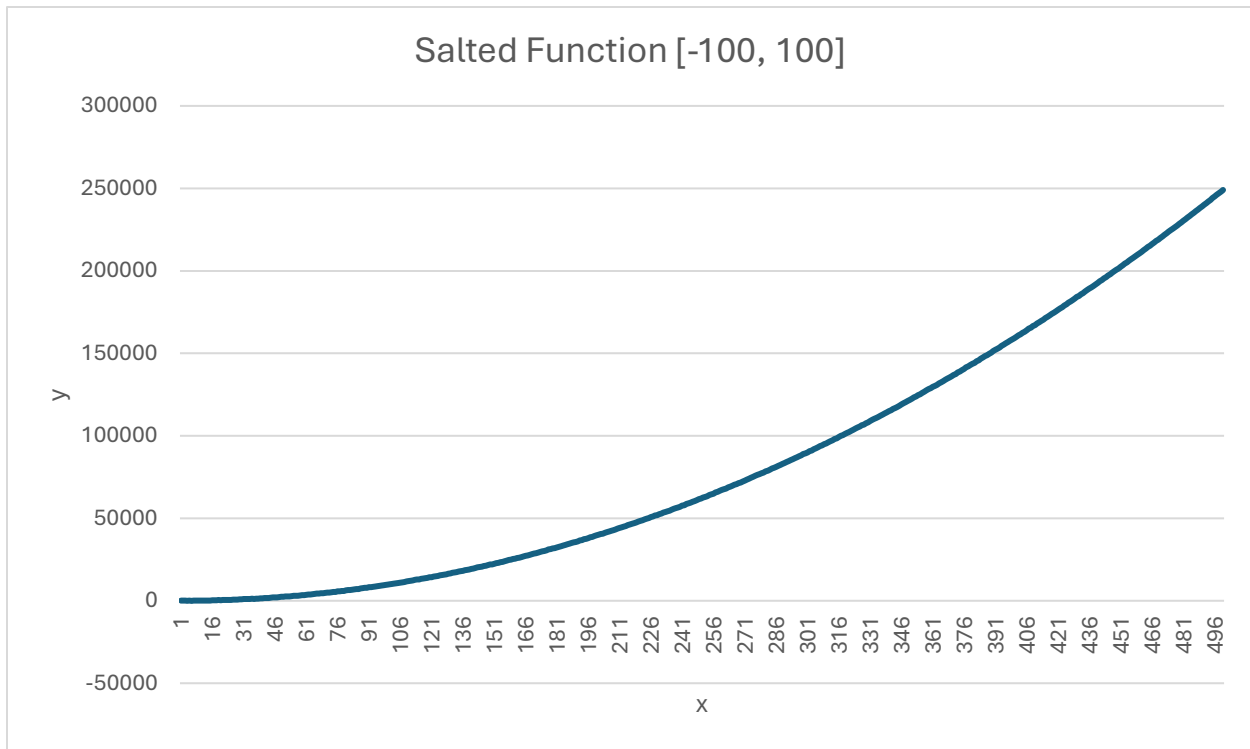
The graph of the function after being smoothed with a window size of 5.

Test 3: 500 inputs



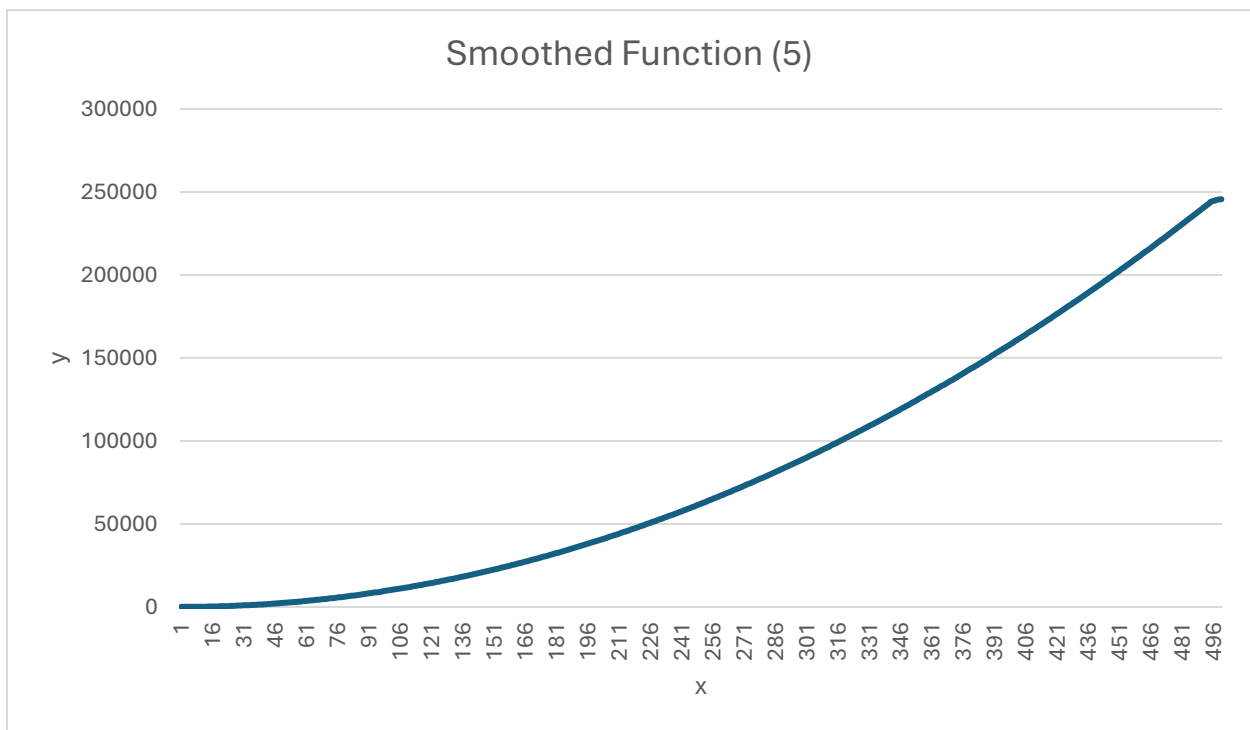
Original Function

The graph of the original function with x-values [0, 499].



Salted Function [-100, 100]

The graph after the original function was salted with salting range [-100, 100].



Smoothed Function (5)

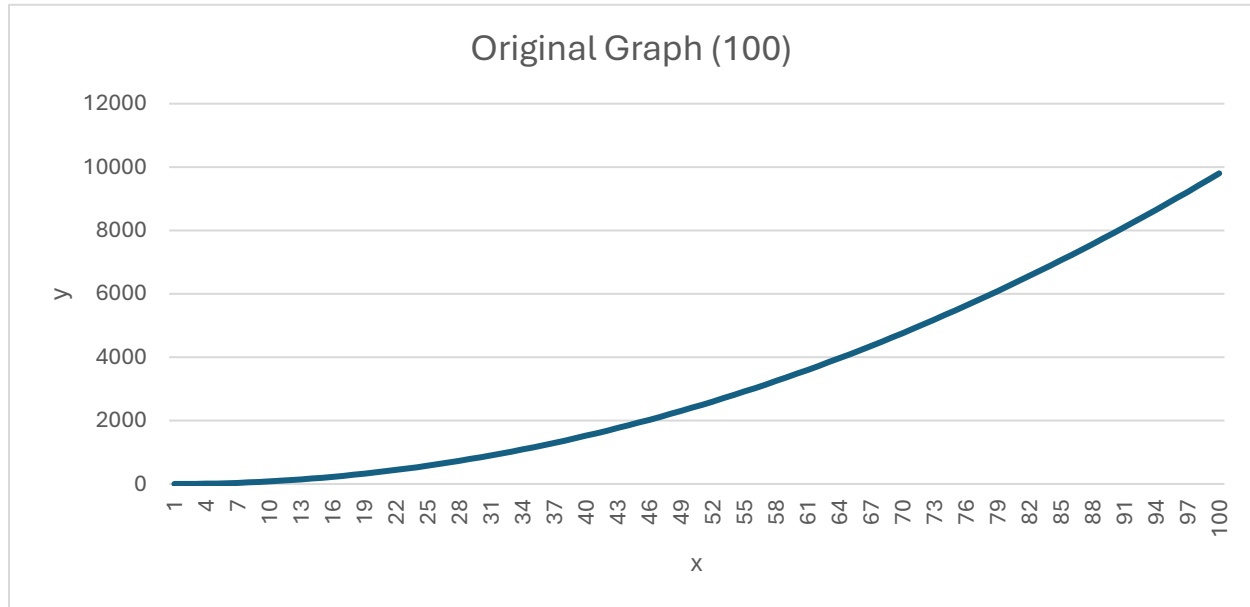
The graph of the salted function after being smoothed with a window value of 5.

Discussion:

1. Smoothing is more effective on larger data sets. This is because there are more “windows” to average. In a smaller data set, there are very few windows – each taking up a substantial amount of the data. In a larger set, there are more chances to average these windows, with each window approaching more accuracy and less overall effectiveness as the data size increases.
2. “Noise” has a larger effect on smaller data sets. If a certain y-value receives a new outlier value, this will greatly distort the graph, because the single point is a large percentage of the entire data. If there are more data points, the noise will be spread out and not as noticeable. So, salting is more effective on smaller data sets.
3. Edge data is greatly distorted the smaller the data set. In any data set, the edge values will be distorted because they don’t have full windows from which to average values. However, when there is a larger data set, these edge values are only a small piece of the data, limiting its impact and noticeability. In small data sets, these values take up a significant portion of the data and greatly affect the graph.
4. Larger data sets are especially helpful for the function $y = x^2$. In a smaller data set, assuming the x-values start from 0, the respective y-values are relative to the numerical value of their x-value. For example, the data set which included x-values [0, 14] had a max value of 196. If we were unlucky and randomly added 100 to that point, the percent change would be around 51%. This is loud noise added to the data and will be very noticeable and difficult to smooth. Now, the data set which included x values [0, 499] had a max value of 249,001. Again, if we get unlucky and randomly add 100 to this value, the percent change is only around .04%. So, large data sets for $y = x^2$ increasingly grow less susceptible to salting.

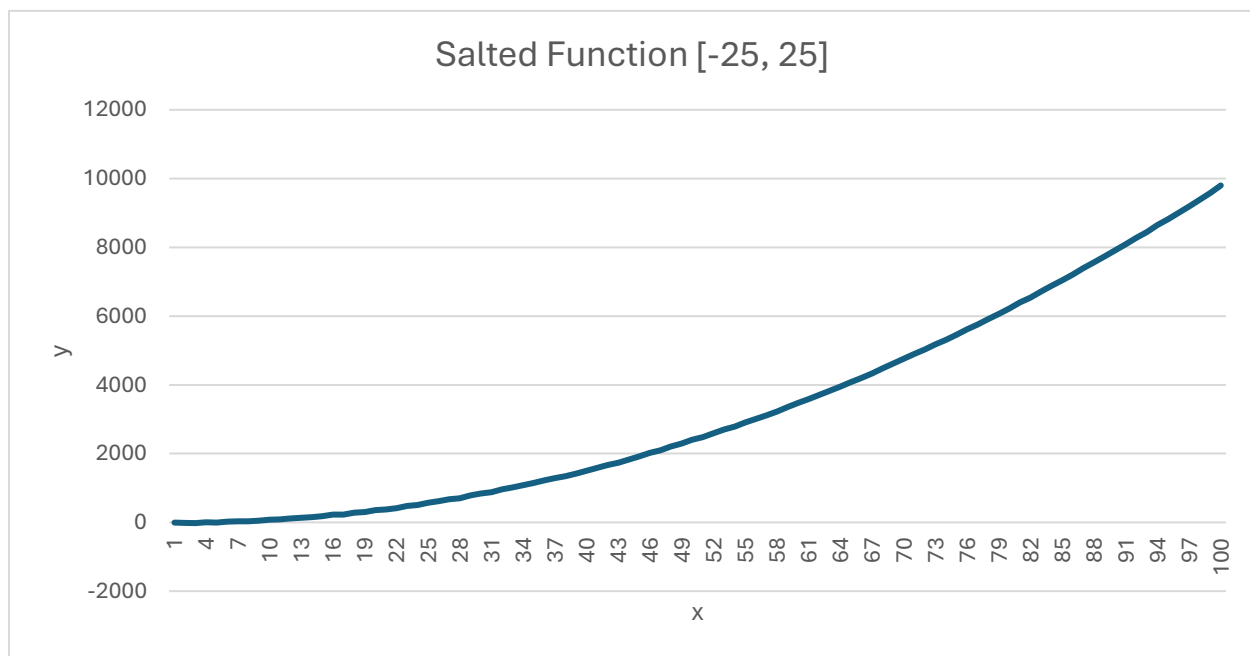
Figure 2: Changing the salting range, keeping the input size and smoothing window value constant.

Test 1: [-25, 25]



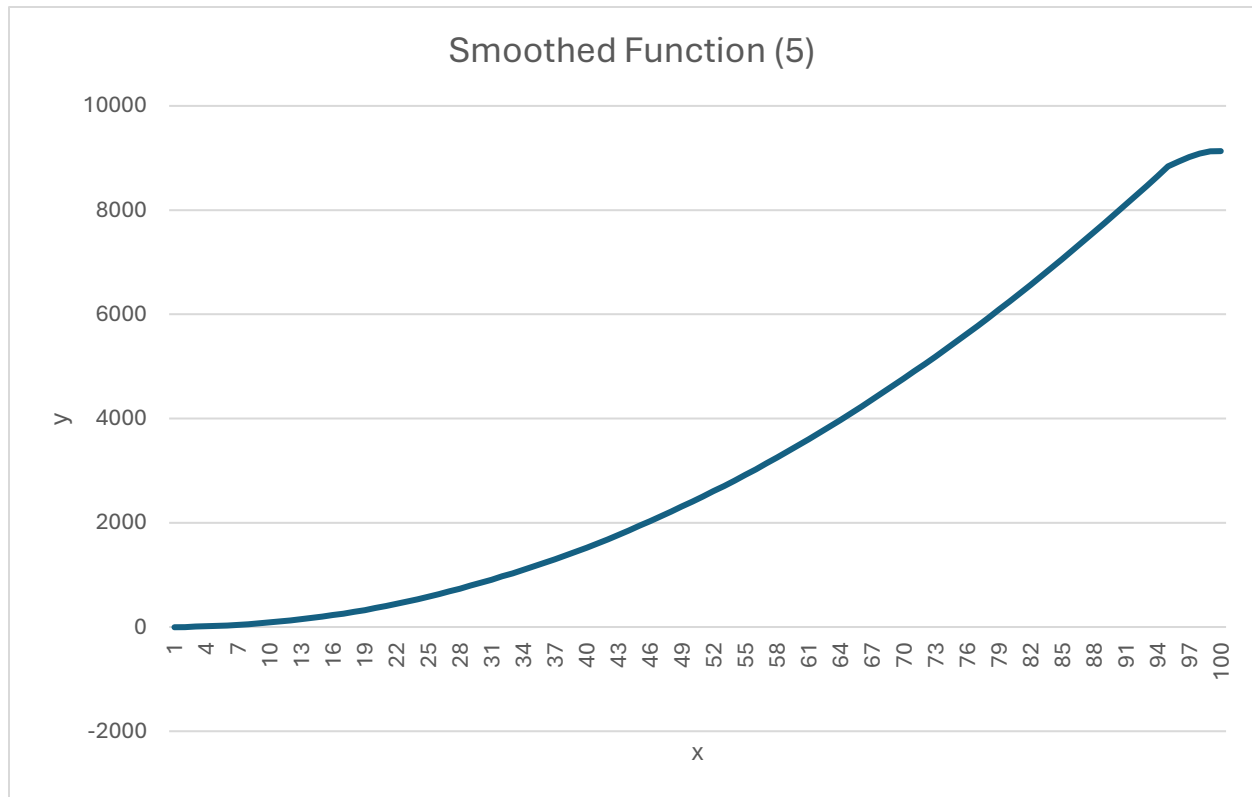
Original Graph (100)

The original function with x-values [0, 99].



Salted Function [-25, 25]

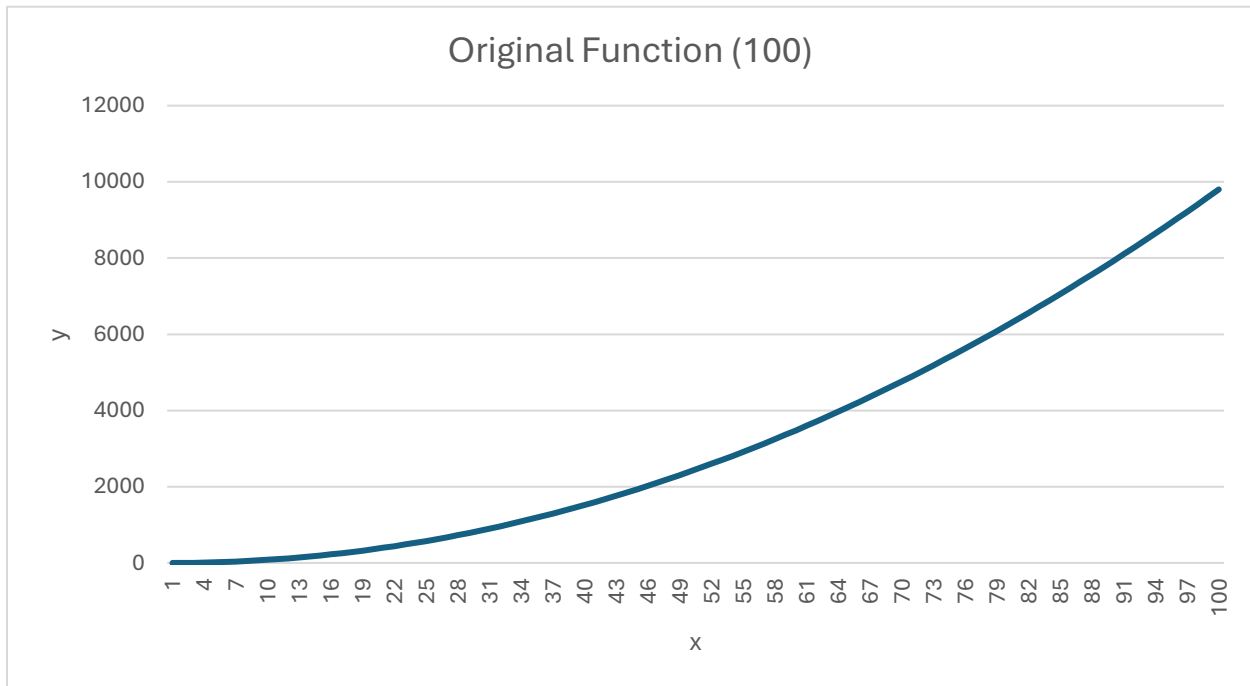
The graph of the original function after being salted with salting range [-25, 25].



Smoothed Function (5)

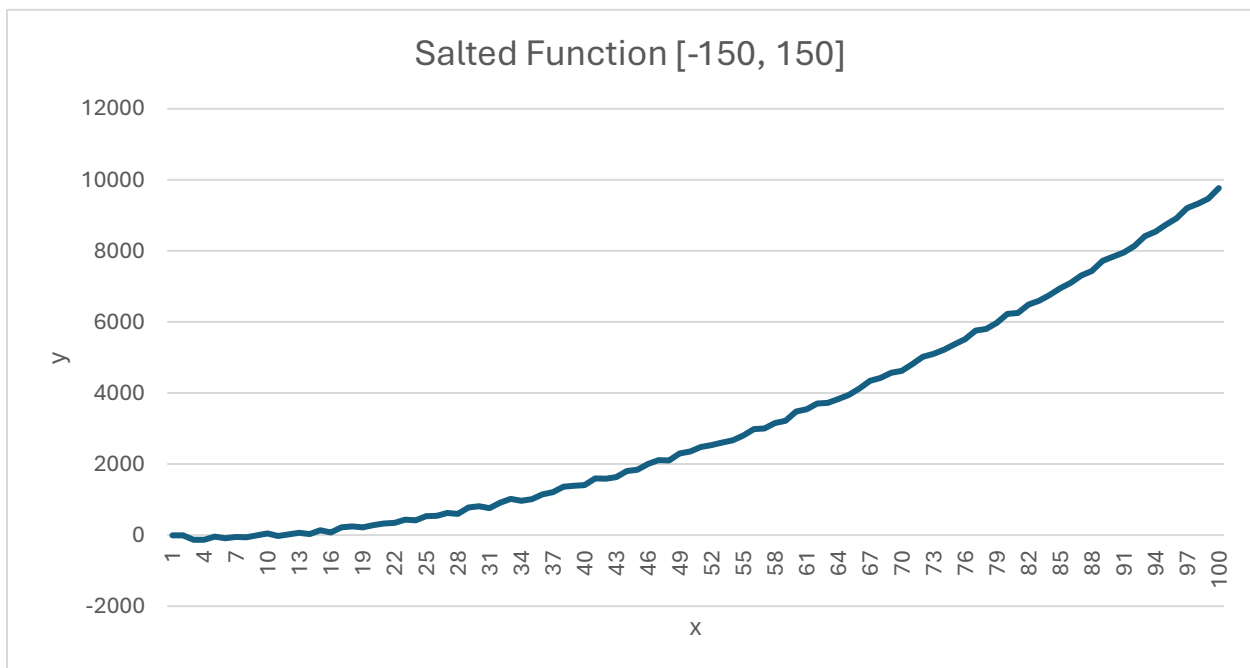
The graph of the salted function after it was smoothed with a window range of 5.

Test 2: [-150, 150]



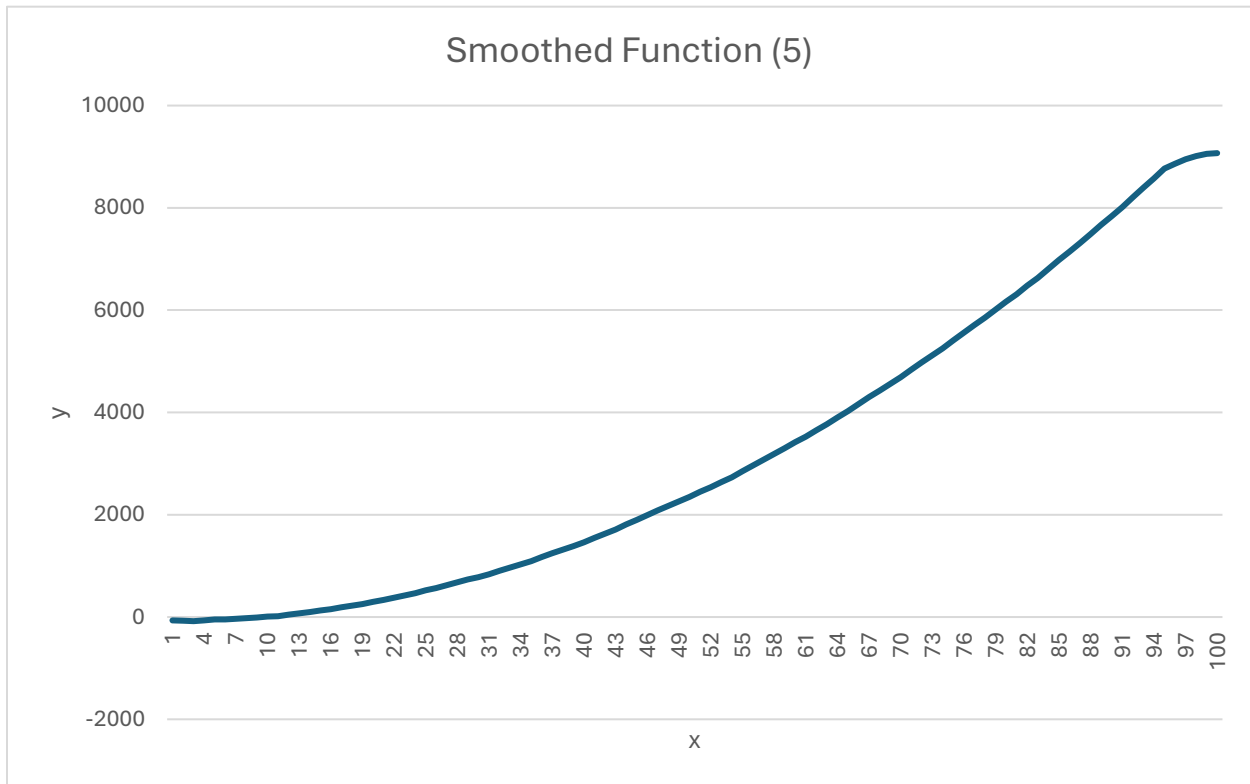
Original Function (100)

The graph of the original function with x-values [0, 99].



Salted Function [-150, 150]

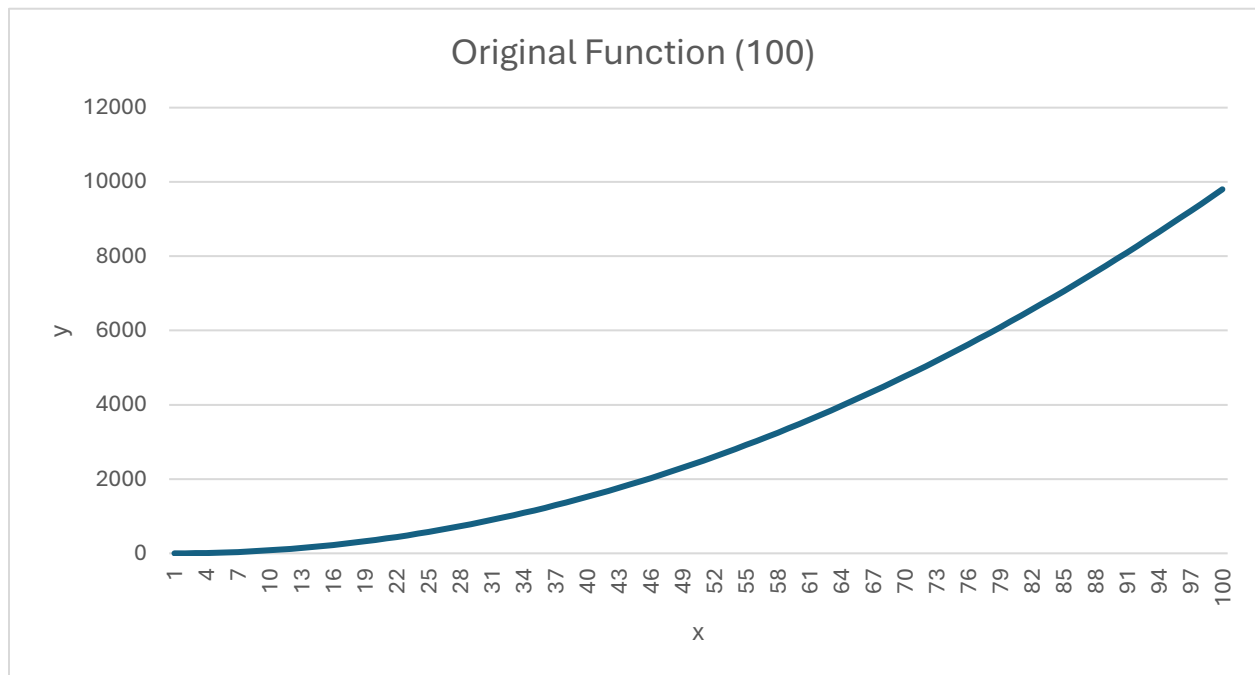
The graph of the original function after being salted with salting range [-150, 150].



Smoothed Function (5)

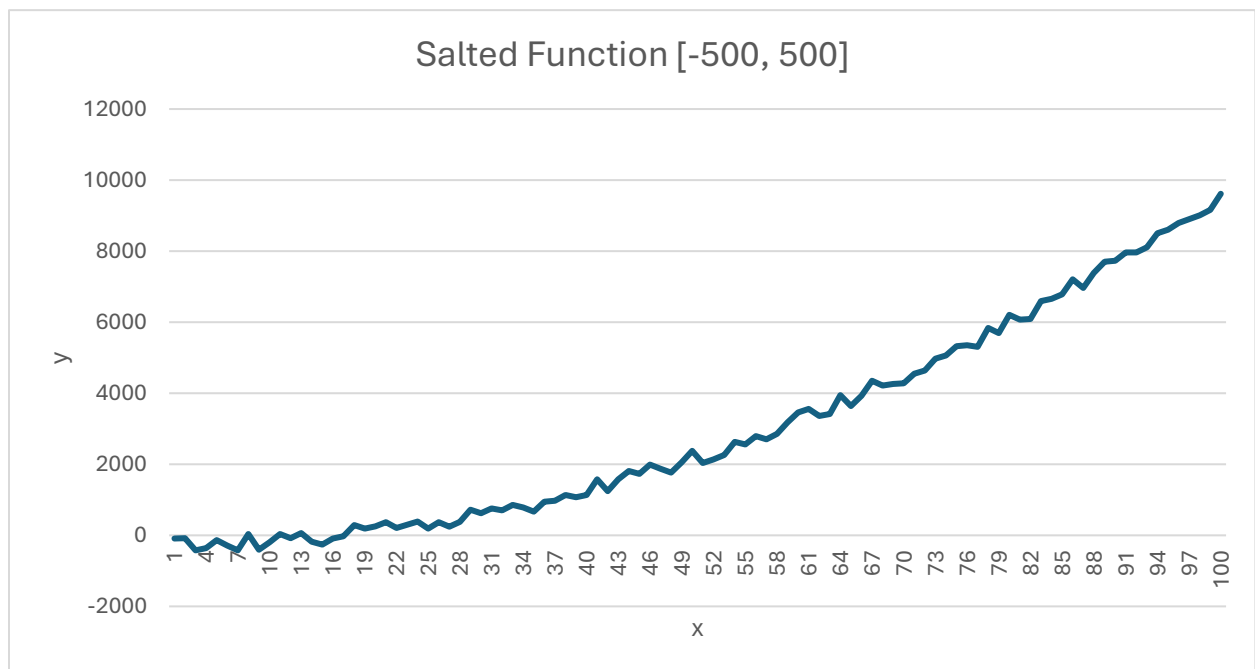
The graph of the salted function after being smoothed with a window of 5.

Test 3: [-500, 500]



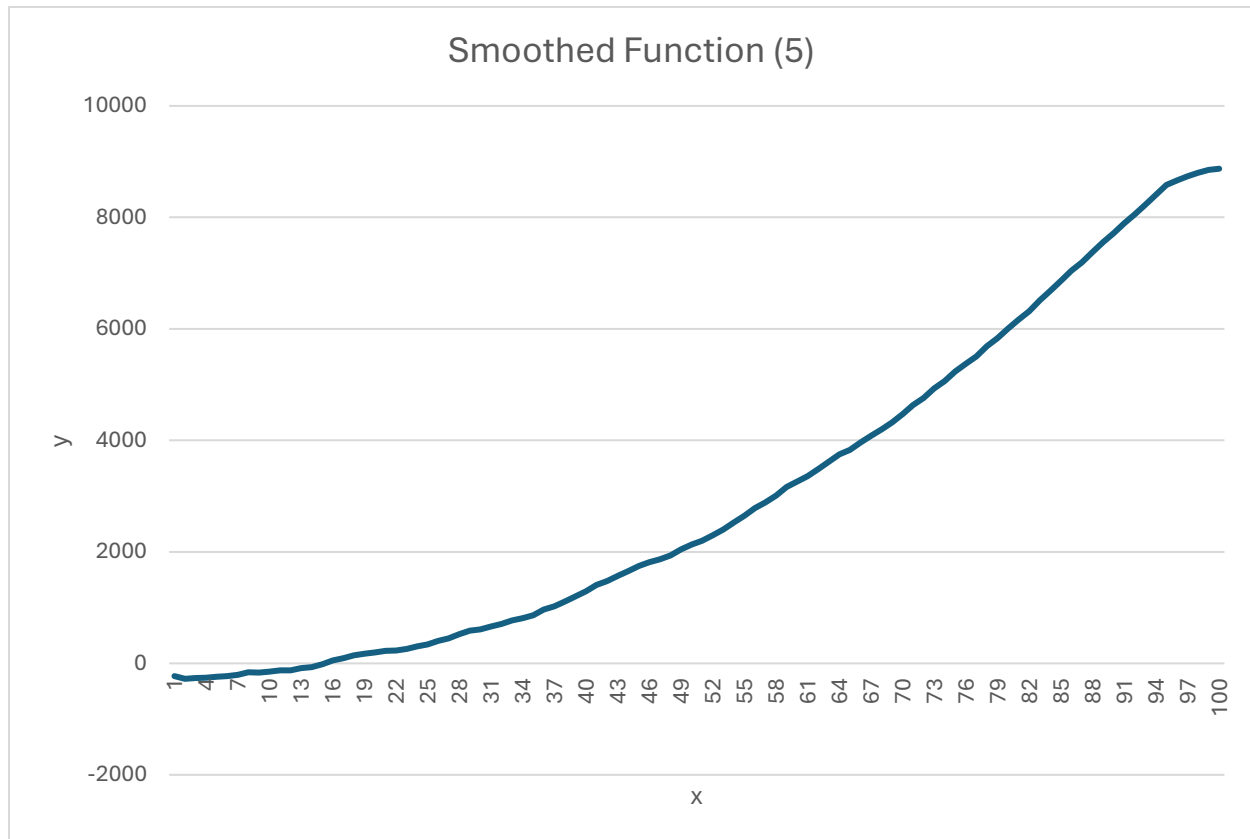
Original Function (100)

The graph of the original function with x-values.



Salted Function [-500, 500]

The graph of the original function after being salted with salting range [-500, 500].



Smoothed Function (5)

The graph of the salted function after being smoothed with a window of 5.

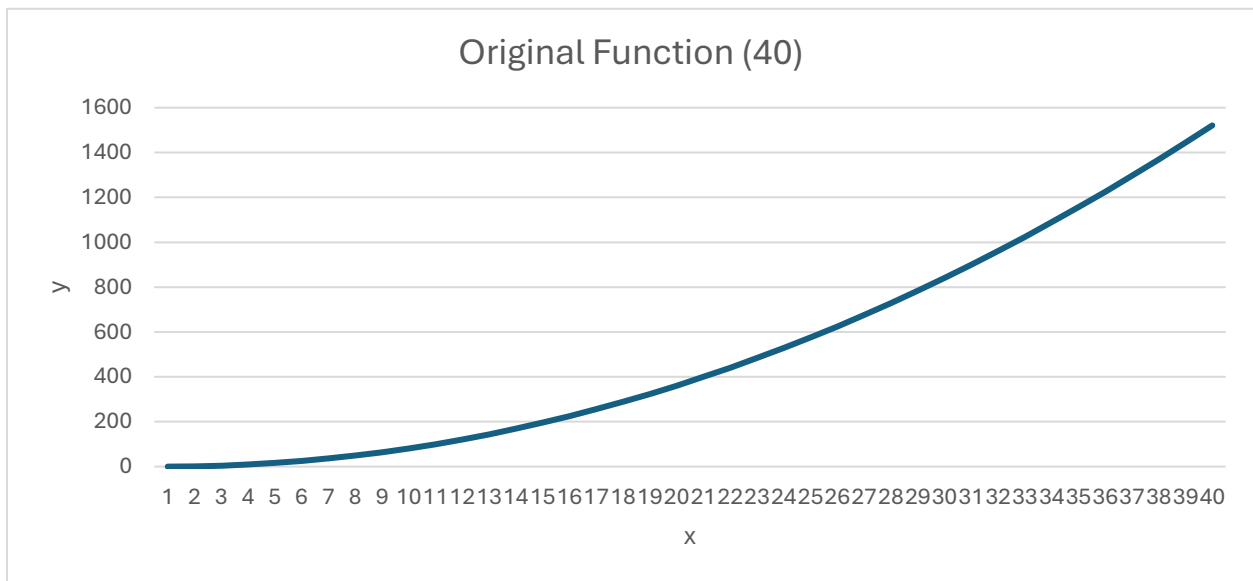
Discussions:

1. Higher salting ranges cause more distortion in the graph. When the possible salting values increase, there is more variability in the range of the new, salted values. This means that the new y-values can, and will, deviate more from their original values than with a smaller salting range.
2. As salting ranges increase, graphs become more unrecognizable from the original. Again, when there is more deviation, the graph begins to lose its original curve.
3. The smaller y-values are more affected by higher salting. We can see from the graphs that smoothing is less effective in the beginning of the graph. This is because these high salting ranges can cause the new y-values to deviate

much farther than the original values. As the y-values increase and the salting range remains the same, the percent change of the salter decreases.

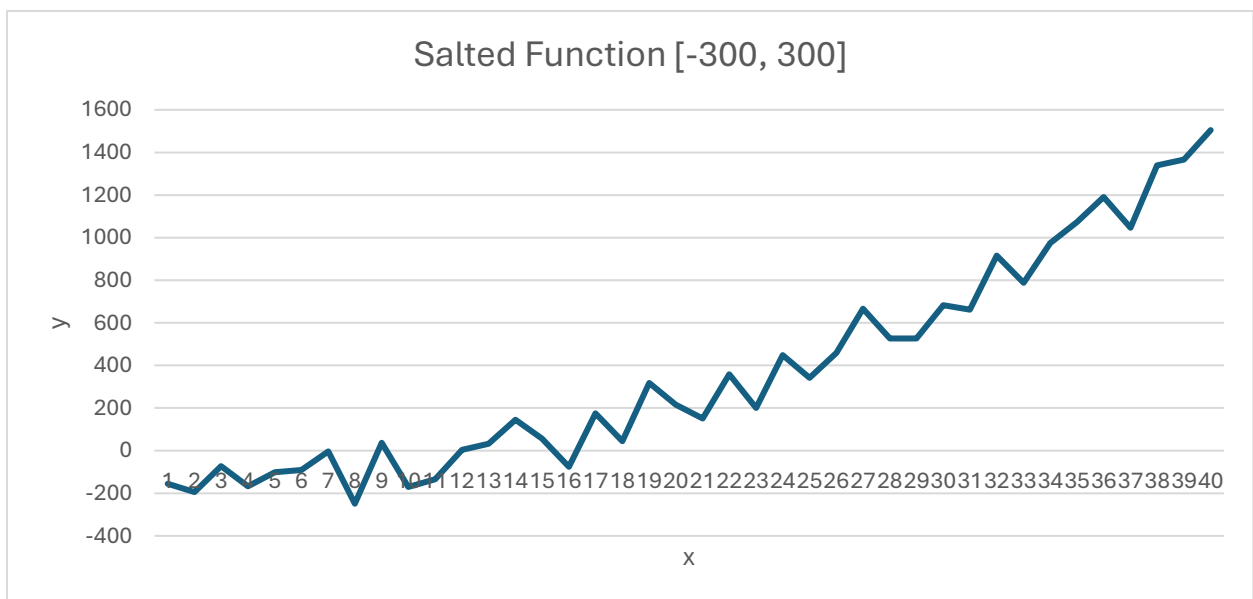
4. Smoothing becomes less effective as salting range increases. When ranges increase, the average of the surrounding y-values will be far less accurate than those from a smaller salting range. Therefore, the average value becomes increasingly less accurate and less effective at smoothing and restoring the data.

Figure 3: Changing the smoothing window; keeping the salting range and data size constant.



Original Function (40)

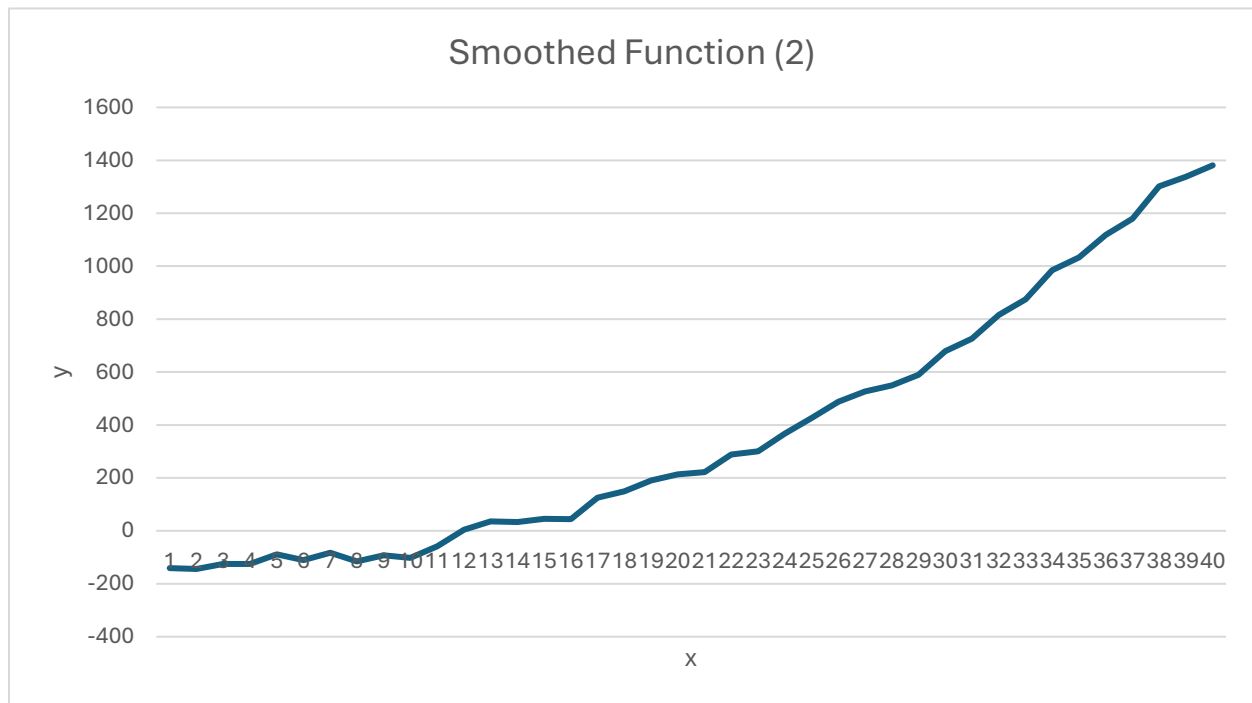
The graph of the original function with x-values $[0, 39]$. This will be the reference for the original graph for all 3 proceeding tests.



Salted Function $[-300, 300]$

The graph of the original function after being salted with salting range $[-300, 300]$. This salted graph will be used as a reference to the 3 proceeding tests.

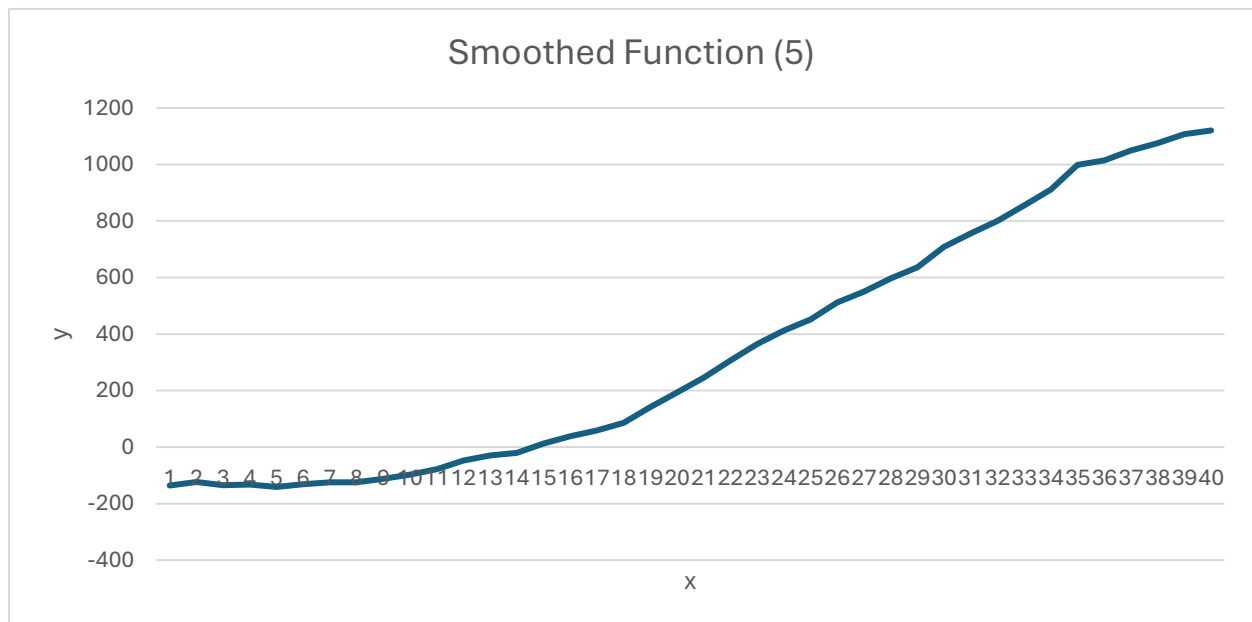
Test 1: 2



Smoothed Function (2)

The graph of the salted function after being smoothed with a window value of 2.

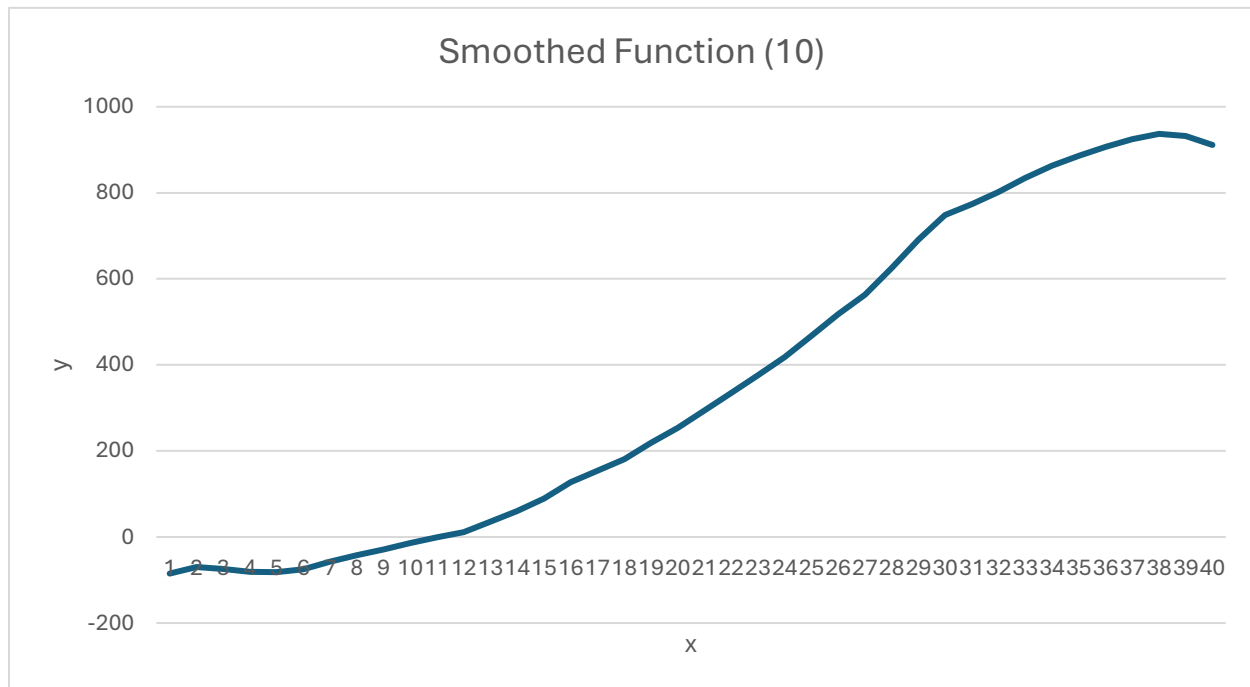
Test 2: 5



Smoothed Function (5)

The graph of the salted function after being smoothed with a window value of 5.

Test 3: 10



Smoothed Function (10)

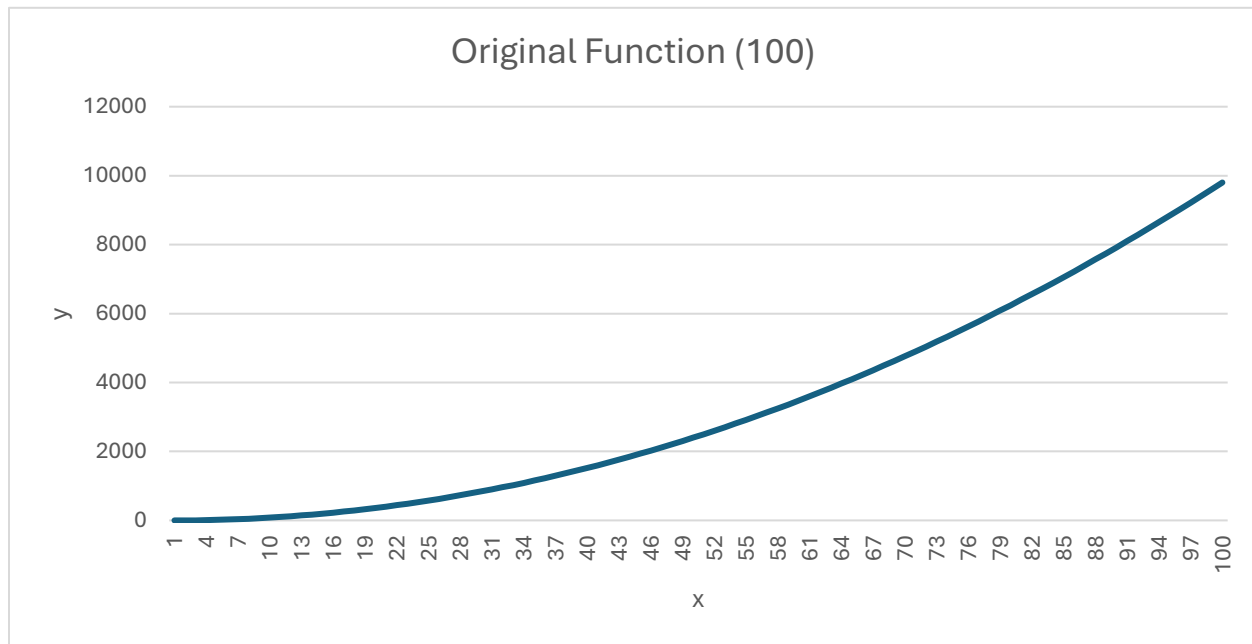
The graph of the salted function after being smoothed with a window value of 10.

Discussion:

1. As the smoothing window increases, the more the graph is restored. This is because more data around the current salted point is able to be collected and averaged. The noise added from the salter is equally likely to add or subtract values from the original function. This means as you take the average of more points, the adding and subtracting from the salter will start to even out and undo each other.
2. Very large windows can over-smooth. This causes the graph to flatten out in places where it should not. This happens because the function increases very quickly. So, when you take points very far away to include in the average, they are not points you want because they are from a very different part of the original curve.
3. There has to be a balance between keeping accuracy and eliminating the noise. When window size increases, the data becomes smoother and noise

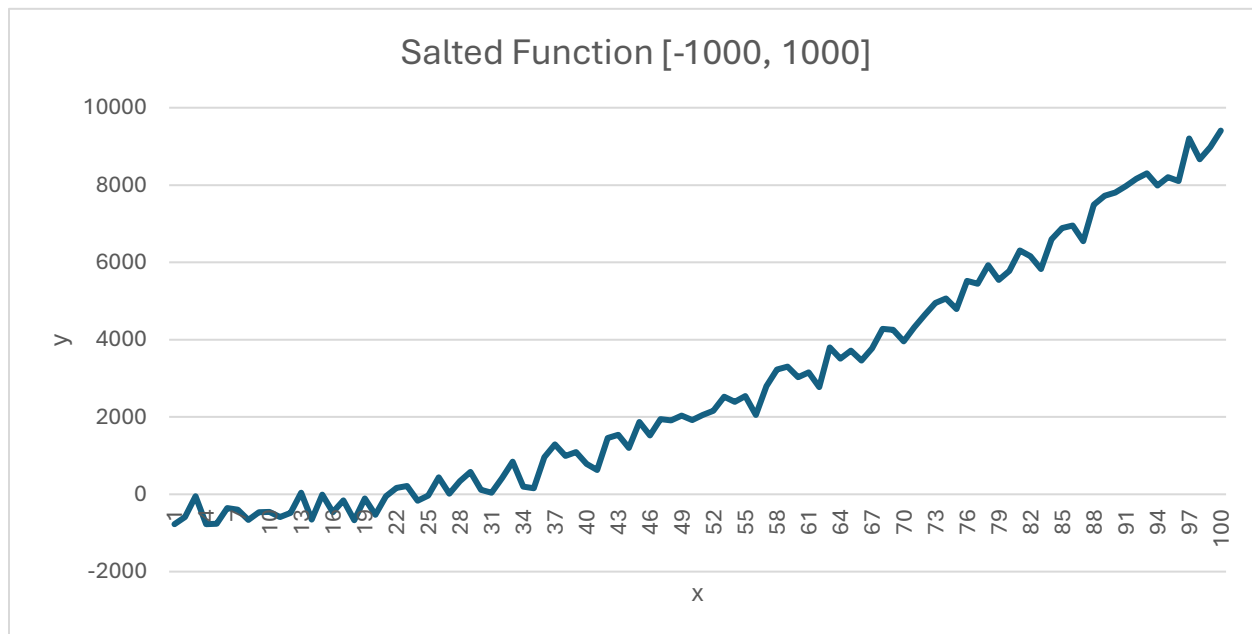
disappears, but you are susceptible to losing correct data. Although smaller windows do not smooth the curve very well, they still retain a lot of accuracy.

Figure 4: Smoothing multiple times.



Original Function (100)

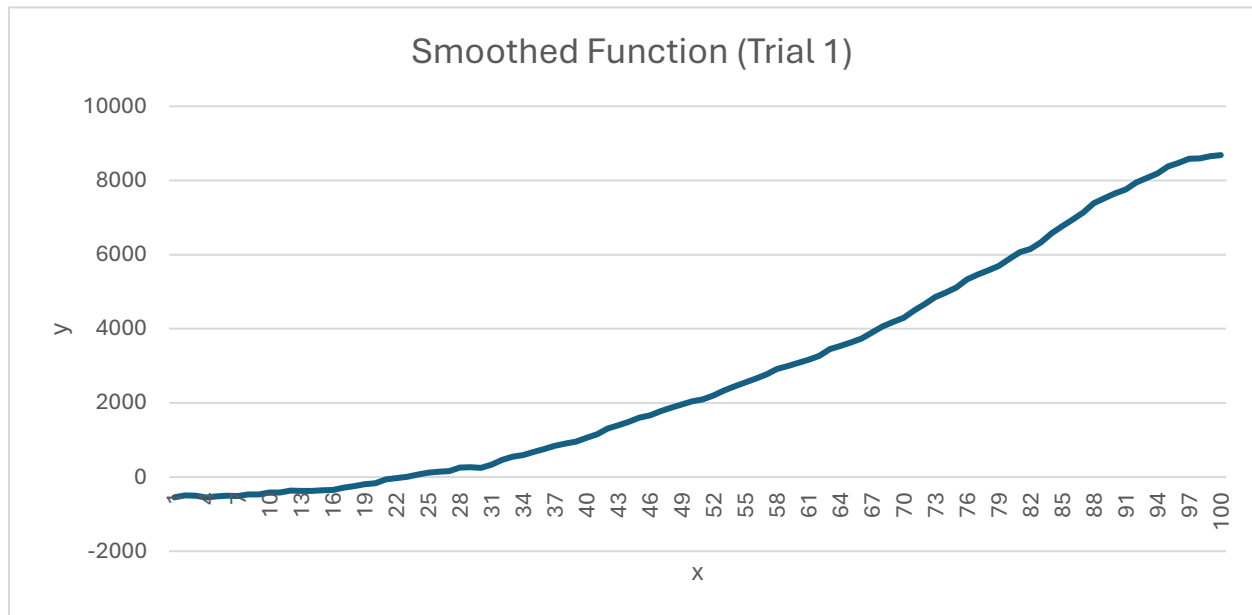
The graph of the original function with x-values [0, 99]. This will be the reference for the proceeding trials.



Salted Function [-1000, 1000]

The graph of the original function after being salted with salting range [-1000, 1000]. This will be used as reference for the proceeding trials.

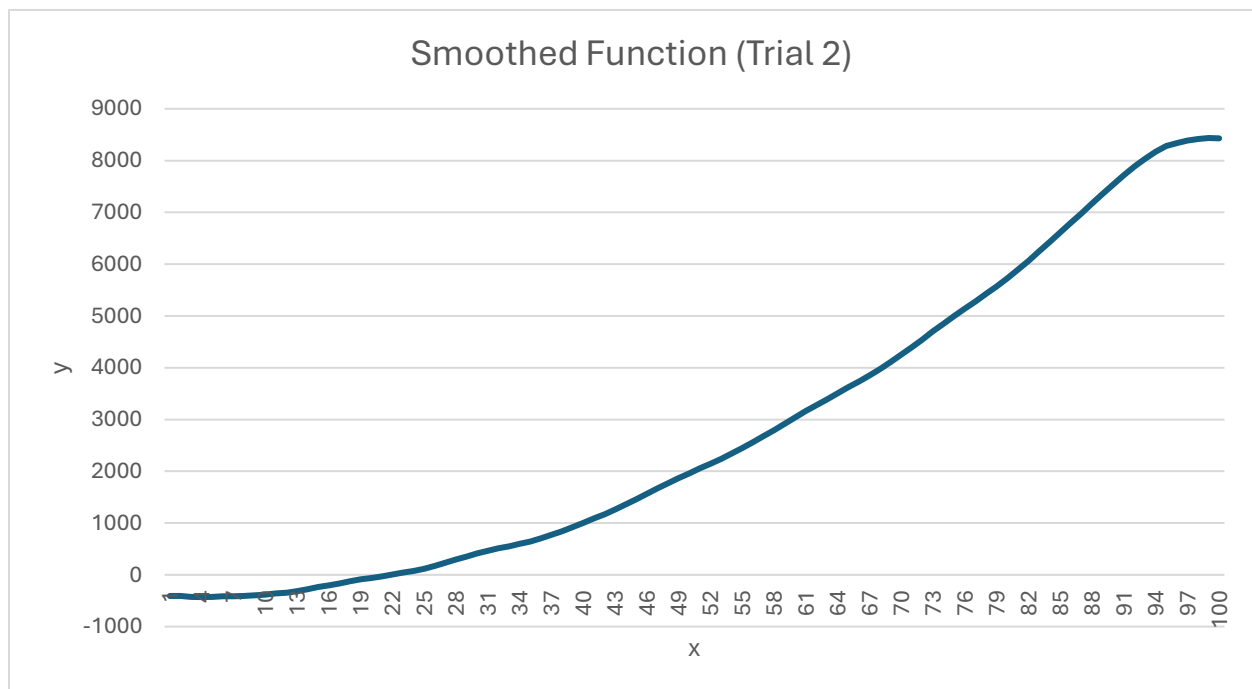
Trial 1:



Smoothed Function (Trial 1)

The graph of the first run of the smoother with a window value of 5.

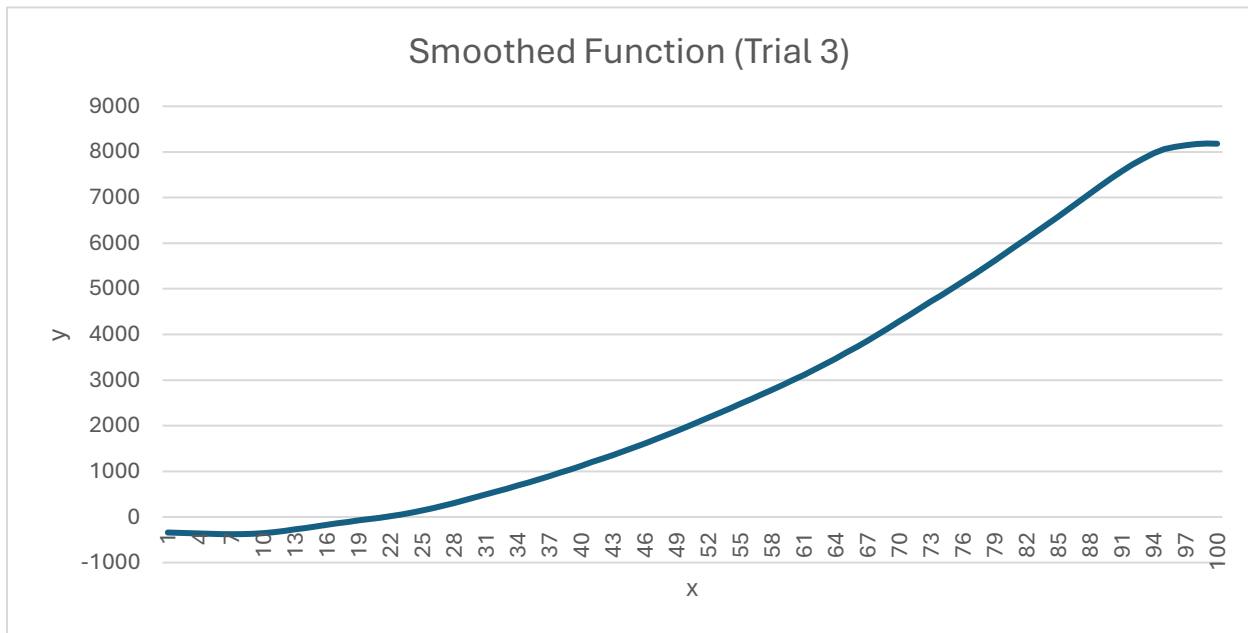
Trial 2:



Smoothed Function (Trial 2)

The graph of the second run of the smoother with a window value of 5.

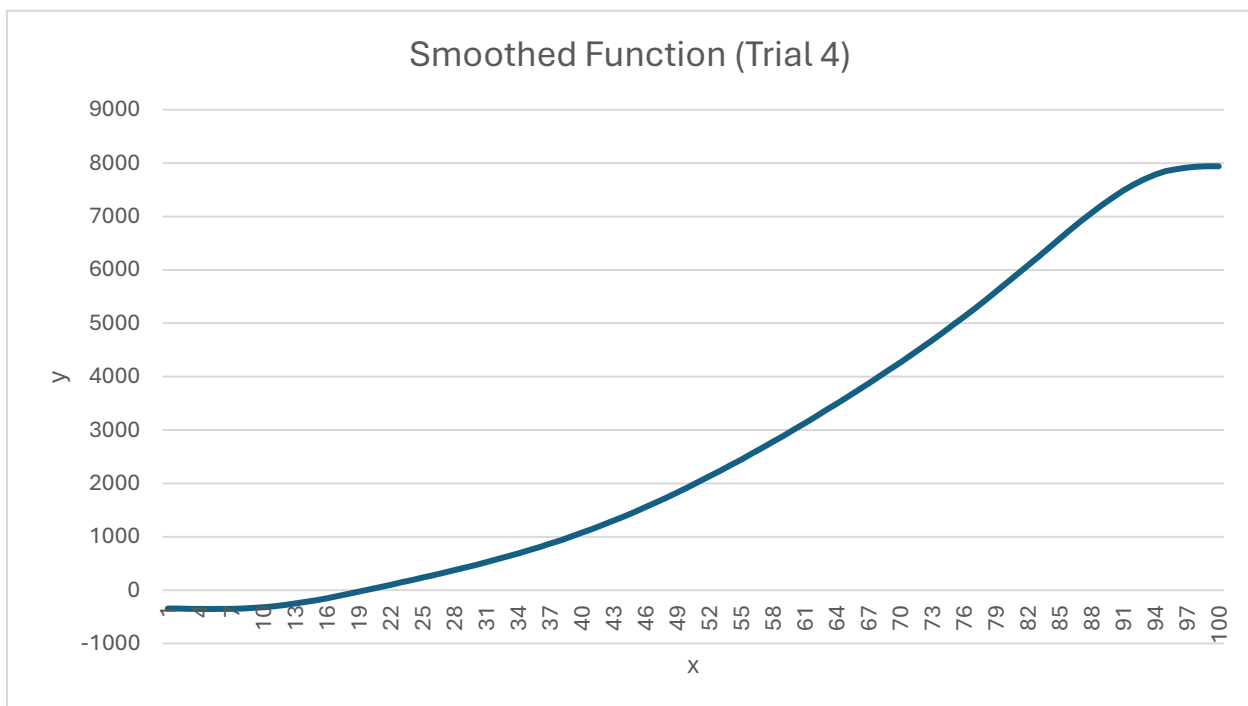
Trial 3:



Smoothed Function (Trial 3)

The graph of the third run of the smoother with a window value of 5.

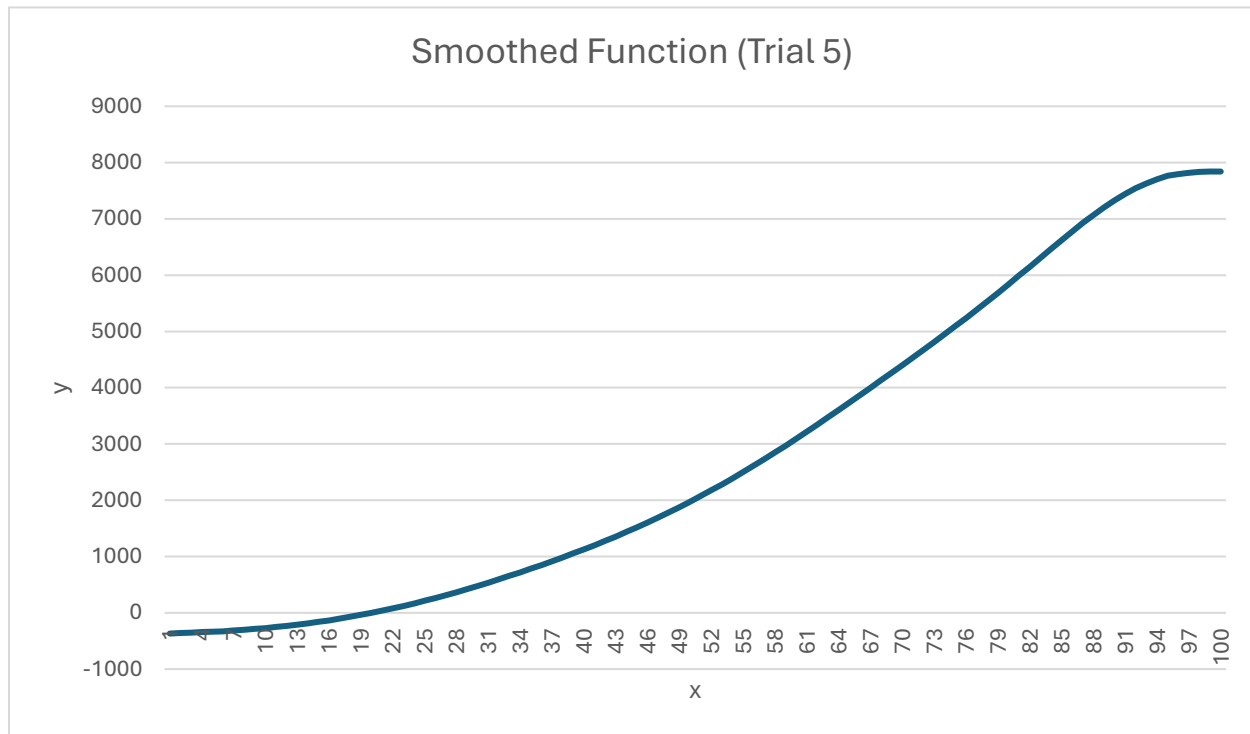
Trial 4:



Smoothed Function (Trial 4)

The graph of the fourth run of the smoother with a window value of 5.

Trial 5:



Smoothed Function (Trial 5)

The graph of the fifth run of the smoother with a window value of 5.

Discussion:

1. The first pass is the most effective. This is because the variance in the data is highest on the 0th pass. So, the first pass will quickly cancel out all of the addition and subtraction from the salter. Simply put, there is less work for the additional passes.
2. The progressive passes smooth the curve, but causes inaccurate flatness. This is because over time, neighboring peaks and valleys will start to pull each other together and create flatness on the curve which is not correct.
3. Higher passes are not necessary. As the number of passes increased, not only were the changes almost unnoticeable, but they were also doing damage to the accuracy. Therefore, smoothers should most likely not be run more than 3 times.

PSS 2 – Octave:

Part 1: Research

Research for Octave will be done using a 4-part video series on YouTube called “Octave Tutorials” uploaded by DrapsTV.

Lesson 1: Introduction and Setup

Step 1: Downloading Octave

I went to the download page of Octave and selected the Wiki link. I then selected Octave 3.2X for Windows MinGW32. Finally, I followed the instructions on the page and had Octave installed on my computer.

Step 2: Testing Octave

I set a custom prompt on the command line. I changed the prompt from two arrows to three arrows. Now I know Octave is up and working on my computer.

Pictures:

Code Segment (Step 2)

The code for changing the command prompt from “>>” to “>>>”.



```
GNU Octave, version 9.4.0
Copyright (C) 1993-2024 The Octave Project Developers.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.

Octave was configured for "x86_64-w64-mingw32".

Home page:      https://octave.org
Support resources: https://octave.org/support
Improve Octave:  https://octave.org/get-involved

For changes from previous versions, type 'news'.

>> ps1('>>> ')
>>> |
```

Discussion:

1. Octave is a high-level interpreted language designed for numerical computations which can visualize data. The primary use for Octave is to prototype solutions before implementing them into another programming language.
2. Octave is useful because it implements mathematical ideas and analysis very quickly. It is free. Octave does not require any external assistance to plot graphs from given data.

Lesson 2: Basic Operation

Step 1: Practicing the four basic operations

In the command window, I did four mathematical operations: addition, subtraction, multiplication, and division. I experimented with integer division in which the divisor does not divide the dividend, decimal multiplication and division, and division by 0.

Step 2: Testing other operations

I then tested another mathematical operation: raising both integers and non-integers to a power.

Step 3: Storing values in variables

I tested entering words, integers, and non-integers into variables and re-printing the variables with the values I gave them. I then assigned values to variables with a semi colon. Additionally, I executed mathematical operations with variables instead of with immediate values.

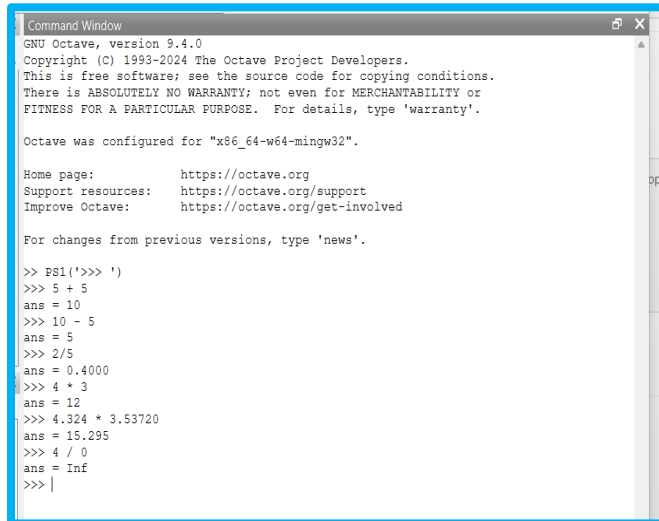
Step 4: Using vectors, matrices, and sets.

I made row and column-oriented vectors and matrices. I then did operations with my matrix and observed the differences between Octave matrices and matrices I used in Discrete Math. I used the colon operator to define a set of values, and then later used indexing to access those values.

Pictures:

Code Segment (Step 1)

The code for testing four basic operations.



```
Command Window
GNU Octave, version 9.4.0
Copyright (C) 1993-2024 The Octave Project Developers.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE. For details, type 'warranty'.

Octave was configured for "x86_64-w64-mingw32".

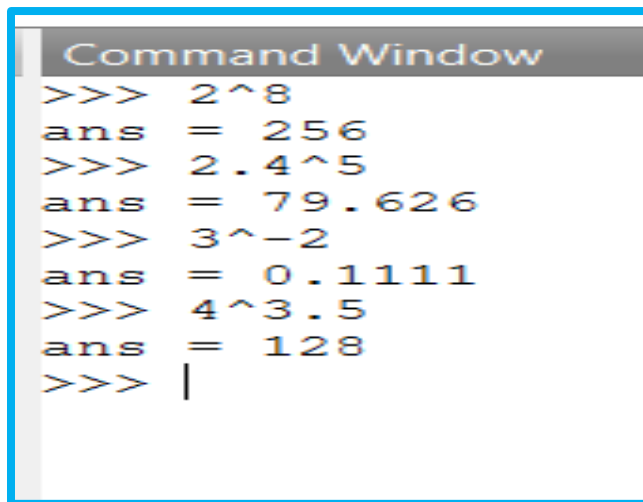
Home page:      https://octave.org
Support resources: https://octave.org/support
Improve Octave: https://octave.org/get-involved

For changes from previous versions, type 'news'.

>> BS1('>>> ')
>>> 5 + 5
ans = 10
>>> 10 - 5
ans = 5
>>> 2/5
ans = 0.4000
>>> 4 * 3
ans = 12
>>> 4.324 * 3.53720
ans = 15.295
>>> 4 / 0
ans = Inf
>>> |
```

Code Segment (Step 2)

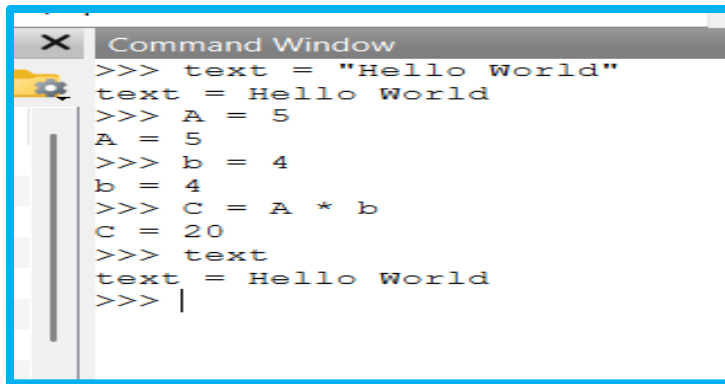
The code for testing raising values to powers.



```
Command Window
>>> 2^8
ans = 256
>>> 2.4^5
ans = 79.626
>>> 3^-2
ans = 0.1111
>>> 4^3.5
ans = 128
>>> |
```

Code Segment (Step 3)

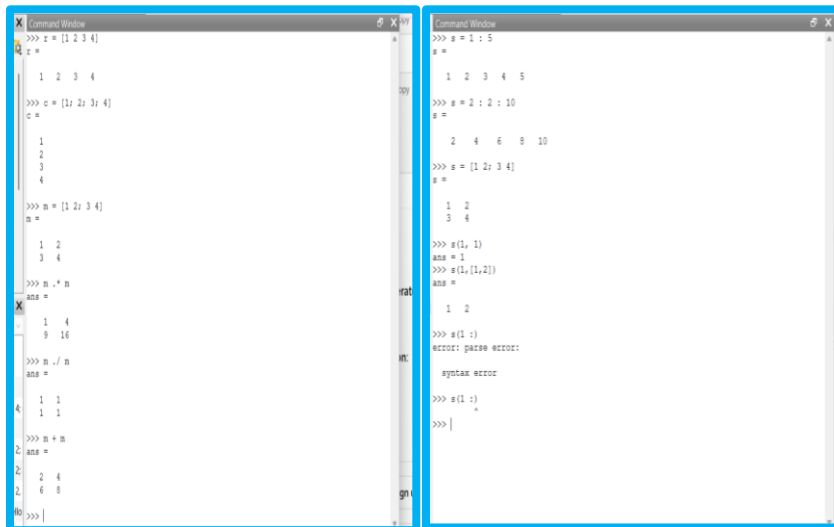
The code for experimenting with variables.



```
>>> text = "Hello World"
text = Hello World
>>> A = 5
A = 5
>>> b = 4
b = 4
>>> C = A * b
C = 20
>>> text
text = Hello World
>>> |
```

Code Segment (Step 4)

The code for experimenting with vectors, matrices, and sets.



```
>>> r = [1 2 3 4]
r =
  1 2 3 4
>>> c = [1; 2; 3; 4]
c =
  1
  2
  3
  4
>>> m = [1 2; 3 4]
m =
  1 2
  3 4
>>> m .* m
ans =
  1 4
  9 16
>>> m ./ m
ans =
  1 1
  1 1
>>> m + m
ans =
  2 4
  6 8
>>> |

>>> s = 1 : 5
s =
  1 2 3 4 5
>>> s = 2 : 2 : 10
s =
  2 4 6 8 10
>>> s = [1 2; 3 4]
s =
  1 2
  3 4
>>> s([1, 1])
ans = 1
>>> s([1, [1, 2]])
ans =
  1 2
  3 4
>>> s([1 :])
error: paren error:
syntax error
>>> s([1 :])
^
>>> |
```

Discussion:

1. Variables are dynamically typed. This means I don't have to label each type of variable I assign, Octave does it for me.
2. Matrices follow their own set of principles in Octave. For example, matrix multiplication is different in Octave than it is in Discrete Math. The matrices must have the same dimensions, and each value is matched with the respective value from another matrix.

3. The colon operator is very powerful and useful for quickly defining sets of values. Each value, row, column, or row/column segments can be accessed very quickly as well.

Lesson 3: Loading and Using Data

Step 1: Navigation, loading and saving, and temporary files.

I learned about changing directories, lists files, and the current path. I learned how to load and save files, and how to use files short-term so they don't permanently affect memory or storage

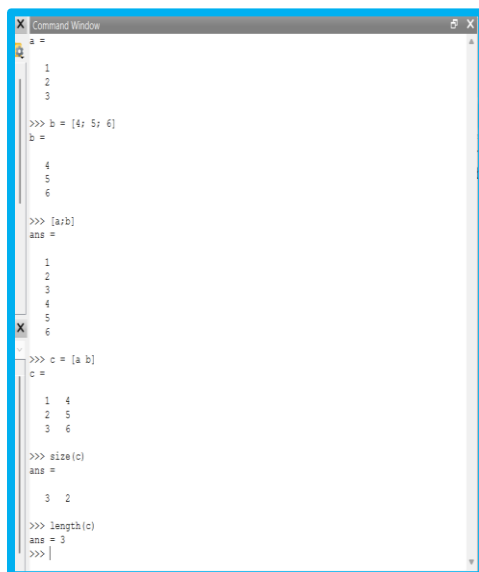
Step 2: More information about matrices

In the command window, I used concatenation for combining matrices, and then learned how to access the size and length of a matrix

Pictures:

Code Segment (Step 2)

The code for combining matrices and finding their size and length.



```
Command Window
a =
     1
     2
     3

>>> b = [4; 5; 6]
b =
     4
     5
     6

>>> [a;b]
ans =
     1
     2
     3
     4
     5
     6

>>> c = [a b]
c =
     1     4
     2     5
     3     6

>>> size(c)
ans =
     3     2

>>> length(c)
ans = 3
>>> |
```

Discussion:

1. Octave uses a working directory and supports most commands from windows. You can change the directory, list the current directory, and print the working directory.
2. Octave can load data from many different formats very easily. It is also very easy to save data.
3. Temporary files are good to use when there is a large amount of data or if another program uses that same data. They delete every time Octave closes so they won't affect memory.

Lesson 4: Plotting Data

Step 1: Scatter plots

I tested entering a function in the command line and graphing the data directly on Octave

Step 2: Histograms

I then tested entering data and then turning it into a histogram directly on Octave.

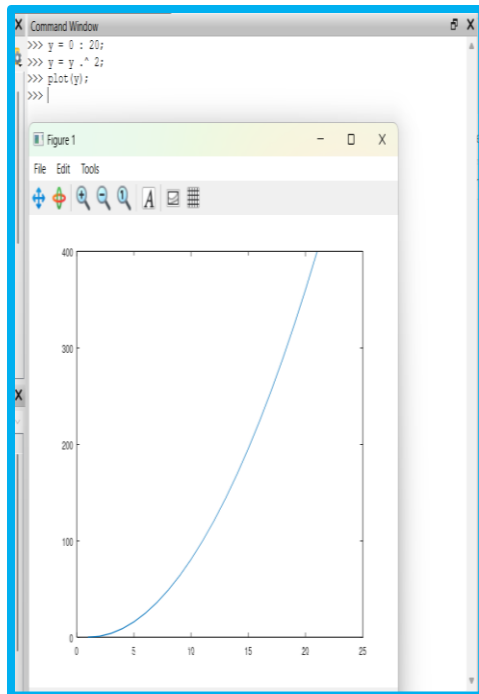
Step 3: Pie Charts

I tested entering data and then turning it into a pie chart directly on Octave.

Pictures:

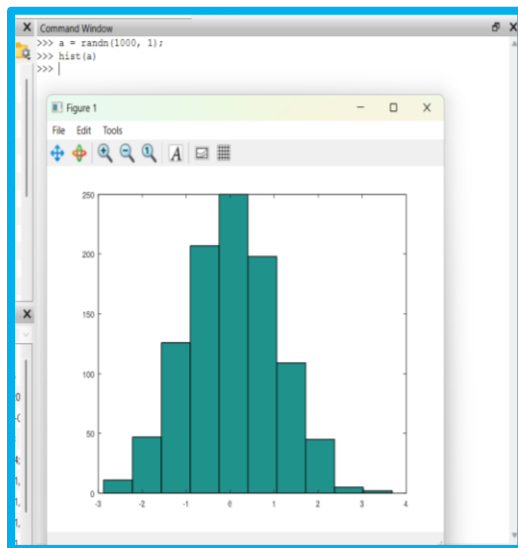
Code Segment and Associated Graph(Step 1)

The code for making a scatter plot for the function $y = x^2$ and its graph.



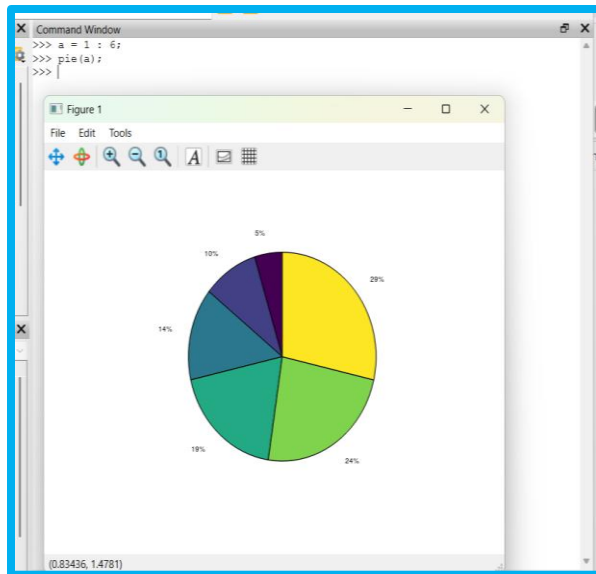
Code Segment and Associated Graph (Step 2)

The code making a random distribution histogram and its graph.



Code Segment and Associated Graph (Step 3)

The code for making a pie chart and its graph.



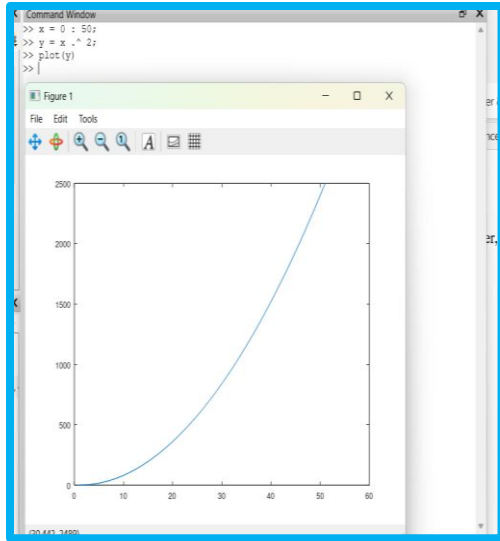
Discussion:

1. Functions and data are much easier to graph on Octave than in Java. Instead of having to export the data and make a graph on excel, Octave can do it directly.
2. Octave supports the creation of many different types of graphs.

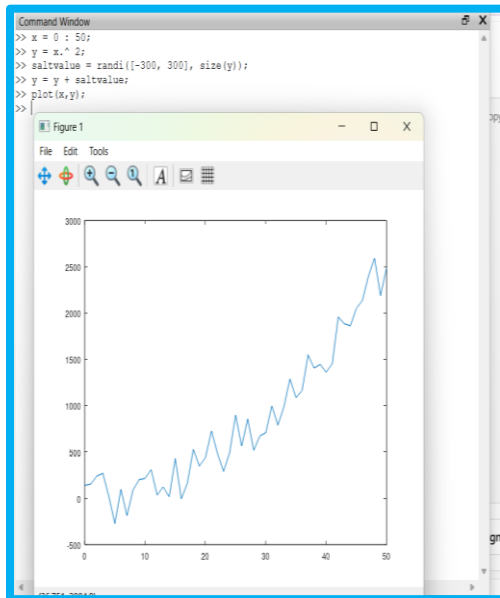
Part 2: Creating the PSS

This section will include code snippets and their respective graphs for a plotter, salter, and smoother for the function $y = x^2$.

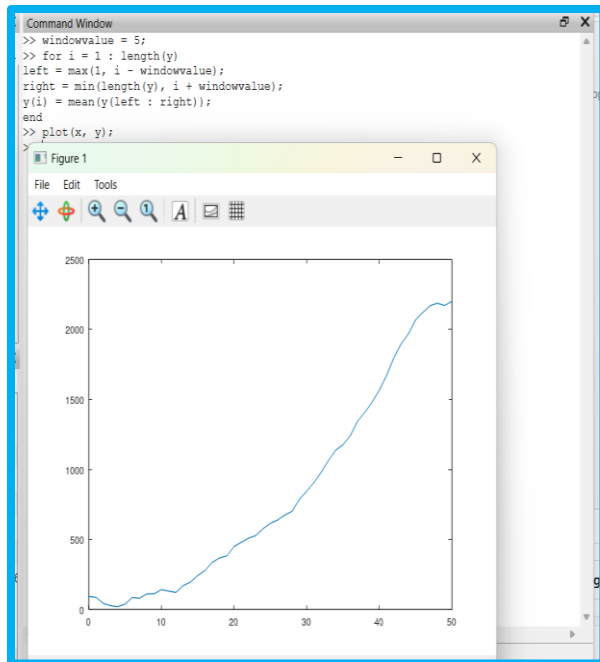
Plotter:



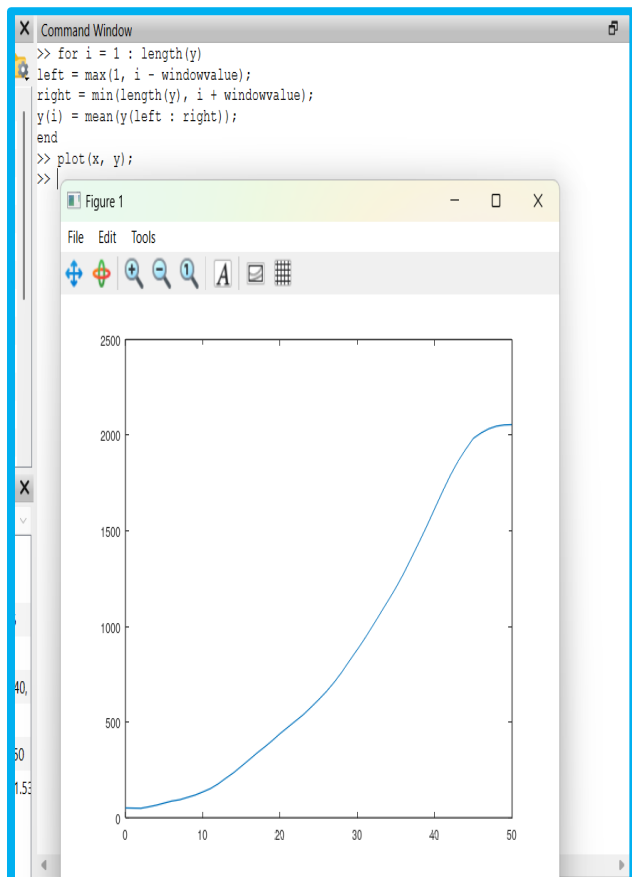
Salter:



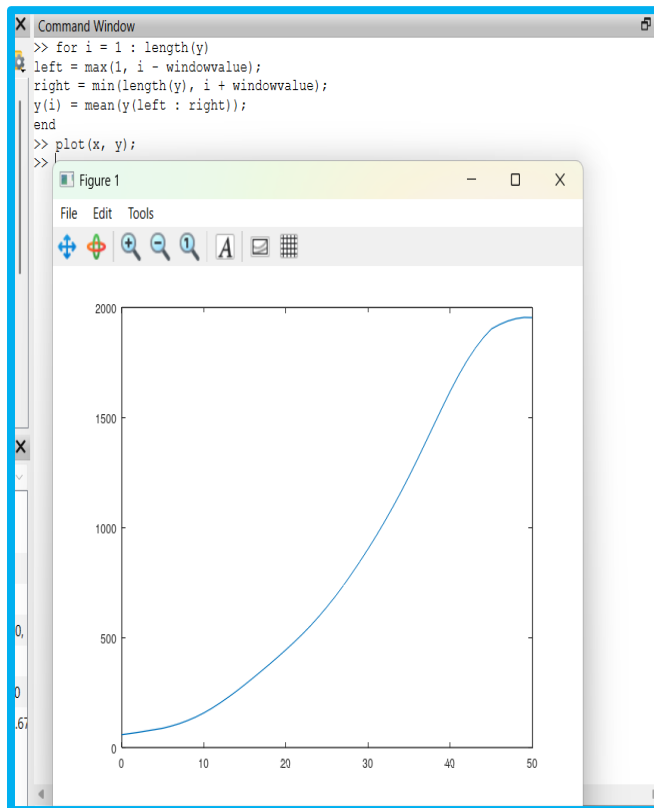
Smoother:



Smoothing #2:



Smoothing #3:



PSS 3 – JFreeChart/Apache Commons Library:

Step 1: Downloading IntelliJ

Normally, I use Eclipse as my IDE, but even when I downloaded the appropriate JARs and added them to my project, I still could not write my import statements for JFreeChart, so I downloaded IntelliJ.

Step 2: Using Gradle to Build My Project

I used Gradle to build my Java project for automatic dependency management. This way, I did not have to download the JFreeChart zip file, extract the file, download the JARs, and then add them to my project pathway.

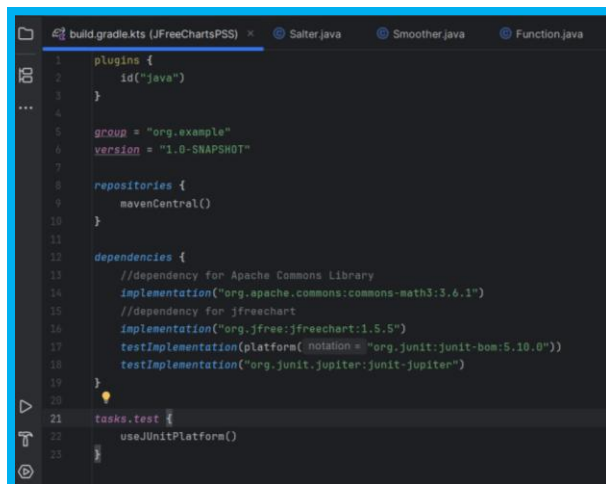
Step 3: Find the Dependencies

I had to find two dependencies, one for JFreeChart and one for Apache Commons. I went to Maven Central and searched them up by name. From there, I was able to copy and paste them in the dependencies section of my build.gradle.kts class. The JFreeChart dependency will allow me to have a pop-up GUI with graphs instead of having to export the data to a CSV file, open Excel, and make a graph using Excel features. The Apache dependency will let me replace some of my plotter, salter, smoother functions with the Apache equivalencies.

Sources:

Maven Central: <https://central.sonatype.com/>

Photos:



Step 4: Research Replacements for Apache Commons

I wanted to try to find as many instances as possible where I can use a feature from the Apache Commons library to replace my original code. I had to do research on the Apache Commons Math Documentation to find apt substitutes for my Java code. Next, I had to find the appropriate import statements for what I wanted to use, which are RandomDataGenerator and DescriptiveStatistics. These were also in the Apache Commons Math Documentation. I copy and pasted in my original Function, Salter, and Smoother classes to get started

First, I replaced the Random class and object from Java with RandomDataGenerator in Apache. The logic is essentially the same: a random number within the salting range is chosen. They both even call the same nextInt method.

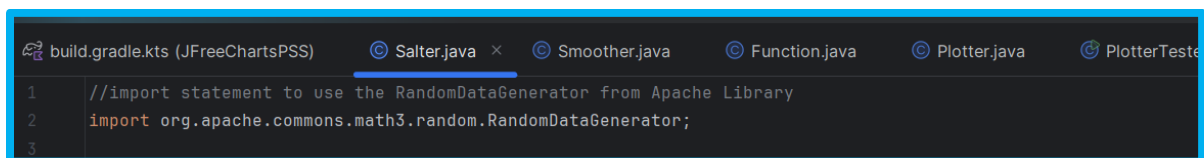
Second, instead of manually calculating the sum and mean from the window value, I used DescriptiveStatistics by adding each y-value to the object, adding the value to the statistics, and getting the mean which is automatically tracked through DescriptiveStatistics.

The Function class remained the same as I could not find a replacement in the Apache Commons Library.

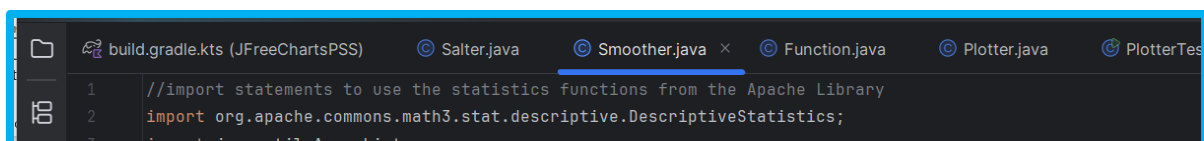
Sources:

Apache Commons Math Documentation: <https://commons.apache.org/proper/commons-math/>

Photos:



```
build.gradle.kts (JFreeChartsPSS)  Salter.java x Smoother.java  Function.java  Plotter.java  PlotterTest.java
1 //import statement to use the RandomDataGenerator from Apache Library
2 import org.apache.commons.math3.random.RandomDataGenerator;
3
```



```
build.gradle.kts (JFreeChartsPSS)  Salter.java  Smoother.java x  Function.java  Plotter.java  PlotterTest.java
1 //import statements to use the statistics functions from the Apache Library
2 import org.apache.commons.math3.stat.descriptive.DescriptiveStatistics;
3
```

```
8      private final RandomDataGenerator gen; 2 usages
9      //Salter constructor
10     public Salter() { 1 usage
11         gen = new RandomDataGenerator();
12     }
13
14     @
15     public void saltValues(ArrayList<Integer> yValues, int saltLow, int saltHigh) { 1 usage
16         //adding a random number within the range to every y-value
17         yValues.replaceAll( Integer integer -> integer + gen.nextInt(saltLow, saltHigh));
18     }
```

```
12         DescriptiveStatistics ds = new DescriptiveStatistics();
13
14
15         //ensuring the window range does not take from below or above the range of the y-values
16         for (int j = Math.max(0, i - windowSize); j <= Math.min(yValues.size() - 1, i + windowSize); j++)
17             //inner loop takes the average of the window and replaces the y-value
18             ds.addValue(yValues.get(j));
19     }
20
21     // Round the mean so it is an integer
22     int smoothedValue = (int) Math.round(ds.getMean());
```

Step 5: Research How to Graph Using JFreeChart

Next, I got rid of any call to the Exporter class in my Function, Salter, and Smoother class. I also did not copy over the Exporter class because it will have to be replaced by a Plotter class which uses JFreeChart. To find out how to do this, I referred to Stack Overflow and YouTube videos. Once I felt confident, I created my Plotter class and found my import statements on JFreeChart Documentation.

From there, I was able to get my Plotter class to use DefaultCategoryDataset to organize and plot my x and y data, creating and labeling a chart using ChartFactory.createLineChart, use ChartPanel to control the size, and use JFrame to make the graphs pop up simultaneously in a window. I made a method to show the graphs and passed through the relevant tile and data.

My tester class remained essentially the same. The main difference was instead of changing the same ArrayList of y-values, I had separate ArrayLists for the original, salted, and smoothed y that took in the y-values from the step before.

Sources:

YouTube Video: <https://www.youtube.com/watch?v=OJa1PPwR8tc>

Stack Overflow: <https://www.youtube.com/watch?v=OJa1PPwR8tc>

JFreeChart Documentation: <https://www.jfree.org/jfreechart/api.html>

Photos:

```
build.gradle.kts (JFreeChartsPSS)  Salter.java  Smoother.java  Function.java  Plotter.java  Plo
1  import org.jfree.chart.ChartFactory;
2  import org.jfree.chart.ChartPanel;
3  import org.jfree.chart.JFreeChart;
4  import org.jfree.data.category.DefaultCategoryDataset;
5
```

```
10
11 @ public void showGraph(ArrayList<Integer> xValues, ArrayList<Integer> yValues, String title) {
12     DefaultCategoryDataset data = new DefaultCategoryDataset();
13
14     // Add the x and y values to the dataset
15     for (int i = 0; i < xValues.size(); i++) {
16         data.addValue(yValues.get(i), rowKey: "Values", xValues.get(i).toString());
17     }
```

```
20     JFreeChart chart = ChartFactory.createLineChart(
21         title,          // Chart title
22         "x",            // x-axis label
23         "y",            // y-axis label
24         data             // Points to plot
25     );
26
```

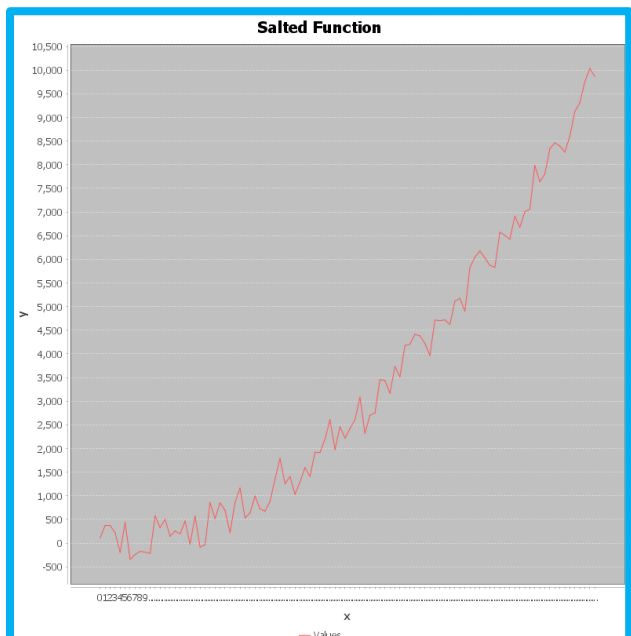
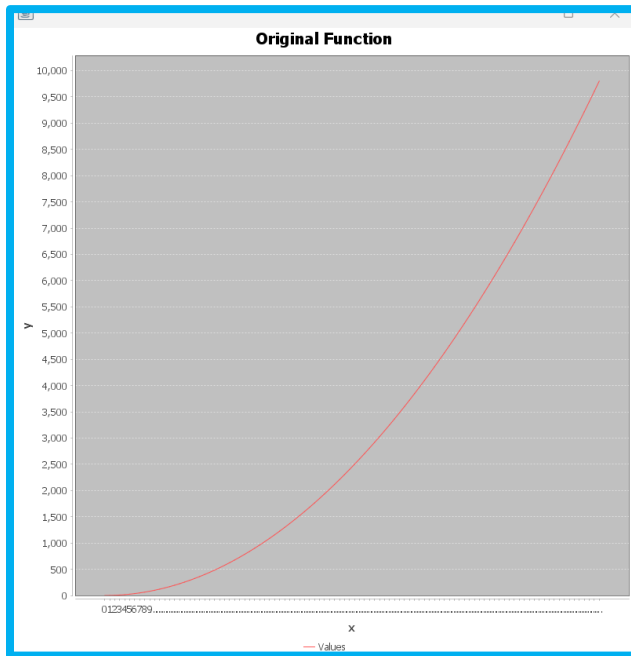
```
31 // Display chart in window and exit the program when window is closed
32 JFrame frame = new JFrame();
33 frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
34 frame.getContentPane().add(chartPanel);
35 frame.pack();
36 frame.setVisible(true);
37 }
38
```

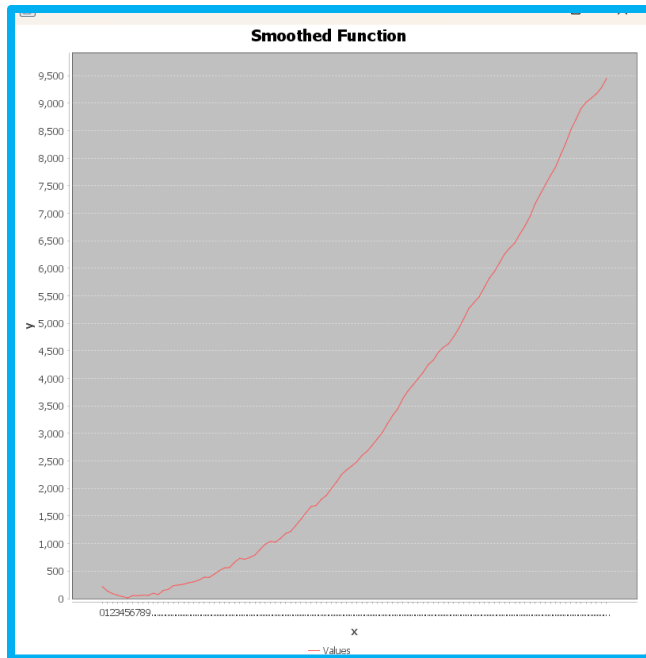
```
28 ChartPanel chartPanel = new ChartPanel(chart);
29 chartPanel.setPreferredSize(new java.awt.Dimension( width: 800, height: 800));
30
```

Step 6: Run the Program

Finally, I just had to run the program, and it was successful. All three graphs, labeled correctly with correct function, popped up in a window at the same time.

Photos:





Discussions:

1. I struggled a lot with this part of the assignment. My computer was the most challenging obstacle. However, once I was able to get the dependencies working on IntelliJ, the research and implementation came easily.
2. Overall, it was more work than making an exporter on Java, exporting the data to CSV, opening Excel, and creating the graph on Excel. But, this was only because a lot of research was required to accomplish making the graph on IntelliJ without external sources.
3. If well versed with JFreeChart, this could be much faster than only using Java and Excel. The most prominent advantages I see are not having to close Excel, reopen it, and re-graph the data after every change you make to your program, and being able to label graphs once with JFreeChart.

Final Conclusions

Through implementing PSS functions in Java, Octave, and with JFreeChart and external JARs, I am able to weigh the pros and cons of each method. Java is the most familiar and easiest to work with, but very tedious for tasks like this which require exporting and viewing data. I was able to program the PSS very quickly, but it definitely slow making the graphs in Excel. Another challenge I faced using Eclipse was the CSV files not showing up in my project explorer until after around 48 hours of waiting. This meant I could not view my results right away, which is frustrating. After doing my research and spending a lot of time learning the language, Octave was definitely the fastest option out of all three. I was able to quickly type the code in only a few lines and see the graph right away. I did not need any project setup or to download any files. However, a drawback is that Octave is only a setup for a future implementation. I cannot formally code on Octave because I cannot go back and edit previous lines. But I can see Octave being useful for quickly experimenting with data to implement it in another language later. JFreeChart posed the most challenge but had the highest reward. Although it was difficult to find the correct files and figure out how to build projects correctly on the IDE, it was very easy to create a good-quality graph on IntelliJ.

Overall, I would use Java alone when I need control over data structures, analyzing data without needing a visual aid, and when the main focus is on algorithms. I would use Octave to execute fast numerical analysis, fast plotting abilities, and when I want to quickly test logic I will use later in another language. Finally, I would use Java with JFreeChart when I want a Java project with a visual component and when I need to access external libraries to complete tasks such as mathematical functions.