Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito

Part 1 – Using Standard Java Library and Excel

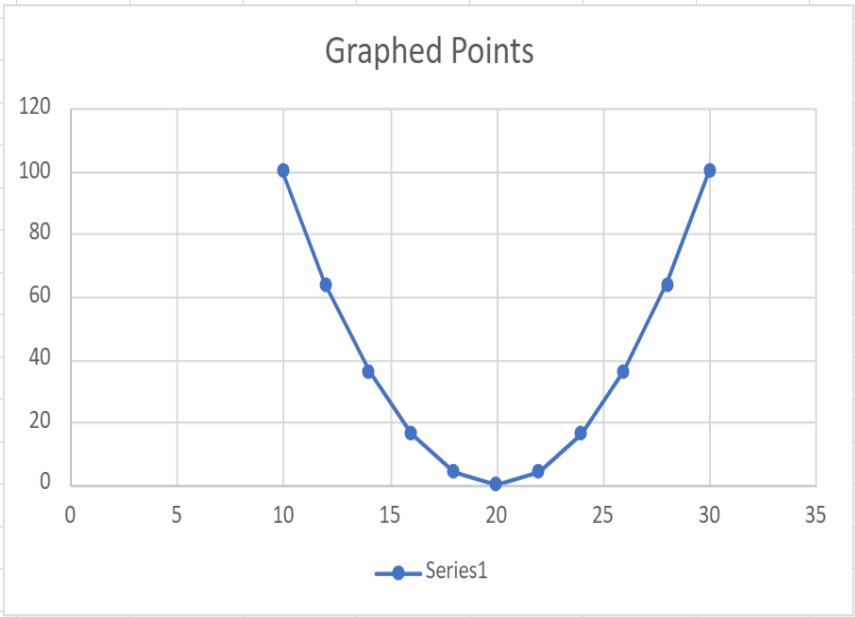
Plotter

To tackle this project, I decided to start by making a class for Point objects. The code for this is found in Point.java I made this class so I could access the x and y values easily and store them as pairs. This would come to be useful later when writing to CSV, salting, and smoothing. After setting up my class to handle point objects I made Plotter.java this is where the lower, upper bound, and interval between points can be specified to make an ArrayList of Point objects. My function method calculates each output for the function $y = (x - 20)^2$. This class also contains a method to write the points to a CSV file by passing the ArrayList of Point objects.

I generated multiple CSV files with x values and their corresponding y values to check the output of my Plotter.java class and obtained the results below. A screenshot of the inputs for each graph are located below each graph generated in excel.

First Graph:

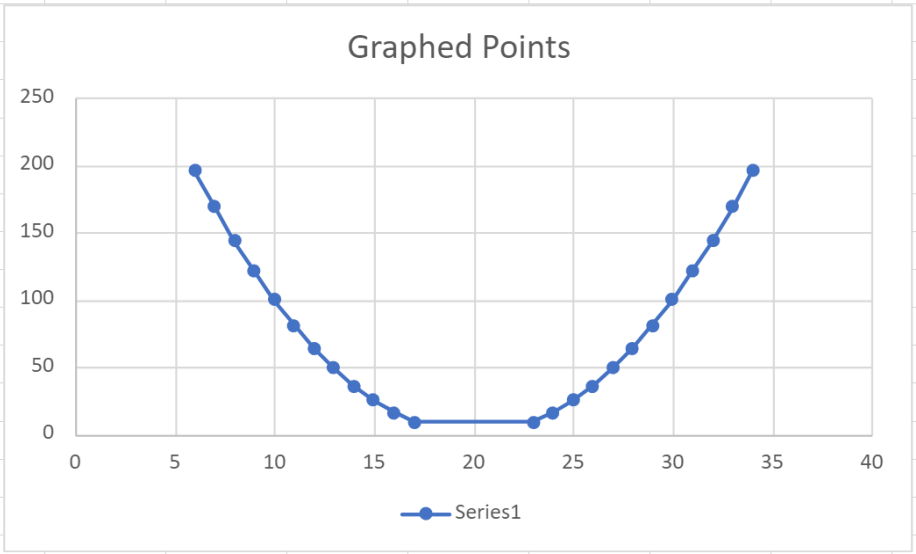
xValue	yValue
10	100
12	64
14	36
16	16
18	4
20	0
22	4
24	16
26	36
28	64
30	100




```
ArrayList<Point> points = test.createPoints( intervalBetweenPoints: 2,  lowerBound: 0,  upperBound: 100);
```

Second Graph:

xValue	yValue
6	196
7	169
8	144
9	121
10	100
11	81
12	64
13	49
14	36
15	25
16	16
17	9
23	9
24	16
25	25
26	36
27	49
28	64
29	81
30	100
31	121
32	144
33	169
34	196

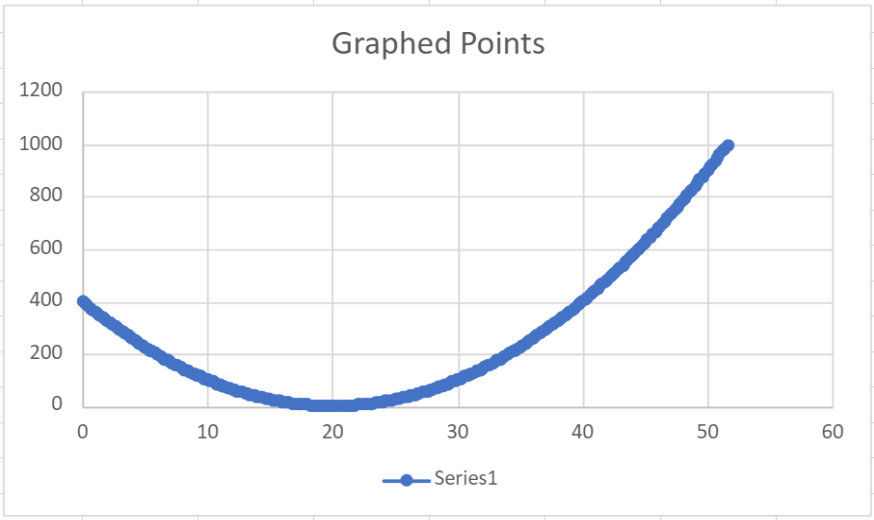



```
ArrayList<Point> points = test.createPoints( intervalBetweenPoints: 1,  lowerBound: 5,  upperBound: 200);
```

Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito

Third Graph:

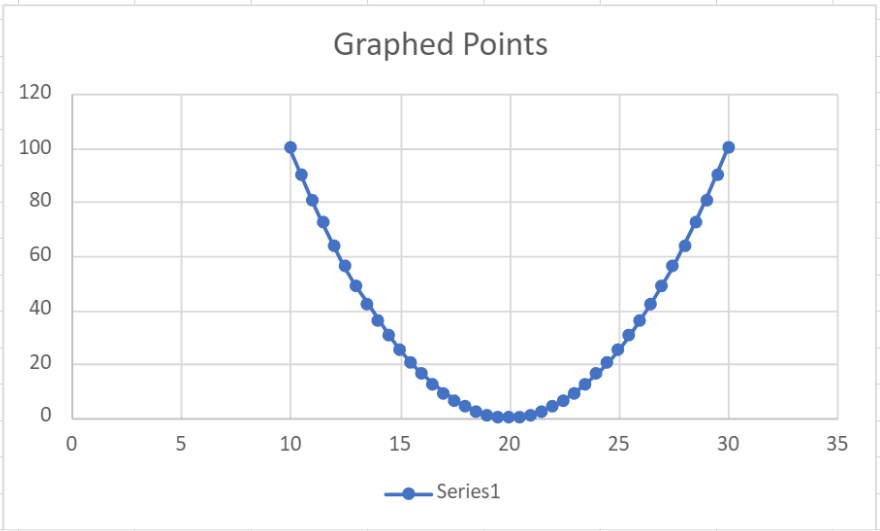
xValue	yValue
0	400
0.2	392.04
0.4	384.16
0.6	376.36
0.8	368.64
1	361
1.2	353.44
1.4	345.96
1.6	338.56
1.8	331.24
2	324
2.2	316.84
2.4	309.76
2.6	302.76
2.8	295.84
3	289
3.2	282.24
3.4	275.56
3.6	268.96
3.8	262.44
4	256
4.2	249.64
4.4	243.36




```
ArrayList<Point> points = test.createPoints( intervalBetweenPoints: .2,  lowerBound: 0,  upperBound: 1000);
```

Fourth Graph:

xValue	yValue
10	100
10.5	90.25
11	81
11.5	72.25
12	64
12.5	56.25
13	49
13.5	42.25
14	36
14.5	30.25
15	25
15.5	20.25
16	16
16.5	12.25
17	9
17.5	6.25




```
ArrayList<Point> points = test.createPoints( intervalBetweenPoints: .5,  lowerBound: 0,  upperBound: 100);
```

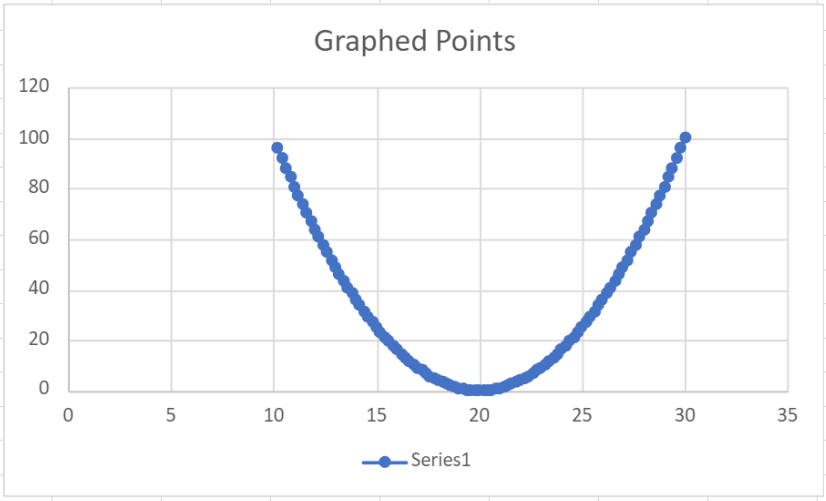
Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito

Salting

The next portion of this involved salting the data that I plotted and wrote to a CSV file with Plotter.java. I created the class Salter.java which contains a function to salt the specified y value and a function to write the salted data to a CSV file. For the salting function you can specify a minimum value and maximum value. saltYValue() function will salt a singular y value by taking a random number within the range specified and randomly adding or subtracting it to a y value. When saltToCsv() is called it will iterate through the point objects made in the csv file and calculate the salted value for every y value and write the array to a new CSV. Below is the process that I went through while testing salting values.

Original Graph To Be Salted:

xValue	yValue
10.2	96.04
10.4	92.16
10.6	88.36
10.8	84.64
11	81
11.2	77.44
11.4	73.96
11.6	70.56
11.8	67.24
12	64
12.2	60.84
12.4	57.76
12.6	54.76
12.8	51.84
13	49
13.2	46.24
13.4	43.56
13.6	40.96
13.8	38.44
14	36
14.2	33.64
14.4	31.36
14.6	29.16
14.8	27.04
15	25
15.2	23.04
15.4	21.16
15.6	19.36
15.8	17.64
16	16

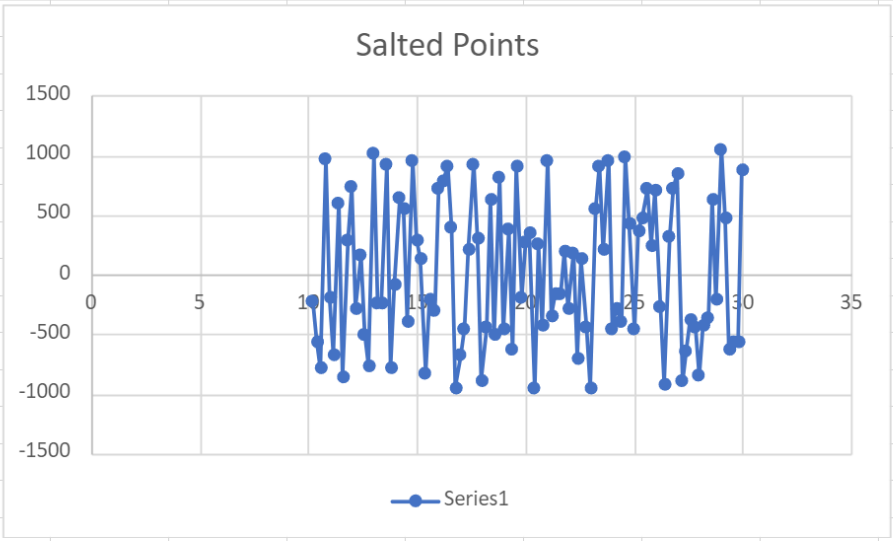



```
ArrayList<Point> points = test.createPoints( intervalBetweenPoints: .2,  lowerBound: 0,  upperBound: 100);
```

Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito

First Salting:

xValue	yValue
10.2	-223.487
10.4	-556.306
10.6	-784.453
10.8	970.6691
11	-197.574
11.2	-665.255
11.4	603.2276
11.6	-853.881
11.8	291.3897
12	738.2532
12.2	-281.311
12.4	161.3927
12.6	-499.269
12.8	-761.559
13	1015.473
13.2	-240.374
13.4	-230.989
13.6	922.978
13.8	-775.875
14	-84.2815

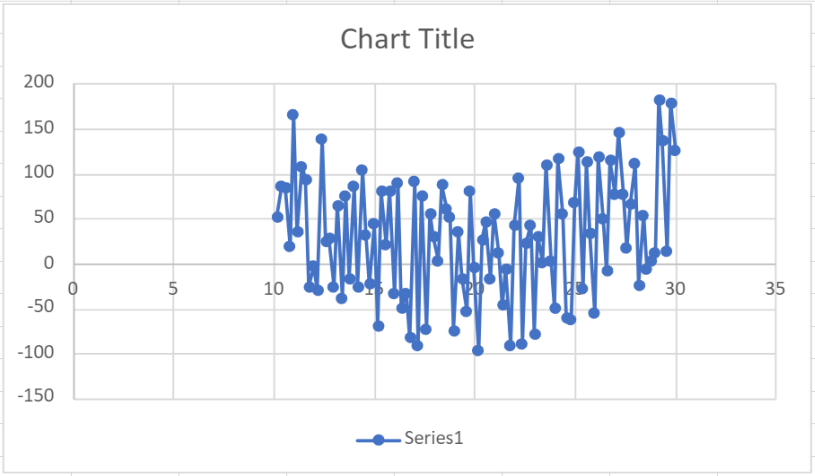



```
test.saltToCSV(100, 1000);
```

For this test I set my minRange to 100 and maxRange to 1000. In this graph there are not enough points in the original graph and the salting values are too drastic with the minimum range starting at the upperBound of the original chart.

Second Salt:

xValue	yValue
10.2	51.88478
10.4	86.02317
10.6	84.28171
10.8	18.71214
11	165.9695
11.2	34.5219
11.4	107.7684
11.6	94.02669
11.8	-25.5311
12	-3.21819
12.2	-29.0447
12.4	139.1325
12.6	24.77714
12.8	27.96718
13	-25.4514
13.2	64.36409

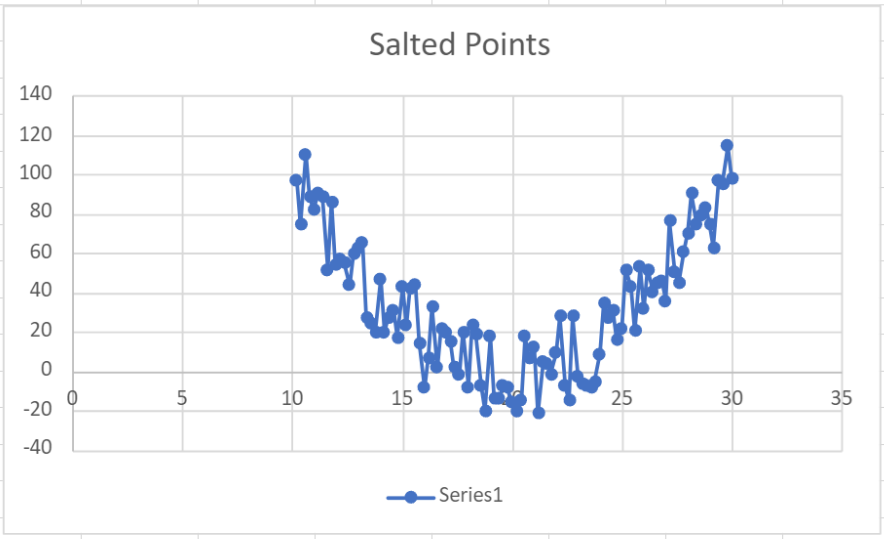



```
test.saltToCSV(0, 100);
```

For the second salt I set the minRange to 0 and the maxRange to 100. This still seemed like a very drastic salt that can be attributed to the small number of plots in the original graph.

Third Salt:

xValue	yValue
10.2	97.52559
10.4	74.41261
10.6	110.5261
10.8	88.53861
11	82.33576
11.2	90.47171
11.4	88.82123
11.6	51.99352
11.8	86.15527
12	54.47939
12.2	57.3689
12.4	54.80305
12.6	43.83926
12.8	59.58559
13	62.8779
13.2	65.7464



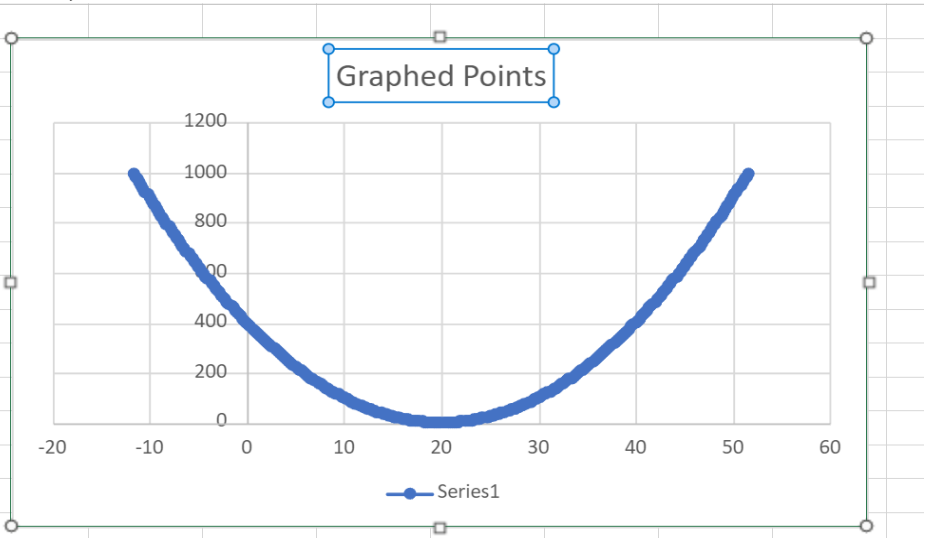

```
test.saltToCSV(0, 25);
```

With the minRange set to 0 and the maxRange set to 25 this seemed to be a more reasonable salting considering the small range of the original graph and the low number of points.

Graph with large number of data points:

After testing Salter.java with a small number of data points I generated a graph with many points and a bigger range to test, seen below.

xValue	yValue
-11.6	998.56
-11.4	985.96
-11.2	973.44
-11	961
-10.8	948.64
-10.6	936.36
-10.4	924.16
-10.2	912.04
-10	900
-9.8	888.04
-9.6	876.16
-9.4	864.36
-9.2	852.64
-9	841
-8.8	829.44
-8.6	817.96

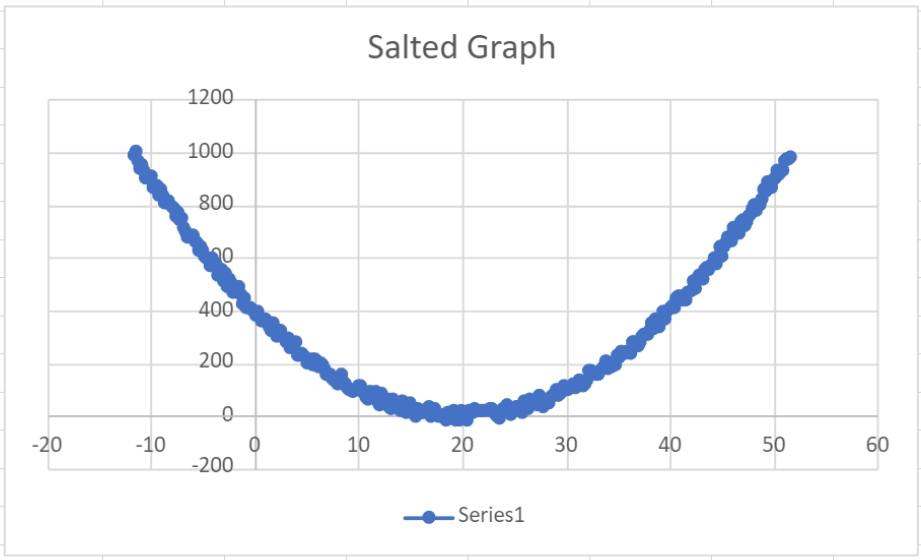



```
ArrayList<Point> points = test.createPoints( intervalBetweenPoints: .2,  lowerBound: -12,  upperBound: 1000);
```

Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito

First Salt:

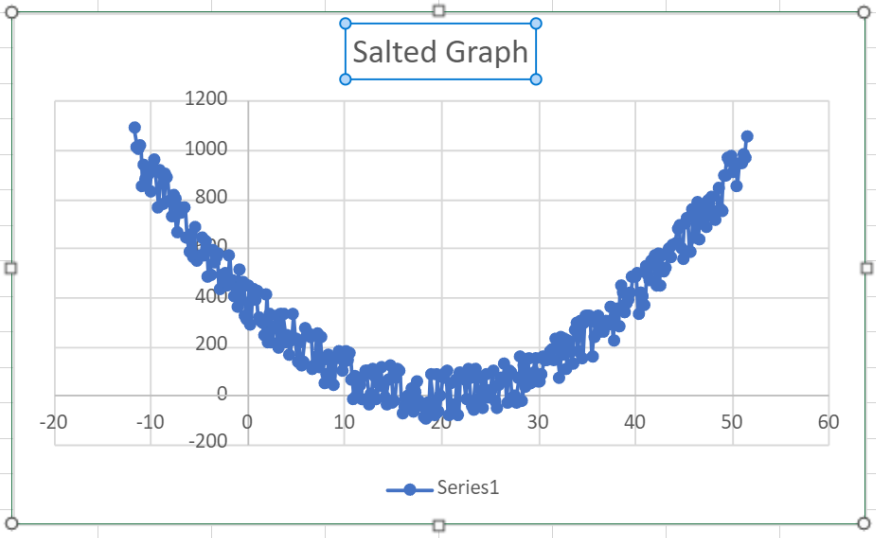
xValue	yValue
-11.6	987.092
-11.4	1002.031
-11.2	967.1465
-11	936.7672
-10.8	954.4723
-10.6	933.0171
-10.4	901.3882
-10.2	902.7114
-10	911.832
-9.8	868.6526
-9.6	872.178
-9.4	874.8249
-9.2	837.5632
-9	856.2331
-8.8	837.2651
-8.6	805.8828




```
test.saltToCSV(0, 25);
```

I began by using the same salting method that seemed to work for the dataset with the smaller range and low number of points, but when applied to the graph with a larger range and a high number of points it did not seem to provide enough variation.

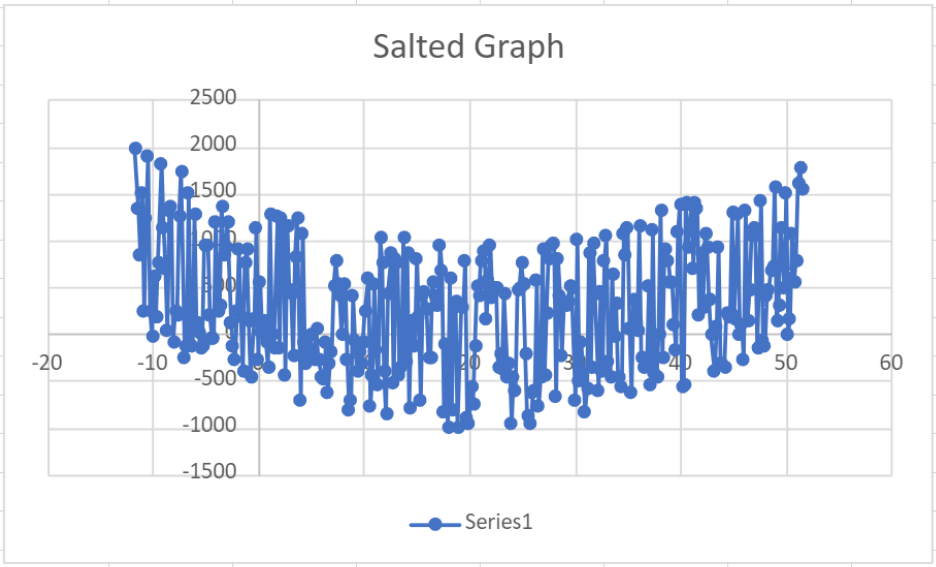
xValue	yValue
-11.6	1091.386
-11.4	1008.226
-11.2	1003.875
-11	1020.272
-10.8	849.6916
-10.6	937.893
-10.4	877.6161
-10.2	902.1198
-10	829.8525
-9.8	932.6637
-9.6	959.4148
-9.4	916.4519
-9.2	767.8906
-9	918.2741
-8.8	902.7058
-8.6	778.8146
-8.4	902.4387
-8.2	891.045




```
test.saltToCSV(0, 100);
```

With the minRange set to 0 and the maxRange set to 100 this provided some variation, but I did not feel like the variation was significant enough.

xValue	yValue
-11.6	1972.44
-11.4	1337.651
-11.2	846.1774
-11	1508.249
-10.8	243.4363
-10.6	1239.979
-10.4	1890.849
-10.2	251.4119
-10	-14.508
-9.8	607.6821
-9.6	177.2572
-9.4	751.2104
-9.2	1825.529
-9	1144.043
-8.8	693.502
-8.6	28.46637
-8.4	1320.689




```
test.saltToCSV(100, 1000);
```

For my last salting test on the dataset with many points I set my minRange to 100 and my maxRange to 1000. This one offered the most variance and lost a bit more of its shape. I assume that is from setting the minRange to 100 no value could remain unchanged.

Graph Smother

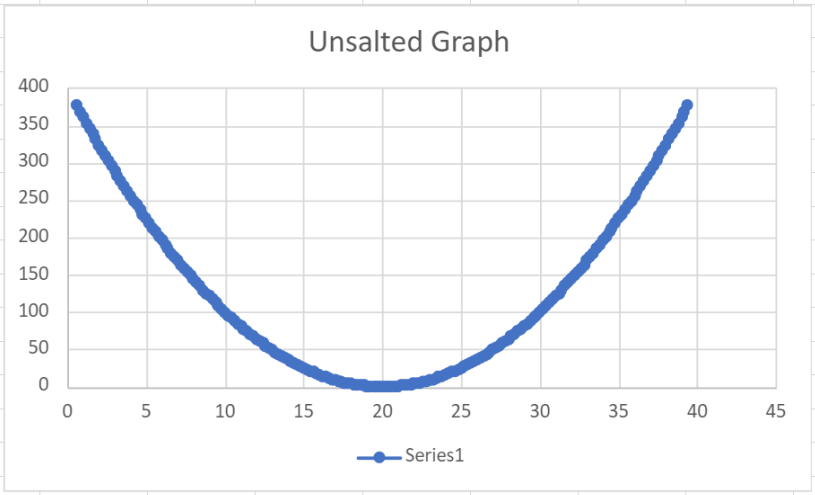
The last portion of the project was the graph smoother. I created GraphSmoother.java for this portion. My smoother() function takes the ArrayList of Point objects and the window size. With these values it extracts the number of y points that are on the left or right of the current point that corresponds with the windowValue. With that it will average for each y value and place it into the ArrayList with all the smoothed points. This returns a new ArrayList after it is complete. The function called smoothToCsv() function to generate the ArrayList that is then written to the CSV file. Below are the values that I used to create the following smoothed graphs.

```
ArrayList<Point> points = test.createPoints( intervalBetweenPoints: .2, lowerBound: 0, upperBound: 380);
test.saltToCsv(0, 100);
test.smoothToCsv( inputFile: "smoothed-points.csv", windowValue: 4);
```

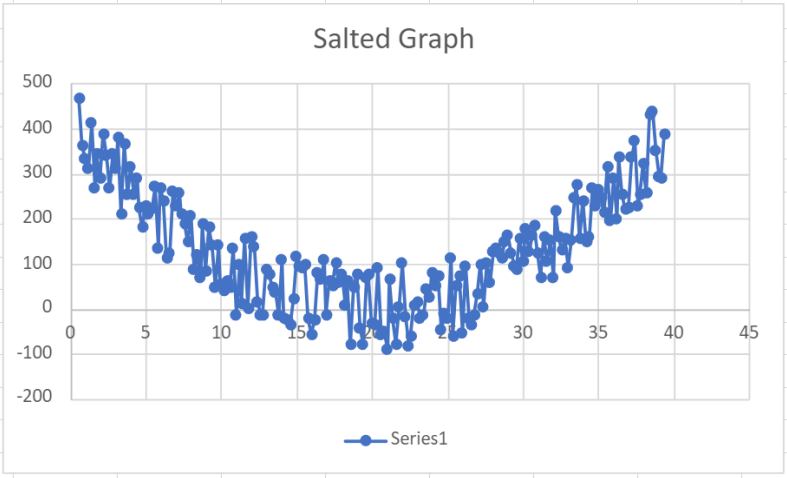
Part 1, 2, & 3 – Report, Results, and Documentation

Alexis Petito

xValue	yValue
0.6	376.36
0.8	368.64
1	361
1.2	353.44
1.4	345.96
1.6	338.56
1.8	331.24
2	324
2.2	316.84
2.4	309.76
2.6	302.76
2.8	295.84
3	289
3.2	282.24
3.4	275.56
3.6	268.96
3.8	262.44
4	256

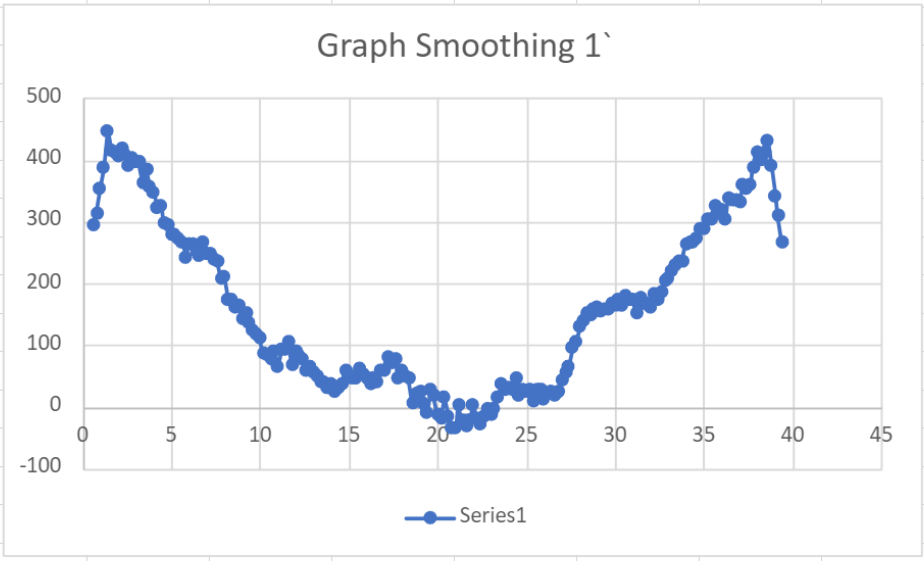


xValue	yValue
0.6	467.2774
0.8	361.1347
1	333.592
1.2	311.017
1.4	412.6388
1.6	269.1975
1.8	343.34
2	288.1742
2.2	389.1635
2.4	341.9099
2.6	267.0441
2.8	342.8688
3	309.9458
3.2	379.1836
3.4	210.5516
3.6	367.5647
3.8	253.5187
4	316.543
4.2	252.5319

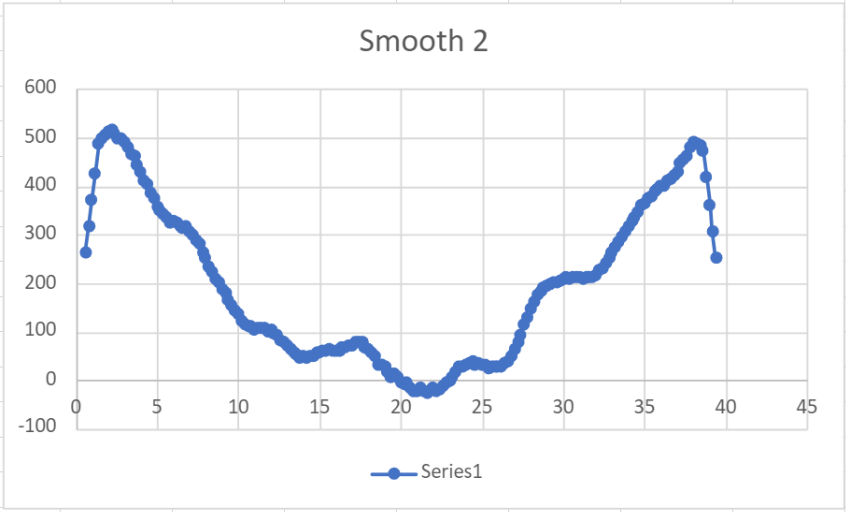


Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito

xValue	yValue
0.6	294.1171
0.8	314.499
1	353.9736
1.2	387.1736
1.4	448.5217
1.6	414.9206
1.8	412.4271
2	406.691
2.2	419.1808
2.4	409.0922
2.6	392.4032
2.8	404.9094
3	396.4621
3.2	396.0392
3.4	363.788
3.6	386.1571
3.8	357.0135
4	348.7349

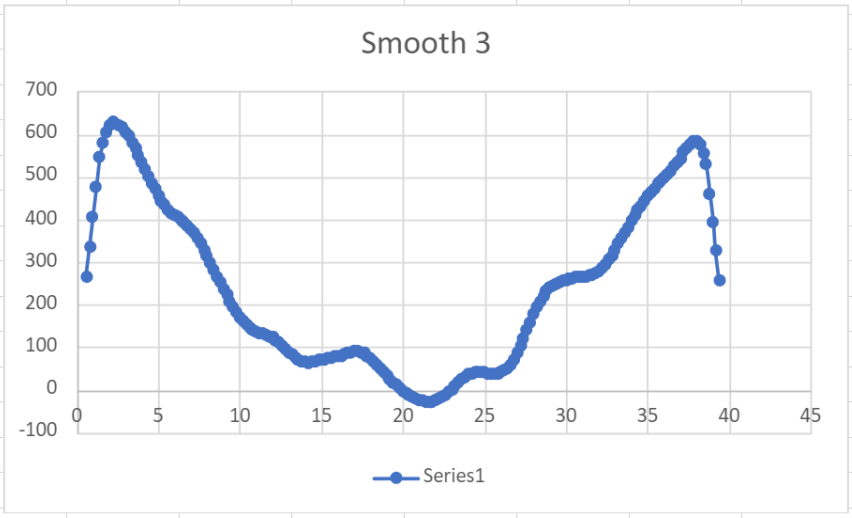


xValue	yValue
0.6	261.5503
0.8	315.9631
1	372.4508
1.2	427.4372
1.4	487.5033
1.6	497.675
1.8	507.1014
2	512.7513
2.2	515.4736
2.4	507.6522
2.6	499.1745
2.8	497.454
3	490.1884
3.2	481.3299
3.4	466.3908
3.6	460.9048
3.8	444.1074
4	430.5618

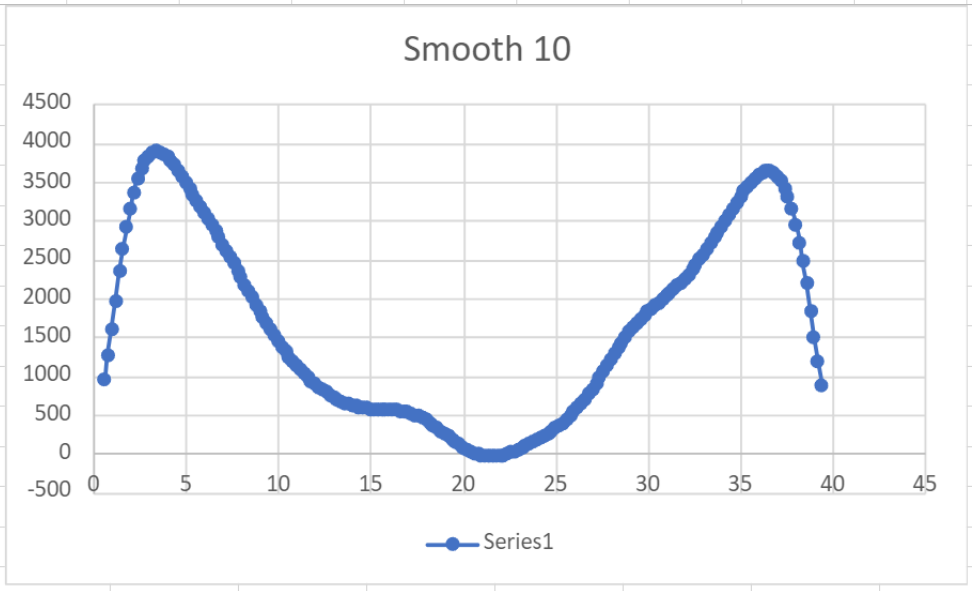


Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito

xValue	yValue
0.6	265.8069
0.8	334.8178
1	405.2665
1.2	476.2337
1.4	548.1762
1.6	580.2104
1.8	604.2901
2	620.6217
2.2	628.8059
2.4	627.0566
2.6	622.0863
2.8	616.0967
3	606.608
3.2	594.8867
3.4	581.1438



xValue	yValue
0.6	950.1755
0.8	1258.138
1	1596.882
1.2	1962.765
1.4	2351.403
1.6	2646.115
1.8	2918.68
2	3161.802
2.2	3368.887
2.4	3534.255
2.6	3666.503
2.8	3766.103
3	3834.754
3.2	3875.283



After I completed the testing for the smoothing function it is shown that the results are in fact smoothing the graph, but if you sooth the values too many times you will start to lose the original function range as you are calculating the mean within a window over many iterations.

Reflections and Possible Improvements

Point.java and Plotter.java:

Overall I was satisfied with the way these two classes worked throughout the full project. Making point objects worked very well. Though I feel there could be some improvements. I chose a fairly simplistic parabala that was shifted to the right I feel if I were more creative with my function it could've had more interesting results.

Salter.java:

I was also satisfied with my Salter.java class and the actual saltYValue() function was clean and straightforward. I feel like saltToCsv() funciton was a little verbose and there may have been

better ways to write the objects to the CSV file. I also had to use trim() due to the formatting of my CSV files.

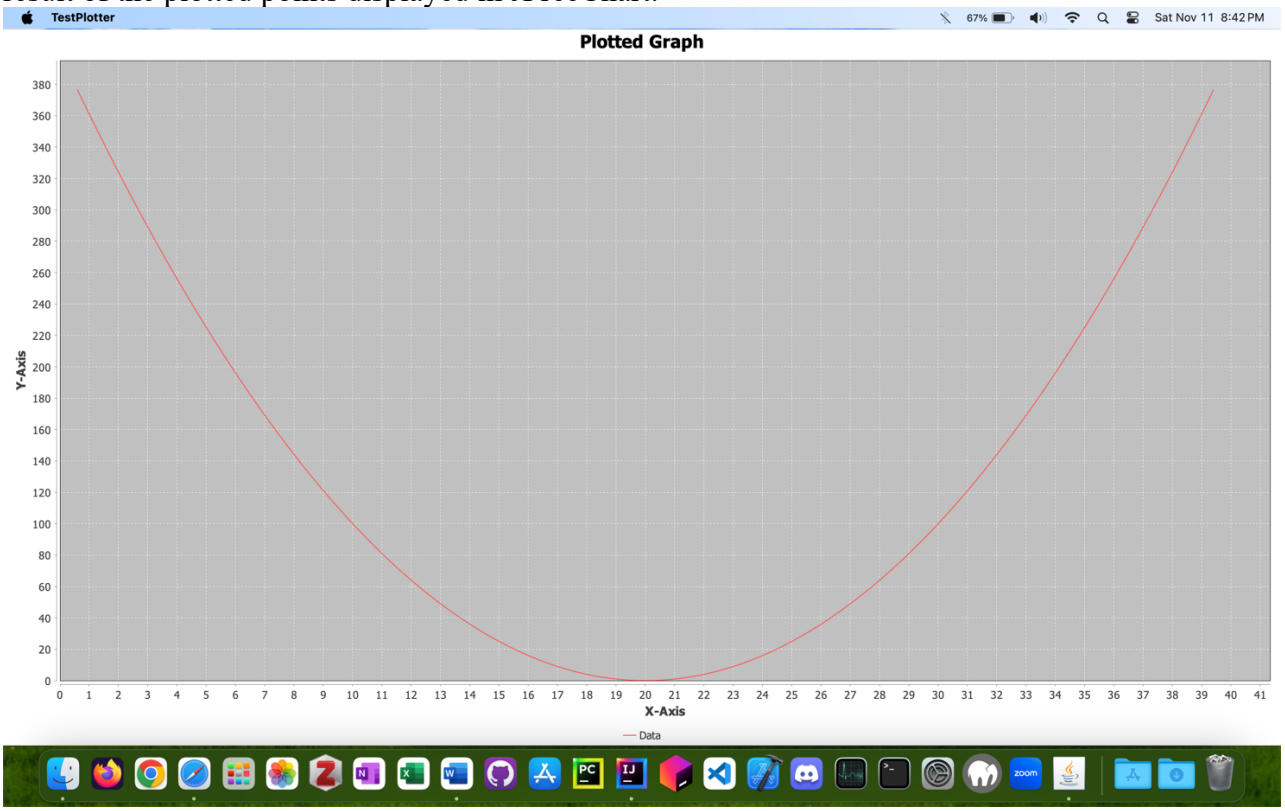
GraphSmoother.java:

This portion of the project used a rolling mean over all of the y values to smoooth the graph. This meant at both ends of the parabala there would be some distortion that was not seen in the middle of the smoothed graph because there were not enough data points to perform a mean on the full window size and adter each smoothing iteration this would become more apparent. If there were something implemented such as handeling the points in the center and then extrapolating the points at the end it could have faired better.

Part 2 – Using Apache Commons and JFreeChart

Plotting

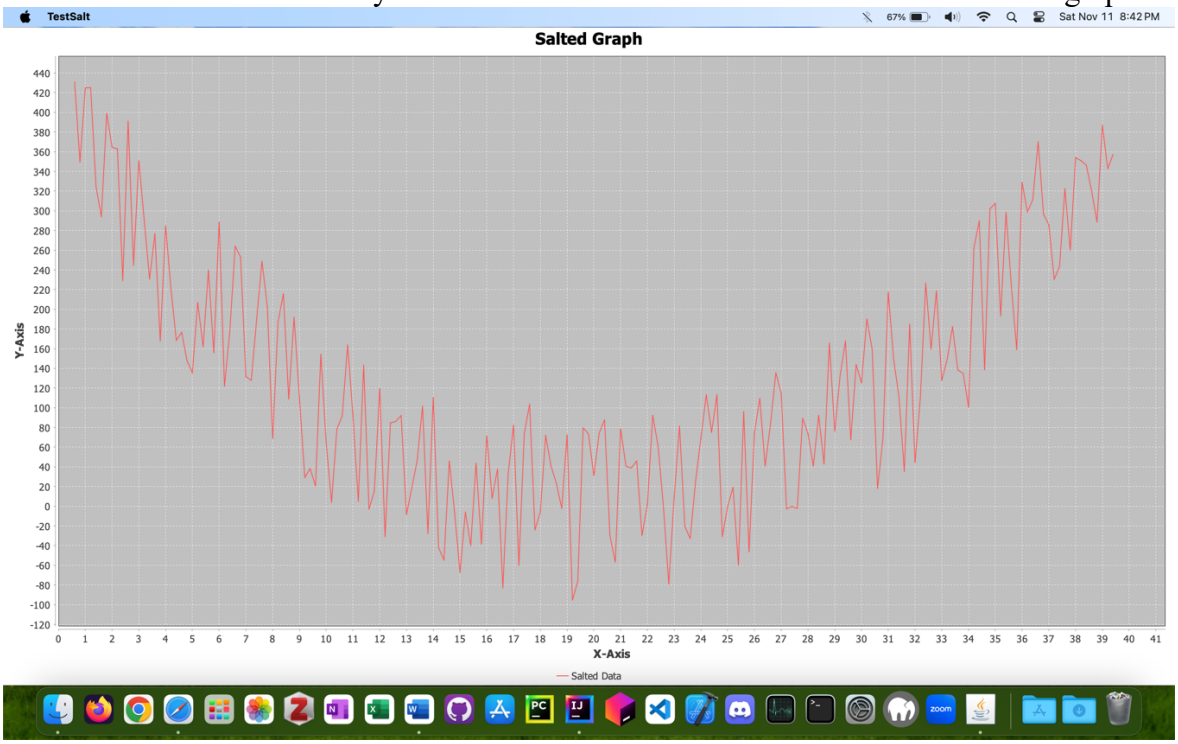
I began by thinking about how I could use Apache Commons Library and JFreeChart to make the programming easier and quicker. I soon realized that since the data for each chart could be passed in double[][] lists. I realized I did not need to have a Point object and did not need to write to a CSV file. So, with that in mind I began to work on Plotter.java. I had to change my logic to instead plot the points and add them to the double[][] list instead of writing them to a CSV. This was much quicker to implement than what I had in part 1 of project 2. Below is a result of the plotted points displayed in JFreeChart.



Salting

For the salting portion of part 3 I began working on Salt.java. I looked at how I could improve the saltYValue() function, but I didn't find anything specific, so I decided to keep that the same.

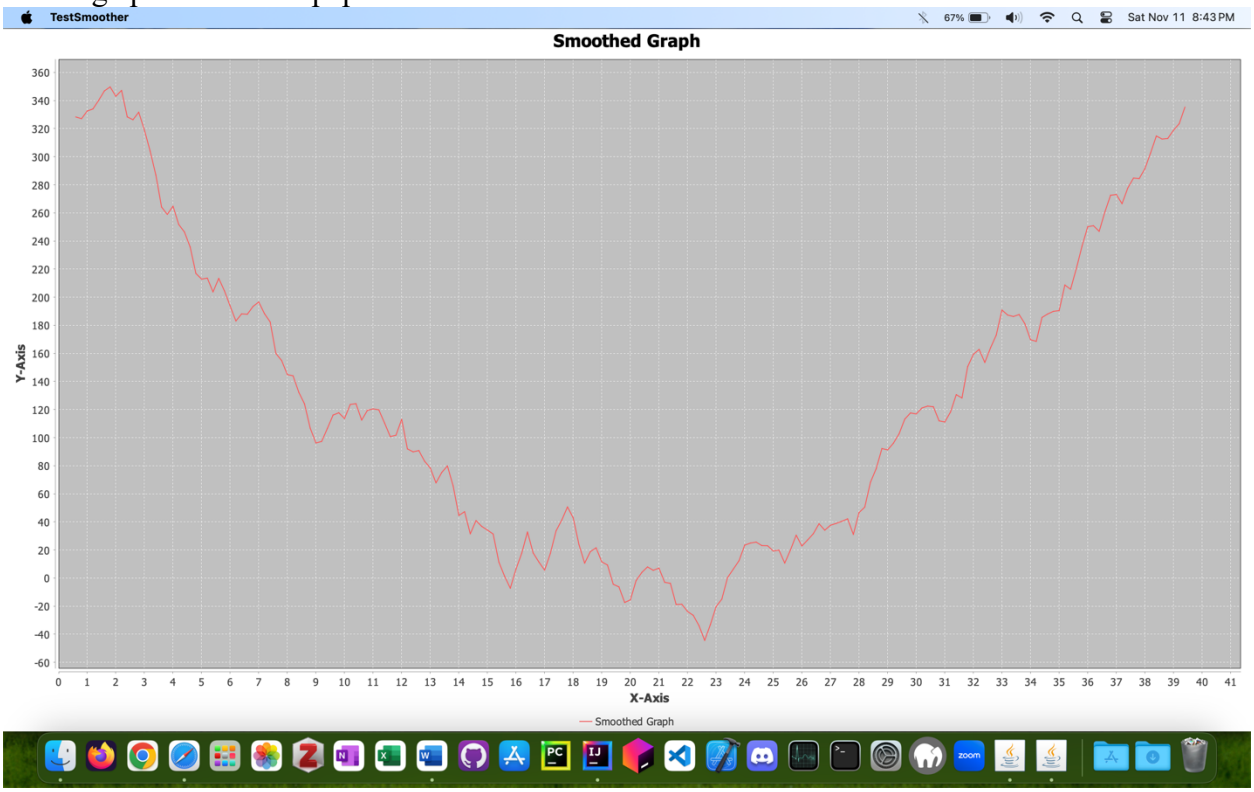
Generating the salted plot did have to change though as I could now directly pass my list with the plotted points to the method with the minRange and maxRange. This made it easier to just plot the chart with JFreeChart after the y values were salted. Below are the results of a salted graph.



Smoothing

For the final portion of part 3 I started working on my smoother named Smoother.java. This portion of the project gave me the most trouble since The Apache Commons method for calculating mean used a different data structure than my JFreeChart output, double[] not double[][], and the getMean() function combined with DescriptiveStats does not actually return a rolling mean. I referred to an example of how to calculate a rolling mean on the Apache Commons Library website and applied it to my logic. I had to make a copy of the y values into a double[] list so I could call getMean() on each y value as I iterated through the list using DescriptiveStats to set my window value. Then I inserted the averaged y values back into the smoothed data array that was already populated with the x values as they never need to be modified. Below is a demonstration of the smoothed salted graph that was shown in in the

"Salting" portion of the paper.



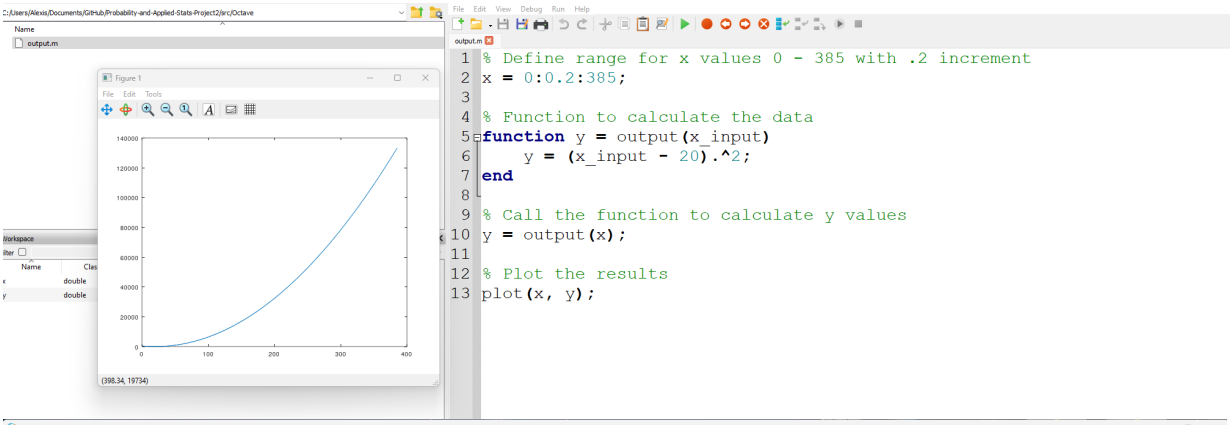
Part 3 – Learning Octave: Documentation and Results

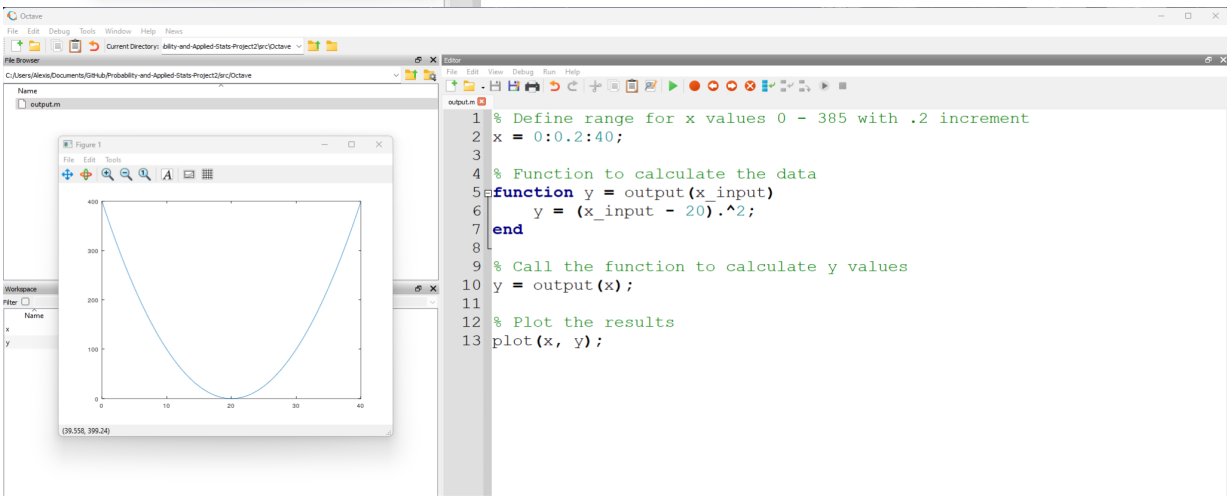
When I first opened Octave, it was overwhelming, I started looking for YouTube tutorials and reading the documentation for Octave online. I stumbled across a few helpful tutorials on TutorialsPoint.com as well. After reading up and watching a few videos I started to explore how to create a project I ended up finding the documentation page and started reading. After reading through some more documentation, I ran into a lot of errors, so I decided to watch a beginner's playlist. The tutorial helped me learn basic operations plotting using saving and loading data and functions and code files.

After I began my project, I made a function and tried to run the file but got a mismatch error between the function and file name. I then discovered that placing "1;" in the beginning of the file was necessary to denote a script file if you did not want to match your function name with the file or needed multiple functions in one file.

After combining all this knowledge, I attempted to plot one of the x y graphs I made in my plotting program. After plotting my first graph I realized that the function was not in the range I wanted. I began to search how to fix that. My error was putting in the values thinking of them as a range. Whereas, in Octave that is the range for just the x values and does not directly restrict the y values.

Part 1, 2, & 3 – Report, Results, and Documentation
Alexis Petito



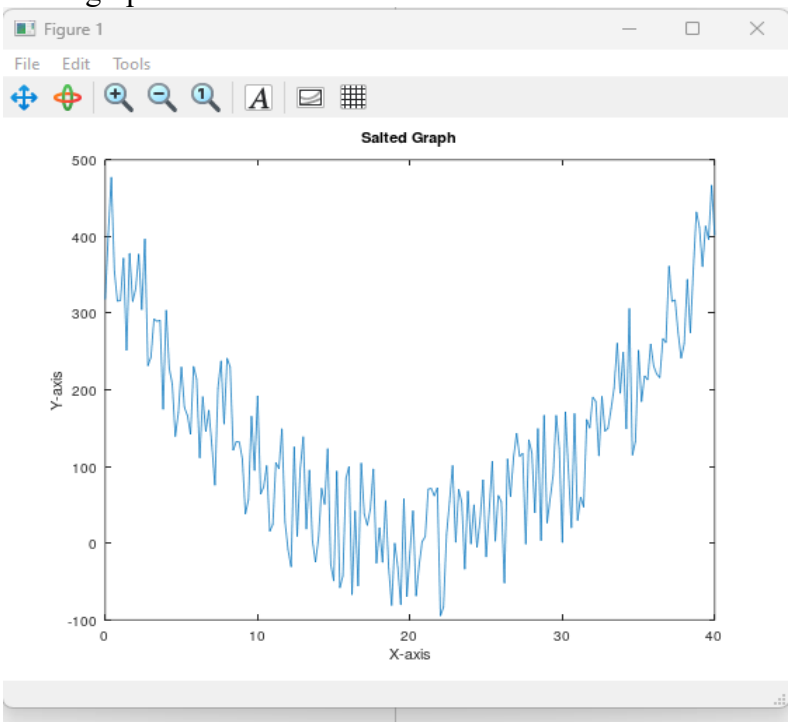


I modified the x inputs to focus on the range I wanted instead of my misconception of attempting to directly modify the y values. This confirmed that my plot function was working properly. I then investigated how to load write to and use csv files in octave as this would be necessary if I wanted to store and read the data for other parts of the project such as salting or smoothing. After figuring out that you must store your data in a matrix then save it to a CSV, I did that and moved on to the next portion of the project which was salting the data.

Salting

At this point I had read a lot of documentation on functions and watched videos about plotting, so a lot of the issues I was having with Octave had more to do with defining functions and not having conflicts than having issues with the actual logic, so I spent most of my time researching about that. After this I was able to implement my salter and I tested my script and plotted the

salted graph which can be seen below.



Graph smoother

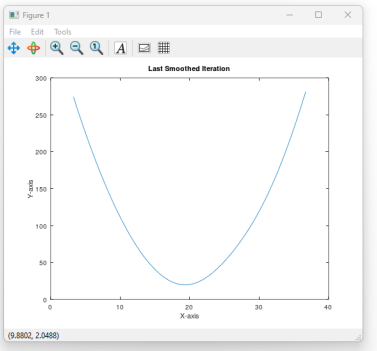
Then I began working on how to smooth my graph in Octave which led me to the docs page on statistics on sliding windows of data. I began to read through to see what function I could use and saw this "Many of the most commonly desired functions, such as the moving average over a window of data (movemean) are already provided." I started to create my smoother around this function. This was much easier than implementing it by hand so that did not take long.

I wanted my smoother to be able to smooth for multiple iterations that the user could define so I began to research about how to loop in octave and found that it was quite like most programming languages, so I implemented that and added formatting so that it would generate a labeled CSV file for each iteration but also plot the final iteration. An example of the program running can be seen below.

```
    % Create a unique output file for each iteration
    unique_output_file = sprintf('%s_iteration%d.csv', output_file, i);

    % Save the smoothed data to the output file
    csvwrite(unique_output_file, smoothed_data);
  end
  % Plot the last smoothed iteration
  last_output_file = sprintf('%s_iteration%d.csv', output_file, iterations);
  last_smoothed_data = csvread(last_output_file);
  plot(last_smoothed_data(:, 1), last_smoothed_data(:, 2));
  title('Last Smoothed Iteration');
  xlabel('X-axis');
  ylabel('Y-axis');
nd

 Call the smooth_and_save with window value and iterations
mooth_and_save('salted-points.csv', 'smoothed-points', 15, 20);
```



References:

For Understanding Octave

https://www.youtube.com/watch?v=X0xLTKRWPgo&list=PL1A2CSdiySGJ6oZe6XB-TTCFuHc5Fs1PO
https://www.youtube.com/watch?v=aD8k4pYUBOk
https://docs.octave.org/latest/Introduction.html.

For Smoother
https://docs.octave.org/latest/Statistics-on-Sliding-Windows-of-Data.html