Project 2 - First Assignment: Plotter, Salter, & Smoother

Kaysey Nguyen

Stockton University

CSCI-3327 - Probability And Applied Statistics

Byron Hoy

December 14, 2023

Contents

Background

What is the Assignment?

For this assignment, plotter, salter, and smoother is working with randomized points of data from a function. To put simply, plotter is generating randomized points, salter is messing up the data from plotter, and smoother is averaging the data from salter to remove the mess and make a clearer image of the function. There will be three experiments or parts for this assignment and they are the following:

1. Coding a plotter, salter, and smoother in Java.
2. Coding and generating a graph for plotter, salter, and smoother in Octave.
3. Coding a plotter, salter, and smoother with jars from JFreeChart and the Apache Statistics Library.

Overall, this report is to document, experiment, and discuss the results of coding a plotter, salter, and smoother in Java and other experiments with the goal of gaining useful skills for a workplace environment. The following subsections will provide information on salting and smoothing for all three experiments and the two libraries that will be used in the third part.

What is Salting and Smoothing?

Salting data helps if the data is skewed and makes it evenly distributed (Shrestha, 2022). From an article on Medium about skewed data in Apache Spark, an application used for big data, goes over three methods on how to deal with skewed data. The last method is Key Salting where “A range of salt values is first selected [and the values are] exploded with a new record generated for each salt value” (Shrestha, 2022). The range of salt values matter in getting the data points such as when “Selecting a large range of salt values allows us to distribute the data much more evenly, allowing for parallel computation” (Shrestha, 2022). However, if the points are “exploded” too largely due to the salt range, the graph can lose its meaning so “a balance is necessary when selecting the salt value” (Shrestha, 2022). In other words, salting adds “garbage” to the data to mess it up.

Smoothing is the process of removing noise from data and “allows important patterns to more clearly stand out” (Dhir, 2022). In the assignment, smoothing helps by removing or reducing the “explosion” made from salting the data. The article, Data Smoothing: Definition, Uses, and Methods by Rajeev Dhir, provides various methods for smoothing data such as using a randomization method, random walk, calculating a moving average, simple moving average (SMA), and exponential moving average (EMA) (2022). Random walk is commonly used and the smoothing “assumes that future data points will equal the last available data point, plus a random variable” (Dhir, 2022). There are pros and cons to smoothing where the pros are helping “identifying real trends by eliminating noise from the data… [and the process is] easily achieved through several techniques including moving averages” (Dhir, 2022). The cons is when data is removed, there is less to analyze and it increases the risk of errors when analyzing and smoothing “may emphasize analysts’ biases and ignore outliers that may be meaningful” (Dhir, 2022).

What is JFreeChart and the Apache Statistics Library?

Both JFreeChart and the Apache Statistics are free libraries which users can use to make their process in coding faster and streamlined. JFreeChart is a Java chart library that was started by David Gilbert and it supports many outputs, has a well-documented API that also supports many chart types, and is open source (Gilbert, 2021). The Apache Statistics Library by Apache Commons Math is actually a package that “provides frameworks and implementations for basic Descriptive statistics, frequency distributions, bivariate regression, and t-, chi-square and ANOVA test statistics” (2022).

Octave and Following a Tutorial

What is Octave?

Octave is a free scientific programming language software and is “largely compatible with Matlab” (Eaton, 2023), Matlab is basically the application, but more powerful than Octave and requires payment. The software provides an interface for solving problems like “numerical

linear algebra…, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations" (Eaton, 2023). Because the software is free, the creators encourage users to contribute additional functions and report any bugs or problems they might encounter to make the software more useful (Eaton, 2023).

Following an Octave Tutorial

The tutorial I will be following to learn Octave is actually a 2009 comprehensive powerpoint called Octave/Matlab Tutorial by Prof. Dr. Kai O. Arras who is head of the Social Robotics Laboratory at the University of Freiburg in Germany (Arras, 2023). The powerpoint is outdated by fourteen years and the current version of Octave is 8.3.0 compared to the 3.2.3 version in 2009 however the powerpoint is good to understand the basics of the software and I can refer to the Octave documentation if I run into any problems. From the powerpoint I will be following the slides on 'Variables and data types', 'Plotting', 'Programming', and lastly 'Functions and Scripts'.

Variables and Data Types

Almost everything is a matrix as Dr. Arras states and lists on slide 10 (2009):

- **Matrices** (real and complex)
- **Strings** (matrices of characters)
- **Structures**
    - Vectors? It's a matrix with one column/row
    - Scalars? It's a matrix of dimension 1x1
    - Integers? It's a double (you never have to worry)
    - Boolean? It's an integer (non-null=true, 0=false)

For an example of a string:

```
str = 'Hello World'
```

`'Hello World'` is saved into a variable `str` and as Dr. Arras stated, the `str` is a matrix of characters. When a user types: `str` in the command line in Octave, `str = Hello World` will be returned. If something else is saved in `str` such as:

```
str = 3
```

And the user types `str` in the command line, `str = 3` will be returned. This shows that "variables have no permanent type" (Arras, 2009, slide 15) while compared to Java, all variables must have a data type. Below is also something not allowed in Java:

```
String str = "Hello World";
str = 3;
```

Because an `int` cannot be converted to a `String` unless you make it into a `String` by doing: `str = String.valueOf(3);` then that is allowed. Octav has the same command to convert an integer to a string simply as: `int2str` and is used as `int2str(3)` as well as `num2str(3)`.

Plotting

The powerpoint covers 'Plotting in 2D' and 'Plotting in 3D', but for this tutorial, I will be following the former. When plotting in 2D, to display and have Octave output a plot, the user will have to type (Arras, 2009, slide 41):

```
plot(x, cox(x))
```

For a figure window 'n', which is a window the plot will appear in, the user will type:

```
figure(n)
```

And lastly, to create a new figure window with an identifier to be incremented by 1 is to just type `figure`. The frequently used commands can be found on slide 43 and includes clearing the figure (`clf`), giving the figure a title (`title('Exp1')`), labeling the x and y-axises (`xlabel('time')` and `ylabel('prob')`), and holding axes (`hold on`). `hold on` means to not "replace plot with new plot, superimpose plots" (Arras, 2009).

An example has the user type into the console (Arras, 2009, slide 47):

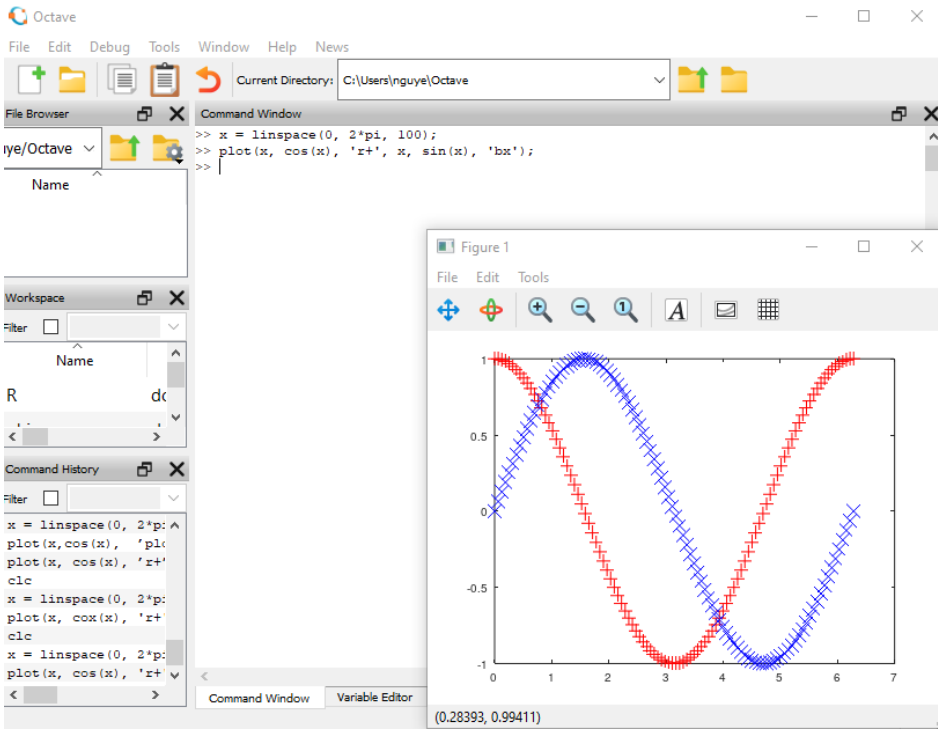
```
x = linspace(0, 2*pi, 100);
```

```
plot(x, cos(x), 'r+', x, sin(x), 'bx');
```

The `r+` means red crosshairs and `bx` means blue crosses which can be seen in Figure 1 below.

Figure 1

Octave Console Window With Plot Output

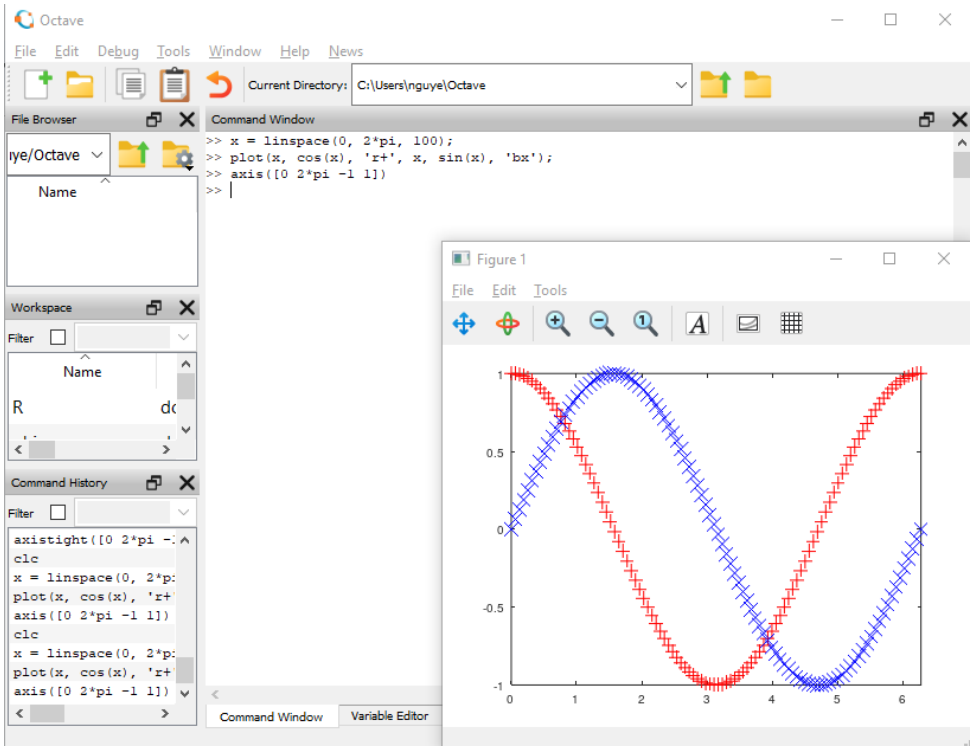


Next is to adjust the axes by typing below and Dr. Arras suggests also trying `axis tight`. The results for both are shown in Figure 2 and 3 respectively. Figure 2 compared to Figure 1 ends the x-axis numbers at 6 instead of 7 so each of the lines touch the end of the graph instead of there being a gap.

```
axis([0 2*pi -1 1])
```

Figure 2

Octave Console with Plot Having Adjusted Axis

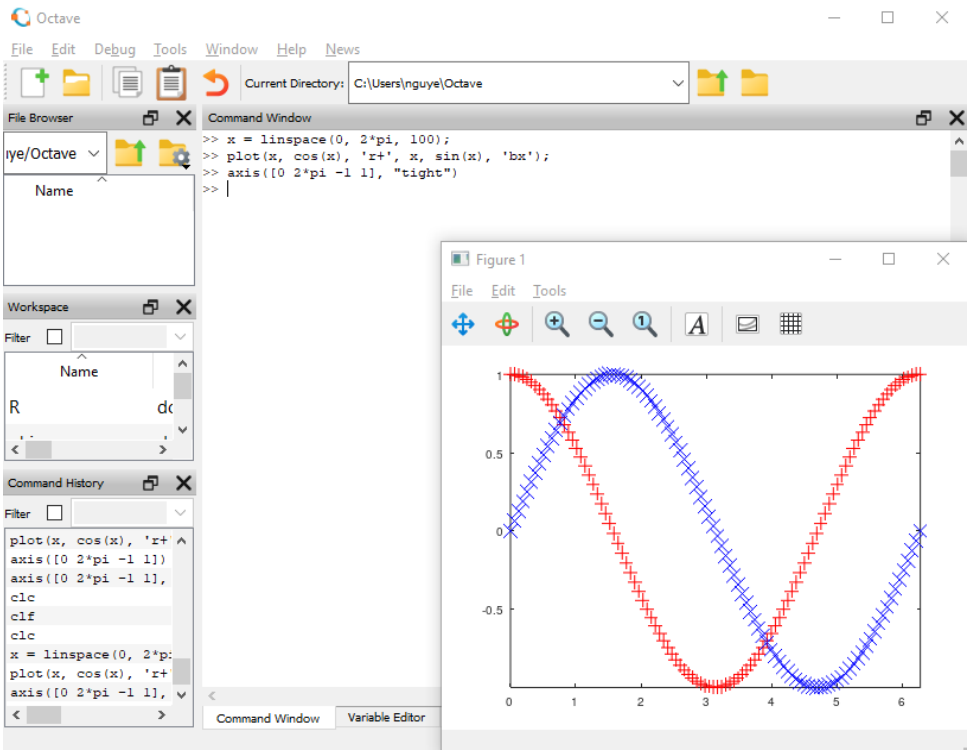


To see what will happen to the plot with axis tight, I was unsure of how to type it. I looked at the documentation for the correct syntax. I found it in '15.2.1.1 Axis Configuration' and an example in 'Example 1: set X/Y limits and force a square aspect ratio' that provided a similar format to the axis command as above: `axis ([1, 2, 3, 4], "square")`. Using that, I applied `"tight"` to the next line at the end, however just typing `axis tight` will work the same way. For Figure 3, the "tight" parameter is made to "fix axes to the limits of the data" (The Octave Project Developers, 2023b).

```
axis([0 2*pi -1 1], "tight")
```

Figure 3

Octave Console with Plot Having Adjusted Axis with "tight" Option

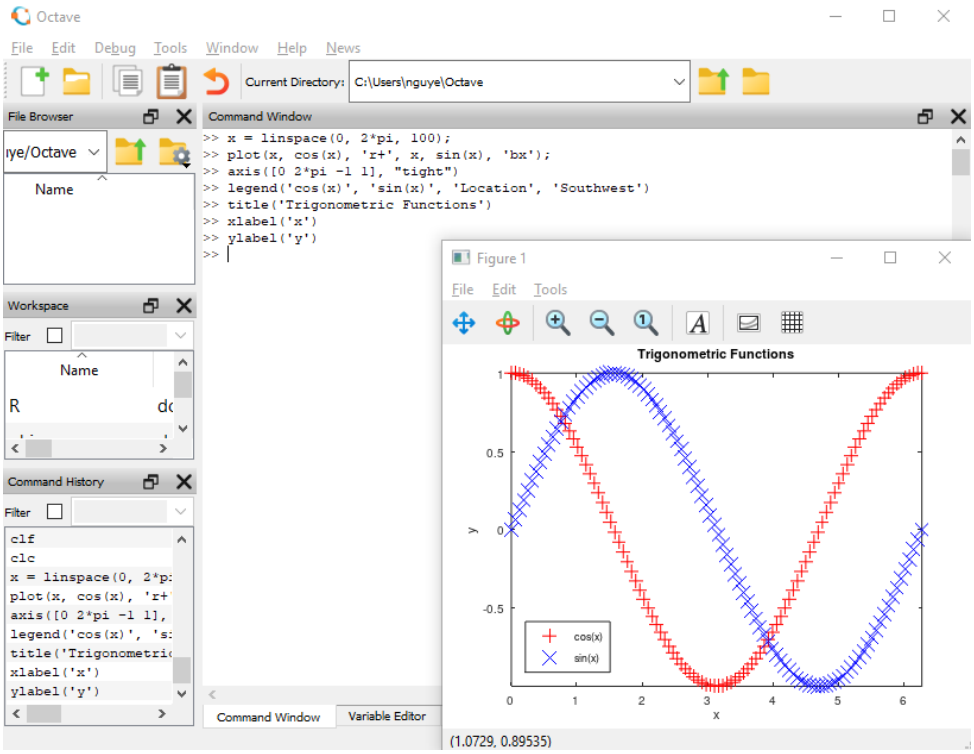


Both Figure 2 and Figure 3 produced the same output and compared to Figure 1, the axes fit to the end of the limits of the data. Using commands from the frequent commands slide previously mentioned, the last step in the example is to add a legend, labels, and a title (Arras, 2009, slide 47) as shown in Figure 4.

```
legend('cos(x)', 'sin(x)', 'Location', 'Southwest')
title('Trigonometric Functions')
xlabel('x')
ylabel('y')
```

Figure 4

Octave Console with Plot Having Legend, Title, and Labels

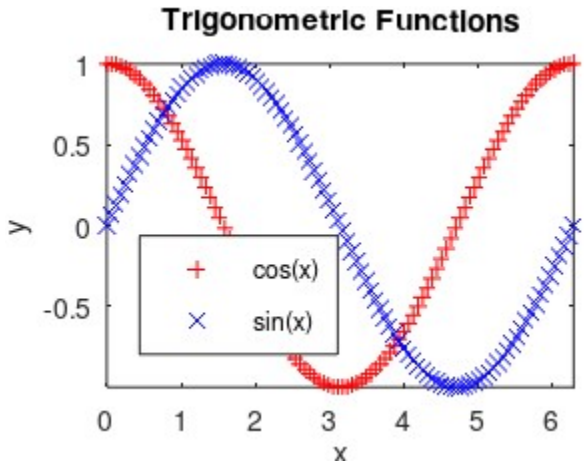


The last two slides of 'Plotting in 2D' (slides 51-52) goes over getting a hardcopy or in other words using print as a command or function to get an .eps (encapsulated PostScript), .jpg, or .png of the plot. `-dpng` is one of the formats that a plot can be saved in and `-r100` is the resolution. The result can be seen in Figure 5 with the resolution being 100 DPI (dots per inch).

```
print -dpng -r100 myPicKay.png
```

Figure 5

Saved Plot from Print Command



Programming

Things to note as a difference from programming in Octave compared to other programming languages is that (2009, slide 55):

- Indices start with 1 !!!
- Octave/Matlab is case-sensitive

For if-else statements, the syntax in the powerpoint is outdated thus the updated syntax is shown on the left (The Octave Project Developers, 2023d) and to compare, Java is shown on the right (Oracle, 2022b). The clear difference is that Octave does not require semicolons and curly braces like they are required in Java.

```
Octave:
if (condition)
      then-body
elseif (condition)
      elseif-body
else
      else-body
endif
```

```
Java:
if (condition) {
      if-body;
} else if(condition) {
      if-body;
}
```

The for statement is also different with the general syntax for Octave showing on the left (The Octave Project Developers, 2023c) and Java on the right (Oracle, 2022a).

```
Octave:
for var = expression
      body
endfor
```

```
Java:
for (initialization; termination;
increment) {
      statement(s)
}
```

The last differences from the powerpoint are on increment operators that are only exclusive to Octave compared to Matlab, relational operators, and boolean expressions. There are slides on comparison operators, however it is self explanatory and the results for comparisons are either 0 or 1 and can be used to compare matrix-to-matrix and matrix-to-scalar (Arras, 2009, slides 61-62).

The increment operators that are exclusive to Octave are (Arras, 2009, slide 60):

- i++ Increment scalar i by 1
- i-- Decrement scalar i by 1
- A++ Increment all elements of matrix A by 1
- v-- Decrement all elements of vector v by 1

In Java, the first two increment operators are also the same, but the last two operators are not able to be done.

For the relational operators, the two relational operators that are different from Java are `x ~= y` and `x <> y` where both are “true if x not equal to y” (Arras, 2009, slide 63). For boolean expression they are for element-wise logic, while to evaluate an expression only uses the `&&` and `||` for ‘and’ and ‘or’ respectively. Below are for boolean expressions with the ‘Octave only’ for ‘Element-wise logical not’ meaning it is only of Octave compared to the expression in Matlab (Arras, 2009, slide 64):

- `B1 & B2` Element-wise logical **and**
- `B1 | B2` Element-wise logical **or**
- `~B` Element-wise logical **not**
- `!B` Element-wise logical not (Octave only)

Functions and Scripts

The last topic I will follow in the powerpoint is ‘Functions and Scripts’, but will mainly focus on the ‘Functions’ part. Scripts are text files with an .m extension that has executable code (Arras, 2009, slide 72). For naming conventions in Octave, it is recommended to use underscore-separated or lowercase notation for functions and to use upper camel case for scripts. The commands for both applications, Octave and Matlab, are lowercase.

The syntax for function is on the left and the syntax when wanting to pass parameters in is on the right (The Octave Project Developers, 2023a):

```
Simplest form:                  Function with parameter:
function name                   function name (arg-list)
      body                            body
endfunction                     endfunction
```

To comment in Octave, the Dr. Arras put a percent sign, `%` on slide 72 (2009), however it can also be a pound sign, `#`. Single line, block, and the help system comments can use both symbols. Examples are below for block and help system comments as single line comments are self explanatory (The Octave Project, 2023e).

```
Block comments:          #{ . . . #} or    %{ . . . %}

Help system comments:    function xdot = f(x,t)
                         # usage: f (x,t)
                         #
                         # This function defines the right-hand
                         # side functions for a set of nonlinear
                         # differential equations.
                               r = 0.25;
                         endfunction
```

Using what I have learned from the powerpoint will be explored more in 'Part 2: Using Octave' for the assignment.

Part 1: Coding in Java

The first task before coding a plotter is to choose a function that is interesting, but not complicated like trigonometric functions. Another caveat is to not choose the linear function because that is a boring graph to look at. Thus, the function I decided to go with is the standard form of the parabola which can be seen in Equation (1).

$$y = ax^2 + bx + c \tag{1}$$

In Equation (1), a, b, and c are coefficients and a cannot equal 0 because the function will not be a parabola. If $a = 0$ and the other coefficients are not equal to 0, it would become a linear equation like in Equation (2).

$$y = bx + c \quad (2)$$

If $a = 0$ and $b = 0$, it would become like Equation (3), a constant or a horizontal line.

$$y = c \quad (3)$$

Therefore, as long as a does not equal 0, the function will be a parabola.

Furthermore, before moving onto the sections of Part 1, I would like to preface that the results will be further discussed in the Discussion and Conclusion section. This also pertains to the other parts for the report.

Plotter

After having chosen a function and coded the method to make the points for the graph, I wrote the points data into a csv file, a comma-separated values file. For the code, I allowed each parameter in the function and graph to be changeable where the user can input the minimum and maximum of the x-axis (`x_min` and `x_max`), the number of points they want on the graph (`z`), and the three coefficients (`a`, `b`, and `c`).

```
plot.plotter(x_min, x_max, z, a, b, c);
plot.plotter(-50, 50, 100, 1, 3, -5);
```

For my graph in Figure 6, I decided to set the minimum of the x-axis at -50, the maximum at 50, have 100 points, the a coefficient at 1, the b coefficient at 3, and the c coefficient at -5. In an equation it will look like in Equation (4). If the a coefficient is negative, the parabola will be facing downwards instead of upwards.

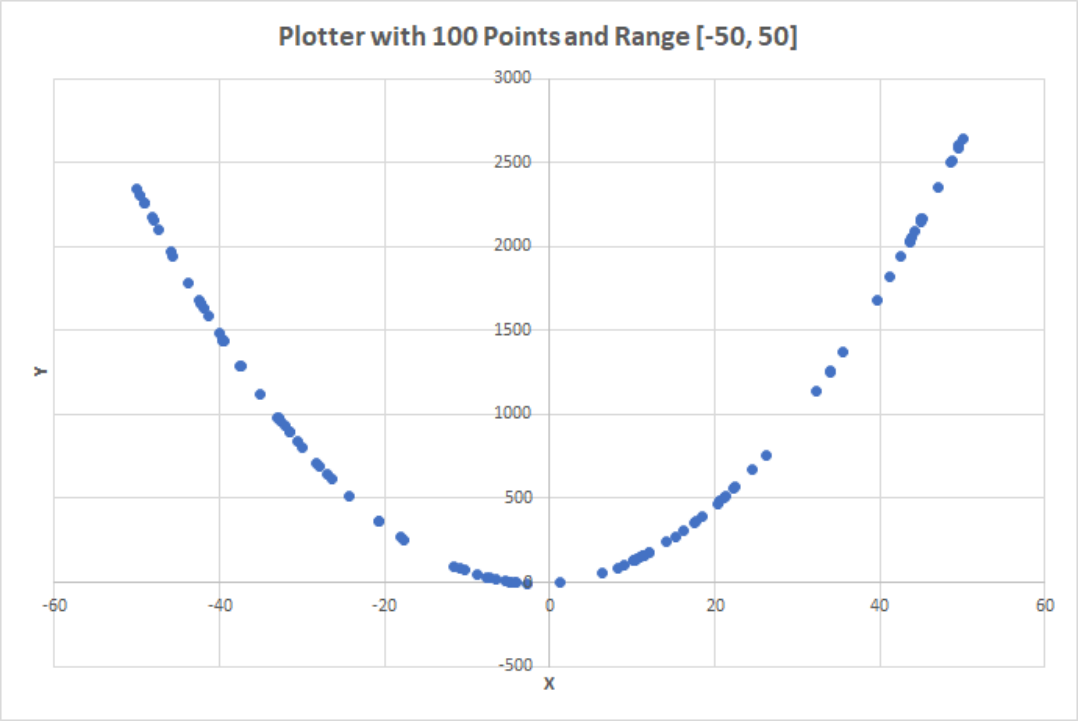
$$y = 1x^2 + 3x + -5 \quad (4)$$

For the ranges, I decided to have the maximum and minimum values be inclusive mainly to my thought process of how I would expect a graph to appear if I input ranges on another application. Of course, this does depend on other people's thought processes and there was no indication of what interval notations were required so I assumed that was left to the programmer.

My attempt at making the ranges be inclusive was inside the for loop, which was what I used to randomly generate the x-coordinates to get the y-coordinates, I ran into a problem. It was a very small problem as there was no noticeably large difference between having the ranges be inclusive or exclusive, but it was a problem to me in my understanding of the ranges. Having the ranges be inclusive and inside the for loop, the output resulted in points that were over the maximum and minimum range by some decimals but never reaching the next integer. Because of this, and to make it easier on myself as I thought on this for too long, I decided to find the y-coordinates of the maximum and minimum values outside of the loop and decrease z by 2 and get y-coordinates for the randomly generated x-coordinates in between the ranges in the loop.

Figure 6

Graphed Results of Plotter with 100 Points and Range [-50, 50]



As expected, Figure 6 is plotted as a parabola with 100 points and with 50 and -50 at the ends. Below are other graphs to show different configurations of the program with the same coefficients for the function.

For Figure 7, I kept the maximum and minimum the same values as the range is 100 and changed the number of points to 20. Not much has changed compared to Figure 6 because the range is the same, just the number of points were decreased.

```
plot.plotter(-50, 50, 20, 1, 3, -5);
```

Figure 7

Graphed Results of Plotter with 20 Points and Range [-50, 50]

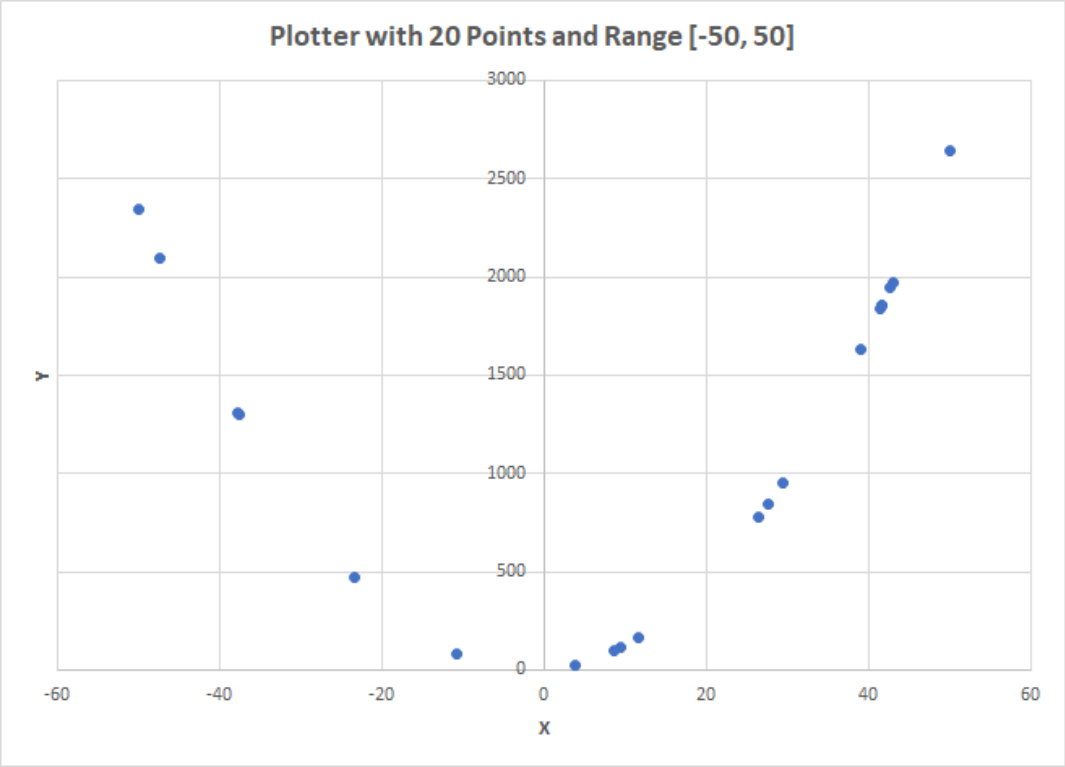
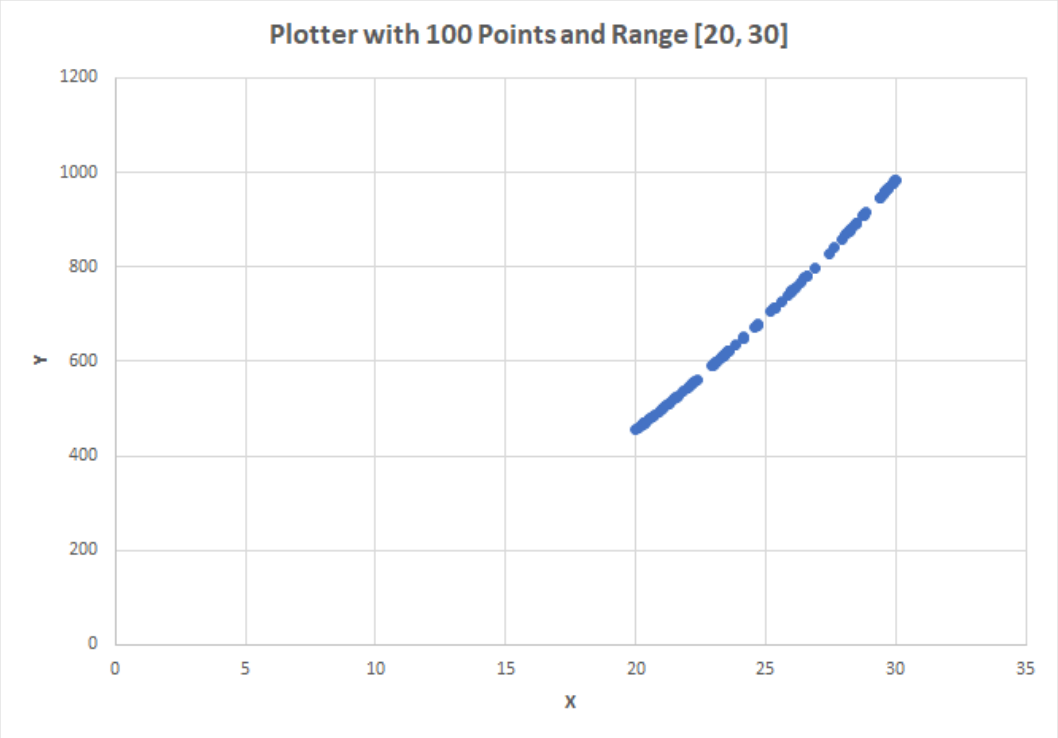


Figure 8 is of low range and high number of points. I changed the max and min to 30 and 20 to have a range of 10 and changed the number of points to 100.

```
plot.plotter(20, 30, 100, 1, 3, -5);
```

Figure 8

Graphed Results of Plotter with 100 Points and Range [20, 30]



The result of Figure 8 is interesting because of the range given the function it was not sufficient enough to become a parabola. It turned into more of a ‘linear’ function.

Figure 9 is of low range and low number of points. I kept the max and min the same as for Figure 8 and changed the number of points to 20.

```
plot.plotter(20, 30, 20, 1, 3, -5);
```

Figure 9

Graphed Results of Plotter with 20 Points and Range [20, 30]

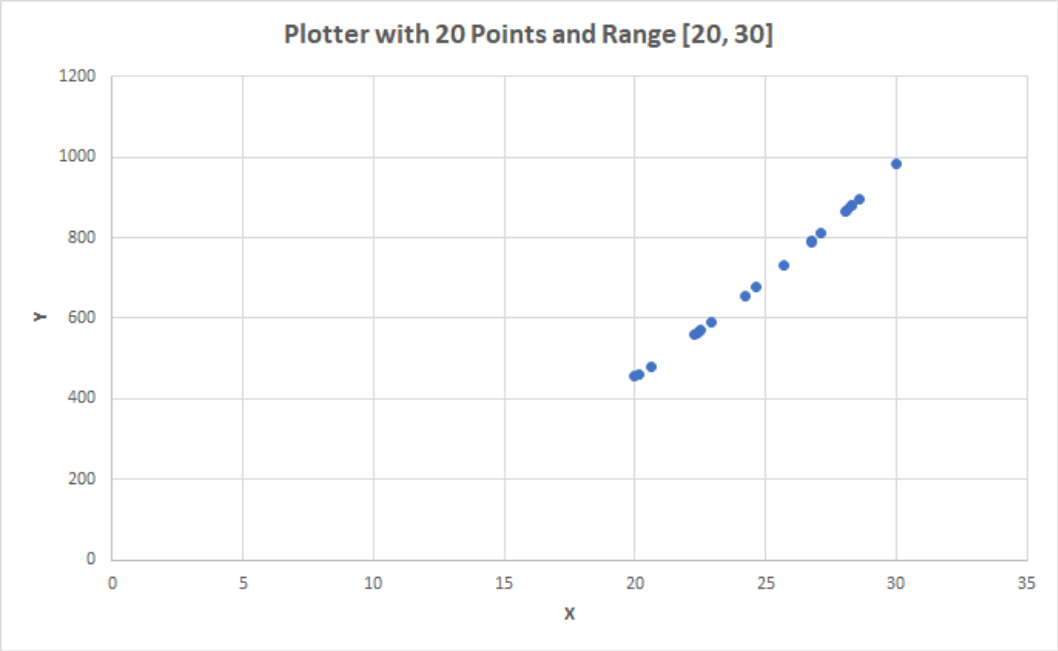


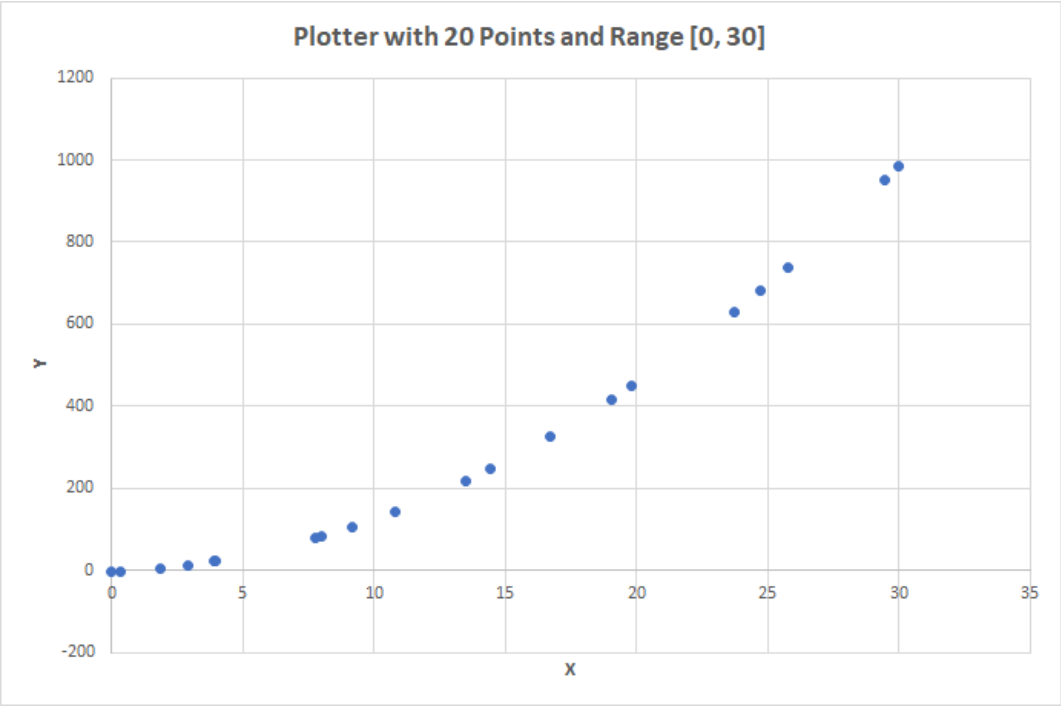
Figure 9 is the same as Figure 8, but with fewer points and the same outcome with it not resulting in a parabola.

For Figure 10 and Figure 11, I kept the same parameters, but changed the minimum variable to 0 and -30 respectively. Figure 10 results in the right half of a parabola while compared to it, Figure 11 results in a parabola.

```
plot.plotter(0, 30, 20, 1, 3, -5);
```

Figure 10

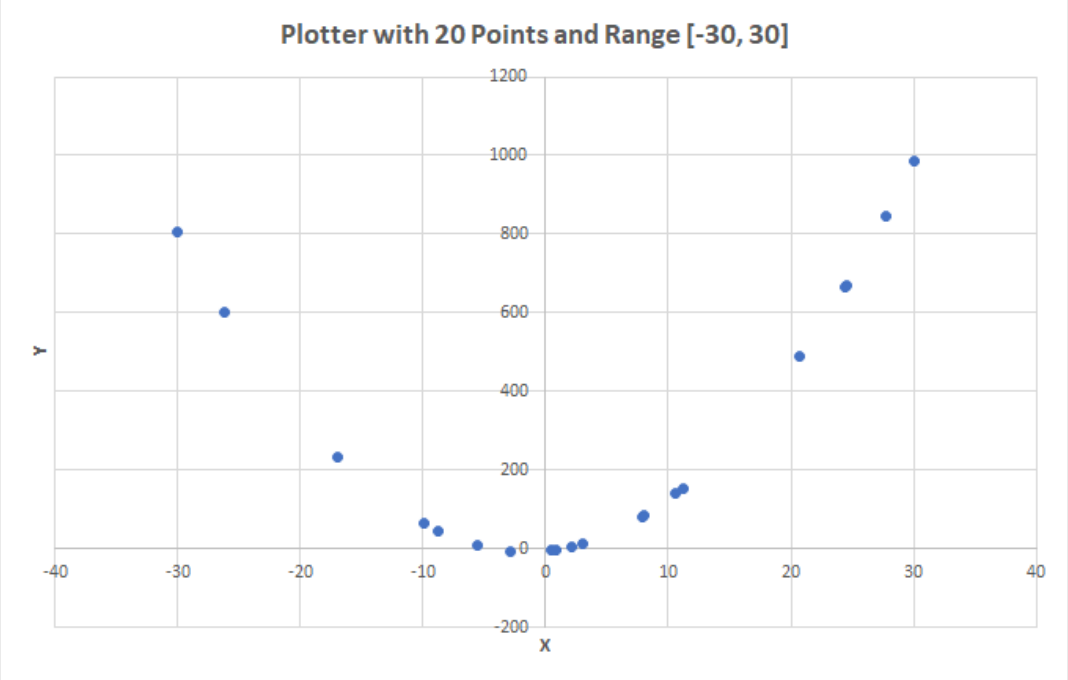
Graphed Results of Plotter with 20 Points and Range [0, 30]




```
plot.plotter(-30, 30, 20, 1, 3, -5);
```

Figure 11

Graphed Results of Plotter with 20 Points and Range [-30, 30]



Since the instructions of the assignment is to allow the user to enter in their desired maximum and minimum values for the x-axis, I did not put any if-statements to check if the range will constitute a parabola. It is the reason why Figure 8, Figure 9, and Figure 10 do not result in full parabolas and only half, much so that it might look linear depending on the range and number of points.

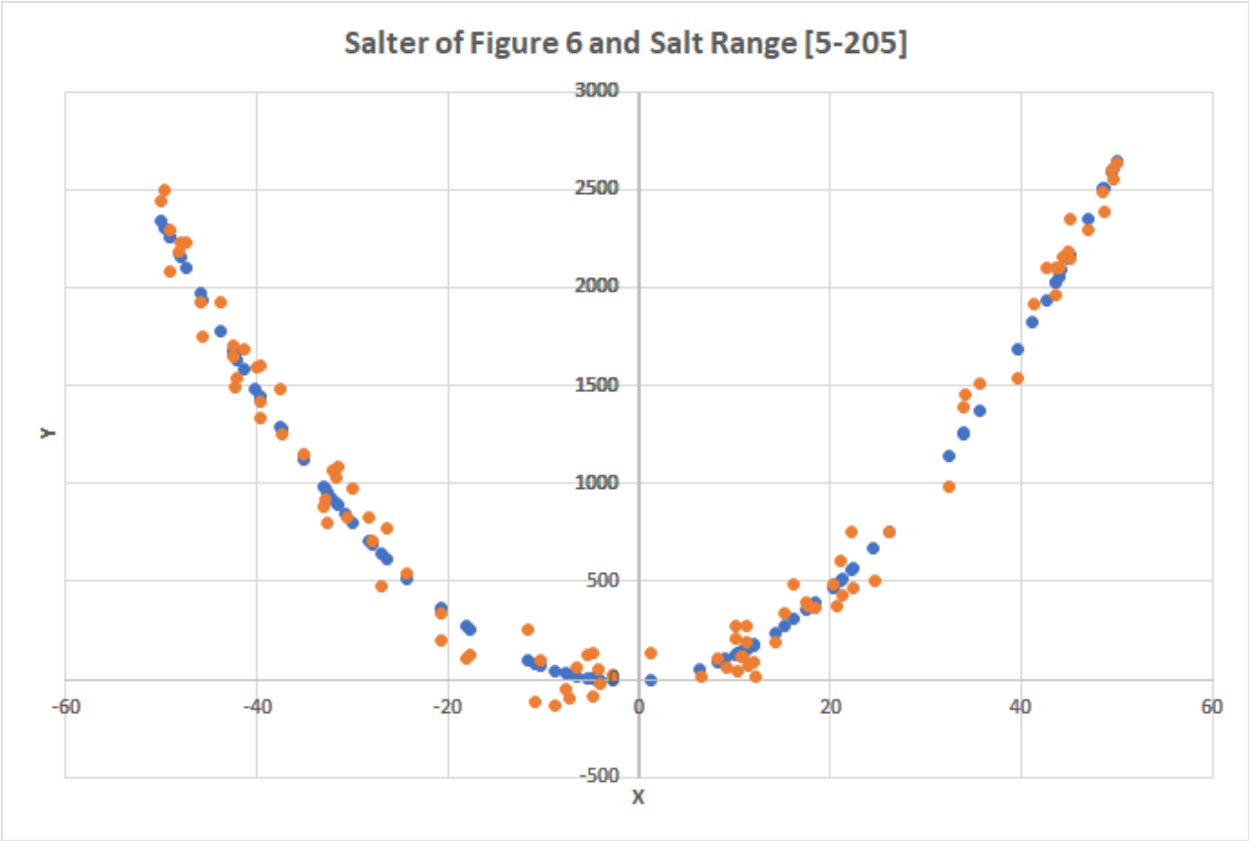
Salter

Next is salting the data from the plotter and I will be using the results from the previous sections for the salting results. Salting also has a range that a user can change, so my salter method takes in two parameters: the salt_min and the salt_max. Compared to the range parameters in the plotter, I made sure to check if both numbers are positive because salting is adding or subtracting random numbers generated from the range instead of allowing them to enter negative values.

Figure 12 is the salting result of Figure 6, having the same number of points and with the salting range of 5 to 205 for the salt_min and salt_max. Figure 6 is shown in the blue dots while the result of Figure 12 is in orange.

Figure 12

Salted Graph of Figure 6 with Salt Range [5-205]



It can be seen in Figure 12 that all the points have been salted with a large range of 200 and with a high number of points at 100. A lower range can be seen in Figure 13, still with 100 points from Figure 6. I kept the min at 5 and changed the max to 55 for a range of 50 compared to the 200 range for Figure 12.

Figure 13

Salted Graph of Figure 6 with Salt Range [5-55]

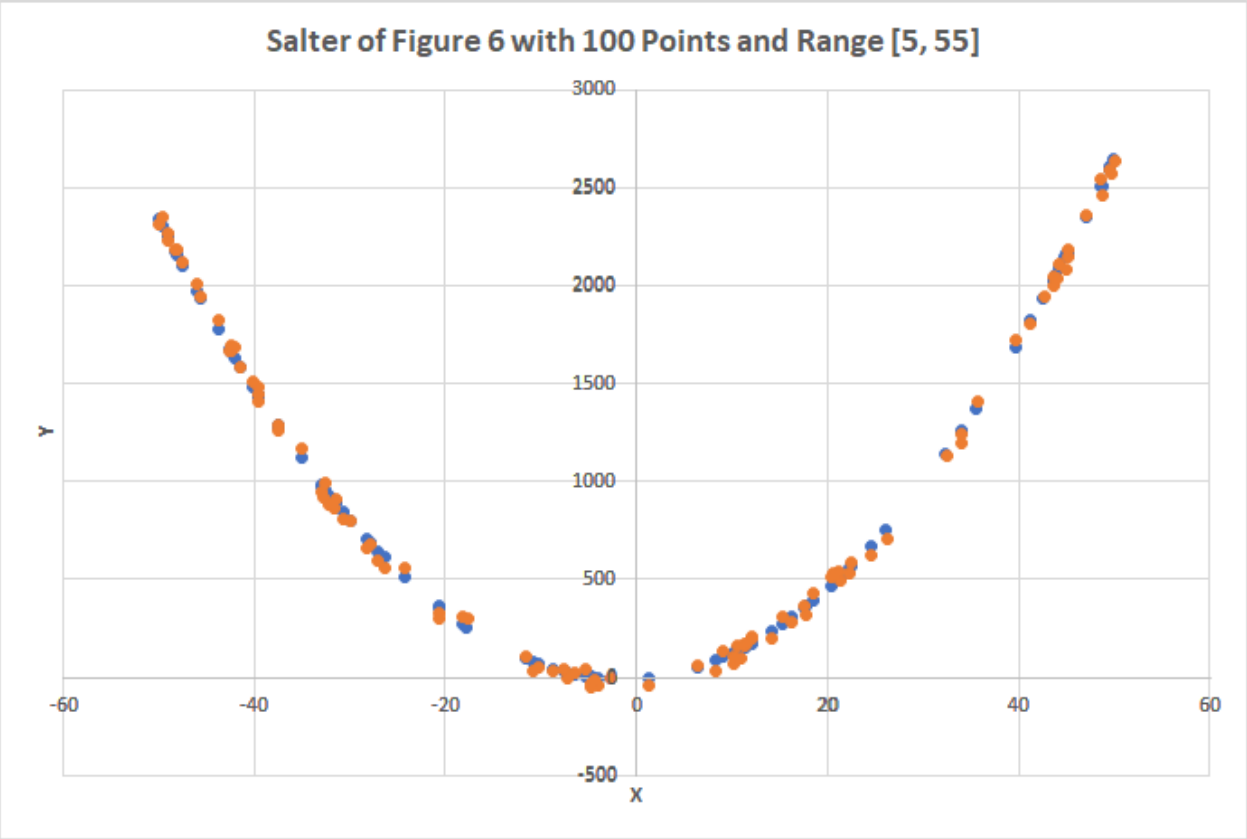
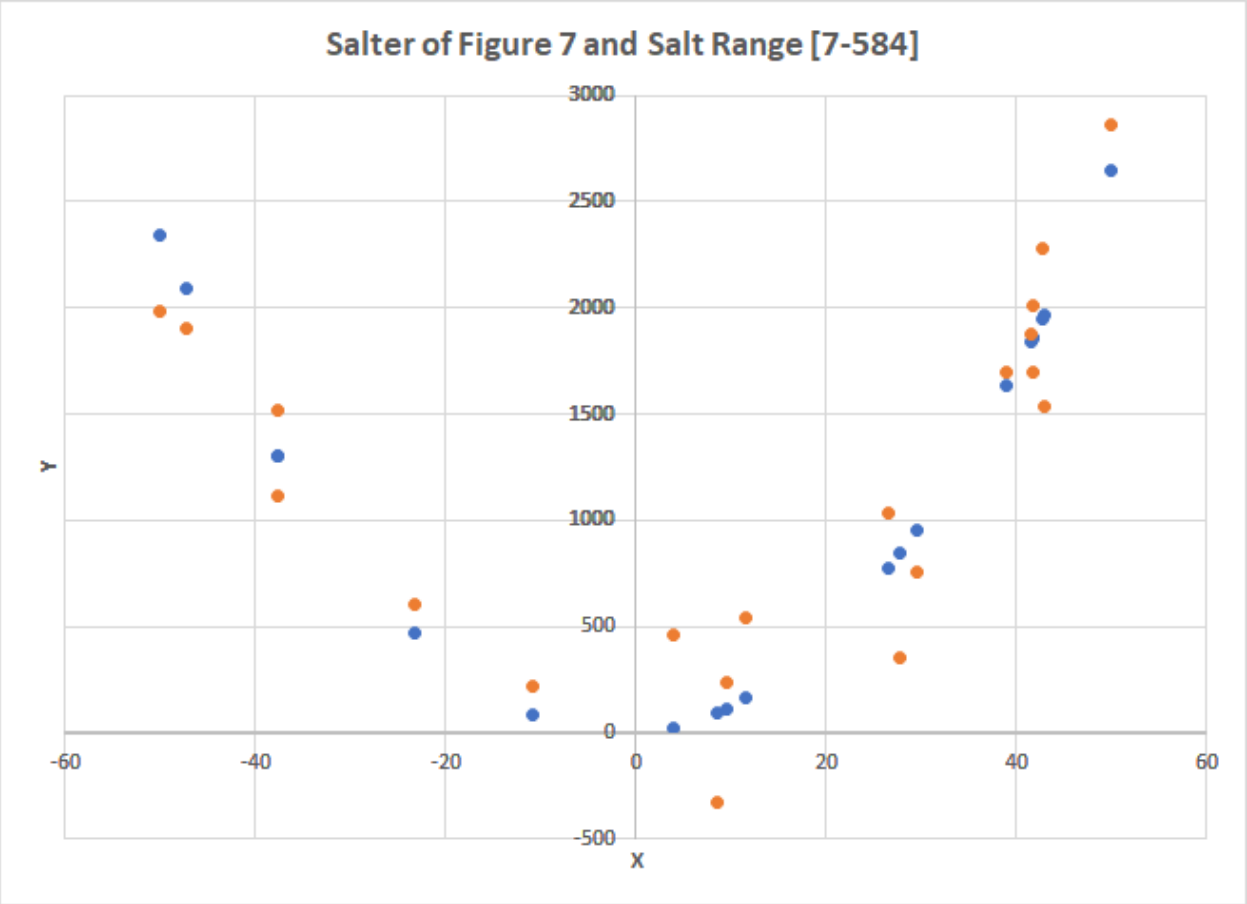


Figure 13, like in Figure 12 has blue for Figure 6 plotted data and orange is for the salted data. It can be observed that salting with a low range shows very little difference from the plotted data from Figure 6 and is not at all volatile compared to Figure 12.

Figure 14 is salting the results from Figure 7 which has the same range of [-50, 50], but much fewer number of points which is at 20. When salting, I chose a large range from 7 to 584.

Figure 14

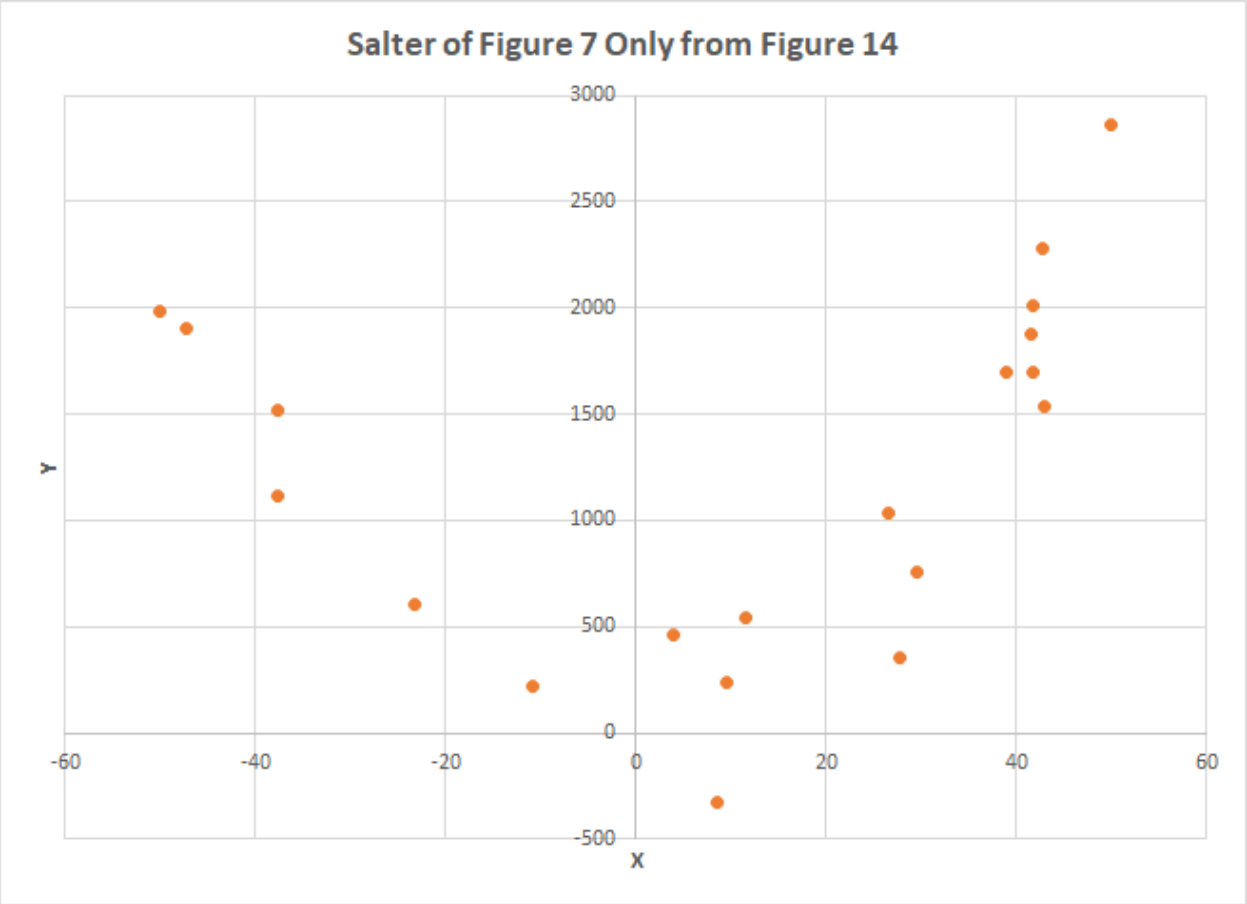
Salted Graph of Figure 7 with Salt Range [7-584]



With a very large range of 577 for salting the results of Figure 7, it still maintains the shape of the parabola. However, after removing the blue dots, it can be seen in Figure 15 the shape is still there but on the verge of the points losing its meaning.

Figure 15

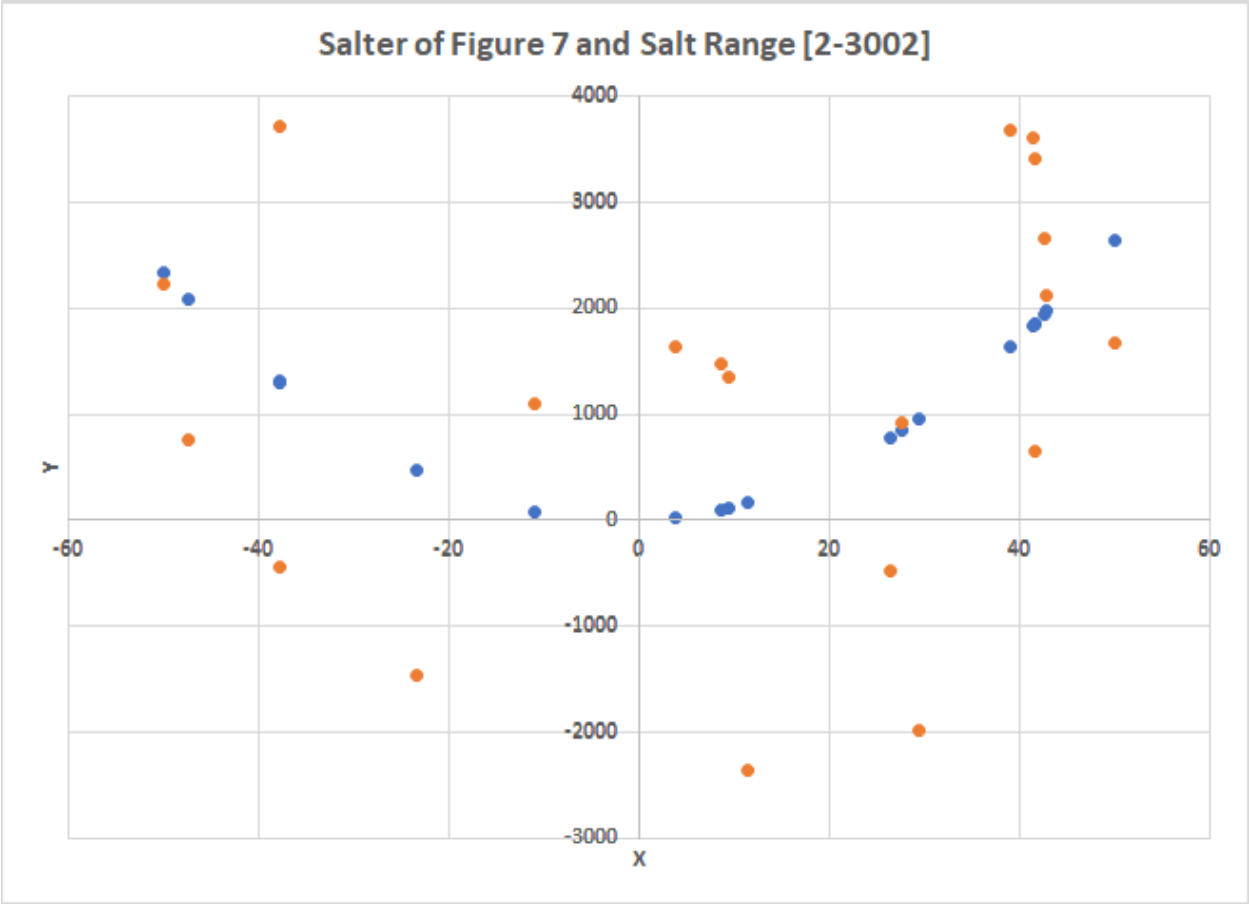
Salted Graph of Figure 7 Only from Figure 14



For Figure 7 to lose its meaning after salting, the range would need to be higher than what I have currently set. If I change the range to be higher than [7-584] to, for example [2-3002] for a range of 3000, the graph would result in Figure 16.

Figure 16

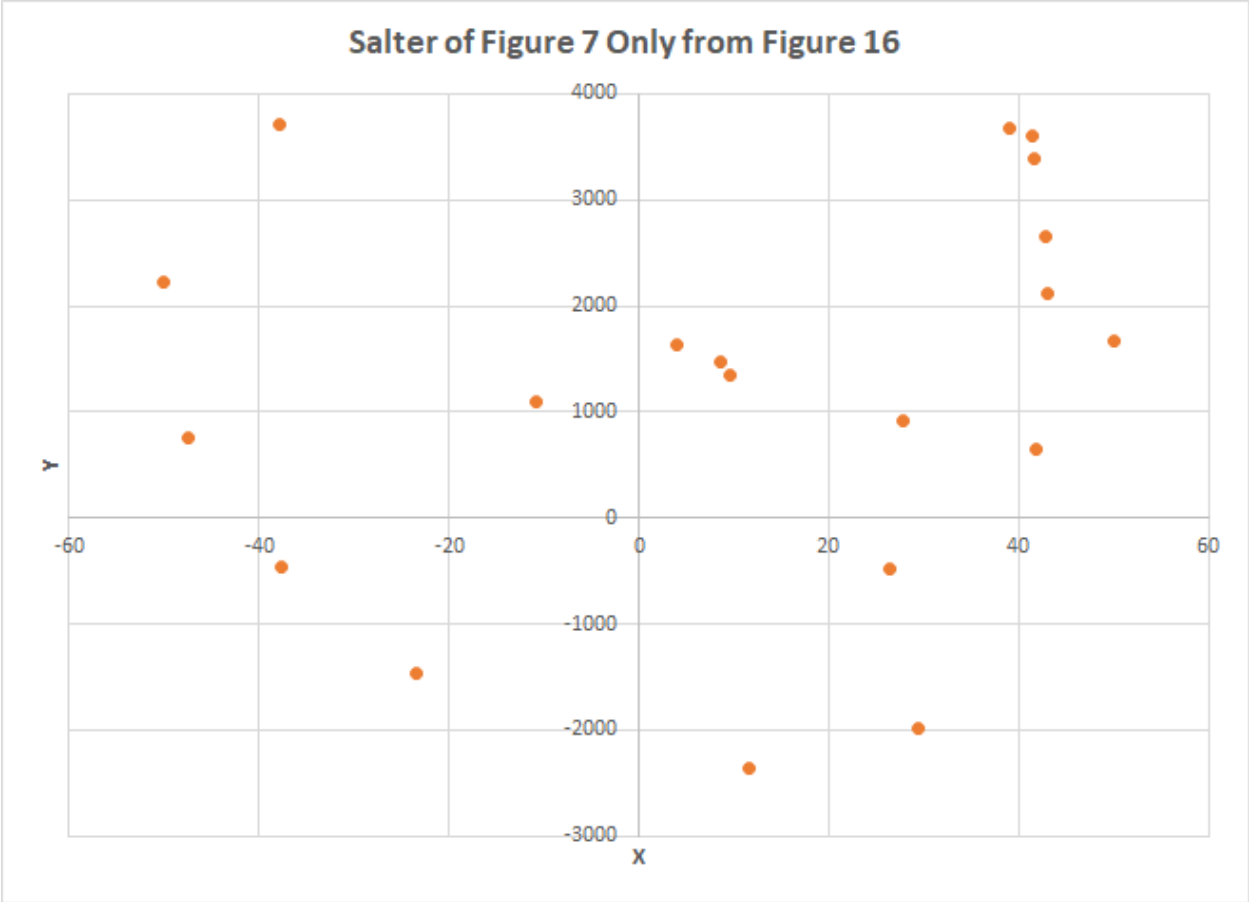
Salted Graph of Figure 7 with Salt Range [2-3002]



Here it is clearly seen that the data points have lost their shape as a parabola and the graph is indecipherable in its meaning compared to the blue data points of Figure 7. Figure 17 are the salted results without the blue data points to show clearly how indecipherable it is.

Figure 17

Salted Graph of Figure 7 Only from Figure 16



The last experiment with salting will be with the same graph of Figure 7, however with a low salting range of 95, with the min being 5 and the max being 100 and it is significantly smaller than the 3000 range for Figure 16.

Figure 18

Salted Graph of Figure 7 with Salt Range [5-100]

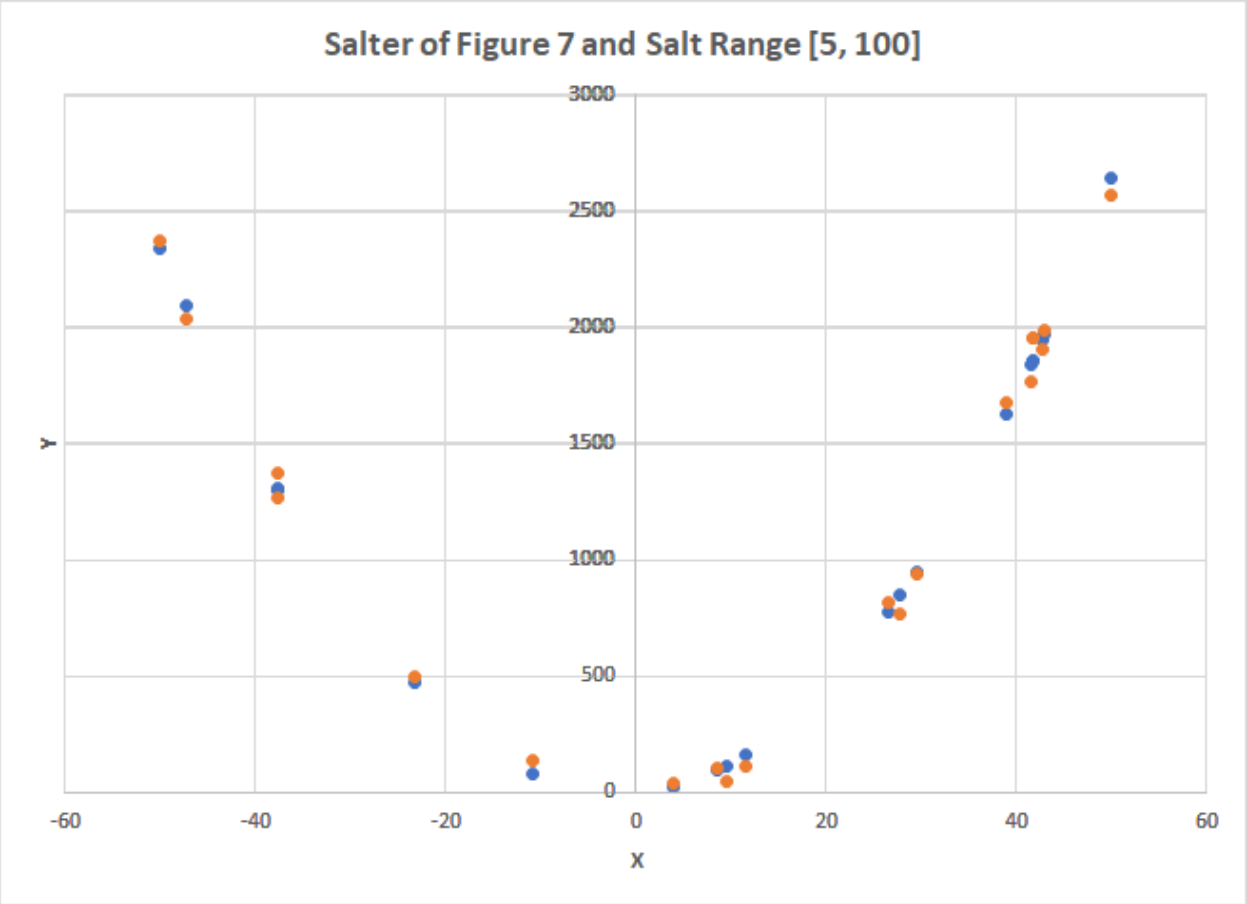


Figure 18, compared to Figure 16 and Figure 17, shows little difference in volatility and the result is the same as Figure 13, where I salted Figure 6 with a low range and it has a larger number of points.

Smoother

Lastly for this Part 1 of the assignment, is the smoother. In my code, I allowed the parameter of windowValue, but the parameter has “checks” for a lack of a better term. Meaning the parameter cannot be negative or zero and it cannot be equal to the number of points in the data and it cannot be greater than half of the number of points. More will be explained in the Discussion section.

Figure 19

Smoothed Graph of Figure 6 with Window Value of 3

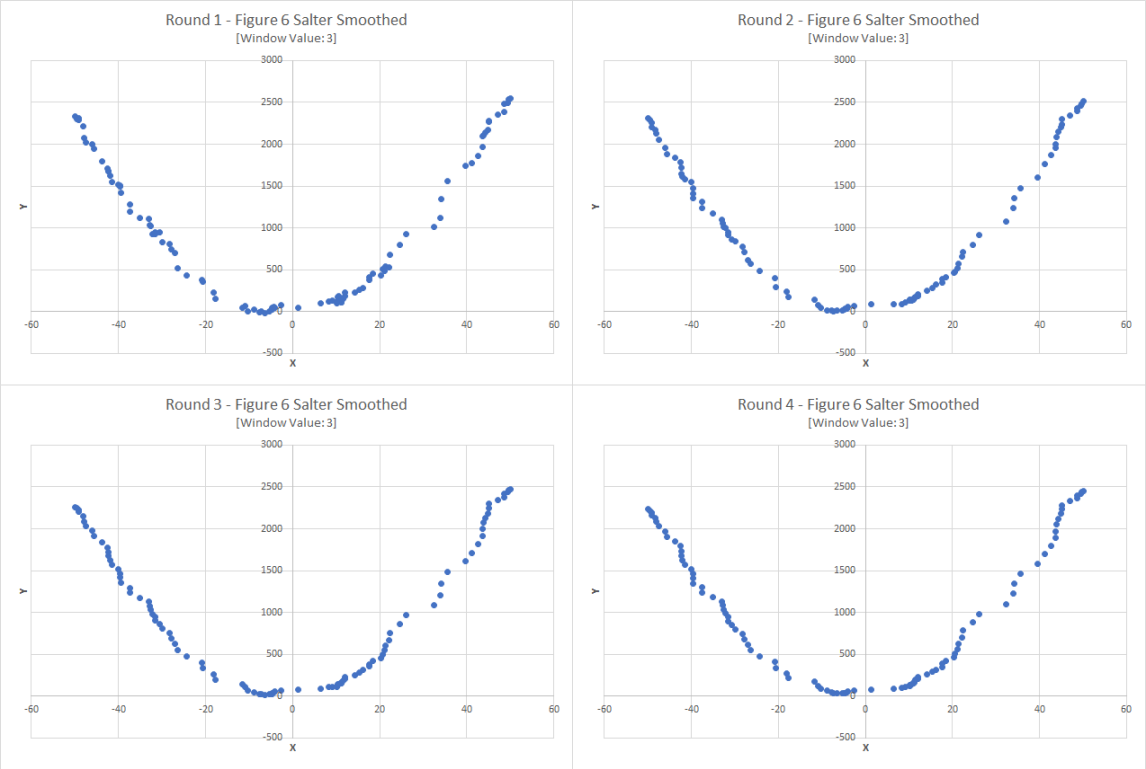
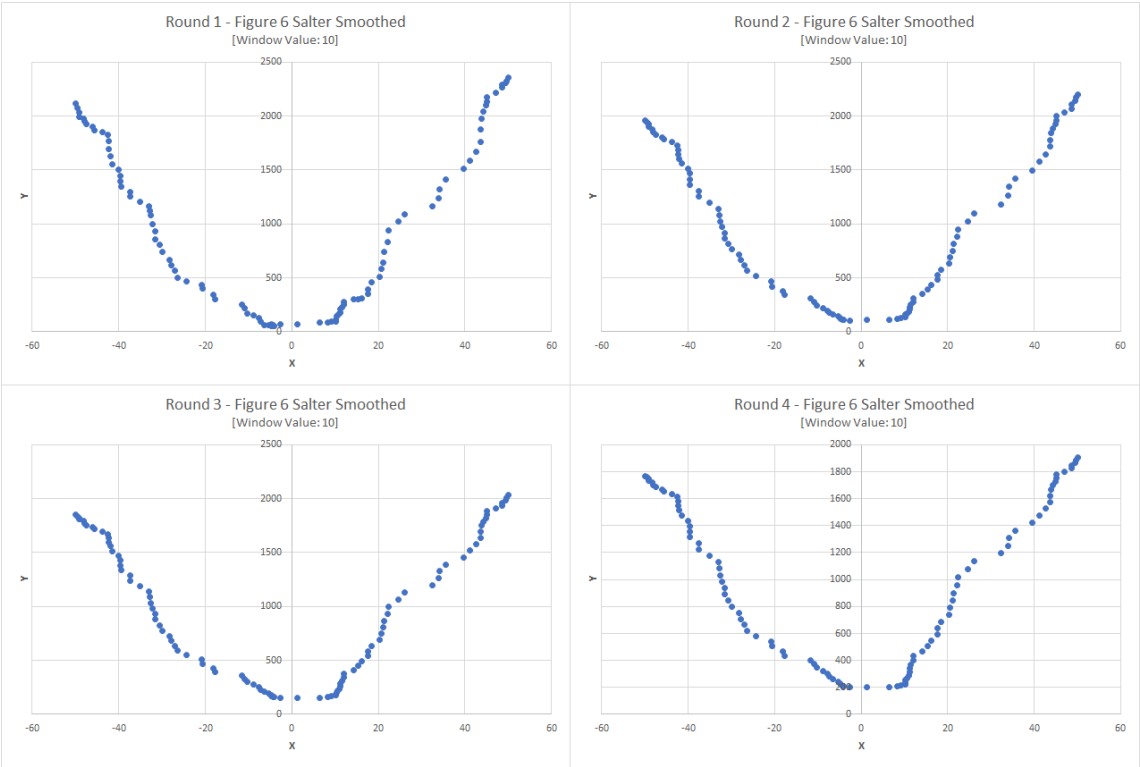


Figure 19 is the smoothing of Figure 7 in four rounds with a window value of 3. In each round it seems to have smoothed it, but by round 4 the little curves are more prominent.

Figure 20

Smoothed Graph of Figure 6 with Window Value of 10



For Figure 20, still smoothing Figure 6 after salting, the smoothing after four rounds has lost its shape and made it wonky.

Below are Figure 21 with window value 20 and Figure 22 with window value 50 to show how a large window has made the parabola lose its shape more compared to Figure 20 with the window value of 10. Small curves are made with Figure 21 while with Figure 22, it can be seen that there are two smaller parabolas that were made, losing the whole shape of the parabola.

Figure 21

Smoothed Graph of Figure 6 with Window Value of 20

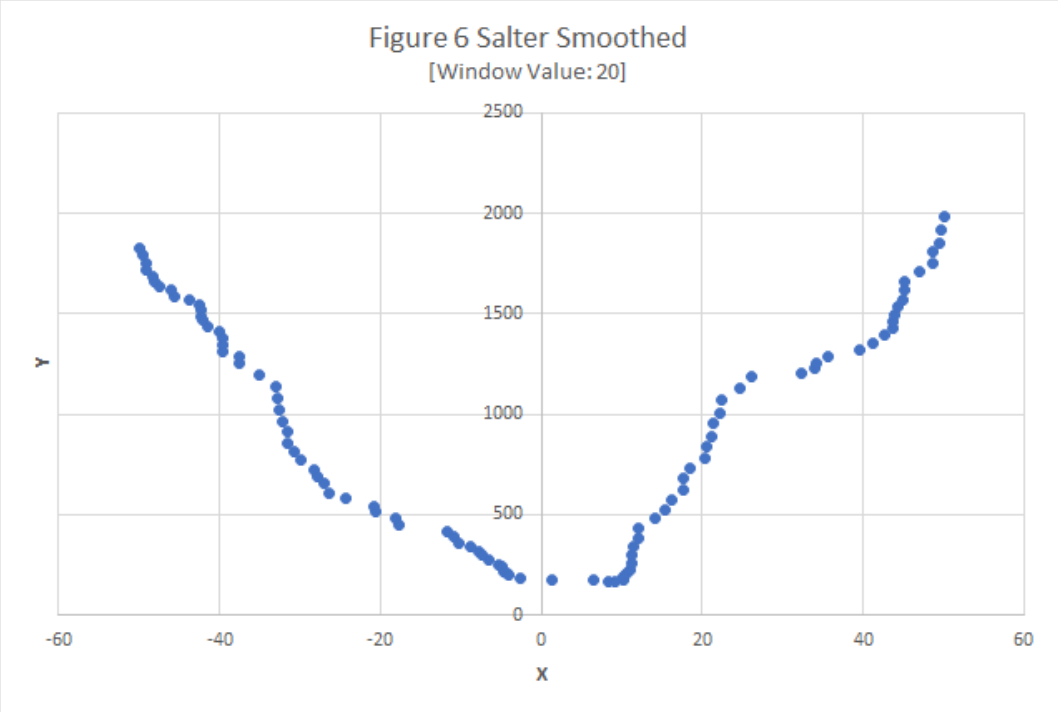
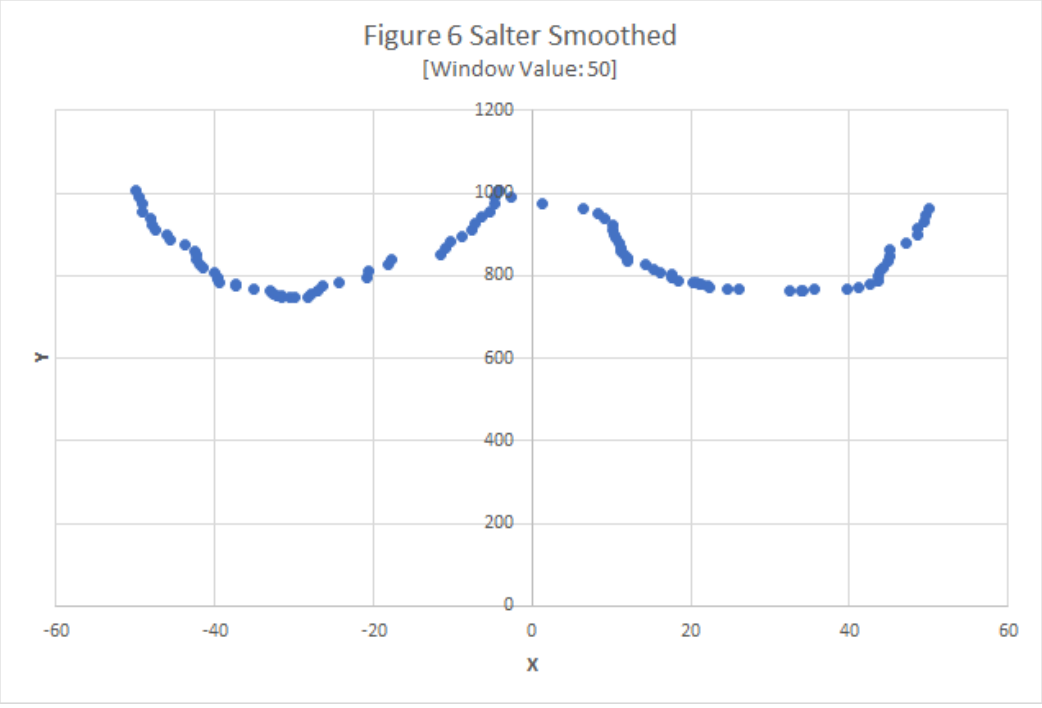


Figure 22

Smoothed Graph of Figure 6 with Window Value of 50



Part 2: Using Octave

Plotter

Below is the code I used to plot Figure 23 with variations in Figure 24 and Figure 25 with different numbers of points and ranges. Figure 25 tests if Octave can make 10,000 points just for fun. In the command window I typed:

```
x=linspace(-50, 50, 100);
y=x.^2+3*x-5;
points=[x(:), y(:)];
```

Then I called my plotter function as so: `plotPlotter(x,y);` and also below is the code for my function as it is a simple plot with the fixings of axis labels, title, and grids on.

```
function plotPlotter(x,y)
      plot(x,y,'.');
      xlabel("X");
      ylabel("Y");
      grid on;
      title("Plotter with 100 Points and Range [-50, 50]");
endfunction
```

Figure 23

Octave Plot of 100 Points and Range [-50, 50]

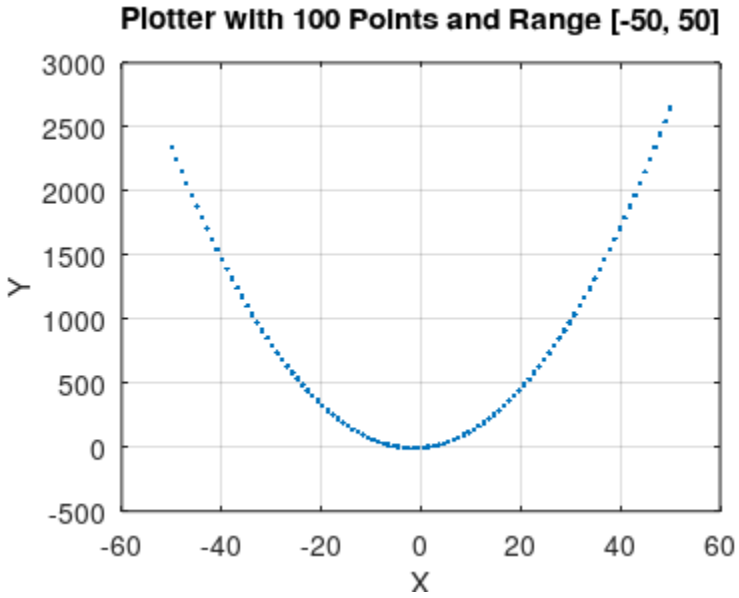


Figure 24

Octave Plot of 20 Points and Range [-50, -50]

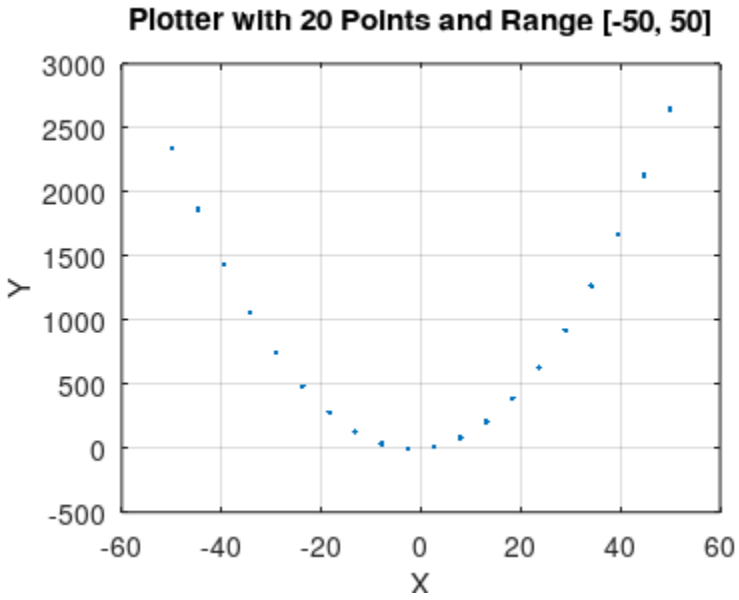
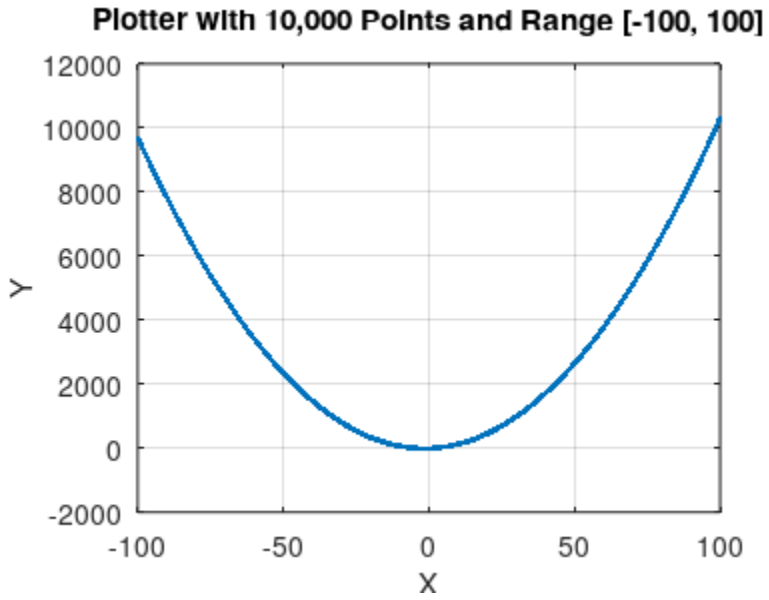


Figure 25

Octave Plot of 10000 Points and Range [-100, 100]



Salter

For salter, I made a saltPoints function that is seen below. Results are shown in Figure 26 to compare with the ones from Figure 12.

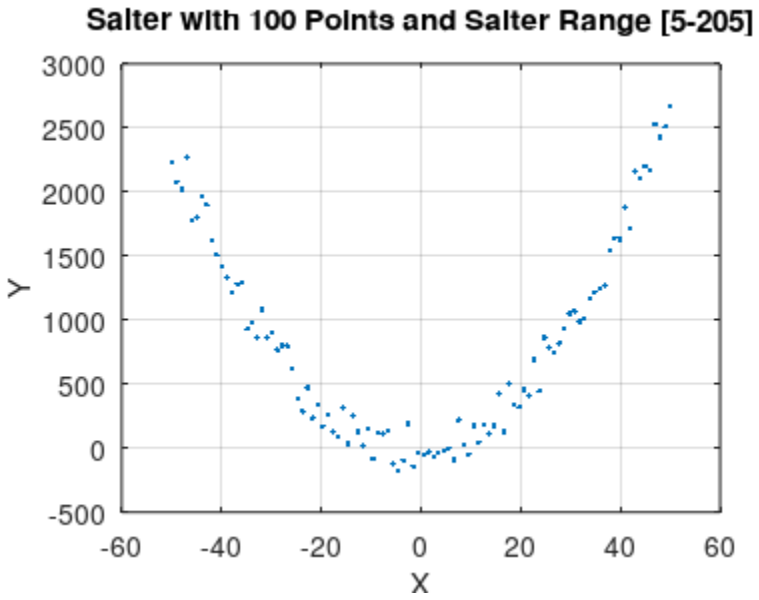
```
function saltPoints(points, saltMin, saltMax)
```

```
        for i=1:length(points)
            if(randi(2)==1)
                points(i,2) = points(i,2) + randi([saltMin, saltMax])
            else
                points(i,2) = points(i,2) - randi([saltMin, saltMax])
            endif
        endfor
    endfunction
```

A problem that occurred for me was not being able to save those new points in another vector so after salting the data, I pasted them into another vector in order to use it for the smoothing part. Figure 26 compared to Figure 12, it is the result that I expected to see.

Figure 26

Octave Salter Plot of 100 Points with Salt Range [2-205]



Discussion and Conclusion

As can be seen from the Contents section that the Smoother is missing from Part 2 and Part 3 involving JFreeChart and Apache is missing and that is due to time constraints and overestimating the workload of this project.

To begin the discussion, there is not a lot to discuss about the plotter in Part 1 as it is self explanatory and by the Figures in that section, it works as intended. If I were to build more on

this and had the capability to do so, I would have liked to allow the user to insert their own function and that would give them more freedom. On the topic of freedom, in the salter portion of Part 1, to change my code to make it better and for the user to have more freedom with it would be to allow the salting ranges to be negative as currently I only allow the ranges to be positive. I think it would make the points be more salted, however I would need to test that out in order to see if it does. The other change would be making the salting range parameters and the parameters for the formula be double instead of int. This can allow more variation in the plotting and salting areas of the project. I have my parameters as int because they were easier to work with and to have a clearer image of the graph.

For my observations for the salting section of Part 1, as seen in Figure 12 a high salt range with a high number of points shows a volatility with the shape of the function being maintained. In Figure 12, the salt range was [5-205] and there were 100 points. For a high salt with a low number of points, it creates a lot of volatility as seen in Figure 14 and Figure 15. However, if the range is really large like in Figure 16 with a range of [2-3002], the volatility would be too much and the function loses its shape entirely as seen in Figure 17. For both the low salt ranges and with high and low number of points, the volatility is either little difference or no difference at all compared to the plotted function and this can be seen in Figure 13 and Figure 18.

My observations on the smoother section of Part 1 is that with a larger window value such as 50, the graph is more weirdly smoothed as seen in Figure 21 and Figure 22. An attempt at describing the shape is sort of making little parabolas as it is smoothing. My assumption for this happening is that my code could possibly have been done incorrectly. When testing my window values, I was not able to have a window value that was greater than half of the number of points, so for 100 points I would not have been able to do a smoothing with 75 or 100 as the window value as it would give an out of bounds error. Having a smaller window value, for a parabola at least, seems to be a better choice as there are not a lot of the little curves compared

to a larger window value like in the previous figures mentioned and in Figure 20 which has a window value of 10. For Figure 19 and Figure 20, I had run the points for four rounds to see if the smoothing would improve to look more similar to Figure 6, but as you can see in the figures mentioned that was not the case. As well for both figures, after each round it can be observed that the parabola gets higher on the graph. For Figure 22, the window value was set to 50 and it resulted in a graph that had two little parabolas instead of one.

Finally for Part 2 and Part 3, as stated before they were not able to be completed or gotten to work on during this time. It can also be seen from the beginnings of this report that I spent a lot of time on researching what the assignment is about and entails as well as spending time on a tutorial for Octave mainly in learning the syntax. However, learning one part and actually applying that knowledge is the other half. So I spent a lot of time on the learning and not a lot on the application portion. For Octave, it is a big learning curve with syntax and trying to understand it with what I want to code. Plotting and salting were easy to get through with the program, but unfortunately smoothing was a hard challenge and I was not getting anywhere with that portion of the assignment. It is the same with Part 3, again due to time constraints, I was not able to put in the time in getting progress for that.

References

Apache Commons Math. (2022, December 22). *Statistics*. Apache Commons. https://commons.apache.org/proper/commons-math/userguide/stat.html.

Arras, K. O. (2023). *Kai Arras Homepage*. University of Freiburg. http://www2.informatik.uni-freiburg.de/~arras/index.html.

Arras, K. O. (2009, October). *Octave/Matlab Tutorial* [PowerPoint Presentation] Social Robotics Laboratory, University of Freiburg. http://ais.informatik.uni-freiburg.de/teaching/ws11/robotics2/pdfs/rob2-03-octave.pdf.

Dhir, R. (2022, July 12). Data Smoothing: Definition, Uses, and Methods. *Investopedia*. https://www.investopedia.com/terms/d/data-smoothing.asp.

Eaton, J. (2023). *About*. GNU Octave. https://octave.org/about.

Gilbert, D. (2021). *JFreeChart*. JFreeChart. https://www.jfree.org/jfreechart/.

The Octave Project Developers. (2023a). *11.2 Defining Functions*. GNU Octave. https://docs.octave.org/latest/Defining-Functions.html.

The Octave Project Developers. (2023b). *15.2.1.1 Axis Configuration*. GNU Octave. https://docs.octave.org/v8.3.0/Axis-Configuration.html.

The Octave Project Developers. (2023c). *10.5 The for Statement*. GNU Octave. https://docs.octave.org/v8.3.0/The-for-Statement.html.

The Octave Project Developers. (2023d). *10.1 The if Statement*. GNU Octave. https://docs.octave.org/latest/The-if-Statement.html.

The Octave Project Developers. (2023e). *2.7 Comments in Octave Programs*. GNU Octave. https://docs.octave.org/latest/Comments.html.

Oracle. (2022a). *The for Statement*. The Java™ Tutorials. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html.

Oracle. (2022b). *The if-then and if-then-else Statements*. The Java™ Tutorials. https://docs.oracle.com/javase/tutorial/java/nutsandbolts/if.html.

Shrestha, S. (2022, February 2). Data Skew in Apache Spark. *Medium*. https://selectfrom.dev/data-skew-in-apache-spark-f5eb194a7e2.