Probability and Applied Statistics

Project 2
Complete Write-Up

Ivan Wang

Table of Contents

Java Plot Library (PSS) 3
Point Class 3
Plotter 3
Salter 4
Smoother 5
Apache/JFreeChart PSS 8
Salted (Using my *PlotLibrary*'s Salt method): 9
Smoother 9
Reflection 16
Learning MATLAB (PSS) 17
Plotting 17
Salting 18
Smoothing Data 19
Reflection 21
Stock Bot Experiment 22
Collecting Stock Data 22
Calculating RSI 22
Data Visualization 23
RSI 23
Creating a Moving Average trend line (MA) 23
Writing the Stock Bot 27
Long Hold Strategy 27
RSI and Moving Average Strategy 27
SI and Moving Average Strategy 28
Testing the Algorithms 28
Algorithm Reflections 28
Reflection 29

Java Plot Library (PSS)

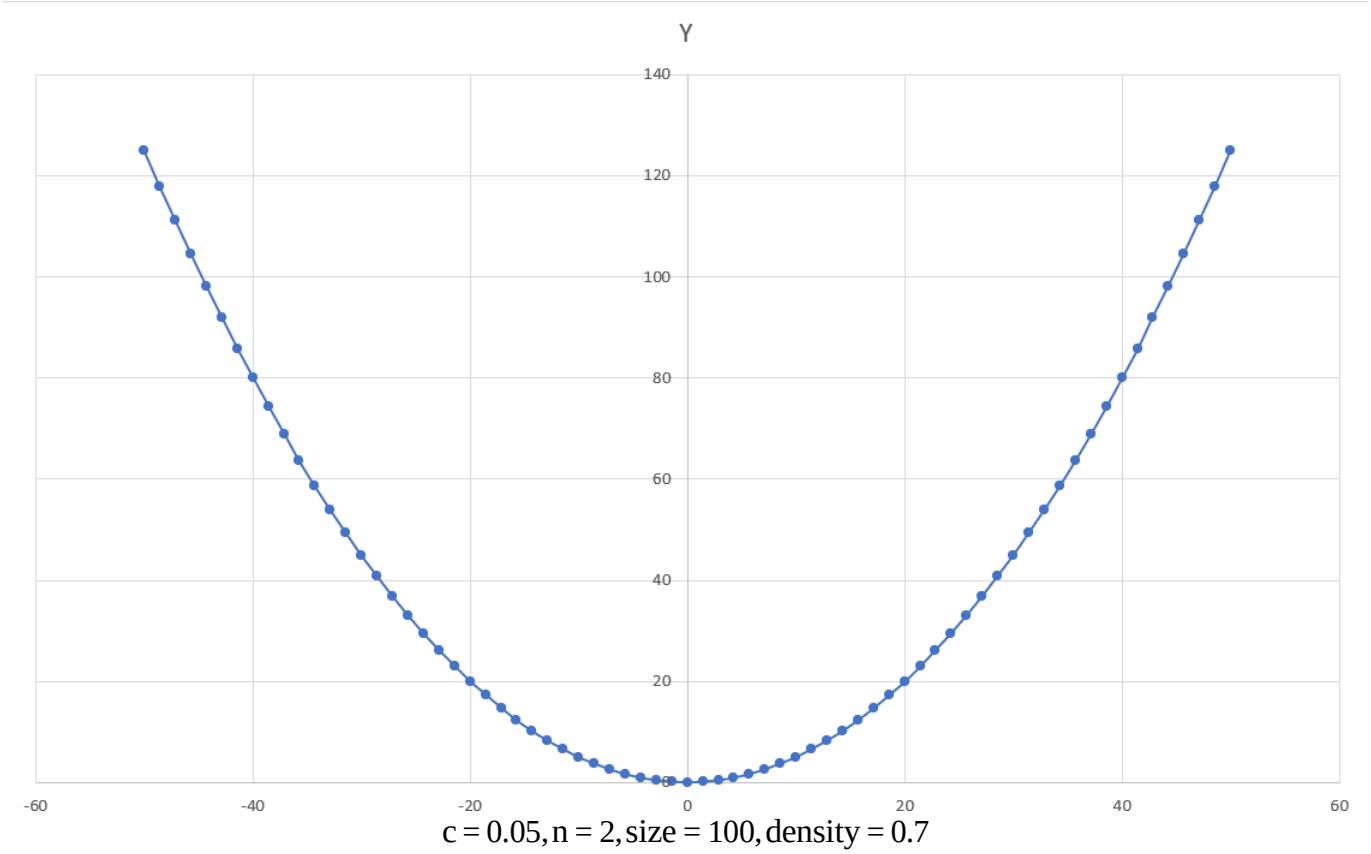
Point Class

For these programs, I made a "Point" class so that I could easily store and modify a point's values easily. A point object has double values x and y which both have a get and set method.

Plotter

Equation: $y = cx^n$

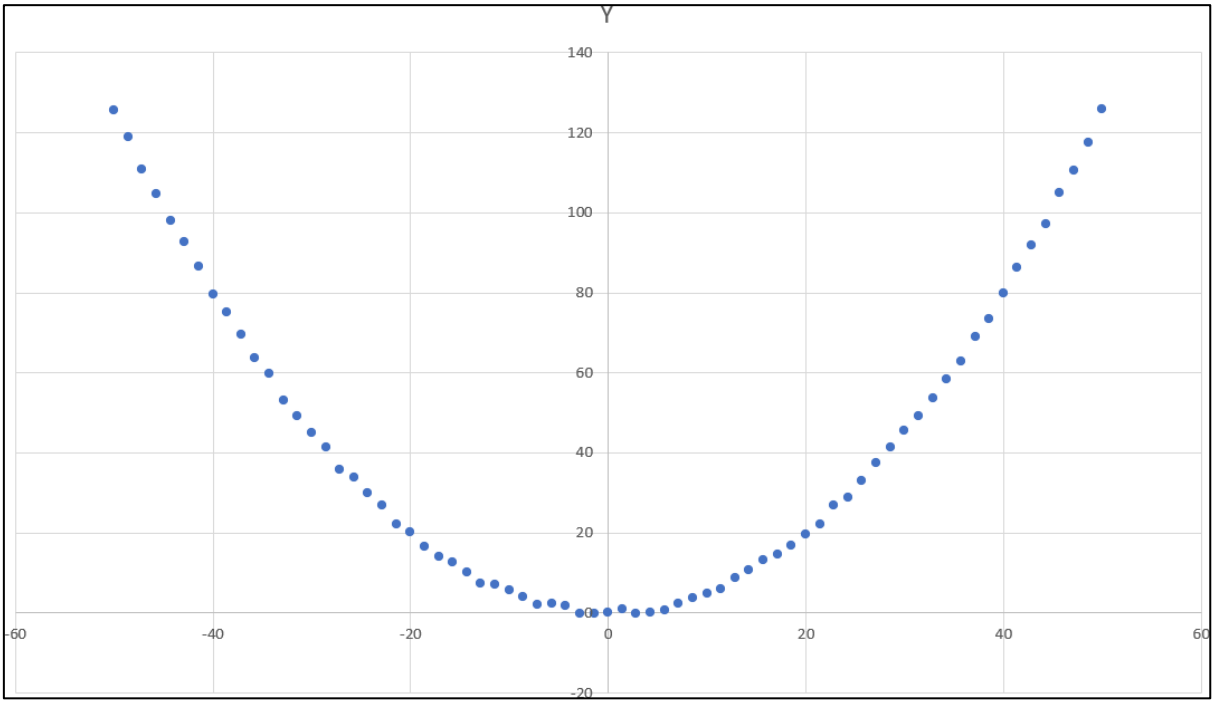
I wrote my plotting program so that the constant c and the power n were flexible. The program also accepts a size and density parameter. Size is essentially how large of an interval the graph should span across. Density is an arbitrary multiplier which determines how many points to plot. Due to the nature of determining the number of points from this density, the x values are (almost) never on full integer values.



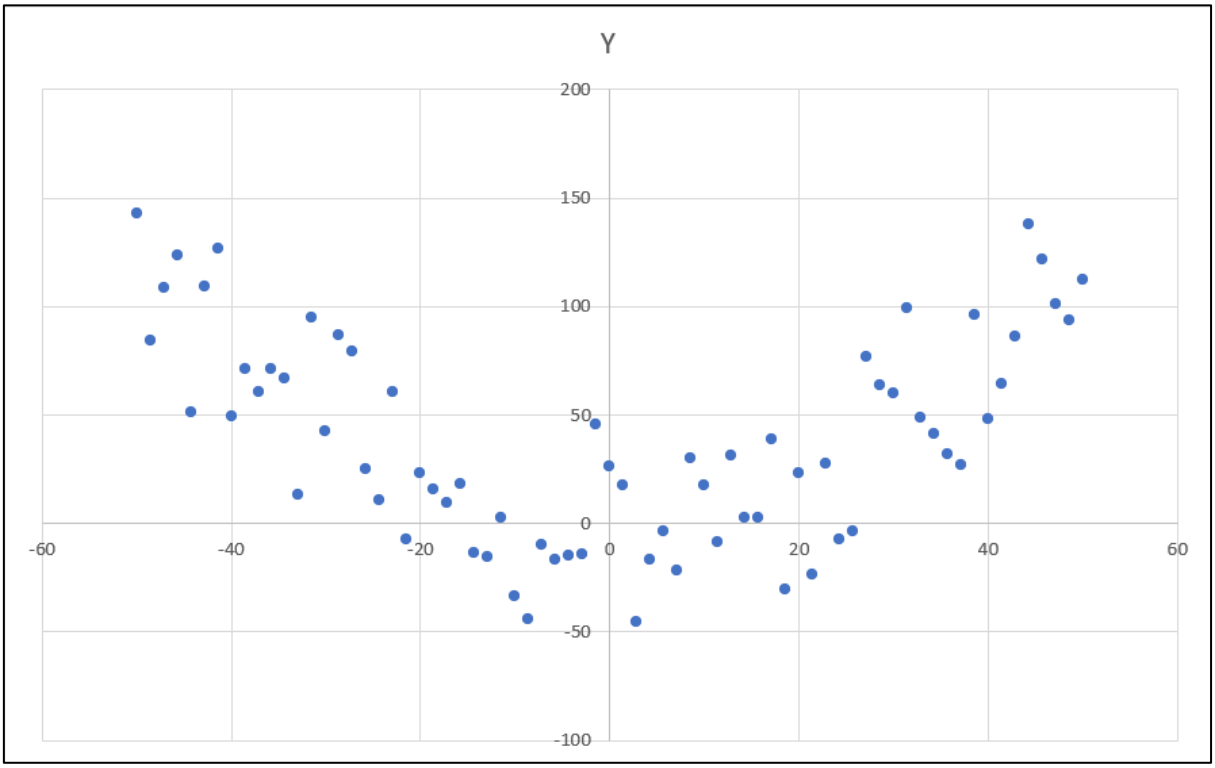
$c = 0.05, n = 2, size = 100, density = 0.7$

Salter

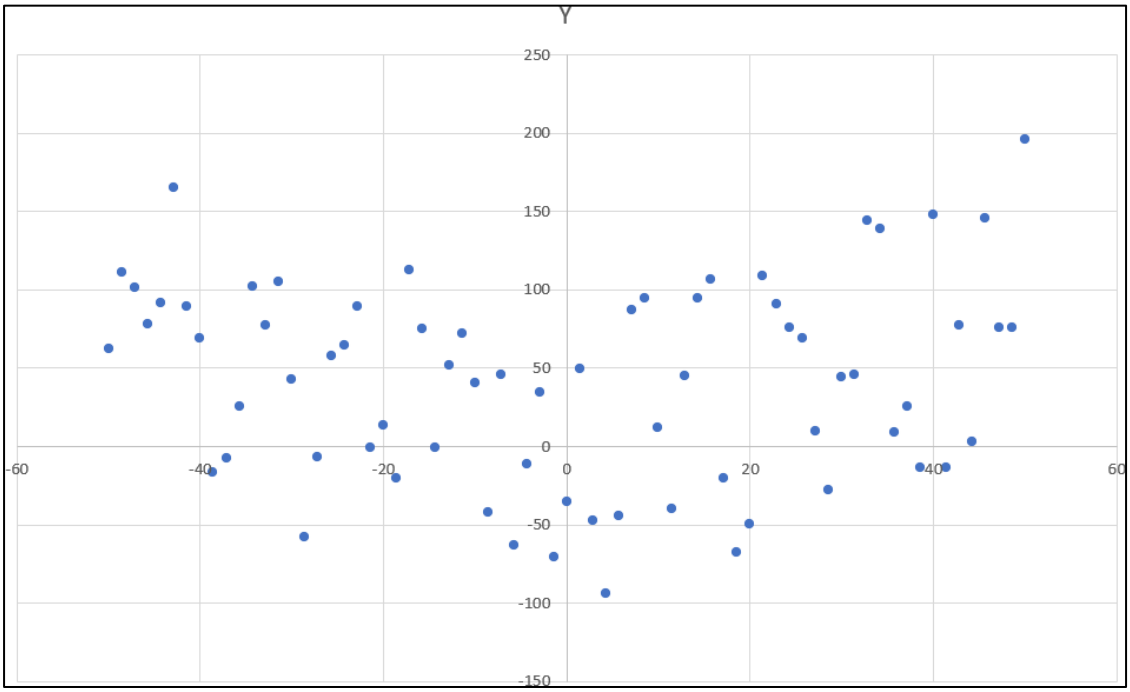
My salter program takes an Array List of Point objects and a "salinity" value, which is an arbitrary multiplier that adjusts the level of "saltiness" of the graph. Using the salter on the original graph above:



2. Salinity of 1



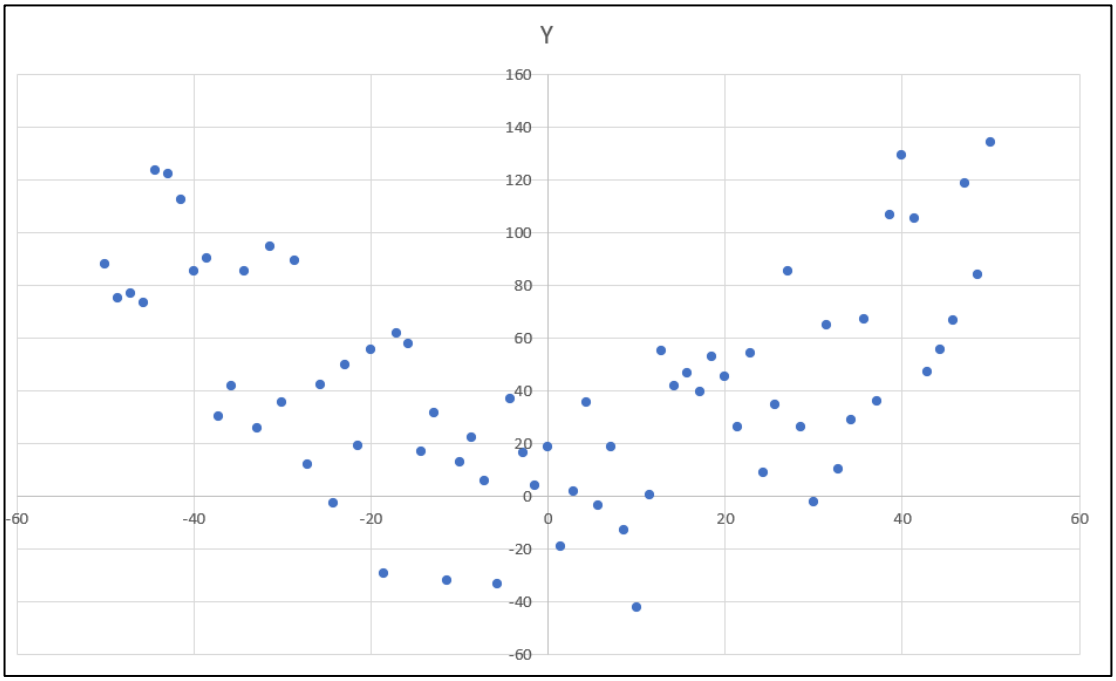
1 Salinity of 50



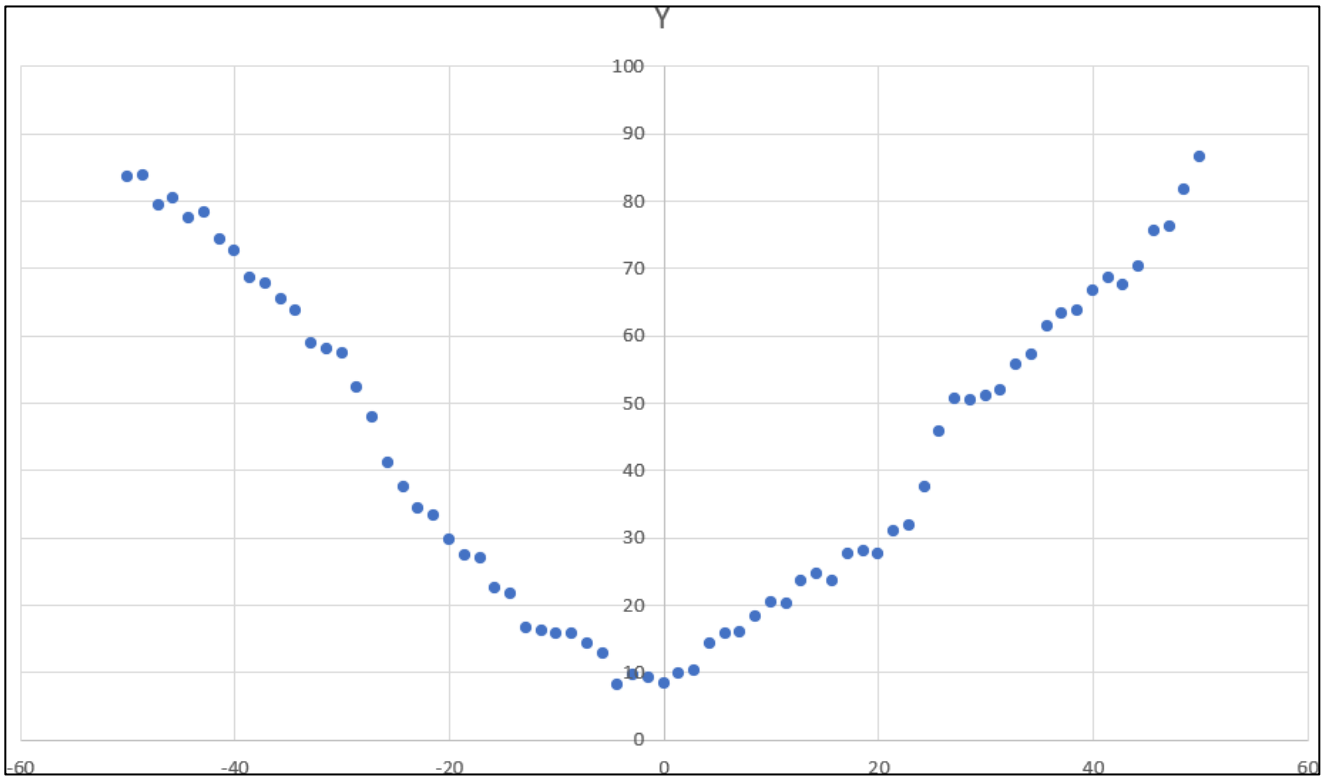
3 Salinity of 100

Smoother

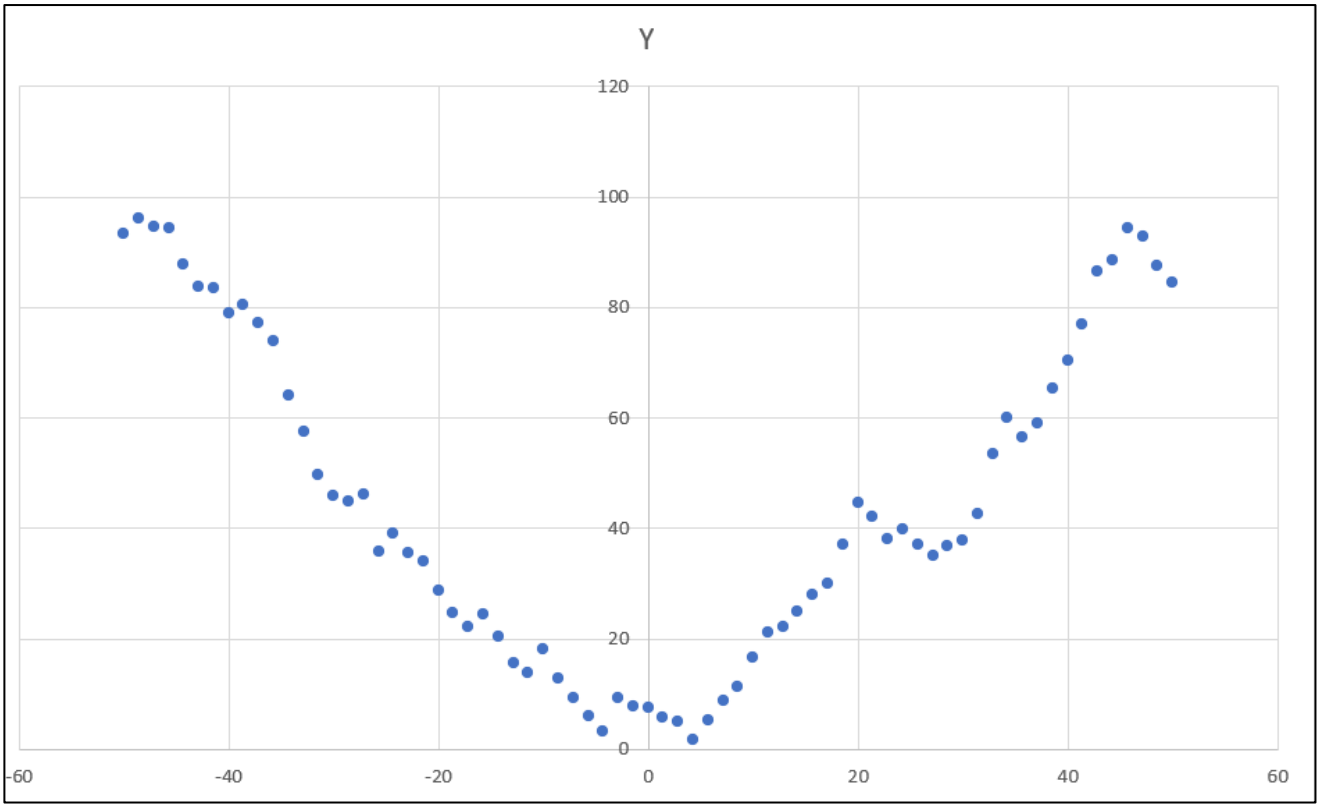
The smoother takes an Array List of Point object and a "window" integer. It takes a "window" amount of points from the left and right of each point and reassigns its y value based on the average from that window. It returns an Array List of smoothed Points.
Original salted graph:



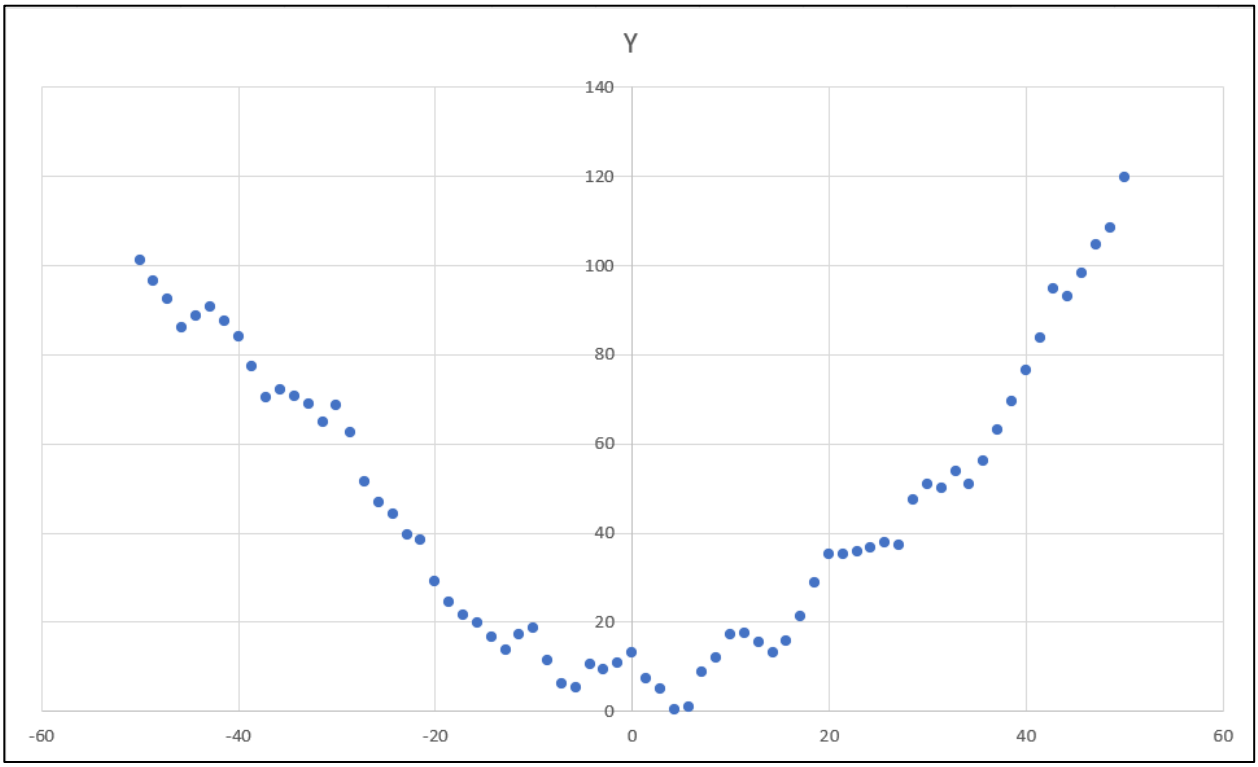
4 Salted with a salinity of 50



6 Smoothed with a window size of 1



5 Smoothed with window size of 5



7 Smoothed with a window size of 10

It seems like a low window value like 1 in the first smoothed graph gives the best results. The graphs for 5 and 10 seem pretty similar but with hiccups in different places. Once we get higher in window value, like 20, the graph seems to want to average out into a straight line. This makes sense since a larger window size would mean more points would share similar averages.

Apache/JFreeChart PSS

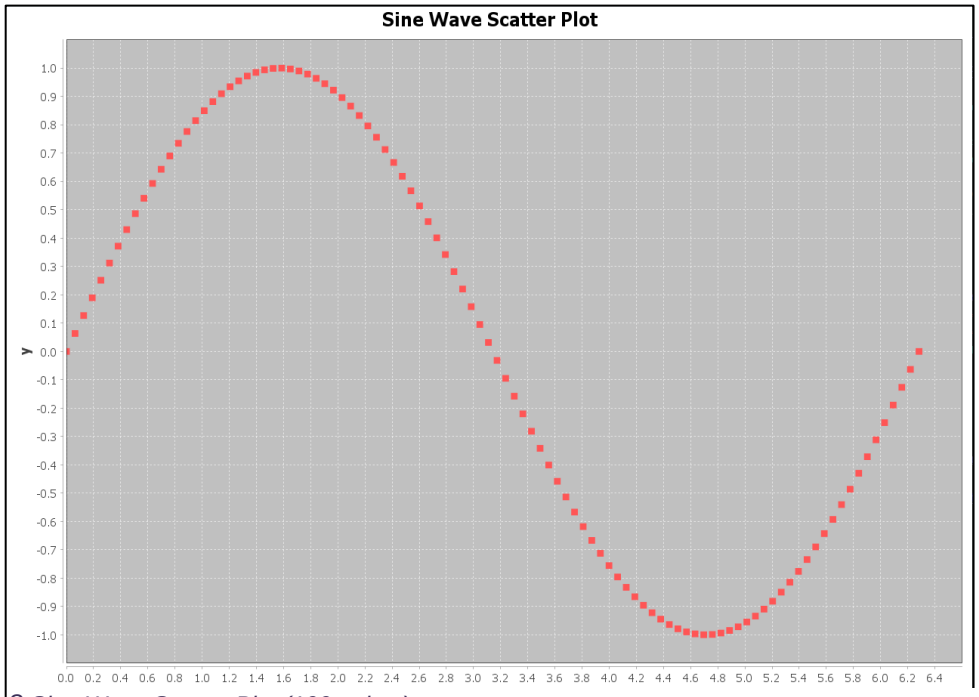
Downloaded .jar file from Apache Commons Math Download Page
Imported the core .jar file from the .zip into my project (using IntelliJ)

Creating a *UnivariateFunction* object:

```
public static ArrayList<Point> SineGenerator(int numberOfPoints, double start, double end){
    UnivariateFunction sine = x -> Math.sin(x);
    ArrayList<Point> points = new ArrayList<>();

    if(end<=start){
        return null;
    }

    double step = (end - start) / (numberOfPoints - 1);

    for (int i = 0; i < numberOfPoints; i++) {
        double x = start + i * step;
        double y = sine.value(x);
        points.add(new Point(x, y));
    }
    return points;
}
```

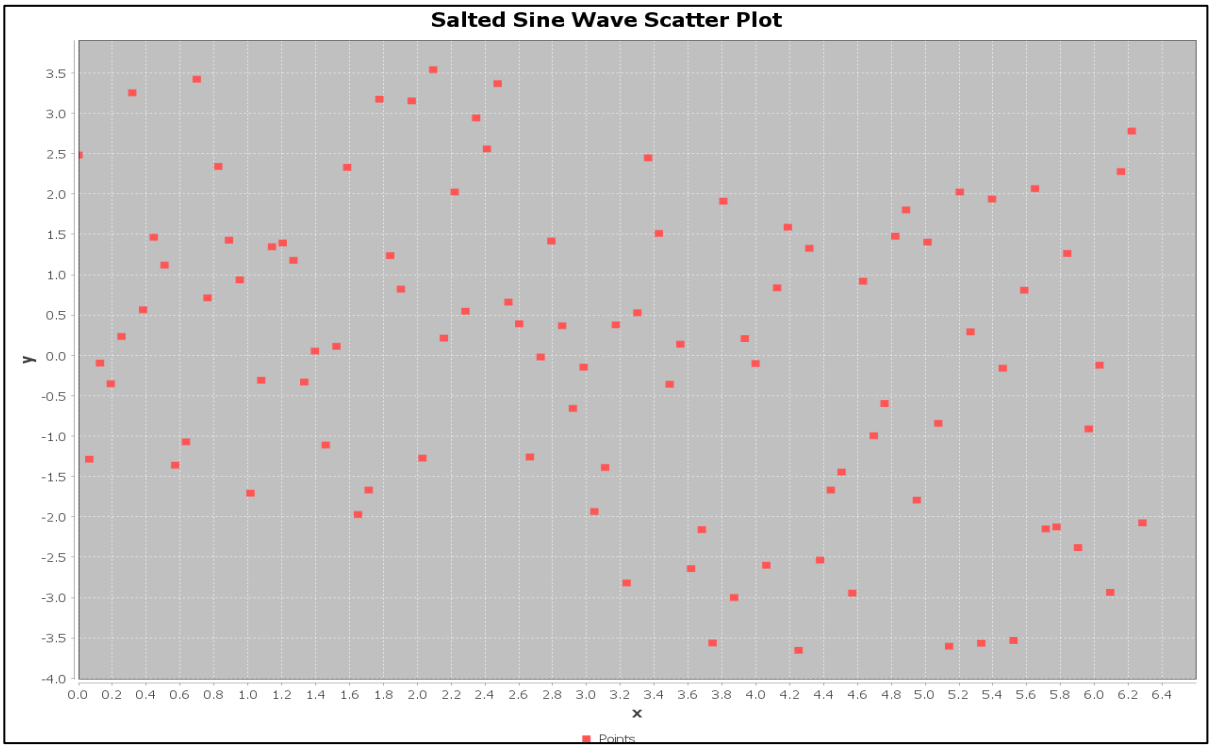
Smoother: rolling mean example
Plotter

JFreeChart Scatter Plot:



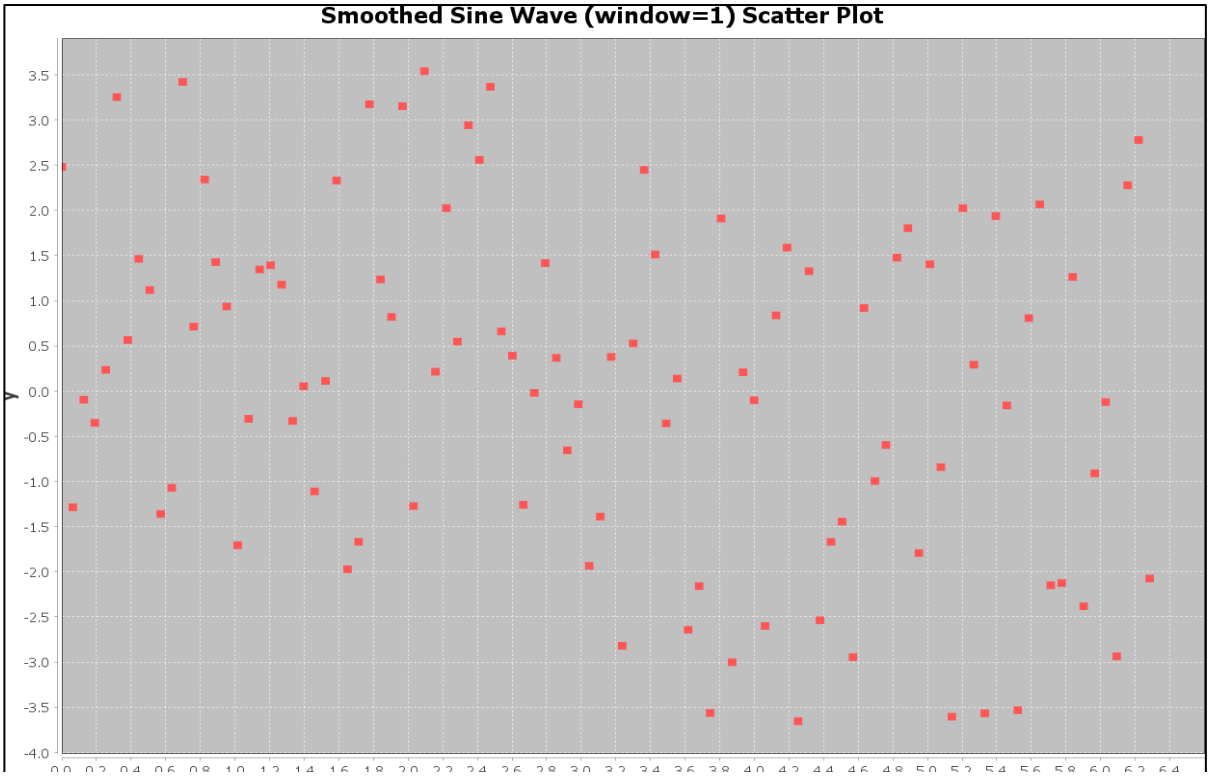
8 Sine Wave Scatter Plot (100 points)

Salted (Using my *PlotLibrary*'s Salt method):



9 Saltiness = 3 (100 points)

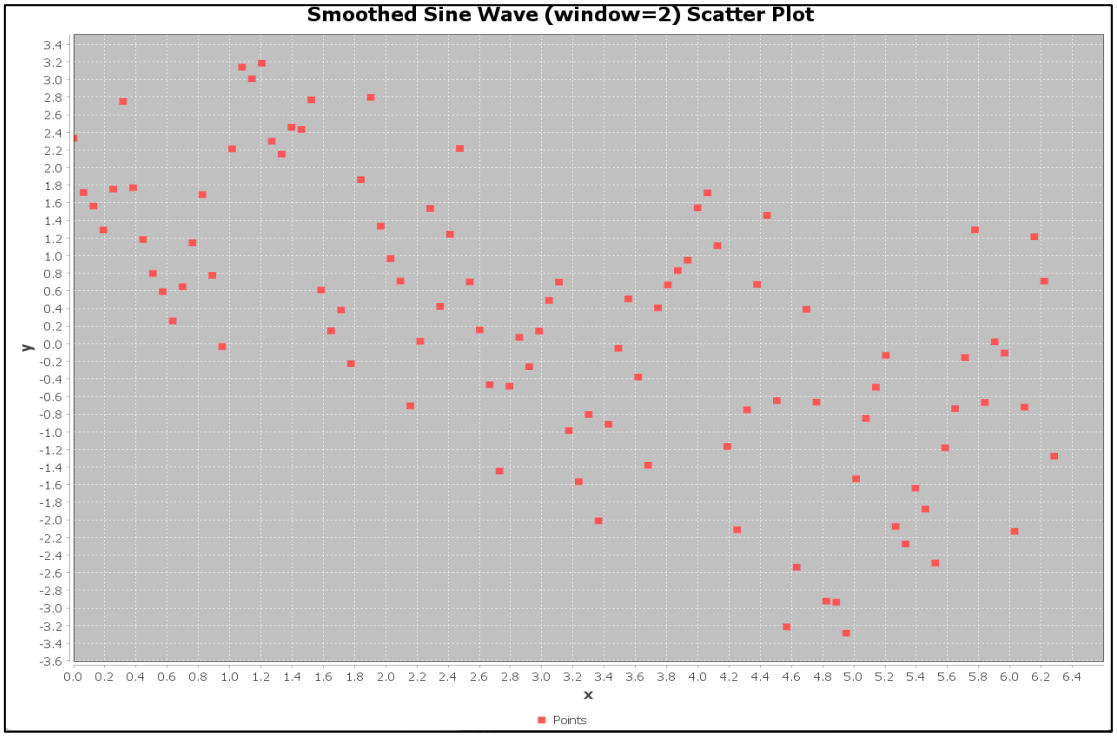
Smoother



10 window size = 1 (100 points)

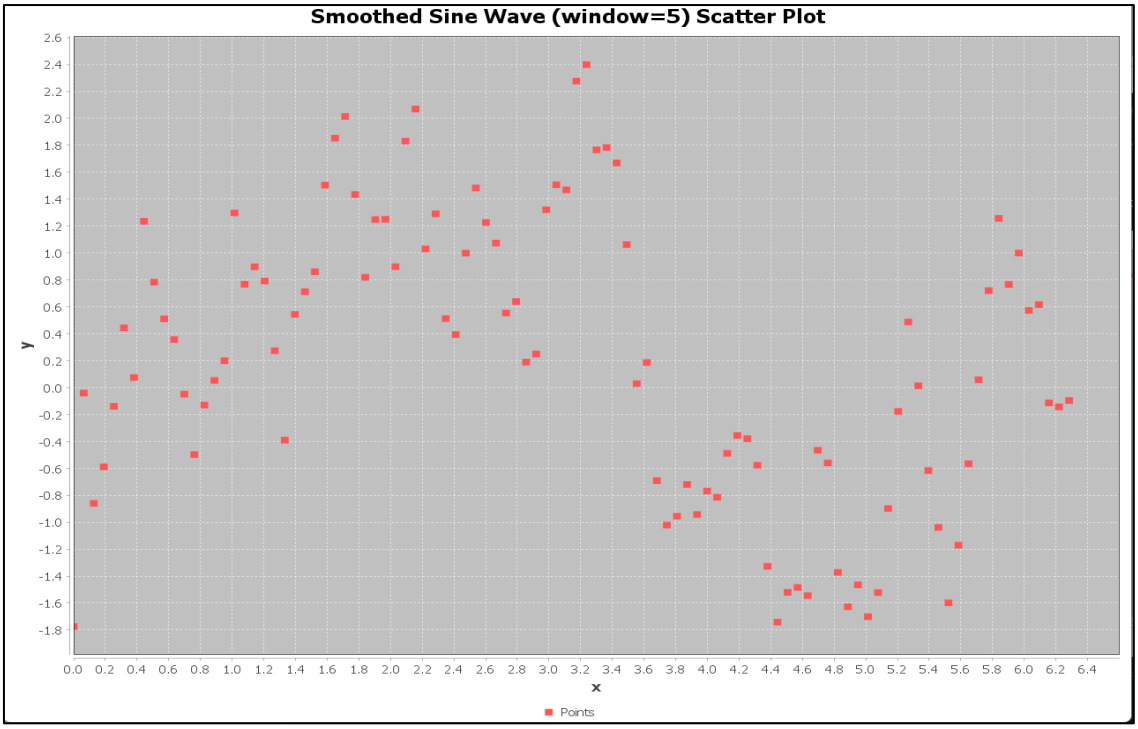
A window size of 1 does not change anything. This is probably because it's only taking itself as the one point, instead of taking one point from the left and one from the right.

Redone with window of 2:



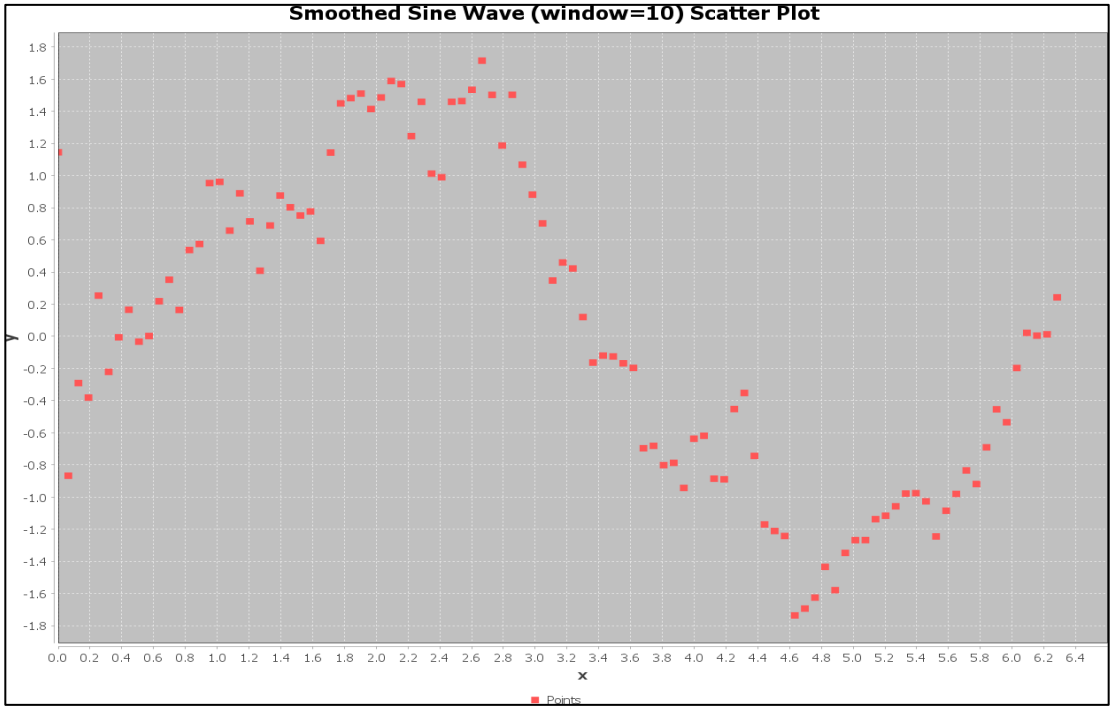
11 Smoothed with window size = 2 (100 points)

Enlarging the window size makes a change. Let's try bigger windows:
window = 5:



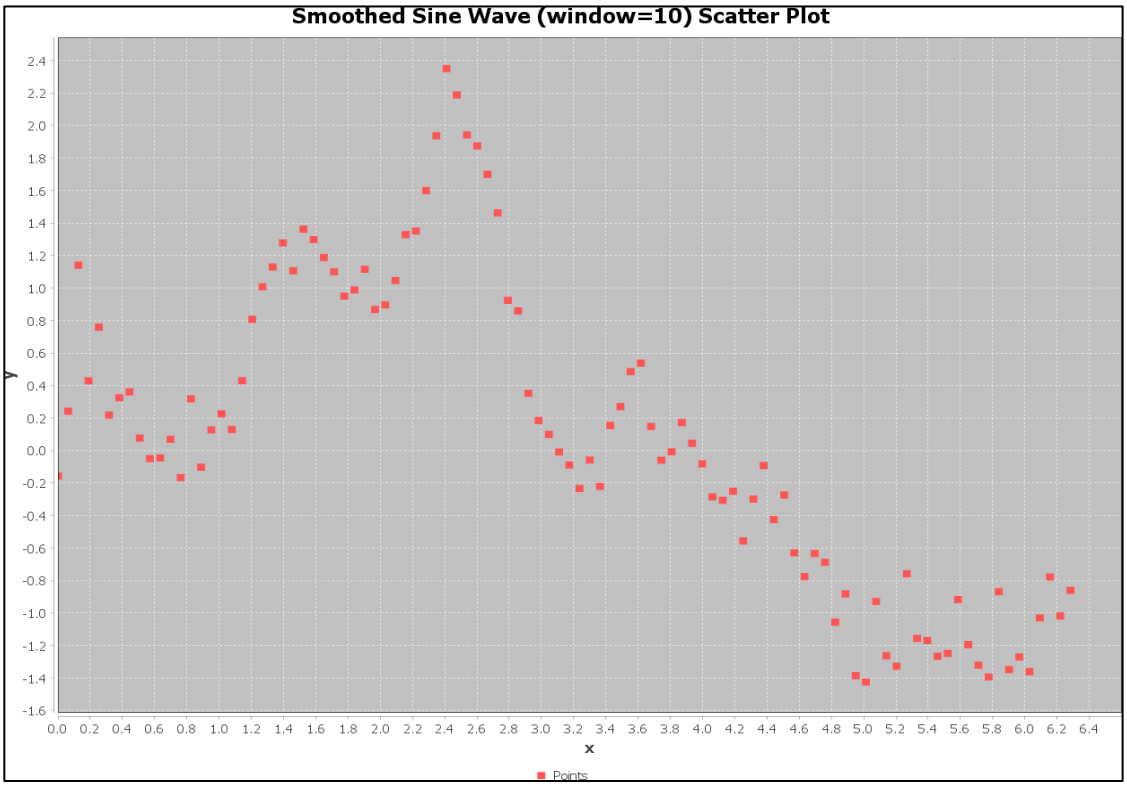
This is much better than the previous two.

window = 10:

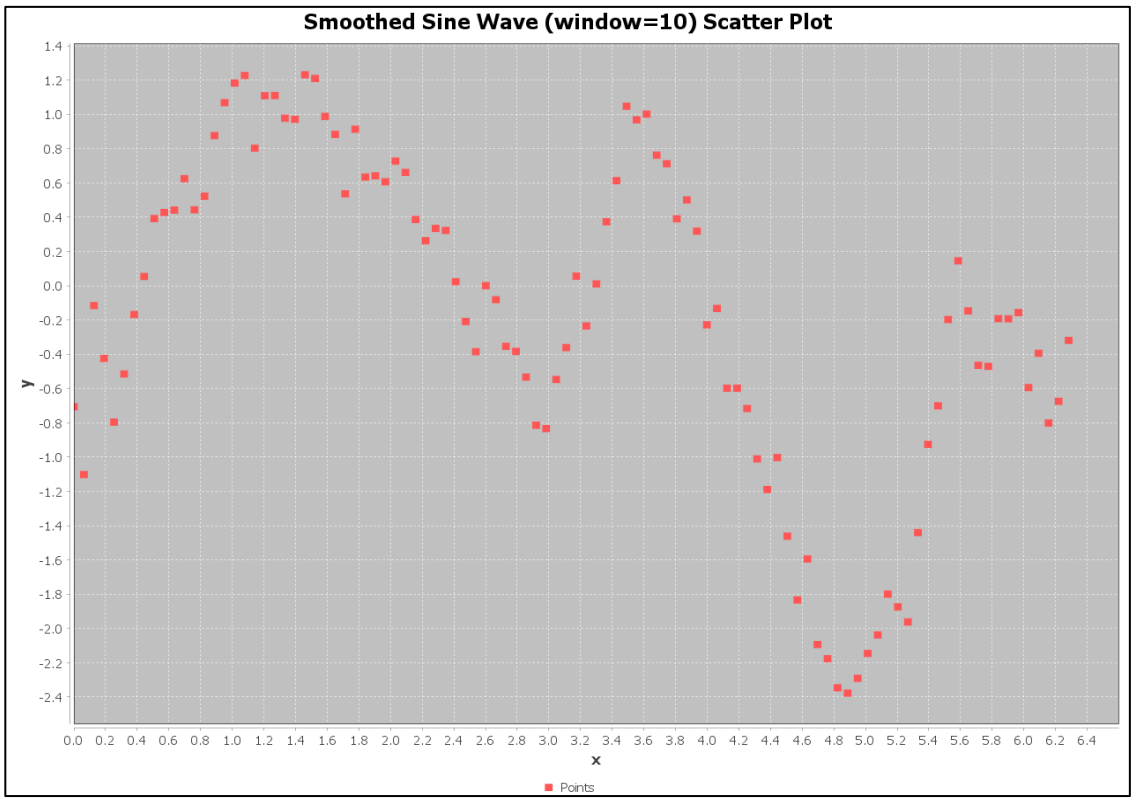


12 This was my third try with window of 10

For some reason, window = 10 will either give me graphs which are very close to the original or very far. No in-betweens.

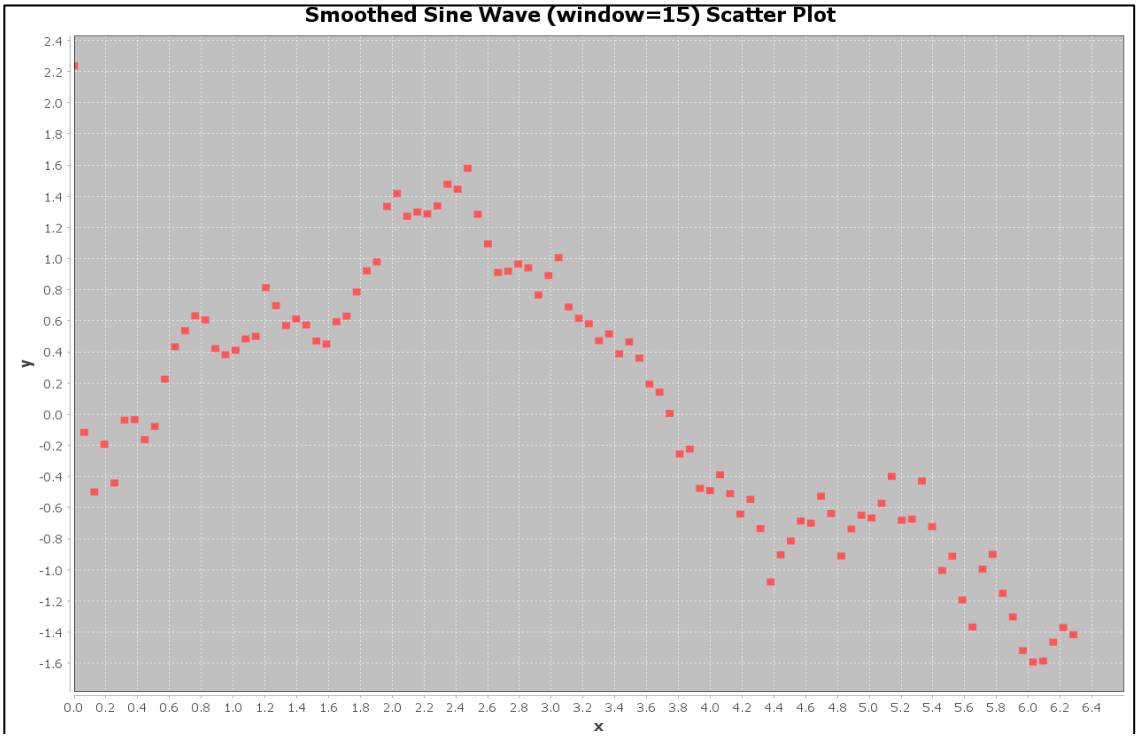


13 An example of a botched smoothing at window 10

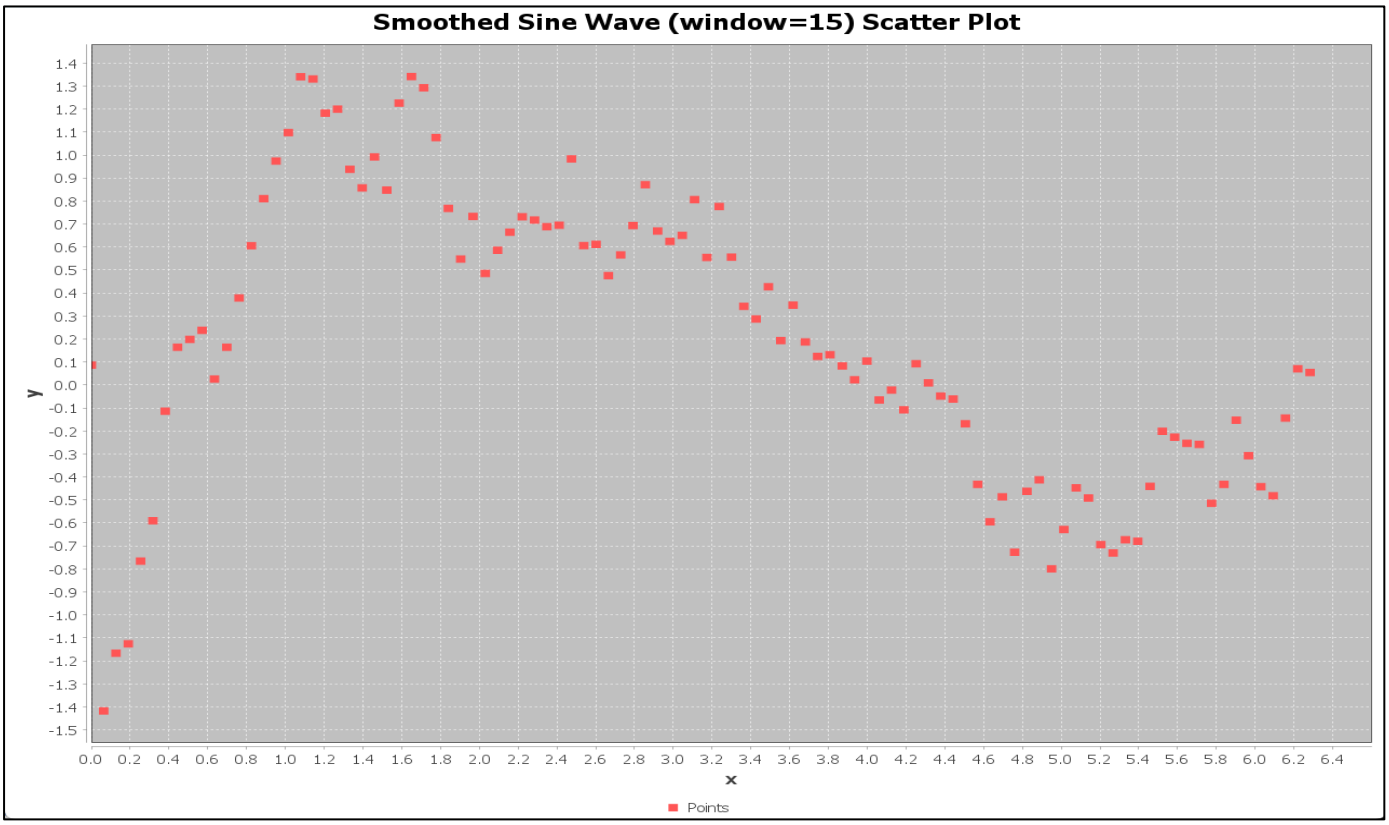


14 Another example of a suspicious smoothing result

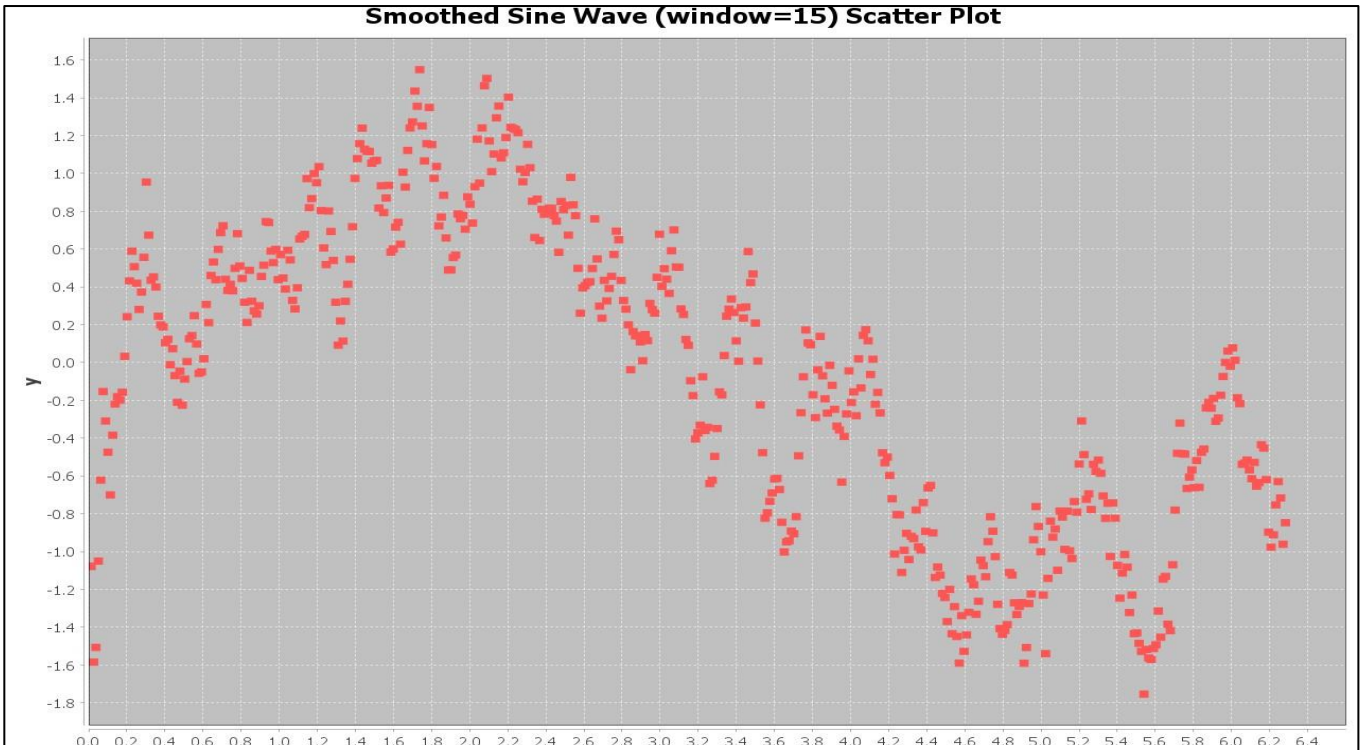
Let's up the window size again. Window of 15:



13 First one generated. Unsure if its better or worse than window = 10

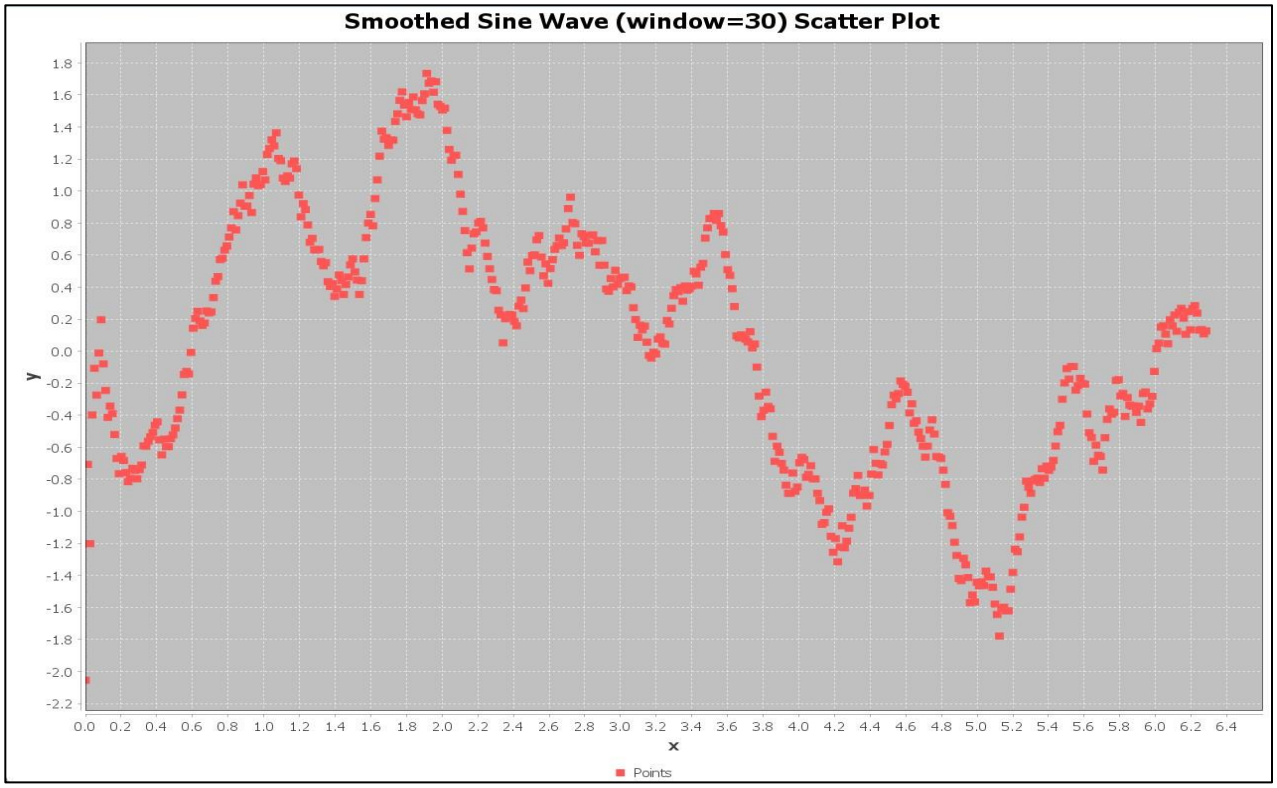


Nothing new is happening. Let's change the number of points to 500 instead of 100. Here is the smoothed graph:



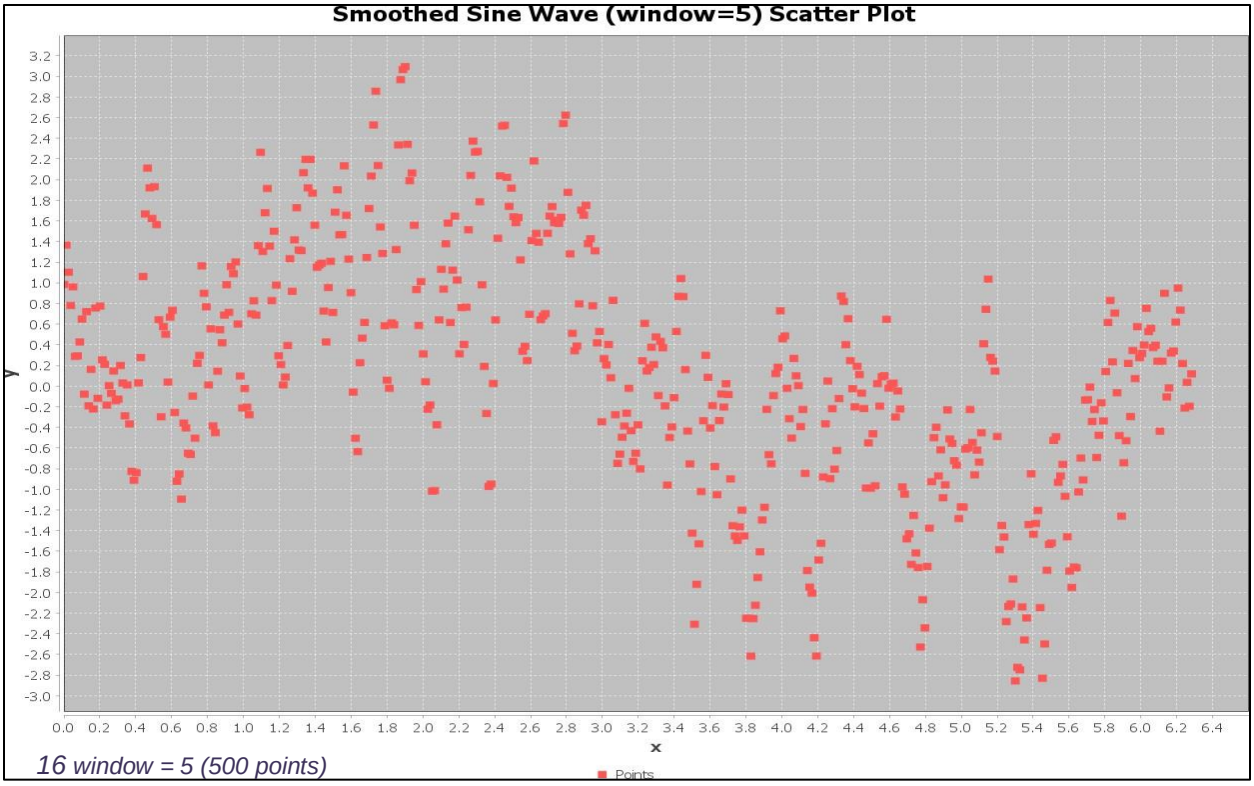
14 window = 15 (500 pts)

Smoothing window of 30 (500 points):



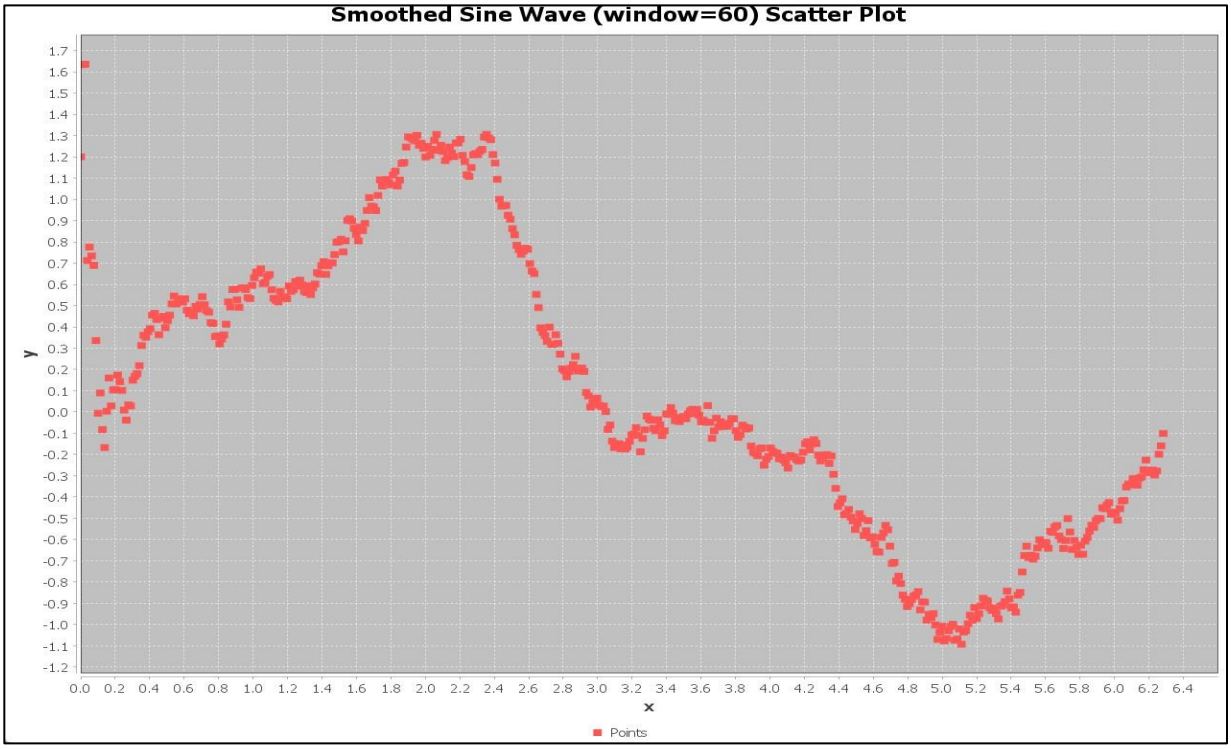
15 window = 30 (500 points)

Smoothing window of 5 (500 points):



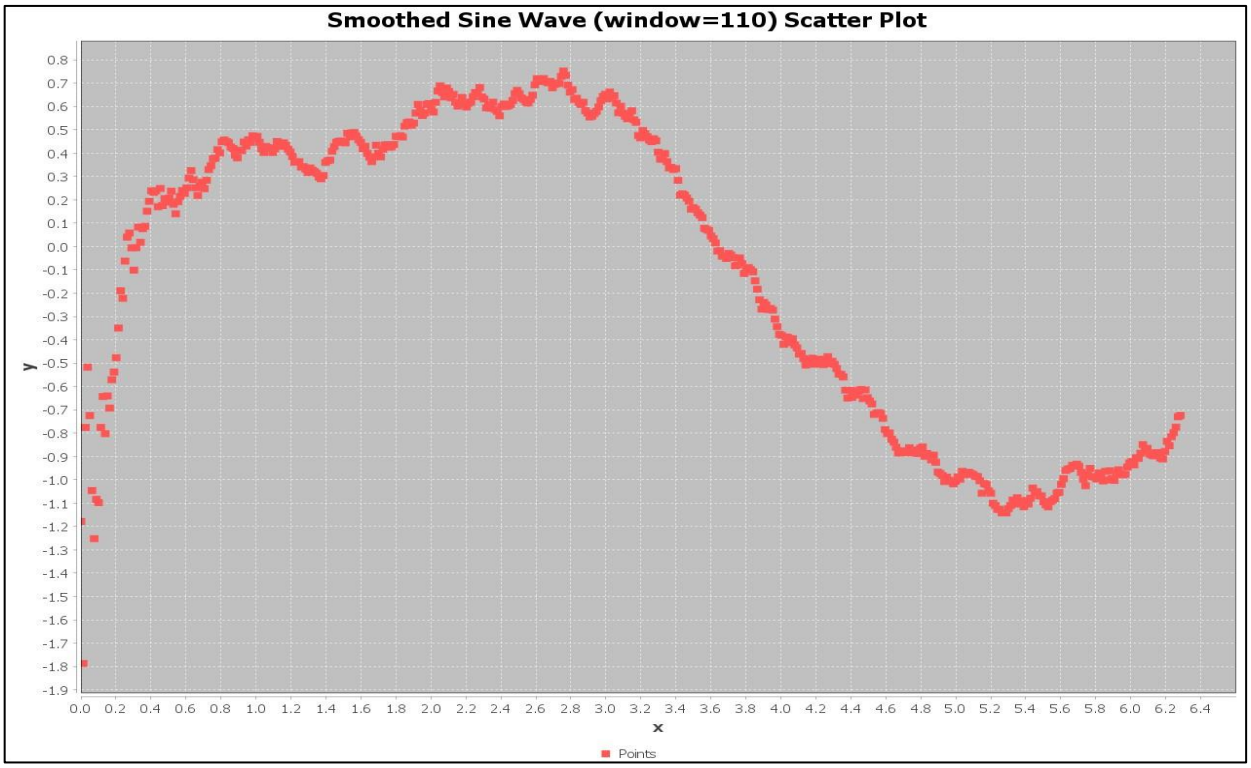
16 window = 5 (500 points)

Smoothing window of 60 (500 points):

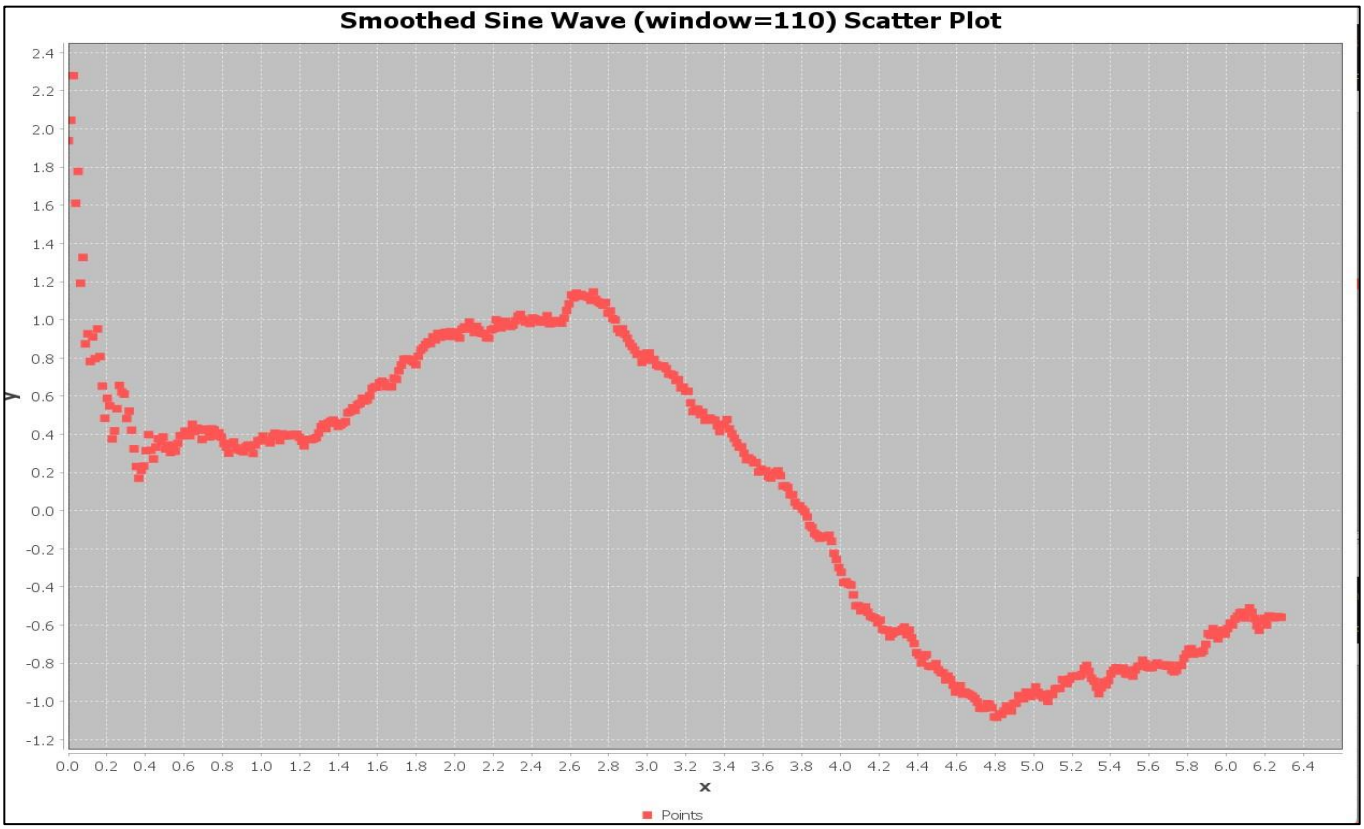


17

Smoothing window of 110 (500 points):



18



19

Reflection

As I generate graphs with higher and higher window sizes, I noticed a weird trailing pattern happening at thebeginning. Since this is utilizing a rolling mean to generate these graphs, a larger window size means the beginning will have a sparse tail. Since the averages of the beginning points are using less data points total there will be more variation, as opposed to the end where the points are on top of each other and start to flatten out. But the tail will either strongly trend up or down, depending on the first few points of the salted data.

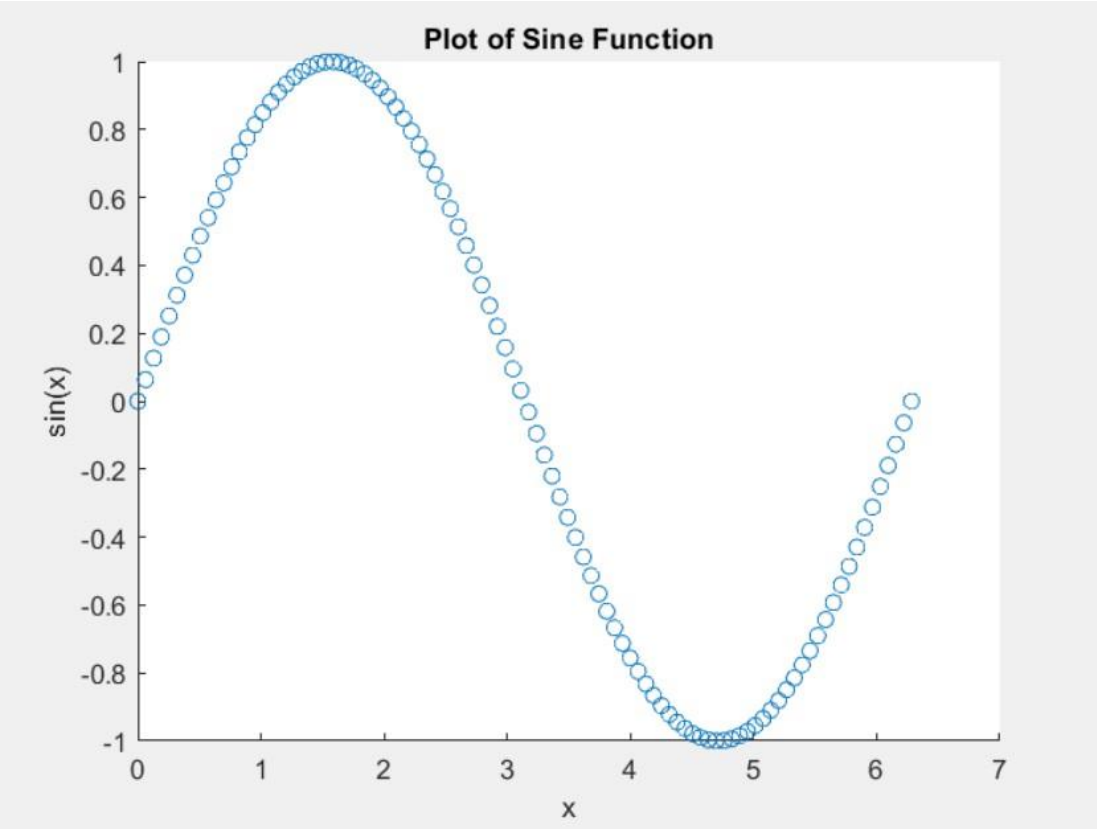
Learning MATLAB (PSS)

Plotting

Creating a scatter plot of a Sine Wave
Scatter plots in MATLAB

$$y = \sin(x)$$

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

y = sin(x);

scatter(x,y)

xlabel('x')

ylabel('sin(x)')

title('Plot of Sine Function')
```



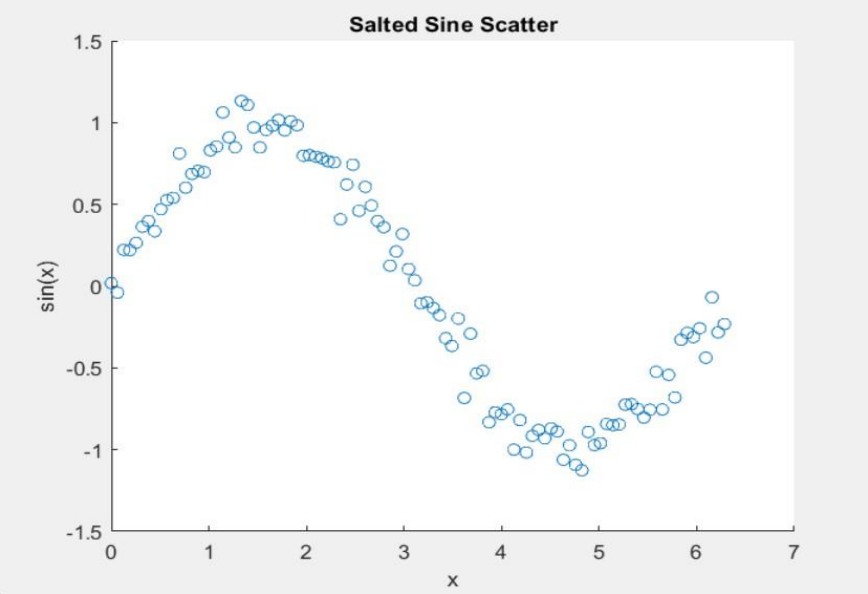
Scatter plot of a sine wave

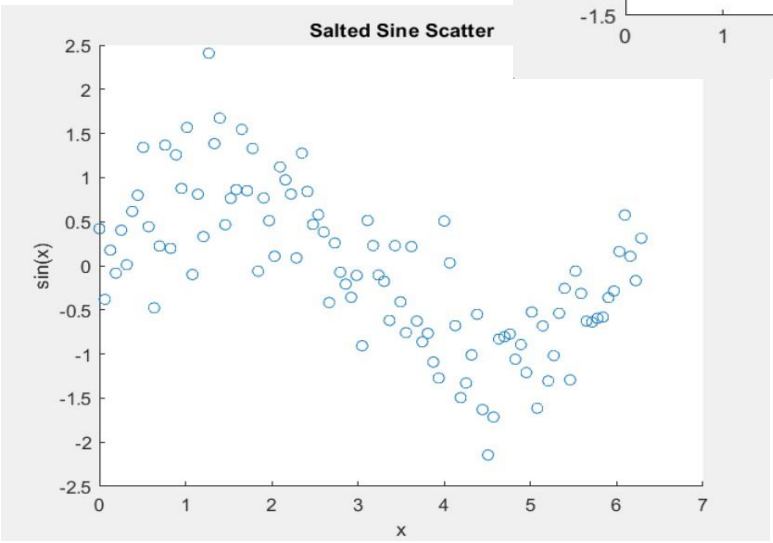
Salting

Noise level(adjustable) is multiplied with a random -1 to 1 value and added to each y value. Therefore a larger noise level would result in a “saltier” graph.

```
noise_level = 0.1;

salty_y = y + noise_level * randn(size(y));

scatter(x, salty_y)

xlabel('x')

ylabel('sin(x)')

title('Salted Sine Scatter')
```

Salting with a noise level at 0.1:





Salting the changed noise level to 0.5

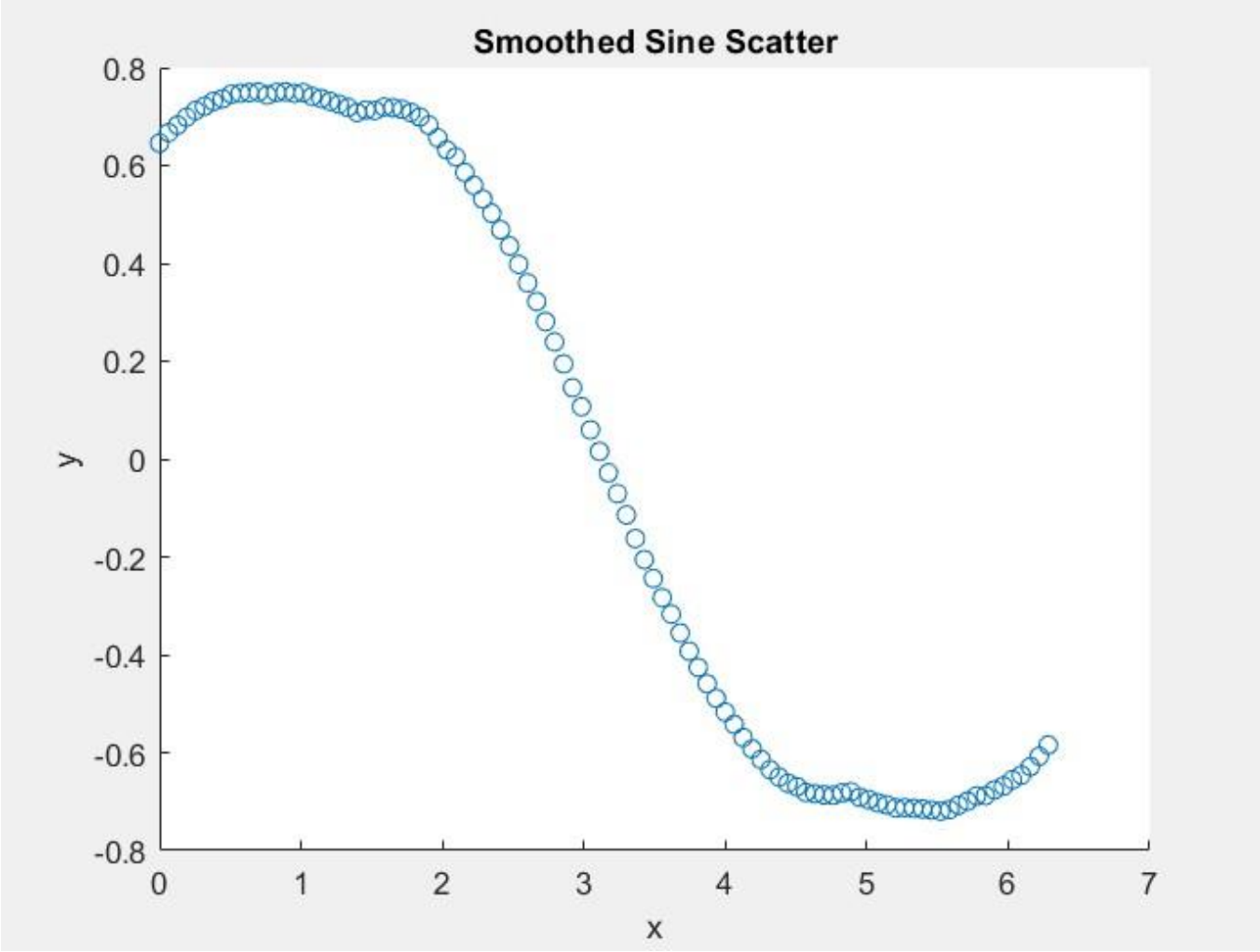
Changing the noise level to 0.5

Smoothing Data

Documentation for *smoothdata*

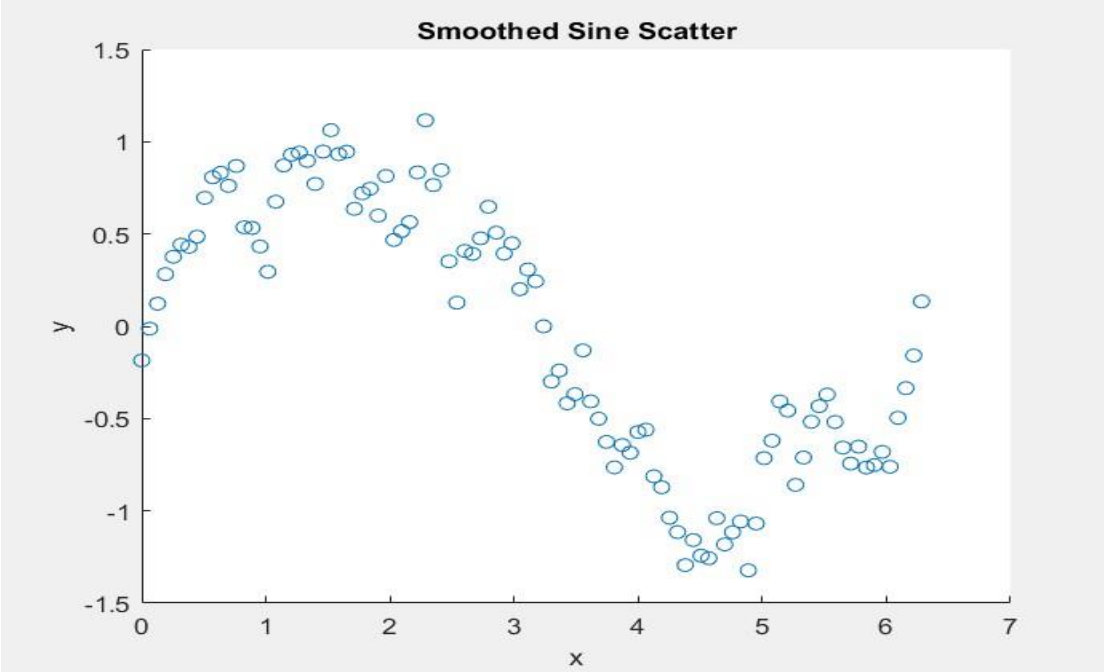
Smoothdata is a function developed MATLAB which smooths noisy data, which is perfect for this project. When called by only passing an array of values in, the function will use a moving average where *smoothdata* determines the window size by looking at the elements in the array.

By entering my salted y values into *smoothdata*, I get this:

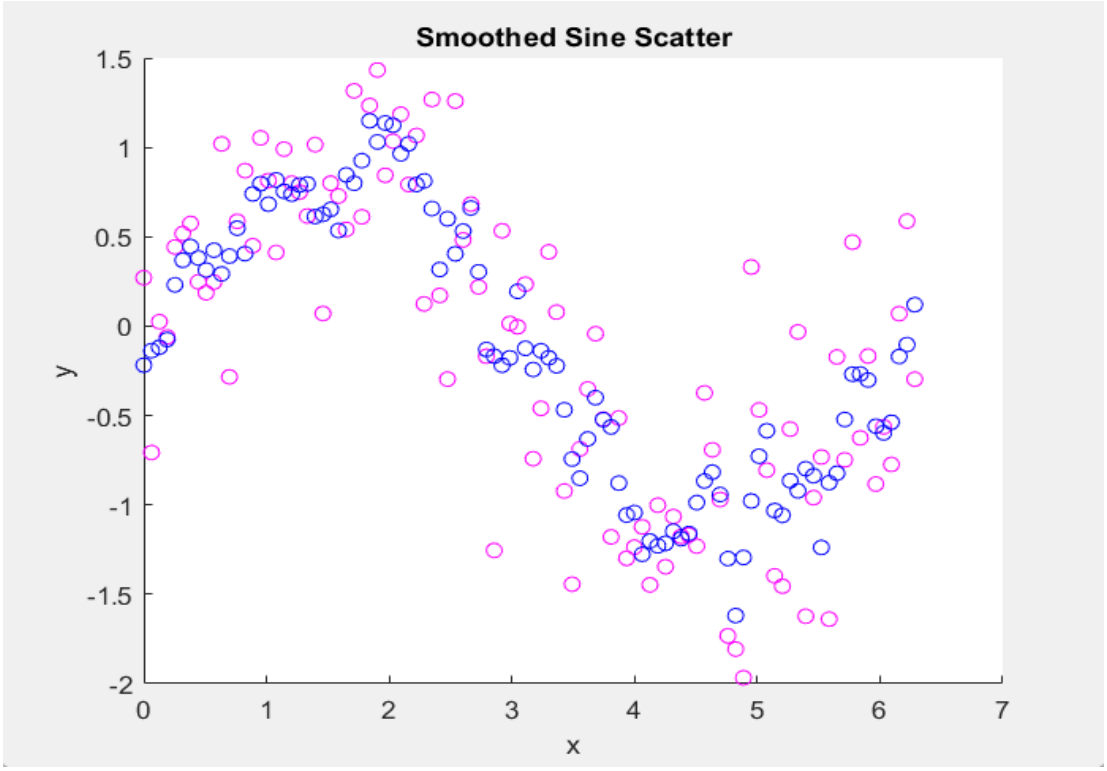


20 Noise Level 0.1 Smoothed. Hey that's pretty good!

Noise Level 0.5 Smoothed:



Noise Level 0.5 Smoothed overlayed the original salted:



21 Magenta: Salted; Blue: Smoothed

Code for the salter, smoother, and overlaying the two:

```
noise_level = 0.5;

salty_y = y + noise_level * randn(size(y));

scatter(x, salty_y)

hold on

smoothed_y = smoothdata(salty_y);

scatter(x, smoothed_y,"blue")

xlabel('x')

ylabel('y')

title('Smoothed Sine Scatter')
```

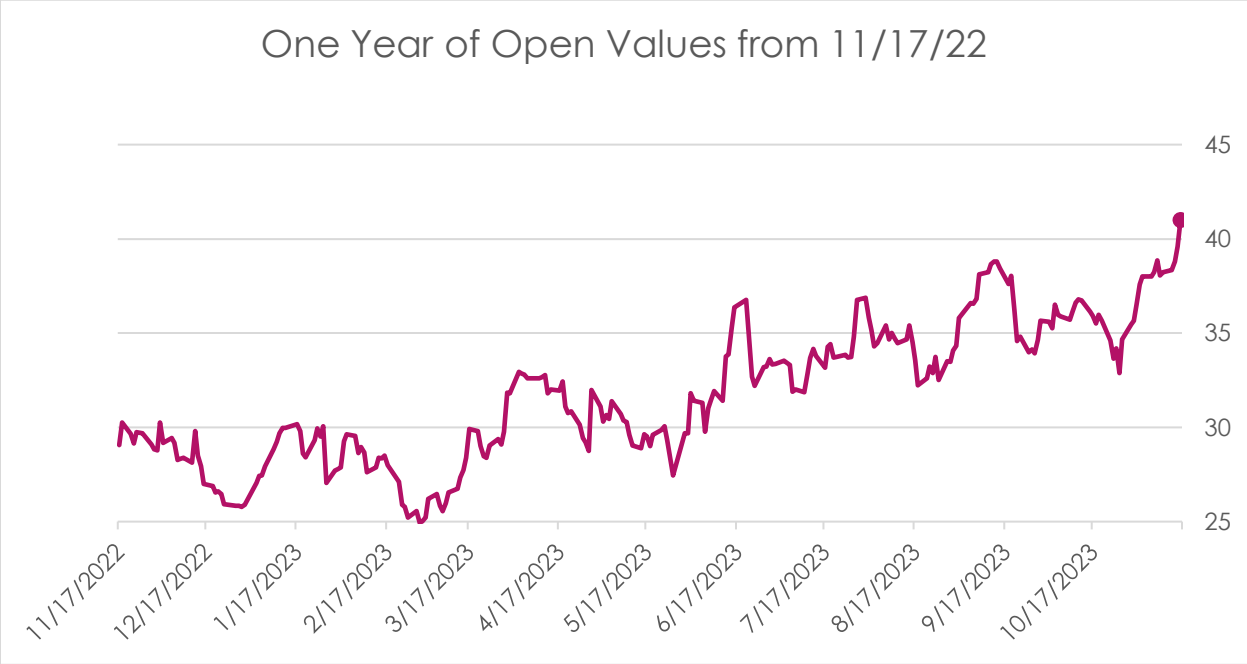
Reflection

Messing around with MATLAB was probably the most fun out of all the projects. Since it is a program made specifically for data manipulation and visualization, it was very easy to play around with graphs and make them pretty. Not needing to implement something like a salter or smoother and having it native to the language is powerful since it cuts out the middleman steps between you and the goal of the data manipulation. I can imagine that we barely experienced a fraction of how powerful MATLAB can be in other fields of study. I plan to revisit MATLAB in the future to learn it in depth.

Stock Bot Experiment

Collecting Stock Data

Visiting Yahoo Finance, I picked INTC (Intel) for my Stock. Since I downloaded the data on 11/16/23, it gave me INTC's historical data (1 year) from 11/17/22 to 11/16/23.



To store this data for use in my programs, I created a Java class named *`StockData`* which holds the information about a single date of the stock. This includes the opening price (open), the highest price of the day (high), the lowest price (low), the closing price (close), the adjusted closing price (adjClose), and the trading volume (volume). This structure allows for easy access and manipulation of the stock data throughout the rest of the Stock Bot.

Calculating RSI

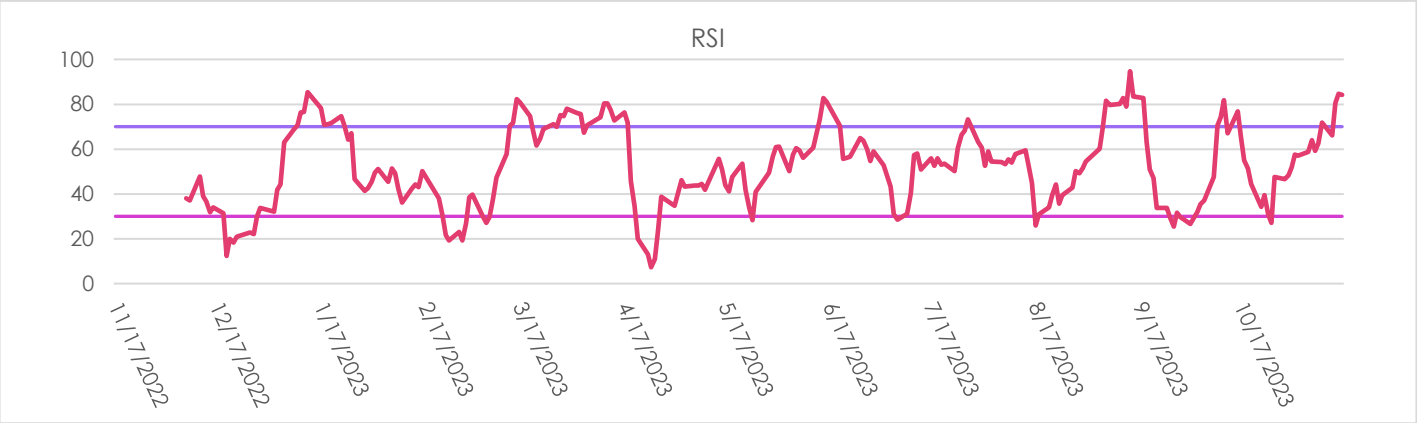
The Relative Strength Index (RSI) is a momentum heuristic used in trading to identify when a stock is overbought or oversold. RSI is a proportion, so it ranges from 0 to 100 and typically uses 30 and 70 as thresholds to determine oversold or overbought conditions, respectively.

To begin a program to calculate RSI and other useful functions, I developed *`StockBot`* class. The *calculateRSI* method takes a list of *`StockData`* objects and computes the RSI based on the formula taken from Macroption. This calculation uses the closing prices of the past 14 days and finds the average gains and losses. These gains and losses are used to find the relative strength (RS) and subsequently the RSI.

Data Visualization

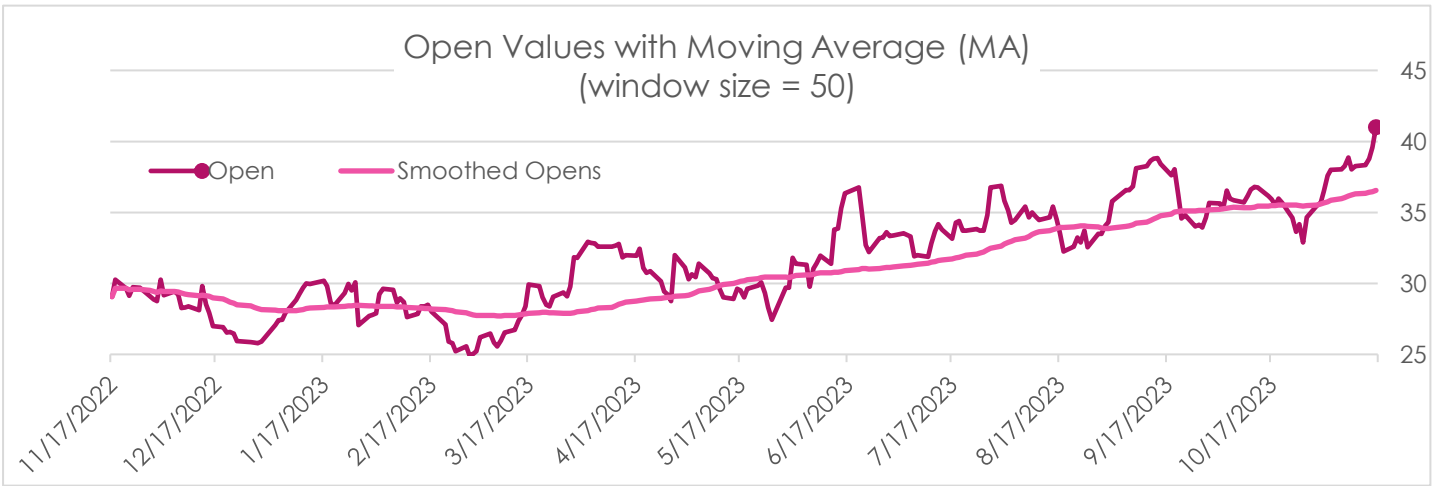
RSI

To visualize my RSI calculations, my program includes a function which exports *StockData/Double* objects to a *.csv* file, which can be used in Excel to manipulate and create visualizations of the data. The StockBot also includes a method which uses the *JFreeChart* library to plot the RSI results (or any stock data) over time. This graph representation of my data is important so that the RSI's behavior meets the expected values.



Creating a Moving Average trend line (MA)

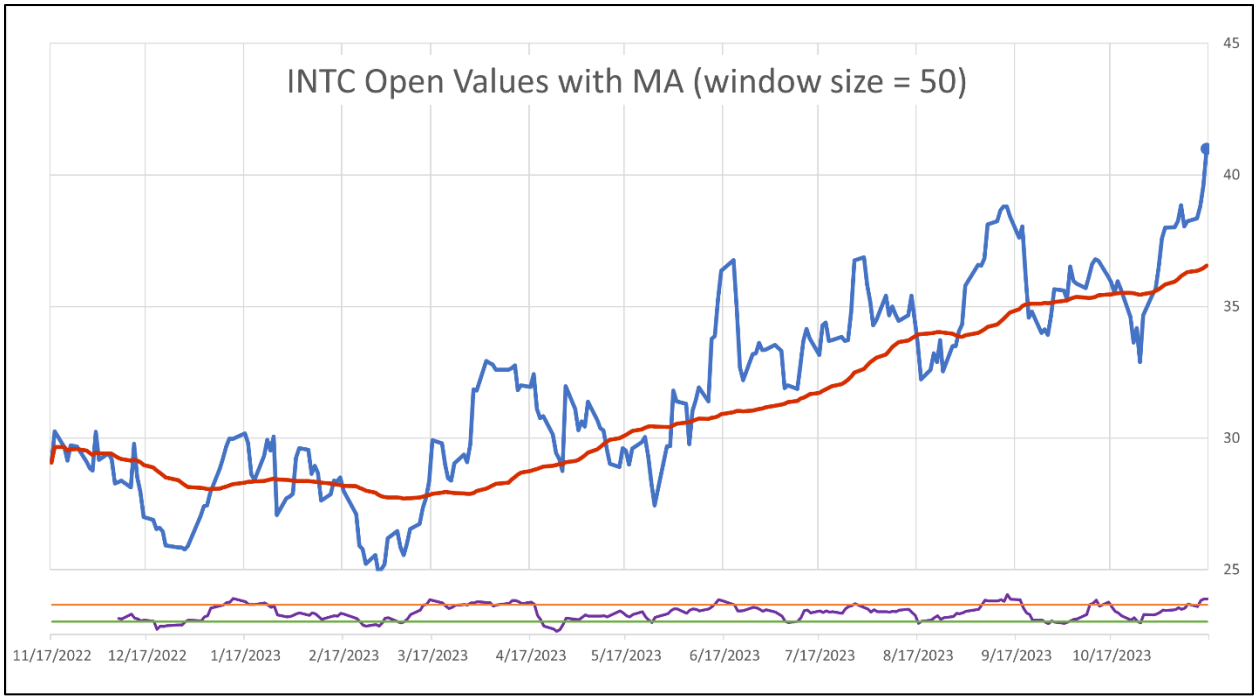
A moving average (MA) is another indicator, like RSI, that helps smooth out stock data over time, which allows us to see the direction of the trend. In the *StockBot* class, the method *movingAvg* calculates the simple moving average for a given period. The method returns the average of the opening values over the last 50 days (or any given period), and the line for the whole graph can be made by looping through this calculation for each day.



Using the RSI and MA, we can make an elementary algorithm for determining when to buy/sell shares for the Stock Bot in the next section.

Yahoo Finance's Graph with MA (window of 50) and RSI:

My Graph with MA (window of 50) and RSI:

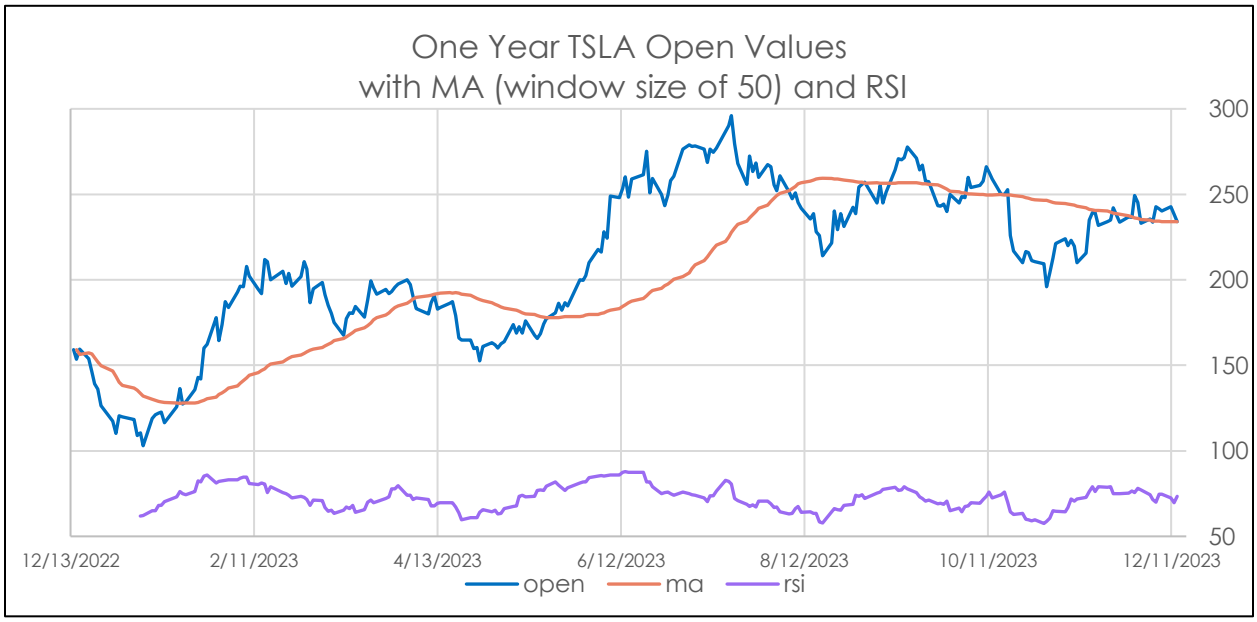


22(Note: The oversold/overbought thresholds on Yahoo's RSI graph are 20/80, while the lines on my RSI graph are at 30/70.)

I'd say they look similar! This took a little bit of Excel magic though. Getting the graph to look like the Yahoo graph was challenging. It was a fight with Excel to get the charts the way I wanted. The RSI chart underneath the main chart is just a second chart cropped nicely, and since there was no built-in way to add the threshold lines, I had to add a column filled with 30, 30, 30… and another column with 70s for all ~250 dates.

Yahoo's TSLA Graph (1 year):

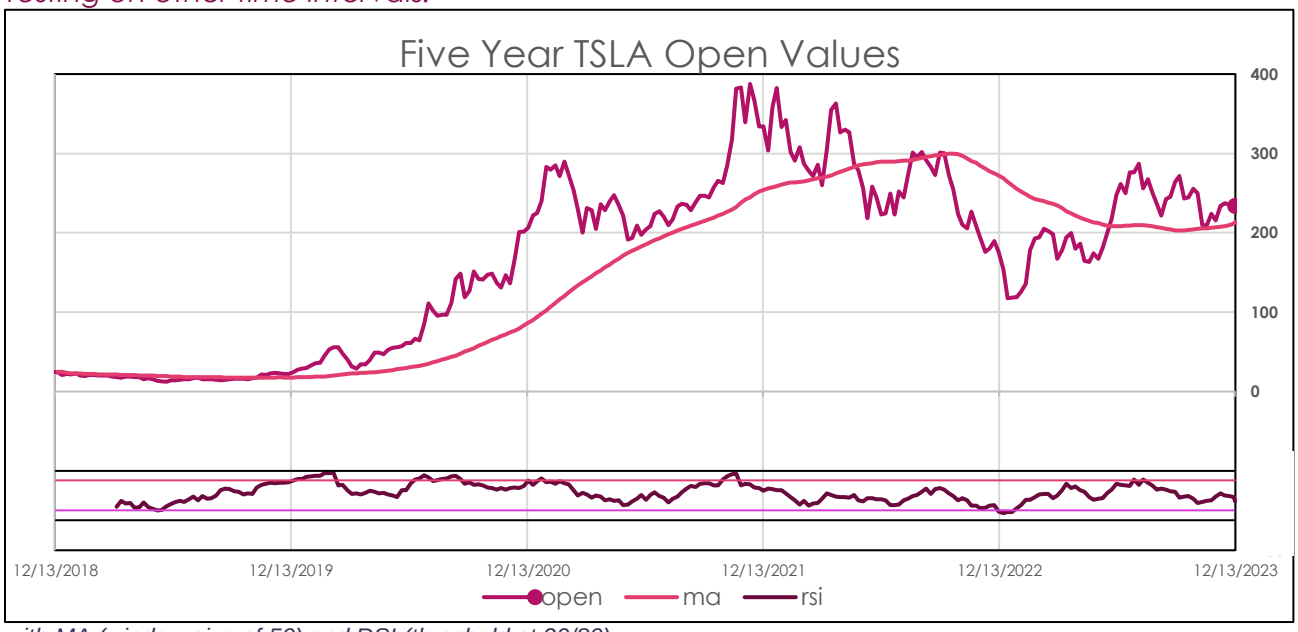
My TSLA Graph (1 year):



By the fifth hour of fighting with Excel, I got pretty good at making graphs. One caveat though: I couldn't make it clear that the RSI was on a separate scale than the rest of the graph. The RSI graph is good as a visual, but when I put the axis values there, it would be something like -50 to 1000, which is overkill and more confusing than to not have axis value labels at all. There also was no graceful way to put the threshold lines, so they were omitted.

The MA (orange line) on my graph does not match the real value's in the beginning (first 49 days) because it aims to use 50 days worth of open values for an average, or whatever is available. On the 50th day and beyond though, the MA line in my graph mimics the real line.

Testing on other time intervals:



with MA (window size of 50) and RSI (threshold at 20/80)

To test the RSI and MA capabilities further, I downloaded the 5 year data of TSLA on a weekly interval and put it through my stock bot initialization phase. I figured out how to make the graphs pretty, even if it's a silly work around. For instance, I used a white rectangle to cover things like negative axis values. Also, I basically just drew the lines on top of the RSI chart instead of plotting the 20/80 threshold with points. For comparison, here is the Yahoo chart of the same information:

Writing the Stock Bot

Long Hold Strategy

Overview

The long hold strategy is a traditional investment strategy based on the notion that, despite short-term market changes, the value of a well-chosen stock would rise over time. We commit a major amount of our assets to purchasing shares and holding them for an extended period, often a year or more, under this strategy. This strategy is frequently preferred due to its simplicity and ability to withstand market volatility.

Implementation

The `LongHoldStrategy` method in our `StockBot` class embodies this idea. On the first day of our investment period, we use the whole balance to buy as many shares at the opening price as possible. This decision is predicated on the belief that the long-term trend will be upward. To cash out, all shares are sold off at the end of the investing period, hopefully at a higher price. This technique is simple, but it is strongly dependent on the strength of the chosen stock (in this case, Intel, and Tesla (…yikes!)).

RSI and Moving Average Strategy

Overview

This slightly more sophisticated approach uses the moving average (MA) and estimated RSI values to make trading decisions. The idea is to buy stocks when they are seen to be undervalued (oversold) and selling them when they are thought to be overvalued (overbought). RSI is very helpful in determining these circumstances due to its momentum-based characteristic.

Implementation

The `rsiAndMaStrategy` method in the `StockBot` class executes this strategy. It considers the RSI and the stock's current price in relation to its moving average. If the RSI is below 30 (indicating oversold conditions) and the stock price is below the MA, the algorithm perceives it as a buying opportunity. Conversely, if the RSI is above 70 (indicating overbought conditions) and the stock price is above the MA, it suggests a selling opportunity. This dual-check system aims to avoid false signals that might be given by relying on just one indicator.

SI and Moving Average Strategy

Overview

The momentum-based trading strategy capitalizes on the continuation of existing trends in the market. It operates under the idea that stocks which have been doing well will continue to do so in the short-term, and vice versa for poorly performing stocks. This strategy aims to be effective in stocks exhibiting strong trends.

Implementation

In the `momentumAndVolumeStrategy` method, the algorithm looks for signals of strong momentum. It combines the analysis of moving averages with trading volumes – a high volume often means strong market interest and can reinforce the momentum signal. If the stock price is above its moving average and the trading volume is higher than the average, it's a confirmation of upward momentum and a signal to buy. Conversely, if the stock price is below the MA with high volume, it's considered a selling signal. This strategy, while potentially lucrative, can backfire, especially with unpredictable and volatile markets.

Testing the Algorithms

Final Net Worth of Bot (starting at $10,000) (in $)					
Companies:		**Intel**		**Tesla**	
Data Description:		Daily 1 Year	Weekly 5 Year	Daily 1 Year	Weekly 5 Year
Strategies	Long Hold	14,063.00	9,362.08	14,519.78	93,676.00
	RSI & MA	10,110.31	5,767.95	80.22	3,380.58
	Momentum-Based	9,979.07	7,950.89	9,621.11	54,393.52

Algorithm Reflections

For each strategy, I want to answer two questions: (a) How did it perform in the short run vs long run? (b) Why did it succeed/fail?

Long Hold Strategy Performance

The Long Hold strategy provided consistent positive returns across all periods, with Tesla showing a particularly high return over the 5 years (🚀to the moon!). This strategy is straightforward and involves little to no effort to manage, with significant returns when the stock has long-term growth. However, it does mean that the investor's capital is tied up and cannot be used for other opportunities.

The impressive returns, especially with Tesla, may be attributed to the company's significant growth over the past few years. The Long Hold strategy benefits from the overall upward trend of the market, reflecting the historical tendency of equities to increase in value over extended periods.

RSI & MA Strategy Performance

The performance of the RSI & MA strategy was mixed, yielding poor short-term results for Tesla but recovering in the long term. This method is based on indicator trends to improve trade timing. It becomes complicated since it needs to be constantly monitored, and will get messy when it encounters false signals, shown by Tesla's significant drop in one-year performance. The modest recovery over 5 years suggests that while the strategy can be effective, it may not be suitable for volatile stocks or over shorter time frames without adequate risk management strategies in place.

Tesla's significant loss in the short-term using the RSI & MA strategy could be due to the stock's high volatility and the method's sensitivity to sudden market movements. The recovery in the long term might be because the strategy had more time to capitalize on the overall upward trend of the stock despite short-term fluctuations.

Momentum-Based Strategy Performance

The Momentum-Based strategy had moderate success with Intel and showed strong long-term results with Tesla, although it did not outperform the Long Hold strategy in the short term. This strategy relies on reacting to market trends for quick gains but requires frequent trading. It also depends heavily on the continuation of market trends and can lead to significant losses if the momentum shifts unexpectedly.

The reason for the strong performance with Tesla over 5 years likely stems from the stock's clear upward trend and strong market interest, which the Momentum-Based strategy could exploit. In contrast, the more moderate performance with Intel and the short-term Tesla data may reflect less pronounced trends or more frequent shifts in momentum, which is where this strategy falls apart.

Reflection

The Long Hold strategy seems most reliable for investors who want consistent growth and are willing to commit capital over a longer period, especially in a stock like Tesla, which has shown significant growth. The RSI & MA strategy, while complex, has potential for gains but might require a longer timeframe to dull the effects of volatility and false signals, shown by the big initial losses with Tesla. The Momentum-Based strategy's performance shows that it has potential for substantial returns in the right market conditions with strong trends but may not be as reliable for stocks with less clear or less stable trends, like Intel in the short term.

In conclusion, the Long Hold strategy seems most suitable for those with a lower

risk appetite and patience for long-term gains, while the RSI & MA and Momentum-Based strategies could appeal to more active traders comfortable with higher risks and equipped with the means to manage them. In the end though, using a simple program to trade is probably unwise. Unless it's Tesla. 🚀🚀🚀🚀 (just kidding)