**MATLAB CSV Demo & Report – Project 2**

**Anthony D’Alessandro**

**Table of Contents**

Methods……………………………………………………………………………………………1

Plotter…………………………………………………………………………………...…1

Salter………………………………………………………………………………………1

Smoother…………………………………………………………………………………..2

Small Scale Trial…………………………………………………………………………………..2

Plotter……………………………………………………………………………………...2

Salter………………………………………………………………………………………3

Smoother Run 1…………………………………………………………………………...3

Smoother Run 2…………………………………………………………………………...4

Smoother Run 3…………………………………………………………………………...4

Larger Scale Trial………………………………………………………………………………….5

Plotter……………………………………………………………………………………...5

Salter………………………………………………………………………………………6

Smoother Run 1…………………………………………………………………………...6

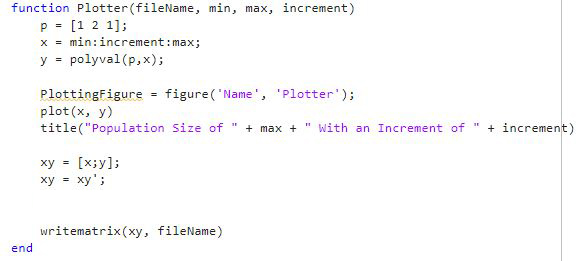
Smoother Run 2..………………………………………………………………………….7

Smoother Run 3..………………………………………………………………………….7

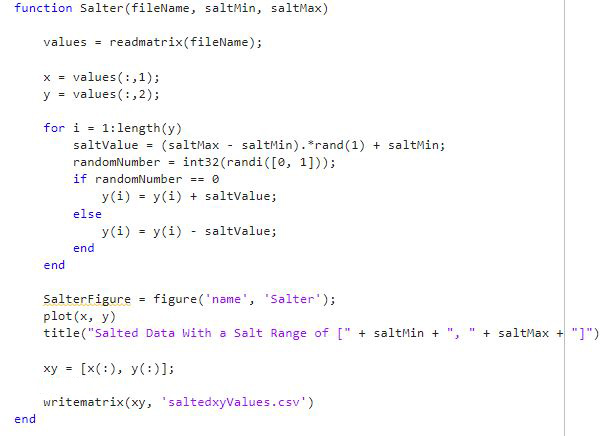
Discussion……………………………………………………………………………………....…8

**Methods**

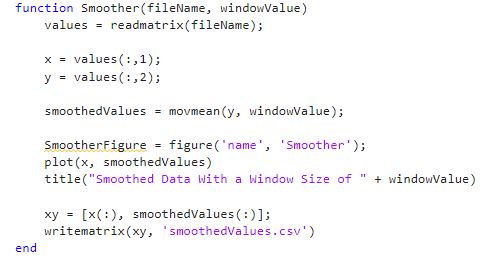
Plotter



Salter



Smoother



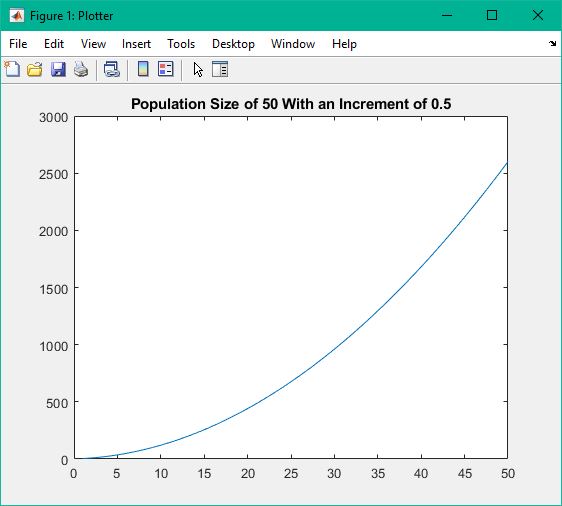
**Small Scale Trial**

Function Used:

Plotter

* Data generated using a population range from [1, 50] with an increment of 0.5

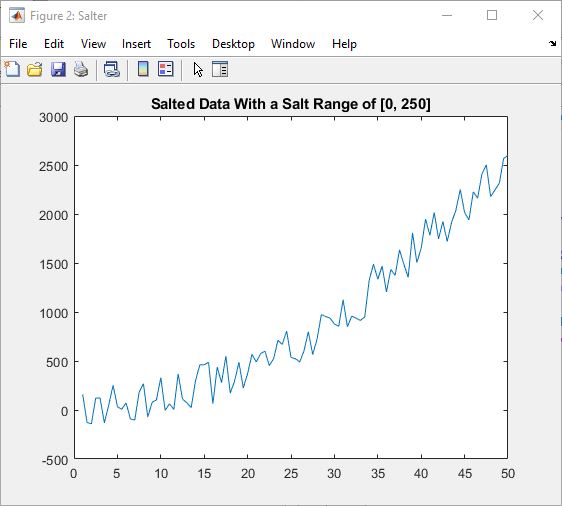




Salter

* Data salted with a salt range from [0, 250]

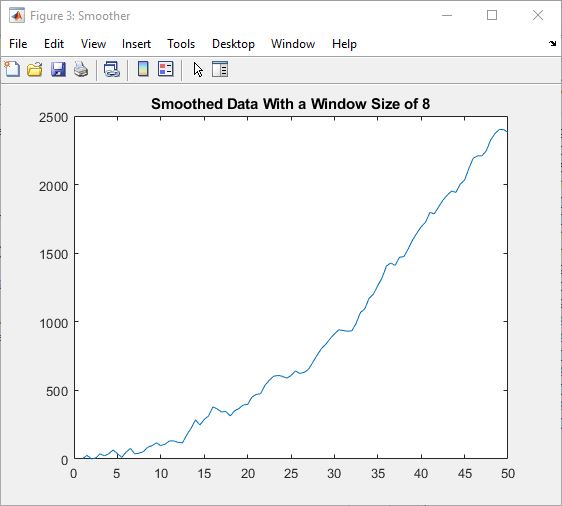




Smoother

* Data smoothed with a window size of 8

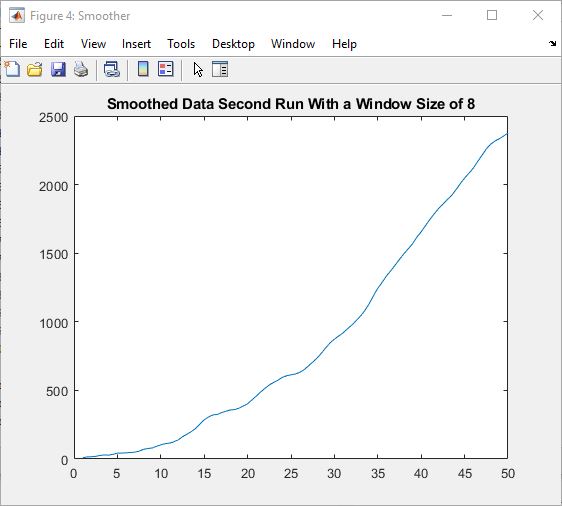




Smoother (Run 2)

* Smoother ran on first set of smoothed data with a window size of 8

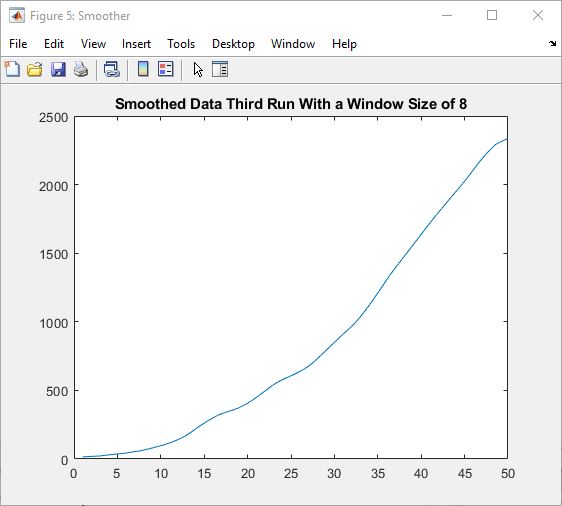




Smoother (Run 3)

* Smoother ran on second set of smoothed data with a window size of 8





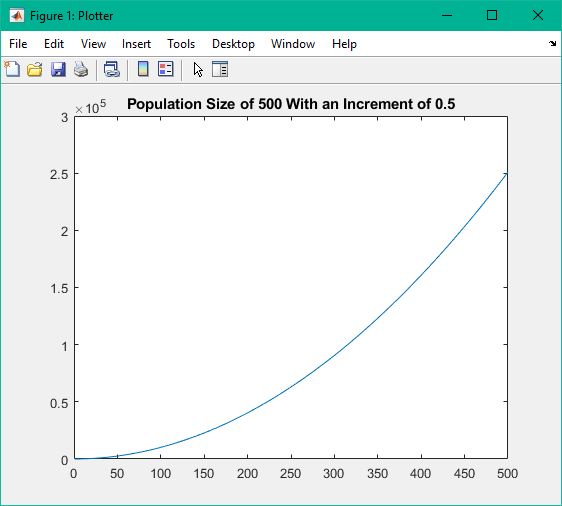
**Larger Scale Trial**

Function Used:

Plotter

* Data generated using a population range from [1, 500] with an increment of 0.5

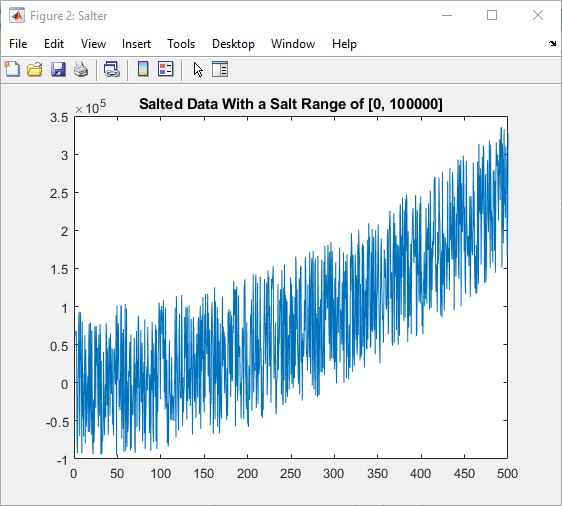




Salter

* Data salted with a salt range from [0, 100000]

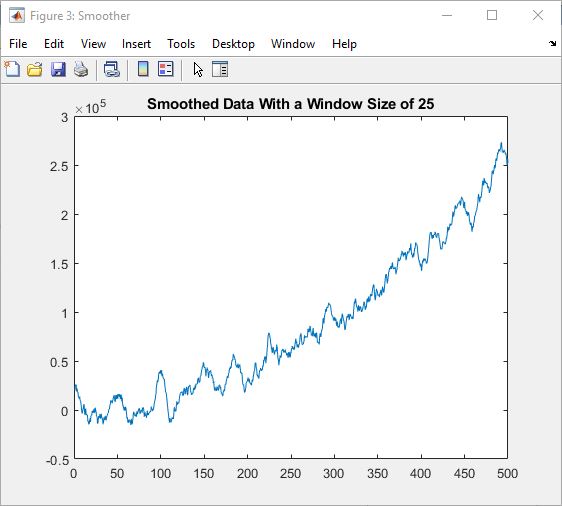




Smoother

* Data smoothed with a window size of 25

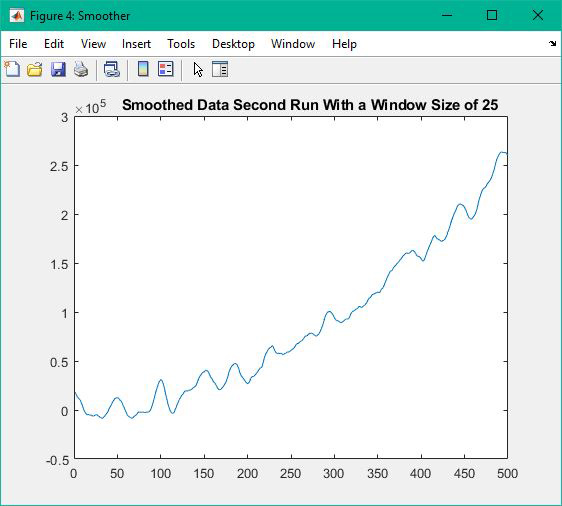




Smoother (Run 2)

* Smoother ran on first set of smoothed data with a window size of 25

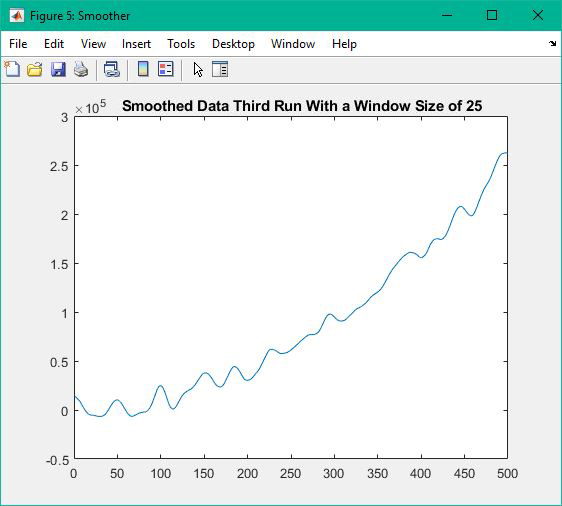




Smoother (Run 3)

* Smoother ran on second set of smoothed data with a window size of 25





**Discussion**

The main goals of this program were identical to the first program that was completed through Java where data was to be generated with some function such as , salted using a specified “salting” range, and then smoothed with a specified window value. This program was also designed to use CSV files to essentially transfer the data to and from each component. The main difference is that this program was written using MATLAB and the graphs that were generated for each data set were done automatically in real time rather than afterwards in an Excel file.

Program Structure

The program starts with the plotter function that takes as parameters a file name, minimum and maximum value for the range of x-values, and an interval for how much each x-value should be incremented by. A polynomial function is defined to represent the function . This is followed by creating a vector to hold the x-values starting at the minimum specified value and going up to the maximum specified value using the increment parameter. A vector of y-values is then created using the *polyval* function to evaluate the function at each of the x-values. Next, using a figure and the plot function, the x and y-values are graphed on a built-in MATLAB chart. Finally, the x and y-vectors are combined into one matrix, the matrix is transposed to have two columns, and the values are written to a CSV file using the *writematrix* function.

The next part of the program is the salter method that takes parameters including a file name and a minimum and maximum value for the range of the salt value. It starts by using the *readmatrix* function to store the x and y-values in a matrix. The matrix is then split into two column vectors of x-values and y-values respectively. Next, the program loops through the y-values, generates a random salt value within the indicated salt range, and generates a random integer of either 0 or 1. Depending on the random integer, the random salt value is either added to or subtracted from the current y-value. The program then graphs the x-values and salted y-values using the same functions as the plotter method. Finally, the salter method recombines the vector of x-values and the vector of y-values into one matrix and writes the values to a CSV file.

The final method is the smoother that takes two parameters, a file name and a window value. It starts just like the salter method where it reads the x and y-values from the CSV file of x and y-values, stores the data in a matrix, and then splits the matrix into two separate column vectors. The smoother method then uses the *movmean* function to calculate a running average for each y-value using the window value and stores the smoothed y-values in a new vector. Next, the program graphs the x and smoothed y-values in the same way as the plotter and salter methods. Finally, the x-values are combined with the smoothed y-values into one matrix and the data is written to a new CSV file.

Trials and Observations

In an effort to remain consistent across all of the programs, the MATLAB trials used the same polynomial function and data parameters as the Java CSV program. The first trial used a population with a range of 1 to 50 with an increment of 0.5 for a total of 100 x-value data points. The plotter method worked as expected and easily graphed the x and y-values in real-time. Using MATLAB to evaluate the polynomial function and plot the resulting data points was considerably easier as it only required 9 lines of code compared to the 22 lines in the Java CSV program. Following the plotter function was the salter function that also salted the data with a salt range of 0 to 250. The resulting graph showed the expected general ascending curve with random upward and downward spikes distributed along the line. This salter method was also easier to write compared to the Java CSV version in which reading and writing to and from the CSV files can be done in one line through MATLAB. However, the actual salting functionality was not much different between the two since the logic is the same and the real differences come from language syntax. The smoother method was then run on the salted data with a window value of 8 and the automatically generated graph displayed a curve that was definitely smoother and maintained an upward slope. Just like the Java CSV trials, the smoother was run two more times. The graph from the second run was noticeably smoother and the third run created an even smoother curve. The main differences between the graphs generated with this program and those in the Java CSV program are that the second and third smoothing runs seemed to make more of a difference and the curves tended not to flatten out as much towards the end.

The second trial was completed using a population with a range of 1 to 500 with an increment of 0.5 for a total of 1000 x-value data points. The graph that was generated was similar to the plotter graph in the first trial except it was scaled up with the highest point being about 250,000. Staying consistent with the Java CSV tests, the salter method in this trial used a salt range of 0 to 100000. The graph was similar to the salted graph from the first trial but the spikes were much more dramatic. The smoother method was then run on the salted data with a window value of 25 and produced a curve that had much smaller upward and downward spikes. While the smoother method seemed to work as intended, the graph was not nearly as smooth as the first smoother run in the larger scale Java CSV trial. This is most likely due to the difference in graphing methods between MATLAB and Excel in which MATLAB scaled the y-axis down by a factor of 105 to show more detail. In the Excel graphs, the y-axis scaled directly with the data where it started around 0 and went up to around 250,000. This produced more of a “big picture” where the data appeared to be much smoother than the MATLAB versions. The smoother method was run two more times in which the variability was reduced in each run but the graphs were not perfectly smooth like the original plotter curve. While the final result was not completely smooth, it made substantial progress from where it started directly after the data was salted.

Conclusions

The programming involved with the MATLAB version of this project mirrored the Java version very closely. One aspect of the program that was relatively confusing was how to manipulate the matrices and vectors that were storing the data points. However, after some research and testing, working with the matrices became much easier and allowed for some functionalities that would not be easily done with something like an ArrayList in Java. While the goals and outcomes of the program were almost identical to the Java version, the actual execution of the MATLAB version was much easier. This was due to the extensive built-in functionalities that substantially cut down on the amount of code that was required to read from and write to CSV files, represent algebraic equations, compute a rolling average, and display accurate graphs. By completing this program through MATLAB, it really highlighted not only the usability but the power of the platform and the endless applications it has for real-world problems.