PSS2 Report

Mouaz Ali

Table of Contents

Intro to Matlab

Matlab (Matrix Laboratory) stands out as a high-level programming language developed by MathWorks, allowing programmers to work efficiently with matrices and arrays. This paper delves into the functionalities of Matlab, its similarities with other programming languages, its advanced features such as plotting and polynomial manipulation, and its widespread applications across various industries. Matlab is well-known for its abilities in data analysis, visualization, and numerical calculation. Matlab is now considered essential in a wide range of domains, from science and engineering to finance and higher education, thanks to its extensive function library and user-friendly interface.

With its specific tools for linear algebra, non-linear functions, statistics, calculus, curve fitting, and other areas, Matlab is a master in computational mathematics. Matlab and object-oriented programming languages like Java are comparable in that they both have loops, conditional statements, variables, data types, and operators. Matlab is known for its dynamic typing, which is similar to Python in that variable characteristics are determined at runtime. For users who are already familiar with traditional programming languages, this feature makes the learning curve simpler.

Advanced features in Matlab include graphics, plotting, and manipulating polynomials. Plotting allows users to easily view data; commands like plot() and bar() make it easy to create basic graphs and bar charts. Three-dimensional plots can be visualized using meshgrid() and surf(), which provide access to surface graphs. Furthermore, Matlab provides powerful functions for manipulating polynomials, including polyval(), polyvalm(), poly(), and polyfit(). These functions make it easier to evaluate polynomials, perform matrix polynomial operations, find roots, and fit polynomial curves. It is essential in industries including academic research, financial modeling, automotive design, aerospace engineering, and automotive design because of its capacity to manage complex mathematical calculations and produce illuminating representations. Matlab is essential to current engineering and technology, as demonstrated by the use of the program in mission-critical jobs by companies such as NASA.

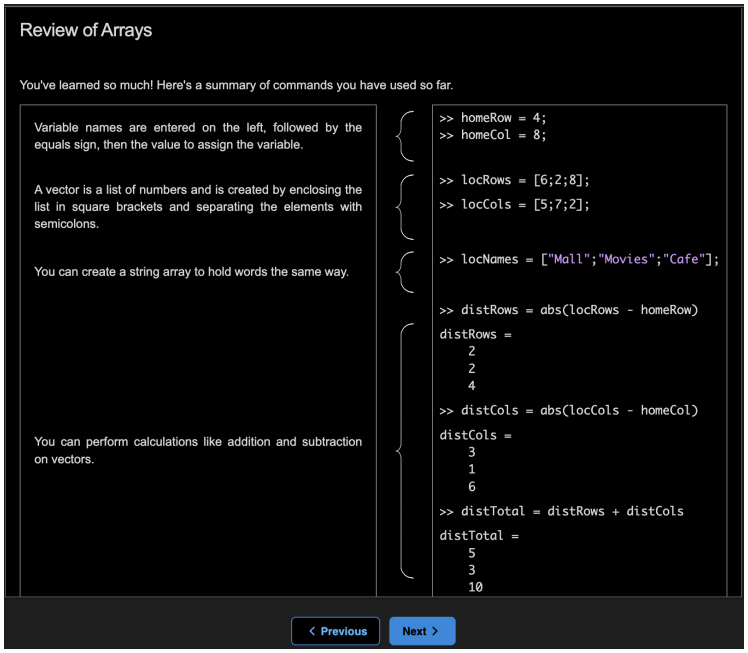
Users learn important lessons about algorithm building, data analysis methods, and computational problem-solving through practical Matlab experience. Workflows are streamlined and productivity is increased when numerical computations, data manipulation, and result visualization are all done with ease in one environment. Furthermore, having a working knowledge of Matlab offers up a wide range of job options in fields where data analysis and computational mathematics are essential.

Tutorial

After searching for many different tutorials online, I found Learn to Code with Matlab to be the most helpful. It was amazing how easy computing numbers were made compared to other programming languages like Java. The tutorial provided videos as well as hand-on coding exercises with Matlab. There were different sections relating to various functions used in the language. The first section of the course provides an overview of Matlab's features, emphasizing its strengths in scientific computing, data visualization, and computational mathematics. It highlights how Matlab's strong built-in functions and simple syntax make it a great option for beginners getting into programming.

Throughout the tutorial, learners are introduced to basic programming constructs such as variables, operators, loops, and conditional statements. I got to experiment with matrices and arrays, data structure, and how to manipulate them to perform various computational tasks. One

of the key features of Matlab covered in the tutorial is plotting and graphics. Users discover how to create different types of plots, including line plots, scatter plots, bar charts, and three-dimensional plots, allowing them to visualize data and analyze trends.



Plotter

The plotter class provides functionality to generate data points for quadratic function $y = x^2 - 4x + 3$ over a specified range of x values and writes these data points to a CSV file.

Here's a description of what the Plotter class consists of:

1. Method:
   - Class includes a method named “Plotter”
2. Input Parameters: The method consists of four parameters
   - filename: A string representing the name of the CSV file to be generated.
   - min: The minimum value of x for the range of data points.
   - max: The maximum value of x for the range of data points.
   - increment: The interval between consecutive x values.
3. Data Plotting:
   - Calculates a vector of x values ranging from min to max with the specified increment.
   - Based on the quadratic function, it computes the corresponding y values for each x value.
4. Plotting:
   - Creates a new plotting figure and plots the generated data points.
   - Title of plot indicates range of x values
5. CSV File Output:
   - Combines the x and y values into a matrix and writes this data matrix to a CSV file with the specified fileName.

Full Plotter Class Code:

```
function Plotter(fileName, min, max, increment)
    % Define the polynomial function: y = x^2 - 4x + 3
    p = [1 -4 3];

    % Generate x values from min to max with given increment
    x = min:increment:max;

    % Calculate corresponding y values based on the polynomial function
    y = polyval(p, x);

    % Create a new plotting figure
    PlottingFigure = figure('Name', 'Plotter');

    % Plot the function using x as x-axis values and y as y-axis values
    plot(x, y)

    % Plot title
    title({'Plotted Data', ['(Range: [' num2str(min) ', ' num2str(max) '])']});

    % Put data into matrix
    xy = [x;y];

    % Arrange x and y values in columns
    xy = xy';

    % Write data to CSV file
    writematrix(xy, fileName)
end
```

Salter

The salter class adds random salt to the y-values of data read from a CSV file and then plotting the modified data.

Some of the functionalities of the salter class include:

1. Method:
    - Class includes a method named “Salter”
2. Input Parameters: The method takes three parameters
    - fileName: A string representing the name of the input CSV file containing the data.
    - saltMin: The minimum value of the random salt to be added or subtracted.
    - saltMax: The maximum value of the random noise to be added or subtracted.
3. Data Salting:
    - Reads the data from the specified CSV file and stores it in a matrix.
    - Splits the matrix into two vectors: x for the x-values and y for the y-values.
    - For each y-value, it generates a random salt value within the specified range and adds or subtracts it randomly.
4. Plotting:
    - Creates a new plotting figure and plots the modified data.
    - Title of plot indicates range of salt values
5. CSV File Output:
    - Writes the data to a CSV file named "SaltedValues.csv"

Full Salter Class Code:

```
function Salter(fileName, saltMin, saltMax)
    % Read data from CSV file and store it in matrix
    values = readmatrix(fileName);

    % Split the matrix
    x = values(:,1);
    y = values(:,2);

    % Add/subtract random salt value
    for i = 1:length(y)
        % Generate random salt within specified range
        saltValue = (saltMax - saltMin).*rand(1) + saltMin;

        % Generate random integer
        randomNumber = int32(randi([0, 1]));

        % Add/subtract salt value based on the random number
        if randomNumber == 0
            y(i) = y(i) + saltValue;
        else
            y(i) = y(i) - saltValue;
        end
    end

    % Create new plot figure
    SalterFigure = figure('name', 'Salter');

    % Plot modified data
    plot(x, y)

    % Plot title
    title({'Salted Data', ['(Salt Range: [' num2str(saltMin) ', ' num2str(saltMax) '])']});

    % Put data into matrix
    xy = [x(:), y(:)];

    % Write data to CSV file
    writematrix(xy, 'SaltedValues.csv')
end
```

Smoother

The Smoother class performs smoothing on data read from a CSV file using a moving average filter and then plots the smoothed data.

Here is a breakdown of the components in this class:

1. Method:
    - Class includes a method named “Smoother”
2. Input Parameters: This method takes three parameters
    - fileName: A string representing the name of the input CSV file containing the data.
    - windowValue: An integer specifying the size of the moving average window for smoothing.
    - outputFileName: A string representing the name of the output CSV file for the smoothed data. If not provided, a default output file name, "SmoothedValues.csv," is used.
3. Data Smoothing:
    - Reads the data from the specified CSV file and stores it in a matrix.
    - Splits the matrix into two vectors: x for the x-values and y for the y-values.
    - Calculates the moving averages for the y-values using the specified window size.
4. Plotting:
    - Creates a new plotting figure and plots the smoothed data.
    - Title of plot indicates window size used for smoothing
5. CSV File Output:
    - Writes the data to a CSV file.

Full Smoother Class Code:

```
function Smoother(fileName, windowValue, outputFileName)
    % Check if the output file name is provided
    if nargin < 3
        % If not, set default output file name
        outputFileName = 'SmoothedValues.csv';
    end

    % Read data from CSV file and store it in a matrix
    values = csvread(fileName);

    % Split the matrix
    x = values(:,1);
    y = values(:,2);

    % Calculate moving averages for y values
    smoothedValues = movmean(y, windowValue);

    % Create new plot figure
    SmootherFigure = figure('name', 'Smoother');

    % Plot smoothed data
    plot(x, smoothedValues)

    % Plot title
    title({'Smoothed Data', ['(Window Size: ' num2str(windowValue) ')']});

    % Put data into matrix
    xy = [x(:), smoothedValues(:)];

    % Write data to CSV file
    writematrix(xy, outputFileName);
end
```

CSVTester

CSVTester script tests the Plotter, Salter, and Smoother functions by generating data points, salting them, and then smoothing them.

Here's what each function call does:

1. Plotter(PlottedValues.csv, -10, 10, 0.1):
    - Generates data points for a quadratic function $y = x^2 - 4x + 3$ over the range of x-values from -10 to 10 with an increment of 0.1. It then writes these data points to a CSV file named PlottedValues.csv.
2. Salter(PlottedValues.csv, 0, 250):
    - Reads data from the CSV file PlottedValues.csv, adds random salt values to the y-values within the specified range, plots the modified data, and writes the modified data to a new CSV file named SaltedValues.csv.
3. Smoother(SaltedValues.csv, 8, SmoothedValues1.csv):
    - Reads data from the CSV file SaltedValues.csv, applies a moving average smoothing algorithm with a specified window size to the y-values, plots the smoothed data, and writes the smoothed data to a new CSV file named SmoothedValues1.csv.
4. Smoother(SmoothedValues1.csv, 8, SmoothedValues2.csv):
    - Applies smoothing algorithm to SmoothedValues1.csv and writes the data to a new CSV file named SmoothedValues2.csv.
5. Smoother(SmoothedValues2.csv, 8, SmoothedValues3.csv):

- Applies smoothing algorithm to SmoothedValues2.csv and writes the data to a new CSV file named SmoothedValues3.csv.

Full CVSTester Class Code:

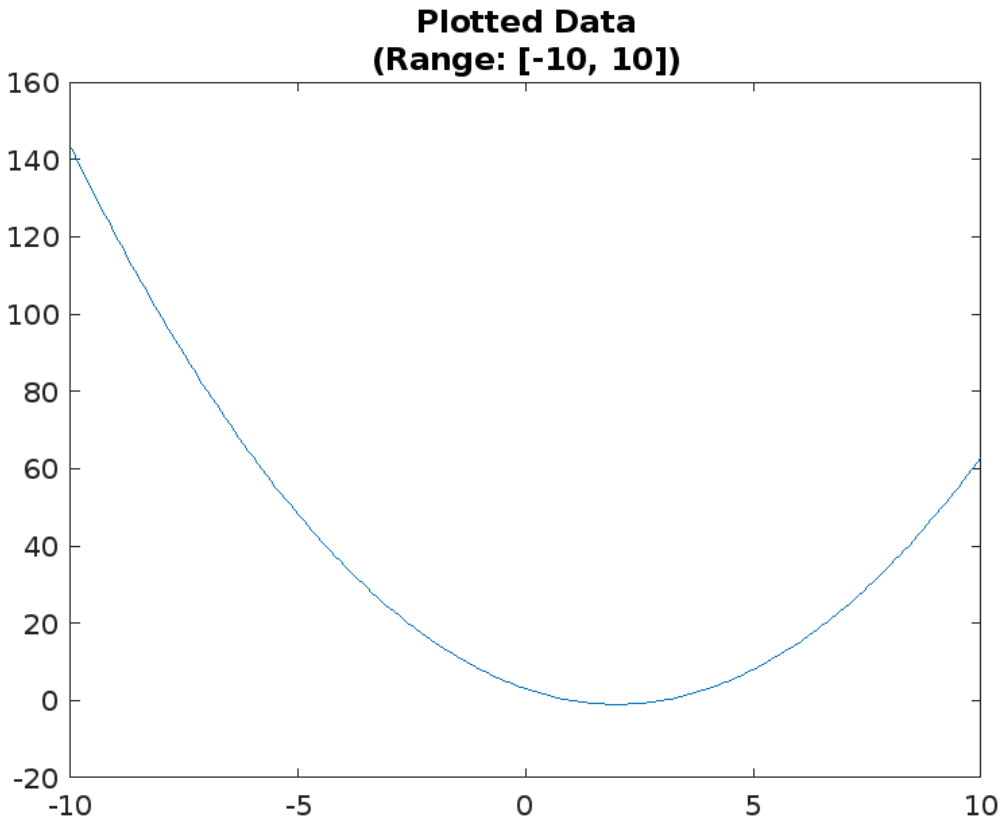
```
% Call the Plotter function
Plotter('PlottedValues.csv', -10, 10, 0.1)

% Call the Salter function
Salter('PlottedValues.csv', 0, 250)

% Call the Smoother function
Smoother('SaltedValues.csv', 8, 'SmoothedValues1.csv')

% Call the Smoother function to smooth "SmoothedValues1.csv" and generate "SmoothedValues2.csv"
Smoother('SmoothedValues1.csv', 8, 'SmoothedValues2.csv');

% Call the Smoother function to smooth "SmoothedValues2.csv" and generate "SmoothedValues3.csv"
Smoother('SmoothedValues2.csv', 8, 'SmoothedValues3.csv');
```

Results - Small Samples

Plotter:

- Data generated using a population range [-10, 10] with an increment of 0.1

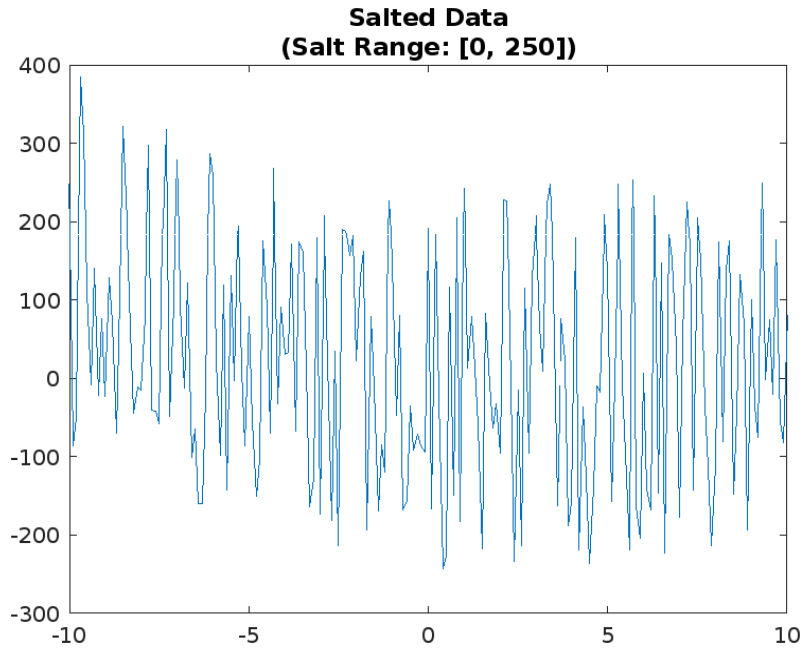
```
% Call the Plotter function
Plotter('PlottedValues.csv', -10, 10, 0.1)
```



Salter

- Data salted with salt range [0, 250]

```
% Call the Salter function
Salter('PlottedValues.csv', 0, 250)
```

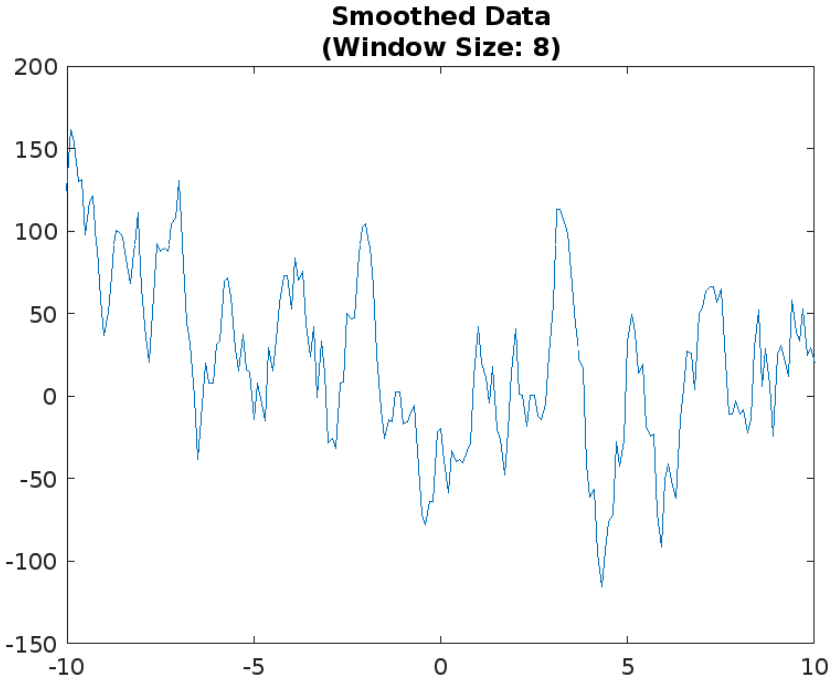


Smoother

First Run

- Data smoothed with a window size of 8

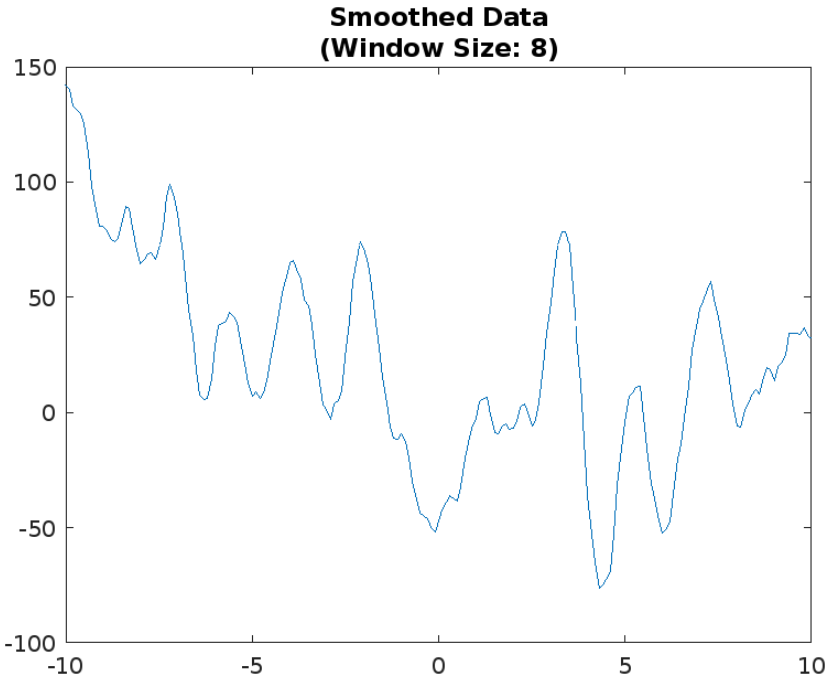
```
% Call the Smoother function
Smoother('SaltedValues.csv', 8, 'SmoothedValues1.csv')
```



Second Run

- Smoother ran on first set of smoothed values

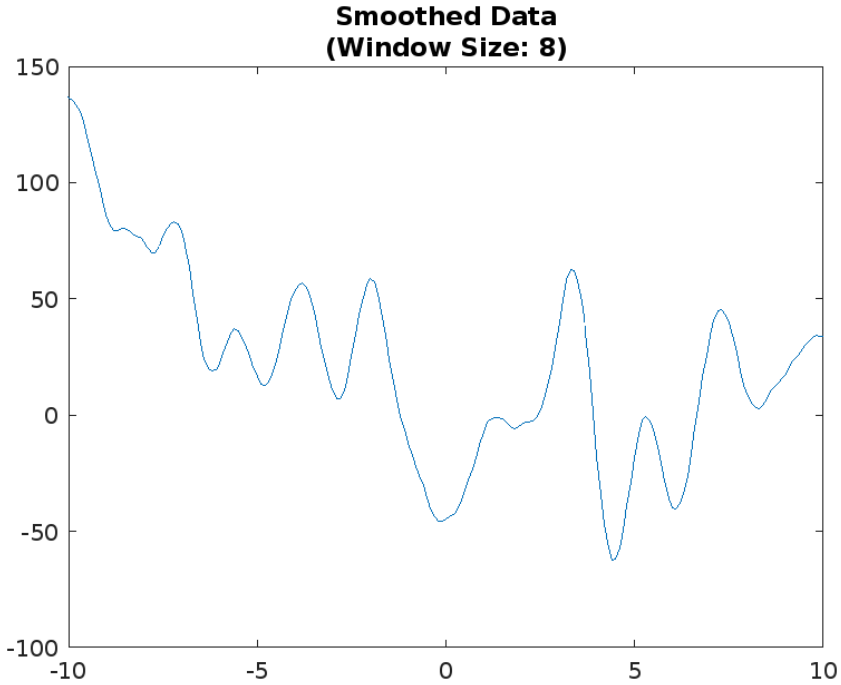
```
% Call the Smoother function to smooth "SmoothedValues1.csv" and generate "SmoothedValues2.csv"
Smoother('SmoothedValues1.csv', 8, 'SmoothedValues2.csv');
```



Third Run

- Smoother ran on the second set of smoothed values

```
% Call the Smoother function to smooth "SmoothedValues2.csv" and generate "SmoothedValues3.csv"
Smoother('SmoothedValues2.csv', 8, 'SmoothedValues3.csv');
```

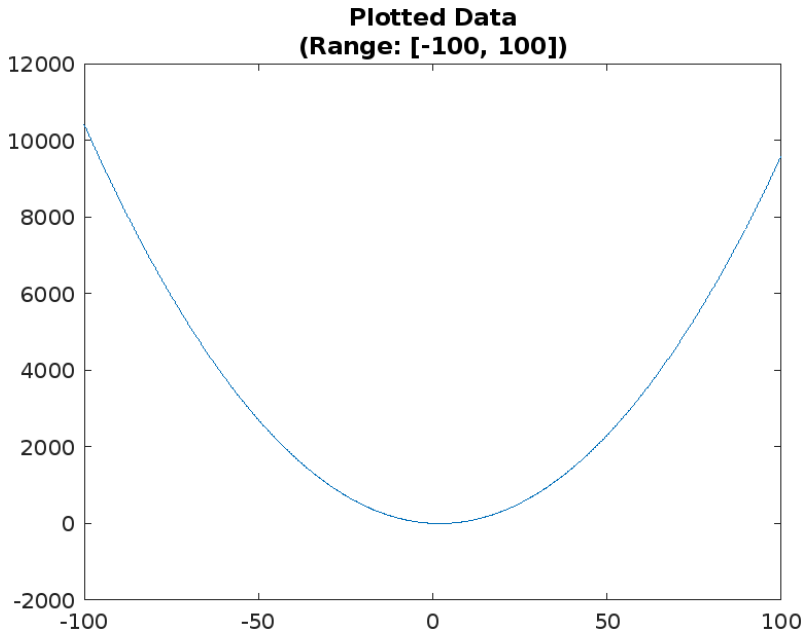


Results - Large Samples

Plotter:

- Data generated using a population range [-100, 100] with an increment of 0.2

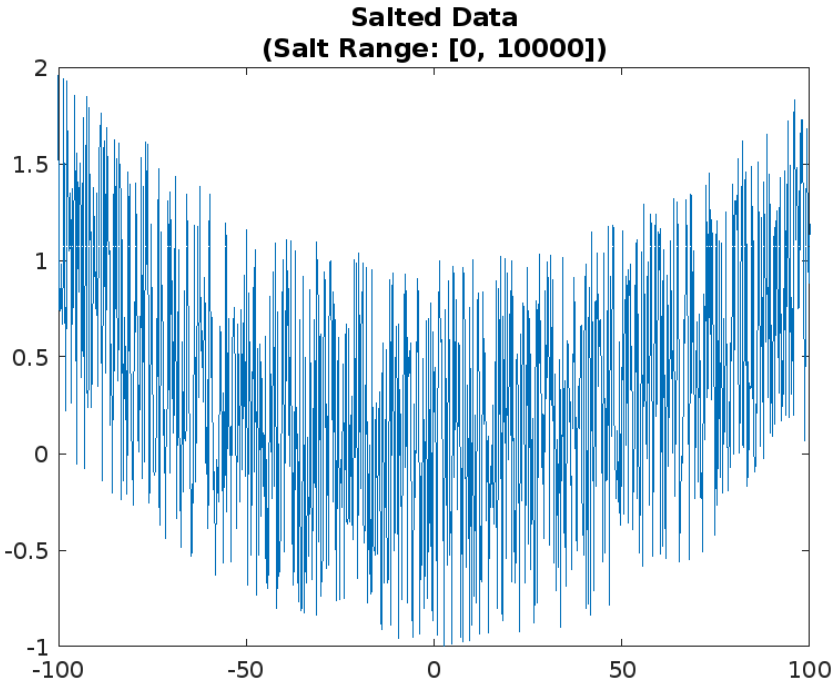
```
% Call the Plotter function
Plotter('PlottedValues.csv', -100, 100, 0.2)
```



Salter:

- Data salted with salt range [0, 10000]

```
% Call the Salter function
Salter('PlottedValues.csv', 0, 10000)
```

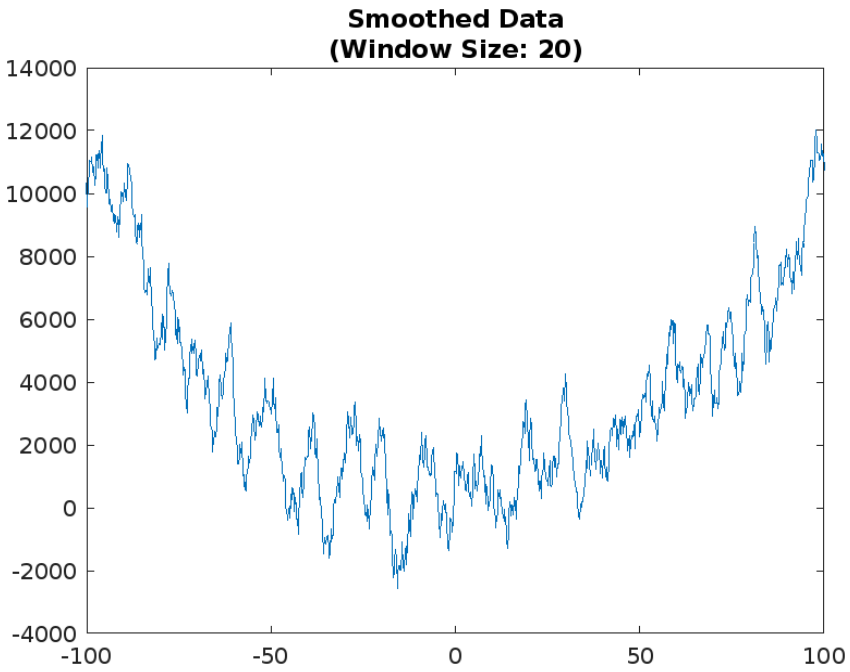


Smoother:

First Run

- Data smoothed with a window size of 20

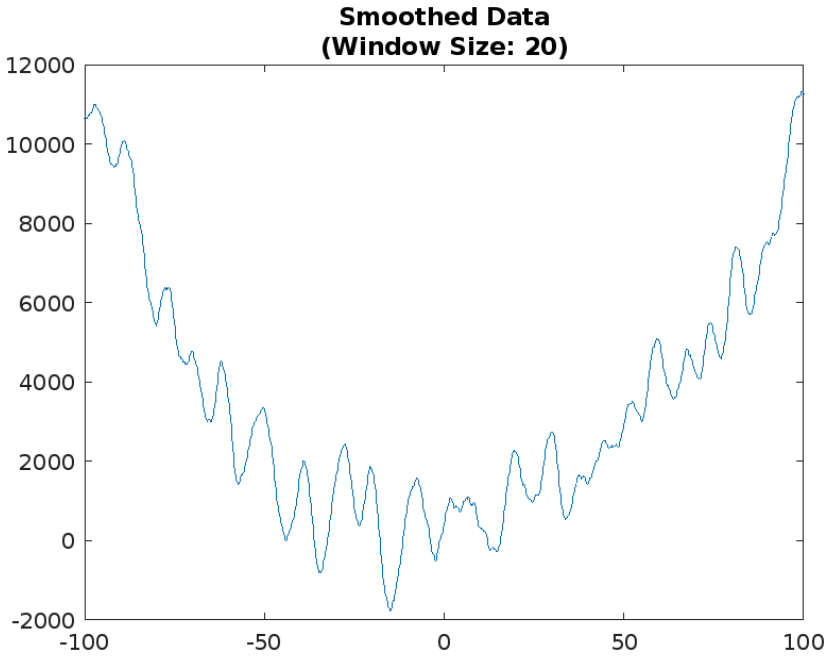
```
% Call the Smoother function
Smoother('SaltedValues.csv', 20, 'SmoothedValues1.csv')
```



Second Run

- Smoother ran on first set of smoothed values

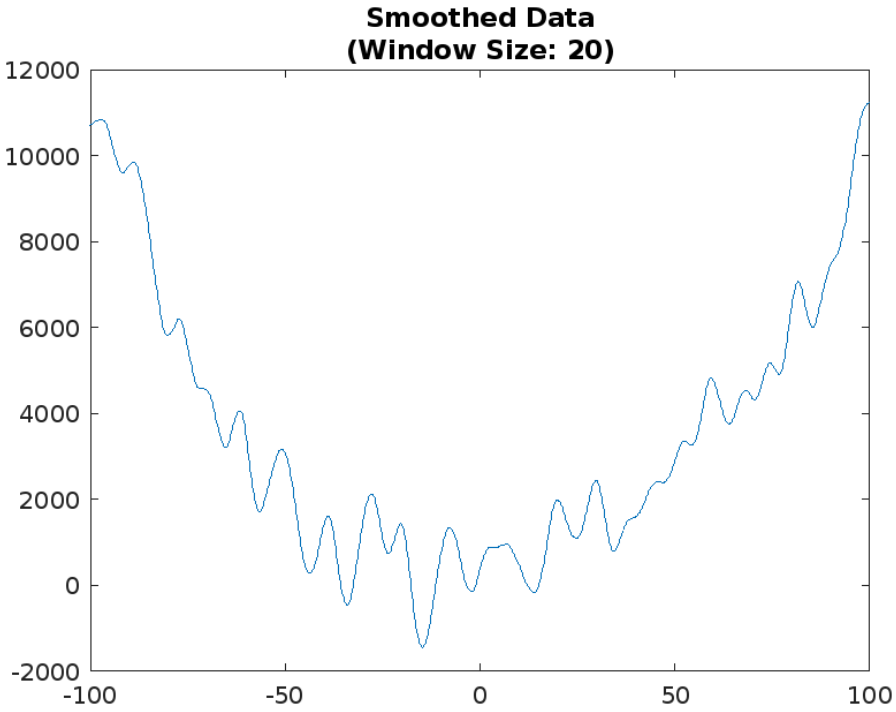
```
% Call the Smoother function to smooth "SmoothedValues1.csv" and generate "SmoothedValues2.csv"
Smoother('SmoothedValues1.csv', 20, 'SmoothedValues2.csv');
```



Third Run

- Smoother ran on the second set of smoothed values

```
% Call the Smoother function to smooth "SmoothedValues2.csv" and generate "SmoothedValues3.csv"
Smoother('SmoothedValues2.csv', 20, 'SmoothedValues3.csv');
```



Discussion

This program's primary goal was to generate data according to a given function, salted within a particular range, and smoothed using a defined window value in Matlab. Although similar to the first part where Java was used to generate CSV files and then Excel for data manipulation and visualization, the difference in this program is that Matlab facilitated the real-time plotting of graphs within the application, hence simplifying the process. The Java application and the Matlab trials both employed the same polynomial function, $y = x^2 - 4x + 3$, and data parameters, in an attempt to maintain consistency between the programs.

The first set of trials consisted of small samples of data, where a population ranging from -10 to 10 was used with an increment of 0.1. The Plotter method efficiently generated and plotted these data points, demonstrating the ease of Matlab for mathematical computations and plotting. The Salter method then introduced random salt values within the specified range (0 to 250), producing a graph with noticeable noise along the curve. The smoother method was then applied to the generated salted dataset with a window value of 8. As described above, three different smoothing statements were used in the testing method, the first smoothing the salted values, the second smoothing the first smoothed file, and the third smoothing the second smoothed data. Once run, the program automatically generated five total graphs: plotted graph, salted graph, and three smoothed graphs. The first smoothed graph looked smoother and more clear, compared to

the graph with salt values. The second smoothed graph was in even better shape. Lastly, the third smoothed graph was the most readable out of all three.

The second set of trials consisted of larger data, with a population range of -100 to 100 in increments of 0.2. The plotted data graph was actually quite similar to the one in the smaller trial, just a bit more complete, as there were more data points and a wider x range. Since the salter method consisted of a range from 0 to 10000, the graph was a lot more jumbled. The data looked all over the place, although still maintaining a mostly consistent slope. When the smoother method was run with a window size of 20, the graph was already starting to look a lot easier to read. Over the next two smoothing runs, the slope of the graph looked pretty similar to the original plotted graph, even though there were still some irregularities. The graph was not nearly as smooth as the initial smoother run in the Java sample, even though the smoother method appeared to function as expected. The reason for this might be the difference in data graphing between Matlab and Excel. To show more detail, Matlab scaled the y-axis down. But in the Excel graphs, the y-axis was scaled exactly with the data, which generated more accurate results, from my perspective.

This program could be modified in several ways for efficiency. Input validation and error handling could be implemented to improve the Plotter function. Additionally, allowing customization of the polynomial function and plot attributes can enhance flexibility and visualization options. For the Salter function, offering different salt distribution options, such as parameter flexibility, can provide users with more control and insight into the data graphing process. Last but not least, the Smoother function can be made more versatile, especially when handling a variety of datasets and real-time applications, by adding support for multiple smoothing methods. All of these improvements can potentially help to make the program more flexible, easy to use, and able to handle a larger variety of requirements.

In summary, this project has demonstrated the effectiveness and efficiency of Matlab in data manipulation and graphing. By replicating a previous Java-based project, I was able to explore the plotting, salting, and smoothing of data points using the application's built-in functionalities. The consistent use of the quadratic function ensured comparability between the two implementations. Programming in Matlab definitely seemed more easier, though, compared to Java, as Matlab's real-time graph plotting capabilities streamlined the workflow and eliminated the need for external software. As a whole, this project demonstrates the flexibility and strength of Matlab as a platform for data visualization. Its many built-in functions and ease of use make it a priceless tool for analysts, engineers, and researchers in a variety of fields. Future development of Matlab's features and interaction with other tools may reveal even more promise for solving practical issues.