Plotting, Salting, and Smoothing

Ian Mays

# Abstract

Salting is a technique used to anonymize data by adding random factors to individual data points so that the value for the individual point can not be determined. Instead, the trends in the data are only visible in aggregate. One way to do this is by smoothing the data, taking the average of all the points within a certain range of a given point and using that as the new value for it. This report explores the plotting, salting, and smoothing of 1,000 points of the function on the interval from -5 to 5 using three different utilities, those being standard Java code, MATLAB, and the Java libraries JFreeChart and Apache Stats Library.

Table of Contents

[Abstract 1](#_Toc153306886)

[Plotting, Salting, and Smoothing in Java 2](#_Toc153306887)

[Plotting 2](#_Toc153306888)

[Salting 2](#_Toc153306889)

[Smoothing 3](#_Toc153306890)

[Plotting, Salting, and Smoothing in MATLAB 5](#_Toc153306891)

[Getting Started with MATLAB 5](#_Toc153306892)

[Desktop Basics 5](#_Toc153306893)

[Matrices and Arrays 6](#_Toc153306894)

[Array Indexing 6](#_Toc153306895)

[Text and Characters 6](#_Toc153306896)

[Calling Functions 6](#_Toc153306897)

[2-D and 3-D Plots 6](#_Toc153306898)

[Programming and Scripts 7](#_Toc153306899)

[Plotting 7](#_Toc153306900)

[Salting 7](#_Toc153306901)

[Smoothing 8](#_Toc153306902)

[Plotting, Salting, and Smoothing with Java Libraries 8](#_Toc153306903)

[Plotting 9](#_Toc153306904)

[Salting 10](#_Toc153306905)

[Smoothing 10](#_Toc153306906)

# Plotting, Salting, and Smoothing in Java

First, we will discuss a manual implementation of the plotting, salting, and smoothing algorithms in Java. WIth the exception of ArrayLists and Lists during the smoothing step, all imports in this implementation are related to input and output, and are included as part of built-in Java packages. The function, meanwhile, is included in the Math class and so does not need to be imported.

## Plotting

The plotting portion of the program is extremely simple: it finds the value of at each point, and then prints the x and y values to a CSV file. This can then be graphed as a scatter plot in Excel as shown above. Notice that this function has a range of -1 to 1, as itself can never return a value outside of this range regardless of the input.

## Salting

The salting portion of the program takes the CSV file generated by the previous step as input, and increases each point’s value by a random number between and , where is the range. For , this produces a plot where the overall trend is still easily visible, but the value of each individual point is much more difficult to see. What happens when we increase to 10?

Now it is hard to spot any sort of trend in this data at a glance. We will see later whether this data can still be smoothed to produce a recognizable result, but first we will return to the graph.

## Smoothing

Smoothing works by finding a moving average of a range of points surrounding a given point in the data. In this case, , so each point looks at its 10 nearest neighbors in each direction. For points near the edges, the bounds of the moving average window are instead set to the respective edge of the data. Once again, the CSV data from the salting step is used as input, and the output is placed in a new CSV file which can be used for graphing.

One notable artifact from the salting/smoothing process is that the curve appears to be much more jagged at peaks and troughs than in the original function. This is most noticeable at the origin, which features a wide trough. Nonetheless, the smoothing algorithm for the most part does a very good job of reproducing the original curve in this case.

For , however, smoothing is not sufficient to restore the original curve. Will work?

Unfortunately, no. Although increasing the moving average window does smooth out more of the randomness, it has the drawback of including points with significantly different values in the original plotting step, thus influencing the moving average. However, we can obtain the best of both worlds here simply by increasing the number of points in our plot.

With 10,000 points instead of 1,000, we can now see some of the original structure of the curve. Although still far from perfect, we can make out a wide trough in the middle with increasingly narrow peaks and valleys in each direction.

# Plotting, Salting, and Smoothing in MATLAB

MATLAB is a popular program for numeric computing which provides an alternative medium through which to implement plotting, salting, and smoothing. Although the program is paid in general, the online web app can be used for free for up to 20 hours per month.

## Getting Started with MATLAB

Before I could begin plotting, salting, and smoothing with MATLAB, I first had to learn how to use the program, so I followed the “Getting Started” tutorial, which is linked when the web app is first loaded. I noticed that all MATLAB files are stored on the MATLAB cloud drive by default, but I discovered that right clicking on them would allow me to download them locally, allowing me to add them to my repository.

### Desktop Basics

I began following along with the online tutorial by entering basic commands for defining variables. This is done simply by typing or a similar expression into the command line.

Having previously used the similar program Maple, where variable assignment requires the use of the syntax, I appreciated the fact that this syntax was much more intuitive. Even still, I often forget to add the colon when programming in Maple, resulting in confusion when my code fails to run properly.

One downside that I noticed, however, was that the output for each assignment takes up several lines in the command line window. Thankfully, the tutorial immediately demonstrated to me that this can be circumvented by adding a semicolon to the end of the expression.

### Matrices and Arrays

An important note about MATLAB is that its name is short for “matrix laboratory”, so it is primarily designed with matrices in mind. All variables are technically matrices: single-value variables are 1x1 matrices, while arrays of length are 1x row vectors.

With this knowledge, I learned the syntax for creating matrices. When defining a matrix, the whole matrix is surrounded by square brackets, rows are separated by semicolons, and individual elements are separated by spaces.

The tutorial then goes on to cover several specialized functions that can be used to generate certain types of matrices, as well as ones covering the basic matrix operations. Some of these are just elementwise operations, meaning they simply apply a function to each element independent of the others. Others, however, are specific to matrices, including transposition, inversion, matrix multiplication, and concatenation. Notably, this section mentions a function which piqued my attention: one that generates a matrix of a given size filled with random numbers. I noted this for later, as this would likely be useful for the salting step of my program.

### Array Indexing

Accessing the elements of a matrix after it is created can be done by placing the index subscript in parentheses. This can either be two numbers, in the case of a specific row and column, or a single number, which traverses the matrix first top-to-bottom followed by left-to-right. Using a start and end index separated by colons allows for the retrieval of a range of values in either or both dimensions.

In MATLAB, arrays index starting from 1 instead of 0, which is somewhat unusual for programming languages but not unprecedented. However, I figured that as long as I changed my array indices appropriately, this would not have a major impact on the final output.

### Text and Characters

The syntax for specifying strings in MATLAB turns out to be nearly identical to that of Java, so I did not learn much from this section. However, this section also reveals that the command for retrieving the metadata for a variable is named “whos”, which I found amusing.

### Calling Functions

Like with text, the syntax for calling functions is mostly the same as in Java, but one notable exception is the existence of functions with multiple return values, which can be handled by placing an array on the left side of the equals sign. This syntax reminded me of Python, which features the ability to use tuples in the same manner.

### 2-D and 3-D Plots

In MATLAB, 2D and 3D plots are rendered using the “plot” and “surf” commands respectively. Specifying bounds for the axes is done by defining a linear space with a start and end bound. Interestingly, however, the commands for labeling the various elements of the plot are placed after the “plot” command itself. I realized I would need to keep this syntax in mind, as I was more accustomed to Python’s “pyplot” module where the opposite is true.

### Programming and Scripts

At this point in the tutorial, I did find myself somewhat frustrated with coding through the command line, as there is no semblance of an undo button. Even starting over from scratch is a tedious process here: I had to manually clear the command line, manually clear the workspace in a separate action to avoid previous variable assignments interfering with the new code, and then manually retype all of my code hoping I do not make another mistake. This not only made it more tedious to correct mistakes in my code, but it also made it more difficult to experiment with said code, both of which are crucial when learning how to use a new program.

Thankfully, the tutorial now showed me how to create and run a script, allowing me to write a proper program which can be executed from the command line simply by typing its name. Although the workspace is not cleared by default, this can be remedied by adding “clear all;” to the beginning of the script. However, I would have appreciated if this section were placed closer to the beginning of the tutorial.

## Plotting

Although the “Getting Started” tutorial does not cover a lot of crucial features that are needed to write a functional program in a given language, the documentation does cover them. In particular, the plotting portion of this program required learning how to define custom functions and how to use loops. Unusually, functions must be defined either in a separate script file or, in this case, at the end of the script file. In many other programming languages, the opposite is true, in that functions must be defined at some point prior to their use. Loops are more straightforward, with the range syntax that is used for matrices also being used within the iteration variable. With my loop setting the and values for each data point, the only thing left to do for this step was to render the plot, setting the title and axis labels appropriately.

A screenshot of a computer

Description automatically generated

## Salting

Since the salting and smoothing portions of the program rely on variables from the previous steps, there is no call to the “clear” command in either case. For salting, the “rand” command generates a list of random numbers between 0 to 1, but this can easily be transformed to between and by subtracting 0.5 (thus moving the range bounds to -0.5 and 0.5) and then multiplying by . In this case, a standard line plot would not be accurate, as the salted function is discontinuous unlike the original. Instead, the “scatter” command is used to make a scatterplot.

A screenshot of a computer

Description automatically generated

## Smoothing

Smoothing works mostly the same in MATLAB as in the Java program, though the program for determining the bounds of each moving average window looks a lot simpler. This is due to a combination of MATLAB’s use of one-based indexing and its use of range boundaries that are inclusive on both sides. Once again, a scatterplot is used to represent the final result.

A screenshot of a computer

Description automatically generated

# Plotting, Salting, and Smoothing with Java Libraries

While still coding in Java, we can bypass some steps of the process using external libraries. These are not included in Java by default, and so must be manually downloaded and added to the project’s build path in Eclipse.

The JFreeChart library does the job of creating and rendering charts, thus bypassing Excel. Instead of outputting the raw numerical data to a CSV file, we can place the data inside an XYDataset object, and then use a method inside the ChartFactory class to automatically create a scatterplot. We then use the ChartUtilities class to render a PNG image of the chart with a desired resolution, which in this case was chosen to be 1000 pixels wide and 500 pixels tall. This PNG is automatically saved at a filename which can be specified by the programmer.

Furthermore, the Apache Commons Math Library includes a method for calculating the mean of an array of values. It includes parameters which specify the subarray to find the mean of, which will assist us in finding the moving average during the smoothing step. Determining the exact bounds, however, still must be done manually, as the mean method will throw an error if either the starting or ending index are out of bounds.

The results of inputting the data into JFreeChart methods are effectively identical to that of the manual Java implementation discussed above, so they are presented here without further comment.

## Plotting

A graph of a function

Description automatically generated

## Salting

A graph of a graph

Description automatically generated

## Smoothing

A graph showing a red line

Description automatically generated