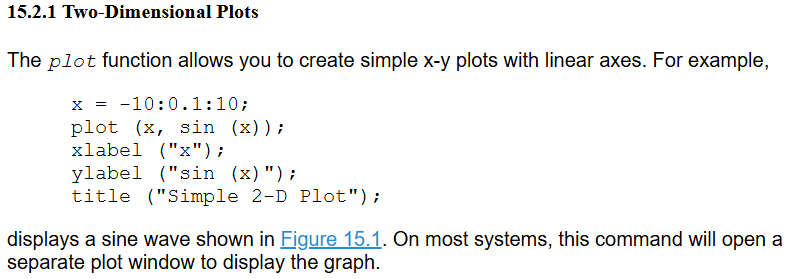
Christopher Ward  
Probability and Statistics

Octave Graphing Documentation

1. **Installation and setup**

The first thing I did was download and install the GNU Octave from their official website, <https://octave.org/download>. After successful installation I simply googled “octave graphing”, and found Octave’s official documentation for graphing ‘Two Dimensional Plots’ here, <https://docs.octave.org/latest/Two_002dDimensional-Plots.html>. In their first example in section ‘**15.2.1: Two-Dimensional Plots**’, they use the following code to plot a simple 2D sin equation.

A graph of a function

Description automatically generated

1. **Plotting our function**

Next, using examples from the same documentation for the 2D plotting, I defined a parabola as a test example, with an x-range of (-10, 10), stepping by 0.5 each time. The function I chose was x2 to make my first trial of salting and smoothing in Octave simple. Since I was unfamiliar with the syntax, I didn’t want do a function too complex and confuse myself. However, I took the liberty of using the ‘Format Arguments’ on the Octave documentation to change the line’s color and adjust the width to a size of 2, making the graph more visible.

A white background with black text

Description automatically generatedA blue line on a white background with Gateway Arch in the background

Description automatically generated

1. **Salting our function**

To salt my function, I made use of the ‘randn’ function in Octave, which generates a random number, and I added it to my y value, creating a new ‘salted\_y’ value. I then adjusted my plot function to plot the new values and I added comments for clarity in my code.

A computer code with text

Description automatically generated with medium confidenceA graph of a function

Description automatically generated

I was unhappy with the salt values after seeing the graph I was given since it’s not very noticeable aside from near the apex of the graph. To force our salt values into showing a bigger change, I multiplied each salt by 5 before adding it to original value. This change produced a salted graph that I was more satisfied with.

A graph of a graph

Description automatically generated  


1. **Smoothing our function**

For smoothing this graph, we need a couple things. First, we need to set our smoothing window. Second, we need to find a moving average of our ‘salted\_y’ data points. After googling moving average in Octave, I found a ‘movmean’ function within the Octave documentation that, “calculates the moving average over a sliding window of length wlen on data x” ([Octave](https://docs.octave.org/v9.1.0/Statistics-on-Sliding-Windows-of-Data.html#XREFmovmean)). Using this ‘movmean’ function and my newly created window variable, I defined a ‘smoothed\_y’ variable to calculate the moving average of our ‘salted\_y’ using our window size and plot the results. This time I also added x and y axis labels to the graph.

A screenshot of a computer code

Description automatically generatedA green line graph with white background

Description automatically generated

1. **Original, Salted, Smoothed Functions Overlayed**

The smoothed result looks like our original graph, but I wanted to know just how similar our smoothed result is to our original graph. Well, in the same documentation we’ve been working with, under section **15.2.7: Manipulation of Plot Windows** the ‘hold on’ function “retains plot data and settings so that subsequent plot commands are displayed on a single graph. Line color and line style are advanced for each new plot added.” ([Octave](https://docs.octave.org/latest/Manipulation-of-Plot-Windows.html)). Adding this simple command, I could overlay all three of my functions (original, salted, smoothed) on one graph to see just how much the function was changed. However now with a combined graph it was difficult to tell which line was which even with the color difference. So, I added a legend to help differentiate between the line colors. Lasty, I realized the title was now incorrect, so I updated it accordingly. This was the final graph I ended up with: *\*\*With the graph screenshot I included a simple ‘hold on’ example to show usage rather than a snippet of my entire code file\*\**

A black text on a white background

Description automatically generatedA graph of a function

Description automatically generated

1. **Closing Remarks**

A graph of a function

Description automatically generated

From this final graph, we can determine that the salting and smoothing I implemented is fairly accurate. The salter added some visible noise to the original graph. The smoother then returned a good portion of the graph to near its original values.

This project provided me with valuable experience with the tools GNU Octave provides, specifically focusing on two-dimensional graphing, data manipulation, and visualization techniques. Through my experimentation with salting and smoothing I enhanced my understanding statistical programming by branching and using valuable data visualization tools such as Octave. By exploring and applying Octave's documentation and functions like ‘randn’, ‘movmean’, and ‘hold on’, I gained practical insights into creating, modifying, and analyzing functions in Octave effectively.

Additionally, this process emphasized the importance of iterative refinement and adaptability, as I adjusted my methods to achieve a more visually meaningful representation of salted and smoothed data. This hands-on approach not only deepened my technical proficiency but also reinforced problem-solving skills critical for data analysis and visualization tasks. Overall, this exercise was both insightful and instrumental in strengthening my real-world skills of Octave for statistical and graphical purposes.