# MatLab Octave Documentation

Source used : https://wiki.octave.org/Using_Octave
Second tutorial :
Source used : https://www.youtube.com/watch?v=ZOs4eqoXPPA
Octave Tutorial #1: Getting Started for Absolute Beginners

For this part of the project, I used GNU Octave to get familiar with the basics and then tried out salt and smooth on a couple of simple functions. After going through some tutorials and testing commands, I picked two functions ; `sin(x)` and `cos(x)`, and added noise to them to simulate what distorted or messy data might look like. Then I applied a smoothing filter to clean it up and see the difference. This write up is accompanied by screenshots depicting what I did. In addition to that the .m files for the code for both sin(x) and cos(x) will be included in the OctaveStuff section of my GitHub submission.
The initial tutorial walkthrough, where I tested variable declarations, basic functions, and how to create and organize vectors. I experimented with assigning values to variables, using basic math operations, and exploring how Octave handles arrays and vector indexing. This helped me get comfortable with the syntax and the way Octave handles data structures compared to other languages like Java or Python.

A few examples of vectorization and plotting, which helped lay the groundwork for the later salt and smooth part. I also learned how to customize these plots with titles, axis labels, colors, and line widths — which became useful when I needed to compare the clean, noisy, and smoothed versions of each function on the same graph.
Using Octave in general was infinitely easier than what we had to do for part one of our project where we had to code each of the plotter, salter, and smoother manually.

Variable declaration

```
>> a = 1;
>> t = 99 + 1;
>> disp(t)
100
```

Basic functions

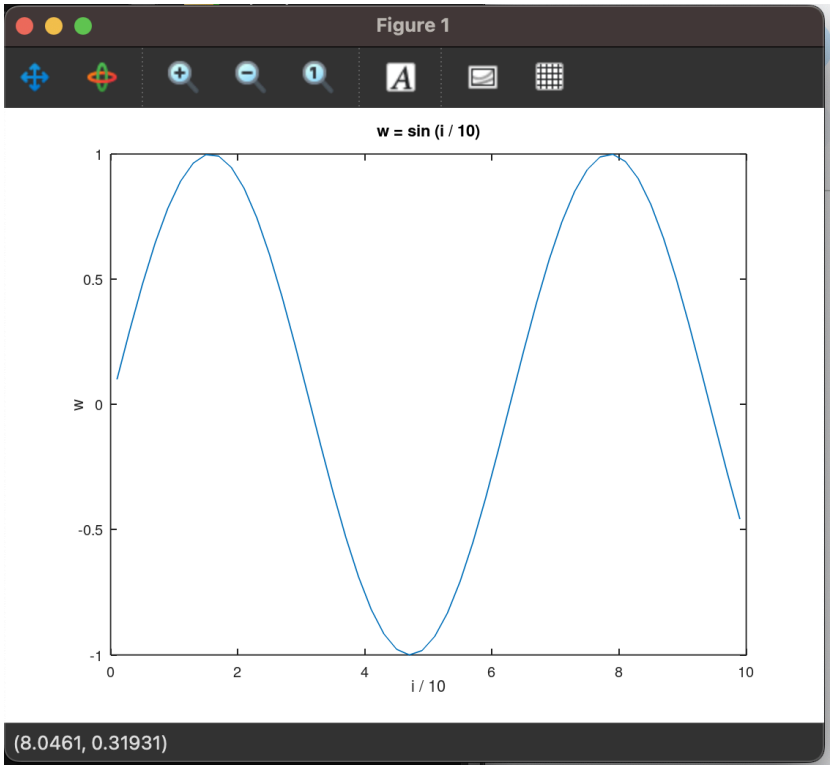
```
>> x = 3/4 * pi;
>> disp(x)
2.3562
>> y = sin(x)
y = 0.7071
```

Vector organization

```
>> columnVec = [8; 6; 4]
columnVec =

   8
   6
   4

>> mat = [8 6 4; 2 0 -2]
mat =

   8   6   4
   2   0  -2
```

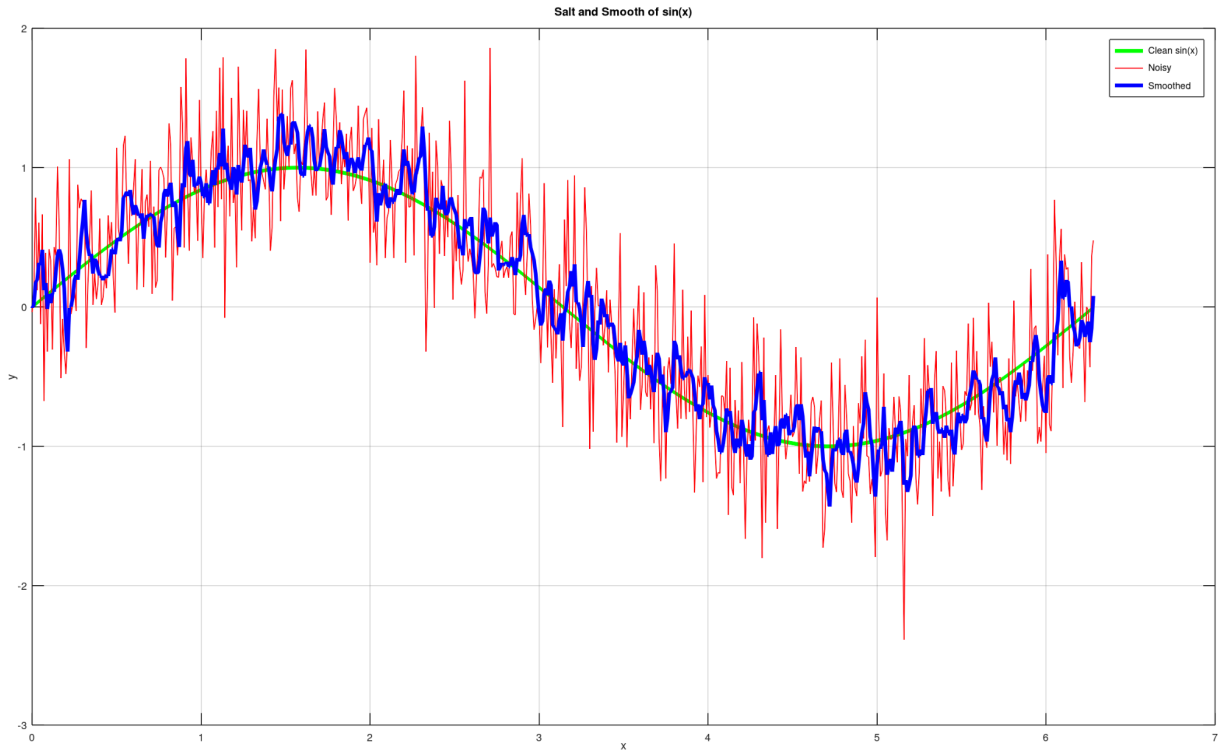
vectorization/plotting

```
>> i = 1:2:100;      # create an array with 50-elements
x = i.^2;          # each element is squared
y = x + 9;         # add 9 to each element
z = y./i;          # divide each element in y by the corresponding value in i
w = sin (i / 10); # take the sine of each element divided by 10
>> plot (i / 10, w);
title ('w = sin (i / 10)');
xlabel ('i / 10');
ylabel ('w');
>>
```



I started with `y = sin(x)` since it's a basic wave and easy to spot when it's been altered. I added random noise using `randn()` to salt the signal. After that, I used a moving average filter with a window size of 5 to smooth it out. The smoothed version brought the wave back into shape without all the choppiness. I plotted all three versions — the original (green), the noisy version (red), and the smoothed version (blue) — on the same graph to compare.

**Salted and Smoothed version of y = Sin(x)**




## Cos(x)

Then I did the same exact process with `y = cos(x)`. It's similar to sine but shifted — starts at a peak instead of zero. The results were pretty much the same: the noise distorted it, and smoothing cleaned it up. It worked well and helped show that the approach isn't just tied to one function.

## Salted and Smoothed Version of y = Cos(x)