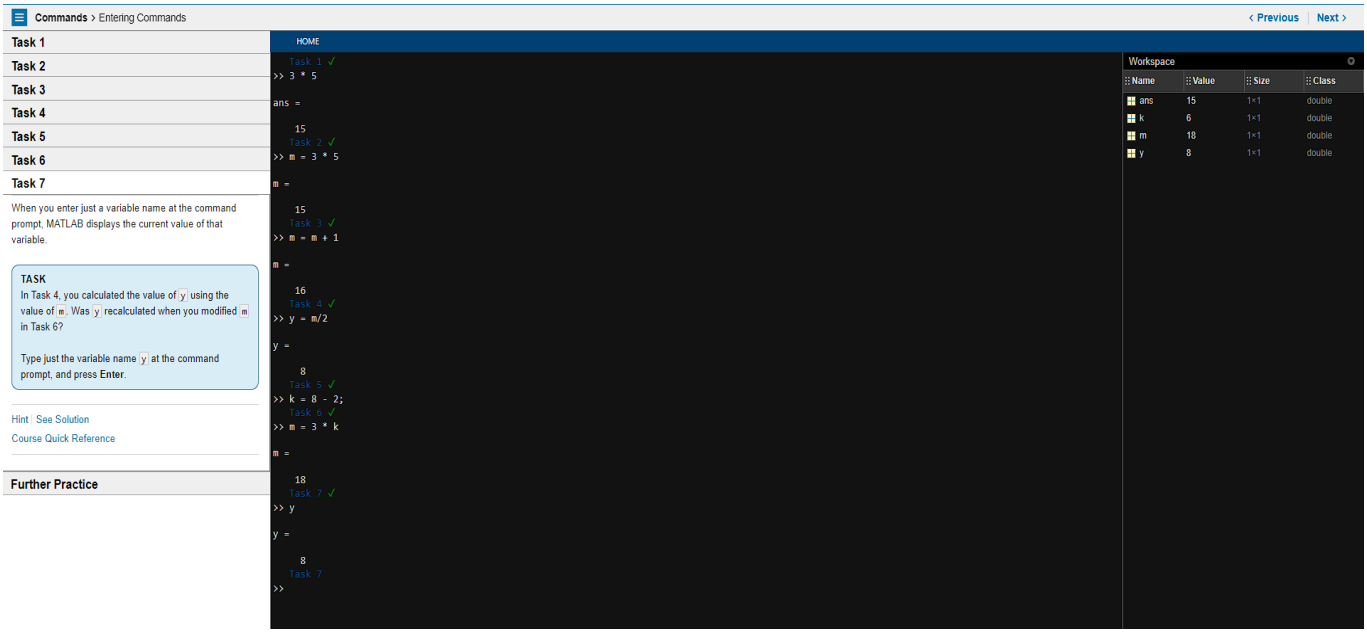
Tyler Smith

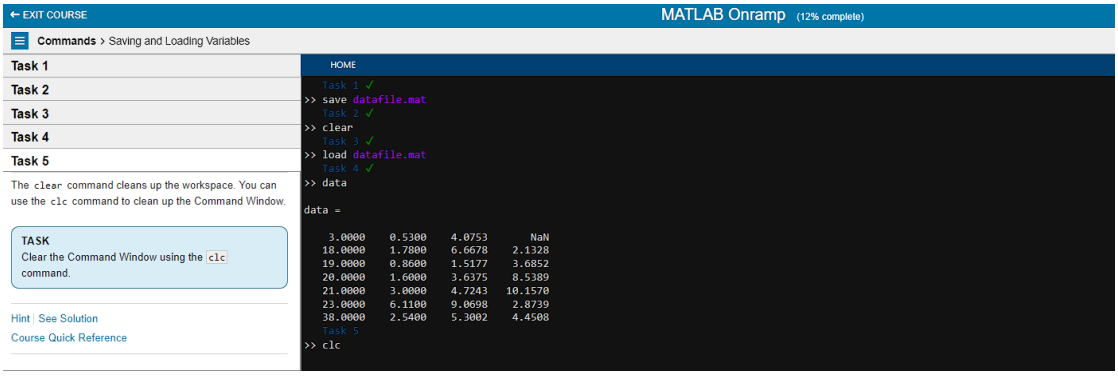
MatLab Programming

For this part of the assignment where we are tasked with learning how to program in either MatLab or Octave, I chose to learn MatLab. I will be first showing my progress along with the tutorial videos that are provided on the MatLab website and then will code my CSV generator, salter, and smoother and details about that. I am doing the MatLab Onramp online courses in order to learn the basics of the language. It was about a two hour tutorial so I won't show everything, but I did take plenty of screenshots along the way and will post them below.

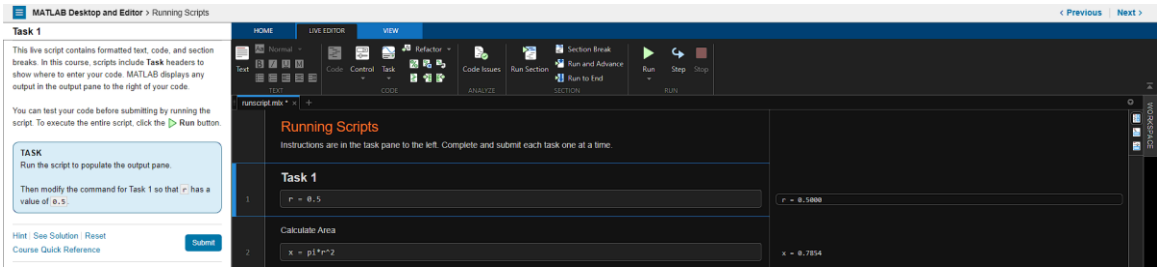
Below is a screenshot of the initial tasks I completed to get a basic understanding of how the language operates



The next section of the tutorial goes over saving and loading variables

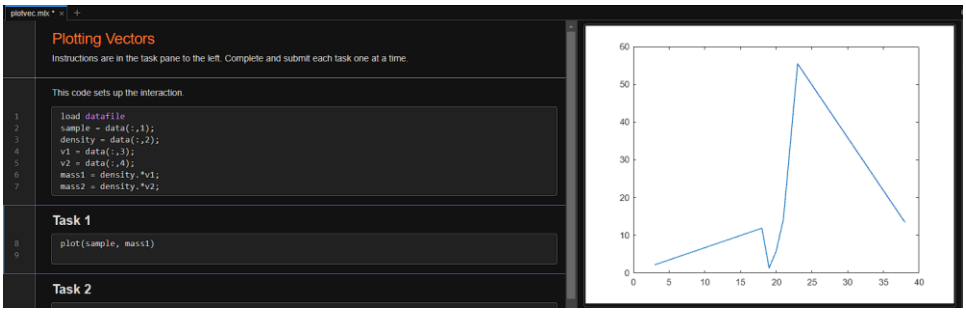


Next, the tutorial taught me how to run scripts in the live editor part of the language

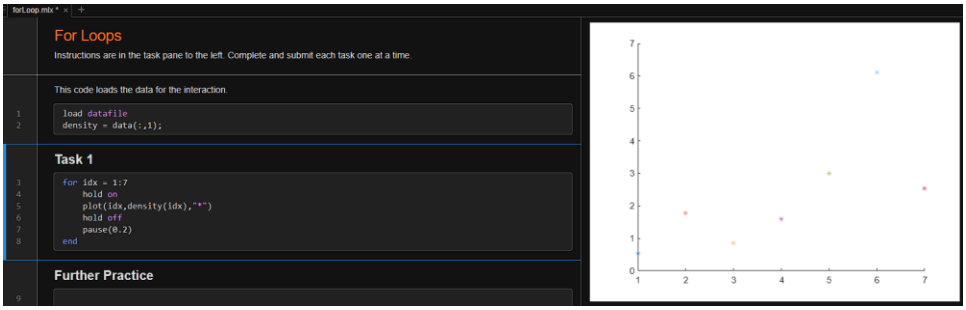


One of the next lessons I learned was how to manually enter arrays, also using the live editor feature on MATLAB

A little bit further down the line in the tutorial, I learned how to plot vectors in the MatLab language



Learning about for loops:

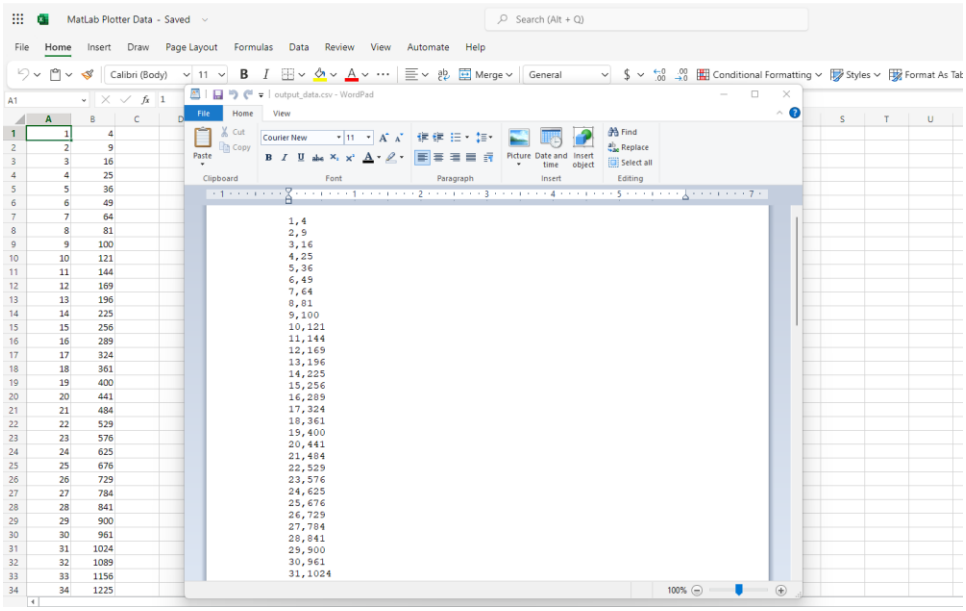


That is about all I captured while completing this tutorial, below I will show my code for the CSV generator, reader, salter, and smoother with some words about the process.

Below this is the code I wrote for the original plotter, which takes the function y = x^2 + 2x + 1, and takes a range from 1-100 and outputs the x and y values into a CSV file name ‘output_data.csv’

```
>> f = @(x) x.^2 + 2*x + 1;
x = 1:100;
y = zeros(size(x));
for i = 1:length(x)
y(i) = f(x(i));
end
data = [x' , y'];
csvwrite('output_data.csv', data);
>>
```

I then took the CSV file that was created, downloaded it, and uploaded it into excel



Below is the screenshot of the method I wrote that salts the previously plotted data similarly to how it was done in java

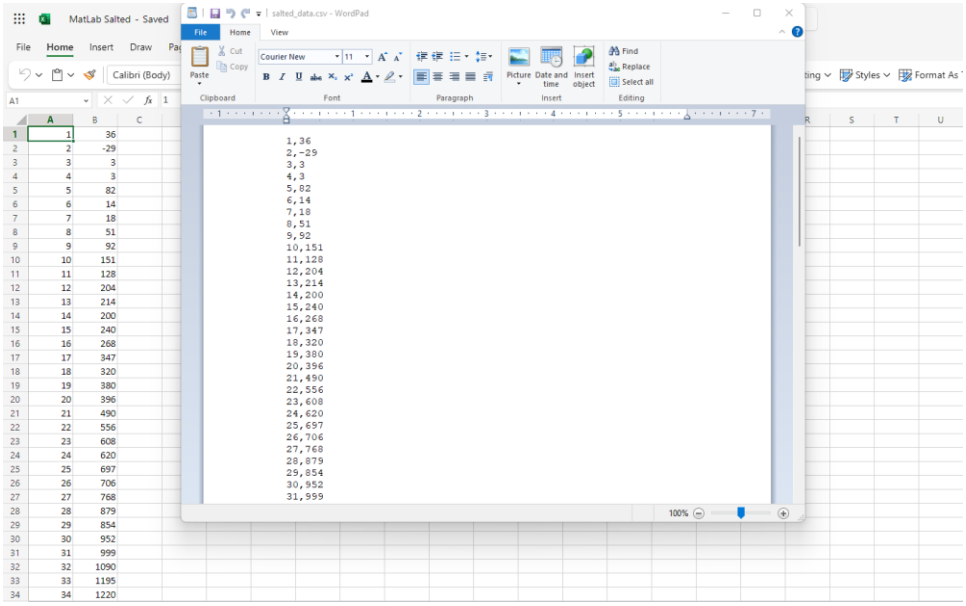
```
>> data = csvread('output_data.csv');
>> x = data(:, 1);
>> y = data(:, 2);
>> for i = 1.length(y)
r = randi([-50, 50], 1);
if randi([0, 1], 1)
y(i) = y(i) + r;
else
y(i) = y(i) - r;
end
end

ans =

   100

>> data_modified = [x, y];
>> csvwrite('salted-data.csv', data_modified);
```

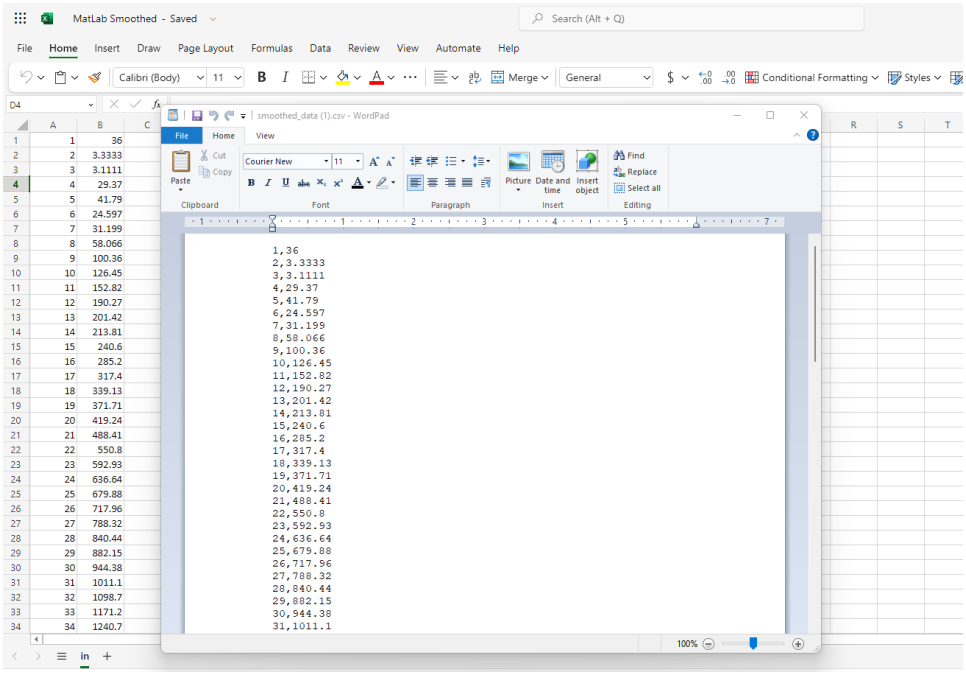
Same way I did for the plotter, I took the salted-data.csv file, downloaded it, and loaded it into an excel spreadsheet.



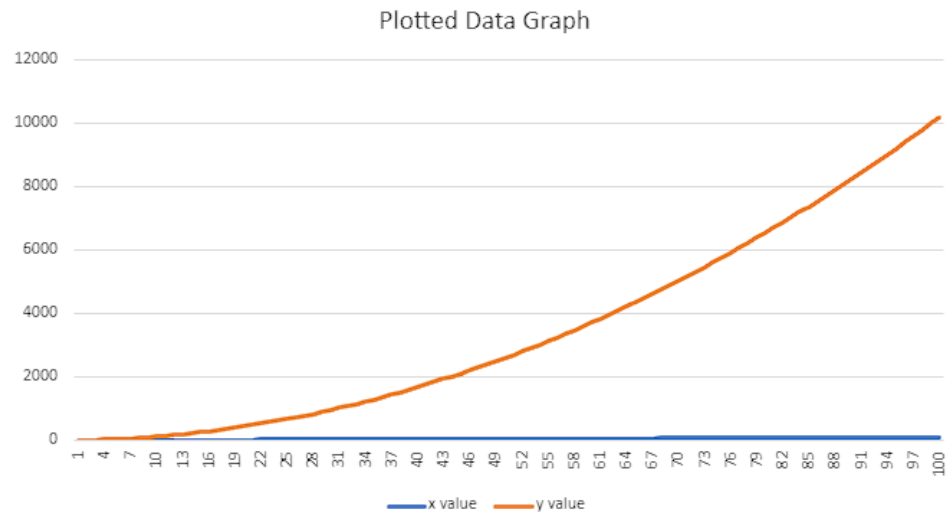
Finally, below is the method I wrote to smooth the previously salted data the same way it was done in the java programming.

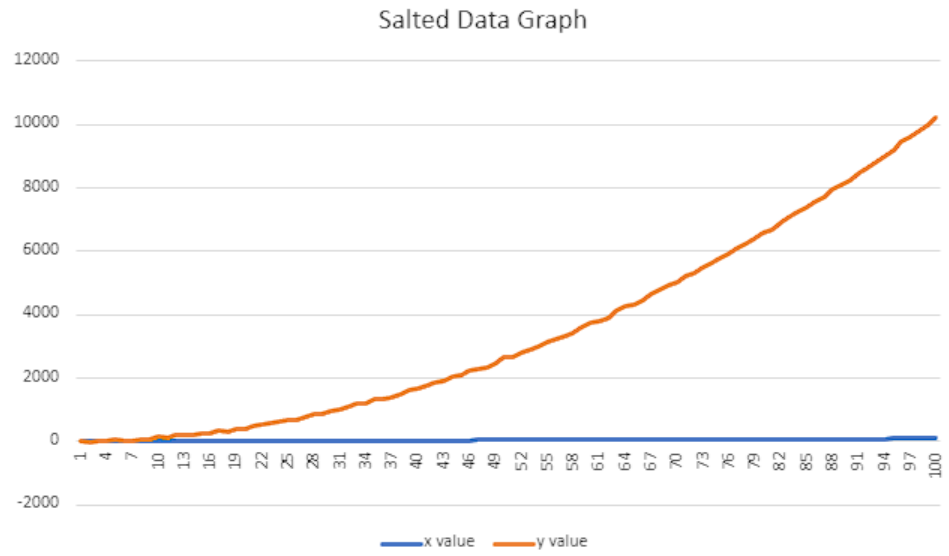
```
>> data = csvread('salted_data.csv');
>> x = data(:, 1);
>>  y = data(:, 2);
>> for i = 2:length(y)-1
y(i) = mean(y(i-1:i+1));
end
>> data_smoothed = [x, y];
>> csvwrite('smoothed_data.csv', data_smoothed);
>>
```

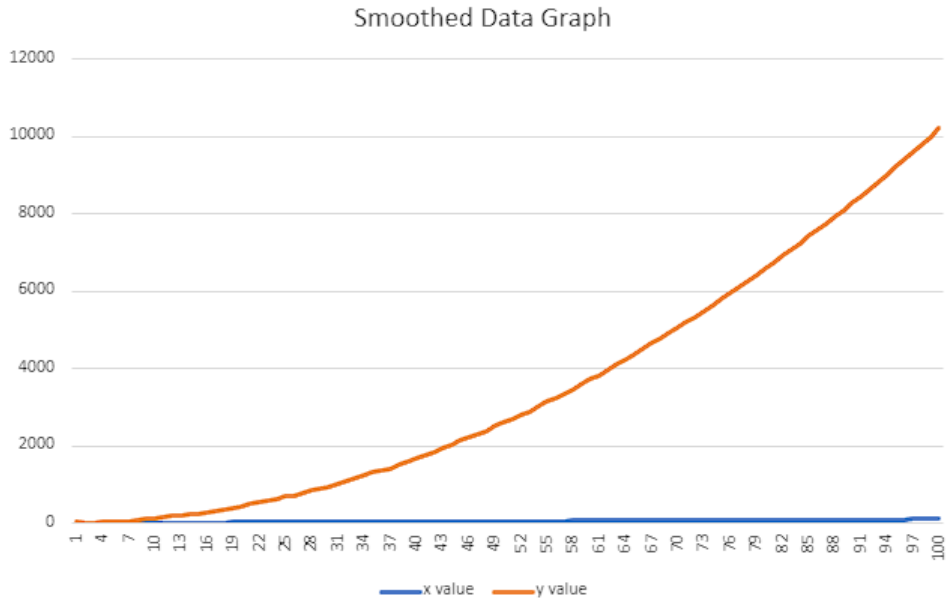
Then I downloaded the created CSV file and transferred it into an excel file as I did with the previous files



I also created graph in excel similarly to the java programming part, and here are screenshots of those below:







As one can see from these graphs, the originally plotted data outputs a normal curved line for the y values as they increase. The salted data sees that line become very bumpy as it trends upwards, due to the y values being decreased and increased by 50. As the numbers get exponentially higher, though, the bumps are not as noticeable because 50 is not a substantial change. Finally, for the smoothed data graph, you can see some bumpiness in the lower numbers for the y values, but again as the numbers get substantially larger there is not a noticeable bump for these numbers.