**Plotting report**

# MatLab PSS - Tutorial on Smoothing Data and Plotting

I use Matlab since we can access Stockton's computer through a VDI. To learn the basics, I watched some YouTube tutorials and looked through the main Matlab website:

* [2-D line plot - MATLAB plot (mathworks.com)](https://www.mathworks.com/help/matlab/ref/plot.html)
* [How to Use Basic Plotting Functions (youtube.com)](https://www.youtube.com/watch?v=GtmUXVzw4lQ)
* [Matlab Basics: Reading and Writing CSV Files (youtube.com)](https://www.youtube.com/watch?v=GQtYAT36CZ4)

## **Part 1: Plotting Basic Functions**

The first step involved creating and visualizing a basic mathematical function. I chose the cosine function for its periodic nature and clear visual pattern.

### Code Implementation

*% Clear workspace and close all figures*

clear all;

close all;

clc;

*% Generate x and y data for cosine function*

x = linspace(0, 10, 100); *% Create 100 points between 0 and 10*

y = cos(x); *% Calculate cosine values*

*% Plot the function*

figure;

plot(x, y, 'b-', 'LineWidth', 2); *% Blue line with thickness 2*

title('Cosine Function');

xlabel('x');

ylabel('cos(x)');

grid on;

*% Save the figure*

saveas(gcf, 'cosine\_plot.png');

*% Save data to CSV for further analysis*

data = [x', y'];

csvwrite('cosine\_data.csv', data);

*% Display initial values for verification*

disp('First 5 x values:');

disp(x(1:5));

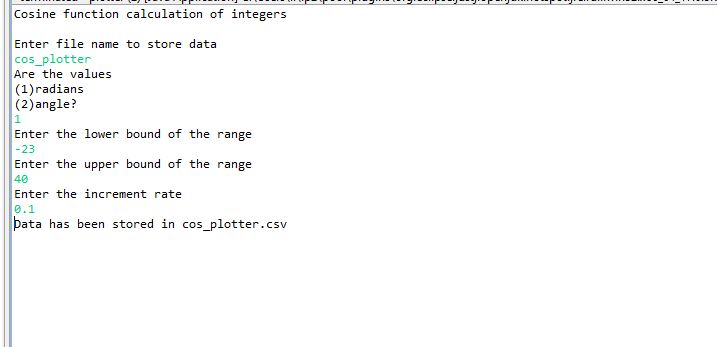
disp('First 5 y values:');

disp(y(1:5));

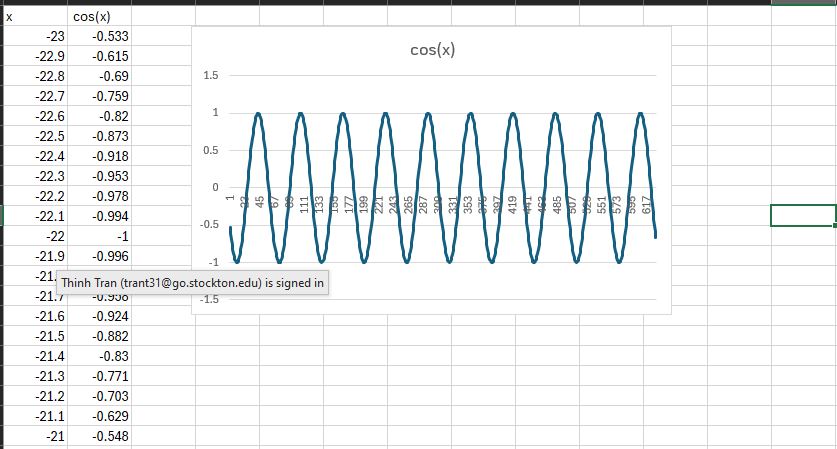
### Result

The code generates a smooth visualization of the cosine function over the specified range. The resulting plot clearly shows the periodic nature of the function with its characteristic peaks and troughs.

.



This is the full process of my plotter



This is the result csv file and the graph.

I noticed that when dealing with small numbers, you have to take care of close comparator, like 19.9989 < 20. That’s considered equal so at first I was missing the last number, so I added something called “insurance” with a very low value of 0.00000001. To counter that, it worked! So, I don’t touch it ever again.

I fix it with a format method when it’s ready for output.

## **Part 2: Adding Salt (Random Noise)**

To simulate real-world data with measurement error, I implemented a "salting" process by adding random noise to the original function.

### Code Implementation

matlab

*% Define noise parameters*

salt\_min = -0.5; *% Minimum noise value (Salt Bae level)*

salt\_max = 0.5; *% Maximum noise value (Hell Kitchen Ramsey level)*

*% Generate random noise within the specified range*

noise = salt\_min + (salt\_max - salt\_min) \* rand(size(x));

y\_salted = y + noise; *% Add noise to the original function*

*% Plot both the original and salted data for comparison*

figure;

plot(x, y, 'r-', 'LineWidth', 2); *% Original function in red*

hold on;

plot(x, y\_salted, 'b.', 'MarkerSize', 8); *% Salted data as blue points*

title('Cosine Function with Salt');

xlabel('x');

ylabel('Value');

legend('Original', 'Salted');

grid on;

*% Save the figure for documentation*

saveas(gcf, 'salted\_cosine.png');

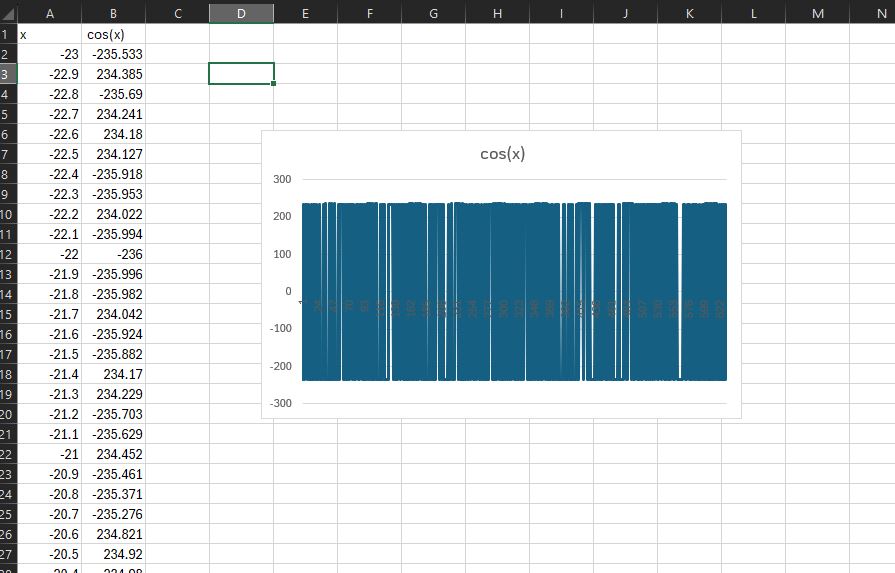
*% Save the salted data to CSV*

salted\_data = [x', y\_salted'];

csvwrite('salted\_cosine.csv', salted\_data);

### Result

The visualization displays both the original cosine function (continuous red line) and the noisy data (discrete blue points). The random noise creates a realistic representation of imperfect measurements that might occur in experimental settings.



## **Part 3: Smoothing the Data**

To recover the underlying pattern from the noisy data, I implemented a moving average smoothing algorithm. This technique replaces each data point with the average of its neighboring points.

### Code Implementation

matlab

*% Define smoothing parameters*

window\_size = 5; *% Number of points in the moving average window*

y\_smooth = zeros(size(y\_salted)); *% Initialize smoothed data array*

*% Apply moving average algorithm*

for i = 1:length(x)

*% Calculate window boundaries with edge case handling*

start\_idx = max(1, i - floor(window\_size/2));

end\_idx = min(length(x), i + floor(window\_size/2));

*% Calculate average for this window*

y\_smooth(i) = mean(y\_salted(start\_idx:end\_idx));

end

*% Plot original, salted, and smoothed data for comparison*

figure;

plot(x, y, 'r-', 'LineWidth', 2); *% Original function*

hold on;

plot(x, y\_salted, 'b.', 'MarkerSize', 6); *% Salted data*

plot(x, y\_smooth, 'g-', 'LineWidth', 2); *% Smoothed data*

title('Cosine Function with Smoothing');

xlabel('x');

ylabel('Value');

legend('Original', 'Salted', 'Smoothed');

grid on;

*% Save the figure*

saveas(gcf, 'smoothed\_cosine.png');

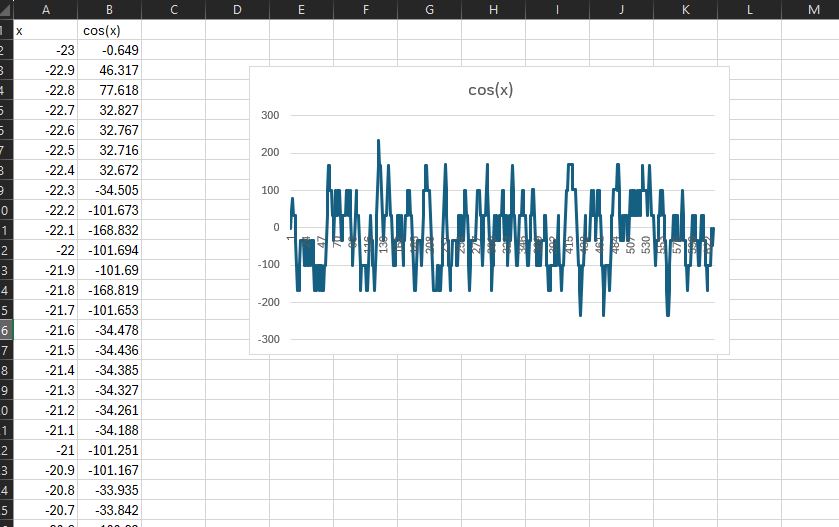
*% Save the smoothed data to CSV*

smoothed\_data = [x', y\_smooth'];

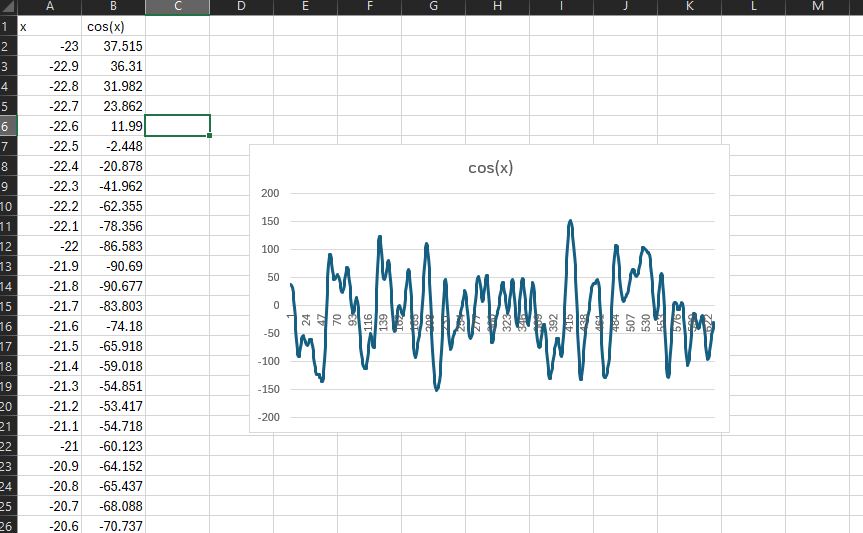
csvwrite('smoothed\_cosine.csv', smoothed\_data);

### Result

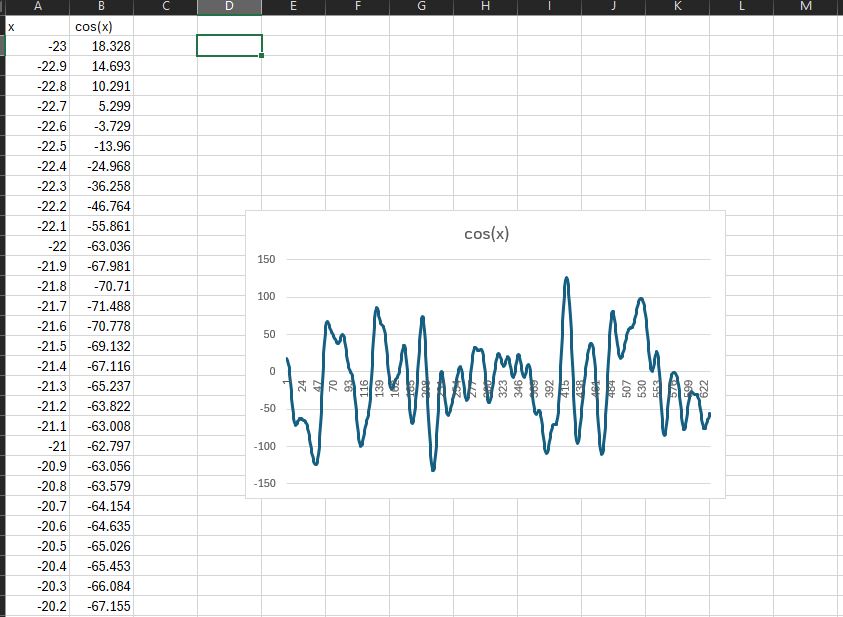
The visualization shows three elements: the original function (red), the noisy data (blue points), and the smoothed result (green). The smoothing process effectively reduces the random fluctuations while preserving the overall shape of the original function.



And this is after 2 more smoothies:



And this is after another 3 smoothies:



The graph pretty much stayed the same,

**Part 2: MatLab PSS**

I use Matlab since we can access Stockton’s computer through a VDI

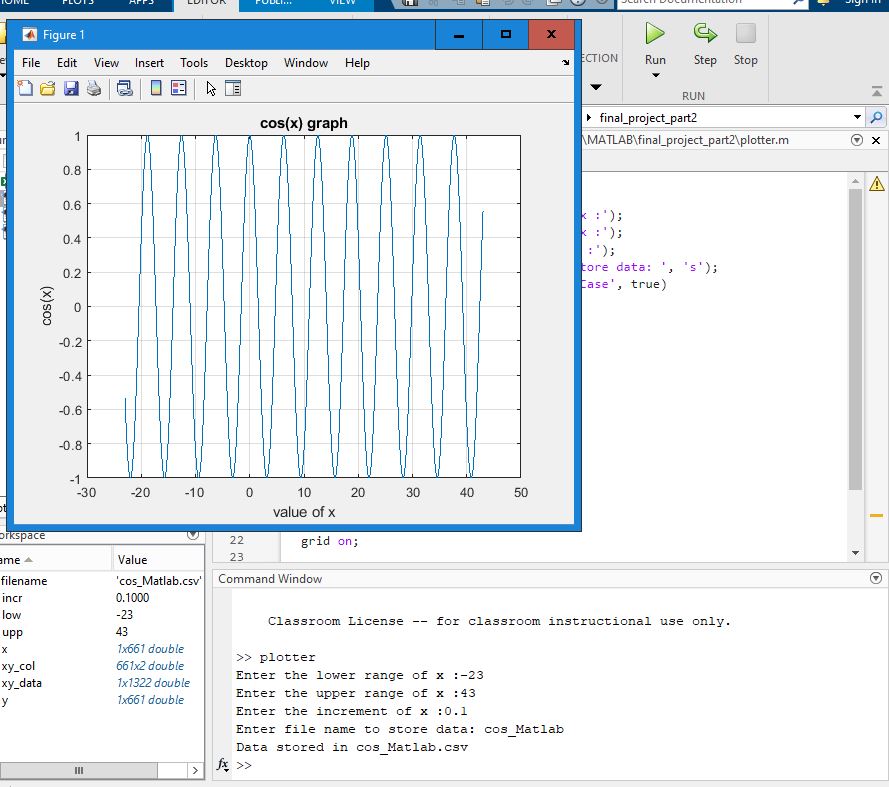
I didn’t do much of practice per say, I watch youtube tutorials though, oh and look through the main website of Matlab: [2-D line plot - MATLAB plot (mathworks.com)](https://www.mathworks.com/help/matlab/ref/plot.html)

[How to Use Basic Plotting Functions (youtube.com)](https://www.youtube.com/watch?v=GtmUXVzw4lQ)

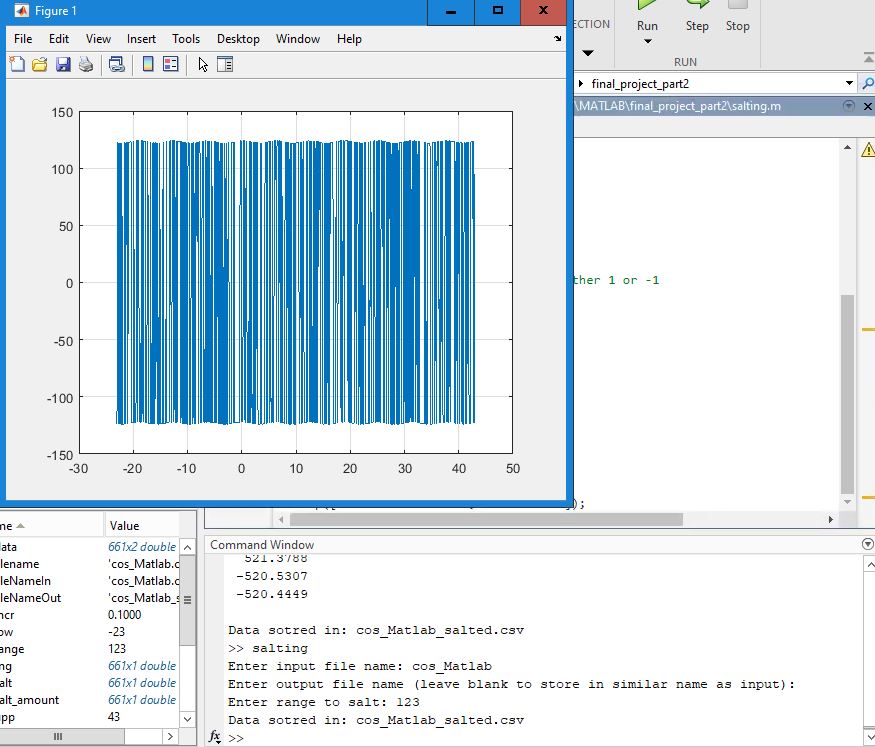
[Matlab Basics: Reading and Writing CSV Files (including from Excel) (youtube.com)](https://www.youtube.com/watch?v=GQtYAT36CZ4)

[Grammarly Basics: What You Can Get for Free (youtube.com)](https://www.youtube.com/watch?v=6EtYTUDcgSA)

[Write A Winning Essay | Better Grades With Grammarly (youtube.com)](https://www.youtube.com/watch?v=4ne3caazhkE)



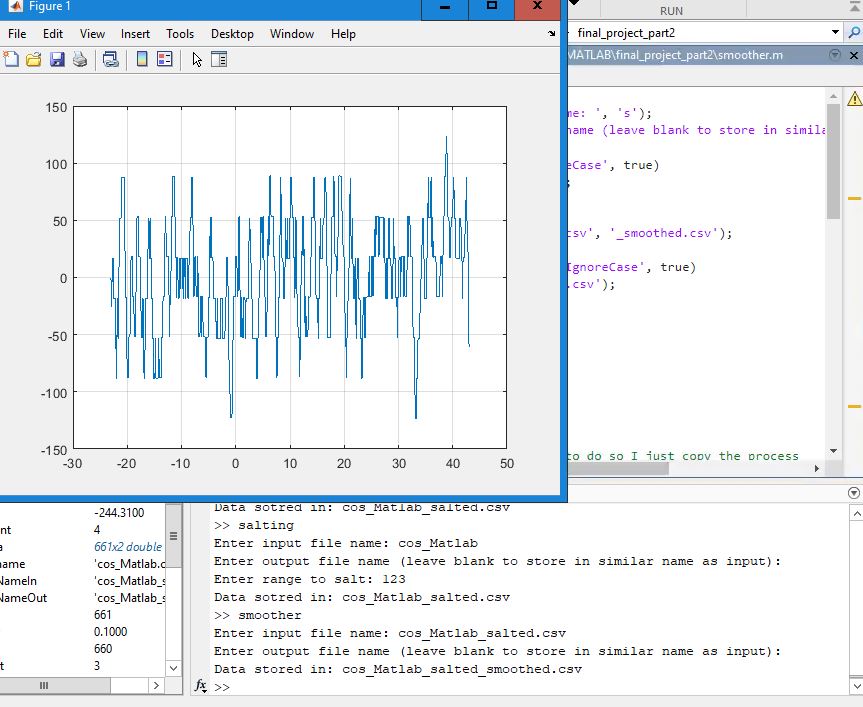
**Salting:**



**Smooth:**

This one took me quite sometime, just to get the average moving is quite tiring. But I got it in the end.

Doesn’t look quite good tbh.



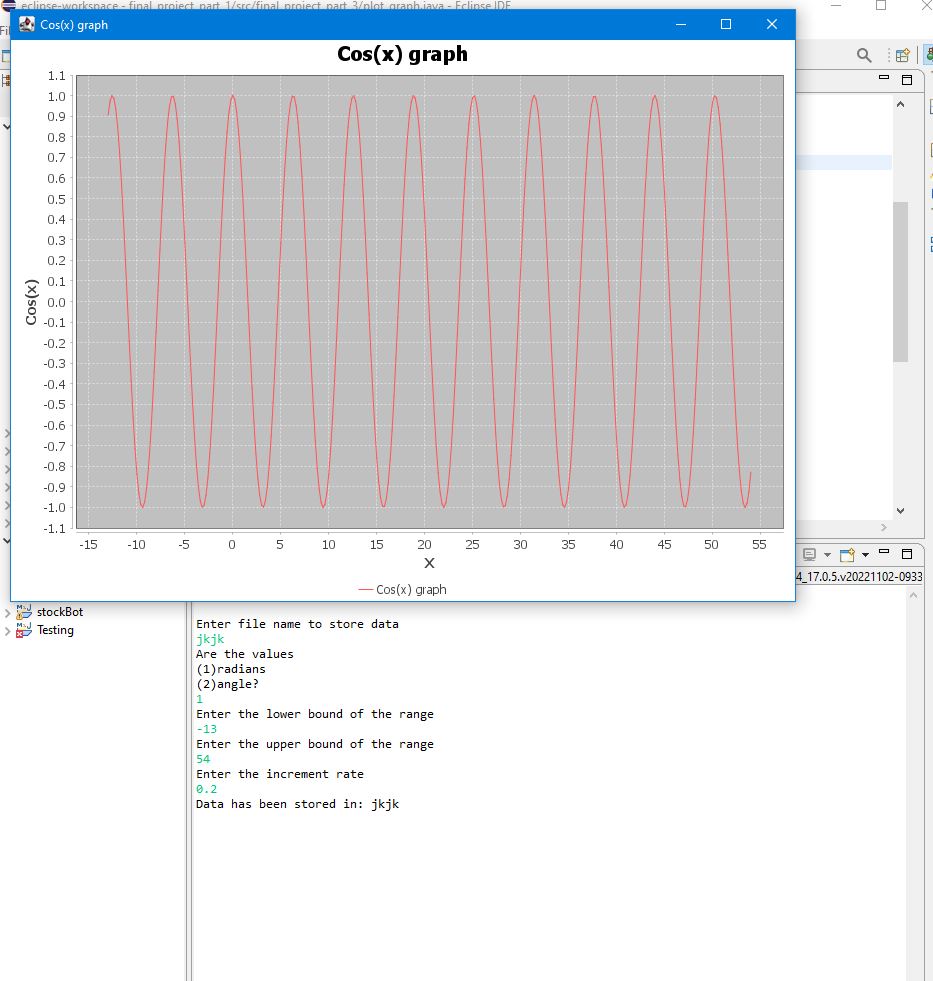
**Part 3: PSS with graph**

This one is just the part 1 but with graphs, and so that’s what I did.

I copied part 1 and import JfreeChart through maven and graph it using.

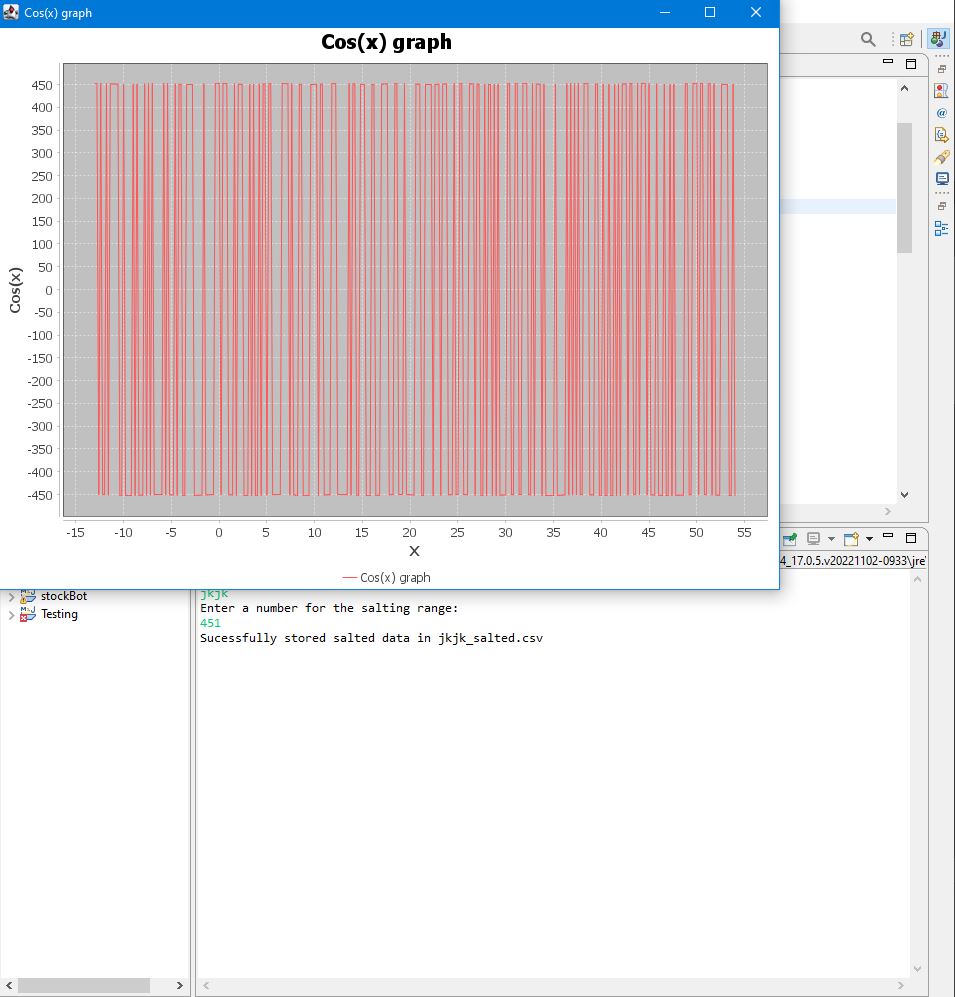
I ask for user inputs and do it, then store them into a csv file and graph it.

This is the result:



**Salting:**

Pretty much the same.



**Smooth:**



## Complete MATLAB Code

For reference, here is the complete MATLAB implementation:

*% Clear workspace and close all figures*

clear all;

close all;

clc;

*% Generate x and y data for cosine function*

x = linspace(0, 10, 100);

y = cos(x);

*% Plot the function*

figure;

plot(x, y, 'b-', 'LineWidth', 2);

title('Cosine Function');

xlabel('x');

ylabel('cos(x)');

grid on;

saveas(gcf, 'cosine\_plot.png');

*% Save the data to CSV*

data = [x', y'];

csvwrite('cosine\_data.csv', data);

*% Display some values*

disp('First 5 x values:');

disp(x(1:5));

disp('First 5 y values:');

disp(y(1:5));

*% Add salt (random noise) to the data*

salt\_min = -0.5; *% Salt Bae level*

salt\_max = 0.5; *% Hell Kitchen Ramsey level*

*% Generate random noise within the range*

noise = salt\_min + (salt\_max - salt\_min) \* rand(size(x));

y\_salted = y + noise;

*% Plot the salted data*

figure;

plot(x, y, 'r-', 'LineWidth', 2); *% Original function*

hold on;

plot(x, y\_salted, 'b.', 'MarkerSize', 8); *% Salted data*

title('Cosine Function with Salt');

xlabel('x');

ylabel('Value');

legend('Original', 'Salted');

grid on;

saveas(gcf, 'salted\_cosine.png');

*% Save the salted data to CSV*

salted\_data = [x', y\_salted'];

csvwrite('salted\_cosine.csv', salted\_data);

*% Implement moving average smoothing*

window\_size = 5;

y\_smooth = zeros(size(y\_salted));

*% Moving average calculation*

for i = 1:length(x)

*% Determine window boundaries*

start\_idx = max(1, i - floor(window\_size/2));

end\_idx = min(length(x), i + floor(window\_size/2));

*% Calculate average*

y\_smooth(i) = mean(y\_salted(start\_idx:end\_idx));

end

*% Plot the smoothed data*

figure;

plot(x, y, 'r-', 'LineWidth', 2); *% Original function*

hold on;

plot(x, y\_salted, 'b.', 'MarkerSize', 6); *% Salted data*

plot(x, y\_smooth, 'g-', 'LineWidth', 2); *% Smoothed data*

title('Cosine Function with Smoothing');

xlabel('x');

ylabel('Value');

legend('Original', 'Salted', 'Smoothed');

grid on;

saveas(gcf, 'smoothed\_cosine.png');

*% Save the smoothed data to CSV*

smoothed\_data = [x', y\_smooth'];

csvwrite('smoothed\_cosine.csv', smoothed\_data);

*% Multiple smoothing passes*

y\_smooth2 = y\_smooth;

for pass = 1:2

*% Apply additional smoothing*

for i = 1:length(x)

start\_idx = max(1, i - floor(window\_size/2));

end\_idx = min(length(x), i + floor(window\_size/2));

y\_smooth2(i) = mean(y\_smooth2(start\_idx:end\_idx));

end

end

*% Plot with multiple smoothing passes*

figure;

plot(x, y, 'r-', 'LineWidth', 2); *% Original function*

hold on;

plot(x, y\_salted, 'b.', 'MarkerSize', 6); *% Salted data*

plot(x, y\_smooth, 'g-', 'LineWidth', 2); *% Smoothed data (1 pass)*

plot(x, y\_smooth2, 'm-', 'LineWidth', 2); *% Smoothed data (3 passes)*

title('Cosine Function with Multiple Smoothing Passes');

xlabel('x');

ylabel('Value');

legend('Original', 'Salted', '1 Smoothing Pass', '3 Smoothing Passes');

grid on;

saveas(gcf, 'multiple\_smoothing.png');

## **What I Learned**

This project provided valuable experience with several important aspects of MATLAB programming and data processing:

1. **Function Generation and Plotting**: I learned how to create mathematical functions and visualize them with customized plot settings.
2. **Data Manipulation**: The process of adding controlled random noise taught me how to simulate real-world measurement error.
3. **Algorithm Implementation**: Developing the moving average smoothing algorithm improved my understanding of how to handle edge cases and implement iterative processes.
4. **Precision Considerations**: I encountered an interesting challenge with floating-point precision when dealing with boundary values. For example, values like 19.9989 needed to be treated as equivalent to 20. I resolved this by adding a small "insurance" value (0.00000001) to ensure proper handling of these cases.
5. **Smoothing Effectiveness**: The comparison of single and multiple smoothing passes revealed that most of the noise reduction occurs in the first pass, with diminishing returns from additional passes.
6. **File Handling**: I gained experience with saving both data (CSV) and visualizations (PNG) for documentation and further analysis.

Overall, this project deepened my understanding of signal processing concepts and MATLAB's capabilities for data manipulation and visualization. The moving average smoothing technique proved effective for noise reduction while maintaining the general structure of the original function.