# **MATLAB Code Documentation**

## **File: plotter.m**

### **Purpose**

Calculates cosine values over a specified range, plots the results, and saves the data to a CSV file.

### **Script Details**

*% ask user for parameters*

*%Calculating cos(x)*

low = input('Enter the lower range of x :');

* Prompts the user to enter the lower bound of the x range.
* Stores the value in the variable low.

upp = input('Enter the upper range of x :');

* Prompts the user to enter the upper bound of the x range.
* Stores the value in the variable upp.

incr = input('Enter the increment of x :');

* Prompts the user to enter the increment value for stepping through the x range.
* Stores the value in the variable incr.

filename = input('Enter file name to store data: ', 's');

* Prompts the user to enter a filename to store the data.
* The 's' parameter specifies that the input should be treated as a string.
* Stores the value in the variable filename.

if ~endsWith(filename, '.csv', 'IgnoreCase', true)

filename = strcat(filename, '.csv');

End

* Checks if the filename ends with '.csv' (case-insensitive).
* If not, appends '.csv' to the filename using strcat.

*% now plot the the data*

x = (low:incr:upp);

Creates a vector x with values from low to upp in steps of incr.

y = cos(x);

* Calculates the cosine of each value in x and stores the results in vector y.

xy\_data = [x, y];

* Creates a matrix xy\_data with x values in the first column and y values in the second column.

xy\_col = [x(:), y(:)];

* Creates a matrix xy\_col with x values in the first column and y values in the second column.
* The : operator ensures the vectors are column vectors.

figure;

plot(x, y);

title('cos(x) graph');

xlabel('value of x');

ylabel('cos(x)');

grid on;

* Creates a new figure window.
* Plots y against x.
* Sets the title of the plot to 'cos(x) graph'.
* Labels the x-axis as 'value of x'.
* Labels the y-axis as 'cos(x)'.
* Adds a grid to the plot.

*% Store data in csv file*

csvwrite(filename, xy\_col);

* Writes the data in xy\_col to a CSV file with the specified filename.

disp(['Data stored in ', filename]);

* Displays a message indicating where the data was stored.

## **File: salting.m**

### **Purpose**

Reads cosine data from a CSV file, adds random noise ("salt") to the y-values, plots the results, and saves the modified data to a new CSV file.

### **Script Details**

*% ask for file names and parameters*

fileNameIn = input('Enter input file name: ', 's');

* Prompts the user to enter the input filename.
* Stores the value in the variable fileNameIn.

fileNameOut = input('Enter output file name (leave blank to store in similar name as input): ', 's');

* Prompts the user to enter the output filename.
* The user can leave this blank to use a modified version of the input filename.
* Stores the value in the variable fileNameOut.

range = input('Enter range to salt: ');

* Prompts the user to enter the range value for "salting".
* Stores the value in the variable range.

if ~endsWith(fileNameIn, '.csv', 'IgnoreCase', true)

fileNameIn = strcat(fileNameIn, '.csv');

end

* Checks if the input filename ends with '.csv' (case-insensitive).
* If not, appends '.csv' to the filename using strcat.

if strcmp(fileNameOut, '')

fileNameOut = strrep(fileNameIn, '.csv', '\_salted.csv');

else

if ~endsWith(fileNameOut, '.csv', 'IgnoreCase', true)

fileNameOut = strcat(fileNameOut, '.csv');

end

end

* If the output filename is empty (user left it blank):
  + Replaces '.csv' in the input filename with '\_salted.csv'.
* Otherwise:
  + Checks if the output filename ends with '.csv' (case-insensitive).
  + If not, appends '.csv' to the filename.

data = csvread(fileNameIn);

* Reads numeric data from the specified CSV file into the variable data.

*%store data*

y\_data = data(:, 2);

* Extracts the second column (y-values) from data and stores it in y\_data.

x\_data = data(:, 1);

* Extracts the first column (x-values) from data and stores it in x\_data.

rng = randi([0, 1], size(y\_data));

* Generates random integers (0 or 1) with the same size as y\_data.
* Stores these values in the variable rng.

salt = (rng \* 2) - 1; *%this will be either 1 or -1*

* Transforms the random integers (0 or 1) into either -1 or 1.
* If rng element is 0, the corresponding salt element will be -1.
* If rng element is 1, the corresponding salt element will be 1.

salt\_amount = salt \* range;

* Multiplies each element in salt by the user-specified range.
* This determines the amount of "salt" to add to each y-value.

y\_salted = y\_data + salt\_amount;

* Adds the salt\_amount to each y-value, creating "salted" data.

figure;

title('Salted graph');

xlabel('value of x');

ylabel('Seasoned cos(x)');

plot(x\_data, y\_salted);

grid on

* Creates a new figure window.
* Sets the title of the plot to 'Salted graph'.
* Labels the x-axis as 'value of x'.
* Labels the y-axis as 'Seasoned cos(x)'.
* Plots the salted y-values against the x-values.
* Adds a grid to the plot.

xy\_salted = [x(:), y\_salted(:)];

* Creates a matrix xy\_salted with x values in the first column and y\_salted values in the second column.
* The : operator ensures the vectors are column vectors.

csvwrite(fileNameOut, xy\_salted);

* Writes the salted data to the specified output CSV file.

disp(['Data sotred in: ', fileNameOut]);

* Displays a message indicating where the data was stored.
* Note: There's a typo in "sotred" which should be "stored".

## **File: smoother.m**

### **Purpose**

Reads data from a CSV file, applies a smoothing algorithm to the y-values, plots the results, and saves the smoothed data to a new CSV file.

### **Script Details**

*% ask for file names and parameters*

fileNameIn = input('Enter input file name: ', 's');

* Prompts the user to enter the input filename.
* Stores the value in the variable fileNameIn.

fileNameOut = input('Enter output file name (leave blank to store in similar name as input): ', 's');

* Prompts the user to enter the output filename.
* The user can leave this blank to use a modified version of the input filename.
* Stores the value in the variable fileNameOut.

if ~endsWith(fileNameIn, '.csv', 'IgnoreCase', true)

fileNameIn = strcat(fileNameIn, '.csv');

end

* Checks if the input filename ends with '.csv' (case-insensitive).
* If not, appends '.csv' to the filename using strcat.

if strcmp(fileNameOut, '')

fileNameOut = strrep(fileNameIn, '.csv', '\_smoothed.csv');

else

if ~endsWith(fileNameOut, '.csv', 'IgnoreCase', true)

fileNameOut = strcat(fileNameOut, '.csv');

end

end

* If the output filename is empty (user left it blank):
  + Replaces '.csv' in the input filename with '\_smoothed.csv'.
* Otherwise:
  + Checks if the output filename ends with '.csv' (case-insensitive).
  + If not, appends '.csv' to the filename.

data = csvread(fileNameIn);

* Reads numeric data from the specified CSV file into the variable data.

*% store data*

y\_data = data(:, 2);

* Extracts the second column (y-values) from data and stores it in y\_data.

x\_data = data(:, 1);

* Extracts the first column (x-values) from data and stores it in x\_data.

*% smoothing process, I don't know what to do so I just copy the process*

*% from part 1 :p*

limit = 3;

* Sets a limit for how many adjacent points to consider in each direction for smoothing.
* This value (3) means up to 3 points before and 3 points after the current point will be used.

average = zeros(1, numel(y\_data));

* Creates a vector average filled with zeros, with the same size as y\_data.
* This will store the smoothed y-values.

for i = 1:numel(y\_data)

count = 1;

avg = y\_data(i);

* Iterates through each element in y\_data.
* Initializes count to 1 (counting the current point).
* Initializes avg to the current y-value.

*%This add the left eleemnt*

for l = max(1, i - limit):i-1

if l <= numel(y\_data) && l >= 1

avg = avg + y\_data(l);

count = count + 1;

end

end

* Processes points to the "left" (before) the current point.
* The range starts at max(1, i-limit) to avoid negative indices.
* The range ends at i-1 (the point just before the current point).
* For each valid point, adds its y-value to avg and increments count.

*% this add the right side*

for u = i + 1:min(numel(y\_data), i + limit)

if u <= 3 && u <= numel(y\_data) || u < numel(y\_data)

avg = avg + y\_data(u);

count = count + 1;

end

end

* Processes points to the "right" (after) the current point.
* The range starts at i+1 (the point just after the current point).
* The range ends at min(numel(y\_data), i+limit) to avoid exceeding the array bounds.
* The condition inside the loop is complex and potentially has issues:
  + u <= 3 && u <= numel(y\_data) checks if u is at most 3 and not beyond the array.
  + u < numel(y\_data) checks if u is not the last element.
  + These conditions are joined with ||, so the check passes if either condition is true.
  + This might not work as intended for arrays larger than 3 elements.

average(i) = avg / count;

end

* Calculates the average by dividing avg by count.
* Stores the result in the average array at position i.

*% store data*

xy\_smooth = [x\_data(:), average(:)];

* Creates a matrix xy\_smooth with x\_data values in the first column and average values in the second column.
* The : operator ensures the vectors are column vectors.

csvwrite(fileNameOut, xy\_smooth);

* Writes the smoothed data to the specified output CSV file.

disp(['Data stored in: ', fileNameOut]);

* Displays a message indicating where the data was stored.

*% Now graph the data*

figure;

title('Smoothed cos(x) graph');

xlabel('value of x');

ylabel('smoothed cos(x)');

plot(x\_data, average);

grid on

* Creates a new figure window.
* Sets the title of the plot to 'Smoothed cos(x) graph'.
* Labels the x-axis as 'value of x'.
* Labels the y-axis as 'smoothed cos(x)'.
* Plots the smoothed y-values against the x-values.
* Adds a grid to the plot.