INTRODUCTION TO MATLAB EXERCISE

This exercise covers the process of creating a coordinate axis, and plotting a function on this axes. The students will become familiar with loops, conditional statements (if/then statements), functions, structures, and plotting.

Task 1: Create a coordinate axis

We want to create an array of N evenly-spaced, consecutive numbers between 0 and 1 that we can use to plot a Gaussian distribution later. Write a function that takes an (integer) number N as input and returns an array of size N containing the evenly-spaced, consecutive numbers between 0 and 1. Use a for-loop in the function to create the array.

For instance if the user sets N = 5 and sends N to the function, it will return an array [0.00 0.25 0.50 0.75 1.00]

Compare your results with the built-in Matlab function "linspace" (See Matlab help on linspace). Are the results the same? If not, what is the difference?

```
%% Task 1
% Creating the function
function f = distribution(N)
    % Pre-creating the array for the values
    array = zeros([1 N]);
    % Calculating the difference between each value, there are N-1 steps
    diff = 1/(N-1);
    % Looping to fill the array with the wanted values
    for i = 1:N
        array(i) = (i-1)*diff;
    end
    % Defining what to return in th function
    f = array;
end
```

Testing the function using N=5

```
%% Task 1
N = 5;
% Calling the functioon
x_values = distribution(N);

% Creating an array using linspace
lin_values = linspace(0,1,N);

% Calculating the difference
difference = x_values - lin_values;

disp(x_values)
disp(difference)
```

The out put of N is:

```
         0    0.2500    0.5000    0.7500    1.0000
```

At N=5, there are no difference, increasing to N=10 gives differences for some of the values in the size order of 10^{-16}:

Difference =

```
1.0e-15 *

         0         0         0         0         0         0         0   -0.1110         0         0
```

Task 2: Generate a log-normal distribution function

Using the coordinate axis generated in the previous task, use a for-loop to create an array that stores the value of a log-normal distribution function defined as follows:

$$f_i = \frac{1}{x_i \sigma \sqrt{2\pi}} e^{-\frac{(\ln x_i - \mu)^2}{2\sigma^2}}$$

In the above equation, x_i is the ith element of the coordinate array generated in task 1, f_i is the ith element of the array containing the value of a log-normal distribution at the corresponding point x_i, and μ and σ are parameters of the distribution; use values of -0.530 and 0.136 for μ and σ, respectively.

Using N=5:

```
%% Task 2
% Using the array created from the function in task 1, we can insert it
% into the formula

% Defining the given parameters
my = -0.530;
sigma = 0.136;

%Creating an empty vector
log_norm_vec = zeros([1 N]);

% Filling the vector with the log-normal values
for i = 1:N
    log_norm_vec(i) = exp(-(log(x_values(i)) - my)^2/(2*sigma^2))/...
        (x_values(i)*sigma*sqrt(2*pi));
end

disp(log_norm_vec)
```

The output is:

```
log_norm_vec =    NaN    0.0000    2.8570    0.7997    0.0015
```

Task 3: Plot the log-normal distribution along the coordinate axis you generated

Using the information from the attached slides and exercise session lecture generate a function that plots the log-normal distribution function. The default MatLab plots are ugly, so we will set the properties of the figure, axes and plot objects using structures and set statements as well as built in functions to make it look nice. The plotting function should take the coordinate axis array, log-normal distribution array and two property structures as inputs and return nothing.

Task 3a: Define a structure called "figProps" with the following two properties: Color = [1 1 1] and OuterPosition = [170, 170, 1280, 960] (Note: you can adjust the numbers in the OuterPosition properties if the plot is too big for your screen).

```
%% Task 3
```

```
%% a
% Had to change the numbers in OuterPosition du to screen size
figProps = struct('Color', [1 1 1], 'OuterPosition', [170, 170, 1000, 700]);
```

Task 3b: Define a structure called "fontProps" with the following three properties: FontName = 'Calibri', FontSize = 18, and FontWeight = 'bold'

```
%% b
fontProps = struct('FontName', 'Calibri', 'FontSize', 18, 'FontWeight', 'bold');
```

Task 3c: Write the function that takes the coordinate axis, the log-normal distribution array and the two property structures as input and plots the log-normal distribution. Use a set statement to define the properties of the figure object, use a set statement to define the font sizes of the axes object and use the built in Matlab functions called “xlim”, “box”, and “grid” to set the x-axis limits t0 [0 1], to place a boarder around the plot, and to turn on the gridlines, respectively. Plot the function with the following properties defined in the call to the Matlab “plot” function: 'Color', 'r'; 'LineWidth', 2; 'Marker', 'o'; 'MarkerEdgeColor', 'r'; 'MarkerFaceColor', 'none'; 'DisplayName', 'f(x)'. Put a title on the plot along with labels for the x and y axes; use set statements to define the fonts for the title and axis labels. Add a legend to the plot and use a set statement to define the font of the legend. Have the function save an image file of the plot.

How large to the value of N have to be in order to make the distribution curve look smooth?

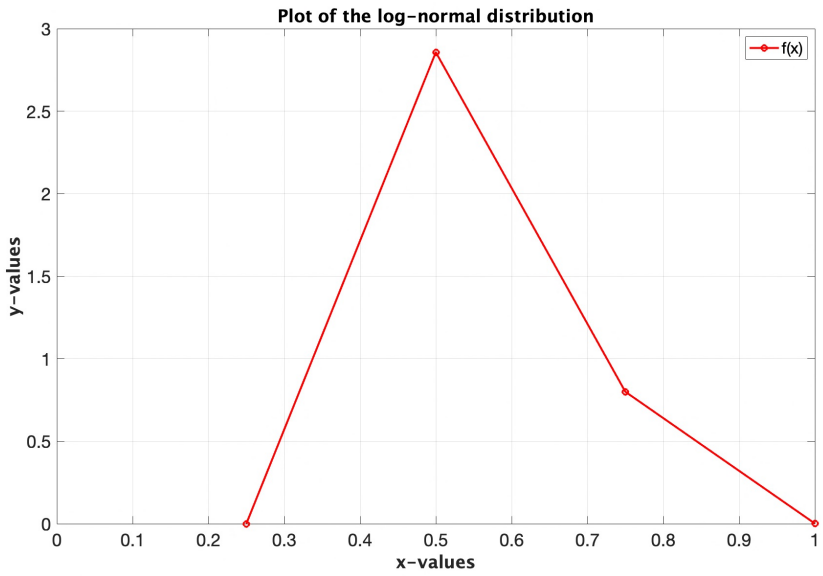
The written function:

```
%% Task 3
function plotter(x, y, figpro, fontpro)
    fig = figure(1);
    ax = axes;
    % Plotting the values with the given proerties
    plot(x, y, 'Color', 'r', 'LineWidth', 2, 'Marker', 'o', 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'none', 'DisplayName', 'f(x)');
    % Set the properties of the figure to the figpro values
    set(fig, figpro);
    % Set the fontsize of the ax = axes object
    set(ax, 'FontSize', fontpro.FontSize);
    % Set labels and title, with the properties defined in fontProps
    set(xlabel('x-values'), fontpro);
    set(ylabel('y-values'), fontpro);
    set(title('Plot of the log-normal distribution'), fontpro);
    % Setting limits on the x-axis
    xlim([0 1]);
    % Place a border around the plot
    box("on");
    % Turn on gridlines
    grid("on");
    % Add a legend and set its font to the font in fontProps
    set(legend(), 'FontName', fontpro.FontName)
    %saveas(fig, 'Ex4_plot', 'jpg')
end
```

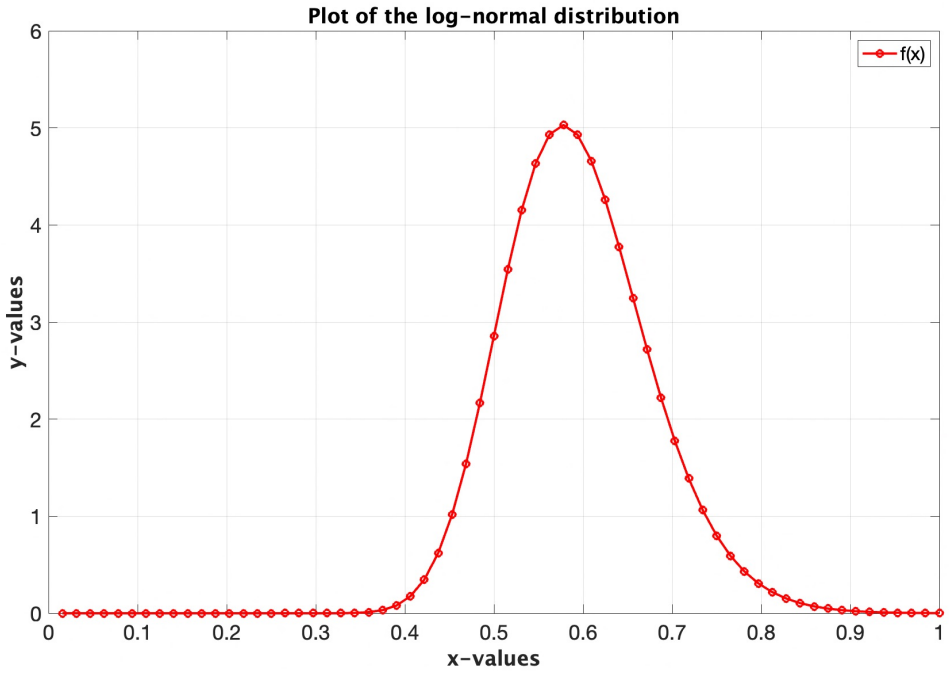
The call:

```
%% c
plotter(x_values, log_norm_vec, figProps, fontProps);
```

The resulting graph for N=5:



Gradually increasing N until the curve is smooth gives a smooth curve at N≈65:



Task 4: Find the peak and the tailing and leading edge of the distribution

Use for-loops, if/then statements and/or while loops to find the following:

a) The array index, x value, and y value of the distribution's peak.
b) The array index, x value, and y value of the distribution's tailing edge.
c) The array index, x value, and y value of the distribution's leading edge.

The leading and tailing edge of the distribution can be assumed to occur at the points where f(x) is less than 0.1% of the y value at the peak of the distribution.

a) Finding the peak using a for-loop, the distribution is strictly increasing until it reaches it's peak. => When log_norm_vec(i+1) < log_norm_vec(i), then log_norm_vec(i) is the peak

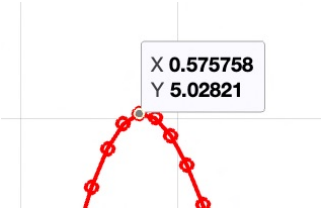
```
%% Task 4
%% a
peak = -1;
peak_index = 0;
length = numel(log_norm_vec);

for i = 1:length
    if log_norm_vec(i+1)<log_norm_vec(i)
        peak = log_norm_vec(i);
        peak_index = i;
        break
    end
end
fprintf('For N = %d\n', N)
fprintf('For the peak, the index is %d, x = %f, and y = %f\n', peak_index, x_values(peak_index), peak);
```

Using N=100, the output is:

```
For N = 100
For the peak, the index is 58, x = 0.575758, and y = 5.028214
```

Which matches the plot:



b) The tailing edge is "before" the wave.

As the task states that the edges occur at the points where $f(x)$ is LESS THAN $0{,}1\% = 0{,}001$ of the y-value of the peak, we iterate until the next y-value exceeds this threshold.

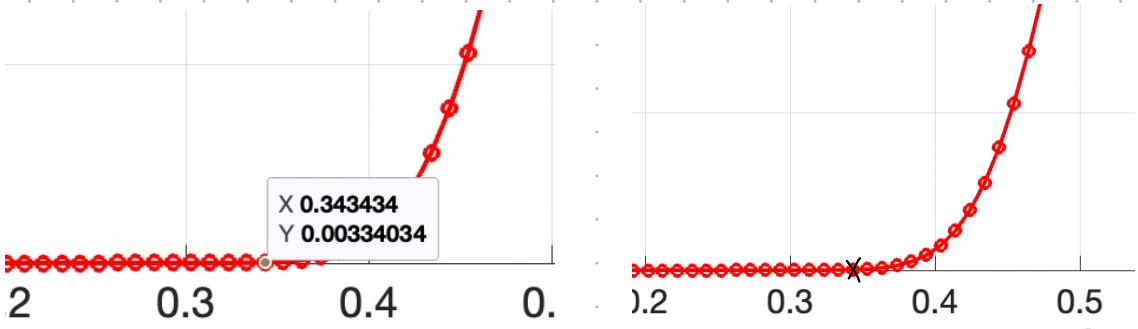
```
%% b
tailing_edge = -1;
tail_index = 0;

for j = 1:length
    if log_norm_vec(j+1) > 0.001*peak
        tailing_edge = log_norm_vec(j);
        tail_index = j;
        break
    end
end
fprintf('For the tailing edge, the index is %d, x = %f, and y = %f\n', tail_index, x_values(tail_index), tailing_edge);
```

the output for N = 100:

```
For the tailing edge, the index is 35, x = 0.343434, and y = 0.003340
```

which corresponds to this point of the plot:



The point is the last point before the y-values increase substantially.

c) Similar to b), but iterating backwards from the end of the array:

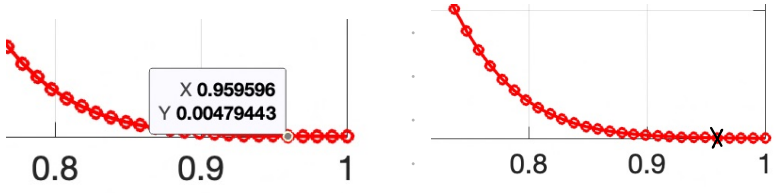
```
%% c
leading_edge = -1;
lead_index = 0;

for k = length:-1:1
    if log_norm_vec(k-1) > 0.001*peak
        leading_edge = log_norm_vec(k);
        lead_index = k;
        break
    end
end
fprintf('For the leading edge, the index is %d, x = %f, and y = %f\n', lead_index, x_values(lead_index), leading_edge);
```

The output is:

```
For the leading edge, the index is 96, x = 0.959596, and y = 0.004794
```

Corresponding to:



Which is the point where the curve is ≈ flat.