**LAB # 04**

**INTRODUCTION TO MATLAB**

****

**Spring 2024**

**CSE-301L**

**Operating System Lab**

Submitted by: **NAVEED AHMAD**

Registration No.: **22PWCSE2165**

Class Section: **B**

“On my honor, as student of University of Engineering and Technology, I have neither given nor received unauthorized assistance on this academic work.”

Student Signature: A blue line drawing on a white background

Description automatically generated

Submitted to:

**Dr. Safdar Nawaz Khan Marwat**

March 24, 2024.

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

**Objectives:**

This lab will help you grasp the following concepts:

* Discrete Signal representation in Matlab
* Matlab Graphics
* Two Dimensional Plots
* Plot and subplot
* Different Plotting Functions Used in Matlab

**Task # 01:**

Given the signals:

x1[n] = [2 5 8 4 3]

x2[n] = [4 3 2]

1. Write a MatLab program that adds these two signals. Use vector addition and multiplication.
2. Instead of using vector addition and multiplication, use for loop to add and multiply the signals.
3. Design a function **SigPlot** that takes the original signals and their sum and product as input and plots them as:
4. Separate Figures,
5. Single Figure overlapping all the signals, and iiI) Single Figure with separate signal plots using subplots.

**Problem Analysis:**

This task involves basic signal processing operations such as addition and multiplication of discrete-time signals. It also requires visualization of the original signals, their sum, and product using MATLAB.

**Algorithm:**

Step by step algorithm given below:

%Define the signals

%Pad the shorter signal with zeros to make their lengths equal

%Vector addition

%Vector multiplication

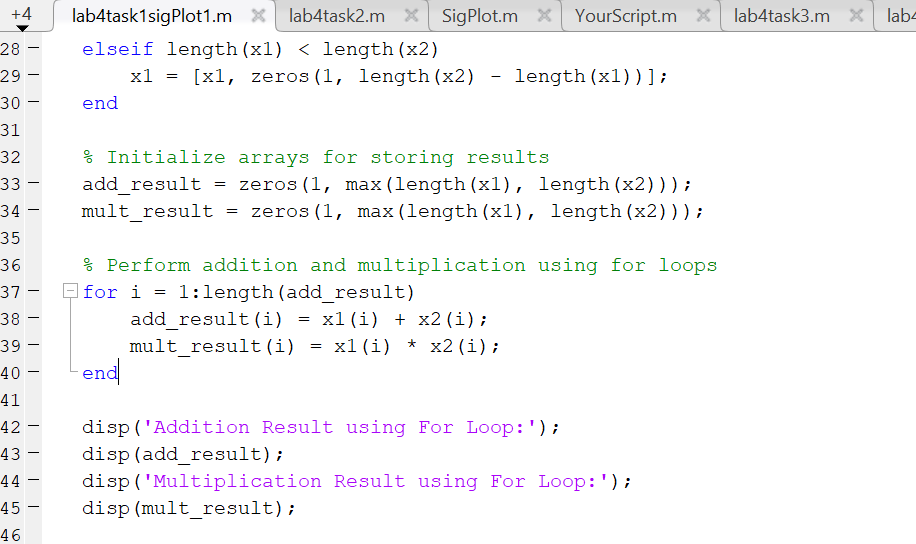
%Define the signals

%Pad the shorter signal with zeros to make their lengths equal

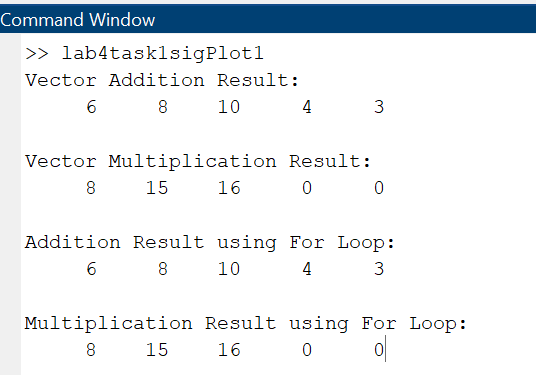
%Initialize arrays for storing results

%Perform addition and multiplication using for loops

**Code:**



Output:



**Task # 02:**

Amplitude scaling by a factor β causes each sample to get multiplied by β. Write a user‐defined function **ScaleSig** that has two input arguments:

1. a signal to be scaled and
2. scaling factor β. The function should return the scaled output to the calling program. In the calling program, get the discrete time signal as well as the scaling factor from user and then call the above‐mentioned function.

Design a function **SigPlot** that takes the original signals and their scaled versions as input from the main calling program and plots them as:

1. Separate Figures,
2. Single Figure overlapping all the signals, and
3. Single Figure with separate signal plots using subplot.

**Problem Analysis:**

This problem requires designing two user-defined functions: ScaleSig for scaling a signal by a given factor, and SigPlot for visualizing original signals and their scaled versions in various formats, to enhance understanding of signal scaling and data visualization techniques.

窗体顶端

窗体底端

**Algorithm:**

Step by step algorithm given below:

% take input

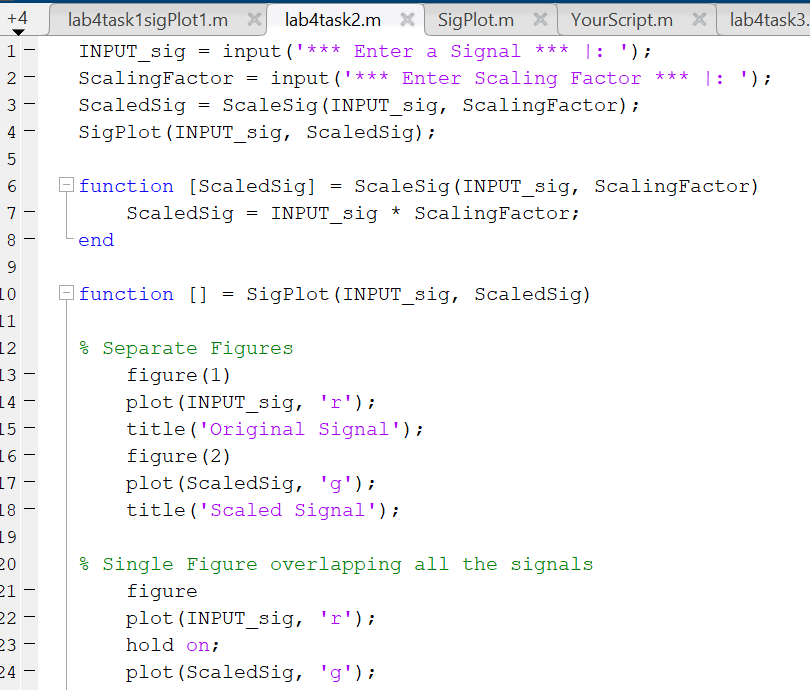
% sig plot function

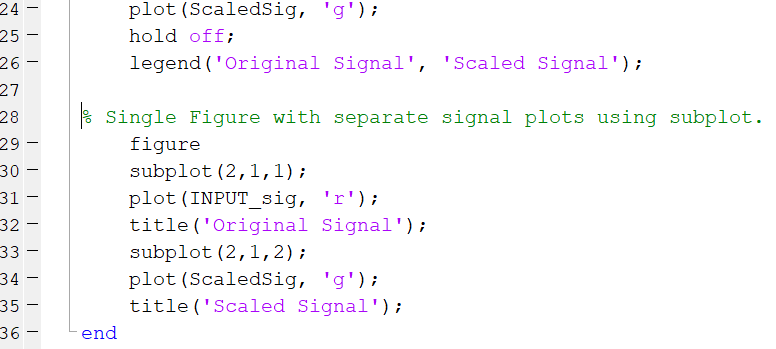
% Separate Figures

% Single Figure overlapping all the signals

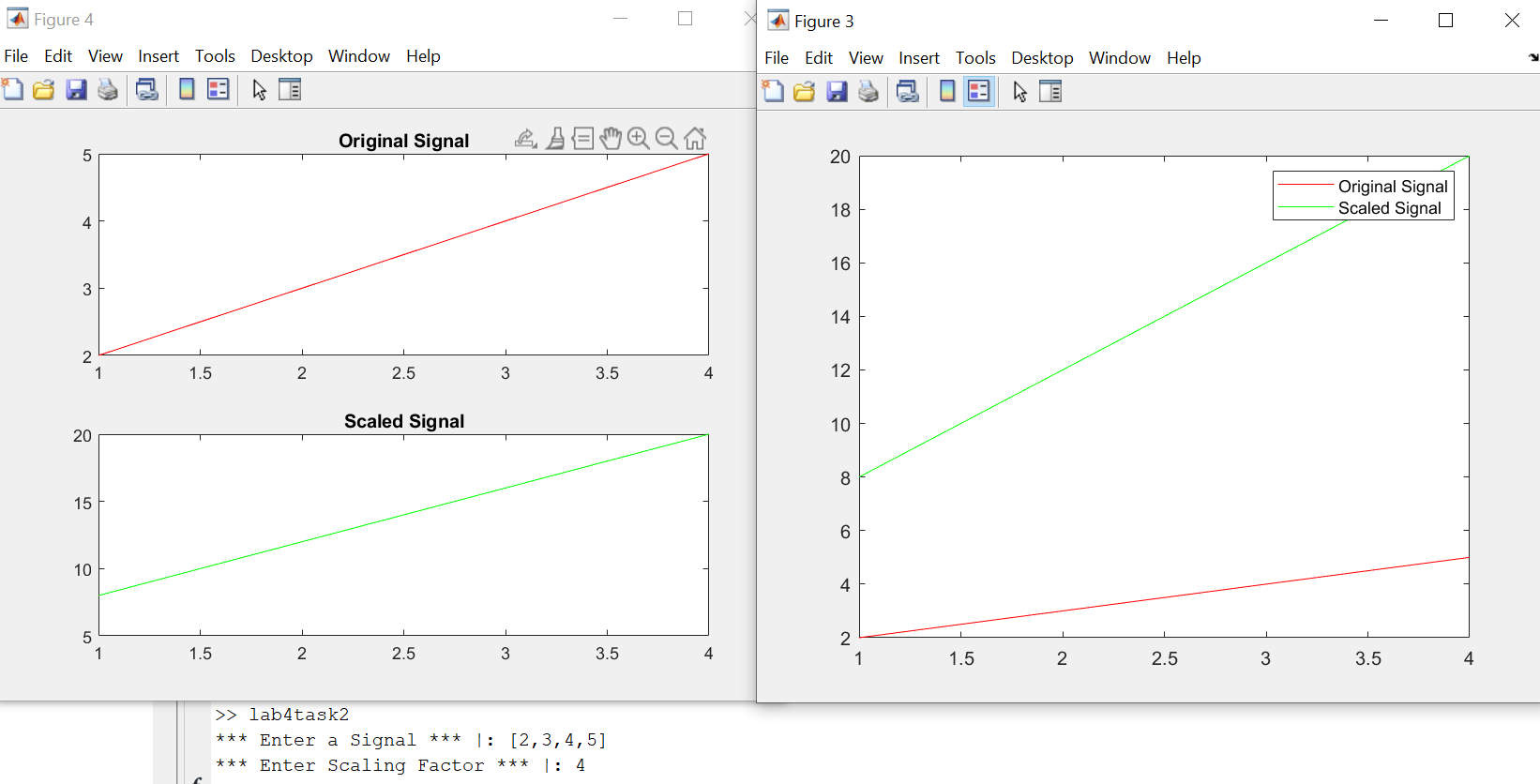
% Single Figure with separate signal plots using subplot.

**Code:**





**Output:**



**Task # 03:**

**X2[n]** 3



**X1[n]**

2

2

2

2

1

1

1

1

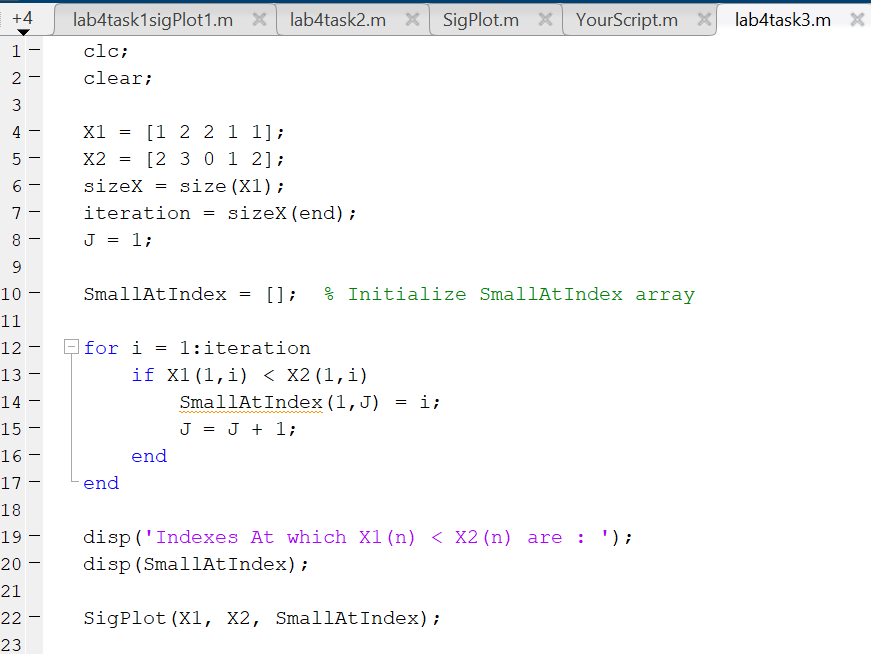
0 1 2 3 4 0 1 2 3 4

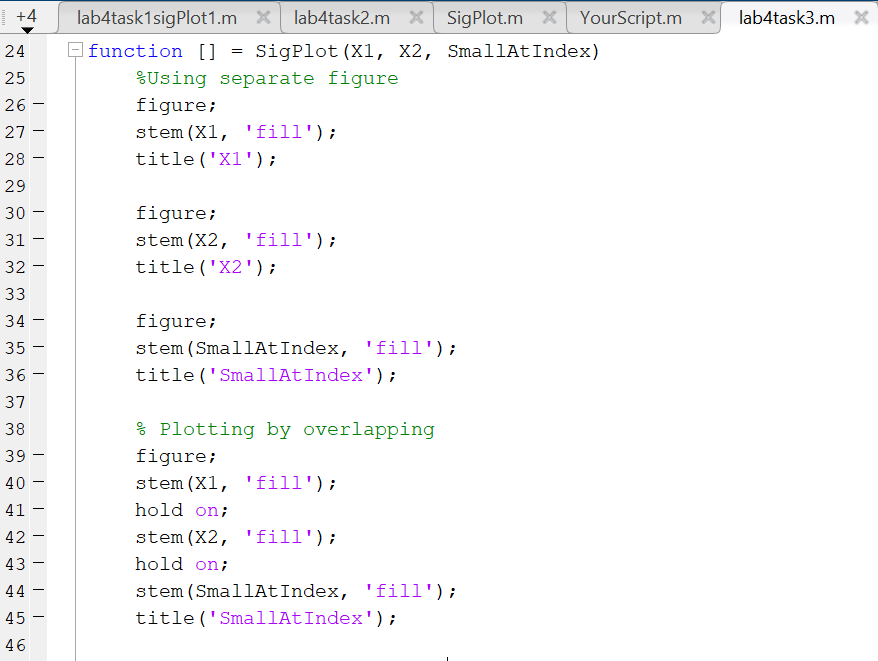
Write a Matlab program to compare the signals x1[n] and x2[n]. Determine the index where a sample of x1[n] has smaller amplitude as compared to the corresponding sample of x2[n]. Use for loop.

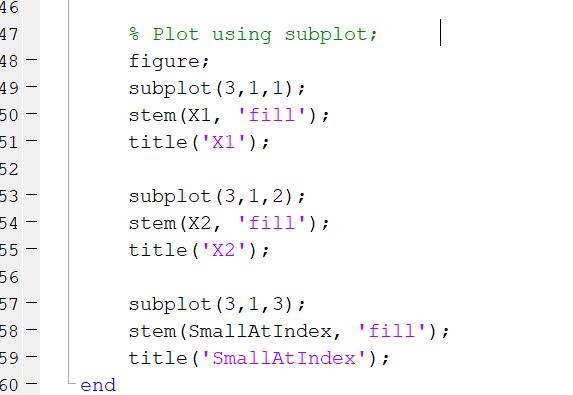
Design a function **SigPlot** that takes the original signals as input from the main calling program and plots them as:

1. Separate Figures using **stem** command,
2. Single Figure overlapping both the signals using multiple **stem** commands with **hold on**,
3. Single Figure with separate signal plots using subplots and stem commands.

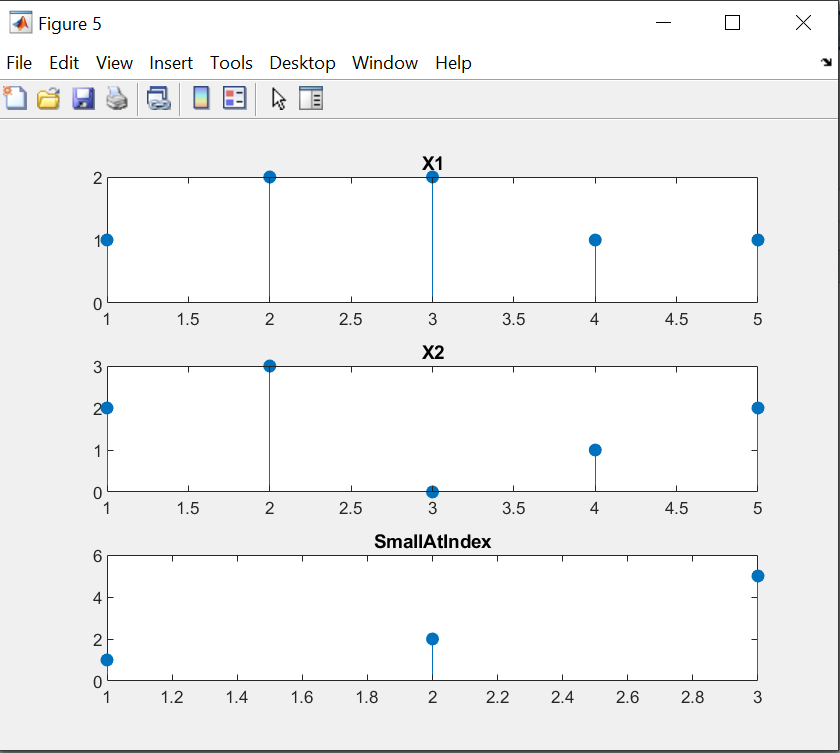
**Code:**







**Output:**



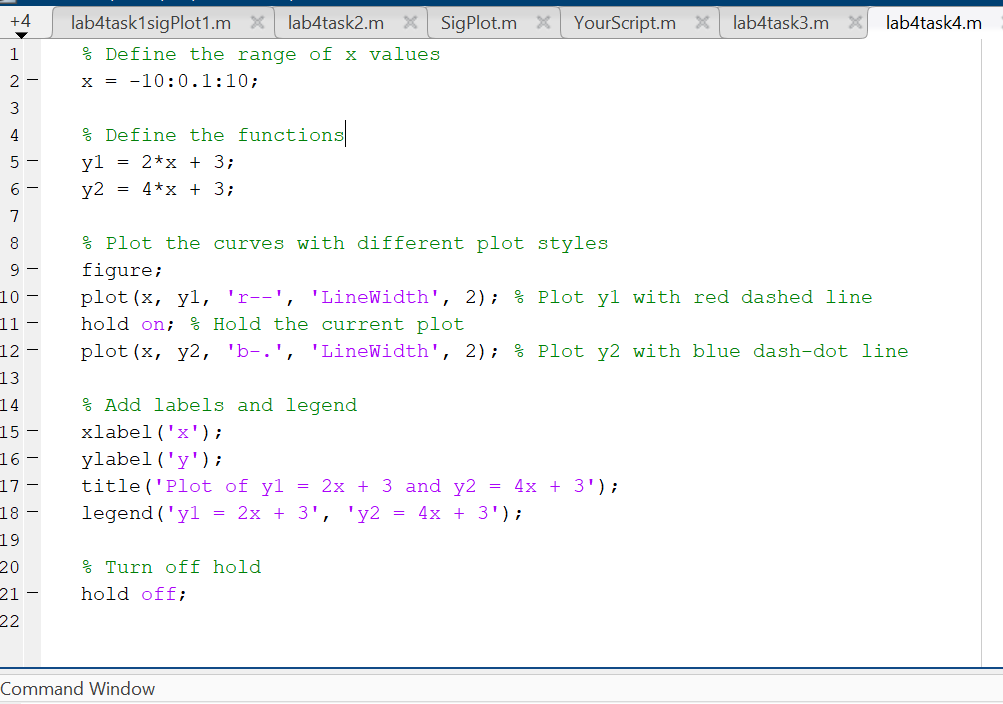
**Task # 04:**

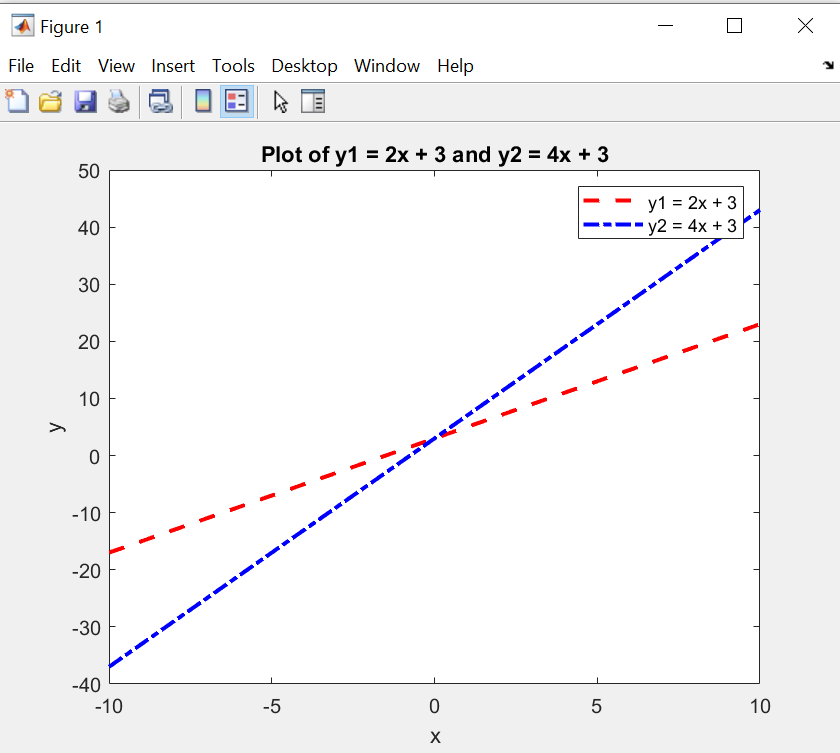
Plot the two curves **y1 = 2x .^2** and **y2 = 4x .^3** on the same graph using different plot styles.

1. Use x as a 31 entries sequence from -15:15.
2. Use x as a 101 entries sequence from -50:50.

(Hint: use plot(x,y1,'b--o',x,y2,'c\*') or same with subplots or same with multiple stems) .

**Code:**

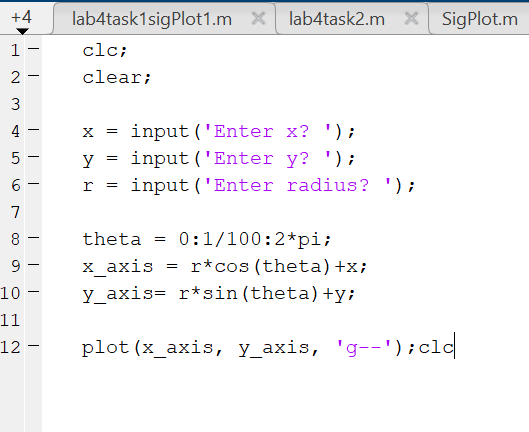


**Output:**

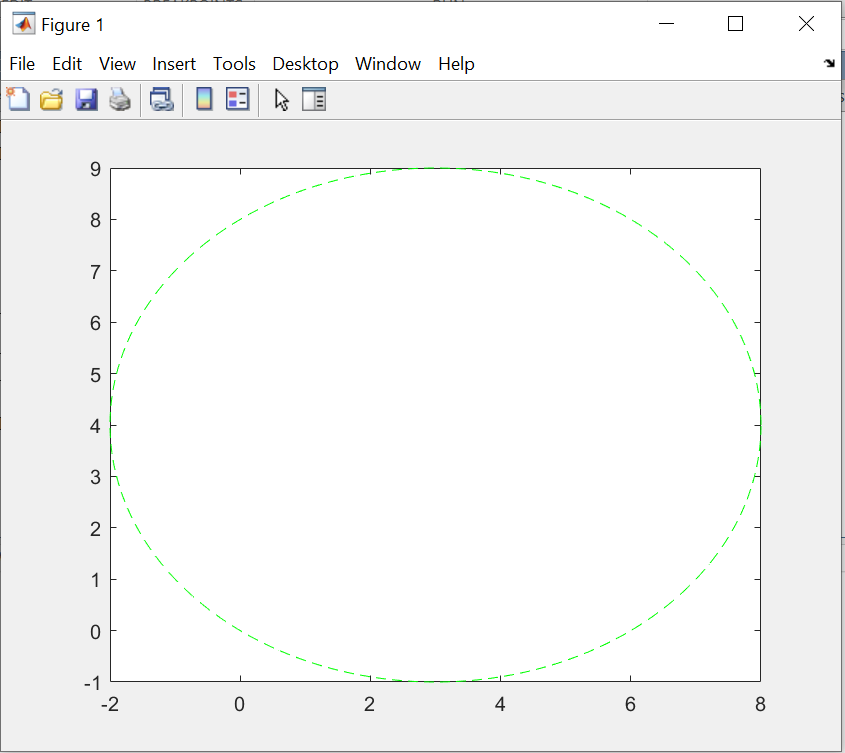
**Task # 05:**

Create a function **PlotCircle** that takes points x, y and radius r from user as inputs and generates a graph of circle centered at point (x,y) with a radius equal to r. Use **axis equal** to use equal data units along each coordinate direction and use **axis square** to view square axis. (Hint: use circle equation: x-axis = r\*cos(theta)+x; y-axis=r\*sin(theta)+y; where theta=0:1/100:2\*pi and plot x-axis versus y-axis)

**Code:**



**Output:**



**Task # 06:**

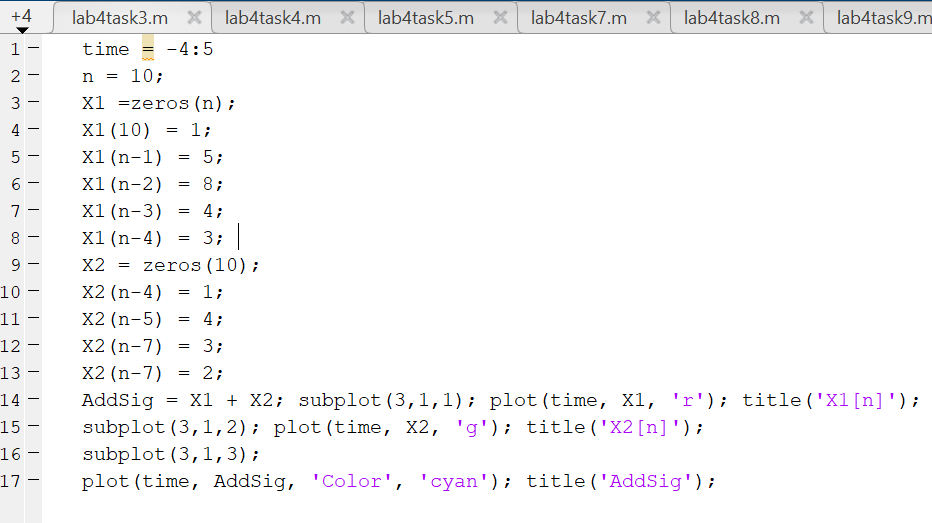
Given the signals:

X1[n] = 2δ[n] + 5δ[n‐1] + 8δ[n‐2] + 4δ[n‐3] + 3δ[n‐4]

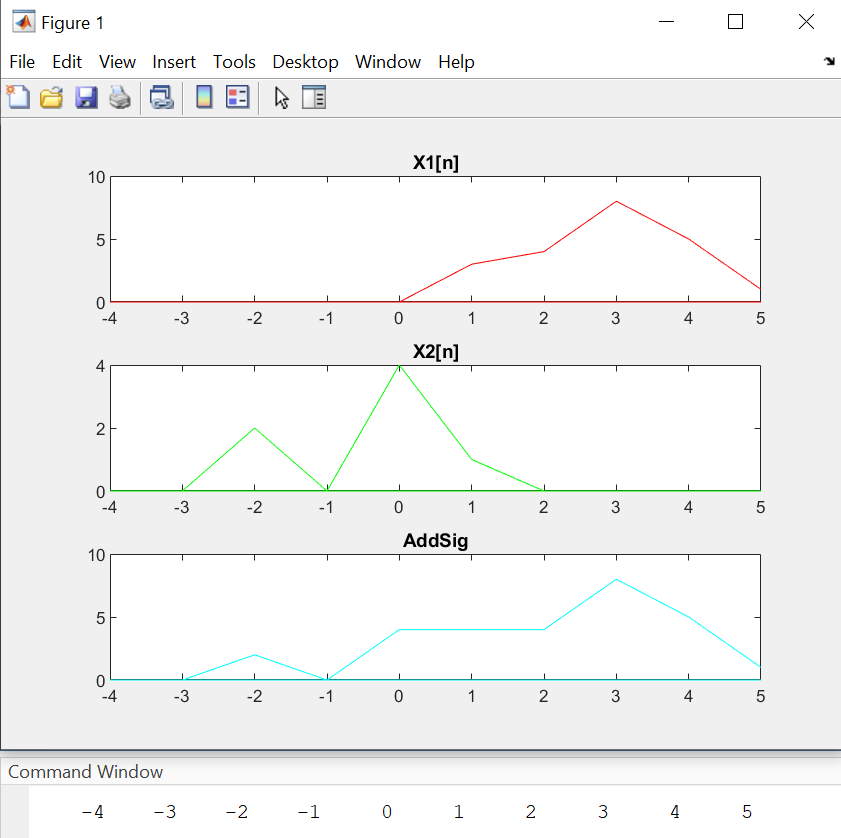
X2[n] = δ[n‐4] + 4δ[n‐5] +3δ[n‐6] + 2δ[n‐7]

Write a Matlab program that adds these two signals. Plot the original signals as well as the final results using different plotting designs.

**Code:**



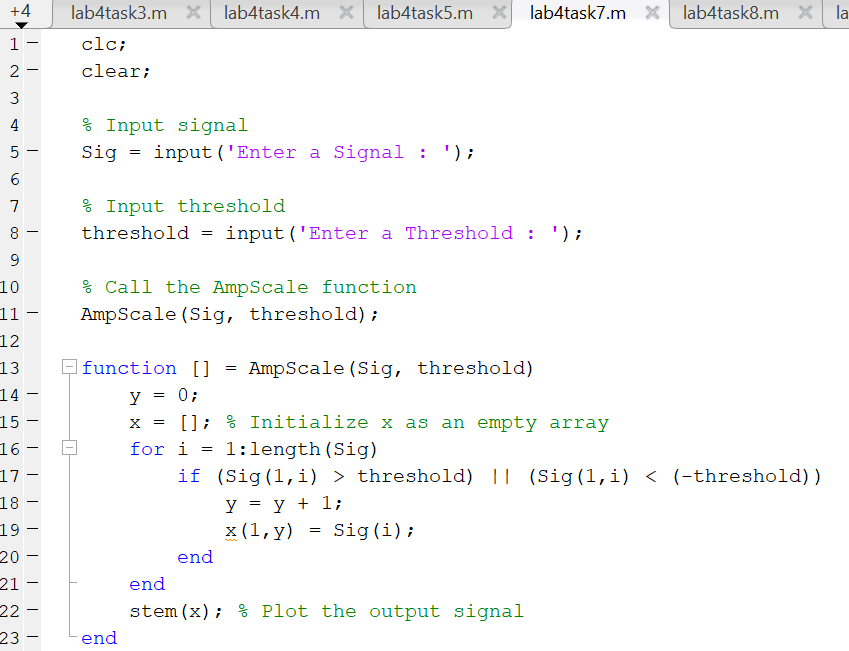
**Output:**



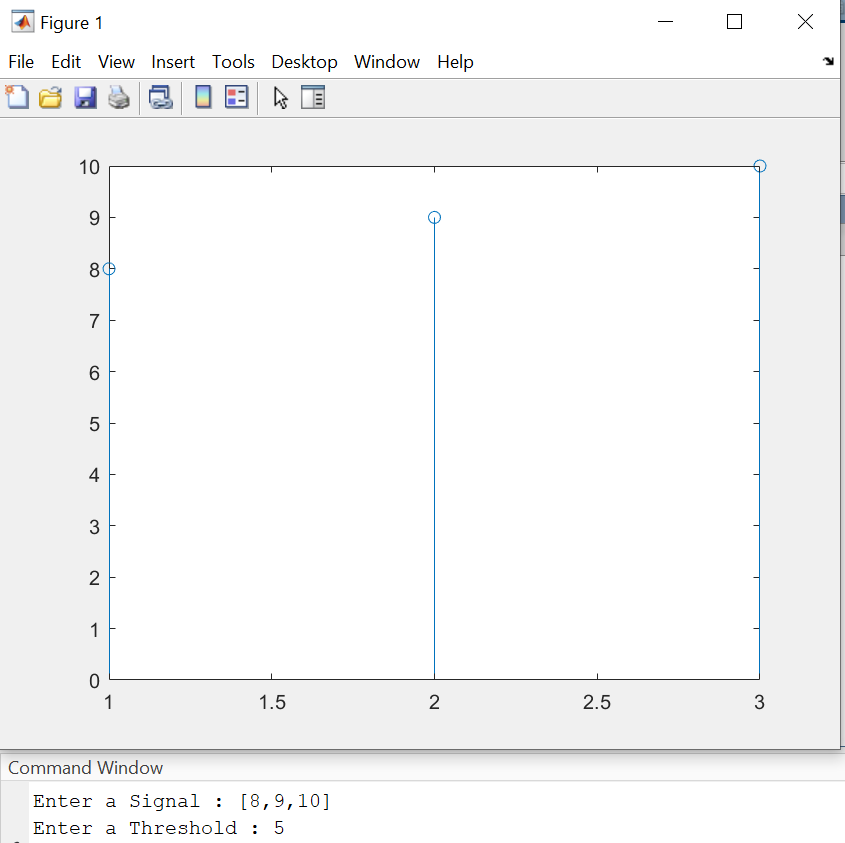
**Task # 07:**

Create a function **AmpScale** that takes a discrete‐time signal **S** and a threshold **T** from user and scales the amplitude of the input signal. The function saves and counts the number of samples with amplitude greater than **T** and less than **-T** and plots the amplitude scaled signal and gives the number of sample within the thresholds as output.

**Code:**



**Output:**



**Task # 08:**

Write your own function **downsamp** that takes a signal as input, retain odd numbered samples of the original signal and discard the even‐numbered (down sampling by 2). The function must return the down sampled version of that signal as output. See Fig for example.

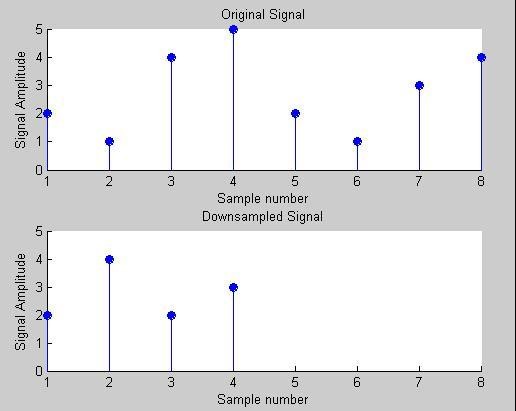
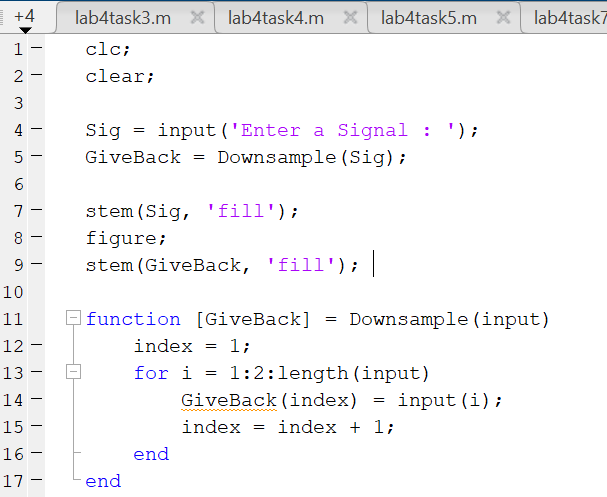


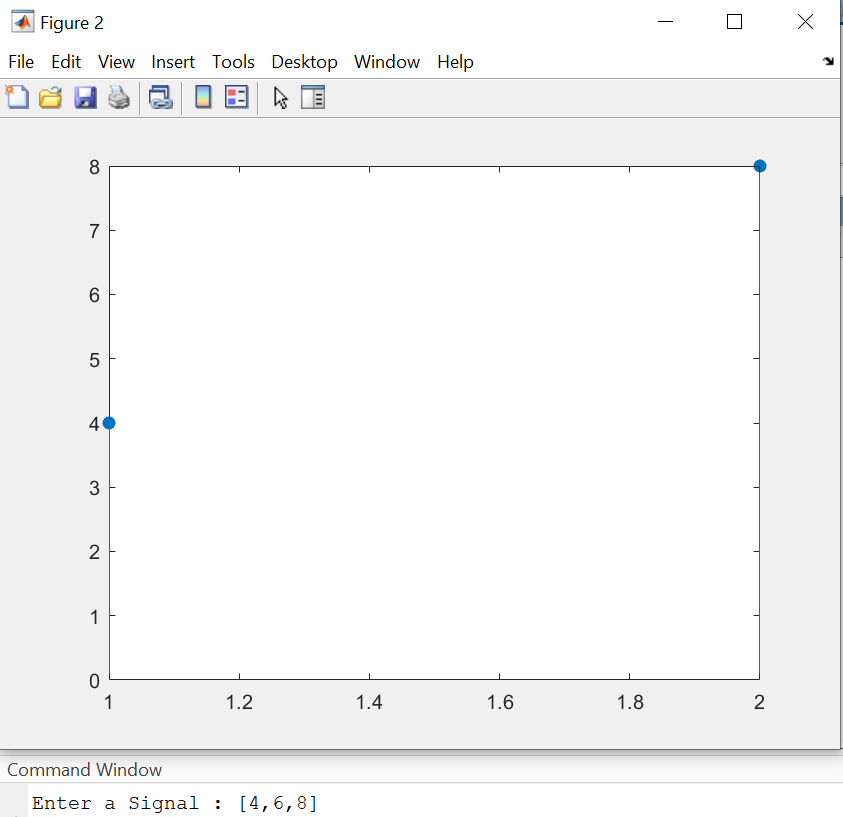
Fig. Down Sampling

1. Call this function from a matlab file. Verify your result by using the built-in command “**downsample**”. Plot the original signal, downsampled signal determined by your program, and downsampled signal obtained by the command **downsample**.
2. Modify your function to work as generic down sampling function that takes both the **input signal** and the **sampling factor** from the user. Your function must also check the possibility of down sampling by comparing the sampling rate with the number of samples in the input signal.

**Code:**



**Output:**



**Task # 09:**

Write your own function to **upsamp** a signal i.e. copy the 1st sample of original signal in the new signal and then place an extra sample of 0, copy the 2nd sample of original signal and then place a 0, and so on. See Fig for example.

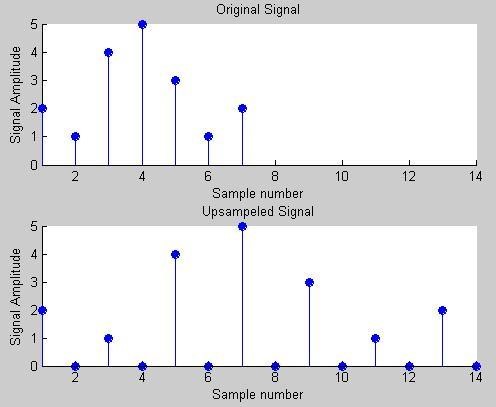
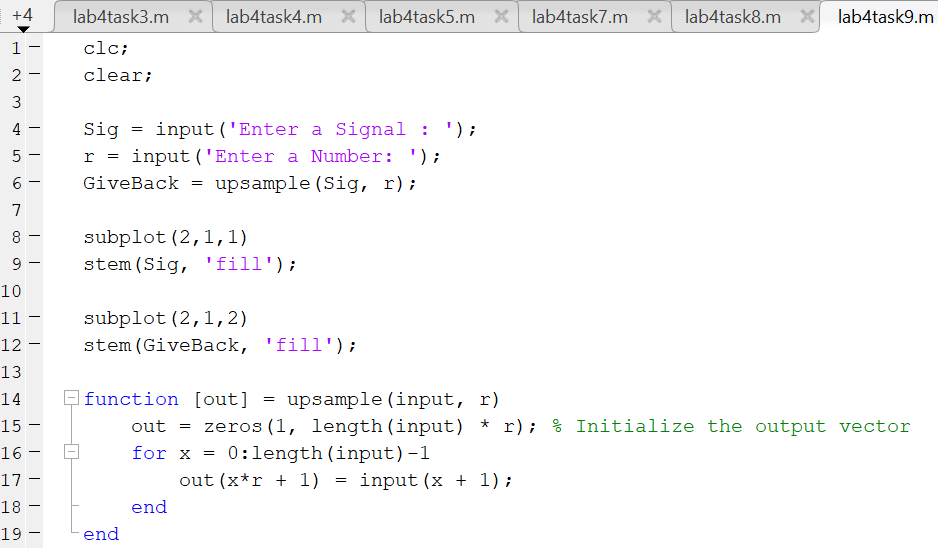


Fig. Up Sample

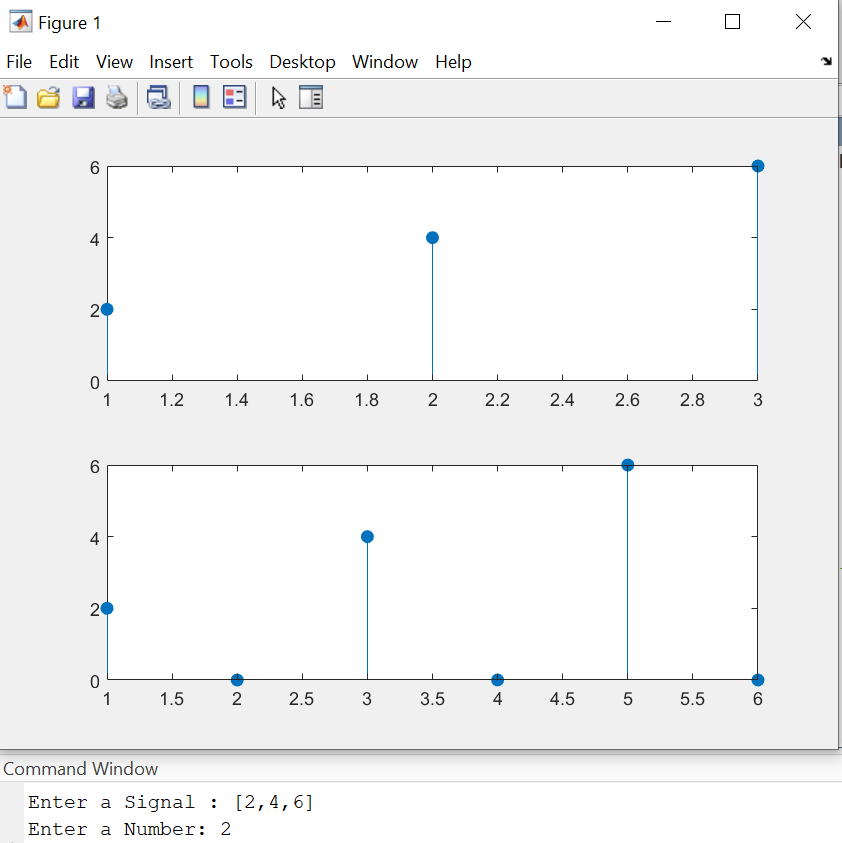
13

1. Call this function from a matlab file. Verify your result by using the built-in command “**upsample**”. Plot the original signal, upsampled signal determined by your function **upsamp**, and upsampled signal obtained by the command **upsample**.
2. Modify your function to work as generic up sampling function that takes both the **input signal** and the **sampling factor** from the user.
3. Your function must also perform other up sampling methods such as instead of **0**, the new sample is the **copy** of preceding or succeeding sample of the original signal or the new sample is the **average** of both. Check for possibility new up sampling methods by comparing the samples in the input signal.

**Code:**



**Output:**



**Task # 10:**

Plotting **3-D graphics** with MatLab. This is a complementary task for practicing 3d graphs in MatLab. **Surf** command is used in Matlab for plotting 3D graphs, the **meshgrid** command is used for setting up 2D plane

#### clear all close all

% set up 2-D plane by creating a-2:.2:2 sequence and copying it to all rows of x size(-2:.2:2) times and vice versa

**[x,y] = meshgrid([-2:.2:2]);**

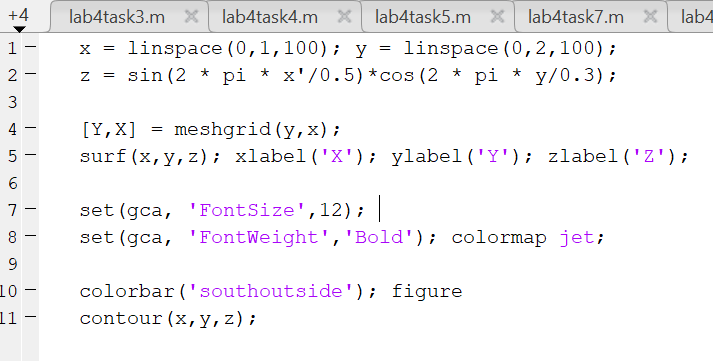
% plot 3D on plane

#### Z = x.\*exp(-x.^2-y.^2); figure

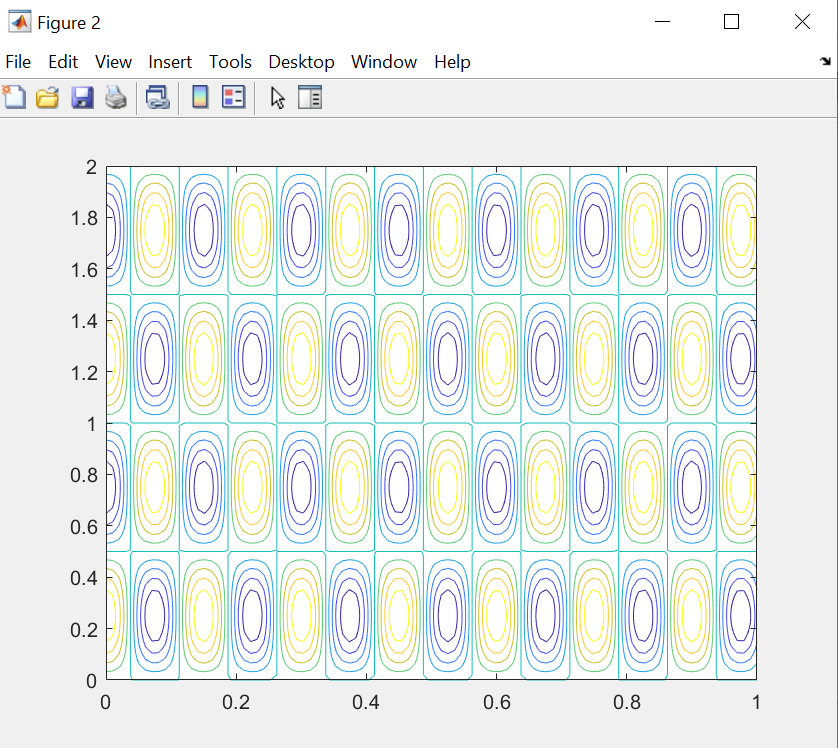
% surface plot, with gradient(Z) determining color distribution **surf(x,y,Z,gradient(Z))**

% display color scale, can adjust location similarly to legend **colorbar**

**Code:**



**Output:**



A screenshot of a computer

Description automatically generated