DISCRETE-TIME SIGNAL REPRESENTATION IN MATLAB

LAB # 04

Spring 2020

CSE301L Signals & Systems Lab

Submitted by: **Shah Raza**

Registration No. : **18PWCSE1658**

Class Section: **B**

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

Student Signature: ______________

Submitted to:

Engr. Durr-e-Nayab

Thursday, June 17, 2020

Department of Computer Systems Engineering

University of Engineering and Technology, Peshawar

Lab Objective(s):

- 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]

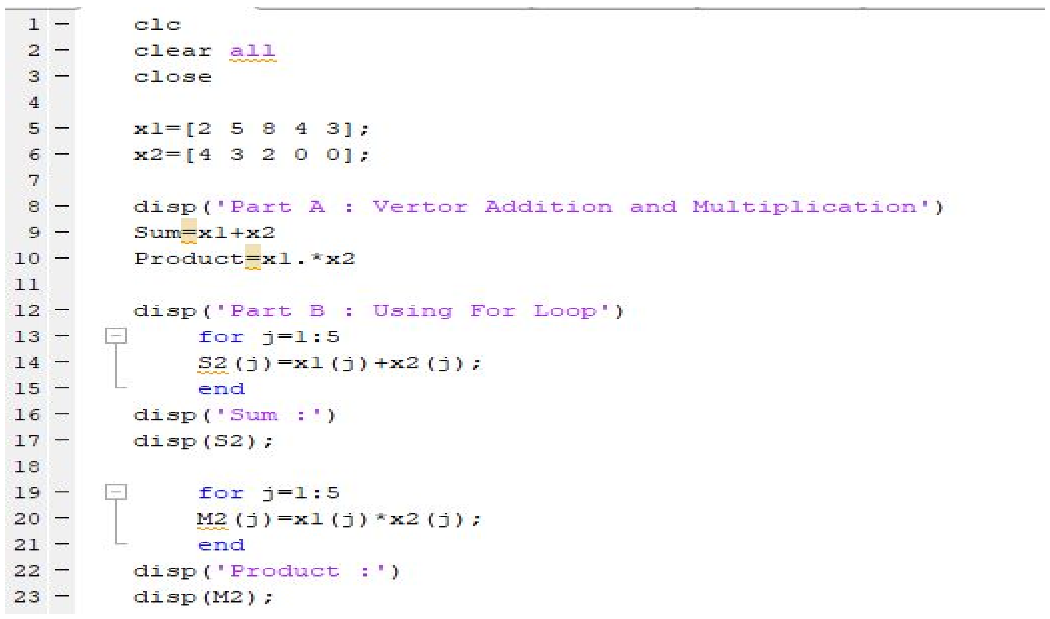
a) Write a Matlab program that adds these two signals. Use vector addition and multiplication.

b) Instead of using vector addition and multiplication, use a For loop to add and multiply the signals.

Problem Analysis:

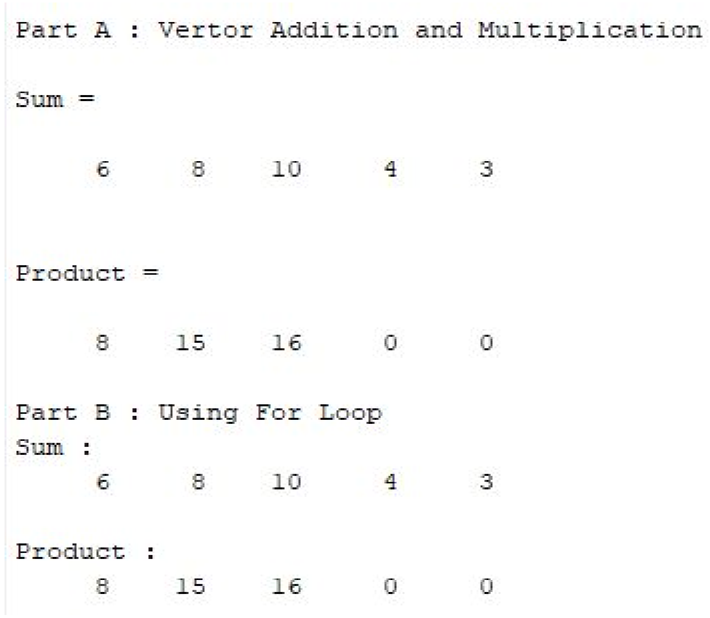
Addition or Multiplication can be done in for loops by adding or multiplying each element of a signal one by one in every iteration.

Code:



```
clc
clear all
close

x1=[2 5 8 4 3];
x2=[4 3 2 0 0];

disp('Part A : Vertor Addition and Multiplication')
Sum=x1+x2
Product=x1.*x2

disp('Part B : Using For Loop')
    for j=1:5
    S2(j)=x1(j)+x2(j);
    end
disp('Sum :')
disp(S2);

    for j=1:5
    M2(j)=x1(j)*x2(j);
    end
disp('Product :')
disp(M2);
```

Output / Graphs / Plots / Results:



```
Part A : Vertor Addition and Multiplication

Sum =

     6     8    10     4     3


Product =

     8    15    16     0     0

Part B : Using For Loop
Sum :
     6     8    10     4     3

Product :
     8    15    16     0     0
```

Task # 02:

Amplitude scaling by a factor β causes each sample to get multiplied by β. Write a user-defined function that has two input arguments: (i) a signal to be scaled and (ii) 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 the user and then call the above-mentioned function.

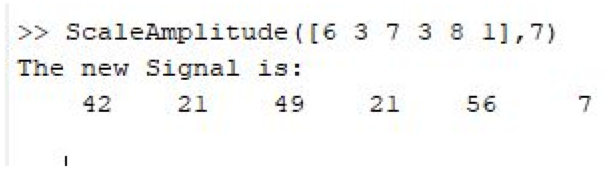
Problem Analysis:

Scaling can be done by using For loop for multiplying the scaling factor with each element of the Signal.

Code:

```
function ScaleAmplitude(Signal,F)
Size=length(Signal);

for j=1:Size

    Signal(j)=Signal(j)*F;

end

disp('The new Signal is: ');

disp(Signal);

end
```

Output / Graphs / Plots / Results:



```
>> ScaleAmplitude([6 3 7 3 8 1],7)
The new Signal is:
    42    21    49    21    56     7
```

Task # 03:

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].

Problem Analysis:

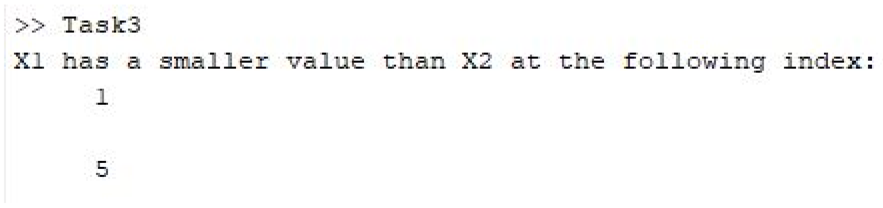
Use for loop and if condition to compare indexes of the two signals.

Code:

```
X1=[1 2 2 1 1];

X2=[3 2 0 1 2];

disp('X1 has a smaller value than X2 at the following index: ');

for i=1:5

    if(X1(i)<X2(i))
```

```
        disp(i);
    end
end
```

Output / Graphs / Plots / Results:



```
>> Task3
X1 has a smaller value than X2 at the following index:
     1

     5
```

Task # 04:

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

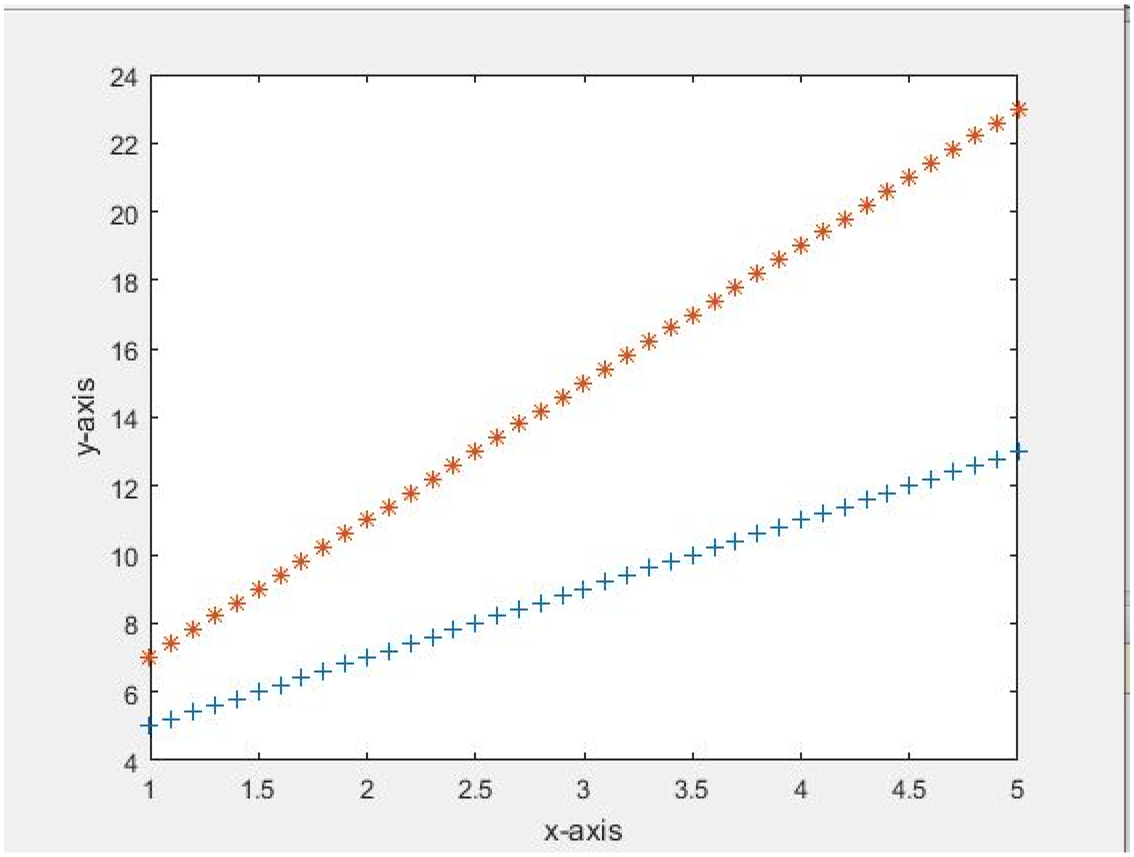
Problem Analysis:

Use plot function to plot the curves and use different plot styles.

Code:

```
x=1:0.1:5;

y1=2*x+3;

y2=4*x+3;

plot (x,y1,'+',x,y2,'*');

xlabel('x-axis');

ylabel('y-axis');
```

Output / Graphs / Plots / Results:



Task # 05:

Make two separate functions for signal addition and multiplication. The functions should take the signals as input arguments and return the resultant signal. In the main program, get the signals from the user, call the functions for signal addition and multiplication, and plot the original signals as well as the resultant signals.

Problem Analysis:

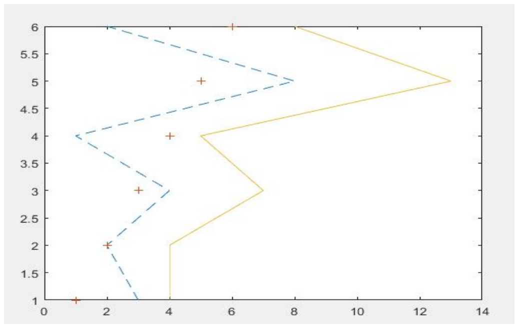
Make different functions for addition and multiplication and plot the original and resultant signals using plot function.

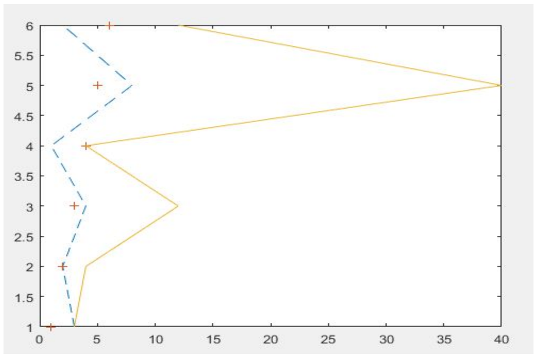
Code:

```
function AddSignals(S1,S2)           %Function for Addition

for i=1:length(S1)

Sum(i)=S1(i)+S2(i);
```

```
end
disp(Sum);
i=1:length(S1)
plot(S1,i,'--',S2,i,'+',Sum,i,'*');
end
function MulSignals(S1,S2)            %Function for Multiplication
for i=1:length(S1)
Mul(i)=S1(i)*S2(i);
end
disp(Mul);
i=1:length(S1)
plot(S1,i,'--',S2,i,'+',Mul,i,'*');
end
```

Output / Graphs / Plots / Results:





Task # 06:

Given the signals:

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

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

Write a Matlab program that adds these two signals. Plot the original signals as well as the final result.

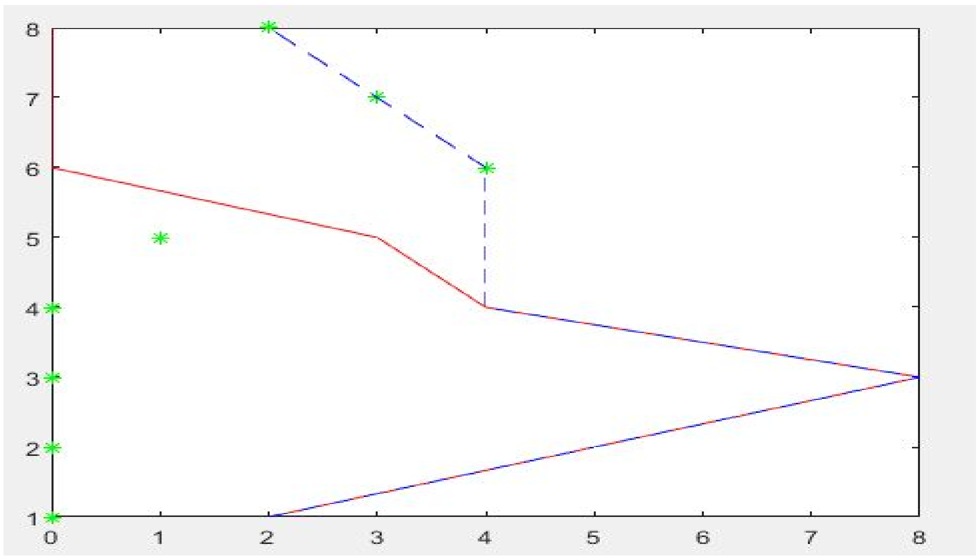
Problem Analysis:

Add extra values to both the Signals in order to balance them and then use for loop to add them together.

Code:

```
X1=[2 5 8 4 3 0 0 0]

X2=[0 0 0 0 1 4 3 2]

for i=1:8

    X(i)=X1(i)+X2(i);

end

disp('Result is: ')

disp(X);

i=1:8;

plot(X1,i,'-r',X2,i,'*g',X,i,'--b')
```

Output / Graphs / Plots / Results:



Task # 07:

Take a discrete-time signal from the user. Count the number of samples with amplitude greater than a threshold of 3 and less than a threshold of -3.

Problem Analysis:

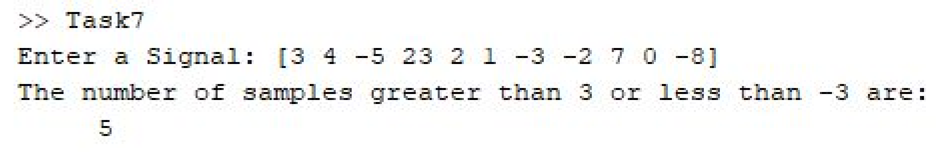
Use a for loop to check all samples and if condition to filter out the samples that are greater than 3 or less than -3.

Code:

```
X=input('Enter a Signal: ');
```

```
count=0;

for i=1:length(X)

    if(X(i)>3 | X(i)<-3)

        count=count+1;

    end

end

disp('The number of samples greater than 3 or less than -3 are: ')

disp(count);
```

Output / Graphs / Plots / Results:



```
>> Task7
Enter a Signal: [3 4 -5 23 2 1 -3 -2 7 0 -8]
The number of samples greater than 3 or less than -3 are:
     5
```

Task # 08:

Write your own function to down-sample a signal i.e. retain odd numbered samples of the original signal and discard the even-numbered (down-sampling by 2). The function must take a signal as input and return the down-sampled version of that signal.

Problem Analysis:

Use if condition to remove the even-numbered samples and store the remaining odd samples in a new array.

Code:

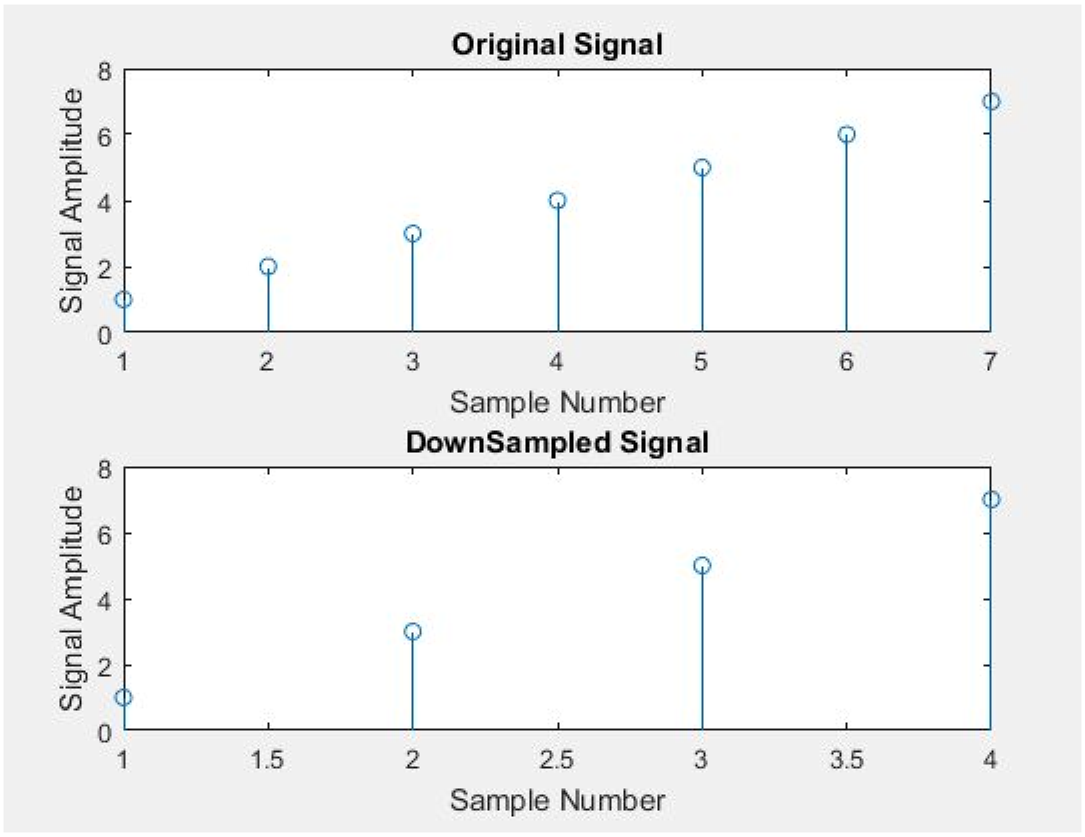
Function File:

```
function DS=Downsample(S)
j=1;
for i=1:length(S)
    if(mod(i,2)~=0)
        DS(j)=S(i);
         j=j+1;
    end
end
end
```

Main File:

```
clear all
S = input('Enter a Signal: ');
DS = Downsample (S);

disp(DS);
%figure
subplot(2,1,1)
stem(S);
title('Original Signal')
xlabel('Sample Number')
ylabel('Signal Amplitude')

%figure2
subplot(2,1,2)
stem(DS);
title('DownSampled Signal')
xlabel('Sample Number')
ylabel('Signal Amplitude')
```

Output / Graphs / Plots / Results:



Task # 09:

Write your own function to up-sample 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.

Problem Analysis:

Use if else condition to separate even and odd numbered samples, shift all the signals one place up and place zeros in even numbered indexes.

Code:

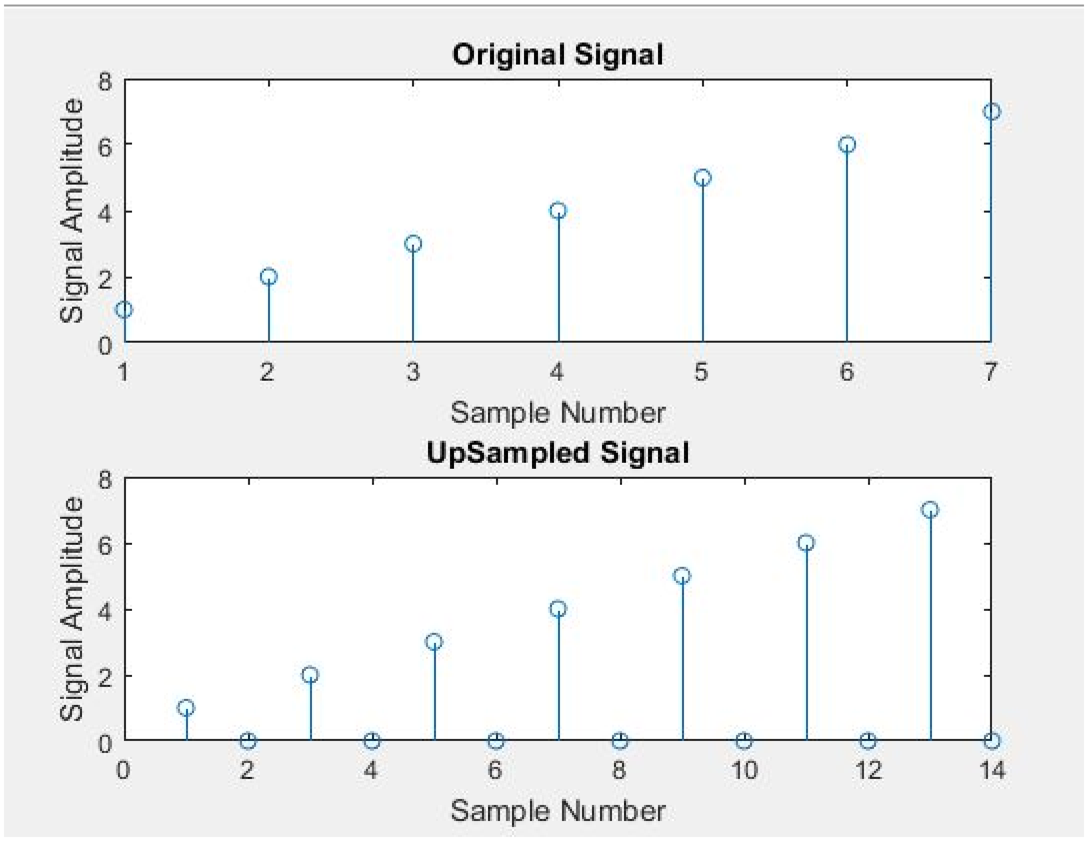
Function File:

```
function US=Upsample(S)

j=1;

for i=1:2*length(S)

    if(mod(i,2)==0)

       US(i)=0;

    else

       US(i)=S(j);

       j=j+1;

    end

end
```

Main File:

```
clear all

S = input('Enter a Signal: ');

US = Upsample (S);


disp(US);

%figure

subplot(2,1,1)

stem(S);

title('Original Signal')

xlabel('Sample Number')

ylabel('Signal Amplitude')

subplot(2,1,2)

stem(US);

title('UpSampled Signal')

xlabel('Sample Number')

ylabel('Signal Amplitude')
```

Output / Graphs / Plots / Results:


Original Signal
Signal Amplitude
Sample Number
UpSampled Signal
Signal Amplitude
Sample Number