National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Department of Electrical Engineering

Faculty Member: Dr. Wajahat Hussain **Dated:** 14-2-2107

Course/Section: BEE6-B **Semester:** 6th Semester

EE-330 Digital Signal Processing

Lab1: MATLAB REVIEW-Signals & Systems Fundamentals

Name	Reg. no.	Report Marks / 10	Lab Quiz-Viva Marks / 5	Total / 15
Saad Iqbal	111394			
Usman Iqbal	111393			
Abdullah Bin Asif	111596			

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Lab1: MATLAB REVIEW-Signals & Systems Fundamentals

Objectives
The purpose of this lab is to review the fundamentals of signals and systems with MATLAB, particularly:

- ✓ Signal transformations (shifting, inversion, scaling)
- ✓ Even and Odd parts of a signal
- ✓ Convolution operator-the basic property of Linear Time Invariant (LTI) Systems
- ✓ Periodization of a finite duration signal using convolution operator

1.1 Matrices/vectors in MATLAB

a) Make sure that you understand the **colon** notation. In particular, explain in words what the following MATLAB code will produce

```
jkl = 0 : 6;
jkl = 2 : 4 : 17;
jkl = 99 : -1 : 88;
ttt = 2 : (1/9) : 4;
tpi = pi * [ 0:0.1:2 ];
```

Comments:

1. jkl contains a 1D array from 0 to 6 with step size 1.
2. jkl contains a 1D array from 2 to 17 with step size 4.
3. jkl contains a 1D array from 99 to 88 with step size -1.
4. ttt contains a 1D array from 2 to 4 with step size (1/9).
5. tpi contains a 1D array scalar multiplied with pi, whereas 1D array from 0 to 2 with step size 0.1.

b) Extracting and/or inserting numbers into a vector is very easy to do. Consider the following definition of xx:

```
xx = [zeros(1,3), linspace(0,1,5), ones(1,4)];
[s1 s2] = size(xx);
s3 = length(xx);
```

Explain the results echoed from the last four lines of the above code.

Comments:

1. xx contains {0,0,0,0,0.25,0.5,0.75,1,1,1,1,1}
2. s1 contains no. of rows and s2 contains no. of column.
3. s3 contains length of largest dimension of xx.

What's the difference between a length and a size statement for a matrix? To test this define a matrix X with arbitrary inputs, having multiple rows and columns and test the output of length() and size() function on it.

Comments:

length() function returns the length of largest dimension of matrix whereas size() function returns both no. of rows and column of the matrix.

c) Assigning selective values in a matrix differently. Comment on the result of the following assignments:

```
yy = xx;
yy(4:6) = pi*(1:3);
```

Comments:

In first command, content of variable xx copies into yy.
In second command, content of indexes 4 to 6 of yy is replace by pi, 2*pi, 3*pi respectively.

1.2 Creating a M-file

Go to File > New > M–file. MATLAB editor will open up. Enter the following code in the editor and then save the file as Namelab1.m

```
tt = -1 : 0.01 : 1;
xx = cos( 5*pi*tt );
zz = 1.4*exp(j*pi/2)*exp(j*5*pi*tt);
plot( tt, xx, 'b-', tt, real(zz), 'r--' ), grid on
%<--- plot a sinusoid
title('TEST PLOT of a SINUSOID')
xlabel('TIME (sec)')
```

Now go to Command Window and type

Output:

mylab1 %<---will run the commands in the file

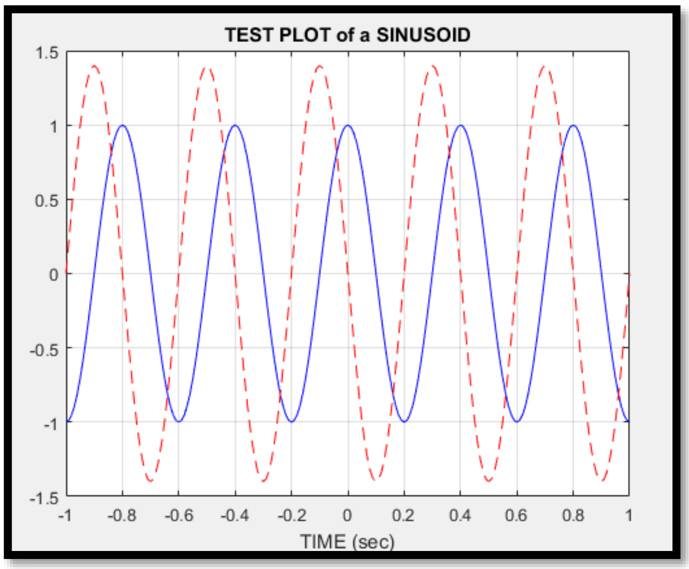
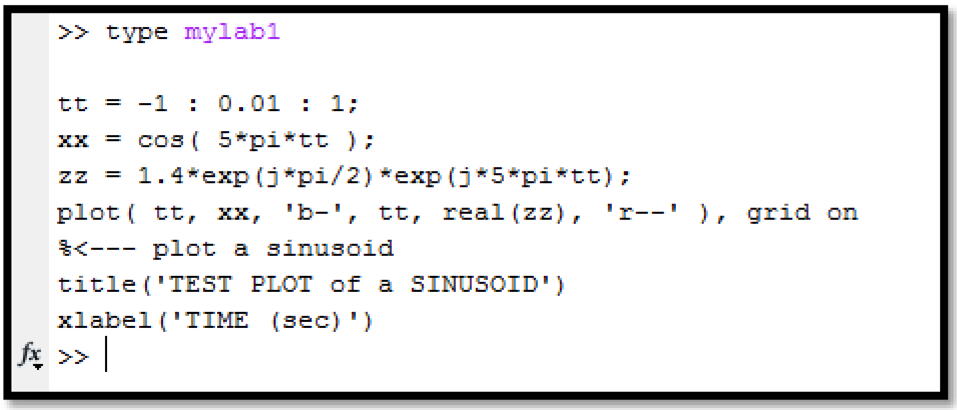


Figure No.1

type mylab1 %<---will type out the contents of
% mylab1.m to the screen



```
>> type mylab1

tt = -1 : 0.01 : 1;
xx = cos( 5*pi*tt );
zz = 1.4*exp(j*pi/2)*exp(j*5*pi*tt);
plot( tt, xx, 'b-', tt, real(zz), 'r--' ), grid on
%<--- plot a sinusoid
title('TEST PLOT of a SINUSOID')
xlabel('TIME (sec)')
>>
```

Figure No.2

1.3 Functions-Key to Efficient Coding

It is often convenient to define functions so that they may use at multiple instances and with different inputs. Functions are a special type of M-file that can accept inputs (matrices and vectors) and may return outputs. The keyword ***function*** must appear as the first word in the M-file that defines the function, and the first line of the M-file defines how the function will pass input and output arguments. The file extension must be lower case "***m***" as in my ***func.m***. The following function has a few mistakes. Before looking at the correct one below, try to find these mistakes (there are at least three):

```
Matlab mfile [xx,tt] = badcos(ff,dur)
%BADCOS Function to generate a cosine wave
% xx = badcos(ff,dur)
% ff = desired frequency in Hz
% dur = duration of the waveform in seconds
tt = 0:1/(100*ff):dur; %-- gives 100 samples per period
badcos = cos(2*pi*freeq*tt);
```

The corrected function should look something like:

```
function [xx,tt] = goodcos(ff,dur)
tt = 0:1/(100*ff):dur; %-- gives 100 samples per period
xx = cos(2*pi*ff*tt);
```

Comments:

Notice the word "function" in the first line. Also, "freeq" has not been defined before being used. Finally, the function has "xx" as an output and hence "xx" should appear in the left-hand side of at least one assignment line within the function body. The function name is *not* used to hold values produced in the function.

1.4 Review of Basic Signals and Systems

a) Even and odd parts of a signal:

Any signal ***x[n]*** can be decomposed into its even part and odd parts as:

$$x_e(n) = \frac{1}{2}[x(n) + x(-n)]$$
$$x_0(n) = \frac{1}{2}[x(n) - x(-n)]$$

Write a simple MATLAB code (in the form of a function) that allows you to decompose a signal into its even and odd parts.
Note: The function takes two inputs ***n***, the timing index and ***x*** the values of the signal at the designated time instants. The function outputs include the two sub-functions, x_e and x_o along with the timing index.

Test your function on the following signal ***x[n]*** and compute it's even and odd parts.

$$x[n] = \begin{cases} 2 & n=0 \\ 3 & n=1 \\ -1 & n=2 \\ 2 & n=3 \\ -3 & n=4 \\ 0 & \text{elsewhere} \end{cases}$$

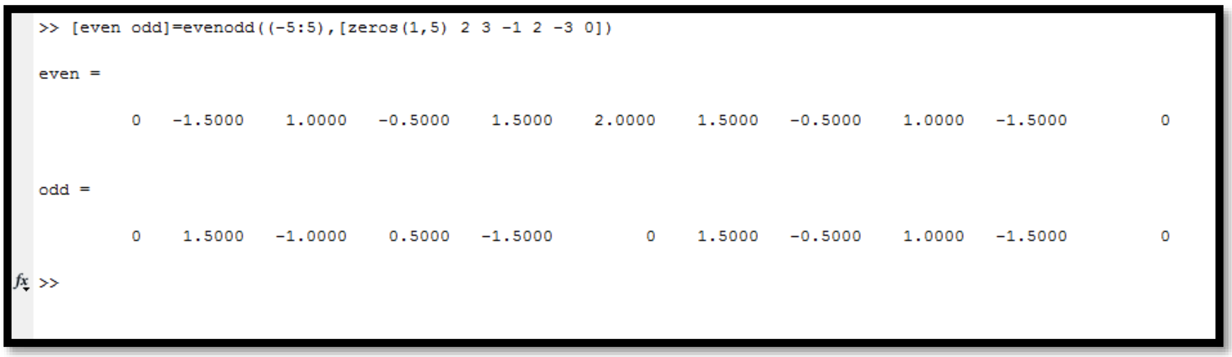
Solution:

Matlab Code:

```
function [even odd] =evenodd(in_indexes, values)
%Check
if(max(in_indexes)==abs(min(in_indexes)) &&
length(in_indexes)== 2*max(in_indexes)+1)
for i=1:length(in_indexes)
odd(i)= (values(i)-values(length(in_indexes)+1-i))/2;   %odd
part
even(i)= (values(i)+values(length(in_indexes)+1-i))/2;  %even
part
end
%% Ploting
subplot(3,1,1)
stem(in_indexes,values)
title('Original')


subplot(3,1,2)
stem(in_indexes,even)
```

```
title('Even part')
subplot(3,1,3)
stem(in_indexes,odd)
title('Odd part')

else
    disp('Inputs are invalid')
end
```

Command window:



```
>> [even odd]=evenodd((-5:5),[zeros(1,5) 2 3 -1 2 -3 0])

even =

         0   -1.5000    1.0000   -0.5000    1.5000    2.0000    1.5000   -0.5000    1.0000   -1.5000         0


odd =

         0    1.5000   -1.0000    0.5000   -1.5000         0    1.5000   -0.5000    1.0000   -1.5000         0

>>
```

Figure No.3

Matlab graph:

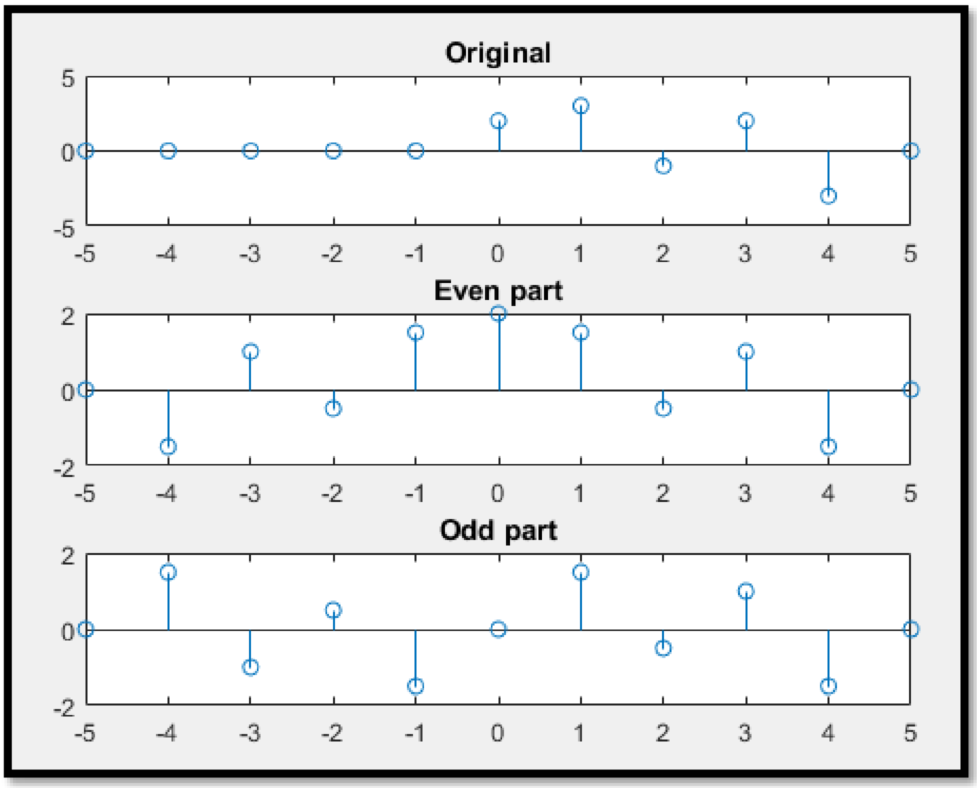


Figure No.4

b) First order Difference equation:

Recall that one way of defining the LTI systems is through the difference equations that relate the input ***x[n]*** to the output ***y[n]***.

Consider the first order system defined by the difference equation as follows (we'll review the discussion on how determination of order for a difference equation later):

$$y[n] = a.\, y[n-1] + x[n]$$

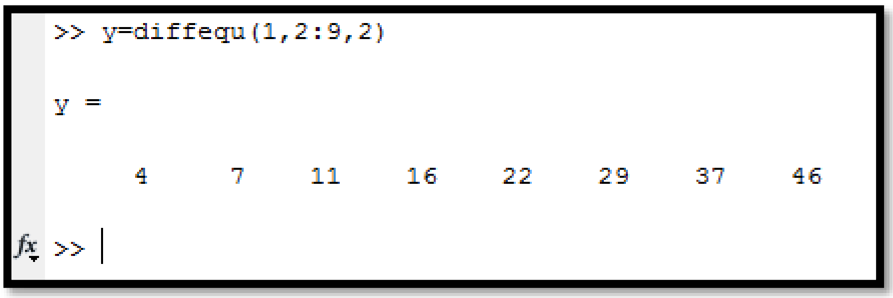
Write a function ***y*** **=** ***diffeqn*** **(*a, x, y[-1]*)** which computes the output ***y[n]*** of the system determined by the given equation. The vectors ***x[n]*** contains the signal as defined in the upper part and ***y[n] = 0 for n < 1***.

Solution:

Matlab Code:

```
function [y]=diffequ(a,x,i) %x starting from zero index
for m=1:length(x) %implement difference function
    y(m)=a*i+x(m);
    i=y(m);
    end
end
```

Command window:



```
>> y=diffequ(1,2:9,2)

y =

     4     7    11    16    22    29    37    46

>> 
```

Figure No.5

c) Convolution of signals

Recall that one the most convenient ways to represent an LTI system is through its impulse response ***h[n]***. Once the impulse response of a system is known, the output (response) of the system to any given input can be computed using the convolution operator as:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]$$

The convolution essentially involves two operations: flipping either the input signal or the impulse response (as in above equation) and then sliding the flipped signal.

i. Write your own convolution function, ***myconv.m*** that computes the convolution between the two signals (or the output of passing an input signal through a system). Designate all the necessary inputs for your function, considering that the input signal and the impulse response may start at some '***n***' that is negative. The function output is obviously the system output along with the timing index for the output ***n1***, which must be set manually. Your function should work on any general signal and the impulse response (of finite length).

Code:

```
function [yn y]=convc(xn,x,hn,h)
%check
if(length(xn)==length(x) && length(hn)==length(h))
min_yn= xn(1)+hn(1);
l_yn= length(xn)+length(hn)-1;
yn=min_yn:min_yn+l_yn-1;
xt=[x zeros(1,length(yn)-length(x))];
ht=[h zeros(1,length(yn)-length(h))];

for i=1:length(yn) %convolution
    y(i)= sum(xt(1:1:i).*ht(i:-1:1));
end
else
    disp('Invalid inputs');
end
```

ii. Test your function on the signal and the impulse response provided in the figures below and verify the correctness of your function through a comparison of manual computation of the convolution for the given signal and a plot of your function's output.

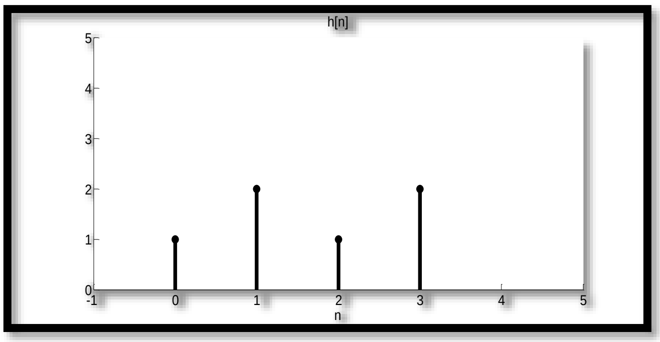


Figure No.6

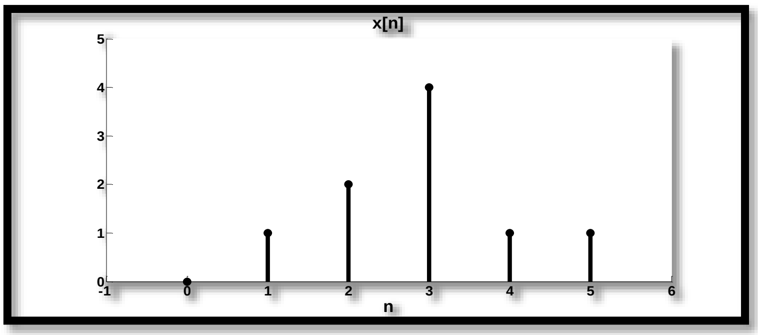
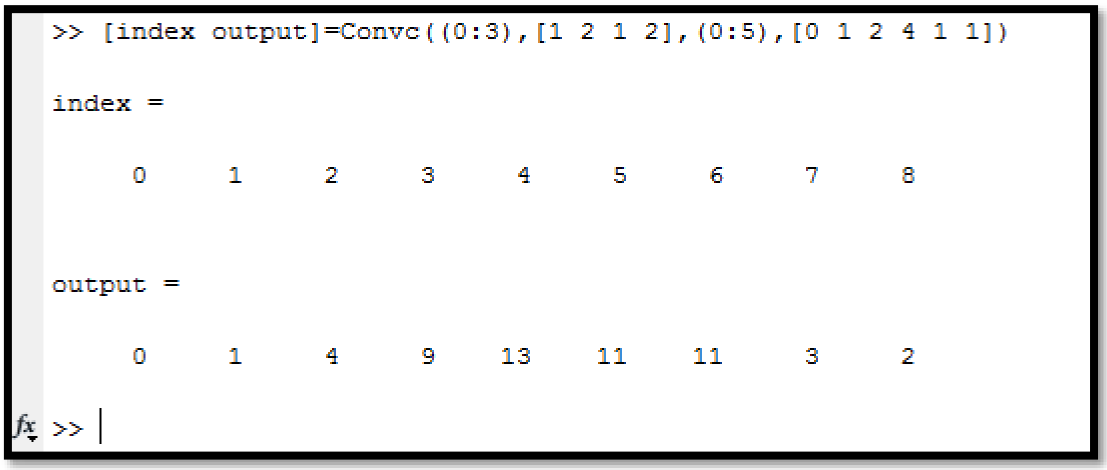


Figure No.7

Command window:

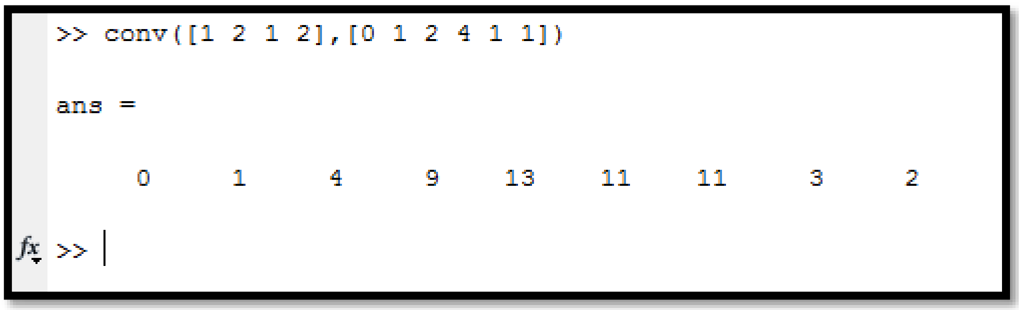


```
>> [index output]=Convc((0:3),[1 2 1 2],(0:5),[0 1 2 4 1 1])

index =

     0     1     2     3     4     5     6     7     8


output =

     0     1     4     9    13    11    11     3     2

>> 
```

Figure No.8

This verifies the manual results.

iii. MATLAB has a built-in function 'conv' that performs the same operation. Compare the results of part (ii) with the conv function of MATLAB.



```
>> conv([1 2 1 2],[0 1 2 4 1 1])

ans =

     0     1     4     9    13    11    11     3     2

>> 
```

Figure No.9

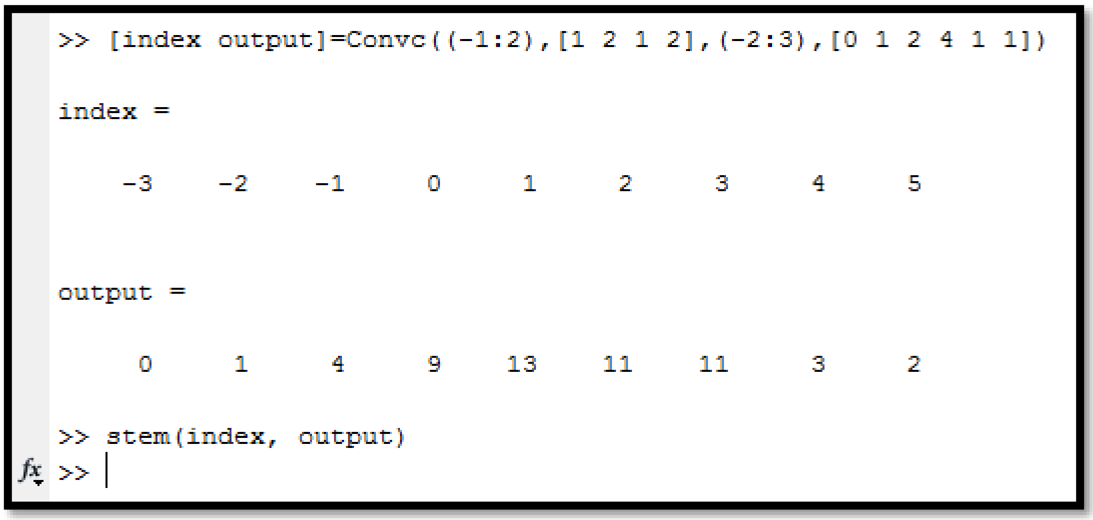
Both gives the same result.

iv. Consider now that x[n] starts from n = -1 and h[n] starts from -2. What will be the result of convolution then? Plot the corresponding output signal using the stem command and proper timing axis.

Comments:

Matlab default conv function does not give any information regarding timing axis. So, it's our duty to set timing axis after using conv function.
In our custom convolution function, function itself take care of timing axis.

Command window:



```
>> [index output]=Convc((-1:2),[1 2 1 2],(-2:3),[0 1 2 4 1 1])

index =

    -3    -2    -1     0     1     2     3     4     5


output =

     0     1     4     9    13    11    11     3     2

>> stem(index, output)
>> 
```

Figure No.9

Matlab graph:

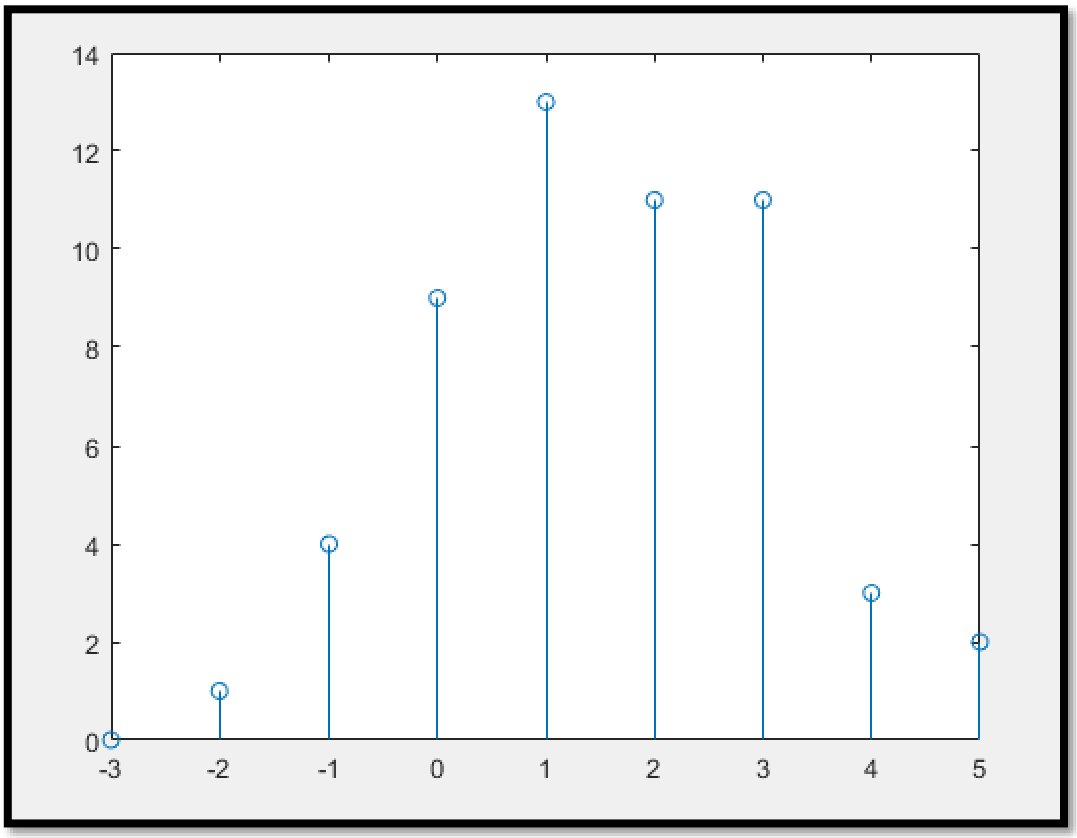
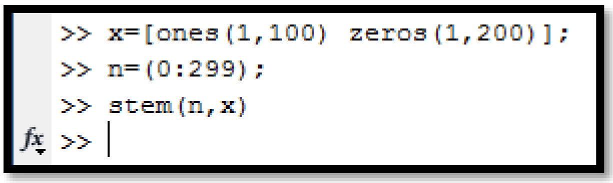


Figure No.10

d) Convolution of signal with an impulse train

i. Define ***x[n]*** as a very simple signal starting at ***n=0*** and containing a sequence of ones in its first 100 samples and a sequence of zeros in its next 200 samples. You may like to use the MATLAB functions ***zeros()*** and ***ones()***.

Command window:



```
>> x=[ones(1,100) zeros(1,200)];
>> n=(0:299);
>> stem(n,x)
>>
```

Figure No.11

ii. Plot this discrete time signal (use ***stem***) as a function of the timing variable ***n***.

Matlab graph:

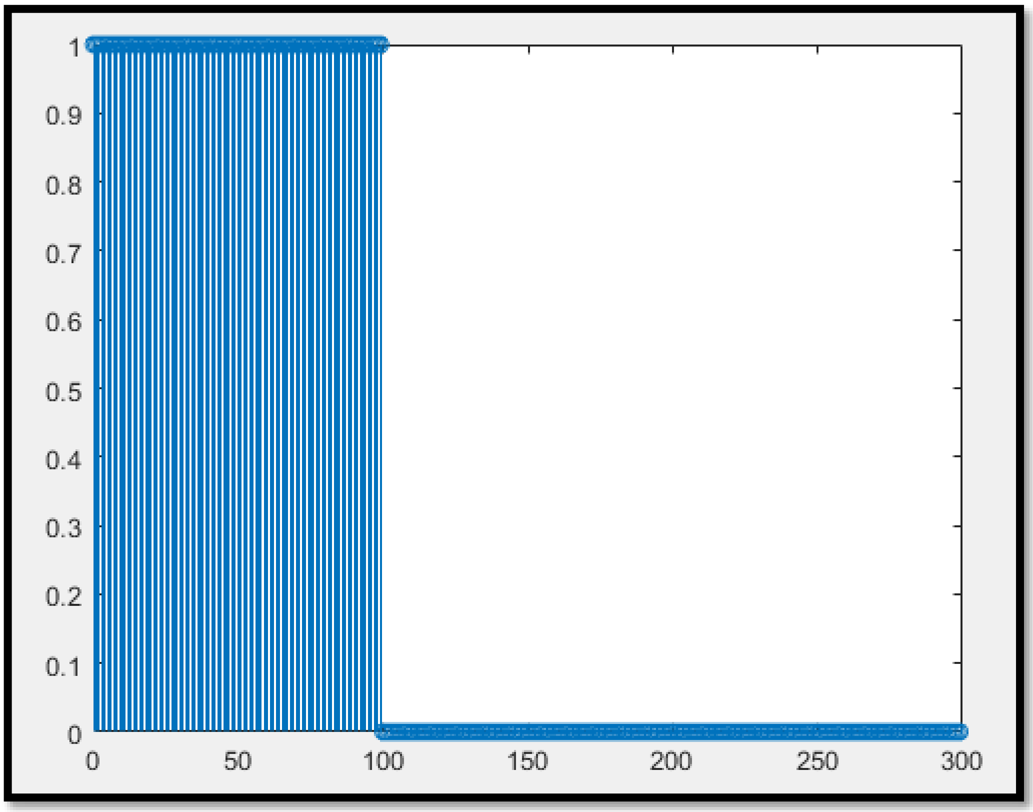


Figure No.12

Now consider an impulse train defined mathematically as:

$$\delta_N[n] = \sum_{k=-\infty}^{\infty} \delta[n-kN],$$

where ***N*** represents the period of this impulse train. A particular example of this train is shown below for ***N* = *3*** and for ***k* =*-3:3***.

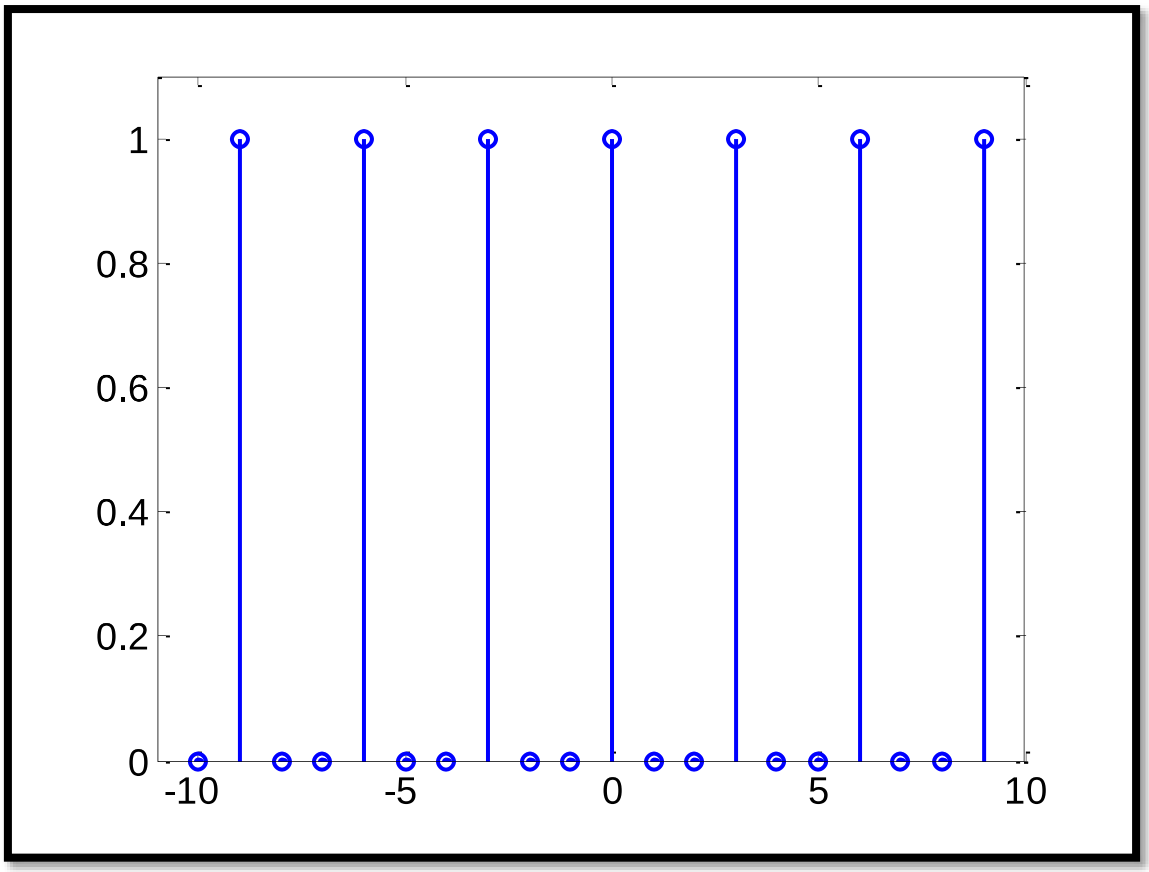


Figure No.13

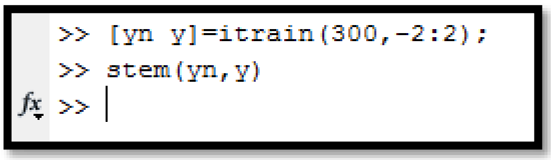
iii. Write a function that lets the user generate an impulse train for a given ***N*** and given range of ***k***, i.e. the function takes at its input ***N*** and range of ***k*** (or one positive value and define within your function array from **–*k*:*k*** to get the whole range) and outputs the impulse train in an array along with a timing index.

Code:

```
function [yn y]=itrain(N,k)
yn_min=k(1)*N-1;
yn_max=k(end)*N+1;
yn=yn_min:yn_max;
y=0;
for i=1:length(k)-1
    y=cat(2,y,[1 zeros(1,N-1)];
end
y=cat(2,y,[1 0]);
```

iv. Use the function that you just defined to generate an impulse train with ***N* = *300*** and ***k* =*-2:2.*** Plot the impulse train as a function of time index that would be generated.

Command window:



```
>> [yn y]=itrain(300,-2:2);
>> stem(yn,y)
>>
```

Figure No.14

Matlab graph:

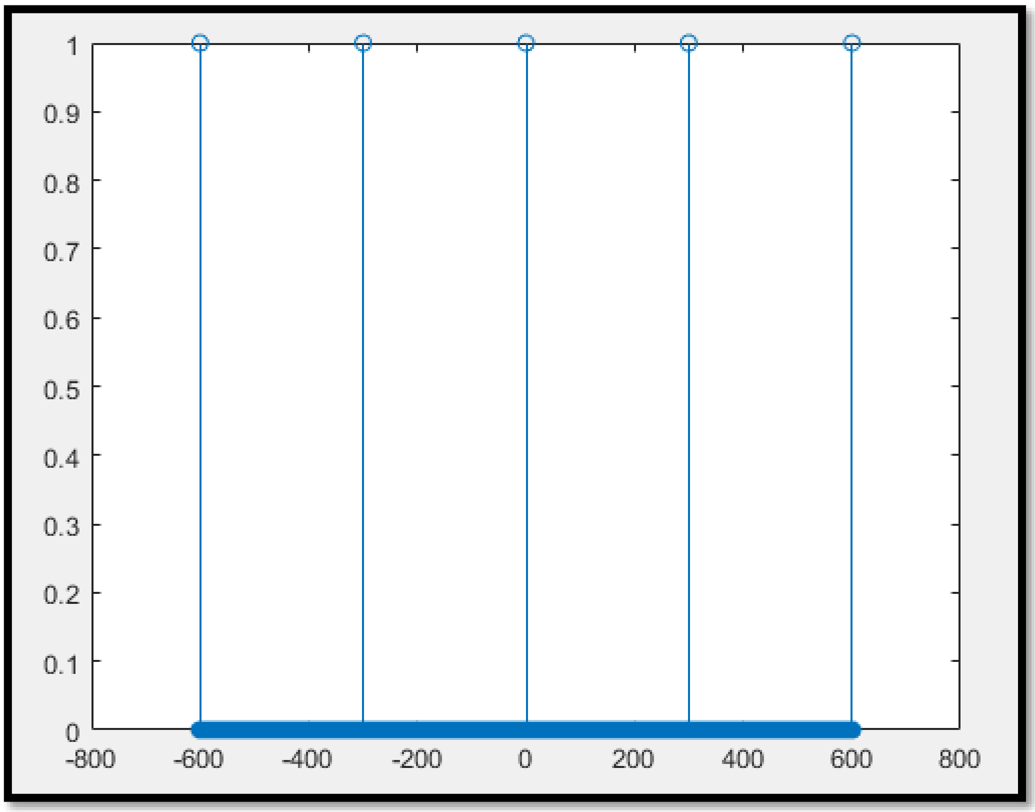


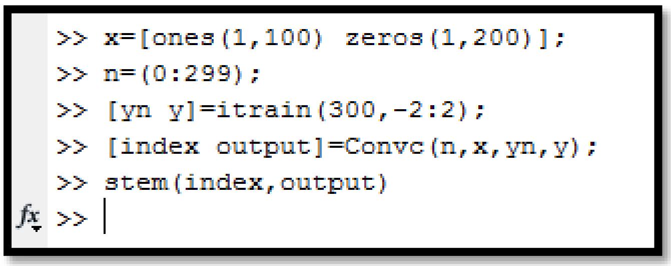
Figure No.15

v. Now comes the interesting part, convolve the signal generated in **(i)** with the impulse train you generated in **(iv)**. Plot the output of the convolution. How's this output related to the original input you plotted in **(i)**?

Comments:

Output is consisted of repeating whole original signal (i), at the indexes containing impulses in impulse-train signal.

Command window:



```
>> x=[ones(1,100) zeros(1,200)];
>> n=(0:299);
>> [yn y]=itrain(300,-2:2);
>> [index output]=Convc(n,x,yn,y);
>> stem(index,output)
fx >> 
```

Figure No.16

Matlab graph:

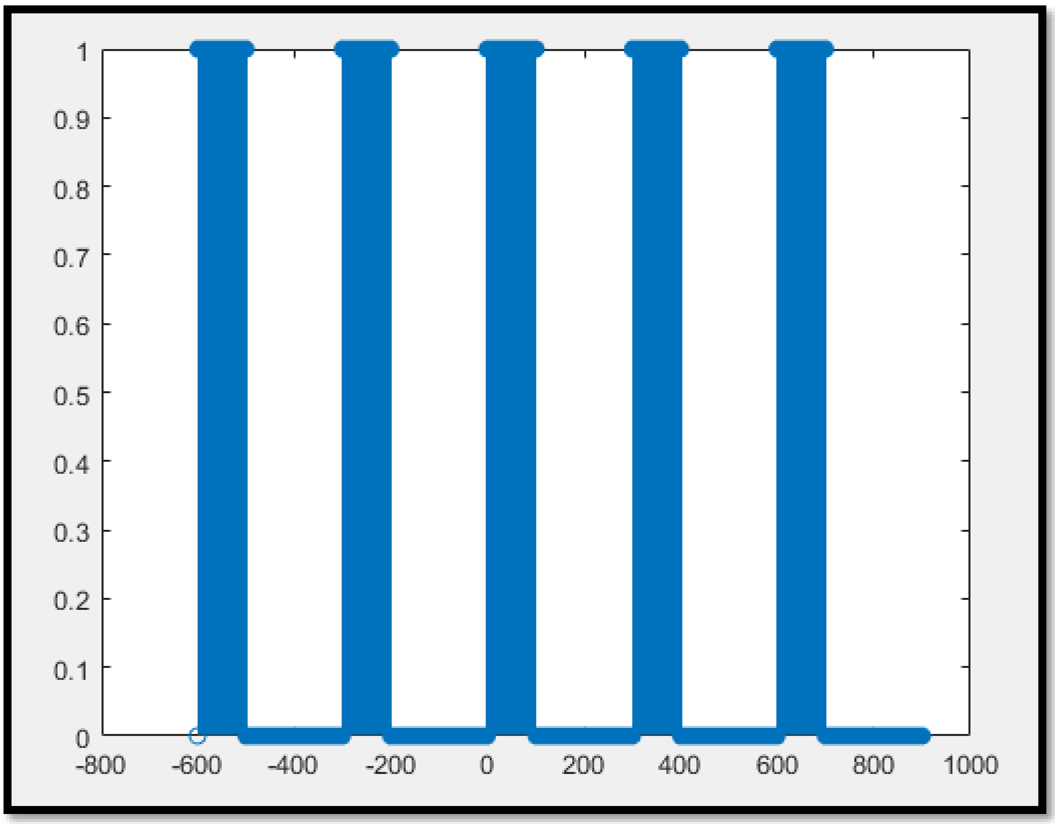


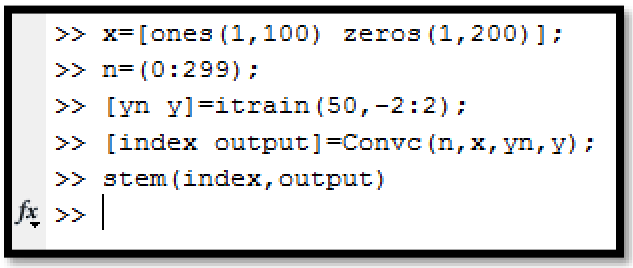
Figure No.17

vi. Repeat **(iv)** and **(v)** with N = 50 for the same input as in **(i)**. Plot the corresponding outputs. Could you still establish a similar relation as in **(v)**.

Comments:

Because of overlap of repeating signal (i), we are not able to establish a similar relation. However, phenomena is still the same.

Command window:



```
>> x=[ones(1,100) zeros(1,200)];
>> n=(0:299);
>> [yn y]=itrain(50,-2:2);
>> [index output]=Convc(n,x,yn,y);
>> stem(index,output)
fx >> 
```

Figure No.18

Matlab graph:

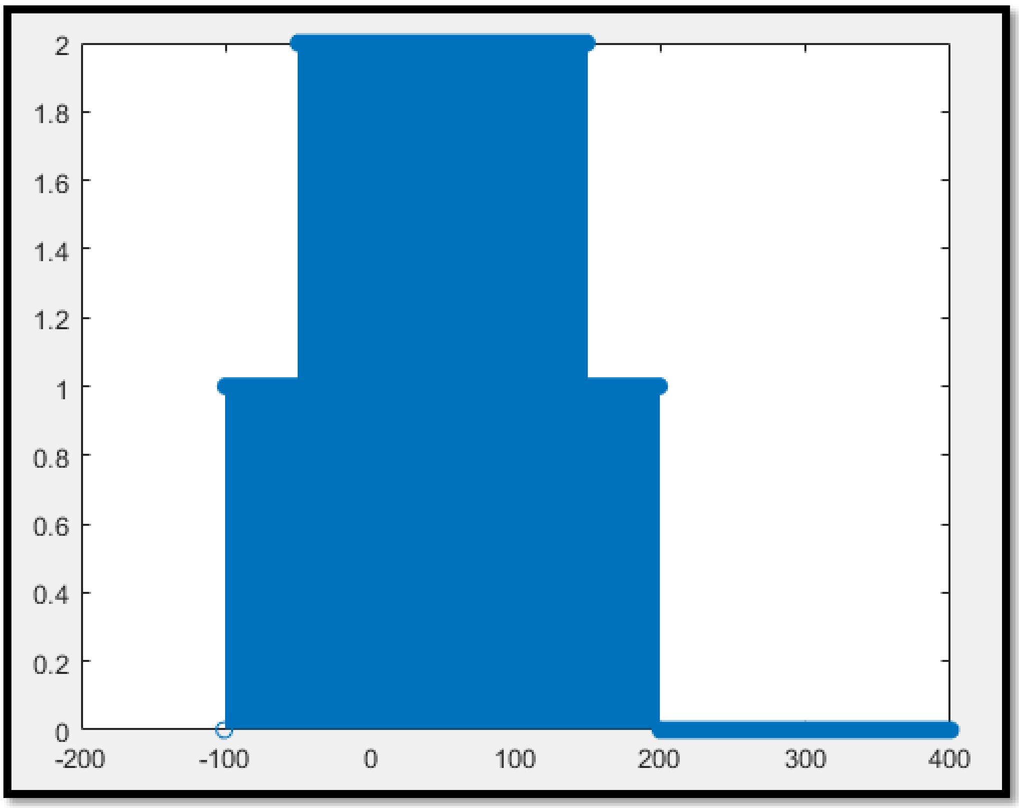


Figure No.19

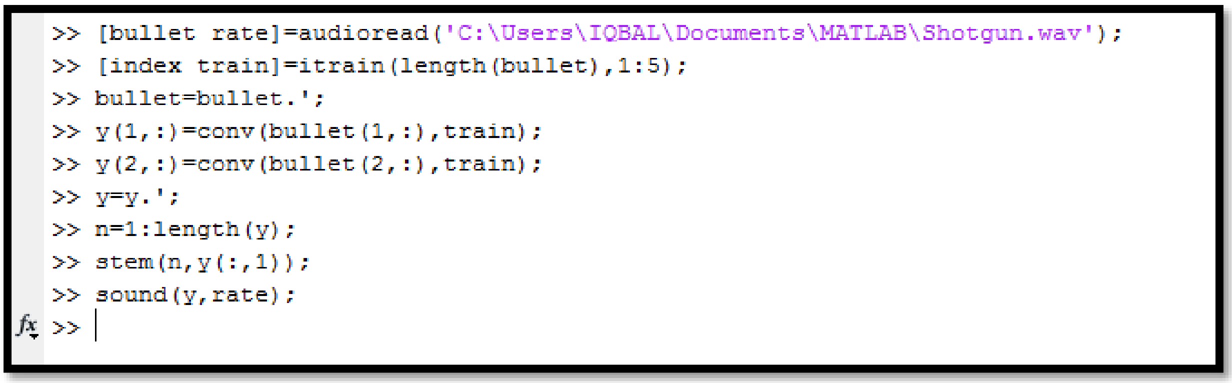
vii. Based on the results of **(v)** & **(vi)**, comment how could you obtain the periodic version of a time limited input signal and if there's any constraint on the value of ***N*** with reference to the input signal.

Comments:

N should be greater than length of original signal.

viii. Load a sample audio clip (.wav file) in MATLAB and generate its periodic version (repetition) using the process above without causing any overlap on the original audio content.

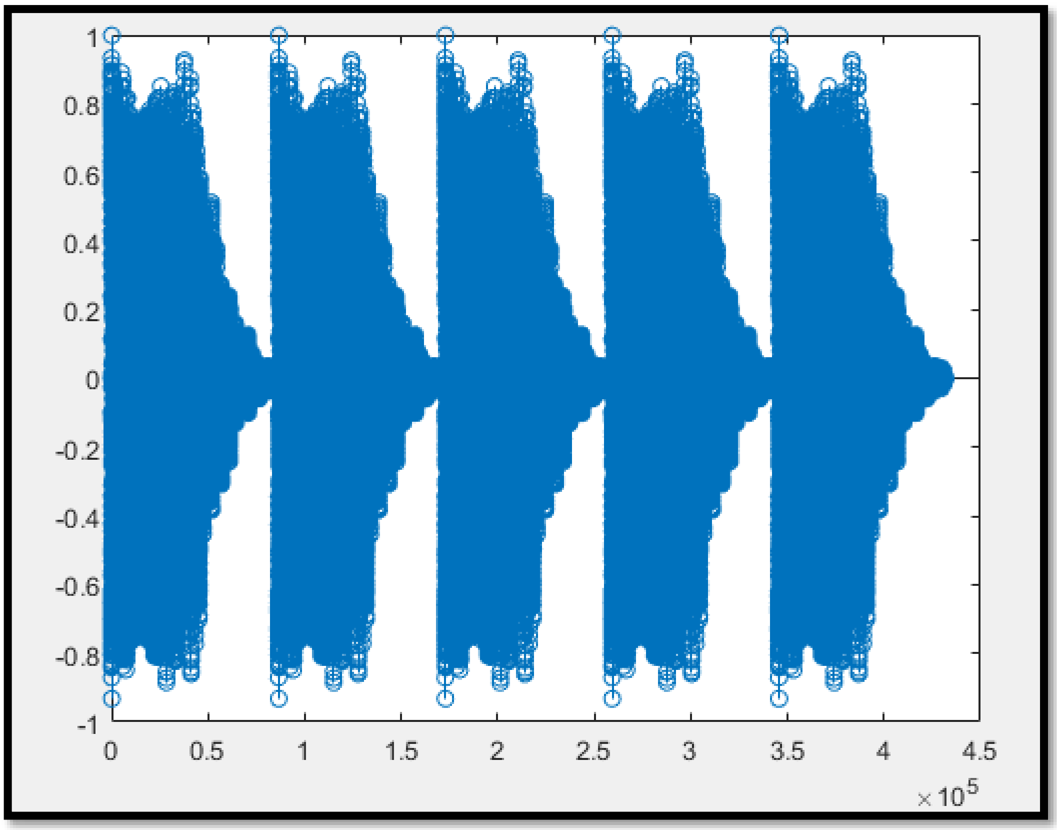
Command window:



```
>> [bullet rate]=audioread('C:\Users\IQBAL\Documents\MATLAB\Shotgun.wav');
>> [index train]=itrain(length(bullet),1:5);
>> bullet=bullet.';
>> y(1,:)=conv(bullet(1,:),train);
>> y(2,:)=conv(bullet(2,:),train);
>> y=y.';
>> n=1:length(y);
>> stem(n,y(:,1));
>> sound(y,rate);
fx >>
```

Figure No.19

Matlab graph:



Note:

.wav file is attached with document.
