Generation of signals and sequences

AIM: Generate various signals and sequences (Periodic and aperiodic), such as Unit Impulse, Unit Step, Square, Saw tooth, Triangular, Sinusoidal, Ramp, Sinc.

Theory: If the amplitude of the signal is defined at every instant of time then it is called continuous time signal. If the amplitude of the signal is defined at only at some instants of time then it is called discrete time signal. If the signal repeats itself at regular intervals then it is called periodic signal. Otherwise they are called aperiodic signals.

Examples: ramp, Impulse, unit step, sinc- Aperiodic signals

square, saw tooth, triangular sinusoidal – periodic signals.

Ramp signal: The **ramp function** is a unitary real function, easily computable as the mean of the independent variable and its absolute value. This function is applied in engineering. The name *ramp function* is derived from the appearance of its graph.

r(t)= t when 0

0 else

Unit impulse signal: One of the more useful functions in the study of linear systems is the "unit impulse function." An ideal impulse function is a function that is zero everywhere but at the origin, where it is infinitely high. However, the *area* of the impulse is finite

Y(t)= 1 when t=0

=0 other wise

Unit step signal: The unit step function and the impulse function are considered to be fundamental functions in engineering, and it is strongly recommended that the reader becomes very familiar with both of these functions.

$$\mathbf{u(t)=}\begin{cases}\mathbf{0 \text{ if } t < 0}\\ \mathbf{1 \text{ if } t > 0}\\ \mathbf{\frac{1}{2} \text{ If } t=0}\end{cases}$$

Sinc signal: There is a particular form that appears so frequently in communications engineering, that we give it its own name. This function is called the "Sinc function".
The Sinc function is defined in the following manner:

$$sinc(x) = \frac{\sin \pi x}{\pi x} \quad \textbf{if } x \neq 0 \textbf{ and sinc(0) =1}$$

The value of sinc(x) is defined as 1 at x = 0, since

$$\lim_{x \to 0} sinc(x) = 1$$

PROGRAM:

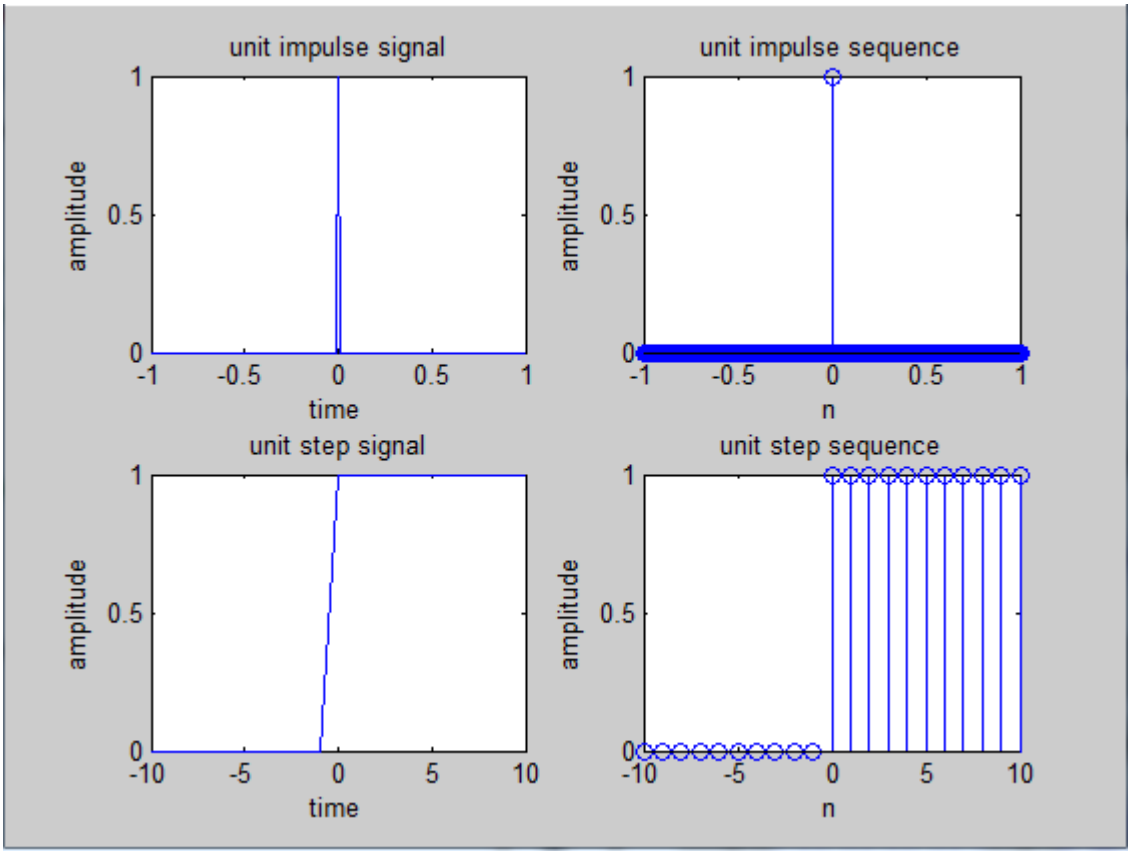
```
% Generation of signals and sequences
clc;
clear all;
close all;
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
%generation of unit impulse signal
t1=-1:0.01:1
y1=(t1==0);
subplot(2,2,1);
plot(t1,y1);
xlabel('time');
ylabel('amplitude');
title('unit impulse signal');
%generation of impulse sequence
subplot(2,2,2);
stem(t1,y1);
xlabel('n');
ylabel('amplitude');
title('unit impulse sequence');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%generation of unit step signal
t2=-10:1:10;
y2=(t2>=0);
subplot(2,2,3);
plot(t2,y2);
xlabel('time');
ylabel('amplitude');
title('unit step signal');
%generation of unit step sequence
subplot(2,2,4);
stem(t2,y2);
xlabel('n');
ylabel('amplitude');
title('unit step sequence');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%generation of square wave signal
t=0:0.002:0.1;
y3=square(2*pi*50*t);
```

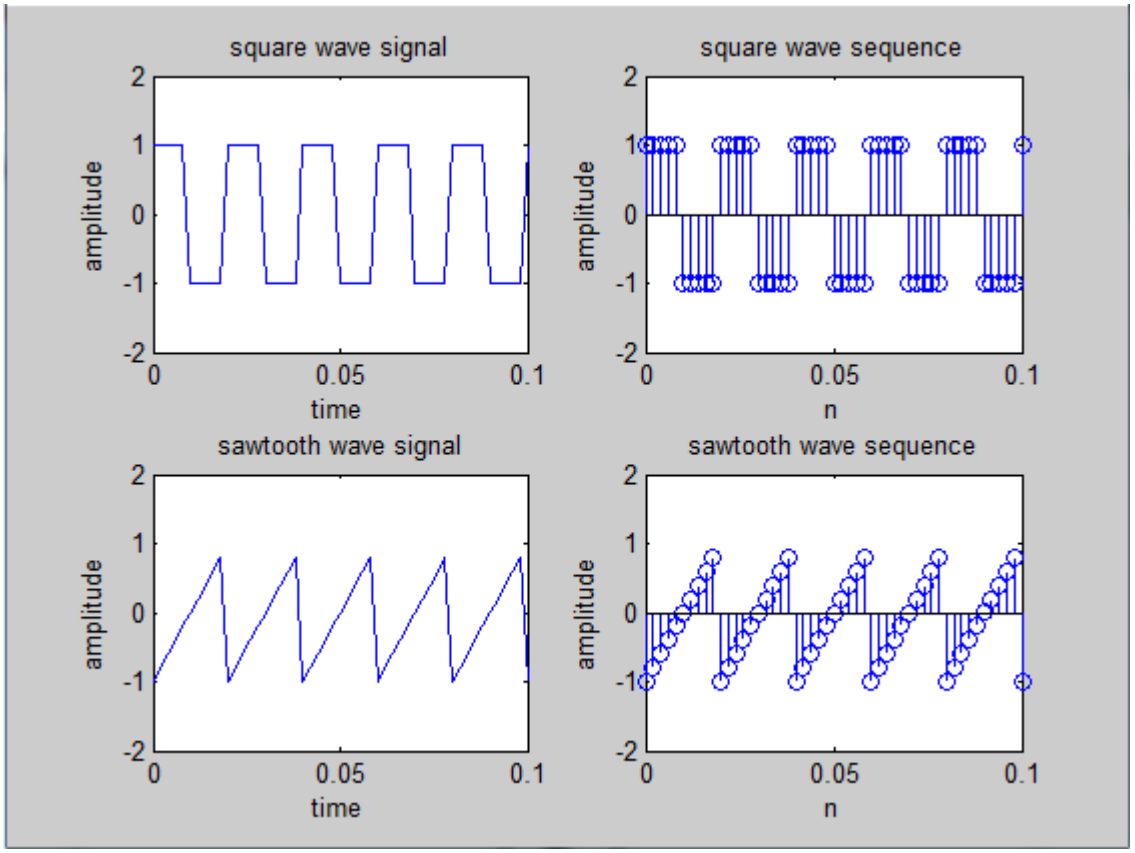
```
figure;
subplot(2,2,1);
plot(t,y3);
axis([0 0.1 -2 2]);
xlabel('time');
ylabel('amplitude');
title('square wave signal');
%generation of square wave sequence
subplot(2,2,2);
stem(t,y3);
axis([0 0.1 -2 2]);
xlabel('n');
ylabel('amplitude');
title('square wave sequence');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%generation of sawtooth signal
y4=sawtooth(2*pi*50*t);
subplot(2,2,3);
plot(t,y4);
axis([0 0.1 -2 2]);
xlabel('time');
ylabel('amplitude');
title('sawtooth wave signal');
%generation of sawtooth sequence
subplot(2,2,4);
stem(t,y4);
axis([0 0.1 -2 2]);
xlabel('n');
ylabel('amplitude');
title('sawtooth wave sequence');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%generation of triangular wave signal
y5=sawtooth(2*pi*50*t,.5);
figure;
subplot(2,2,1);
plot(t,y5);
axis([0 0.1 -2 2]);
xlabel('time');
ylabel('amplitude');
title(' triangular wave signal');
%generation of triangular wave sequence
subplot(2,2,2);
stem(t,y5);
axis([0 0.1 -2 2]);
xlabel('n');
ylabel('amplitude');
title('triangular wave sequence');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

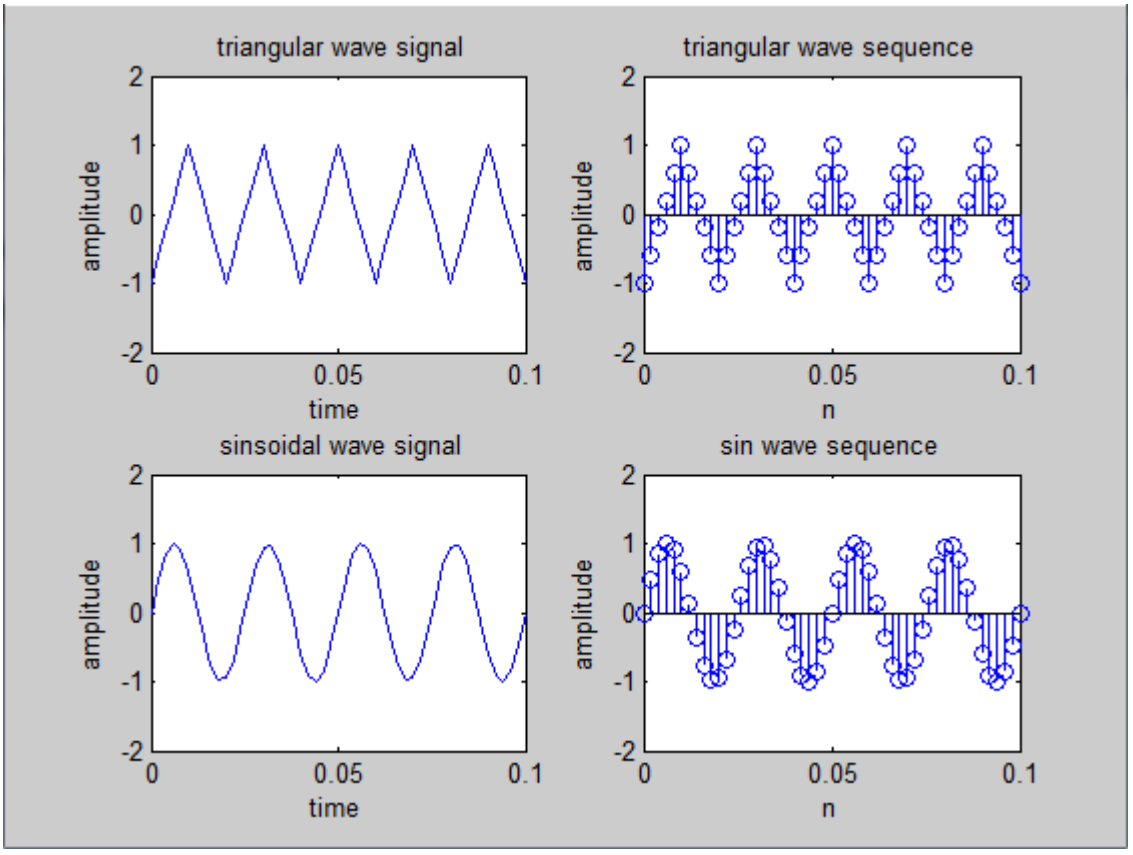
```
%generation of sinsoidal wave signal
y6=sin(2*pi*40*t);
subplot(2,2,3);
plot(t,y6);
axis([0 0.1 -2 2]);
xlabel('time');
ylabel('amplitude');
title(' sinsoidal wave signal');
%generation of sin wave sequence
subplot(2,2,4);
stem(t,y6);
axis([0 0.1 -2 2]);
xlabel('n');
ylabel('amplitude');
title('sin wave sequence');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%generation of ramp signal
y7=t;
figure;
subplot(2,2,1);
plot(t,y7);
xlabel('time');
ylabel('amplitude');
title('ramp signal');
%generation of ramp sequence
subplot(2,2,2);
stem(t,y7);
xlabel('n');
ylabel('amplitude');
title('ramp sequence');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

%generation of sinc signal
t3=linspace(-5,5);
y8=sinc(t3);
subplot(2,2,3);
plot(t3,y8);
xlabel('time');
ylabel('amplitude');
title(' sinc signal');
%generation of sinc  sequence
subplot(2,2,4);
stem(y8);
xlabel('n');
ylabel('amplitude');
title('sinc sequence');
```

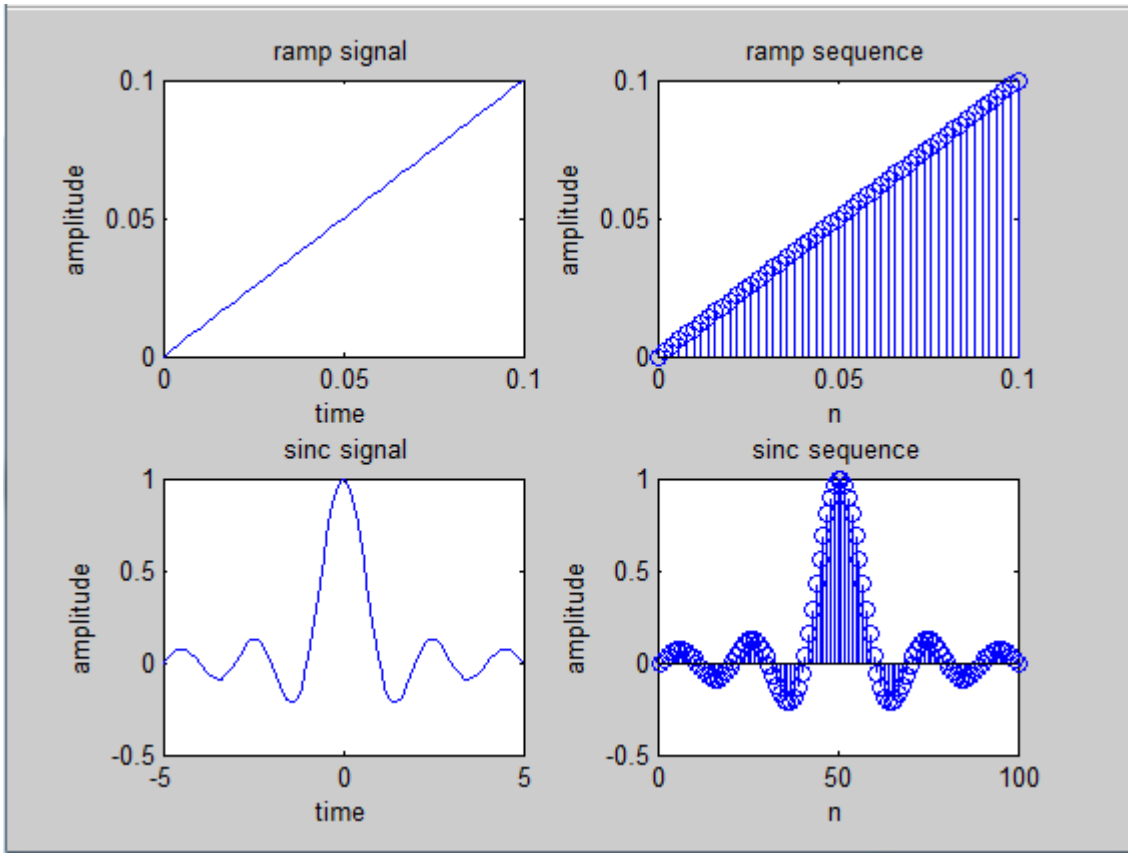
Result: Various signals & sequences generated using Matlab software.

output:


unit impulse signal
amplitude
time
unit impulse sequence
amplitude
n
unit step signal
amplitude
time
unit step sequence
amplitude
n


square wave signal
amplitude
time
square wave sequence
amplitude
n
sawtooth wave signal
amplitude
time
sawtooth wave sequence
amplitude
n


triangular wave signal
amplitude
time
triangular wave sequence
amplitude
n
sinsoidal wave signal
amplitude
time
sin wave sequence
amplitude
n


ramp signal
amplitude
time
ramp sequence
amplitude
n
sinc signal
amplitude
time
sinc sequence
amplitude
n

Basic Operations on Signals and sequences

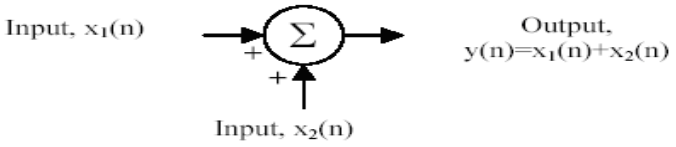
AIM: perform the operations on signals and sequences such as addition, multiplication, scaling, shifting, folding and also compute energy and power.

Theory:

Signal Addition

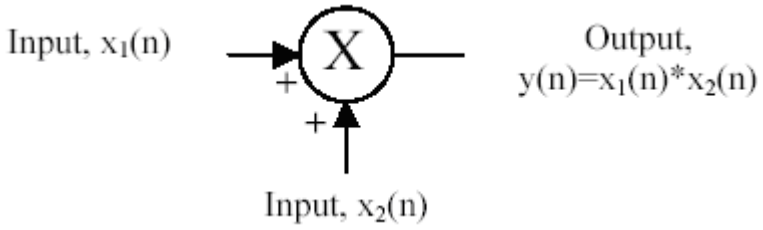
Addition: any two signals can be added to form a third signal,

$$z(t) = x(t) + y(t)$$



Multiplication:

Multiplication of two signals can be obtained by multiplying their values at every instants. z
z(t) = x (t) y (t)

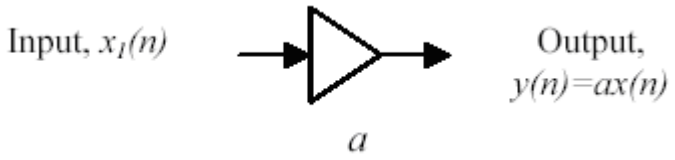


Time reversal/Folding:

Time reversal of a signal x (t) can be obtained by folding the signal about t=0.
Y (t) =y (-t)

Signal Amplification/Scaling: Y (n) =ax (n) if a < 1 attenuation

a >1 amplification



Time shifting: The time shifting of x (n) obtained by delay or advance the signal in time by using y (n) =x (n+k)

If k is a positive number, y (n) shifted to the right i.e., the shifting delays the signal

If k is a negative number, y (n) it gets shifted left. Signal Shifting advances the signal

Energy:

$$E[n] = \lim_{N\to\infty} \sum_{n=-N}^{N} |x[n]|^2 = \sum_{n=-\infty}^{\infty} |x[n]|^2$$

Average power:

$$P[n] = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{n=-N}^{N} |x[n]|^2$$

Program:

```
clc;
clear all;
close all;
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% generating two input signals
t=0:.01:1;
x1=sin(2*pi*4*t);
x2=sin(2*pi*8*t);
subplot(2,2,1);
plot(t,x1);
xlabel('time');
ylabel('amplitude');
title('input signal 1');
subplot(2,2,2);
plot(t,x2);
xlabel('time');
ylabel('amplitude');
title('input signal 2');

% addition of signals
y1=x1+x2;
subplot(2,2,3);
plot(t,y1);
xlabel('time');
ylabel('amplitude');
title('addition of two signals');

% multiplication of signals
y2=x1.*x2;
subplot(2,2,4);
plot(t,y2);
xlabel('time');
ylabel('amplitude');
title('multiplication of two signals');
```

```
% scaling of a signal1
A=2;
y3=A*x1;
figure;
subplot(2,2,1);
plot(t,x1);
xlabel('time');
ylabel('amplitude');
title('input signal')
subplot(2,2,2);
plot(t,y3);
xlabel('time');
ylabel('amplitude');
title('amplified input signal');

% folding of a signal1
h=length(x1);
nx=0:h-1;
subplot(2,2,3);
plot(nx,x1);
xlabel('nx');
ylabel('amplitude');
title('input signal')
y4=fliplr(x1);
nf=-fliplr(nx);
subplot(2,2,4);
plot(nf,y4);
xlabel('nf');
ylabel('amplitude');
title('folded signal');

%shifting of a signal 1
figure;
subplot(3,1,1);
plot(t,x1);
xlabel('time t');
ylabel('amplitude');
title('input signal');
subplot(3,1,2);
plot(t+2,x1);
xlabel('t+2');
ylabel('amplitude');
title('right shifted signal');
subplot(3,1,3);
plot(t-2,x1);
xlabel('t-2');
ylabel('amplitude');
title('left shifted signal');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
%operations on sequences
n1=1:1:9;
s1=[1 2 3 0 5 8 0 2 4];
figure;
subplot(2,2,1);
stem(n1,s1);
xlabel('n1');
ylabel('amplitude');
title('input sequence1');
s2=[1 1 2 4 6 0 5 3 6];
subplot(2,2,2);
stem(n1,s2);
xlabel('n2');
ylabel('amplitude');
title('input sequence2');

% addition of sequences
s3=s1+s2;
subplot(2,2,3);
stem(n1,s3);
xlabel('n1');
ylabel('amplitude');
title('sum of two sequences');

% multiplication  of sequences
s4=s1.*s2;
subplot(2,2,4);
stem(n1,s4);
xlabel('n1');
ylabel('amplitude');
title('product of two sequences');
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

% program for energy of a sequence
z1=input('enter the input sequence');
e1=sum(abs(z1).^2);
disp('energy of given sequence is');e1

% program for energy of a signal
t=0:pi:10*pi;
z2=cos(2*pi*50*t).^2;
e2=sum(abs(z2).^2);
disp('energy of given signal is');e2

% program for power of a sequence p1=
(sum(abs(z1).^2))/length(z1);
disp('power of given sequence is');p1

% program for power of a signal
p2=(sum(abs(z2).^2))/length(z2);
disp('power of given signal is');p2
```

OUTPUT:

enter the input sequence[1 3 2 4 1]

energy of given sequence is

e1 = 31

energy of given signal is

e2 = 4.0388

power of given sequence is

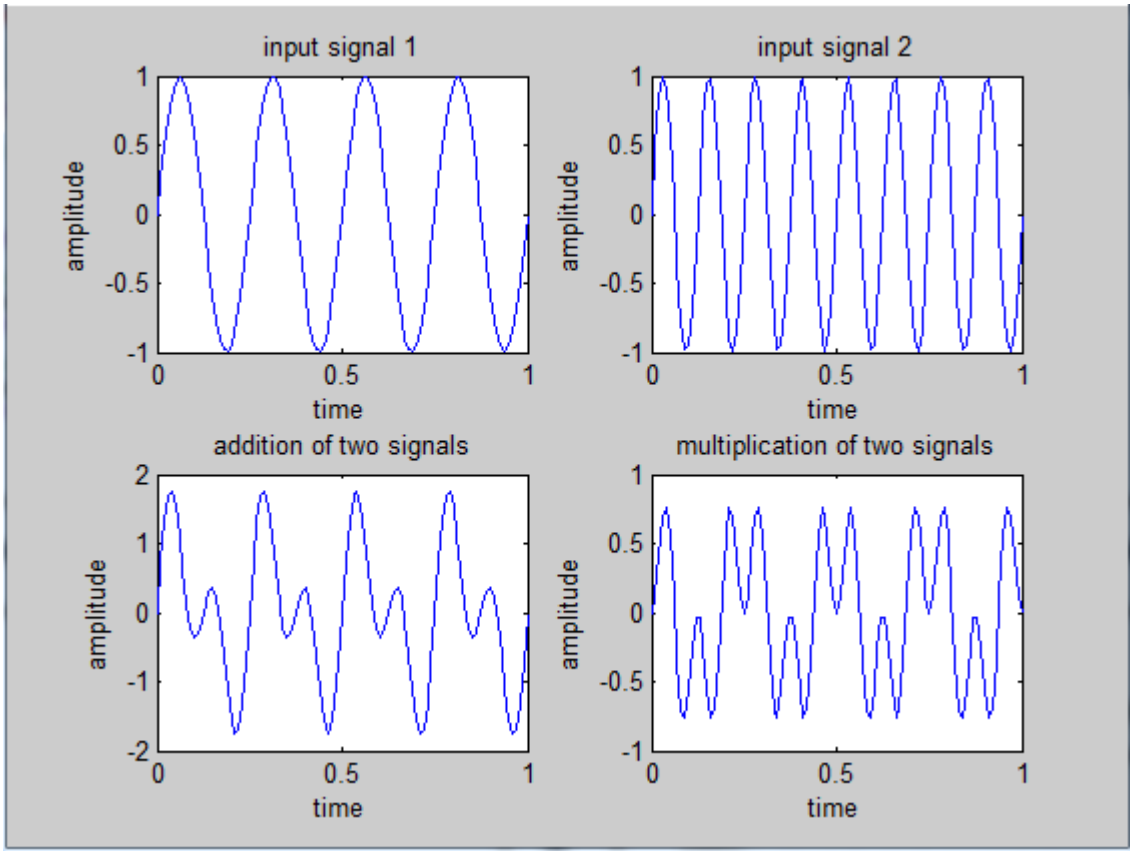
p1 = 6.2000

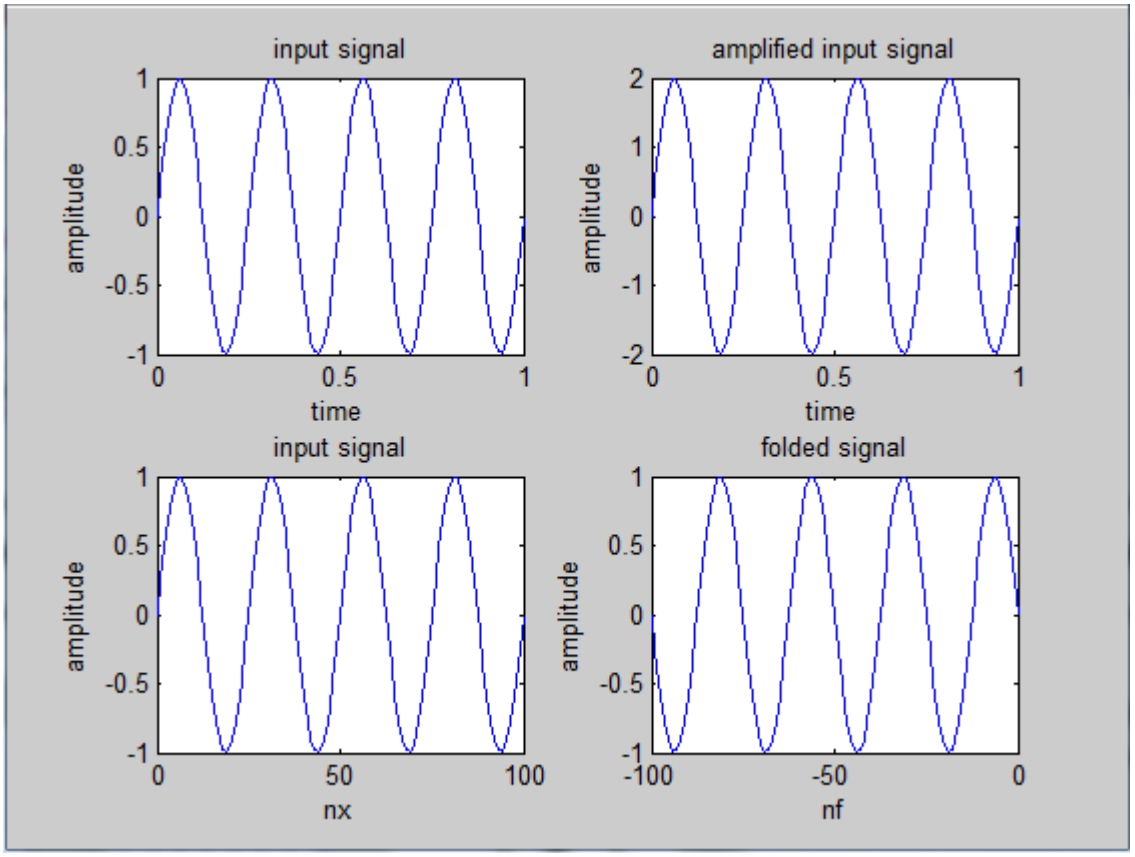
power of given signal is

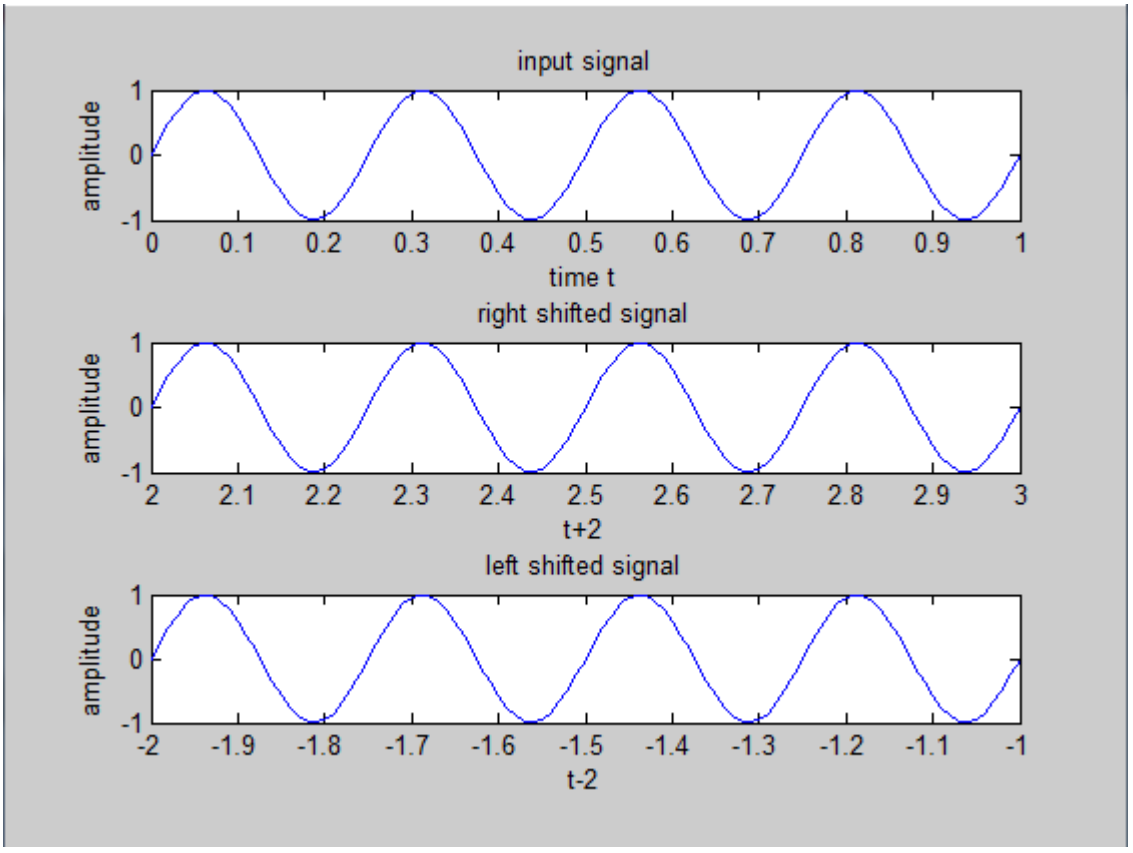
p2 = 0.3672

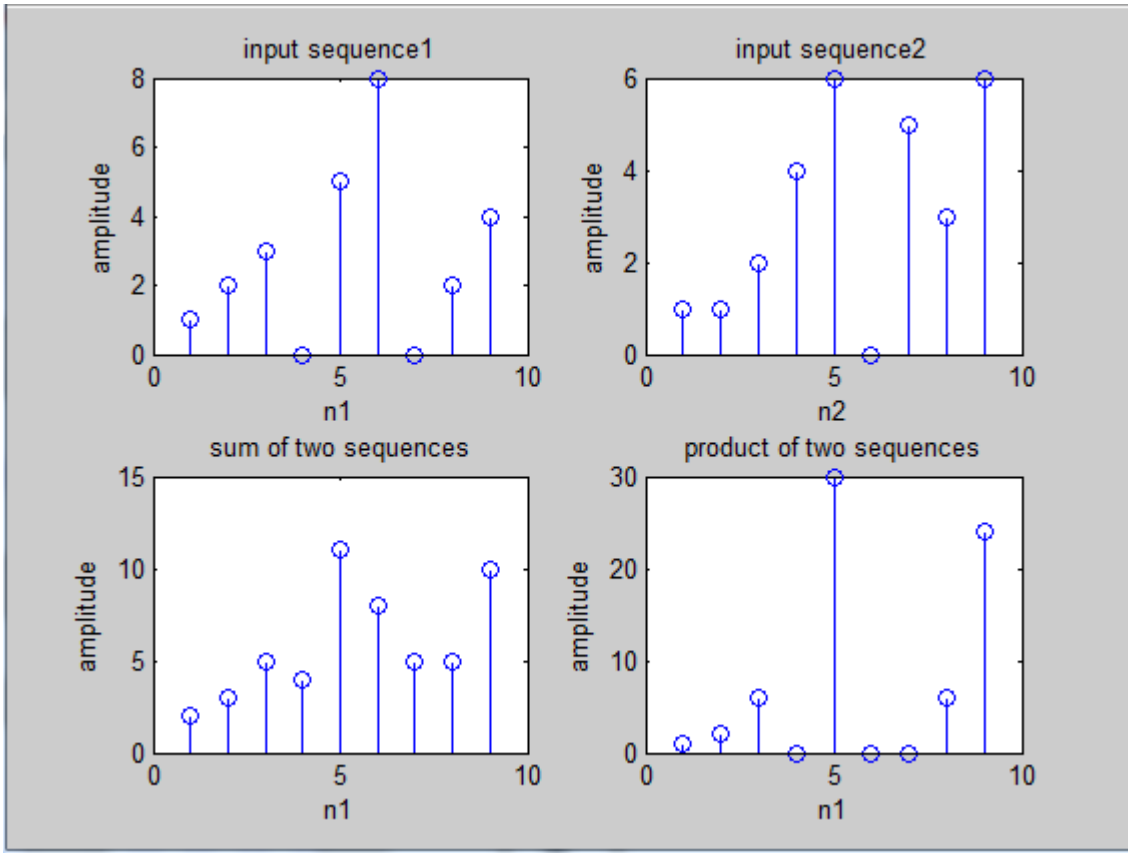
Result: Various operations on signals and sequences are performed.

Output:


input signal 1
input signal 2
addition of two signals
multiplication of two signals
amplitude
time


input signal
amplified input signal
input signal
folded signal
amplitude
time
nx
nf


input signal
amplitude
time t
right shifted signal
t+2
left shifted signal
t-2


input sequence1
input sequence2
sum of two sequences
product of two sequences
amplitude
n1
n2

Even and odd parts of signal and sequence & Real and imaginary parts of Signal

AIM: Finding even and odd part of the signal and sequence and also find real and imaginary parts of signal.

Software Required: Matlab software

Theory: One of characteristics of signal is symmetry that may be useful for signal analysis. Even signals are symmetric around vertical axis, and Odd signals are symmetric about origin.

Even Signal: A signal is referred to as an even if it is identical to its time-reversed counterparts; x(t) = x(-t).

Odd Signal: A signal is odd if x(t) = -x(-t).

An odd signal must be 0 at t=0, in other words, odd signal passes the origin.

Using the definition of even and odd signal, any signal may be decomposed into a sum of its even part, xe(t), and its odd part, xo(t), as follows

Even and odd part of a signal: Any signal x(t) can be expressed as sum of even and odd components i.e .,

$x(t)=x_e(t)+x_o(t)$

$$x_e(t) = \frac{1}{2}\{x(t) + x(-t)\}$$

$$x_o(t) = \frac{1}{2}\{x(t) - x(-t)\}$$

$$x(t) = x_e(t) + x_o(t)$$

$$= \frac{1}{2}\{x(t) + x(-t)\} + \frac{1}{2}\{x(t) - x(-t)\}$$

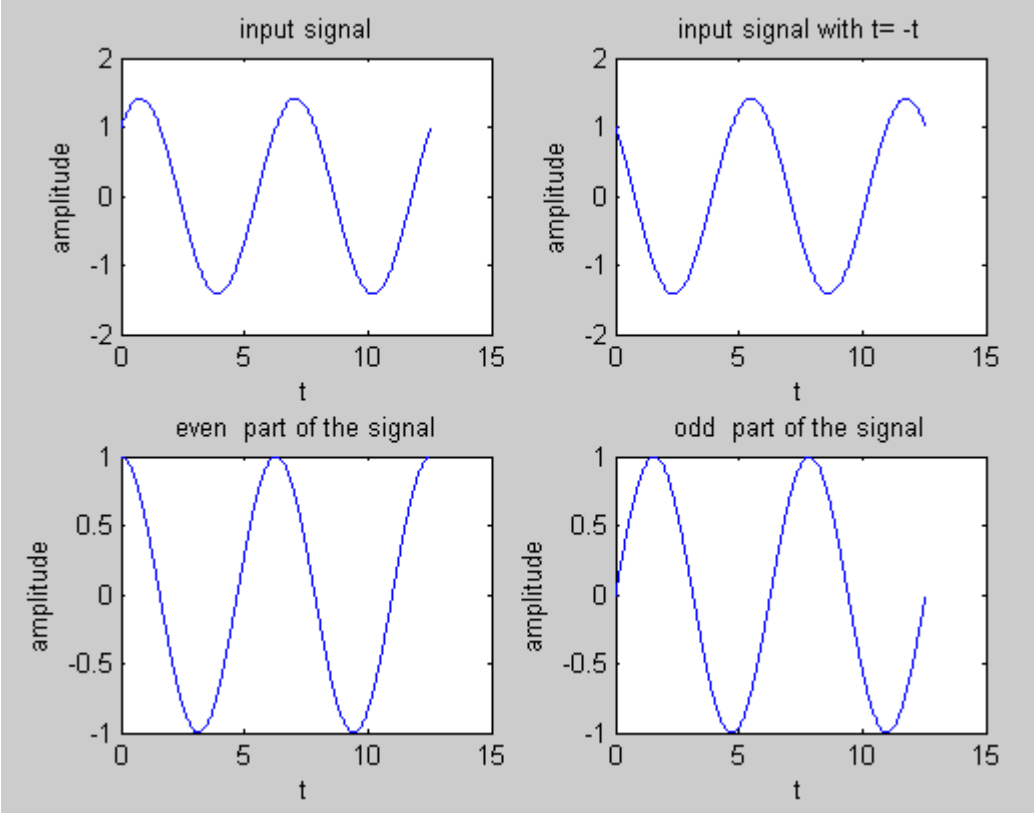
Program:

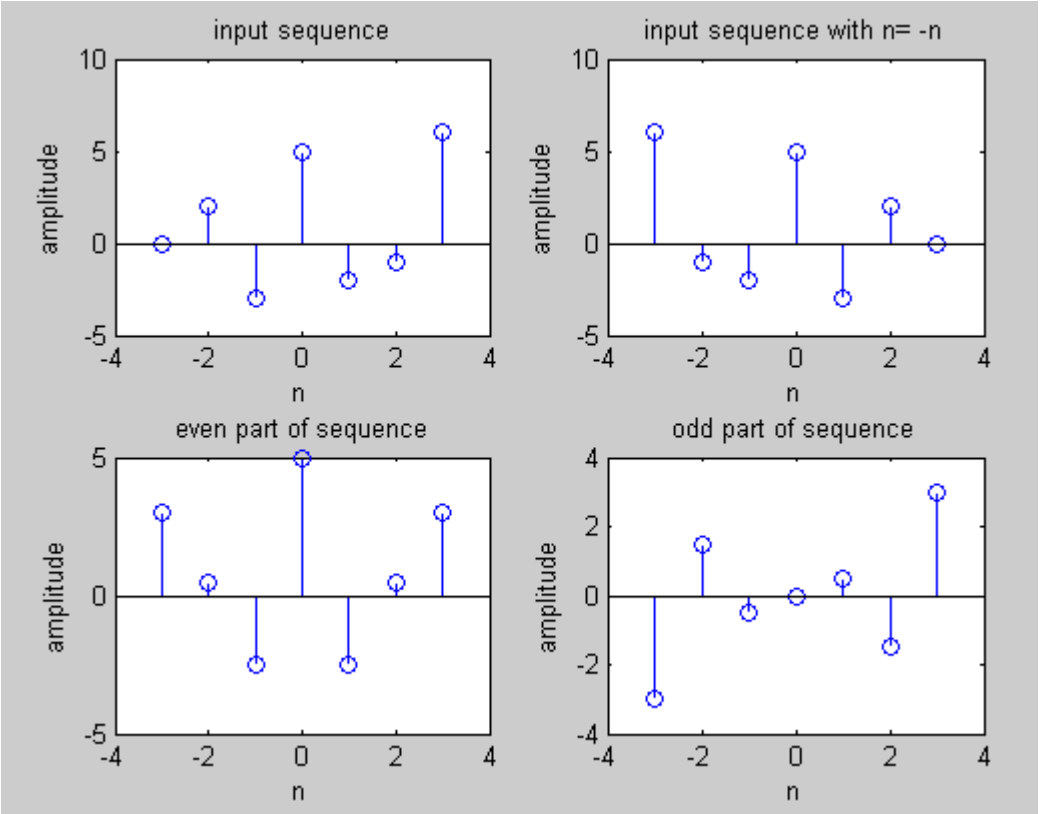
```
clc
close all;
clear all;

%Even and odd parts of a signal
t=0:.001:4*pi;
x=sin(t)+cos(t);      % x(t)=sint(t)+cos(t)
subplot(2,2,1)
plot(t,x)
xlabel('t');
ylabel('amplitude')
title('input signal')

y=sin(-t)+cos(-t);   % y(t)=x(-t)
subplot(2,2,2)
plot(t,y)
xlabel('t');
ylabel('amplitude')
title('input signal with t= -t')
even=(x+y)/2;
subplot(2,2,3)
plot(t,even)
xlabel('t');
```

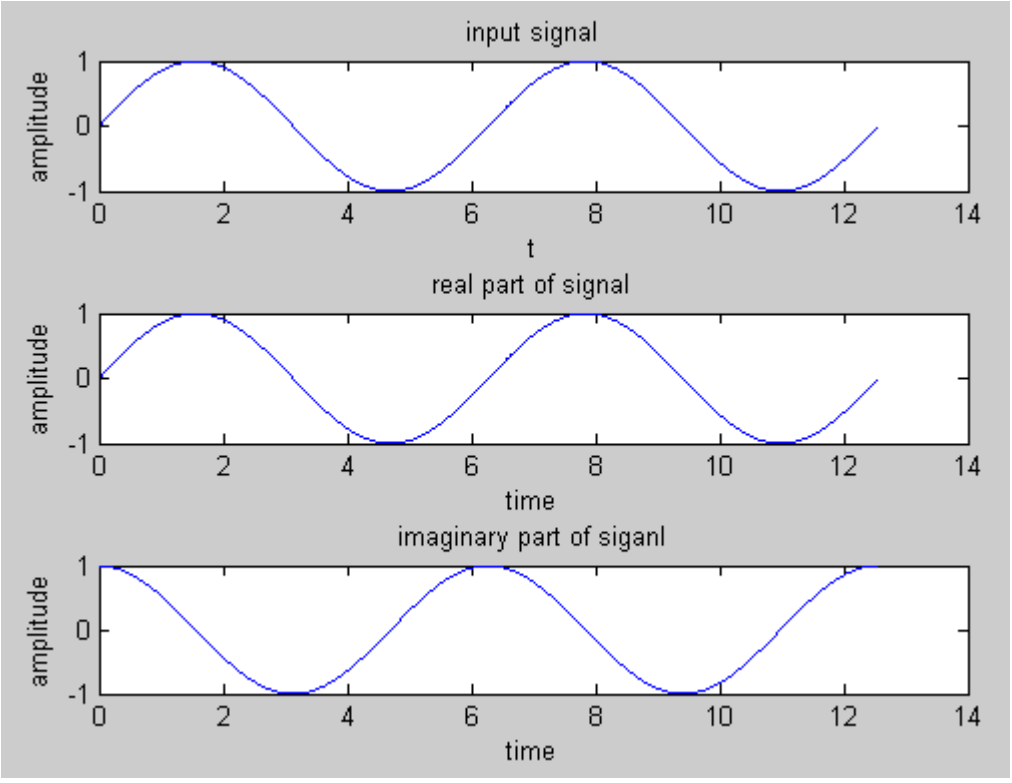
```
ylabel('amplitude')
title('even  part of the signal')
odd=(x-y)/2;
subplot(2,2,4)
plot(t,odd)
xlabel('t');
ylabel('amplitude');
title('odd  part of the signal');

% Even and odd parts of a sequence
x1=[0,2,-3,5,-2,-1,6];
n=-3:3;
y1= fliplr(x1);%y1(n)=x1(-n)
figure;
subplot(2,2,1);
stem(n,x1);
xlabel('n');
ylabel('amplitude');
title('input sequence');
subplot(2,2,2);
stem(n,y1);
xlabel('n');
ylabel('amplitude');
title('input sequence with n= -n');
even1=.5*(x1+y1);
odd1=.5*(x1-y1);

% plotting even and odd parts of the sequence
subplot(2,2,3);
stem(n,even1);
xlabel('n');
ylabel('amplitude');
title('even part of sequence');
subplot(2,2,4);
stem(n,odd1);
xlabel('n');
ylabel('amplitude');
title('odd part of sequence');

% plotting real and imginary parts of the signal
x2=sin(t)+j*cos(t);
figure;
subplot(3,1,1);
plot(t,x2); xlabel('t');
ylabel('amplitude');
title('input signal');
subplot(3,1,2)
plot(t,real(x2));
xlabel('time');
ylabel('amplitude');
title('real part of signal');
subplot(3,1,3)
plot(t,imag(x2));
xlabel('time');
ylabel('amplitude');
title('imaginary part of signal');
```

RESULT: Even and odd part of the signal and sequence, real and imaginary parts of signal are computed.

Output:



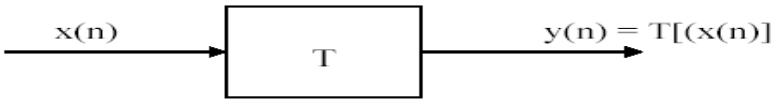

input sequence
input sequence with n= -n
even part of sequence
odd part of sequence
amplitude
n


input signal
real part of signal
imaginary part of siganl
amplitude
t
time

Convolution between signals& sequences

Aim: Write the program for convolution between two signals and also between two sequences.

Theory:



Convolution involves the following operations.
1. Folding
2. Multiplication
3. Addition
4. Shifting

$$y[n] = T \sum_{k=-\infty} x[k]. \delta(n-k)$$

These operations can be represented by a Mathematical Expression as follows:
x[n]= Input signal Samples
h[n-k]= Impulse response co-efficient.
y[n]= Convolution output.
n = No. of Input samples
h = No. of Impulse response co-efficient.
Example : X(n)={1 2 -1 0 1}, h(n)={ 1,2,3,-1}

Program:

```
clc;
close all;
clear all;
%program for convolution of two sequences
x=input('enter input sequence: ');
h=input('enter impulse response: ');
y=conv(x,h);
subplot(3,1,1);
stem(x);
xlabel('n');
ylabel('x(n)');
title('input sequence')
subplot(3,1,2);
stem(h);
xlabel('n');
ylabel('h(n)');
title('impulse response sequence')
subplot(3,1,3);
stem(y);
xlabel('n');
ylabel('y(n)');
title('linear convolution')
disp('linear convolution y=');
```

```
disp(y)
%program for signal convolution
t=0:0.1:10;
x1=sin(2*pi*t);
h1=cos(2*pi*t);
y1=conv(x1,h1);
figure;
subplot(3,1,1);
plot(x1);
xlabel('t');
ylabel('x(t)');
title('input signal')
subplot(3,1,2);
plot(h1);
xlabel('t');
ylabel('h(t)');
title('impulse response')
subplot(3,1,3);
plot(y1);
xlabel('n');
ylabel('y(n)');
title('linear convolution');
```

RESULT: convolution between signals and sequences is computed.

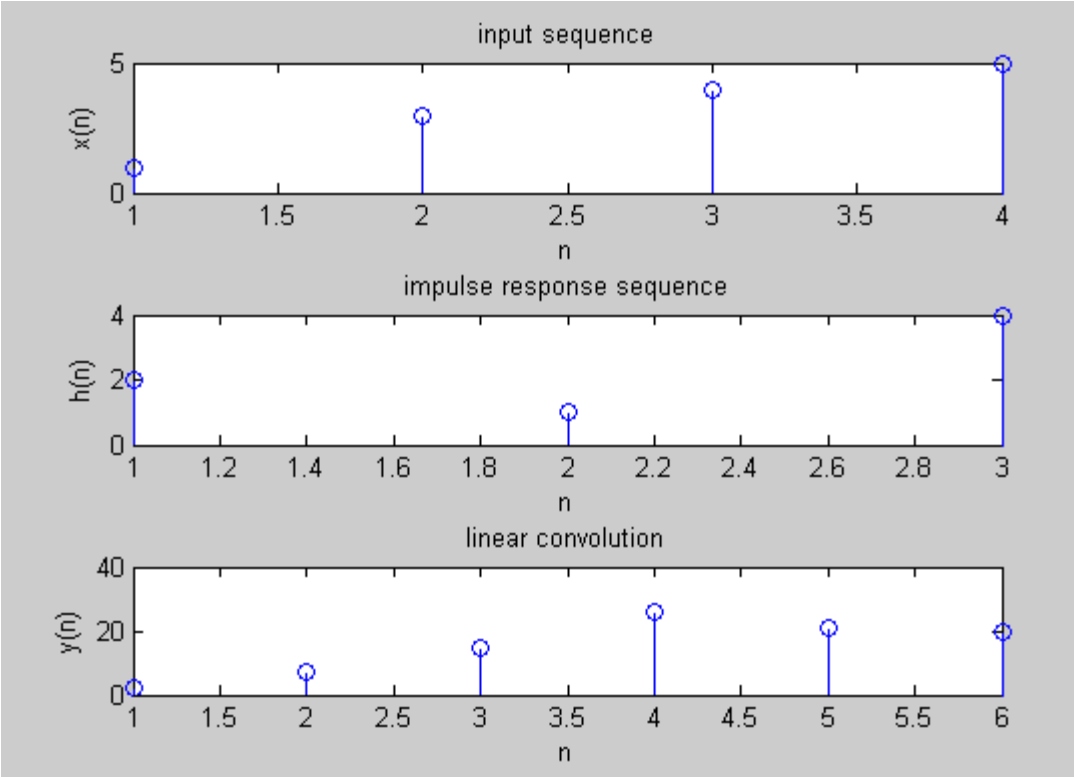
Output:

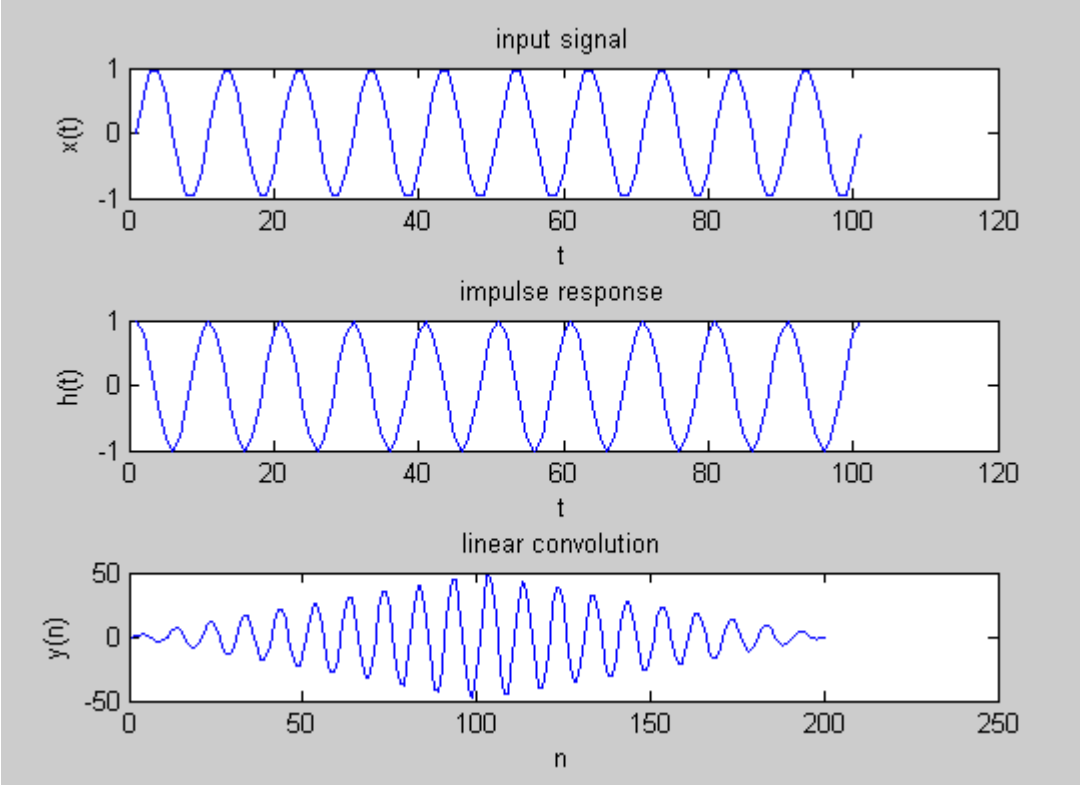
enter input sequence: [1 3 4 5]

enter impulse response: [2 1 4]

linear convolution y=

2 7 15 26 21 20


input sequence
x(n)
n
impulse response sequence
h(n)
n
linear convolution
y(n)
n


input signal
x(t)
t
impulse response
h(t)
t
linear convolution
y(n)
n

Auto correlation and Cross correlation

Aim: To compute Auto correlation and Cross correlation between signals and sequences.

Theory:
Correlations of sequences:

It is a measure of the degree to which two sequences are similar. Given two real-valued sequences $x(n)$ and $y(n)$ of finite energy,

Convolution involves the following operations.

1. Shifting
2. Multiplication
3. Addition

These operations can be represented by a Mathematical Expression as follows:

Cross correlation

$$r_{x,y}(l) = \sum_{n=-\infty}^{+\infty} x(n)y(n-l)$$

The index l is called the shift or lag parameter

Autocorrelation

$$r_{x,x}(l) = \sum_{n=-\infty}^{+\infty} x(n)x(n-l)$$

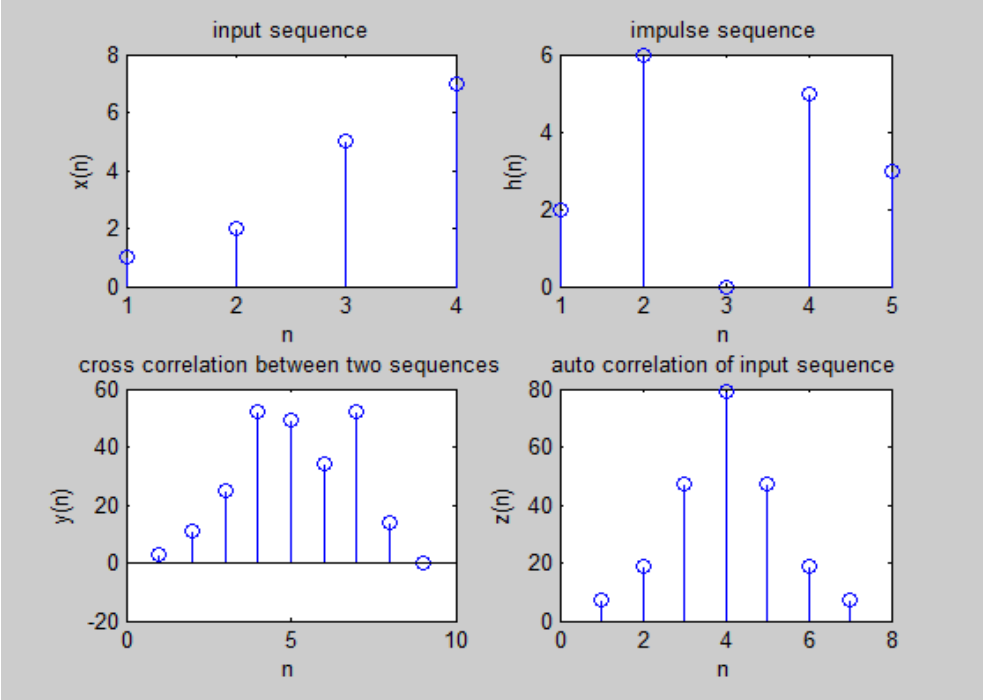
Program:

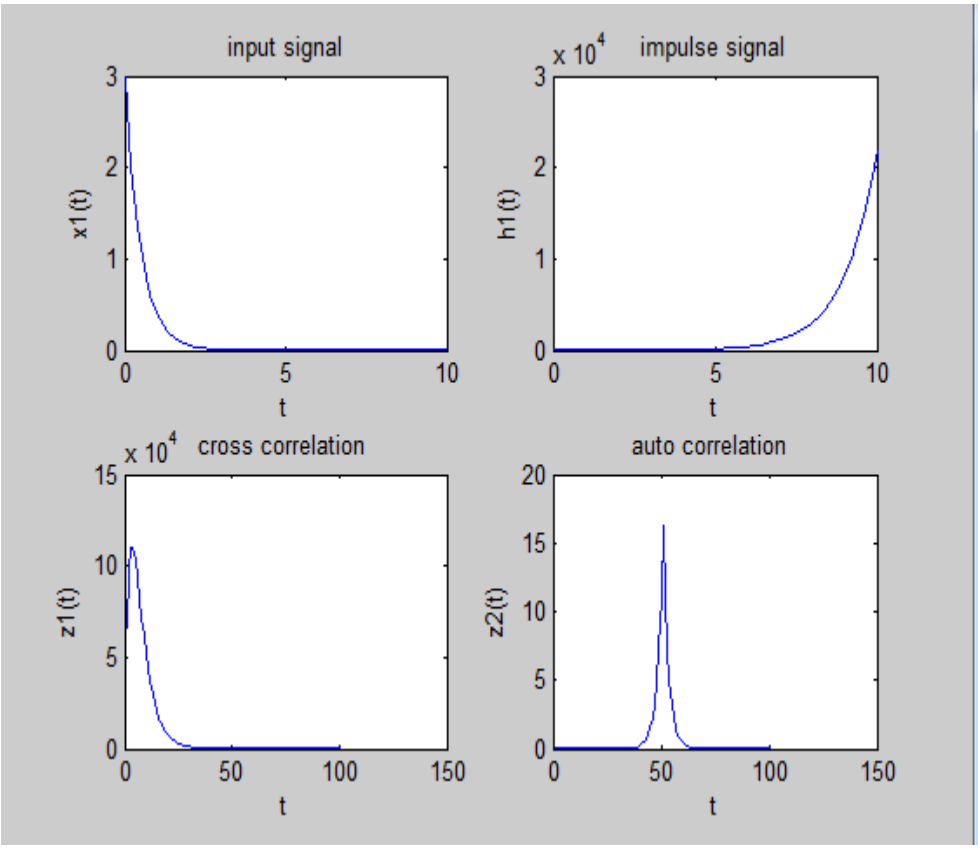
```
clc;
close all;
clear all;

% two input sequences x=input('enter
input sequence'); h=input('enter the
impulse suquence'); subplot(2,2,1);
stem(x); xlabel('n');
ylabel('x(n)');
title('input sequence');
subplot(2,2,2);
stem(h);
xlabel('n'); ylabel('h(n)');
title('impulse sequence');
% cross correlation between two sequences
```

```
y=xcorr(x,h);
subplot(2,2,3);
stem(y);
xlabel('n');
ylabel('y(n)');
title(' cross correlation between two sequences ');
% auto correlation of input sequence
z=xcorr(x,x);
subplot(2,2,4);
stem(z);
xlabel('n');
ylabel('z(n)');
title('auto correlation of input sequence');

%  cross correlation between two signals
% generating two input signals
t=0:0.2:10;
x1=3*exp(-2*t);
h1=exp(t);
figure;
subplot(2,2,1);
plot(t,x1);
xlabel('t');
ylabel('x1(t)');
title('input signal');
subplot(2,2,2);
plot(t,h1);
xlabel('t');
ylabel('h1(t)');
title('impulse signal');

%  cross correlation
subplot(2,2,3);
z1=xcorr(x1,h1);
plot(z1);
xlabel('t');
ylabel('z1(t)');
title('cross correlation ');
% auto correlation
subplot(2,2,4);
z2=xcorr(x1,x1);
plot(z2);
xlabel('t');
ylabel('z2(t)');
title('auto correlation ');
```

Result: Auto correlation and Cross correlation between signals and sequences is computed.

Output: enter input sequence [1 2 5 7]
enter the impulse sequence [2 6 0 5 3]


input sequence
x(n)
n
impulse sequence
h(n)
n
cross correlation between two sequences
y(n)
n
auto correlation of input sequence
z(n)
n


input signal
x1(t)
t
x 10^4
impulse signal
h1(t)
t
x 10^4
cross correlation
z1(t)
t
auto correlation
z2(t)
t

Verification of Linearity of a Discrete System

AIM: Verify the Linearity of a given Discrete System.

Theory:

LINEARITY PROPERTY:

Any system is said to be linear if it satisfies the superposition principal. superposition principal state that Response to a weighted sum of input signal equal to the corresponding weighted sum of the outputs of the system to each of the individual input signals.

If x(n) is a input signal and y(n) is a output signal then

y(n)=T[x(n)]

y1(n)=T[x1(n)] and y2(n)=T[x2(n)]

x3=[a*x1(n) +b *x2(n)]

Y3(n)= T [x3(n)]

T [a*x1(n)+b*x2(n)] = a y1(n)+ b y2(n)

Program:

```
% Verification of Linearity of a given System.
% a) y(n)=nx(n)  b) y=x^2(n)
clc;
clear all;
close all;

n=0:40;
a1=input('enter the scaling factor a1=');
a2=input('enter the scaling factor a2=');
x1=cos(2*pi*0.1*n);
x2=cos(2*pi*0.4*n);
x3=a1*x1+a2*x2;
%y(n)=n.x(n);
y1=n.*x1;
y2=n.*x2;
y3=n.*x3;
yt=a1*y1+a2*y2;
yt=round(yt);
y3=round(y3);
if y3==yt
    disp('given system [y(n)=n.x(n)]is Linear');
else
    disp('given system [y(n)=n.x(n)]is non Linear');
end
%y(n)=x(n).^2
x1=[1 2 3 4 5];
x2=[1 4 7 6 4];
```

```
x3=a1*x1+a2*x2;
y1=x1.^2;
y2=x2.^2;
y3=x3.^2;
yt=a1*y1+a2*y2;
if y3==yt
    disp('given system [y(n)=x(n).^2 ]is Linear');
else
    disp('given system is [y(n)=x(n).^2 ]non Linear');
end
```

Result: The Linearity of a given Discrete System is verified.

Output:

enter the scaling factor a1=3
enter the scaling factor a2=5
given system [y(n)=n.x(n)]is Linear
given system is [y(n)=x(n).^2]non Linear

Verification of Time Invariance of a Discrete System

AIM: Verify the Time Invariance of a given Discrete System.

Theory:

TIME INVARIENT SYSTEMS (TI):

A system is called time invariant if its input – output characteristics do not change with time

X(t)---- input : Y(t) ---output
X(t-k) -----delay input by k seconds : Y(t-k) ------ Delayed output by k seconds

If Y(t)=T[X(t)] then Y(t-k)=T[X(t-k)] then system is time invariant system.

Program:

```
% Verification of Time Invariance of a Discrete System
% a)y=x^2(n)   b) y(n)=nx(n)
clc;
clear all;
close all;
n=1:9;
x(n)=[1 2 3 4 5 6 7 8 9];
d=3; % time delay
xd=[zeros(1,d),x(n)];%x(n-k)
y(n)=x(n).^2;
yd=[zeros(1,d),y];%y(n-k)
disp('transformation of delay signal yd:');disp(yd)
dy=xd.^2; % T[x(n-k)]
disp('delay of transformation signal dy:');disp(dy)
if dy==yd
    disp('given system [y(n)=x(n).^2 ]is time invariant');
else
    disp('given system is [y(n)=x(n).^2 ]not time invariant');
end y=n.*x;
yd=[zeros(1,d),y(n)];
disp('transformation of delay signal yd:');disp(yd);
n1=1:length(xd);
dy=n1.*xd;
disp('delay of transformation signal dy:');disp(dy);
if yd==dy
    disp('given system [y(n)=nx(n)]is a time invariant');
else
    disp('given system [y(n)=nx(n)]not a time invariant');
end
```

Result: The Time Invariance of a given Discrete System is verified.

Output:

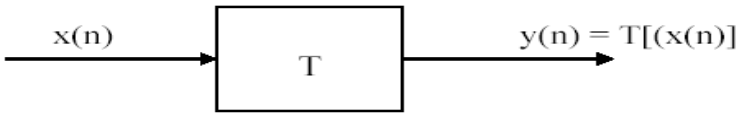
```
transformation of delay signal yd:
    0     0     0     1     4     9    16    25    36    49    64    81

delay of transformation signal dy:
    0     0     0     1     4     9    16    25    36    49    64    81

given system [y(n)=x(n).^2 ]is time invariant
transformation of delay signal yd:
    0     0     0     1     4     9    16    25    36    49    64    81

delay of transformation signal dy:
    0     0     0     4    10    18    28    40    54    70    88   108

given system [y(n)=nx(n)]not a time invariant
```

Unit sample, unit step and sinusoidal response of the given LTI system and verifying its Stability

AIM: Compute the Unit sample, unit step and sinusoidal response of the given LTI system and verifying its stability

Theory:

A discrete time system performs an operation on an input signal based on predefined criteria to produce a modified output signal. The input signal x(n) is the system excitation, and y(n) is the system response. The transform operation is shown as,



If the input to the system is unit impulse i.e. x(n) = δ(n) then the output of the system is known as impulse response denoted by h(n) where,

$$h(n) = T[\delta(n)]$$

we know that any arbitrary sequence x(n) can be represented as a weighted sum of discrete impulses. Now the system response is given by,

$$y(n) = T[x(n)] = T\left[\sum_{k=-\infty}^{\infty} x(k)\,\delta(n-k)\right]$$

For linear system (1) reduces to

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)\,T[\delta(n-k)]$$

%given difference equation y(n)-y(n-1)+.9y(n-2)=x(n);

$$\mathbf{H(Z)=}\ \frac{\sum_{k=0}^{M} b_k X(n-k)}{\sum_{k=1}^{N} a_k X(n-k)}$$

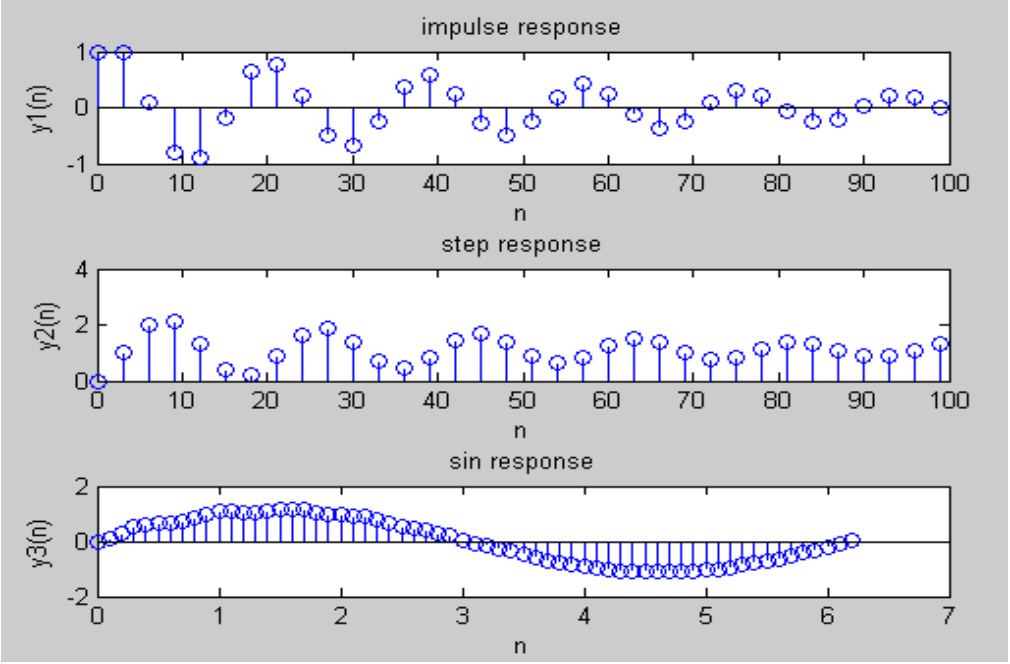
$$\mathbf{H(z)=}\ \frac{b_0+b_1Z^{-1}+b_2Z^{-2}+\ldots\ldots\ldots\ldots b_{N-1}Z^{(N-1)}+b_NZ^{-N}}{1+a_1Z^{-1}+a_2Z^{-2}+\ldots\ldots\ldots\ldots a_{N-1}Z^{(N-1)}+a_NZ^{-N}}$$

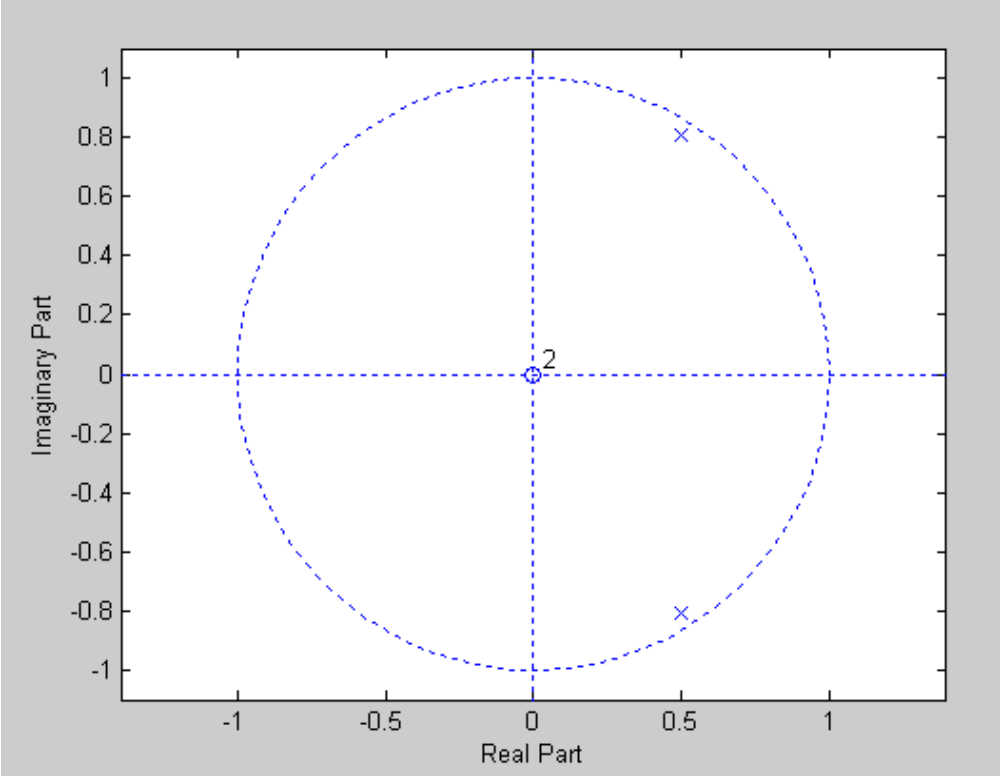
Program:

```
%given difference equation y(n)-y(n-1)+.9y(n-2)=x(n);
clc;
clear  all;
close  all;
b=[1];
a=[1,-1,.9];
n =0:3:100;

%generating impulse signal
x1=(n==0);
%impulse response
y1=filter(b,a,x1);
subplot(3,1,1);
stem(n,y1);
xlabel('n');
ylabel('y1(n)');
title('impulse response');

%generating step signal
x2=(n>0);
% step response
y2=filter(b,a,x2);
subplot(3,1,2);
stem(n,y2);
xlabel('n');
ylabel('y2(n)')
title('step response');
%generating sinusoidal signal
t=0:0.1:2*pi;
x3=sin(t);
% sinusoidal response
y3=filter(b,a,x3);
subplot(3,1,3);
stem(t,y3);
xlabel('n');
ylabel('y3(n)');
title('sin response');

% verifing stability
figure;
zplane(b,a);
```

Result: The Unit sample, unit step and sinusoidal response of the given LTI system is computed and its stability verified. Hence all the poles lie inside the unit circle, so system is stable.

Output:


impulse response
y1(n)
n
step response
y2(n)
n
sin response
y3(n)
n


Imaginary Part
Real Part
2

Gibbs phenomenon

AIM: Verify the Gibbs phenomenon.

Theory:

The **Gibbs phenomenon,** the Fourier series of a piecewise continuously differentiable periodic function behaves at a jump discontinuity. The n the approximated function shows amounts of ripples at the points of discontinuity. This is known as the Gibbs Phenomena. Partial sum of the Fourier series has large oscillations near the jump, which might increase the maximum of the partial sum above that of the function itself. The overshoot does not die out as the frequency increases, but approaches a finite limit

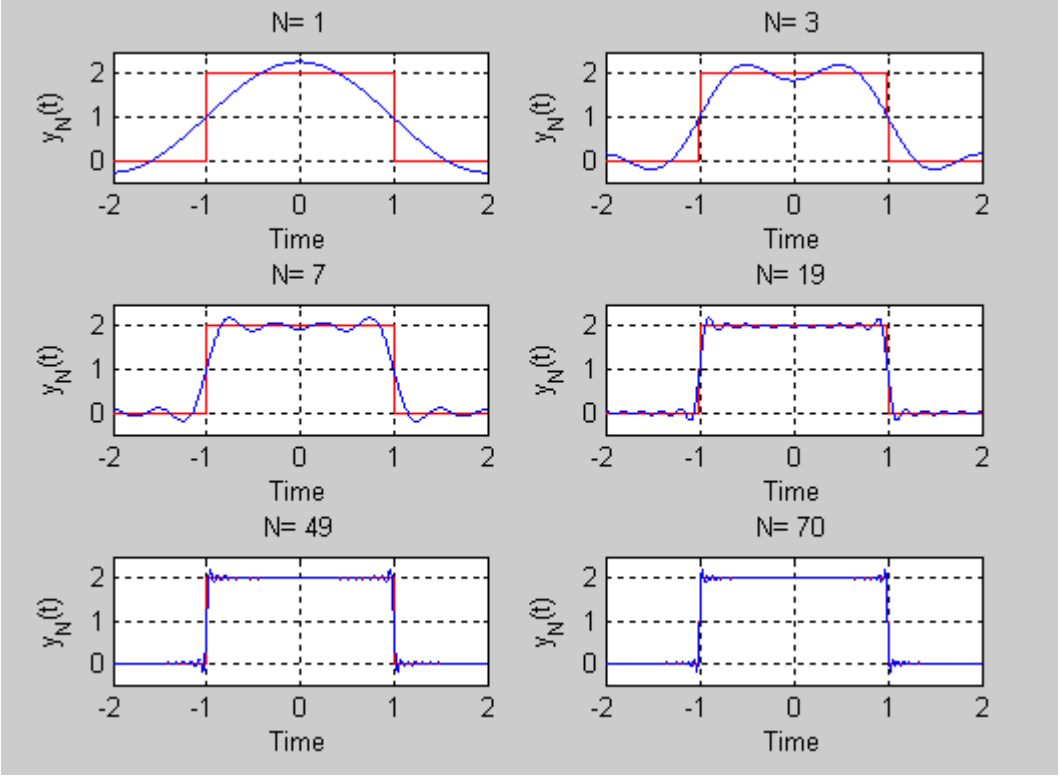
The Gibbs phenomenon involves both the fact that Fourier sums overshoot at a jump discontinuity, and that this overshoot does not die out as the frequency increases.

Program:

```
% Gibb's phenomenon..
clc;
clear all;
close all;
t=linspace(-2,2,2000); u=linspace(-2,2,2000);
sq=[zeros(1,500),2*ones(1,1000),zeros(1,500)];
k=2;
N=[1,3,7,19,49,70];
for n=1:6;
an=[];
for  m=1:N(n)
    an=[an,2*k*sin(m*pi/2)/(m*pi)];
end;
fN=k/2;
for m=1:N(n)
    fN=fN+an(m)*cos(m*pi*t/2);
end; nq=int2str(N(n));
subplot(3,2,n),plot(u,sq,'r');hold on;
plot(t,fN); hold off; axis([-2 2 -0.5 2.5]);grid;
xlabel('Time'), ylabel('y_N(t)');title(['N= ',nq]);
end;
```

Result: In this experiment Gibbs phenomenon have been demonstrated using MATLAB.

Output:



Finding the Fourier Transform of a given signal and plotting its magnitude and phase spectrum

AIM: To find the Fourier Transform of a given signal and plotting its magnitude and phase spectrum.

Theory:

Fourier Transform:

The Fourier transform as follows. Suppose that f is a function which is zero outside of some interval $[-L/2, L/2]$. Then for any $T \geq L$ we may expand f in a Fourier series on the interval $[-T/2, T/2]$, where the "amount" of the wave $e^{2\pi inx/T}$ in the Fourier series of f is given by

By definition Fourier Transform of signal f(t) is defined as

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-j\omega t}\, dt$$

Inverse Fourier Transform of signal F(w) is defined as

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} F(\omega) e^{j\omega t}\, d\omega$$

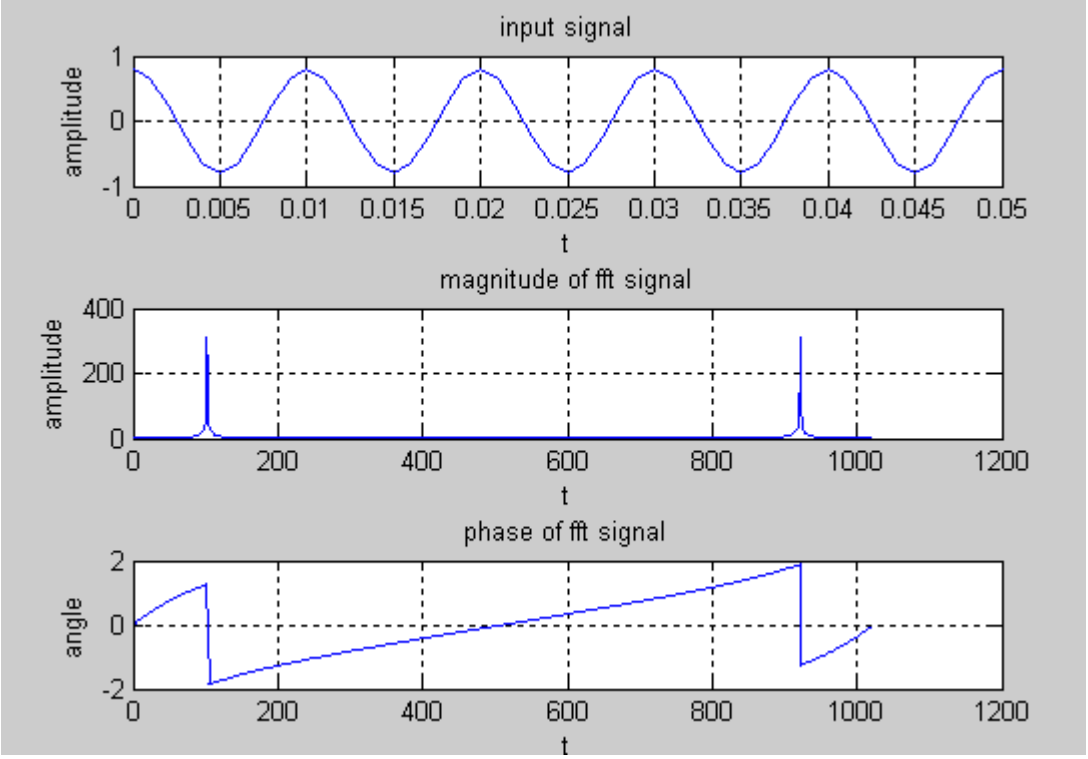
Program:

```
clc;
clear all;
close all;
 fs=1000;
 N=1024;   % length of fft sequence
t=[0:N-1]*(1/fs);
% input signal
x=0.8*cos(2*pi*100*t);
subplot(3,1,1);
plot(t,x);
axis([0 0.05 -1 1]);
grid; xlabel('t');
ylabel('amplitude');
title('input signal');
% Fourier transformof given signal
x1=fft(x);
% magnitude spectrum
k=0:N-1;
Xmag=abs(x1);
subplot(3,1,2);
```

```
plot(k,Xmag);
grid;
xlabel('t');
ylabel('amplitude');
title('magnitude of fft signal')
%phase spectrum
Xphase=angle(x1);
subplot(3,1,3);
plot(k,Xphase);
grid;
xlabel('t');
ylabel('angle');
title('phase of fft signal');
```

Result: Magnitude and phase spectrum of FFT of a given signal is plotted.

Output:


input signal
amplitude
1
0
-1
0
0.005
0.01
0.015
0.02
0.025
0.03
0.035
0.04
0.045
0.05
t
magnitude of fft signal
amplitude
400
200
0
0
200
400
600
800
1000
1200
t
phase of fft signal
angle
2
0
-2
0
200
400
600
800
1000
1200
t

Generation of Gaussian Noise

AIM: Write the program for generation of Gaussian noise and computation of its mean, mean square value, standard deviation, variance, and skewness.

Theory:

Gaussian noise is statistical noise that has a probability density function of the normal distribution (also known as Gaussian distribution). In other words, the valuestha the noise can take on are Gaussian-distributed. It is most commonly used as additive white noise to yield additive white Gaussian noise (AWGN).Gaussian noise is properly defined as the noise with a Gaussian amplitude distribution. says nothing of the correlation of the noise in time or of the spectral density of the noise. Labeling Gaussian noise as 'white' describes the correlation of the noise. It is necessary to use the term "white Gaussian noise" to be correct. Gaussian noise is sometimes misunderstood to be white Gaussian noise, but this is not the case.

Program:

```
clc;
clear all;
close all;

%generates a set of 2000 samples of Gaussian distributed random  numbers
x=randn(1,2000);
%plot the joint distribution of both the sets using dot.
subplot(211)
plot(x,'.');
title('scatter plot of Gaussian distributed random numbers');
ymu=mean(x)
ymsq=sum(x.^2)/length(x)
ysigma=std(x)
yvar=var(x)
yskew=skewness(x)

p=normpdf(x,ymu,ysigma);
subplot(212);
stem(x,p);
title(' gaussian distribution');
```

Output:

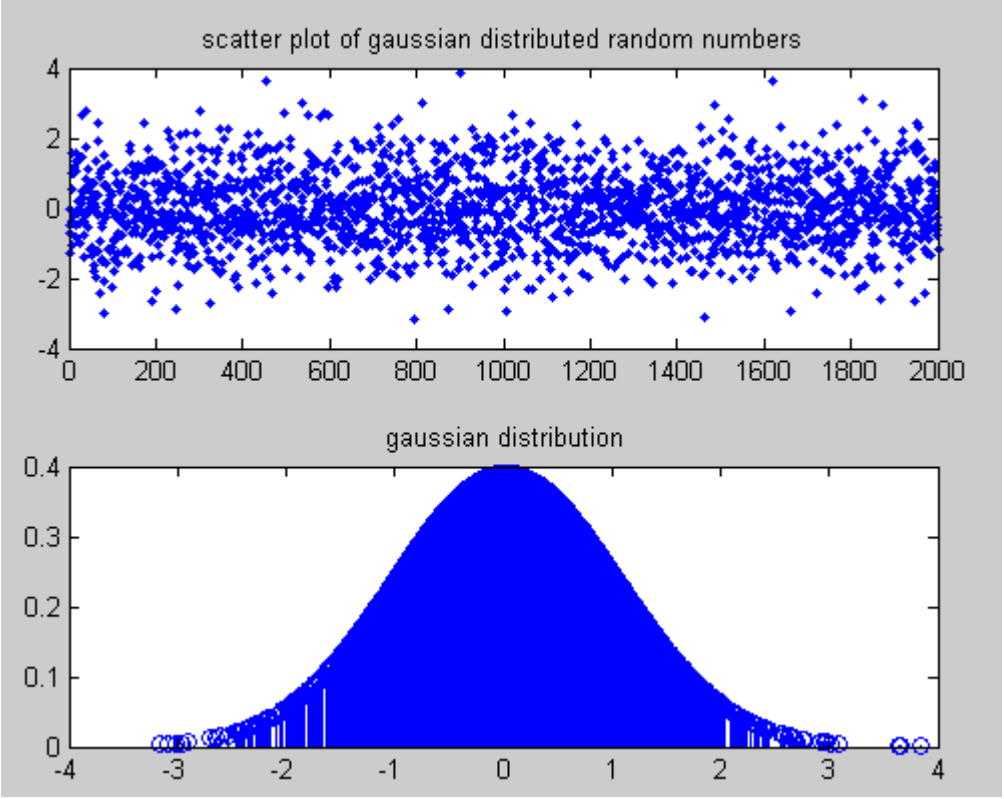
ymu = 0.0403

ymsq = 0.9727

ysigma = 0.9859

yvar = 0.9720

yskew = 0.0049


scatter plot of gaussian distributed random numbers
4
2
0
-2
-4
0
200
400
600
800
1000
1200
1400
1600
1800
2000
gaussian distribution
0.4
0.3
0.2
0.1
0
-4
-3
-2
-1
0
1
2
3
4

Sampling theorem verification

AIM: Verify the sampling theorem.

Theory:

Sampling Theorem:

A bandlimited signal can be reconstructed exactly if it is sampled at a rate at least twice the maximum frequency component in it." Figure 1 shows a signal g(t) that is bandlimited.

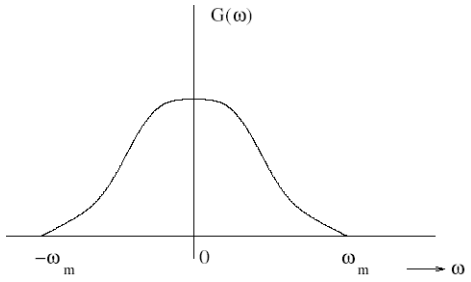


Figure 1: Spectrum of band limited signal g(t)

The maximum frequency component of g(t) is fm. To recover the signal g(t) exactly from its samples it has to be sampled at a rate fs ≥ 2fm.

The minimum required sampling rate fs = 2fm is called ' Nyquist rate

Proof: Let g(t) be a bandlimited signal whose bandwidth is fm (wm = 2πfm).

Figure 2: (a) Original signal g(t) (b) Spectrum G(w)

δ (t) is the sampling signal with fs = 1/T > 2fm.

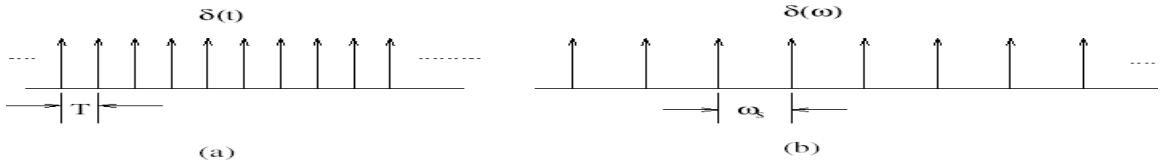


Figure 3: (a) sampling signal δ (t) (b) Spectrum δ (w)

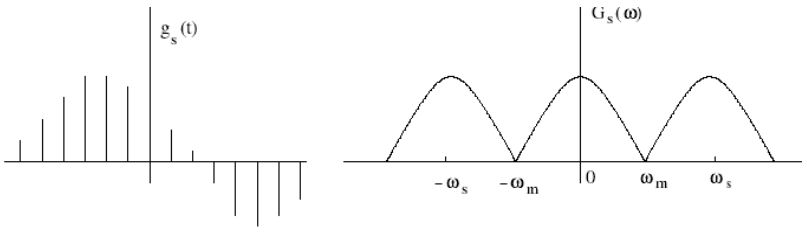


Figure 4: (a) sampled signal gs(t) (b) Spectrum Gs(w)

To recover the original signal G(w):
1. Filter with a Gate function, $H_{2wm}(w)$ of width 2wm
Scale it by T.

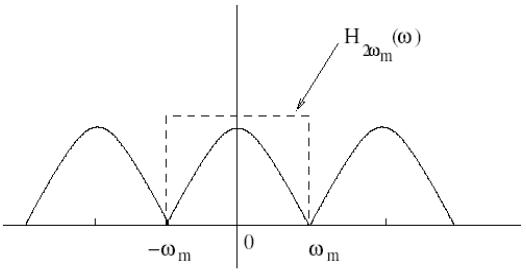


Figure 5: Recovery of signal by filtering with a filter of width 2wm Aliasing ws < 2wm.

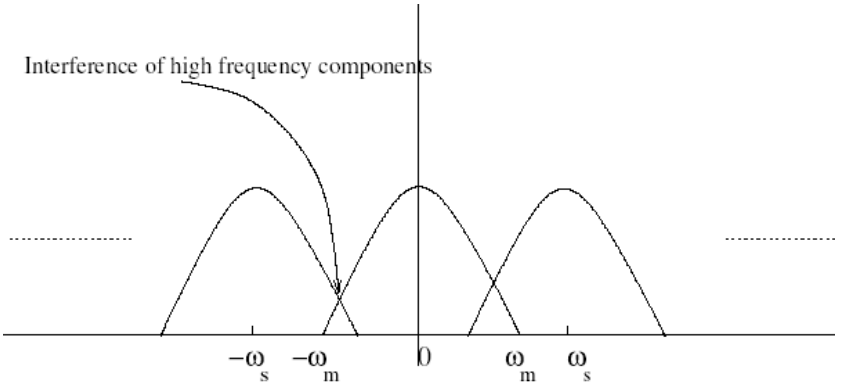


Figure 6: Aliasing due to inadequate sampling

Aliasing leads to distortion in recovered signal. This is the reason why sampling frequency should be at least twice the bandwidth of the signal. Oversampling ws >2wm. This condition avoid aliasing.

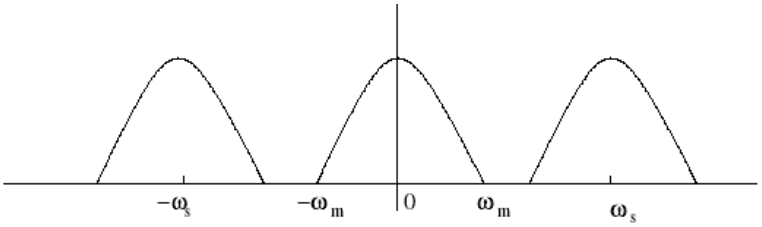


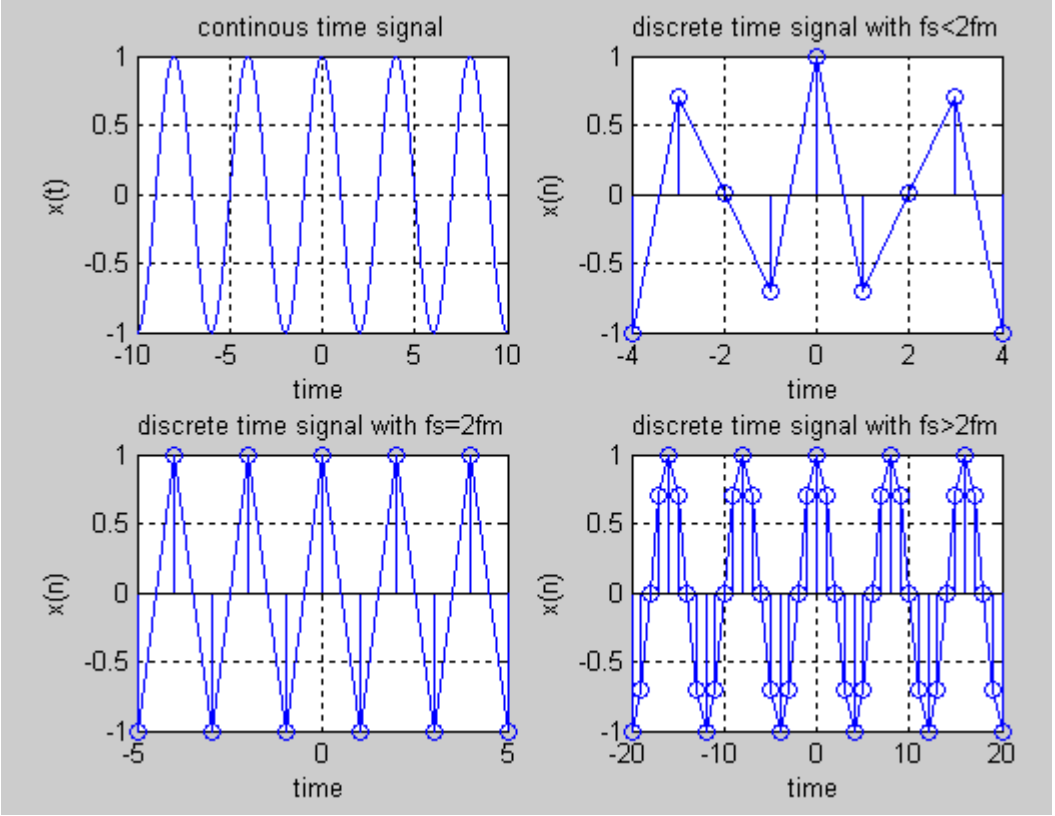
Figure 7: Oversampled signal-avoids aliasing

Program:

```
clc;
clear all;
close all;
t=-10:.01:10;
T=4;
fm=1/T;
x=cos(2*pi*fm*t);
subplot(2,2,1);
plot(t,x);
xlabel('time');
ylabel('x(t)');
title('continous time signal');
grid;
n1=-4:1:4;
fs1=1.6*fm;
fs2=2*fm;
fs3=8*fm;
x1=cos(2*pi*fm/fs1*n1);
subplot(2,2,2);
stem(n1,x1);
xlabel('time');
ylabel('x(n)');
title('discrete time signal with fs<2fm');
hold on;
subplot(2,2,2);
plot(n1,x1);
grid;
n2=-5:1:5;
x2=cos(2*pi*fm/fs2*n2);
subplot(2,2,3);
stem(n2,x2);
xlabel('time');
ylabel('x(n)');
title('discrete time signal with fs=2fm');
hold on;
subplot(2,2,3);
plot(n2,x2)
grid;
n3=-20:1:20;
x3=cos(2*pi*fm/fs3*n3);
subplot(2,2,4);
stem(n3,x3);
xlabel('time');
ylabel('x(n)');
title('discrete time signal with fs>2fm')
hold on;
subplot(2,2,4);
plot(n3,x3)
grid;
```

Result: Sampling theorem is verified.

OUTPUT:


continous time signal
x(t)
time
discrete time signal with fs<2fm
x(n)
time
discrete time signal with fs=2fm
x(n)
time
discrete time signal with fs>2fm
x(n)
time

REMOVAL OF NOISE BY AUTO CORRELATION/CROSS CORRELATION

AIM: Write the program for Removal of noise by using auto correlation.

Theory:

Detection of a periodic signal masked by random noise is of great importance .The noise signal encountered in practice is a signal with random amplitude variations. A signal is uncorrelated with any periodic signal. If s(t) is a periodic signal and n(t) is a noise signal then

$$\lim_{T\to\infty} \frac{1}{T}\int_{-T/2}^{T/2} S(t)n(t-T)\,dt = 0 \quad \text{for all } T$$

$Q_{sn}(T)$= cross correlation function of s(t) and n(t) Then $Q_{sn}(T)=0$

Detection of noise by Auto-Correlation:

S(t)=Periodic Signal (Transmitted) , mixed with a noise signal n(t).

Then f(t) is received signal is [s(t) + n(t)]

Let $Q_{ff}(T)$ =Auto Correlation Function of f(t)
$Q_{ss}(t)$ = Auto Correlation Function of S(t)
$Q_{nn}(T)$ = Auto Correlation Function of n(t)

$$Q_{ff}(T) = \lim_{T\to\infty} \frac{1}{T}\int_{-T/2}^{T/2} f(t)f(t-T)\,dt$$

$$= \lim_{T\to\infty} \frac{1}{T}\int_{-T/2}^{T/2} [s(t)+n(t)]\,[s(t-T)+n(t-T)]\,dt$$

$$= Q_{ss}(T)+Q_{nn}(T)+Q_{sn}(T)+Q_{ns}(T)$$

The periodic signal s(t) and noise signal n(t) are uncorrelated

$$Q_{sn}(t)=Q_{ns}(t)=0 \ ;$$

Then $Q_{ff}(t)=Q_{ss}(t)+Q_{nn}(t)$

The Auto correlation function of a periodic signal is periodic of the same frequency and the Auto correlation function of a non-periodic signal is tends to zero for large value of T since s(t) is a periodic signal and n(t) is non periodic signal so $Q_{ss}(T)$ is a periodic where as a $Q_{nn}(T)$ becomes small for large values of T Therefore for sufficiently large values of T $Q_{ff}(T)$ is equal to $Q_{ss}(T)$.

Program:

```
clc;
clear all;
close all;
t=0:0.2:pi*8;
%input signal
s=sin(t);
subplot(6,1,1);
plot(s);
title('signal s');
xlabel('t');
ylabel('amplitude');
%generating noise
n = randn([1 126]);
%noisy signal
f=s+n;
subplot(6,1,2)
plot(f);
title('signal f=s+n');
xlabel('t');
ylabel('amplitude');
% auto correlation of input
signal as=xcorr(s,s);
subplot(6,1,3);
plot(as);
title('auto correlation of s');
xlabel('t');
ylabel('amplitude');
%aucorrelation of noise signal
an=xcorr(n,n);
subplot(6,1,4)
plot(an);
title('auto correlation of n');
xlabel('t');
ylabel('amplitude');
%aucorrelation of transmitted signal
cff=xcorr(f,f);
subplot(6,1,5)
```

EXTRACTION OF PERIODIC SIGNAL MASKED BY NOISE USING CORRELATION

AIM: Write the program for extraction of periodic signal using correlation.

Theory:

A signal is masked by noise can be detected either by correlation techniques or by filtering. Actually, the two techniques are equivalent. The correlation technique is a measure of extraction of a given signal in the time domain whereas filtering achieves exactly the same results in frequency domain.

Program:

```
clear all;
close all;
clc;
t=0:0.1: pi*4;
%input signal1
s=sin(t);
subplot(7,1,1)
plot(s);
title('signal s');
xlabel('t');
ylabel('amplitude');
%input signal2
c=cos(t);
subplot(7,1,2)
plot(c);
title('signal c');
xlabel('t');
ylabel('amplitude');

%generating noise signal
n = randn([1 126]);
%signal+noise
f=s+n;
subplot(7,1,3);
plot(f);
title('signal f=s+n');
xlabel('t');
ylabel('amplitude');

%crosscorrelation of signal1&signal2
```

```
asc=xcorr(s,c);
subplot(7,1,4)
plot(asc);
title(' correlation of s and c');
xlabel('t');
ylabel('amplitude');
%crosscorrelation of noise&signal2
anc=xcorr(n,c);
subplot(7,1,5)
plot(anc);
title(' correlation of n and c');
xlabel('t');
ylabel('amplitude');
%crosscorrelation of f&signal2
cfc=xcorr(f,c);
subplot(7,1,6)
plot(cfc);
title(' correlation of f and c');
xlabel('t');
ylabel('amplitude');
%extracting periodic signal
hh=asc+anc;
subplot(7,1,7)
plot(hh);
title('addition of sc+nc');
xlabel('t');
ylabel('amplitude');
```

Result: Periodic signal is extracted by using correlation.

Output:

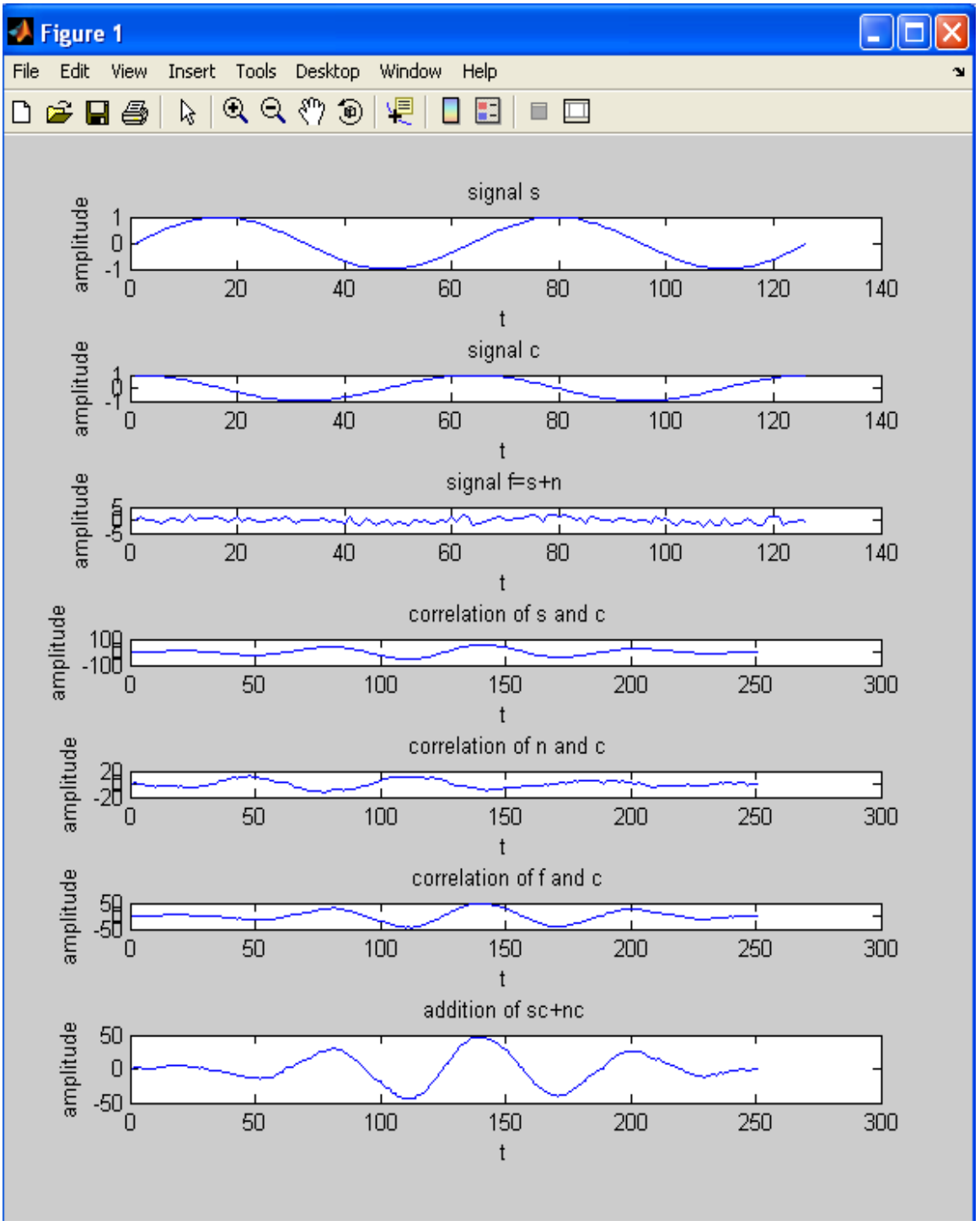


Figure 1
File Edit View Insert Tools Desktop Window Help
signal s
amplitude
t
signal c
amplitude
t
signal f=s+n
amplitude
t
correlation of s and c
amplitude
t
correlation of n and c
amplitude
t
correlation of f and c
amplitude
t
addition of sc+nc
amplitude
t

VERIFICATION OF WIENER-KHINCHIN RELATION

AIM: Verification of Wiener-Khinchin relation

Theory:

The Wiener-Khinchin theorem states that the power spectral density of a wide sense stationary random process is the flourier transform of the corresponding autocorrelation function.

PSD= $S_{XX}(\omega) = FT[R_{XX}(\tau)] = \int_{-\infty}^{\infty} R_{XX}(\tau)e^{-j\omega t}dt$

ACF= $R_{XX}(\tau) = IFT[S_{XX}(\omega)] = \frac{1}{2\pi}\int_{-\infty}^{\infty} S_{XX}(\omega)e^{j\omega t}d\omega$

Program:

```
clc;
clear all;
close all;
t=0:0.1:2*pi;
%input signal
x=sin(2*t);
subplot(3,1,1);
plot(x);
xlabel('time');
ylabel('amplitude');
title('input signal');
%autocorrelation of input signal
xu=xcorr(x,x);
%fft of autocorrelation signal
y=fft(xu);
subplot(3,1,2);
plot(abs(y));
xlabel('f');
ylabel('amplitude');
title('fft of autocorrelation of input signal');
%fourier transform of input signal
y1=fft(x);
%finding the power spectral density
y2=(abs(y1)).^2;
subplot(3,1,3);
plot(y2);
xlabel('f');
```

```
ylabel('magnitude');
title('PSD of input signal');
```

Result: Wiener-Khinchin relation is verified.

Output: