BAPATLA ENGINEERING COLLEGE

DIGITAL SIGNAL PROCESSING LAB

EC-453

PREPARED BY

T. Krishna Chaitanya

Lecturer

Department of Electronics and Communications Engineering
Bapatla Engineering College
(Affiliated to Acharya Nagarjuna University)
Bapatla-522101
2009-2010

List of Experiments

1. Simulation of AM.
2. Simulation of FM.
3. Simulation of DFT and IDFT.
4. (a) Simulation of LPF.
   (b) Simulation of HPF.
5. Simulation of M-ary PSK.
6. Simulation of DPCM.
7. Evaluation of DFT and IDFT of 16 sample sequence using DIT Algorithm.
8. Evaluation of DFT and IDFT of 16 sample sequence using DIF Algorithm.
9. Design of IIR Butterworth Filter using Impulse Invariant Method.
10. Design of FIR Filter using Windowing Technique.
11. Convolution of Two signals.
12. Correlation of Two signals.

1. Simulation of AM

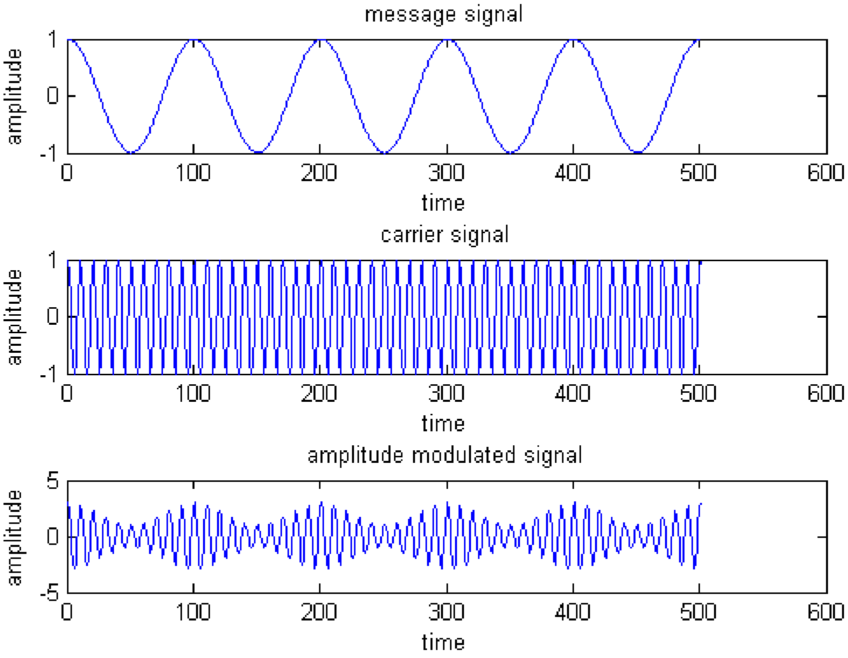
Aim: To simulate amplitude modulation and demodulation.

Program:

```
AM=input('enter message signal amplitude');
Ac=input('enter carrier signal amplitude');
fm=input('enter message signal frequency');
fc=input('enter carrier signal frequency');
n=input('enter no of cycles');
T=1/fm;
t=0:T/100:n*T;
m=AM*cos(2*pi*fm*t);
c=Ac*cos(2*pi*fc*t);
subplot(311);
plot(m);
xlabel('time');
ylabel('amplitude');
title('message signal');
subplot(312);
plot(c);
xlabel('time');
ylabel('amplitude');
title('carrier signal');
ka=AM/Ac;
s=c+c.*(1+ka*m);
subplot(313);
plot(s);
```

```
xlabel('time');
ylabel('amplitude');
title('amplitude modulated signal');
```

Output:



2. Simulation of frequency modulation

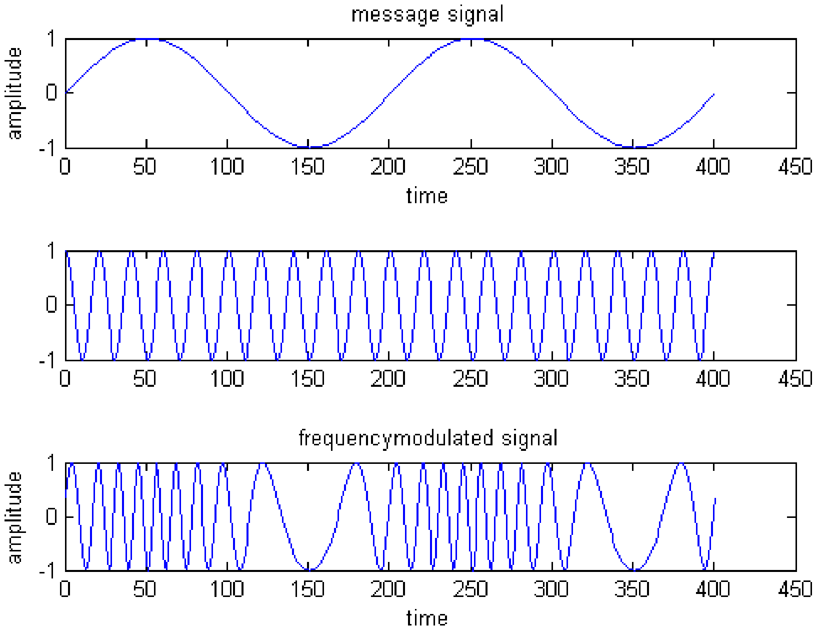
Aim: To simulate frequency modulation and demodulation.

Program:

```
AM=input('enter message signal amplitude');
Ac=input('enter carrier signal amplitude');
fm=input('enter message signal frequency');
fc=input('enter carrier signal frequency');
kf=input('enter frequency sensitivity');
T=1/fm;
t=0:T/200:2*T;
m=AM*sin(2*pi*fm*t);
c=Ac*cos(2*pi*fc*t);
subplot(311);
plot(m);
xlabel('time');
ylabel('amplitude');
title('message signal');
subplot(312);
plot(c);
fi=kf*AM;
b=fi/fm;
s=Ac*cos(2*pi*fc*t-(b*cos(2*pi*fm*t)));
subplot(313);
plot(s);
xlabel('time');
ylabel('amplitude');
```

```
title('frequencymodulated signal');
```

Output:



3. (i) Simulation of Lowpass filter

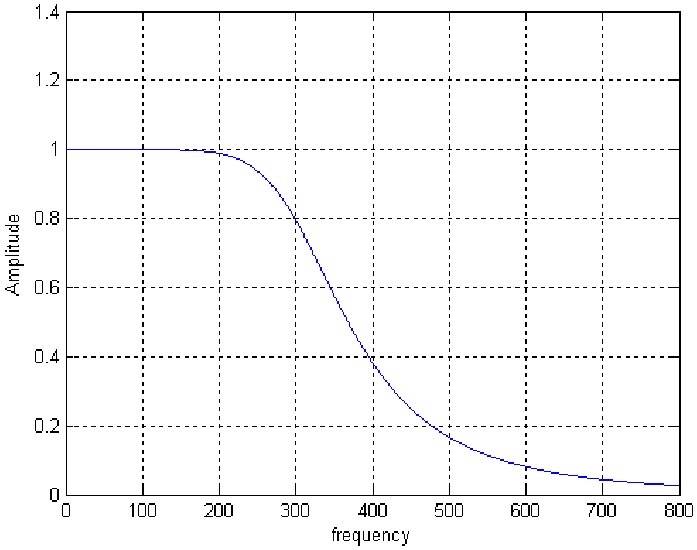
Aim: To simulate frequency response of Lowpass filter

Program:

```
function []=LPF (AP,AS,wp,ws)
b=(10^(0.1*AS)-1)/(10^(0.1*AP)-1);
c=log10(b^0.5);
d=log10(ws/wp);
n=(c/d);
t=ceil(n);
w=0:0.5:800;
m=(10^(0.1*AP)-1)^(1/(2*t));
wc=wp/m;
s=j*w/wc;
switch t
    case 1
        h=1./(s+1);
    case 2
        h=1./((s.^2)+1.414*s+1);
    case 3
        h=1./((s+1).*(s.^2+s+1));
    case 4
h=1./(((s.^2)+0.76537*s+1).*((s.^2)+1.8477*s+1));
    otherwise
        disp('order exceeded');
end;
plot(w,abs(h));
```

```
xlabel('frequency')
ylabel('Amplitude')
grid on
```

Output:



3 (b). Simulation of Highpass Filter

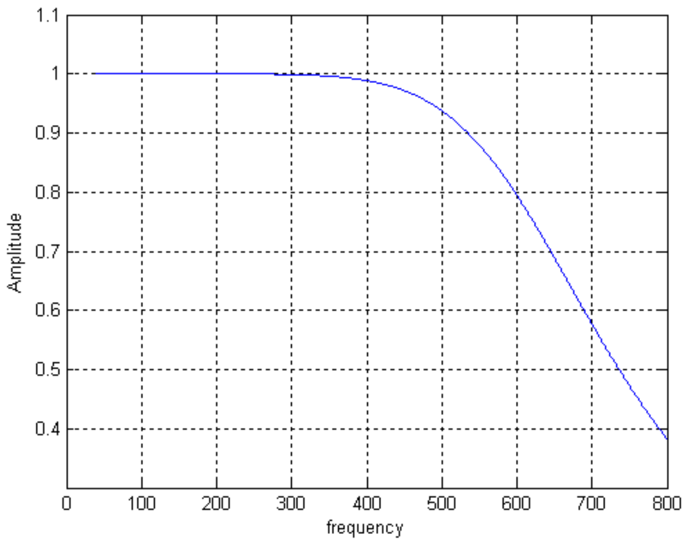
Aim: To simulate frequency response of high pass filter.

Program:

```
function []=HPF (AP,AS,wp,ws)
b=(10^(0.1*AS)-1)/(10^(0.1*AP)-1);
c=log10(b^0.5);
d=log10(wp/ws);
n=(c/d);
t=ceil(n);
w=40:0.4:800;
m=(10^(0.1*AP)-1)^(1/(2*t));
wc=wp/m;
s=-j*wc/w;
switch t
    case 1
        h=1./(s+1);
    case 2
        h=1./((s.^2)+1.414*s+1);
    case 3
        h=1./((s+1).*(s.^2+s+1));
    case 4
h=1./(((s.^2)+0.76537*s+1).*((s.^2)+1.8477*s+1));
    otherwise
        disp('order exceeded');
end;
plot(w,abs(h));
```

```
xlabel('frequency')
ylabel('Amplitude')
grid on
```

Output:



4. Simulation of DFT and IDFT

Aim: To simulate DFT and IDFT.

Program:

```
x=input('enter sequence');
N=length(x);
s=zeros(1,N);

for k=1:N
    for n=1:N
        s(k)=s(k)+x(n)*exp(-j*2*pi*(k-1)*(n-1)/N);
    end
end
disp(s);
% INVERSE DISCRETE FOURIER TRANSFORM.
m=input('enter sequence');  %m=s;
N=length(m);
G=zeros(1,N);
for n=1:N
    for k=1:N
        G(n)=G(n)+m(k)*exp(j*2*pi*(k-1)*(n-1)/N);
    end;
end;
disp(G/N);
```

Output:

```
 enter sequence[1 2 3 4 5]
 Columns 1 through 4
   15.0000            -2.5000 + 3.4410i  -2.5000
+ 0.8123i  -2.5000 - 0.8123i
   Column 5
   -2.5000 - 3.4410i
 enter sequence s
   Columns 1 through 4
    1.0000 - 0.0000i   2.0000 - 0.0000i   3.0000
4.0000 + 0.0000i
   Column 5
    5.0000 + 0.0000i
```

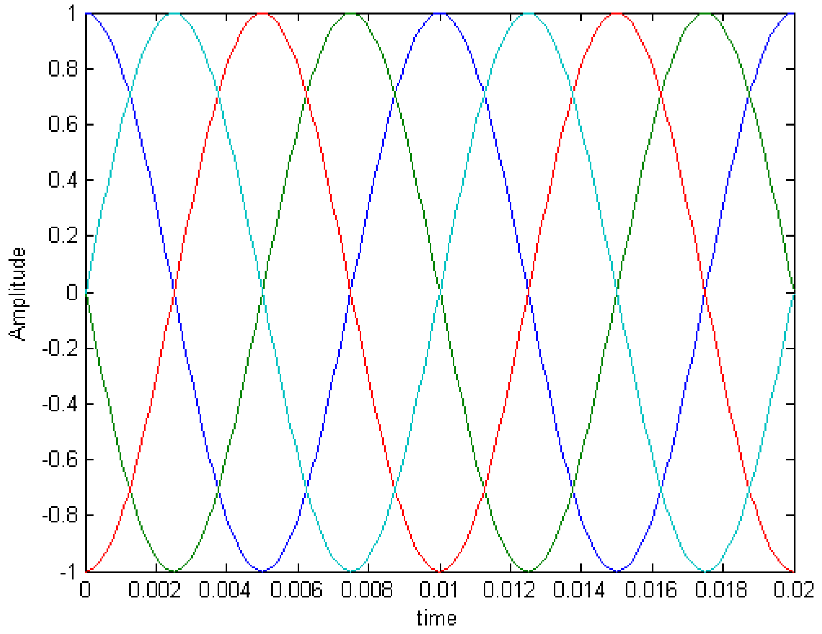
5. Simulation of M- ARY PHASE SHIFT KEYING

Aim: To simulate M-ary PSK.

Program:

```
N=input('enter the no of combinations');
fc=input('enter frequency of the carrier');
ac=input('enter amplitude value');
T=1/fc;
t=0:T/100:2*T;
for n=0:N-1
    s=cos(2*pi*fc*t+2*pi*n/N);
    plot(t,s);
    hold all;
end
xlabel('time');
ylabel('Amplitude');
```

Output:



6. Simulation of DPCM

```
close all
clear
% signal sampling
fs=1/2000;
tn=0:fs:1/25;
%SELECT A SIGNAL ****************************
s=.5*sin(2*pi*50*tn);
%[s,fs]=wavread('bye441');  %read .wav file
 %lpc and encoder-decoder parameters
lpclen=20;
bitsize=input('bitsize=');
fprintf('\nPlease wait... data length is
%i\n',length(s))
%LPF parameters
tap=100;
cf=.15;
 % DPCM with predictor
[Q,b, ai] = dpcm_enco_lpc(s, lpclen, bitsize);
[st]=dpcm_deco_lpc(b, ai, bitsize);
Sa=LPF(tap,cf,st);
%[xa,ya]=stairs(tn,Sa);
figure
subplot(3,1,1):plot(s,'r');
ylabel('amplitude');
title('DPCM with predictor (red:input,
green:decoder output, blue: LPF output)');
subplot(3,1,2):plot(st,'g');
```

```
ylabel('amplitude');
subplot(3,1,3):plot(Sa,'b');
ylabel('amplitude');
xlabel('index, n');
grid
```

Sub Functions:

Dpcm encoding

```
function [Q,b,ai] = dpcm_enco_lpc(s, lpclen,
bitsize)
%function [Q,b, ai] = dpcm_enco_lpc(s, lpclen,
bitsize)
% s : input signal
% bitsize : encoder bit size
% Q : qunatizer output
% b : encoder output
% e = s(i+1) - s(i)
%s = 2^(-1)*s/max(abs(s));
slen = length(s);
e(1) = s(1);
[Q(1),b(1,:)] = pcm_quan_enco(e(1), bitsize);
st(1) = Q(1);
for i=2:slen
   if i<=lpclen
      e(i) = s(i)-st(i-1);
      [Q(i),b(i,:)] = pcm_quan_enco(e(i),
bitsize);
      st(i) = st(i-1) + Q(i);
   else
      m=0;
      [a,G]=lpc(s(i-lpclen:i-1),lpclen);
      a = a*G;
      m = 1:lpclen;
      j=2:lpclen + 1;
      sth = sum(a(j).*st(i-m));
      ai(i,:)=a;
      e(i) = s(i)-sth;
      [Q(i),b(i,:)] = pcm_quan_enco(e(i),
bitsize);
      st(i) = sth + Q(i);
   end
end
b=b';
b=b(:)';
```

LPF for DPCM

```
function Sa=lpf(tap, cf, Sn)
%LPF lowpass filter
%function Sa=LPF(tap,cf,Sn)
%
%tap: filter order.
%cf: cut-off frequency.
%Quantized reconstructed signal.
%Sa: decoder output.
b=fir1(tap,cf);
Sa = conv2(Sn,b,'same');
```

PCM Quantization Encoding:

```
function [Q,B] = pcm_quan_enco(e,bitsize)
%function [Q,b] = pcm_quan_enco(e,bitsize)
% e : input to quantizer
% bitsize : encoder bit size
% Q : qunatazer output
% B : encoder output
 if bitsize<4
  slen = length(e);
  D = 2^(-bitsize);
  switch bitsize
  case 3,
      for i=1:slen
         if e(i) < -3*D
             Q(i) = -(7/2)*D;
          b(i,:) = [ 0 0 0 ];
        elseif e(i) >= -3*D & e(i) < -2*D
            Q(i) = -(5/2)*D;
            b(i,:) = [ 0 0 1 ];
        elseif e(i) >= -2*D & e(i) < -D
            Q(i) = -(3/2)*D;
             b(i,:) = [ 0 1 0 ];
            elseif e(i) >= -D & e(i) < 0
           Q(i) = -(1/2)*D;
             b(i,:) = [ 0 1 1 ];
       elseif e(i) >= 0 & e(i) < D
           Q(i) = (1/2)*D;
            b(i,:) = [ 1 0 0 ];
          elseif e(i) >= D & e(i) < 2*D
          Q(i) = (3/2)*D;
           b(i,:) = [ 1 0 1 ];
            elseif e(i) >= 2*D & e(i) < 3*D
          Q(i) = (5/2)*D;
           b(i,:) = [ 1 1 0 ];
          elseif e(i) >= 3*D
          Q(i) = (7/2)*D;
           b(i,:) = [ 1 1 1 ];
         end
      end
   case 2,
      for i=1:slen
         if e(i) < -D
            Q(i) = -(3/2)*D;
             b(i,:) = [ 0 0 ];
            elseif e(i) >= -D & e(i) < 0
           Q(i) = -(1/2)*D;
            b(i,:) = [ 0 1 ];
         elseif e(i) >= 0 & e(i) < D
            Q(i) = (1/2)*D;
             b(i,:) = [ 1 0 ];
            elseif e(i) >= D
           Q(i) = (3/2)*D;
             b(i,:) = [ 1 1 ];
         end
      end
   case 1,
      for i=1:slen
```

```
         if e(i) < 0
            Q(i) = -(1/2)*D;
             b(i,:) = [ 0 ];
            elseif e(i) >= 0
           Q(i) = (1/2)*D;
            b(i,:) = [ 1 ];
         end
      end
   otherwise
      fprintf('choose a bit size 1,2 or 3.\n');
   end
   b=b';
   B=b(:)';
else
  [b0, b, bb] = dbc(e, bitsize);
  [Q] = bdc(b0,b);
  B = [b0 b];
end
```

Decimal to Binary:

```
function [b0,b,bb]=dbc(x,B);
% [b0,b,bb]=dbc(x,B)
% Decimal-to-Binary conversion using B+1 bit
precision
% x  : a constant in decimal
% B  : number of the precision bit
% b0 : sign bit ( 0 represent + sign, and 1
represents - sign)
% b  : binary bits after the binary point
% bb : (B+1)st bit
B = B-1;
if x>1, error(' x is not normilized.'); end
if x>=0
    b0=0;       % + sign is assigned.
    z=x;
else
    b0=1;       % - sign is assigned.
    z=-x;
end
if z >= 0
    for i=1:B,
    a=2*z;
     if a>=1
    b(i)=1;
    z=a-1;
     else
        b(i)=0;
        z=a;
     end
    end
    a=2*z;
     if a>=1
    bb=1;
      else
        bb=0;
     end
end
```

Dpcm Decoding

```
function [st]=dpcm_deco_lpc(b, ai, bitsize)
%function [st]=dpcm_deco_lpc(b, ai, bitsize)
% b : input to decoder from communication channel
% bitsize : encoder bit size
% st : s_tilda (decoder output to lpf)
[jj,size_ai] = size(ai);
[Q] = pcm_deco_quan(b, bitsize);
st=cumsum(Q(1:size_ai-1));
slen=length(Q);

m=1:size_ai-1;
j=2:size_ai;
for i=size_ai:slen
   sth = sum(ai(i,j).*st(i-m));
   st(i) = Q(i) + sth ;
end
```

Binary-to-Decimal:

```
function [x]=bdc(b0,b);
%  [X]=BDC(b0,b)
%  Binary-to-Decimal  conversion
% b0 : sign bit ( 0 represent + sign, and 1
represents - sign)
% y  : binary bits after the binary point
% x  : a constant in decimal
N=length(b);      % finds the bit precision, B
y=0;
for i=1:N,
y=y+b(i)*2^(-i);  % for  x >0, converts from
binary to decimal
end
x=-b0+y;
if x < 0
[b0,b,bb]=dbc(x,N+1); %+1 bit is for sign
y=0;
  for i=1:N,
  y=y+b(i)*2^(-i);    end
end
x=-b0+y;
```

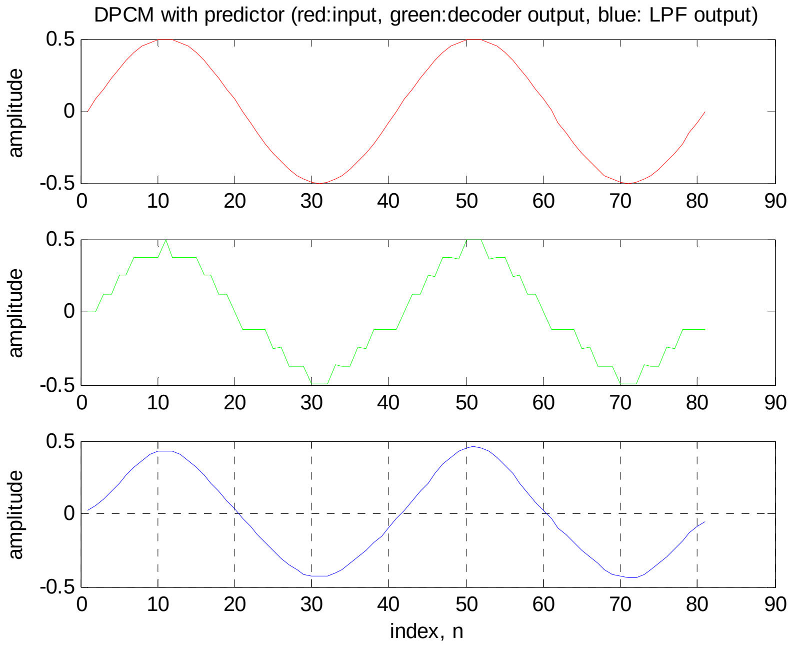
PCM Quantization Decoding:

```
function [Q] = pcm_deco_quan(B,bitsize)
% function [Q,b] = pcm_deco_quan(b,bitsize)
% pcm decoder and quantizer
% bitsize : encoder bit size
% B : input to decoder from encoder ouput
% Q : qunatazer output
if bitsize<4
  b=B;
  slen = length(b);
  D = 2^(-bitsize);
  i = 0;
  for j=1:bitsize:slen
   mask=j:j+bitsize-1;
   i = i+1;
   switch bitsize
      case 3,
         if b(mask) == [ 0 0 0 ]
             Q(i) = -(7/2)*D;
        elseif b(mask) == [ 0 0 1 ]
            Q(i) = -(5/2)*D;
         elseif b(mask) == [ 0 1 0 ]
            Q(i) = -(3/2)*D;
            elseif b(mask) == [ 0 1 1 ]
           Q(i) = -(1/2)*D;
       elseif b(mask) == [ 1 0 0 ]
           Q(i) = (1/2)*D;
          elseif b(mask) == [ 1 0 1 ]
          Q(i) = (3/2)*D;
```

```
            elseif b(mask) == [ 1 1 0 ]
          Q(i) = (5/2)*D;
          elseif b(mask) == [ 1 1 1 ]
          Q(i) = (7/2)*D;
         end
      case 2,
         if b(mask) == [ 0 0 ]
            Q(i) = -(3/2)*D;
            elseif b(mask) == [ 0 1 ]
           Q(i) = -(1/2)*D;
         elseif b(mask) == [ 1 0 ]
            Q(i) = (1/2)*D;
            elseif b(mask) == [ 1 1 ]
           Q(i) = (3/2)*D;
       end
      case 1,
         if b(mask) == [ 0 ]
            Q(i) = -(1/2)*D;
            elseif b(mask) == [ 1 ]
           Q(i) = (1/2)*D;
         end
      otherwise
         fprintf('choose a bit size 1,2 or
3.\n');
      end
   end
else
  slen=length(B)
  i = 0;
  for j=1:bitsize:slen
     i = i + 1;
   mask=j:j+bitsize-1;
     bb = B(mask);
     b0 = bb(1);
     b = bb(2:bitsize);
     Q(i)=bdc(b0,b);
  end
end
```

Output:

```
bitsize=4
data length is 81
slen =324
```



7. (i) DFT USING DIT

```
function b=ditdft(a)
a=bitrevorder(a);
N=length(a);
l=log2(N);
for     m=1:1:l
for t=0:1:((2^(m-1))-1)
    k(t+1)=(N*t)/2^m;
    wn(t+1)=exp((-2*i*pi*k(t+1))/N);
end
for p=1:2^m:(N-2^(m-1))
    r=1;
    x=p;
    for q=x+2^(m-1):1:(x+2^m)-1
        b(p)=a(p)+a(q)*wn(r);
        b(q)=a(p)-a(q)*wn(r);
        p=p+1;
        r=r+1;
    end
end
end
return;
Output:
 a=[1 2 2 2]
 b=ditdft(a)          b=[7    -1    -1    -1]
```

7. (ii)IDFT USING DIT

```
function a =idftdit(a)
a=conj(a);
a=ditdft(a);
a=(a/length(a));
a=conj(a);
return;
 a=[7 -1 -1 -1];
b=idftdit(a)
     b= 1     2     2     2
```

8. (i)FFT USING DIF

```
function a=difdft(a)
N=length(a);
l=log2(N);
for     m=l:-1:1
for t=0:1:((2^(m-1))-1)
    k(t+1)=(N*t)/2^m;
    wn(t+1)=exp((-2*i*pi*k(t+1))/N);
end
for p=1:2^m:(N-2^(m-1))
    r=1;
    x=p;
    for q=x+2^(m-1):1:(x+2^m)-1
        b(p)=a(p)+a(q);
        b(q)=(a(p)-a(q))*wn(r);
        p=p+1;
        r=r+1;
    end
end
a=b;
end
a=bitrevorder(a);
return;
```

<u>Output:</u>

```
a=[1 2 3 4]
X=dftdif(a)
X =  10.0000            -2.0000 + 2.0000i  -
2.0000            -2.0000 - 2.0000i
```

8. (ii)IDFT USING DIF

```
function r =idftdif(a)
a=conj(a);
a=difdft(a);
a=(a/length(a));
r=conj(a);
return;
```

Output:

```
 A =  10.0000            -2.0000 + 2.0000i  -
2.0000            -2.0000 - 2.0000i
r =idftdif(a)
a=[1  2  3  4]
```

9. DESIGN OF BUTTERWORTH FILTER USING IMPULSEINVARIENT METHOD

```
function [b,a] =butteriit(wp,ws,ap,as,t);
p=(10^(0.1*ap)-1)^0.5;
s=(10^(.1*as)-1)^0.5;
n=ceil(log(p/s)/log(wp/ws));
wc=wp/(p^(1/n)*t);
[u,v]=butter(n,wc,'s');
[r,p,k]=residue(u,v);
p=exp(p*t);
[b,a]=residue(r,p,k)
```

Output:

```
[b,a] =butteriit(.2*pi,.3*pi,7,16,1)
  a =
     1.0000   -2.0233    1.4675   -0.3690
  b =
          0    0.0438    0.0314
```

10.DESIGN OF LPF USING HAMMING WINDOW

```
function []=hamm(wp,ws)
T=ws-wp;
wc=ws-wp;
wc=(ws+wp)/2;
M=ceil(6.6*pi/t)+1;
hd=ideal-lp(wc,M);
function hd=ideal-lp(wc,M)
x=(M-1)/2;
n=[0:M-1];
m=n-x+eps;
hd=sine(wc*m)./(pi*m);
for i=1:m
w-ham(1)=0.54-.46*cos(2*pi*()i-1)/M;
end
M=hd.*w-ham;
subplot(2,2,1);stem(h,hd);titll('Ideal Imp
Responce');
axis([0 M-1-.1 0.4]);Xlable('n');Ylble('hd(n));
subplot(2,2,2);steam(n,w-ham);
title('Hamming window')
axis([0 M-1 0 1.1);Xlable('n');Ylable('hd(n));
subplot(2,2,3);stem(n,H);title('Actual Imp
Responce')
axis([0 N-1 .1 .4]);Xlable('n');
Y lable('H(n)');
```

11. (i) LINEAR CONVOLUTION

```
x=input ('Enter first sequence;')
h=input ('Enter Second sequence;')
m=length(x);
n=length (h);
l=m+n-1;
for i=1:l
    r(i)=0;
end
for k=1:l
    for j=max(1,k+1-n):min(k, m)
        r(k)=r(k)+x(j)*h(k+1-j);
    end
end
```

```
Enter first sequence; [1 2 3]
Enter Second sequence; [1 2 3 4]
r = [ 1     4    10    16    17    12]
```

11. (ii) CIRCULAR CONVOLUTION

```
a=input('Enter first sequence:');
b=input('Enter Second sequence:');
x=length(a);
y=length(b);
z=max(x,y);
if x>y
    for i=y+1:x
        b(i)=0;
    end
elseif x<y
    for j=x+1:y
    a(j)=0
        end
end
for i=1:z
     c(i)=0;
    for k=1:z
    j=(i-k)+1;
    if (j<=0)
        j=z+j;
    end
    c(i)=c(i)+(a(k)*b(j));
    end
end
```

Enter 1st sequence: [1 2 2 2]

Enter 2nd sequence: [1 2 3]

```
          c =[   11    10     9    12]
```

12. (i) AUTO-CORRELATION

```
a=input('Enter the sequence:');
n1=length (a);
 n=2*n1-1;
b=zeros(1,n);
c=zeros(1,n);
d=zeros(1,n);
n2=n-n1;
 for i=1:n1
b(i)=a(i);
end
 for i=n2:n-1
c(i+1)=a(i+1-n2);
end
for i=1:n
d(i)=sum(b.*c);
for j=n:-1:2
b(j)=b(j-1);
end
b(i)=0;
end
Enter first sequence: [1 2 3 4]
Ans = [4  11  20  30  20  11  4]
```

12. (ii) CROSS-CORRELATION

```
x=input('Enter first sequence:');
h1=input('Enter Second sequence:');
m=length(x);
n=length(h1);
h(n:-1:1)=h1(1:n);
l=m+n-1;
for i=1:l
    r(i)=0;
end
for k=1:l
    for j=max(1,k+1-n):min(k,m)
        r(k)=r(k)+x(j)*h(k+1-j);
    end
end

Enter first sequence:[1 2 2 2]
Enter Second sequence:[2 2 3]
ans: r =[  3     8    12    14     8     4]
```