1.clear all;

%Function Analog to Digital Convertor

%A=input('Enter the Amplitude of Sine wave');

L= input('no of Samples');

N= input('the Bit Length in Signed Value');

%f=input('Enter the Fequency of the sine wave : ');

fs= input('Sampling Frequency');

x=(1:L);

n=x/fs;

y=sin(2\*pi\*n);

M=round(y\*(2^(N-1)));

for i=1:length(M)

if M(i) >=0

M(i)=M(i)-1;

else

M(i)=M(i)+1;

end

end

for i=1:length(M)

if M(i) >=0

t=dec2bin(M(i));

L=length(t);

bin(i,1:N)=[48\*ones(1,8-L),t];

end

if M(i) < 0

N(i)=-M(i);

t=dec2bin(N(i));

L=length(t);

bin(i,1:N)=[49,48\*ones(1,8-L-1),t];

end

end

fid=fopen('data.txt','wb');

for i=1:length(bin)

fwrite(fid,bin(i,1:8),'int8');

fprintf(fid,'\n');

end

fclose(fid);

2. m=[];

n=[];

%generate the input sinusoidal signal

f=20/500;

N=input('type in length of the input sequence=');

A=input('type in amplitude =');

W0=2\*pi\*f;

n=0:.525:N-.525;m=n-1;t=0:.525:N-.525;

x1=A\*cos(W0\*t);

plot(x1);

pause

x=A\*cos(W0\*n);

axis([0 N -A A]);

stem(m,x);

xlabel('time index');

ylabel('amplitude');

title('input digital signal');

pause

%generation of quantised output %delta sigma modulation

y=zeros(1,N+1);

a=zeros(1,N+1);

e1=[];

e=0;

for k=2:length(n)

a(k)=x(k-1)-e;

if a(k)>=0

y(k)=1;

else

y(k)=-1;

end

e=y(k)-a(k);

e1=[e,e1];

end

%yn=y(1:N+1);

axis([0 N -1 1]);

%plot the quantised output

stem(t,y);

%tairs(t,y);

xlabel('time');

ylabel('amplitude');

title('digital output of sigma delta quantizer');

**3. %generate the input sinusoidal signal**

N=input('type in length of the input sequence=');

A=input('type in amplitude =');

W0=2\*pi\*20;

n=0:.125:N-.125;

m=n-1;

t=0:.125:N-.125;

x1=A\*cos(W0\*t);

plot(x1);

pause

x=A\*cos(W0\*n);

axis([0 N -A A]);

stem(m,x);

xlabel('time index');

ylabel('amplitude');

title('input digital signal');

pause

%generation of quantised output %delta sigma modulation

y=zeros(1,N+1);

a=zeros(1,N+1);

e1=[];

e=0;

for k=2:length(n)

a(k)=x(k-1)-e;

if a(k)>=0

y(k)=1;

else

y(k)=-1;

end

e=y(k)-a(k);

e1=[e,e1];

end

%yn=y(1:N+1);

axis([0 N -1 1]);

%plot the quantised output

stem(t,y);

%tairs(t,y);

xlabel('time');

ylabel('amplitude');

title('digital output of sigma delta quantizer');

pause

z=(y+1)/.2 >.5

samples\_per\_bit=1;

%generatinf bits

no\_of\_bits=length(z);

Bits=z;

%generating bpsk signals

[syms]=2\*Bits - 1;

bit=z

%bit=input('give 8 bit vector');

%x1=A\*cos(W0\*t);

figure(2);

bpsk111(bit);

4) function filteredx = adapfilt(preamble,received);

y=preamble;

lpre=length(y);

x=received(1:lpre);

lrec=length(received);

%define step size

counter=0;

Rxx\_current=zeros(3);

for(i=3:lpre)

counter=counter+1;

Rxx\_current=[x(i) x(i-1) x(i-2)]'\*[x(i) x(i-1) x(i-2)]+Rxx\_current;

end

Rxx\_hat=Rxx\_current./counter;

mu=(1/3)/(sum(eig(Rxx\_hat)));

%initial guesses at n=0

y\_hat(1)=0;

w1(1)=0;

w2(1)=0;

w3(1)=0;

y\_hat(2)=w1(1)\*x(1);

w1(2)=w1(1)+mu\*x(1)\*(y(2)-y\_hat(2));

w2(2)=w2(1);

w3(2)=w3(1);

y\_hat(3)=w1(2)\*x(2)+w2(2)\*x(1);

w1(3)=w1(2)+mu\*x(2)\*(y(3)-y\_hat(3));

w2(3)=w2(2)+mu\*x(1)\*(y(3)-y\_hat(3));

w3(3)=w3(2);

for(k=4:lpre)

y\_hat(k)=w1(k-1)\*x(k-1)+w2(k-1)\*x(k-2)+w3(k-1)\*x(k-3);

w1(k)= w1(k-1) + mu\*x(k-1)\*(y(k)-y\_hat(k));

w2(k)= w2(k-1) + mu\*x(k-2)\*(y(k)-y\_hat(k));

w3(k)= w3(k-1) + mu\*x(k-3)\*(y(k)-y\_hat(k));

end

filteredx=conv(received((lpre+1):lrec),[w1(lpre) w2(lpre) w3(lpre)]);

5) Fs = 5000; % Sampling frequency

t= [0:1/Fs:1];

p = 40; % Linear predictive order

An = 0.0; % Noise amplitude

% Least-squares FIR filter that preserves constants

f1 = 60; f2 = 180;

x=1.5 + sin(2\*pi\*f1\*t) + sin(2\*pi\*f2\*t) + An\*randn(size(t));

corr = xcorr(x,x,p,'biased');

R = toeplitz(corr(p+1:2\*p+1));

b = R\ones(p+1,1);

b = b/sum(b);

boxcar = ones(1,p+1)/(p+1);

y = filter(b,1,x);

yave = filter(boxcar,1,x);

L = 200;

subplot(211);

plot(t(1:L),x(1:L),t(1:L),yave(1:L),'g',t(1:L),y(1:L),'r');

subplot(212);

N = 256;

f = [0:N/2]\*Fs/N;

H = fft(b, N);

Hboxcar = fft(boxcar, N);

semilogy(f,abs(H(1:N/2+1)),'r',f,abs(Hboxcar(1:N/2+1)),'g');

grid

ax = axis;

line([f1 f1],[ax(3) ax(4)]);line([f2 f2],[ax(3) ax(4)]);

6) N=input('enter the no of array values');

N=7;

i=1:7;

x1(i)=[ones(1,7)]

for i=1:7

x2(i)=exp(-2\*i).\*x1(i);

x3(i)=exp(j\*((2\*i)+(pi)./4));

x4(i)=cos(i);

x5(i)=((0.5).^i).\*x1(i);

x6(i)=sin((pi)./(4)\*i);

end

a=2;

for i=1:7

y1(i)=a.\*x1(i);

y2(i)=a.\*x2(i);

y3(i)=a.\*x3(i);

y4(i)=a.\*x4(i);

y5(i)=a.\*x5(i);

y6(i)=a.\*x6(i);

end

subplot(12,1,1);plot(x1);

subplot(12,1,2);plot(x2);

subplot(12,1,3);plot(x3);

subplot(12,1,4);plot(x4);

subplot(12,1,5);plot(x5);

subplot(12,1,6);plot(x6);

subplot(12,1,7);plot(y1);

subplot(12,1,8);plot(y2);

subplot(12,1,9);plot(y3);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(y5);

subplot(12,1,12);plot(y6);

7) % Program illustrating amplitude scaling operation on the given input sequence by an amount 1

N=input('enter the no of sample points of discrete time sequence');

N=6;

for i=1:N

x(i)=input('enter the input');

end

%the following loop indicates the time reversal sequence of the input sequence

a=2;

for i=1:N

y(i)=a.\*x(i);

end

subplot(4,1,1),plot(x);

subplot(4,1,2),plot(y);

subplot(4,1,3),stem(x);

subplot(4,1,4),stem(y);

8)

%to generate the spreaded gold code sequence vector

i=input('enter the number of shift registers');

n=input('enter the no of input bits to the shift registers');

%n=4

m=input('enter the no of pn sequences to be generated');

%m=2

for i=1:m

for i=1:4

a(i)=input('enter the array values for a(i)');

end

for i=1:15

y=xor(a(2),a(4));

a(4)=a(3);

a(3)=a(2);

a(2)=a(1);

a(1)=y;

pn(i)=[a(4)];

end

end

gc=xor(pn(i),pn(i));

data=input('enter the input data');

for i=1:data

d(i)=input('enter the array values for d(i)');

end

out=[];

for k=1:data

for j=1:length(gc)

y1=xor(d(i),gc(i));

out=[out,y1];

end

end

8) %MATLAB PROGRAM TO DISPLAYY ALL BASIC ANALOGUE SIGNALS

%impulse function

N=input('enter the number of array values');

N=16;

t=1:N;

xt=[zeros(1,N),ones(1,1),zeros(1,N)]

subplot(7,1,1);plot(xt);

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

x1t=[ones(1,N)];

subplot(7,1,2);plot(x1t);

xlabel('time');ylabel('magnitude');

title('unit step sequence');

x2t=t.\*x1t;

subplot(7,1,3);plot(x2t);

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

a=1.5;

x3t=exp(j\*a\*t);

x4t=exp(-j\*a\*t);

x5t=(x3t+x4t)./2;x6t=(x3t-x4t)./(2\*j);

subplot(7,1,4);plot(x3t);

xlabel('time');ylabel('magnitude');

title('exponentially increasing signal');

subplot(7,1,5);plot(x4t);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(7,1,6);plot(x5t);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

subplot(7,1,7);plot(x6t);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

9) % A D A P T I V E F I L T E R

Fs = 5000; %Sampling frequency

t = [0:1/Fs:1];

p =40; %Linear predictive order

An= 0.0; %Noise Amplitude

%Least-squares FIR filter that preservs constants

f1 = 60;

f2 = 180;

x=1.5 + sin(2\*pi\*f1\*t) + sin(2\*pi\*f2\*t) +An\*randn(size(t));

corr = xcorr(x,x,p,'biased');

R = toeplitz(corr(p+1:2\*p+1));

b = R\ones(p+1,1);

b = b/sum(b);

boxcar = ones(1,p+1)/(p+1);

y = filter(b,1,x);

yave =filter(boxcar,1,x);

L = 200;

subplot(211);

plot(t(1:L),x(1:L),t(1:L),yave(1:L),'g',t(1:L),y(1:L),'r');

subplot(212);

N = 256;

f = [0:N/2]\*Fs/N;

H = fft(b,N);

Hboxcar = fft(boxcar, N);

semilogy(f,abs(H(1:N/2+1)),'r',f,abs(Hboxcar(1:N/2+1)),'g');

grid

ax = axis;

line([f1 f1],[ax(3) ax(4)]);

line([f2 f2],[ax(3) ax(4)]);

9) % A D A P T I V E F I L T E R

Fs = 5000; %Sampling frequency

t = [0:1/Fs:1];

p =40; %Linear predictive order

An= 0.0; %Noise Amplitude

%Least-squares FIR filter that preservs constants

f1 = 60;

f2 = 180;

x=1.5 + sin(2\*pi\*f1\*t) + sin(2\*pi\*f2\*t) +An\*randn(size(t));

corr = xcorr(x,x,p,'biased');

R = toeplitz(corr(p+1:2\*p+1));

b = R\ones(p+1,1);

b = b/sum(b);

boxcar = ones(1,p+1)/(p+1);

y = filter(b,1,x);

yave =filter(boxcar,1,x);

L = 200;

subplot(211);

plot(t(1:L),x(1:L),t(1:L),yave(1:L),'g',t(1:L),y(1:L),'r');

subplot(212);

N = 256;

f = [0:N/2]\*Fs/N;

H = fft(b,N);

Hboxcar = fft(boxcar, N);

semilogy(f,abs(H(1:N/2+1)),'r',f,abs(Hboxcar(1:N/2+1)),'g');

grid

ax = axis;

line([f1 f1],[ax(3) ax(4)]);

line([f2 f2],[ax(3) ax(4)]);

10) %ANALOG TO DIGITAL CONVERSION

clear all;

%FUNCTION analog to digital comverter

%A=input('enter the amplitude of sinwave');

L= input('no of samples');

N=input('the Bit length in signed value');

%f=input( enter the frequency of the sine wave:');

fs= input ('sampling frequency');

x=(1:L);

n=x/fs;

y=sin(2\*pi\*n);

M=round(y\*(2^(N-1)));

for i=1:length(M)

if M(i) >=0

M(i)=M(i)-1;

else

M(i)=M(i)+1;

end

end

for i=1:length(M)

if M(i) >=0

t=dec2bin(M(i));

L=length(t);

bin(i,1:N)=[48\*ones(1,8-L),t];

end

if M(i) <0

N(i)=-M(i);

t=dec2bin(N(i));

L=length(t);

bin(i,1:N)=[49,48\*ones(1,8-L-1),t];

end

end

fid=fopen('data.txt','wb');

for i=1:length(bin)

fwrite(fid,bin(i,1:8),'int8');

fprintf(fid,'\n');

end

fclose(fid);

11) %ANALOG TO DIGITAL CONVERSION

clear all;

%FUNCTION analog to digital comverter

%A=input('enter the amplitude of sinwave');

L= input('no of samples');

N= input('the Bit length in signed value');

%f=input( enter the frequency of the sine wave:');

fs= input ('sampling frequency');

x=(1:L);

n=x/fs;

y=sin(2\*pi\*n);

M=round(y\*(2^(N-1)));

figure(1);

plot(x,M);

for i=1:length(M)

if M(i) >=0

M(i)=M(i)-1;

else

M(i)=M(i)+1;

end

end

figure(2);

plot(x,M);

for i=1:length(M)

if M(i) >=0

t=dec2bin(M(i));

L=length(t);

bin(i,1:N)=[48\*ones(1,8-L),t];

end

if M(i) <0

N(i)=-M(i);

t=dec2bin(N(i));

L=length(t);

bin(i,1:N)=[49,48\*ones(1,8-L-1),t];

end

end

fid=fopen('c:\MATLABR11\data\data.txt','wb');

for i=1:length(bin)

fwrite(fid,bin(i,1:8),'int8');

fprintf(fid,'\n');

end

fclose(fid);

12) %ANALOG TO DIGITAL CONVERSION

clear all;

%FUNCTION analog to digital comverter

%A=input('enter the amplitude of sinwave');

L= input('no of samples');

N= input('the Bit length in signed value');

%f=input( enter the frequency of the sine wave:');

fs= input ('sampling frequency');

x=(1:L);

n=x/fs;

y=sin(2\*pi\*n);

M=round(y\*(2^(N-1)));

figure(1);

plot(x,M);

for i=1:length(M)

if M(i) >=0

M(i)=M(i)-1;

else

M(i)=M(i)+1;

end

end

figure(2);

plot(x,M);

for i=1:length(M)

if M(i) >=0

t=dec2bin(M(i));

L=length(t);

bin(i,1:N)=[48\*ones(1,8-L),t];

end

if M(i) <0

N(i)=-M(i);

t=dec2bin(N(i));

L=length(t);

bin(i,1:N)=[49,48\*ones(1,8-L-1),t];

end

end

fid=fopen('c:\MATLABR11\data\data.txt','wb');

for i=1:length(bin)

fwrite(fid,bin(i,1:8),'int8');

fprintf(fid,'\n');

end

fclose(fid);

13) % C D M A

%CDMA Transmits signals using CDMA.

%CDMA(signals) Given a matrix of signals (sampled in the time domain),

%will use CDMA to transmit signals through

%communication channel.uses adaptive filtering and thresholding

%in conjunction with CDMA to recover signals.

%if nargin<1

% error('please provide input signals.');

%end;

clear

%step 1:Chunk song into N length pieces(N=100)

%Sig1 is the entire signal of the first song(user)

disp('Wav2vec')

song1 = wavread('c:\WINDOWS\MEDIA\Chord.wav');

song2 = wavread('c:\WINDOWS\MEDIA\Ding.wav');

figure(100)

subplot(1,2,1),plot(song1);

title('original laser');

subplot(1,2,2),plot(song2);title('original down waves');

pause;

%[sig1, num\_zeros] = wav2vec(song1);

[sig2, num\_zeros] =wav2vec(song2);

[rowSig1, colSig1] = size(sig1);

[rowSig2, colSig2] = sizw(sig2);

minrow = min(rowSig1,rowSig2);

%% TESTING!!!

%num\_zeros = 0;

%Sig1(1,1:128)= [ones(1,64),zeros(1,64)];

%Sig1(2,1:128) = [zeros(1,64),ones(1,64)];

%signal1 = [sig1(1,:) sig1(2,:)];

%figure(300),plot(signal1),title('signal1');

%[rowSig1,colSig1]= size(sig1);

%set first row of sig1 to the first row of the sender signal

preamble=[ones(1,50) -1\*ones(1,50)];

for songChunk=1:minrow,

progress = songChunk./minrow.\*100

signals(1,:) = sig1(songChunk,:);

signals(2,:) = sig2(songChunk,:);

[num\_sig,num\_elts] = size(signals);

%step 2: Quantize signals in to bits.

disp('Quantization......')

for quantIndex=1:num\_sig,

quant\_completed = quantIndex./num\_sig.\*100;

signals\_quant(quantIndex,:) =

quantize(signals(quantIndex,:));

end;

%step 3:Encode the signals by modulating them with Hadamard

%codeset

disp('Encoding...');

[code sig\_mod] = encoder(signals\_quant);

[rows\_sig\_mod,cols\_sig\_mod]=size(sig\_mod);

%Step 4: PASS MODULATED SIGNALS THROUGH COMMUNICATION CHANNEL.

disp('Channel...');

[channel\_out,noise] = channel2(sig\_mod,'varying');

%For now,attempt to recover every signal.

disp('Recovery...')

for recIndex=1:num\_sig,

%Step 5:Apply adaptive filter

y\_mod\_hat(recIndex,:) =

wiener(channel\_out(101:(100+cols\_sig\_mod)),

noise(101:(100+cols\_sig\_mod)),sig\_mod,recIndex);

%y\_mod\_hat2(recIndex.:) =

% adapfilt(preamble,channel\_out);

%Step 6:Apply threshold

y\_recover(recIndex,:) = decoder(recIndex,code,

y\_mod\_hat(recIndex,:));

%y\_recover(recIndex,:)= decoder(recIndex,code,y\_mod\_hat2(recIndex,:));

end;

%Step 7:Dequantize bits

for dequantIndex=1:num\_sig,

dequant\_completed = dequantIndex./num\_sig.\*100

sig\_recover(dequantIndex,:)=

dequantize(y\_recover(dequantIndex,:));

end;

FS

%Some useful plots.

for sigIndex=1:num\_sig,

figure(1)

subplot(1,num\_sig,sigIndex),

plot(signals(sigIndex,:)),title(sigIndex);grid;

figure(2)

subplot(1,num\_sig,sigIndex),

plot(sig\_recover(sigIndex,:)),grid;title(sigIndex);

figure(3)

error(sigIndex,:) = sig\_recover(sigIndex,:)-

signals(sigIndex,:);

subplot(1,num\_sig,sigIndex),

plot(error(sigIndex,:)),grid;title(sigIndex);

end;

sig1\_recover(songChunk,:) = sig\_recover(1,:);

sig2\_recover(songChunk,:) = sig\_recover(2,:);

end;

%Step 8: Concatenate song chunks in to a vector.

sig1\_rec\_vec = vec2wav(sig1\_recover,num\_zeros);

sig2\_rec\_vec = vec2wav(sig2\_recover,num\_zeros);

figure(101)

subplot(1,2,1),plot(sig1\_rec\_vec),title('Laser ');

subplot(1,2,2),plot(sig2\_rec\_vec),title('Down');

wavwrite(sig1\_rec\_vec,8000,8, 'rlaser');

wavwrite(sig2\_rec\_vec,8000,8, 'rdown');

figure(102)

subplot(1,2,1),plot(sig1\_rec\_vec\_

song1(1:length(sig1\_rec\_vec))'),

title('Laser Error');

subplot(1,2,2),plot(sig2\_rec\_vec\_

song2(1:length(sig2\_rec\_vec))'),title('Down Error');

14) %C H A N N E L

function[channel\_out,noise]= channel(sig\_mod,mean,var)

%CHANNEL Transmits modulated signals over a communication channel

%with Additive White Gaussian Noise (AWGN).

%CHANNEL(sig\_mod) takes in a matrix of modulated signals and

%generates an output matrix of the same dimensions after passing

%through the channel,assuming noise is Gaussian with mean=0 and

%variance=0.01.Returns both output matrix and noise.

%CHANNEL(sig\_mod,mean,var)genetares an output matrix assuming

%Gaussian noise with specified mean and variance.

%Returns both

%outputmatrix and noise.

if nargin == 1

mean = 0;

var = 0.01;

end

[rowsSig\_Mod,colsSig\_Mod] = size(sig\_mod);

%Interference added to transmitted signal

%modeled as random noise sequence (mean 0,var 0.01)

noise = random('norm',mean,var,1,colsSig\_Mod);

channel\_out = noise;

for nIndex=1:rowsSig\_Mod,

channel\_out = channel\_out+sig\_mod(nIndex,:);

end;

14) % D E C O D E R

function[y\_recover] = decoder(recIndex,code,channel\_out)

%DECODER Decodes a set of siganls modulated using an ortogonal

%(Hadamard) code set .Each receiver must know the codeset

%of the sender.The signal of a particular sender is then

%determined based on the threshold of the conditional

%probability density function of each bit.since BPSK is used

%and each signal occurs with equal probality 1/2,

%threshold=0;

%DECODER(recIndex,code,channel\_out) decodes the signal assuming

%the recIndex signal is to be recovered.Takes in y\_mod\_hat,the original

%signal modulated by a code set and possibly filtered ,and

%genetares a vector y\_recover thet is the recovery of the

% original signal

%Receiver knows codeset of sender.

%To decode ,multiply outout of adaptive filter bits by codeset

%Then,recover data.

[rowsCode,colsCode]=size(code);

index = 1;

j=1;

for i =1:length(channel\_out),

y\_unmod\_hat(recIndex,i)=

channel\_out(i).\*code(recIndex,j);

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

j=1;%reset back to beginning of code set

y\_hat(recIndex,index)=

(sum(y\_unmod\_hat(recIndex,:)))./colsCode;%perform decode of bit

y\_unmod\_hat(recIndex,:) = 0;%reset to 0 for next bit

index = index+1;

else

j=mod(i,colsCode)+1;

end

end;

%y\_hat

%Determine signal sent based on position relative to threshold

%Because of binary signals,threshold at 0.

for i=1:length(y\_hat(recIndex,:)),

if (y\_hat(recIndex,i) >= 0)

y\_recover(recIndex,i) =1;

elseif (y\_hat(recIndex,i) < 0)

y\_recover(recIndex,i)= -1;

end

end

y\_recover = y\_recover(recIndex,:);

15) %D E Q U A N T I Z A T I O N

function[decimalvect]=dequantize(x)

%converts signal with "binary" values (1=1 -1=0) to a signal with

%quantized decimal values ranging between -1 and 1

%b=number of bits used to represent each value

b=5;

min=-1;

max=1;

range=max-min;

delta=range/(2^b-1);

x\_bin=(x+1)/2;

intvect\_intermed=[0];

for i=1:(length(x)/b)

intval=0;

for j=1:b

intval=intval+x\_bin((i-1)\*b+j)\*2^(b-j);

end

intvect\_intermed=[intvect\_intermed intval];

end

intvect=zeros(length(intvect\_intermed)-1);

intvect=intvect\_intermed(2:length(intvect\_intermed));

intvect\_off=intvect-(2^(b-1)-1);

decimalvect=intvect\_off\*delta;

16) % E N C O D E R

function [y\_recover,sig\_mod]= encoder2(y)

% ENCODER implements the spread spectrum Technique and

%modulates the input signal with an orthogonal code set.

if nargin<1

error('pleaase provide input signals. ');

end;

[num\_sig,num\_elts]=size(y);

sprintf('Number of signals: %d',num\_sig)

%Generate Pseudo\_Random Noise Code (PNCode) which is our chip set

%A practical code is the Walsh Hadamard code.

%Walsh Hadamard codes are orthogonal and gets rid of any

%multi\_access interference.

code = hadamard(num\_sig);

[rowsY,colsY]=size(y)

[rowsCode,colsCode]=size(code)

%Store modulated signals in matrix sig\_mod(initialized to 0);

%sig\_mod(num\_sig,)=0;

k=1;

for i=1:colsY,

for j=1:colsCode,

%Assign values of sig\_mod matrix

for matIndex=1:rowsY,

sig\_mod(matIndex,k)=Y(matIndex,i).\*code(matIndex,j);

end;

k=k+1;

end;

end;

%%%%%%%%%%%%CHANNEL.M%%%%%%%%%%%%%%%%%%%%%

%[rowsSig\_Mod,colsSig\_Mod]=size(sig\_mod);

%Interference added to transmitted signal

%modelled as random noise sequence (mean 0,var 0.01)

%noise=random('norm', 0,0.01,1,colsSig\_Mod);

%channel\_out=noise;

%for nIndex=1:num\_sig,

%channel\_out=channel\_out+sig\_mod(nIndex,:);

%end;

%channel\_out

%%%%%%%%%%%%%%%%END CHANNEL.M%%%%%%%%%%%%%%%%%%

%subplot(2,2,4),bar(channel\_out);title('channel output:

%sum(y\_x)+noise');grid;

%%%%%%%%%%%RECEIVER IMPLEMENTATION

for recIndex=1:num\_sig,

%Adaptive Wiener Filter (similar implementation to hw14)

%H(w)=S\_ss/(S\_ss(w) + S\_noise(w));

%S\_ss(w)=R\_ss(0)+mean^2 =variance

S\_ss(recIndex) =

var(sig\_mod(recIndex,:))+(mean(sig\_mod(recIndex,:))).^2;

%S\_ss1\_old=var(y1\_mod)+(mean(y1\_mod)).^2

totalNoise=noise;

for noiseIndex=1:num\_sig,

if (mod(recIndex,noiseIndex) ~=0)

totalNoise=totalNoise+sig\_mod(nioiseIndex,:);

end

end

S\_noise=var(totalNoise)+(mean(totalNoise)).^2;

%S\_noise\_old=

% var(noise+y2\_mod+y3\_mod+y4\_mod)+(mean(noise+y2\_mod+y3\_mod+y4\_mod)).^2

%Step 1:Compute FFT of channel\_out[n],h[n],length 16

N=16;

%X[k]

channel\_out\_padded=zeros(1,N);%zero-pad x[n] to length 2N-1;

channel\_out\_padded(1:length(channel\_out))=channel\_out;

channel\_out\_fft=fft(channel\_out\_padded);

%H[k]

%H(recIndex)=zeros(num\_sig,N);%Reset H to length 512

H(recIndex)=S\_ss(recIndex)./(S\_ss(recIndex)+S\_noise);

%S\_ss = zeros(1,N);%Reset S\_ss to length 512

%for k=0:N-1,

%W=2\*pi\*k./N;

%S\_ss(k+1)=var(y1);

%H(k+1) = S\_ss(k+1)./(S\_ss(k+1)+0.5);

%optimal weiner filter derived above

%end

%figure(2)

%subplot(3,1,1),plot(abs(H1)),grid,title('H[k]');

%subplot(3,1,2),plot(s),grid,title('Original s[n]');

%step 2:Multiply

%HTemp = H(recIndex);

S\_twidle\_fft(recIndex,:) = H(recIndex).\*channel\_out\_fft;

%pause;

%step 3:IFFT

s\_twidle(recIndex,:) =

real(ifft(S\_twidle\_\_\_fft(recIndex,:)));

s\_twidleTemp = s\_twidle(recIndex,:);

s\_hat(recIndex,:) =s\_twidleTemp(1:colsSig\_Mod);

y\_mod\_hat(recIndex,:)= s\_hat(recIndex,:);

%Different terminology

%subplot(3,1,2),bar(s\_hat1),grid,

title('y1\\_mod\\_hat[n],where w~N(0,0.5');

%Receiver knows codeset of sender.

%To decode,multiply output of adaptive filter bits by code set.

%Then, recover data.

index = 1;

j = 1;

for i=1:length(y\_mod\_hat(recIndex,:)),

%i

%j

y\_unmod\_hat(recIndex,i) =

channel\_out(i).\*code(recIndex,j);

%y\_unmod\_hat(recIndex,i)=

% channel\_out(i).\*code1(recIndex,j)

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

j =1;%reset back to beginning of code set

y\_hat(recIndex,index) =

(sum(y\_unmod\_hat(recIndex,:)))./cols Code;%perform decode of bit

y\_unmod\_hat(recIndex,:) = 0;%reset to 0 for next bit

% y1\_hat(index)=

(sum(y1\_unmod\_hat))./length(code1)%perform decode of bit

% y1\_unmod\_hat = 0;% reset to 0 for next bit

index = index+1;

else

j = mod(i,colsCode)+1;

end

% y1\_unmod\_hat(i)= y1\_mod\_hat(i).\*code1(i)

end;

%Recover data

%for i=1:length(y1\_unmod\_hat),

%y1\_hat(i)= sum(y1\_unmod\_hat(i))./length(code1)

%end;

%subplot(3,1,3),bar(y1\_hat),grid,title('y1\\_hat[n]');

%Determine signal sent based on position relative to threshold

%Bcause of BPSK,threshold at 0.

for i = 1:length(y\_hat(recIndex,:)),

if (y\_hat(recIndex,i) >= 0)

y\_recover(recIndex,i) =1;

elseif(y\_hat(recIndex,i)< 0 )

y\_recover(recIndex,i) = -1;

end

end

%figure(3)

%bar(y1\_recover),grid,title('y1\\_recovered[n]');

disp('y\_recover:')

y\_recover(recIndex,:)

error = Y\_recover(recIndex,:)-y(recIndex,:)

end;

17) %QUANTIZATION

function[binaryvect]quantize(x)

%converts signal with decimal values rangeing from -1 and 1 toa "binary"

%output (1=1,-1=0)

%b=number of bits used to represent each value

b=5;

min=-1;

max=1;

range=max-min;

delta=range/(2^b-1);

%quantize decimal values

x\_quant=quant(x,delta);

%convert decimal values to integers ranging from 0 to 2^b-1

x\_int=x\_quant/delta;

%offset integers to have range from 0 to 2^b-1

x\_int\_off=x\_int+2^(b-1);

%convert integers to binary where 1=1,-1=0;

int\_off=x\_int\_off;

binaryvect\_intermed=[0];

for i=1:length(int\_off)

convert\_compleated=i./length(int\_off)\*100;

for j=b:-1:1

if(int\_off(i)>=2^(j-1);

int\_off(i)=int\_off(i)-2^(j-1);

binaryvect\_intermed=[binaryvect\_intermed 1];

else

binaryvect\_intermed=[binaryvect\_intermed -1];

end

end

end

binaryvect=zeros(length(binaryvect\_intermed) -1);

binaryvect=binaryvect\_intermed(2:length(binaryvect\_intermed));

19) function [songoutBit, num\_zeros] = wav2vec(songout)

%[songout, Fs, nbits] = wavread('Kokomo.wav');

songout = songout';

%soundsc(songout)

%TESTING!!

%sig1(1,1:128) = [ones(1,64), zeros(1,64)];

%sig1(2,1:128) = [zeros(1,64), ones(1, 64)];

%%signal1 = [ones(1,300), zeros(1,300)];

%signal1 = [sig1(1,:) sig1(2,:)];

%%figure(100), subplot(1,2,1), plot(signal1), title('signal1');

%%songout = signal1;

index=1;

N = 1000;

%zeropad the song so that it is a multiple of N

remain = mod(length(songout),N);

num\_zeros = N-remain;

%songout2 = zeros(length(songout)+num\_zeros);

%songout2(1,1:length(songout))= songout(1,:);

%songout = songout2;

for songIndex=1:N:length(songout), %length(songout),

if ((length(songout) - songIndex) < (N-1))

songoutBit(index,1:remain) = songout(1, songIndex:length(songout));

songoutBit(index,remain+1:N) = zeros(1,num\_zeros);

else

songoutBit(index,1:N) = songout(1, songIndex:songIndex+N-1);

end

index = index+1;

end;

20) function [songoutBit, num\_zeros] = wav2vec(songout)

%[songout, Fs, nbits] = wavread('Kokomo.wav');

songout = songout';

%soundsc(songout)

%TESTING!!

%sig1(1,1:128) = [ones(1,64), zeros(1,64)];

%sig1(2,1:128) = [zeros(1,64), ones(1, 64)];

%%signal1 = [ones(1,300), zeros(1,300)];

%signal1 = [sig1(1,:) sig1(2,:)];

%%figure(100), subplot(1,2,1), plot(signal1), title('signal1');

%%songout = signal1;

index=1;

N = 1000;

%zeropad the song so that it is a multiple of N

remain = mod(length(songout),N);

num\_zeros = N-remain;

%songout2 = zeros(length(songout)+num\_zeros);

%songout2(1,1:length(songout))= songout(1,:);

%songout = songout2;

for songIndex=1:N:length(songout), %length(songout),

if ((length(songout) - songIndex) < (N-1))

songoutBit(index,1:remain) = songout(1, songIndex:length(songout));

songoutBit(index,remain+1:N) = zeros(1,num\_zeros);

else

songoutBit(index,1:N) = songout(1, songIndex:songIndex+N-1);

end

index = index+1;

end;

21) % W I E N E R

function [y\_mod\_hat] = wiener(channel\_out,noise,sig\_mod,recIndex)

%WIENER Implements a wiener filter on the given channel output

%signal.Assumes partial knowledge of signal and noise

%characteristics namely their autocorrelation functons.

%WIENER(channel\_out,noise,sig\_mod,recIndex)takes the output

%of the channel and the noise and creates an adaptive filter

% TODO: fix code so that takes in PSD of signal and noise.

if nargin < 4

error('Please provede all required input signals.');

end

%Adaptive Wiener Filter (similar implementation to hw14)

%H(w) = S\_ss/ (S\_ss(w) + S\_noise(w));

%S\_ss(w) = R\_ss(0)+ mean^2 = variance + mean.^2

[num\_sig,num\_elts]= size(sig\_mod);

S\_ss(recIndex) =

var(sig\_mod(recIndex,:))+(mean(sig\_mod(recIndex,:))).^2;

total noise = noise;

for noiseIndex=1:num\_sig,

if (mod(recIndex,noiseIndex) ~= 0)

totalNoise = totalNoise = sig\_mod(noiseIndex,:);

end

end

S\_noise = var(totalNoise)+ (mean(totalNoise)).^2;

%step 1: compute FFT of channel\_out[n],h[n],length 16

%N = 16;

%X[k]

%channel\_out\_padded = zeros(1,N);%zero-pad x[n] to length 2N-1;

channel\_out\_padded(1:length(channel\_out)) = channel\_out;

channel\_out\_fft = fft(channel\_out\_padded);

%H[k]

H(recIndex) = S\_ss(recIndex)./(S\_ss(recIndex) + S\_noise);

%Step 2:Multiply

S\_twidle\_fft(recIndex,:) = H(recIndex).\*channel\_out\_fft;

%Step 3: IFFT

S\_twidle\_(recIndex,:) =

real(ifft(S\_twidle\_fft(recIndex,:)));

S\_twidleTemp = S\_twidle(recIndex,:);

s\_hat(recIndex,:)= S\_twidleTemp(1:num\_elts);

%y\_mod\_hat(recIndex,:) = S\_hat(recIndex,:)%Different terminology

y\_mod\_hat = S\_hat(recIndex,:);%Different terminology

22) %MATLAB PROGRAM TO DISPLAYY ALL BASIC DISCRETE SIGNALS

%impulse function

N=input('enter the number of array values');

N=6;

n=1:N;w=2\*pi\*0.125;

xn=[zeros(1,N),ones(1,1),zeros(1,N)]

subplot(9,1,1);plot(xn);

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

x1n=[ones(1,N)];

subplot(9,1,2);plot(x1n);

xlabel('time');ylabel('magnitude');

title('unit step sequence');

x2n=n.\*x1n;

subplot(9,1,3);plot(x2n);

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

x3(n)=exp(j\*w\*n);

x4(n)=exp(-j\*w\*n);

x5(n)=(x3(n)+x4(n))./2;x6(n)=(x3(n)-x4(n))./(2\*j);

x7(n)=sin(w\*n);

x8(n)=cos(w\*n);

subplot(9,1,4);plot(x3);

xlabel('time');ylabel('magnitude');

title('exponentially increasing signal');

subplot(9,1,5);plot(x4);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(9,1,6);plot(x5);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal using Euler theorem');

subplot(9,1,7);plot(x6);

xlabel('time');ylabel('magnitude');

title('sinusoidal signalusing Eulers theorem');

subplot(9,1,8);plot(x7);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

subplot(9,1,9);plot(x8);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

23) %matlab program to print all discrete time signals in the range -6 to 6

N=input('enter the number of array values');

N=6;

n=-6:1:6;

xn=[zeros(1,N),ones(1,1),zeros(1,N)]

subplot(7,1,1);plot(xn);

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

n=-6:1:6

x1n=[zeros(1,6),ones(1,6)]

subplot(7,1,2);plot(x1n);

xlabel('time');ylabel('magnitude');

title('unit step sequence');

n=-6:1:5

x2n=n.\*x1n

subplot(7,1,3);plot(x2n);

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

w=2\*pi\*600

n=-6:1:6

x3n=exp(j\*w\*n);

n=-6:1:6

x4n=exp(-j\*w\*n)

n=-6:1:6

x5n=(x3n+x4n)./2

n=-6:1:6

x6n=(x3n-x4n)./(2\*j)

n=-6:1:6

x7n=sin(w\*n)

n=-6:1:6

x8n=cos(w\*n)

subplot(7,1,4);plot(x3n);

xlabel('time');ylabel('magnitude');

title('exponentially increasing signal');

subplot(7,1,5);plot(x4n);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(7,1,6);plot(x5n);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

subplot(7,1,7);plot(x6n);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

24) %matlab program to print all discrete time signals in the range -6 to 6

N=input('enter the number of array values');

N=6;

n=1:N;

xn=[zeros(1,N),ones(1,1),zeros(1,N)]

subplot(7,1,1);plot(xn);

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

n=1:6

x1n=[ones(1,N)]

subplot(7,1,2);plot(x1n);

xlabel('time');ylabel('magnitude');

title('unit step sequence');

n=1:6

x2n=n.\*x1n

subplot(7,1,3);plot(x2n);

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

w=2\*pi\*600

n=1:6

x3n=exp(j\*w\*n)

n=1:6

x4n=exp(-j\*w\*n)

n=1:6

x5n=(x3n+x4n)./2

x9n=x4n.\*x1n

n=1:6

x6n=(x3n-x4n)./(2\*j)

n=1:6

x7n=sin(w\*n)

n=1:6

x8n=cos(w\*n)

subplot(7,1,4);plot(x3n);

xlabel('time');ylabel('magnitude');

title('exponentially increasing signal');

subplot(7,1,5);plot(x4n);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(7,1,6);plot(x5n);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

subplot(7,1,7);plot(x6n);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

25) %matlab program to print all discrete time signals in the range -6 to 6

N=input('enter the number of array values');

N=6;

n=1:N;

xn=[zeros(1,N),ones(1,1),zeros(1,N)]

subplot(7,1,1);plot(xn);

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

n=1:6

x1n=[ones(1,N)]

subplot(7,1,2);plot(x1n);

xlabel('time');ylabel('magnitude');

title('unit step sequence');

n=1:6

x2n=n.\*x1n

subplot(7,1,3);plot(x2n);

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

w=2\*pi\*600

n=1:6

x3n=exp(j\*w\*n)

n=1:6

x4n=exp(-j\*w\*n)

n=1:6

x5n=(x3n+x4n)./2

x9n=x4n.\*x1n

n=1:6

x6n=(x3n-x4n)./(2\*j)

n=1:6

x7n=sin(w\*n)

n=1:6

x8n=cos(w\*n)

subplot(7,1,4);plot(x3n);

xlabel('time');ylabel('magnitude');

title('exponentially increasing signal');

subplot(7,1,5);plot(x4n);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(7,1,6);plot(x5n);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

subplot(7,1,7);plot(x6n);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

26) %MATLAB PROGRAM TO DISPLAYY ALL BASIC DISCRETE SIGNALS

%impulse function

N=input('enter the number of array values');

N=16;

n=1:N;

xn=[zeros(1,N),ones(1,1),zeros(1,N)]

subplot(8,1,1);stem(xn);

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

x1n=[ones(1,N)];

subplot(8,1,2);stem(x1n);

xlabel('time');ylabel('magnitude');

title('unit step sequence');

x2(n)=n.\*x1n;

subplot(8,1,3);stem(x2);

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

x3(n)=exp(j\*n);

x4(n)=exp(-j\*n);

x5(n)=(x3n+x4n)./2;x6(n)=(x3n-x4n)./(2\*j);

a=2;

x7(n)=a.^n.\*x1n;

subplot(8,1,4);stem(x3);

xlabel('time');ylabel('magnitude');

title('exponentially increasing signal');

subplot(8,1,5);stem(x4);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(8,1,6);stem(x5);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

subplot(8,1,7);stem(x6);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

subplot(8,1,8);stem(x7);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

27) %MATLAB PROGRAM TO DISPLAYY ALL BASIC DISCRETE SIGNALS

%impulse function

N=input('enter the number of array values');

N=15;

n=1:N;

xn=[ones(1,1),zeros(1,N)];

subplot(9,1,1);stem(xn);

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

x1n=[ones(1,N)];

subplot(9,1,2);stem(x1n);

xlabel('time');ylabel('magnitude');

title('unit step sequence');

x2(n)=n.\*x1n;

subplot(9,1,3);stem(x2);

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

n=1:N;

x3(n)=exp(j\*n);

x4(n)=exp(-j\*n);

x5(n)=(x3(n)+x4(n))./2;x6(n)=(x3(n)-x4(n))./(2\*j);

a=2;

x7(n)=a.^n.\*x1n;

x8(n)=sin(2\*pi\*100\*n);

x9(n)=cos(2\*pi\*100\*n);

subplot(9,1,4);stem(x4);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(9,1,5);stem(x5);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

subplot(9,1,6);stem(x6);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

subplot(9,1,7);stem(x7);

xlabel('time');ylabel('magnitude');

title('Real exponential signal');

subplot(9,1,8);stem(x8);

xlabel('time');ylabel('magnitude');

title('sinusoidalsignal');

subplot(9,1,9);stem(x9)

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

28) %MATLAB PROGRAM TO DISPLAYY ALL BASIC DISCRETE SIGNALS

%impulse function

N=input('enter the number of array values');

N=15;

n=1:N;

xn=[ones(1,1),zeros(1,N)];

subplot(10,1,1);stem(xn);

x1n=[ones(1,N)];

subplot(10,1,2);stem(x1n);

x2(n)=n.\*x1n;

subplot(10,1,3);stem(x2);

n=1:N;

x9(n)=exp(n).\*x1n;

x10(n)=exp(-n).\*x1n;

x3(n)=(1/2).^n.\*x1n;

x4(n)=sin(2\*pi\*600\*n);

x5(n)=cos(2\*pi\*600\*n);

x6(n)=(1/3).^n.\*x1n;

x7(n)=exp(j\*((n\*(pi./2))+(pi)./(4)));

x8(n)=sin(n\*(pi./4));

subplot(10,1,4);stem(x4);

subplot(10,1,5);stem(x5);

subplot(10,1,6);stem(x6);

subplot(10,1,7);stem(x7);

subplot(10,1,8);stem(x8);

subplot(10,1,9);stem(x9)

subplot(10,1,10);stem(x10);

29)

%MATLAB PROGRAM TO DISPLAYY ALL BASIC DISCRETE SIGNALS

%impulse function

N=input('enter the number of array values');

N=6;

n=1:N;

x(n)=exp(j\*n\*(pi./6));

x1(n)=exp(3/5\*(n+0.5));

x2(n)=cos(n\*2\*(pi./3));

x3(n)=cos(n\*(pi./3))+cos(3\*(pi./4)\*n);

subplot(4,1,1);stem(x);

subplot(4,1,2);stem(x1);

subplot(4,1,3);stem(x2);

subplot(4,1,4);stem(x3);

30) %MATLAB PROGRAM TO DISPLAY DISCRETE SIGNALS depicted in the problem 1.3

N=input('enter the number of array values');

N=6;

n=1:N;

u(n)=1;

x(n)=(n+2).\*cos(((2\*n)+1)\*(pi./3));

x1(n)=((1/2).^n.\*u(n))+(2).^n.\* u(n);

x2(n)=cos(2\*(pi./3)\*n)+sin((n+1)\*(pi./4));

x3(n)=sin((pi./3)\*n)+2\*cos(3\*(pi./4)\*n)

subplot(4,1,1);stem(x);

subplot(4,1,2);stem(x1);

subplot(4,1,3);stem(x2);

subplot(4,1,4);stem(x3);

31) %MATLAB PROGRAM TO DISPLAY DISCRETE SIGNALS depicted in the problem 1.4

N=input('enter the number of array values');

N=6;

n=1:N;

x(n)=exp(j\*2\*(pi./3)\*n)+exp(j\*(3\*(pi./4)\*n));

x1(n)=(n./2);

x2(n)=n.\*((1/2).^n);

x31(n)=cos(n);

x32(n)=n.\*((n+1)./2);

x3(n)=x31(n).\*x32(n);

subplot(4,1,1);stem(x);

subplot(4,1,2);stem(x1);

subplot(4,1,3);stem(x2);

subplot(4,1,4);stem(x3);

32) function m=bpsk111(bit)

bit=input('enter the 8 bit vector');

x=(1:100)/200;

pos=cos(2\*pi\*x);

neg=-pos;

m=bit;

for i=1:length(bit)

if bit(i)==1

m(1+(i-1)\*length(x):i\*length(x))=pos;

end

if bit(i)==0

m(1+(i-1)\*length(x):i\*length(x))=-neg;

end

end

subplot(2,1,1),stairs(1:length(bit),bit),axis([0,length(bit),-2,2]);

title('bit stream');

subplot(2,1,2),plot(m);

title('BPSK modulation')

33) t=0:0.001:2; % 2 secs @ 1kHz sample rate

y=chirp(t,0,1,150); % Start @ DC, cross 150Hz at t=1sec

plot(y)

%specgram(y,256,1E3,256,250); % Display the spectrogram

34) %CDMA Transmits signals using CDMA.

% CDMA(signals) Given a matrix of signals (sampled in the time

% domain), will use CDMA to transmit signals through

% communication channel. Uses adaptive filtering and

% thresholding in conjunction with CDMA to recover signals.

%

%if nargin<1

% error('Please provide input signals. ');

%end;

clear

%Step 1: Chunk song into N length pieces (N=100)

%Sig1 is the entire signal of the first song (user)

disp('Wav2Vec')

song1 = wavread('Ding.wav');

song2 = wavread('Chord.wav');

figure(1)

subplot(1,2,1), plot(song1); title('Original Laser');

subplot(1,2,2), plot(song2); title('Original Down Waves');

[sig1, num\_zeros] = wav2vec(song1);

[sig2, num\_zeros] = wav2vec(song2);

[rowSig1, colSig1] = size(sig1);

[rowSig2, colSig2] = size(sig2);

minrow = min(rowSig1, rowSig2);

%% TESTING!!!

%num\_zeros = 0;

%sig1(1,1:128) = [ones(1,64), zeros(1,64)];

%sig1(2,1:128) = [zeros(1,64), ones(1, 64)];

%signal1 = [sig1(1,:) sig1(2,:)];

%figure(300), plot(signal1), title('signal1');

%[rowSig1, colSig1] = size(sig1);

%Set first row of sig1 to the first row of the sender signal

preamble=[ones(1,50) -1\*ones(1,50)];

for songChunk=1:minrow,

progress = songChunk./minrow.\*100

signals(1,:) = sig1(songChunk,:);

signals(2,:) = sig2(songChunk,:);

[num\_sig, num\_elts] = size(signals);

%Step 2: Quantize signals into bits.

disp('Quantization...')

for quantIndex=1:num\_sig,

quant\_completed = quantIndex./num\_sig.\*100

signals\_quant(quantIndex,:) = quantize(signals(quantIndex,:));

end;

%Step 3: Encode the signals by modulating them with Hadamard

% codeset.

disp('Encoding...')

[code sig\_mod] = encoder(signals\_quant);

[rows\_sig\_mod,cols\_sig\_mod]=size(sig\_mod);

%Step 4: Pass modulated signals through communication channel.

disp('Channel...')

[channel\_out, noise] = channel2(sig\_mod,'varying');

%For now, attempt to recover every signal.

disp('Recovery...')

for recIndex=1:num\_sig,

%Step 5: Apply adaptive filter

y\_mod\_hat(recIndex,:) = wiener(channel\_out(101:(100+cols\_sig\_mod)), noise(101:(100+cols\_sig\_mod)), sig\_mod,recIndex);

%y\_mod\_hat2(recIndex,:) = adapfilt(preamble,channel\_out);

%Step 6: Apply threshold

y\_recover(recIndex,:) = decoder(recIndex, code, y\_mod\_hat(recIndex,:));

%y\_recover(recIndex,:) = decoder(recIndex, code, y\_mod\_hat2(recIndex,:));

end;

%Step 7: Dequantize bits.

for dequantIndex=1:num\_sig,

dequant\_completed = dequantIndex./num\_sig.\*100

sig\_recover(dequantIndex,:) = dequantize(y\_recover(dequantIndex,:));

end;

%Some useful plots.

for sigIndex=1:num\_sig,

figure(1)

subplot(1, num\_sig, sigIndex),

plot(signals(sigIndex,:)),title(sigIndex); grid;

figure(2)

subplot(1, num\_sig, sigIndex),

plot(sig\_recover(sigIndex,:)), grid; title(sigIndex);

figure(3)

error(sigIndex,:) = sig\_recover(sigIndex,:) - signals(sigIndex,:);

subplot(1, num\_sig, sigIndex),

plot(error(sigIndex,:)), grid; title(sigIndex);

end;

sig1\_recover(songChunk,:) = sig\_recover(1,:);

sig2\_recover(songChunk,:) = sig\_recover(2,:);

end;

%Step 8: Concatenate song chunks into a vector.

sig1\_rec\_vec = vec2wav(sig1\_recover, num\_zeros);

sig2\_rec\_vec = vec2wav(sig2\_recover, num\_zeros);

figure(101)

subplot(1,2,1), plot(sig1\_rec\_vec), title('Laser ');

subplot(1,2,2), plot(sig2\_rec\_vec), title('Down');

wavwrite(sig1\_rec\_vec, 8000, 8, 'rlaser');

wavwrite(sig2\_rec\_vec, 8000, 8, 'rdown');

figure(102)

subplot(1,2,1), plot(sig1\_rec\_vec-song1), title('Laser Error');

subplot(1,2,2), plot(sig2\_rec\_vec-song2), title('Down Error');

35) % generation of pn sequence for 2 users

i=input('enter the no of shift registers');

for i=1:2

a(i)=input('enter the array values for a(i)');

end

for i=1:4

y=xor(a(2),a(2));

a(2)=a(1);

a(1)=y;

pn(i)=[a(2)];

end

36) function [channel\_out, noise] = channel(sig\_mod, mean, var)

%CHANNEL Transmits modulated signals over a communication channel

% with additive white Gaussian noise (AWGN).

% CHANNEL(sig\_mod) takes in a matrix of modulated signals and

% generates an output matrix of the same dimensions after passing

% through the channel, assuming noise is Gaussian with mean=0 and

% variance=0.01. Returns both output matrix and noise.

%

% CHANNEL(sig\_mod, mean, var) generates an output matrix assuming

% Gaussian noise with specified mean and variance. Returns both

% output matrix and noise.

%

if nargin == 1

mean = 0;

var = 0.01;

end

[rowsSig\_Mod, colsSig\_Mod] = size(sig\_mod);

%Interference added to transmitted signal

% modeled as random noise sequence (mean 0, var 0.01)

noise = random('norm', mean, var, 1, colsSig\_Mod);

channel\_out = noise;

for nIndex=1:rowsSig\_Mod,

channel\_out = channel\_out+sig\_mod(nIndex,:);

end;

37) function [channel\_out, noise] = channel2(sig\_mod, type)

%need to change comments on this

%CHANNEL Transmits modulated signals over a communication channel

% with additive white Gaussian noise (AWGN).

% CHANNEL(sig\_mod) takes in a matrix of modulated signals and

% generates an output matrix of the same dimensions after passing

% through the channel, assuming noise is Gaussian with mean=0 and

% variance=0.01. Returns both output matrix and noise.

%

% CHANNEL(sig\_mod, mean, var) generates an output matrix assuming

% Gaussian noise with specified mean and variance. Returns both

% output matrix and noise.

%

[rowsSig\_Mod, colsSig\_Mod] = size(sig\_mod);

channel\_out=zeros(1,colsSig\_Mod);

for nIndex=1:rowsSig\_Mod,

channel\_out = channel\_out+sig\_mod(nIndex,:);

end;

preamble=[ones(1,50) -1\*ones(1,50)];

channel\_out=[preamble channel\_out];

mean=0;

var=.01;

if(strcmp(type,'fixed'))

%Interference added to transmitted signal

% modeled as random noise sequence (mean 0, var 1)

noise = random('norm', mean, var, 1, (colsSig\_Mod+100));

%%%%channel\_out = noise;

%%%%for nIndex=1:rowsSig\_Mod,

%%%% channel\_out = channel\_out+sig\_mod(nIndex,:);

%%%%end;

channel\_out=channel\_out+noise;

end

if(strcmp(type,'varying'))

noise=random('norm', mean, var/2, 1, (colsSig\_Mod+100));

for(i=1:(colsSig\_Mod+100))

noise(i)=.005\*sin(i)+noise(i);

end

for(i=1:(colsSig\_Mod+100))

channel\_out(i)=channel\_out(i)+noise(i);

end

end

38) % This function is used for decoding the source code %i.e,

%converting the binary data into the original decimal form

function channelencoding1(x);

[rows,colos]=size(x);

fid=fopen('binary.value','r');

fread(fid);

s=setstr(ans');

l=size(s);

%isp ' size of the mat is'

fclose(fid);

size(l);

m=1;

lm=[ ];

pp=[];

while m<=colos

kk=s(m);

lm=[lm bin2dec(kk)];

m=m+1;

end

q=(rows\*colos)

while m<q

pp=[];

oo=m+colos;

while m<oo

kk=s(m);

pp=[pp bin2dec(kk)];

m=m+1;

end

lm=[lm;pp];

end

[rows colos]=size(lm)

a=colos/16;

b=floor(a);

if a>b

b=b+1;

newcolos=b\*16;

addcolos=newcolos-colos;

p=ones(rows,addcolos);

lm=[lm p];

end

disp ' after modifying colos'

[rows newcolos]=size(lm)

a=rows/16;

b=floor(a);

if a>b

b=b+1;

newrows=b\*16;

addrows=newrows-rows;

q=ones(addrows,newcolos);

lm=[lm;q];

end

disp ' after modifying rows'

size(lm)

i=1;

while i<rows

n1=code(i,lm);

im=[im;n1];

i=i+16;

end

imshow(im);

figure(1);

% Function for Dividingthe pixels into blocks

function [n]=code(i,lm)

j=i+15;

[rows colos]=size(lm);

p=1;

n=[];

while p<colos

q=p+15;

s11=lm(i:j,p:q);

ls11=linearize(s11);

mes11=rscoding(ls11);

rmes11=rsdecoding(mes11);

n=[n rmes11];

p=p+16;

end

%RS- CODING Function

function [code11]=rscoding(s11);

N=511;

K=257;

M=9;

ls11=linearize(s11);

s111=zeros(1,1:257);

s111(2:257)=ls11;

code11=rsenco(s111,N,K);

%RS-DECODING Function

function [mes1]=rsdecoding(code11);

N=511;

K=257;

M=9;

mes11=rsdeco(code11',N,K);

mes21=mes11(2:257,1);

mes1=vec2mat(mes21',16);

size(mes1);

%Function for linearizing i.e, converting 2-D into 1-D matrix

function i = linearize(j);

%linearize takes a 2-D matrix and turns it into a 1-D matrix

i = [ ];

len = length(j(:,1));

for r = 1:len

i = [i j(r,:)];

end

size(i);

%writing decimal data (or) original pixels values into the file

%decimal.value

%fid1=fopen('decimal.value','w');

%fprintf(fid1,'%i',lm);

%fclose(fid1);

%imshow(lm);

%figure(2);

39)

for t=1:1:500

f=2/500;

a=1;

x(t)=a\*(sin((2\*pi\*f\*(t-1))+.15\*pi));

y(t)=x(t);

end

plot(x);

pause

for i=1:1:500

if x(i)>0

y(i)=x(i);

else

y(i)=0;

end

end

plot(y);

40) function [y\_recover] = decoder(recIndex, code, channel\_out)

%DECODER Decodes a set of signals modulated using an orthogonal

% (Hadamard) code set. Each receiver must know the codeset

% of the sender. The signal of a particular sender is then

% determined based on the threshold of the conditional

% probability density functions of each bit. Since BPSK is

% used and each signal occurs with equal probability 1/2,

% threshold=0.

% DECODER(recIndex, code, channel\_out) decodes the signal assuming

% the recIndex signal is to be recovered. Takes in y\_mod\_hat, the original

% signal modulated by a codeset and possibly filtered, and

% generates a vector y\_recover that is the recovery of the

% original signal.

%

%Receiver knows codeset of sender.

%To decode, multiply output of adaptive filter bits by codeset.

%Then, recover data.

[rowsCode, colsCode] = size(code);

index = 1;

j = 1;

for i=1:length(channel\_out),

y\_unmod\_hat(recIndex,i) = channel\_out(i).\*code(recIndex,j);

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

j = 1; %reset back to beginning of codeset

y\_hat(recIndex, index) = (sum(y\_unmod\_hat(recIndex,:)))./colsCode; %perform decode of bit

y\_unmod\_hat(recIndex,:) = 0; %reset to 0 for next bit

index = index+1;

else

j = mod(i,colsCode)+1;

end

end;

%y\_hat

%Determine signal sent based on position relative to threshold

%Because of binary signals, threshold at 0.

for i=1:length(y\_hat(recIndex,:)),

if (y\_hat(recIndex, i) >= 0)

y\_recover(recIndex, i) = 1;

elseif (y\_hat(recIndex, i) < 0)

y\_recover(recIndex, i) = -1;

end

end

y\_recover = y\_recover(recIndex,:);

41) K=imread('logo.tif');

figure(1);

imshow(K);

a=size(K)

%a=K(:,:);

%size(a)

a1=a(1,1)

a2=a(1,2)

p=[];

disp ' i am before the for loop before converting into binary'

for i=1:a1

for j=1:a2

p = [p dec2bin(double(K(i,j)),4)];

end

%disp ' after the j loop'

i

end

ll=size(p)

% writing binary data into the file binary.value

fid=fopen('binaryl.value','w');

%fprintf(fid,'The binary values of the given image are');

fwrite(fid ,p);

fclose(fid);

% reading data from the file binary.value

% and converting it into decimal form

fid=fopen('binaryl.value','r');

disp ' before writing the data from the file'

fread(fid);

s=setstr(ans');

l=size(s)

disp ' after writing the data from the file'

m=1;

lm=[ ];

pp=[];

jj=(a2\*4);

while m<jj

kk=s(m:m+3);

lm=[lm bin2dec(kk)];

m=m+4;

end

oo=m;

while oo<l(1,2)

oo=m+jj;

pp=[];

while m<oo

kk=s(m:m+3);

pp=[pp bin2dec(kk)];

m=m+4;

end

lm=[lm;pp];

end

fclose(fid);

%writing decimal data into the file decimal.value

fid1=fopen('decimal.value','w');

%printf(fid1,'the data in decimal value is');

fprintf(fid1,'%i',lm);

fclose(fid1);

figure(2)

imshow(lm)

42) fid=fopen('binaryl.value','r');

fid1=fopen('hufman.value','w');

disp ' before reading the data from the file'

fread(fid);

s=setstr(ans');

l=size(s)

disp ' after reading the data from the file'

c=1;

q=0

i1=0

j0=0

q=q+1

fwrite(fid1,' before writing anything');

while q<=l(1,2)

% fprintf(fid1,'jfhgjf');

w=s(1,c)

if w==1

if j0~=0

fprintf(fid1,'0')

fprintf(fid1,j0)

j0=0;

c=c+1;

end

i1=i1+1

else

if i1~=0

fprintf(fid1,'1');

i1

fprintf(fid1,i1);

i1=0;

end

j0=j0+1

c=c+1;

end

q=q+1;

end

c

%m=1;

%lm=[ ];

%pp=[];

%jj=(a2\*4);

%while m<jj

% kk=s(m:m+3);

% lm=[lm bin2dec(kk)];

% m=m+4;

% end

% oo=m;

%while oo<l(1,2)

% oo=m+jj;

% pp=[];

% while m<oo

% kk=s(m:m+3);

% pp=[pp bin2dec(kk)];

% m=m+4;

% end

% lm=[lm;pp];

%end

fwrite(fid1,' after writing');

fclose(fid1);

fclose(fid);

43) ad=[];

bd=[];

ak=[];

bk=[];

n=input('Enter length of data array')

ad=input('Enter ad=')

bd=input('Enter bd=')

if length(ad)==n

if length(bd)==n

c=input('Enter the count for Hadamard matrix')

y=hadamard(c);

ak=y(1:4\*n);

bk=y(c\*c:-1:c\*c-4\*n+1);

for j=1:4\*n,

if ak(j)>0

ak(j)=1;

else ak(j)=0;

end

end

for j=1:4\*n,

if bk(j)>0

bk(j)=1;

else bk(j)=0;

end

end

y=1;

for j=1:n,

for i=y:y+3,

as(i)=xor(ad(j),ak(i));

bs(i)=xor(bd(j),bk(i));

end

y=y+4;

end

cs=or(as,bs);

for j=1:4\*n,

if cs(j)<=0

cs(j)=-1;

else cs(j)=1;

end

end

for j=1:4\*n,

if ak(j)<=0

ak(j)=-1;

else ak(j)=1;

end

end

for j=1:16,

if bk(j)<=0

bk(j)=-1;

else bk(j)=1;

end

end

ar1=cs.\*ak;

br1=cs.\*bk;

p=1;

s=[];

q=[];

for k=1:n,

s(k)=sum(ar1(p:p+3)');

if s(k)>0

s(k)=1;

else s(k)=0;

end

p=p+4;

end

p=1;

for k=1:n,

q(k)=sum(br1(p:p+3)');

if q(k)>0

q(k)=1;

else q(k)=0;

end

p=p+4;

end

ar=[s]

br=[q]

else

error('enter bd of length =n')

end

else

error('enter ad of length =n')

end

subplot(9,1,1)

bar(ad)

subplot(9,1,2)

bar(bd)

subplot(9,1,3)

bar(ak)

subplot(9,1,4)

bar(bk)

subplot(9,1,5)

bar(as)

subplot(9,1,6)

bar(bs)

subplot(9,1,7)

bar(cs)

subplot(9,1,8)

bar(ar)

subplot(9,1,9)

bar(br)

44) %MATLAB PROGRAM TO DISPLAYY ALL BASIC DISCRETE SIGNALS

%impulse function

N=input('enter the number of array values');

N=15555;a=2;

for i=1:N

x(i)=1;

end

for i=1:N

x1(i)=i.\*x(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

temp=0;

for n=1:N

e(n)=temp+(x(n).^2);

e1(n)=temp+(x1(n).^2);

e2(n)=temp+(x2(n).^2);

e3(n)=temp+(x3(n).^2);

e4(n)=temp+(x4(n).^2);

e5(n)=temp+(x5(n).^2);

end

temp1=0;

for n=1:N

p(n)=(temp1+(x(n).^2))./((2\*N)+1);

p1(n)=(temp1+(x1(n).^2))./((2\*N)+1);

p2(n)=(temp1+(x2(n).^2))./((2\*N)+1);

p3(n)=(temp1+(x3(n).^2))./((2\*N)+1);

p4(n)=(temp1+(x4(n).^2))./((2\*N)+1);

p5(n)=(temp1+(x5(n).^2))./((2\*N)+1);

end

subplot(12,1,1);plot(e);

subplot(12,1,2);plot(p);

subplot(12,1,3);plot(e1);

subplot(12,1,4);plot(p1);

subplot(12,1,5);plot(e2);

subplot(12,1,6);plot(p2);

subplot(12,1,7);plot(e3);

subplot(12,1,8);plot(p3);

subplot(12,1,9);plot(e4);

subplot(12,1,10);plot(p4);

subplot(12,1,11);plot(e5);

subplot(12,1,12);plot(p5);

45) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=(N-1)./2

for n=1:N

h(n)=(-1./(n\*pi)).\*sin(w\*n\*T);

end

k=1:l;

y(k)=h(k);

y1(k)=cos(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

end

w=0:0.15:pi;

w1=w(1:11);

H=freqw(b,1,w1);

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(w1./pi,a);

subplot(8,2,3);plot(w1./pi,b);

subplot(8,2,4);plot(w1./pi,20\*log10(abs(H)));

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(w1./pi,y2)

subplot(8,2,6);plot(w1./pi,y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

46) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=(N-1)./2

for n=1:N

h(n)=(-1./(n\*pi)).\*sin(w\*n\*T);

end

k=1:l;

y(k)=h(k);

y1(k)=cos(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

H(k)=b(k).\*exp(-j\*w\*k);

end

w=0:0.15:pi;

w1=w(1:11);

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(w1./pi,a);

subplot(8,2,3);plot(w1./pi,b);

subplot(8,2,4);plot(w1./pi,20\*log10(abs(H)));

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(w1./pi,y2)

subplot(8,2,6);plot(w1./pi,y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

47) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=N./2

for n=1:N

h(n)=(-1./(n\*pi)).\*sin(w\*n\*T);

end

k=1:l;

y(k)=h(k);

y1(k)=sin(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;theta=pi./2;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

H(k)=b(k).\*exp(-j\*w\*k).\*exp(j\*theta);

end

w=0:0.15:pi;

w1=w(1:11);

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(w1./pi,a);

subplot(8,2,3);plot(w1./pi,b);

subplot(8,2,4);plot(w1./pi,20\*log10(abs(H)));

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(w1./pi,y2)

subplot(8,2,6);plot(w1./pi,y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

48) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=(N)./2

for n=1:N

h(n)=(-1./(n\*pi)).\*sin(w\*n\*T);

end

k=1:l;

y(k)=h(k);

y1(k)=cos(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

H(k)=b(k).\*exp(-j\*w\*k);

end

w=0:0.15:pi;

w1=w(1:11)

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(a);

subplot(8,2,3);plot(b);

subplot(8,2,4);plot(w1./pi,20\*log10(abs(H)));

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(y2)

subplot(8,2,6);plot(y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

49) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=(N-1)./2

for n=1:N

h(n)=(-1./(n\*pi)).\*sin(w\*n\*T);

end

k=1:l;

y(k)=h(k);

y1(k)=sin(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;beta=pi./2;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

H(k)=b(k).\*exp(-j\*(w\*k)-beta);

end

w=0.1:0.005:pi;

w1=w(1:11);

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(w1./pi,a);

subplot(8,2,3);plot(w1./pi,b);

subplot(8,2,4);plot(w1./pi,H);

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(w1./pi,y2)

subplot(8,2,6);plot(w1./pi,y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

50) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=(N)./2

for n=1:N

h(n)=(-1./(n\*pi)).\*sin(w\*n\*T);

end

k=1:l;

y(k)=h(k);

y1(k)=sin(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;beta=pi./2;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

H(k)=b(k).\*exp(-j\*(w\*k)-beta);

end

w=0.1:0.005:pi;

w1=w(1:11);

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(w1./pi,a);

subplot(8,2,3);plot(w1./pi,b);

subplot(8,2,4);plot(w1./pi,H);

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(w1./pi,y2)

subplot(8,2,6);plot(w1./pi,y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

51) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=(N-1)./2

for n=1:N

h(n)=(-1./(n\*pi)).\*(sin(w\*n\*T)-cos(w\*n\*T));

end

k=1:l;

y(k)=h(k);

y1(k)=sin(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

H(k)=b(k).\*exp(-j\*w\*k);

end

w=0.1:0.005:pi;

w1=w(1:11);

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(w1./pi,a);

subplot(8,2,3);plot(w1./pi,b);

subplot(8,2,4);plot(w1./pi,H);

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(w1./pi,y2)

subplot(8,2,6);plot(w1./pi,y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

52) N=input('enter the sample points of linearphaseFIRfilters');

%N=23;

fs=input('enter sampling frequency');

w=2\*pi\*200;T=1./fs

l=(N-1)./2

for n=1:N

h(n)=(1./(n\*pi)).\*(cos(w\*n\*T)-sin(w\*n\*T));

end

k=1:l;

y(k)=h(k);

y1(k)=cos(w\*k);

k=2:l;

h1(k)=y(k);

k=3:l;

h2(k)=y(k);

k=4:l;

h3(k)=y(k);

k=5:l;

h4(k)=y(k);

k=6:l;

h5(k)=y(k);

k=7:l;

h6(k)=y(k);

k=8:l;

h7(k)=y(k);

k=9:l;

h8(k)=y(k);

k=10:l;

h9(k)=y(k);

k=11:l

h10(k)=y(k);

h12=h1+h2+h3+h4+h5+h6+h7+h8+h9+h10;

temp=0;

for k=1:11

a(k)=2\*h12(k);

b(k)=temp+a(k).\*y1(k);

H(k)=b(k).\*exp(-j\*w\*k);

end

w=0.1:0.005:pi;

w1=w(1:11);

y2=(abs(H));

y3=angle(H);

[p]=periodogram(y3,[],'twosided',fs,23);

y4=p';y5=-y4;

subplot(8,2,1);plot(h);

title('impulse response of given highpass filter');

subplot(8,2,2);plot(w1./pi,a);

subplot(8,2,3);plot(w1./pi,b);

subplot(8,2,4);plot(w1./pi,H);

title('impulse response of given linear phase FIR-HPF using equiripple method');

subplot(8,2,5);plot(w1./pi,y2)

subplot(8,2,6);plot(w1./pi,y3);

subplot(8,2,7);plot(y4);

subplot(8,2,8);plot(y5);

title('plot of groupdelay of given highpass filter');

53) N=input('enter the no of array values');

N=7;

n=1:7

x1(n)=ones(1,7)

x2(n)=exp(-2\*n).\*x1(n)

x3(n)=exp(j\*(2\*n+(pi)./4))

x4(n)=cos(n)

x5(n)=((0.5).^n).\*x1(n)

x6(n)=sin((pi./4)\*n)

y2(n)=exp(-2\*(-n))

y3(n)=exp(j\*(2\*(-n)+(pi)./4))

y4(n)=cos(-n)

y5(n)=((0.5).^(-n))

y6(n)=sin((pi)./4 \*(-n))

y7(n)=-y2(n);y8(n)=-y3(n);y9(n)=-y4(n);y10(n)=-y5(n);y11(n)=-y6(n);

if(x2(n)==y2(n))

disp('the given x2 is even')

end

if(x2(n)== -y2(n))

disp('the given x2 is odd ')

end

if (x3(n)==y3(n))

disp('the given signal x3 is even')

end

if (x3(n)==-y3(n))

disp('the given signal x3 is odd')

end

if (x4(n)==y4(n))

disp('the given signal x4 is even')

end

if(x4(n)==-y4(n))

disp('the given x4 is odd')

end

if (x5(n)==y5(n))

disp('the given signal x5 is even')

end

if (x5(n)==-y5(n))

disp('the given signal x5 is odd')

end

if (x6(n)==y6(n))

disp('the given signal x6 is even')

end

if(x6(n)==-y6(n))

disp('the given x6 is odd')

end

54) %to generate pn sequence for six users

i=input('enter the no of shift registers');

for i=1:6

a(i)=input('enter the array values for a(i)');

end

for i=1:63

y=xor(a(2),a(4));

a(6)=a(5);

a(5)=a(4);

a(4)=a(3);

a(3)=a(2);

a(2)=a(1);

a(1)=y;

pn(i)=[a(6)];

end

55) N=input('enter the no of samples');

N=23;alpha=(N-1)./2;

%fs=input('enter the sampling method');

%fs=200

%fc=input('enter the cutoff frequency');

%fc=50

%wc=2\*pi\*fc;ws=2\*pi\*fs;

t=1:alpha;

H(t)=1

k=1:alpha

thetak=(-alpha\*(2\*pi)./(2\*N))\*((2\*k)+1);

H1(k)=H(k).\*(exp(i\*thetak));

n=1:alpha;

hn=real(ifft(H1,N));

w=0:0.15:pi;

w1=w(1:11);

H2=freqz(hn,1,w1);

plot(w1./pi,20\*log10(abs(H2)));

56) N=input('enter the no of samples');

N=33;

fs=input('enter the sampling method');

fs=200

fc=input('enter the cutoff frequency');

fc=50

wc=2\*pi\*fc;ws=2\*pi\*fs;

t=wc:ws./2;

H(t)=1

alpha=(N-1)./2

k=1:alpha

thetak=(-alpha\*(2\*pi)./N)\*k

57) N=input('enter the no of samples');

N=23;alpha=(N-1)./2;

%fs=input('enter the sampling method');

%fs=200

%fc=input('enter the cutoff frequency');

%fc=50

%wc=2\*pi\*fc;ws=2\*pi\*fs;

t=1:alpha;

H(t)=1

k=1:alpha

thetak=(-alpha\*(2\*pi)./(2\*N))\*((2\*k)+1);

H1(k)=H(k).\*(exp(i\*thetak));

n=1:alpha;

hn=real(ifft(H1,N));

w=0:0.15:pi;

w1=w(1:11);

H2=freqz(hn,1,w1);

plot(w1./pi,20\*log10(abs(H2)));

58) fs=input('enter the sampling method');

fs=200

fc=input('enter the cutoff frequency');

fc=50

wc=2\*pi\*fc;ws=2\*pi\*fs;

w=314:628

H(w)=1;

w1=-628:-314

H1(w1)=1;

H2=H+H1

plot(H2)

59) function two\_channelencoding(x)

imshow(x)

[rows colos]=size(x);

a=rows/16;

b=floor(a);

if a>b

b=b+1;

newrows=b\*16;

addrows=newrows-rows;

q=ones(addrows,colos);

x=[x;q];

q=[ ];

end

[newrows colos]=size(x);

a=colos/16;

b=floor(a);

if a>b

b=b+1;

newcolos=b\*16;

addcolos=newcolos-colos;

q=ones(newrows,addcolos);

x=[x q];

q=[ ];

end

disp 'size is'

size(x)

imshow(x)

60) % Program illustrating time reversal operation on the given input sequence by an amount 1

N=input('enter the no of sample points of discrete time sequence');

N=6;

for i=1:N

x(i)=input('enter the input');

end

%the following loop indicates the time reversal sequence of the input sequence

for i=1:N

y(i)=-x(i);

end

subplot(4,1,1),plot(x);

subplot(4,1,2),plot(y);

subplot(4,1,3),stem(x);

subplot(4,1,4),stem(y);

61) N=input('enter the no of array values');

N=7;

i=1:7;

x1(i)=[ones(1,7)]

for i=1:7

x2(i)=exp(-2\*i).\*x1(i);

x3(i)=exp(j\*((2\*i)+(pi)./4));

x4(i)=cos(i);

x5(i)=((0.5).^i).\*x1(i);

x6(i)=sin((pi)./(4)\*i);

end

l=(N-1)./2;

for i=1:l

y1(i)=x1(i);

y2(i)=x2(i);

y3(i)=x3(i);

y4(i)=x4(i);

y5(i)=x5(i);

y6(i)=x6(i);

end

subplot(12,1,1);plot(x1);

subplot(12,1,2);plot(x2);

subplot(12,1,3);plot(x3);

subplot(12,1,4);plot(x4);

subplot(12,1,5);plot(x5);

subplot(12,1,6);plot(x6);

subplot(12,1,7);plot(y1);

subplot(12,1,8);plot(y2);

subplot(12,1,9);plot(y3);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(y5);

subplot(12,1,12);plot(y6);

62) % Program illustrating time scaling operation on the given input sequence

% by an amount 1/2

N=input('enter the no of sample points of discrete time sequence');

N=6;

for i=1:N

x(i)=input('enter the input');

end

%the following loop indicates the time scaled version of the given sequence of the input sequence

for i=1:N/2

y(i)=x(i);

end

subplot(4,1,1),plot(x);

subplot(4,1,2),plot(y);

subplot(4,1,3),stem(x);

subplot(4,1,4),stem(y);

63) N=input('enter the no of sample points of discrete time systems');

N=6;

l=N./2;

for i=1:6

x(i)=1;

end

for i=1:N

x1(i)=i.\*x(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

for i=1:3

y(i)=x(i);

y1(i)=x1(i);

y2(i)=x2(i);

y3(i)=x3(i);

y4(i)=x4(i);

y5(i)=x5(i);

end

subplot(12,1,1);plot(x);

subplot(12,1,2);plot(y);

subplot(12,1,3);plot(x1);

subplot(12,1,4);plot(y1);

subplot(12,1,5);plot(x2);

subplot(12,1,6);plot(y2);

subplot(12,1,7);plot(x3);

subplot(12,1,8);plot(y3);

subplot(12,1,9);plot(x4);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(x5);

subplot(12,1,12);plot(y5);

64) N=input('enter the no of sample points of discrete time systems');

N=6

l=-N;

x(1:6)=1;

i=l:N;

yi=x(1:6);

for i=1:N

x1(i)=i.\*x(i);

end

for i=1:N

y1(i)=(-i).\*yi(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

for i=1:N

y2(i)=(a.^(-i)).\*yi(i) ;

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=1:N

y3(i)=exp(-j\*w\*i)+exp(-j\*w\*-i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

for i=1:N

y4(i)=sin(-w\*i);

y5(i)=cos(-w\*i);

end

subplot(12,1,1);plot(x);

subplot(12,1,2);plot(yi);

subplot(12,1,3);plot(x1);

subplot(12,1,4);plot(y1);

subplot(12,1,5);plot(x2);

subplot(12,1,6);plot(y2);

subplot(12,1,7);plot(x3);

subplot(12,1,8);plot(y3);

subplot(12,1,9);plot(x4);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(x5);

subplot(12,1,12);plot(y5);

65) for t=1:1:500

f=2/500;

a=.10;

x(t)=a\*(cos(2\*pi\*f\*(t-1)));

y(t)=x(t);

end

plot(x);

pause

for i=1:1:500

if x(i)>0

y(i)=.1;

else

y(i)=-.1;

end

end

plot(y);

66) for t=1:1:500

f=2/500;

a=.10;

x(t)=a\*(sin(2\*pi\*f\*(t-1)));

y(t)=x(t);

end

plot(x);

pause

for i=1:1:500

if x(i)>0

y(i)=.1;

else

y(i)=-.1;

end

end

plot(y);

67) for t=1:1:500

f=2/500;

a=.10;

x(t)=a\*[1+cos(2\*pi\*f\*(t-1)+1\*pi)];

y(t)=x(t);

end

plot(x);

pause

for i=1:1:500

if x(i)>0

y(i)=x(i);

else

y(i)=0;

end

end

plot(y);

68) for t=1:1:500

f=2/500;

a=.10;

x(t)=a\*(cos(2\*pi\*f\*(t-1)));

y(t)=x(t);

end

plot(x);

pause

for i=1:1:500

if x(i)>0

y(i)=x(i);

else

y(i)=0;

end

end

plot(y);

69) for t=1:1:500

f=2/500;

a=.10;

x(t)=a\*(sin(2\*pi\*f\*(t-1)));

y(t)=x(t);

end

plot(x);

pause

for i=1:1:500

if x(i)>0

y(i)=x(i);

else

y(i)=0;

end

end

plot(y);

70) m=[];

n=[];

%generate the input sinusoidal signal

N=input('type in length of the input sequence=');

A=input('type in amplitude =');

W0=2\*pi\*20;

n=0:.125:N-.125;

m=n-1;

t=0:.125:N-.125;

x1=A\*cos(W0\*t);

plot(t,x1);

pause

x=A\*cos(W0\*n);

axis([0 N -A A]);

stem(m,x);

xlabel('time index');

ylabel('amplitude');

title('input digital signal');

pause

%generation of quantised output %delta sigma modulation

y=zeros(1,N+1);

a=zeros(1,N+1);

e1=[];

e=0;

for k=2:length(n)

a(k)=x(k-1)-e;

if a(k)>=0

y(k)=1;

else

y(k)=-1;

end

e=y(k)-a(k);

e1=[e,e1];

end

%yn=y(1:N+1);

axis([0 N -1 1]);

%plot the quantised output

stem(t,y);

%tairs(t,y);

xlabel('time');

ylabel('amplitude');

title('digital output of sigma delta quantizer');

71) a=[1 0 1];

b(1:6)=a(1);

b(7:14)=a(2);

b(15:24)=a(3);

c=[1 1 0];

d(1:6)=c(1);

d(7:14)=c(2);

d(15:24)=c(3);

bkey=[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 ];

dkey=[0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 ];

bs=xor(b,bkey);

ds=xor(d,dkey);

es=or(bs,ds);

br=es.\*bkey;

dr=es.\*dkey;

for l=1:length(br)

if br(l) > 0

br(l) = 1;

else

br(l)=-1;

end

end

for m=1:length(dr)

if dr(m) > 0

dr(m) = 1;

else

dr(m)=-1;

end

end

z=[ ];

a1=sum(br(1:6));

z=[z,a1]

a1=sum(br(7:14));

z=[z,a1]

a1=sum(br(15:24));

z=[z,a1]

for i=1:length(z)

if z(i) > 0

z(i)=1;

else

z(i)=0;

end

end

y=[ ];

a2=sum(dr(1:6));

y=[y,a2]

a2=sum(dr(7:14));

y=[y,a2]

a2=sum(dr(15:24));

y=[y,a2]

for j=1:length(y)

if y(j) > 0

y(j)=1;

else

y(j)=0;

end

end

72) clear;

Ad=[];

Bd=[];

data1=input('Give binary data for:A Station \n');

data2=input('Give binary data for:Station B \n');

Ad=[Ad,data1];

Bd=[Bd,data2];

Ak=[];

Bk=[];

Akey=input('give Key for A\n');

Bkey=input('Give Key for B\n');

Ak=[Ak,Akey];

Bk=[Bk,Bkey];

r = 1;

d = length(Ak)/length(Ad);

y =1;

z=d;

for i = 1:1:(length(Ad)),

% if d <= 3

As(y:z) = xor(Ad(i),Ak(y:(z)));

Bs(y:z) = xor(Bd(i),Bk(y:(z)));

y = d\*(i)+1;

z=z+d;

% end

end;

Cs= or(As,Bs);

for i=1:1:(length(Cs));

if Cs(i) > 1

Cs(i)=1;

end

if Cs (i) < 1

Cs(i)=-1;

end

end

for i=1:length(Ak)

if Ak(i)==0

Ak(i)= -1;

elseif Ak(i)==1

Ak(i)=1;

end

end

%%% Transmitted signal is Cs here.

for i=1:length(Bk)

if Bk(i)==0

Bk(i)= -1;

elseif Bk(i)==1

Bk(i)=1;

end

end

x=sum(Ak.\*Bk);

if x==0

fprintf('The Keys are Orthogonal\n');

Ar1 = Cs.\*Ak;

Br1 = Cs.\*Bk;

% It is assumed that along with sender data, length of data is also received at the receiver.

d(1) = length(Ad);

d(2) = length(Bd);

D(1) = length(Ak)/d(1);

D(2) = length(Bk)/d(2);

sum = 0;

r = 1;

y = 1;

for i = 1:1:d(1)

while(r<= D(1) )

sum = sum+ Ar1(y);

r = r+1;

y = y+1;

end

Ar(i) = sum;

r =1;

sum = 0;

if (Ar(i) > 0)

Ar(i) = 1;

elseif (Ar(i)<=0)

Ar(i) = 0;

end

end

sum =0;

y =1;

r = 1;

for i = 1:1:d(2)

while(r<= D(2))

sum = sum+ Br1(y);

r = r+1;

y = y+1;

end

Br(i) = sum;

r =1;

sum = 0;

if (Br(i) > 0)

Br(i) = 1;

else

Br(i) = 0;

end

end

Subplot(9,1,1), bar(Ad,'c'),ylabel('Ad');

Subplot(9,1,2), bar(Bd,'c'),ylabel('Bd');

Subplot(9,1,3), bar(Ak,'c'),ylabel('Ak');

% axis([1 4 0 1]);

% axis([x,y,0,4,0,1]);

Subplot(9,1,4), bar(Bk,'c'),ylabel('Bk');

% axis([1 4 0 1]);

% axis([x,y,0,4,0,1]);

Subplot(9,1,5), bar(As,'c'),ylabel('As');

Subplot(9,1,6), bar(Bs,'c'),ylabel('Bs');

Subplot(9,1,7), bar(Cs,'c'),ylabel('Cs');

Subplot(9,1,8), bar(Ar,'c'),ylabel('Ar');

Subplot(9,1,9), bar(Br,'c'),ylabel('Br');

% subplot(7,1,1);

else

fprintf('The Keys are not Orthogonal:There is No Transmittion\n');

end

73) function sinn= sinfn(r);

theta=linspace(0,2\*pi,100);

r=2.5678;

sinn=r\*sin(theta);

plot(sinn);

xlabel('x');

ylabel('theta');

title('sin wave');

74) t=1:1:500

f=2/500;

a=1;

x=a\*(sin((2\*pi\*f\*t)+.15\*pi));

for i=1:length(x)

if x(i)>0

y=x(i);

else

y=0;

end

end

for i=1:length(y)

r=y(i);

end

plot(r);

75) for t=1:.1:50

f=2/500;a=1;x=1;

i(x)=a\*(cos((2\*pi\*f\*t)+.15\*pi));

if i<0

break;

end

i=i+1;

end

plot(i);

76) for t=1:1:500

f=2/500;

a=1;

x=1;

i(x)=a\*(cos((2\*pi\*f\*t)+.15\*pi));

if i(x)>=0

z(x)=i(x)

else

z(x)=0;

end

x=x+1;

end

plot(x,i,'bd');

pause

plot(x,z);

77) N=input('enter the no of array values');

N=7;

i=1:7;

x1(i)=[ones(1,7)]

for i=1:7

x2(i)=exp(-2\*i).\*x1(i);

x3(i)=exp(j\*((2\*i)+(pi)./4));

x4(i)=cos(i);

x5(i)=((0.5).^i).\*x1(i);

x6(i)=sin((pi)./(4)\*i);

end

for i=1:7

y1(i)=x1(i).\*x2(i);

y2(i)=x2(i).\*x3(i);

y3(i)=x3(i).\*x4(i);

y4(i)=x4(i).\*x5(i);

y5(i)=x5(i).\*x6(i);

y6(i)=x6(i).\*x1(i);

end

subplot(12,1,1);plot(x1);

subplot(12,1,2);plot(x2);

subplot(12,1,3);plot(x3);

subplot(12,1,4);plot(x4);

subplot(12,1,5);plot(x5);

subplot(12,1,6);plot(x6);

subplot(12,1,7);plot(y1);

subplot(12,1,8);plot(y2);

subplot(12,1,9);plot(y3);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(y5);

subplot(12,1,12);plot(y6);

78) N=input('enter the no of sample points of discrete time systems');

N=6

for i=1:6

x(i)=1;

end

for i=1:N

x1(i)=i.\*x(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

for i=1:N

y1(i)=x(i).\*x1(i);

y2(i)=x1(i).\*x2(i);

y3(i)=x2(i).\*x3(i);

y4(i)=x3(i).\*x4(i);

y5(i)=x4(i).\*x5(i);

end

subplot(12,1,1);plot(x);

%subplot(12,1,2);plot(y);

subplot(12,1,3);plot(x1);

subplot(12,1,4);plot(y1);

subplot(12,1,5);plot(x2);

subplot(12,1,6);plot(y2);

subplot(12,1,7);plot(x3);

subplot(12,1,8);plot(y3);

subplot(12,1,9);plot(x4);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(x5);

subplot(12,1,12);plot(y5);

79) N=input('enter the no of sample points of discrete time systems');

N=6

for i=1:6

x(i)=1;

end

for i=2:6

y(i)=x(i);

end

for i=1:N

x1(i)=i.\*x(i);

end

for i=2:N

y1(i)=x1(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

for i=2:N

y2(i)=x2(i) ;

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=2:N

y3(i)=x3(i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

for i=2:N

y4(i)=x4(i);

y5(i)=x5(i);

end

subplot(12,1,1);plot(x);

subplot(12,1,2);plot(y);

subplot(12,1,3);plot(x1);

subplot(12,1,4);plot(y1);

subplot(12,1,5);plot(x2);

subplot(12,1,6);plot(y2);

subplot(12,1,7);plot(x3);

subplot(12,1,8);plot(y3);

subplot(12,1,9);plot(x4);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(x5);

subplot(12,1,12);plot(y5);

80) N=input('enter the no of array values');

N=7;

i=1:7;

x1(i)=[ones(1,7)]

for i=1:7

x2(i)=exp(-2\*i).\*x1(i);

x3(i)=exp(j\*((2\*i)+(pi)./4));

x4(i)=cos(i);

x5(i)=((0.5).^i).\*x1(i);

x6(i)=sin((pi)./(4)\*i);

end

for i=1:7

y1(i)=x1(i)+x2(i);

y2(i)=x2(i)+x3(i);

y3(i)=x3(i)+x4(i);

y4(i)=x4(i)+x5(i);

y5(i)=x5(i)+x6(i);

y6(i)=x6(i)+x1(i);

end

subplot(12,1,1);plot(x1);

subplot(12,1,2);plot(x2);

subplot(12,1,3);plot(x3);

subplot(12,1,4);plot(x4);

subplot(12,1,5);plot(x5);

subplot(12,1,6);plot(x6);

subplot(12,1,7);plot(y1);

subplot(12,1,8);plot(y2);

subplot(12,1,9);plot(y3);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(y5);

subplot(12,1,12);plot(y6);

81) % Program illustrating signal multiplier operation on the given input sequences

N=input('enter the no of sample points of discrete time sequence');

N=4;

for i=1:N

x1(i)=input('enter the input x1');

x2(i)=input('enter the input x2');

end

%the loop indicates the signal multiplier sequences of the given input sequences

for i=1:N

y(i)=x1(i)+x2(i);

end

subplot(6,1,1),plot(x1);

subplot(6,1,2),plot(x2);

subplot(6,1,3),plot(y);

subplot(6,1,4),stem(x1);

subplot(6,1,5),stem(x2);

subplot(6,1,6),stem(y);

82) N=input('enter the no of sample points of discrete time systems');

N=6

for i=1:6

x(i)=1;

end

for i=1:N

x1(i)=i.\*x(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

for i=1:N

y1(i)=x(i)+x1(i);

y2(i)=x1(i)+x2(i);

y3(i)=x2(i)+x3(i);

y4(i)=x3(i)+x4(i);

y5(i)=x4(i)+x5(i);

end

subplot(12,1,1);plot(x);

%subplot(12,1,2);plot(y);

subplot(12,1,3);plot(x1);

subplot(12,1,4);plot(y1);

subplot(12,1,5);plot(x2);

subplot(12,1,6);plot(y2);

subplot(12,1,7);plot(x3);

subplot(12,1,8);plot(y3);

subplot(12,1,9);plot(x4);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(x5);

subplot(12,1,12);plot(y5);

82) N=input('enter the no of array values');

N=7;

i=1:7;

x1(i)=[ones(1,7)]

for i=1:7

x2(i)=exp(-2\*i).\*x1(i);

x3(i)=exp(j\*((2\*i)+(pi)./4));

x4(i)=cos(i);

x5(i)=((0.5).^i).\*x1(i);

x6(i)=sin((pi)./(4)\*i);

end

for i=2:7

y1(i)=x1(i);

y2(i)=x2(i);

y3(i)=x3(i);

y4(i)=x4(i);

y5(i)=x5(i);

y6(i)=x6(i);

end

subplot(12,1,1);plot(x1);

subplot(12,1,2);plot(x2);

subplot(12,1,3);plot(x3);

subplot(12,1,4);plot(x4);

subplot(12,1,5);plot(x5);

subplot(12,1,6);plot(x6);

subplot(12,1,7);plot(y1);

subplot(12,1,8);plot(y2);

subplot(12,1,9);plot(y3);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(y5);

subplot(12,1,12);plot(y6);

83) % Program illustrating shifting of the given input sequence by an amount 1

N=input('enter the no of sample points of discrete time sequence');

N=6;

for i=1:N

x(i)=input('enter the input');

end

%the following loop indicates the shifed sequence of the input sequence

for i=2:N

y(i)=x(i);

end

subplot(4,1,1),plot(x);

subplot(4,1,2),plot(y);

subplot(4,1,3),stem(x);

subplot(4,1,4),stem(y);

84) N=input('enter the no of sample points of discrete time systems');

N=6

for i=1:6

x(i)=1;

end

for i=2:6

y(i)=x(i);

end

for i=1:N

x1(i)=i.\*x(i);

end

for i=2:N

y1(i)=x1(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

for i=2:N

y2(i)=x2(i) ;

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=2:N

y3(i)=x3(i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

for i=2:N

y4(i)=x4(i);

y5(i)=x5(i);

end

subplot(12,1,1);plot(x);

subplot(12,1,2);plot(y);

subplot(12,1,3);plot(x1);

subplot(12,1,4);plot(y1);

subplot(12,1,5);plot(x2);

subplot(12,1,6);plot(y2);

subplot(12,1,7);plot(x3);

subplot(12,1,8);plot(y3);

subplot(12,1,9);plot(x4);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(x5);

subplot(12,1,12);plot(y5);

85) %matlab program to print all discrete time signals in the range -6 to 6

N=input('enter the number of array values');

N=6;

n=1:N;

xn=[ones(1,1),zeros(1,N)];

n=2:N;

yn=xn

subplot(7,1,1);plot(yn)

xlabel('time');ylabel('magnitude');

title('unit impulse signal');

n=1:N

x1n=[ones(1,N)];

n1=2:6;

y1n1=x1n

subplot(7,1,2);plot(y1n)

xlabel('time');ylabel('magnitude');

title('unit step sequence');

n=1:6;

x2n=n.\*x1n;

n=2:6

y2n=x2n

subplot(7,1,3);plot(y2n)

xlabel('time');ylabel('magnitude');

title('unit ramp sequence');

w=2\*pi\*600

n=1:6;

x3n=exp(j\*w\*n);

n=2:6;

y3n=x3n

n=1:6;

x4n=exp(-j\*w\*n);

n=2:6;

y4n=x4n

n=1:6;

x5n=(x3n+x4n)./2;

n=2:6;

y5n=x5n

n=1:6;

x6n=(x3n-x4n)./(2\*j);

n=2:6;

y6n=x6n

n=1:6;

x7n=sin(w\*n);

n=2:6;

y7n=x7n

n=1:6;

x8n=cos(w\*n);

n=2:6;

y8n=x8n

subplot(7,1,4);plot(y3n);

xlabel('time');ylabel('magnitude');

title('exponentially increasing signal');

subplot(7,1,5);plot(y4n);

xlabel('time');ylabel('magnitude');

title('exponentially decreasing signal');

subplot(7,1,6);plot(y5n);

xlabel('time');ylabel('magnitude');

title('cosinusoidal signal');

subplot(7,1,7);plot(y6n);

xlabel('time');ylabel('magnitude');

title('sinusoidal signal');

86) N=input('enter the number of array values');

N=123;

n=1:123;

x1(n)=exp(j\*n);

n=1:123;

x2(n)=exp(-j\*n);

a=2.5;b=1;

n=1:123;

y1(n)=a.\*x1(n)+b;

y2(n)=a.\*x2(n)+b;

a1=1.5;a2=4.5;

n=1:123;

z1(n)=(a1.\*y1(n))+(a2.\*y2(n));

z2(n)=(a.\*(x1(n)+x2(n)))+b;

if (z1==z2)

disp('the given system is linear one');

end

if(z1~=z2)

disp('the given system is nonlinear one');

end

subplot(2,1,1);plot(z1);

subplot(2,1,2);plot(z2);

88) z=[];

n=[];

%generate the input sinusoidal signal

N=input('type in length of input sequence');

A=input('type in amplitude');

f0=input('enter the value of synthesizing frequency');

%f0=10;

W0=2\*pi\*f0;

n=0:.006250:N-.006250;

m=n-1;

t=0:.006250:N-.006250;

x=A\*cos(W0\*t);

plot(x);

xlabel('time');

ylabel('amplitude');

title('input audio signal');

pause

%plot the sampled signal

x1=A\*cos(W0\*n);

axis([0 N -A A]);

stem(n,x1);

title('input digital signal sampled');

pause

%generation of quantised output

a=zeros(1,420);

for t1=(1:420)

a(t1)=round(x(t1));

end

plot(a);

xlabel('time');

ylabel('amplitude');

title('digital output of quantiser');

pause

89) N=input('enter the no of sample points of discrete time systems');

N=6

for i=1:6

x(i)=1;

end

for i=1:6

y(i)=2.\*x(i);

end

for i=1:N

x1(i)=i.\*x(i);

end

for i=1:N

y1(i)=2.\*x1(i);

end

a=0.5;

for i=1:N

x2(i)=(a.^i).\*x(i);

end

for i=1:N

y2(i)=2.\*x2(i) ;

end

w=2\*pi\*100;

for i=1:N

x3(i)=exp(j\*w\*i)+exp(-j\*w\*i);

end

for i=1:N

y3(i)=2.\*x3(i);

end

for i=1:N

x4(i)=sin(w\*i);

x5(i)=cos(w\*i);

end

for i=1:N

y4(i)=2.\*x4(i);

y5(i)=2.\*x5(i);

end

subplot(12,1,1);plot(x);

subplot(12,1,2);plot(y);

subplot(12,1,3);plot(x1);

subplot(12,1,4);plot(y1);

subplot(12,1,5);plot(x2);

subplot(12,1,6);plot(y2);

subplot(12,1,7);plot(x3);

subplot(12,1,8);plot(y3);

subplot(12,1,9);plot(x4);

subplot(12,1,10);plot(y4);

subplot(12,1,11);plot(x5);

subplot(12,1,12);plot(y5);

90) % function m= bpsk111(bit)

% to understand the concept of spreading

tb=input('enter the array of tb');

%tb=[0 1];

tc=input('enter the array of tc');

%tc=[0 1 1 0 1 0 1 0 1 1 0 1 0 1];

sf=length(tc)/length(tb);

rtb(1:sf)=0;

rtc(sf:2\*sf)=1;

rs=xor(tc,rtc);

subplot(3,1,1);bar(tb);xlabel('tb');

subplot(3,1,2);bar(tc);xlabel('tc');

subplot(3,1,3);bar(rs);xlabel('rs');

%to generate spreaded pnsequence

i=input('enter the number of shift registers');

n=input('enter the no of bits applied as input to the shift registers');

m=input('enter the no of pn sequences to be generated');

%n=3,m=2;

for i=1:length(m)

for i=1:n

a(i)=input('enter the array values for a(i)');

end

for i=1:((2^n)-1)

y=xor(a(2),a(3));

a(3)=a(2);

a(2)=a(1);

a(1)=y;

pn(i)=[a(1)];

end

gc=xor(pn,pn);

end

data=input('enter the input data');

for i=1:data

d(i)=input('enter the array values for d(i)');

end

out=[];

for k=1:data

for j=1:length(gc)

y1=xor(d(k),gc(j));

out=[out,y1];

end

end

end

91) % Program illustrating time reversal operation on the given input sequence by an amount 1

N=input('enter the no of sample points of discrete time sequence');

N=6;

for i=1:N

x(i)=input('enter the input');

end

%the following loop indicates the time reversal sequence of the input sequence

for i=1:N

y(i)=-x(i);

end

subplot(4,1,1),plot(x);

subplot(4,1,2),plot(y);

subplot(4,1,3),stem(x);

subplot(4,1,4),stem(y);

92) tb=input('enter the array of tb');

%tb=[0 1];

tc=input('enter the array of tc');

%tc=[0 1 1 0 1 0 1 0 1 1 0 1 0 1];

sf=length(tc)/length(tb);

rtb(1:sf)=0;

rtc(sf:2\*sf)=1;

rs=xor(tc,rtc);

subplot(3,1,1);bar(tb);

xlabel('tb');

subplot(3,1,2);bar(tc);

xlabel('tc');

subplot(3,1,3);bar(rs);

xlabel('rs')

93) N=input('enter the no of array values');

N=233;

for i=1:233

x1(i)=1;

x2(i)=exp(-2\*i).\*x1(i);

x3(i)=exp(j\*(2\*i+(pi)./4));

x4(i)=cos(i);

x5(i)=((0.5).^i).\*x1(i);

x6(i)=sin((pi)./(4)\*i);

end

temp=0;

for i=1:233

e1(i)=temp+(x1(i).^2);

e2(i)=temp+(x2(i).^2);

e3(i)=temp+(x3(i).^2);

e4(i)=temp+(x4(i).^2);

e5(i)=temp+(x5(i).^2);

e6(i)=temp+(x6(i).^2);

end

temp1=0;

for i=1:233

p1(i)=temp1+(e1(i).^2)./((2\*N)+1);

p2(i)=temp1+(e2(i).^2)./((2\*N)+1);

p3(i)=temp1+(e3(i).^2)./((2\*N)+1);

p4(i)=temp1+(e4(i).^2)./((2\*N)+1);

p5(i)=temp1+(e5(i).^2)./((2\*N)+1);

p6(i)=temp1+(e6(i).^2)./((2\*N)+1);

end

subplot(12,1,1);plot(e1);

subplot(12,1,2);plot(e2);

subplot(12,1,3);plot(e3);

subplot(12,1,4);plot(e4);

subplot(12,1,5);plot(e5);

subplot(12,1,6);plot(e6);

subplot(12,1,7);plot(p1);

subplot(12,1,8);plot(p2);

subplot(12,1,9);plot(p3);

subplot(12,1,10);plot(p4);

subplot(12,1,11);plot(p5);

subplot(12,1,12);plot(p6);

if(0<e1<inf)

disp('the given signal is an example of energy signal');

end

if(0<p1<inf)

disp('the given signal is an example of power signal');

end

if(0<e2<inf)

disp('the given signal is an example of energy signal');

end

if(0<p2<inf)

disp('the given signal is an example of power signal');

end

if(0<e3<inf)

disp('the given signal is an example of energy signal');

end

if(0<p3<inf)

disp('the given signal is an example of power signal');

end

if(0<e4<inf)

disp('the given signal is an example of energy signal');

end

if(0<p4<inf)

disp('the given signal is an example of power signal');

end

if(0<e5<inf)

disp('the given signal is an example of energy signal');

end

if(0<p5<inf)

disp('the given signal is an example of power signal');

end

if(0<e6<inf)

disp('the given signal is an example of energy signal');

end

if(0<p6<inf)

disp('the given signal is an example of power signal');

end

94) N=input('enter the no of array values');

N=7;

n=-7:1:7

x1n=[zeros(1,7),ones(1,7)];

n=-7:1:6;

x2n=exp(-2\*n).\*x1n;

x3n=exp(j\*(2\*n+(pi)./4))

x4n=cos(n);

x5n=((0.5).^n).\*x1n;

x6n=sin((pi)./(4)\*n);

subplot(5,1,1);plot(n,x2n);

subplot(5,1,2);plot(n,x3n);

subplot(5,1,3);plot(n,x3n);

subplot(5,1,4);plot(n,x4n);

subplot(5,1,5);plot(n,x5n);

95) N=input('enter the number of array values');

N=123;

n=1:123;

x1(n)=exp(j\*n);

n=1:123;

x2(n)=exp(-j\*n);

a=2.5;b=0;

n=1:123;

y1(n)=a.\*x1(n)+b;

y2(n)=a.\*x2(n)+b;

a1=1.5;a2=4.5;

n=1:123;

z1(n)=(a1.\*y1(n))+(a2.\*y2(n));

z2(n)=(a.\*(a1.\*x1(n)+a2.\*x2(n)))+b;

if (z1 == z2)

disp('the given system is satisfying linearity property one');

end

if(z1~=z2)

disp('the given system is nonlinear one');

end

subplot(2,1,1);stem(z1);

subplot(2,1,2);stem(z2);

96) N=input('enter the number of array values');

N=123;

n=1:123;

x1(n)=exp(j\*n);

x2(n)=exp(-j\*n);

a=2.5;

y1(n)=a.\*x1(n);

y2(n)=a.\*x2(n);

a1=1.5;a2=4.5;

z1(n)=(a1.\*y1(n))+(a2.\*y2(n));

z2(n)=(a.\*(a1.\*x1(n)+a2.\*x2(n)));

if (z1(n)==z2(n))

disp('the given system is satisfying linearity property one');

end

if(z1(n)~=z2(n))

disp('the given system is nonlinear one');

end

subplot(2,1,1);stem(z1);

subplot(2,1,2);stem(z2);

97) N=input('enter the no of array values');

N=233;

n=1:233

x1(n)=ones(1,233)

x2(n)=exp(-2\*n).\*x1(n)

x3(n)=exp(j\*(2\*n+(pi)./4))

x4(n)=cos(n)

x5(n)=((0.5).^n).\*x1(n)

x6(n)=sin((pi./4)\*n)

x8(n)=exp(-2\*(n+N))

y2(n)=x8(n)

x9(n)=exp(j\*(2\*(n+N)+(pi)./4));

y3(n)=x9(n)

x10(n)=cos(n+N);

y4(n)=x10(n)

x11(n)=((0.5).^(n+N));

y5(n)=x11(n)

x12(n)=sin((pi)./4 \*(n+N));

y6(n)=x12(n)

if(y2(n) == x2(n))

disp('the given x2 is periodic');

end

if(y2(n) ~= x2(n))

disp('the given x2 is not periodic');

end

if (y3(n)==x3(n))

disp('the given signal x3 is peridic');

end

if (y3(n)~=x3(n))

disp('the given signal x3 is aperidic');

end

if (y4(n)==x4(n))

disp('the given signal x4 is peridic');

end

if(y4(n) ~= x4(n))

disp('the given x4 is not periodic');

end

if (y5(n)==x5(n))

disp('the given signal x5 is peridic');

end

if (y5(n)~=x5(n))

disp('the given signal x5 is peridic');

end

if (y6(n)==x6(n))

disp('the given signal x6 is peridic');

end

if(y6(n) ~= x6(n))

disp('the given x6is not periodic');

end

98) %To check periodicity and aperiodicity of given discrete time signal

N1=input('enter the no of array values');

n=1:N1;

w=2\*pi\*N1;

x(n)=sin(w\*n);

x1(n)=sin(w\*(n+N1));

y(n)=x1(n);

if (y(n)==x(n))

disp('the given signal is periodic');

end

if(y(n)~=x(n))

disp('the given x is not periodic');

end

99) %matlab program to verify the properties of systems on the setof i/ps

N=input('enter the no of array values');

N=7;

n=1:7;

x1(n)=ones(1,7);

x2(n)=n.\*x1(n);

a=2.5;

a1=1.5;a2=2.5;

y1(n)=a.\*x1(n);

y2(n)=a.\*x2(n);

z1(n)=a1.\*y1(n)+a2.\*y2(n);

z2(n)=a.\*(a1.\*x1(n)+a2.\*x2(n));

if(z1(n)==z2(n))

disp('the given system is a linear one');

end

if(z1(n)~=z2(n))

disp('the given system is an ex of nonlinear one');

end

subplot(8,1,1),plot(z1);

subplot(8,1,2),plot(z2);

subplot(8,1,3),stem(z1);

subplot(8,1,4),stem(z2);

n=2:7;

y3(n)=x1(n);

y4(n)=x2(n);

y5(n)=y1(n);

y6(n)=y2(n);

if(y5(n)==y3(n))

disp('the system is time in variant');

end

if(y5(n)~=y3(n))

disp('the system is timevariant');

end

if(y6(n)==y4(n))

disp('the system is time invariant');

end

if(y6(n)~=y4(n))

disp('the system is timevariant');

end

subplot(8,1,5),stem(y3);

subplot(8,1,6),stem(y4);

subplot(8,1,7),stem(y5);

subplot(8,1,8),stem(y6);

n=1:7;

y7(n)=3.\*x1(n);

y8(n)=3.\*x2(n);

if(y7(n)< inf)

disp('the system is a stable one');

end

if(y8(n)<inf)

disp('the system is stable one');

end

100) %matlab program to verify the properties of systems on the setof i/ps

N=input('enter the no of array values');

N=7;

n=1:7;

x1(n)=exp(j\*n);

x2(n)=exp(-j\*n);

a=2.5;

a1=1.5;a2=2.5;

y1(n)=a.\*x1(n);

y2(n)=a.\*x2(n);

z1(n)=a1.\*y1(n)+a2.\*y2(n);

z2(n)=a.\*(a1.\*x1(n)+a2.\*x2(n));

if(z1(n)==z2(n))

disp('the given system is a linear one');

end

if(z1(n)~=z2(n))

disp('the given system is an ex of nonlinear one');

end

subplot(8,1,1),plot(z1);

subplot(8,1,2),plot(z2);

subplot(8,1,3),stem(z1);

subplot(8,1,4),stem(z2);

n=2:7;

y3(n)=x1(n);

y4(n)=x2(n);

y5(n)=y1(n);

y6(n)=y2(n);

if(y5(n)==y3(n))

disp('the system is time in variant');

end

if(y5(n)~=y3(n))

disp('the system is timevariant');

end

if(y6(n)==y4(n))

disp('the system is time invariant');

end

if(y6(n)~=y4(n))

disp('the system is timevariant');

end

subplot(8,1,5),stem(y3);

subplot(8,1,6),stem(y4);

subplot(8,1,7),stem(y5);

subplot(8,1,8),stem(y6);

n=1:7;

y7(n)=3.\*x1(n);

y8(n)=3.\*x2(n);

if(y7(n)< inf)

disp('the system is a stable one');

end

if(y8(n)<inf)

disp('the system is stable one');

end