```
%ADM
%Signal Parameters
fs = 1000;
T = 1/fs;
t = 0:T:1;
fc = 10;
Ac = 1;
m = sawtooth(2*pi*fc*t);

%ADM parameters
L = 2;
delta = 0.5;
xhat(1) = 0;
n = length(m);

%ADM loop
for i=2:n
   e(i) = m(i) - xhat(i-1);
   if e(i)>=0
      d(i) = L;
   else
      d(i) = -L;
   end
   xhat(i) = xhat(i-1)+delta*d(i);
end

%plotting
subplot(3,1,1)
```

```
plot(t,m)
title('Original Signal')
xlabel('Time')
ylabel('Amplitude')
subplot(3,1,2)
stairs(t,xhat)
title('Quantized Signal')
xlabel('Time')
ylabel('Amplitude')
subplot(3,1,3)
plot(t,e)
title('Error Signal')
xlabel('Time')
ylabel('Amplitude')

clear all
%Companding law
%signal parameter
fs = 260;
T = 1/fs;
t = 0:T:1;
f = 100;
A = 1;
x = A*sin(2*pi*f*t);

%uLaw companding
u = 255;
y_u = sign(x).*log(1+u*abs(x))/log(1+u);
x_u = sign(y_u).*((1+u).^abs(y_u)-1)/u;
```

```
%Alaw companding
Amax = 1;
A1 = 87.6;
A2 = 1/(1-A1);
y_A = sign(x).*log(1+A1*abs(x)/Amax)/log(1+A1);
x_A = sign(y_A).*((1+A1).^abs(y_A)-1)/(Amax*A2);

%plotting
plot(t,x)
title('Original Signal')
plot(t,y_u)
title('u-Law Companded')
plot(t,x_u)
title('u-Law Expanded')
plot(t,y_A)
title('A-Law Companded')
plot(t,x_A)
title('A-Law Expanded')

clear all
%Define the pulse shape
p_width = 10;
p_amplitude = 1;
p_shape = p_amplitude*ones(1,p_width);

%Define the signal to be transmitted
signal_length = 100;
signal = randi([0 1], 1, signal_length);
```

```
signal(signal==0) = -1;
txSig = kron(signal, p_shape);

%Add noise to the signal
noise_power = 0.01;
noise = sqrt(noise_power)*randn(size(txSig));
rxSig = txSig + noise;

%Define the matched filter
m_filter = fliplr(p_shape);

%Apply the matched filter to the received signal
filtered_signal = conv(rxSig, m_filter, 'same');

%Plot
subplot(3,1,1)
plot(txSig)
title('Tx Signal')
subplot(3,1,2)
plot(rxSig)
title('Rx Signal')
subplot(3,1,3)
plot(filtered_signal)
title('Filtered Signal')

%%%

%haffman code
clc;
```

```
p=input('enter the probabilities:');
n= length(p);
symbols=[1:n];
[dict,avglen]=huffmandict(symbols,p);
temp=dict;
t=dict;
t=dict(:,2);
for i=1:length(temp)
   temp{i,2}=num2str(temp{i,2});
end
disp('the huffman code dict:');
disp(temp)

fprintf('enter the symbols between 1 to %d in[]',n);
sym=input(':')
encod=huffmanenco(sym,dict);
disp('the encoded output:');
disp(encod);
bits=input('enter the bit stream in[];');
decod=huffmandeco(bits,dict);
disp(decod);



H=0;
Z=0;
for(k=1:n)
   H=H+(p(k)*log2(1/p(k)));
end
fprintf(1,'entropy is %f bits',H);
```

```
N=H/avglen;

fprintf('\n efficency is:%f',N);
```

```
clear all

% Define the signal parameters

N = 100; % Number of symbols

M = 16; % Modulation order

pilot_idx = 10; % Index of the pilot symbol

pilot_value = 1 + 1i; % Value of the pilot symbol

SNR = 20; % Signal-to-Noise Ratio (dB)


% Generate the random data symbols

data = randi([0 M-1], 1, N);


% Modulate the data symbols

mod_data = qammod(data, M);


% Insert the pilot symbol

mod_data(pilot_idx) = pilot_value;


% Generate the noise

sigma2 = 10^(-SNR/10); % Noise variance

noise = sqrt(sigma2/2)*(randn(1,N)+1i*randn(1,N));


% Add the noise to the modulated data

rx_data = mod_data + noise;
```

```
% Estimate the channel using the pilot symbol
H_est = rx_data(pilot_idx)/pilot_value;

% Equalize the received data
eq_data = rx_data/H_est;

% Demodulate the received data
demod_data = qamdemod(eq_data, M);

% Calculate the entropy of the data symbols
p = histcounts(demod_data, 0:M-1, 'Normalization', 'probability');
entropy = -sum(p.*log2(p));

% Plot the results
subplot(3,1,1);
stem(real(mod_data));
hold on;
stem(real(rx_data));
title('Real part of modulated and received data');
legend('Modulated data', 'Received data');
xlabel('Symbol index');
ylabel('Amplitude');

subplot(3,1,2);
stem(imag(mod_data));
hold on;
stem(imag(rx_data));
title('Imaginary part of modulated and received data');
```

```
legend('Modulated data', 'Received data');
xlabel('Symbol index');
ylabel('Amplitude');

subplot(3,1,3);
stem(data);
hold on;
stem(demod_data);
title('Original and demodulated data');
legend('Original data', 'Demodulated data');
xlabel('Symbol index');
ylabel('Data symbol');

fprintf('Entropy of the data: %.4f bits/symbol\n', entropy);
```

```
% Define the input voltage range

Vin = linspace(0, 1, 1000);


% Define the mu values to plot

mu_values = [1, 10, 50, 100, 255];


% Loop over the µ values and calculate the output voltage for each one

for i = 1:length(mu_values)

   mu = mu_values(i);


   Vout = sign(Vin) .* log(1 + mu*abs(Vin)) ./ log(1 + mu);


   % Plot the output voltage vs input voltage for the current mu value

   plot(Vin, Vout);

   hold on;

end


% Add axis labels and a title

xlabel('Input Voltage (V)');

ylabel('Output Voltage (V)');

title('µ-Law Companding Curve');
```

```
% Add a legend

legend('µ=1', 'µ=10', 'µ=50', 'µ=100', 'µ=255',"Location","best");
```

```
   %Define the pulse shape

p_width = 10;

p_amplitude = 1;

p_shape = p_amplitude*ones(1,p_width);


%Define the signal to be transmitted

signal_length = 100;

signal = randi([0 1], 1, signal_length);

signal(signal==0) = -1;

txSig = kron(signal, p_shape);


%Add noise to the signal

noise_power = 0.01;

noise = sqrt(noise_power)*randn(size(txSig));

rxSig = txSig + noise;
```

```
%Define the matched filter
m_filter = fliplr(p_shape);

%Apply the matched filter to the received signal
filtered_signal = conv(rxSig, m_filter, 'same');

%Plot
subplot(3,1,1)
plot(txSig)
title('Tx Signal')
subplot(3,1,2)
plot(rxSig)
title('Rx Signal')
subplot(3,1,3)
plot(filtered_signal)
title('Filtered Signal')
```

```
clear all
N_bits = 1000; % number of bits to transmit
EbNo_dB = 0:0.25:10; % SNR values in dB
Pe = zeros(size(EbNo_dB)); % initialize bit error rate vector
```

```
bpskModulator = comm.BPSKModulator;
bpskDemodulator = comm.BPSKDemodulator;
channel = comm.AWGNChannel('NoiseMethod', 'Signal to noise ratio (Eb/No)', 'EbNo', 0);



txData = randi([0 1], N_bits, 1); %Generate Data
txSig = bpskModulator(txData); %Modulate data
for i = 1:length(EbNo_dB)
    channel.EbNo = EbNo_dB(i);
    rxSig = channel(txSig);          % Pass through AWGN
    rxData = bpskDemodulator(rxSig);      % Demodulate
    err = comm.ErrorRate;
    x = err(txData, rxData);
    Pe(i) = x(1);
    %PE(i) = mean(abs(txData - rxData));
end

scatterplot(rxSig)
%display(Pe)
%display(PE)
%Pe = Pe./N_bits
EbNo = 10.^(EbNo_dB/10);
Pe_theoretical = (0.5)*erfc(sqrt(EbNo));
semilogy(EbNo_dB, Pe, 'bo', EbNo_dB, Pe_theoretical, 'r-')
%semilogy(EbNo_dB, PE, 'bo', EbNo_dB, Pe_theoretical, 'r-')
title ('Probabolity of Error vs Eb/No');
xlabel ('Eb/No (dB)');
ylabel ('Probability of Error');
legend('Practical curve','Theoratical curve');
```

```
grid on;
```

Hoffman

```
%Binary Huffman Code
symbols = (1:5); % Alphabet vector
prob = [.3 .3 .2 .1 .1]; % Symbol probability vector
[dict,avglen] = huffmandict(symbols,prob)
samplecode = dict{5,2} % Codeword for fifth signal value

clc;
clear all;
s=input('Enter symbols- ')     %format ['a','b','c','d','e','f'];
p=input('Enter value of probabilty- ')  %format [0.22,0.20,0.18,0.15,0.13,0.12];

if length(s)~=length(p)
   error('Wrong entry.. enter again- ')
end

i=1;
for m=1:length(p)
   for n=1:length(p)
       if(p(m)>p(n))
         a=p(n);  a1=s(n);
         p(n)=p(m);s(n)=s(m);
```

```
        p(m)=a;   s(m)=a1;
      end
  end
end
display(p) %arranged prob. in descending order.

tempfinal=[0];
sumarray=[];
w=length(p);
lengthp=[w];
b(i,:)=p;

while(length(p)>2)
tempsum=p(length(p))+p(length(p)-1);
sumarray=[sumarray,tempsum];
p=[p(1:length(p)-2),tempsum];
p=sort(p,'descend');
i=i+1;
b(i,:)=[p,zeros(1,w-length(p))];
w1=0;
lengthp=[lengthp,length(p)];

for temp=1:length(p)
    if p(temp)==tempsum;
      w1=temp;
    end
end
tempfinal=[w1,tempfinal];  % Find the place where tempsum has been inserted
display(p);
```

```
end

sizeb(1:2)=size(b);
tempdisplay=0;
temp2=[];

for i= 1:sizeb(2)
  temp2=[temp2,b(1,i)];
end
sumarray=[0,sumarray];
  var=[];
  e=1;
for ifinal= 1:sizeb(2)
  code=[s(ifinal),'      ']
for j=1:sizeb(1)
     tempdisplay=0;

for i1=1:sizeb(2)
    if( b(j,i1)==temp2(e))
    tempdisplay=b(j,i1);
  end
   if(tempdisplay==0 & b(j,i1)==sumarray(j))
           tempdisplay=b(j,i1);
  end
end
    var=[var,tempdisplay];
if tempdisplay==b(j,lengthp(j))       %assign 0 & 1
      code=[code,'1'];
elseif tempdisplay==b(j,lengthp(j)-1)
```

```
      code=[code,'0'];
else
      code=[code,''];
   end
      temp2(e)=tempdisplay;
  end
   display(code) %display final codeword
    e=e+1;
end
```