Hamming code in AWGN channels

Step 1

For every symbol in 4-PAM, 2 bits are required, and for every 4 bits of signal, 3 parity bits are added for (7,4) Hamming code.

For a BPSK (uncoded) signal, 1 bit per symbol is required. And for every 4 bits (4 symbols), 3 parity bits are added for (7,4) Hamming code.

Let C1, C2, C3 and C4 be the symbol bits.

$C1 \oplus C2 \oplus C3 \oplus C5 = 0$

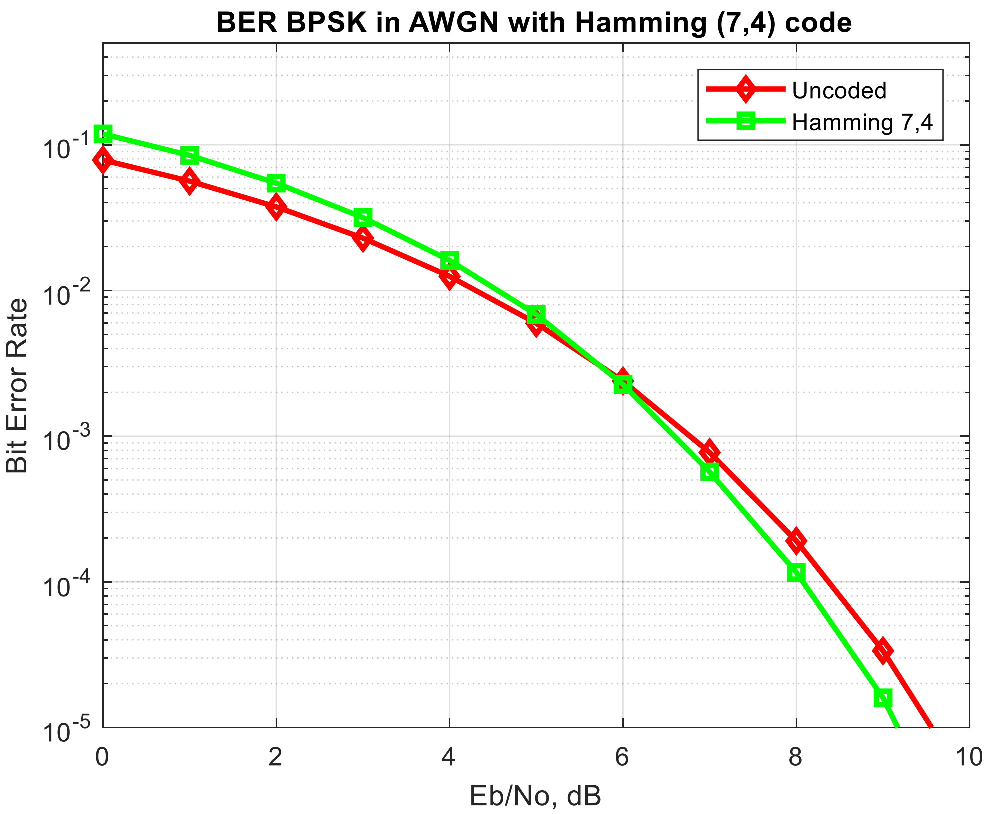
$C1 \oplus C3 \oplus C4 \oplus C6 = 0$

$C1 \oplus C2 \oplus C4 \oplus C7 = 0$

Hamming decoding table

0	0	0	0	0	0	0
1	0	0	0	1	1	1
0	1	0	0	1	0	1
0	0	1	0	1	1	0
0	0	0	1	0	1	1
1	1	0	0	0	1	0
1	0	1	0	0	0	1
1	0	0	1	1	0	0
0	1	1	0	0	1	1
0	1	0	1	1	1	0
0	0	1	1	1	0	1
1	1	1	0	1	0	0
1	1	0	1	0	0	1
1	0	1	1	0	1	0
0	1	1	1	0	0	0
1	1	1	1	1	1	1

Step2

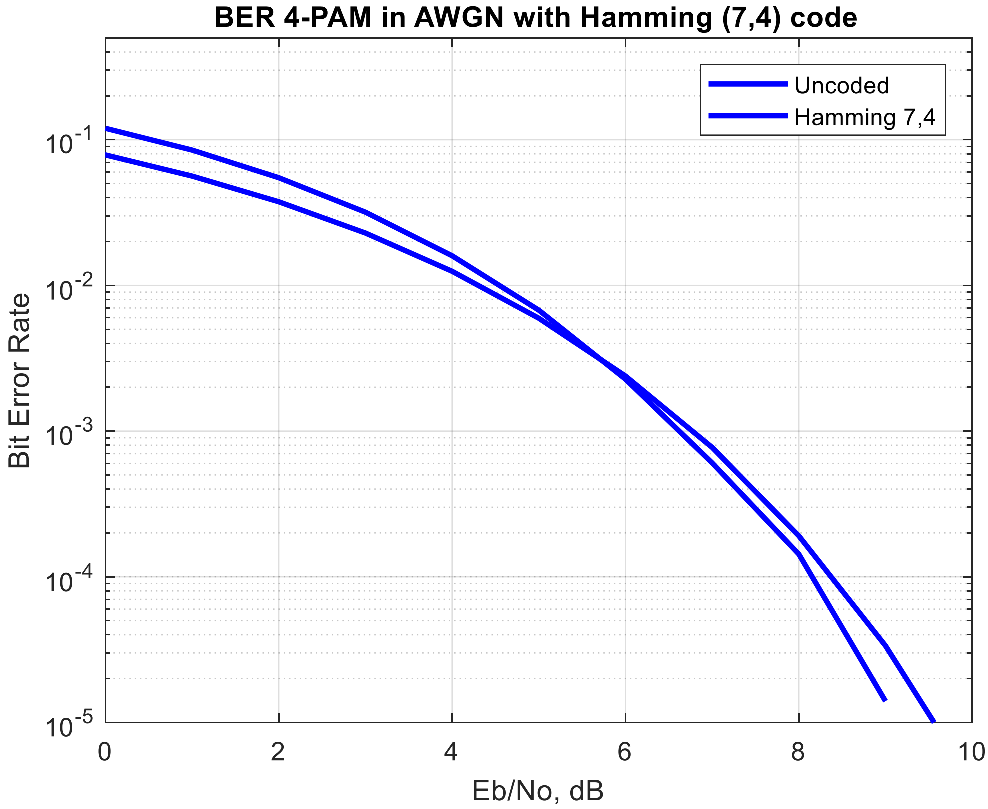


MATLAB CODE

```
clear

N = 10^6 ;% number of bits


EbN0dB = [0:1:10]; % multiple Eb/N0 values

EcN0dB = EbN0dB - 10*log10(7/4);


trans_f = [ 1   0   1 ;    1   1   1;    1   1   0;    0   1   1];

trans_f_t = [trans_f ;eye(3)];

g = [eye(4) trans_f];

synRef = [ 5 7 6 3  ];

T_bits = [ 7 7 4 7 1 3 2].';
```

```
for yy = 1:length(EbN0dB)

  % Transmitter
  gen_bits = rand(1,N)>0.5; % generating 0,1 with equal probability

  % Hamming coding (7,4)
  gen_bits_M = reshape(gen_bits,4,N/4).';
  gen_bits_C = mod(gen_bits_M*g,2);
  c_gen_bits = reshape(gen_bits_C.',1,N/4*7);

  % Modulation
  s = 2*c_gen_bits-1; % BPSK modulation 0 -> -1; 1 -> 0

  % Channel - AWGN
  noise_n = 1/sqrt(2)*[randn(size(c_gen_bits)) + j*randn(size(c_gen_bits))];

  % Noise addition
  y = s + 10^(-EcN0dB(yy)/20)*noise_n; % additive white gaussian noise

  % Receiver
  cipHard = real(y)>0; % hard decision

  % Hamming decoder
  hard_gen_bits_M_C    = reshape(cipHard,7,N/4).';
  syndrome_bits    = mod(hard_gen_bits_M_C*trans_f_t,2); % find the syndrome_bits
  syndromeDec_bits = sum(syndrome_bits.*kron(ones(N/4,1),[4 2 1]),2); % converting the three bit
syndrom to decimal
  syndromeDec_bits(find(syndromeDec_bits==0)) = 1;
  Correl_bits  = T_bits(syndromeDec_bits); % find the bits to correct
  Correl_bits  = Correl_bits + [0:N/4-1].'*7; % finding the index in the array
```

```
  cipHard(Correl_bits) = ~cipHard(Correl_bits); % correcting bits
  idx = kron(ones(1,N/4),[1:4]) + kron([0:N/4-1]*7,ones(1,4)); % index of data bits
  gen_bits_PHat = cipHard(idx); % selecting data bits

  % counting the errors
  N_err(yy) = size(find([gen_bits- gen_bits_PHat]),2);

end



t_BER = 0.5*erfc(sqrt(10.^(EbN0dB/10))); % theoretical BER
sim_BER    = N_err/N;

close all
figure
semilogy(EbN0dB,t_BER,'rd-','LineWidth',2);
hold on
semilogy(EbN0dB,sim_BER,'gs-','LineWidth',2);
axis([0 10 10^-5 0.5])
grid on
legend('Uncoded', 'Hamming 7,4');
xlabel('Eb/No, dB');
ylabel('Bit Error Rate');
title('BER BPSK in AWGN with Hamming (7,4) code');
```



MATLAB CODE

```
N = 10^6 ;% number of bits


EbN0dB = [-3:1:20]; % multiple Eb/N0 values

EcN0dB = EbN0dB - 10*log10(7/4);



trans_f = [ 1   0   1 ;    1   1   1;    1   1   0;    0   1   1];

trans_f_t = [trans_f ;eye(3)];

g = [eye(4) trans_f];

synRef = [ 5 7 6 3  ];

T_bits = [ 7 7 4 7 1 3 2].';
```

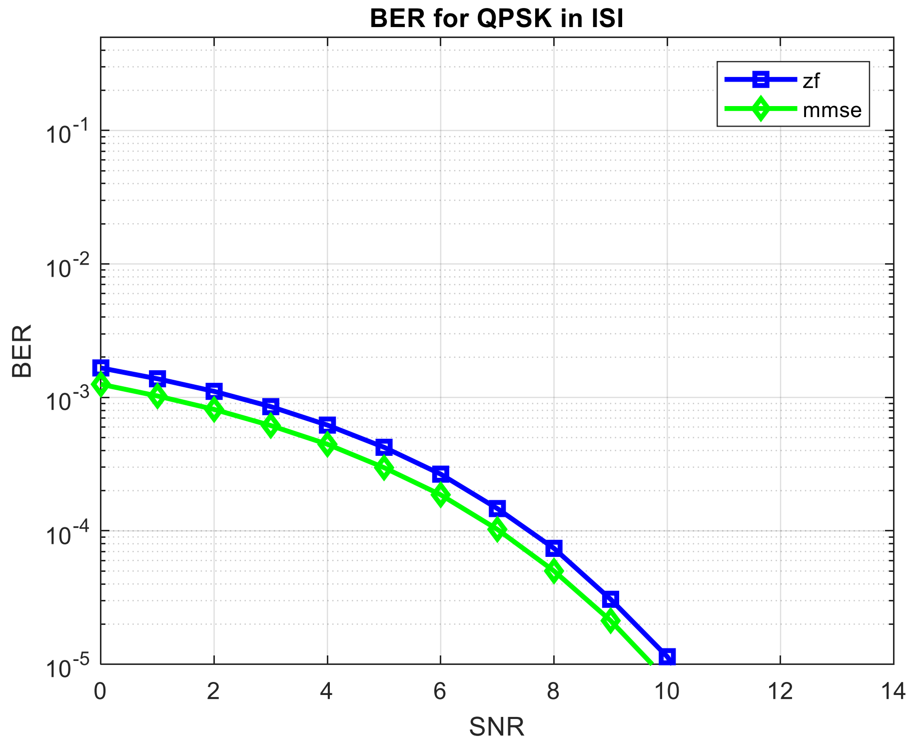
```
alpha4pam = [-3 -1 1 3]; % 4-PAM alphabets
Es_N0_dB = [-3:20]; % multiple Error_b/N0 values
ipHat = zeros(1,N);
for k_iter = 1:length(Es_N0_dB)
ip = randsrc(1,N,alpha4pam);
s = (1/sqrt(5))*ip; % normalization of energy to 1
n = 1/sqrt(2)*[randn(1,N) + j*randn(1,N)]; % white guassian noise, 0dB variance

y = s + 10^(-Es_N0_dB(k_iter)/20)*n; % additive white gaussian noise

% demodulation
r = real(y); % taking only the real part

ipHat(find(r< -2/sqrt(5))) = -3;
ipHat(find(r>= 2/sqrt(5))) = 3;
ipHat(find(r>=-2/sqrt(5) & r<0)) = -1;
ipHat(find(r>=0 & r<2/sqrt(5))) = 1;

n_err(k_iter) = size(find([ip- ipHat]),2); % couting the number of errors
end




for yy = 1:length(EbN0dB)

  % Transmitter
  gen_bits = rand(1,N)>0.5;
  % Hamming coding (7,4)
  gen_bits_M = reshape(gen_bits,4,N/4).';
  gen_bits_C = mod(gen_bits_M*g,2);
```

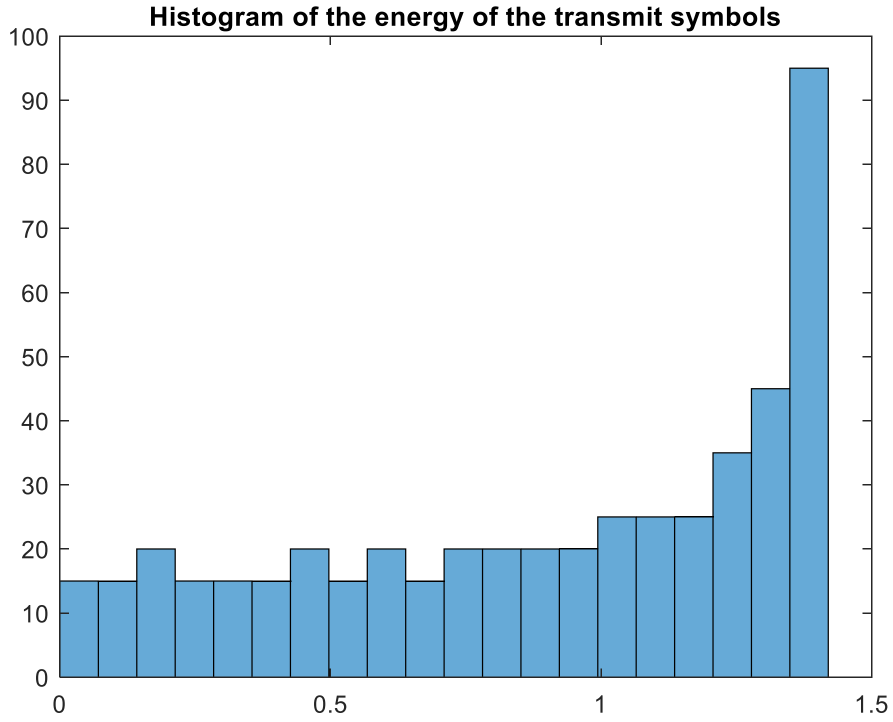
```
  c_gen_bits = reshape(gen_bits_C.',1,N/4*7);


  % Modulation

  s = 2*c_gen_bits-1; % BPSK modulation 0 -> -1; 1 -> 0


  % Channel - AWGN

  noise_n = 1/sqrt(2)*[randn(size(c_gen_bits)) + j*randn(size(c_gen_bits))];


  % Noise addition

  y = s + 10^(-EcN0dB(yy)/20)*noise_n; % additive white gaussian noise


  % Receiver

  cipHard = real(y)>0; % hard decision


  % Hamming decoder

  hard_gen_bits_M_C    = reshape(cipHard,7,N/4).';

  syndrome_bits    = mod(hard_gen_bits_M_C*trans_f_t,2); % find the syndrome_bits

  syndromeDec_bits = sum(syndrome_bits.*kron(ones(N/4,1),[4 2 1]),2); % converting the three bit
syndrom to decimal

  syndromeDec_bits(find(syndromeDec_bits==0)) = 1;

  Correl_bits  = T_bits(syndromeDec_bits); % find the bits to correct

  Correl_bits  = Correl_bits + [0:N/4-1].'*7; % finding the index in the array

  cipHard(Correl_bits) = ~cipHard(Correl_bits); % correcting bits

  idx = kron(ones(1,N/4),[1:4]) + kron([0:N/4-1]*7,ones(1,4)); % index of data bits

  gen_bits_PHat = cipHard(idx); % selecting data bits


  % counting the errors

  N_err(yy) = size(find([gen_bits- gen_bits_PHat]),2);


end
```

```
t_BER = 0.5*erfc(sqrt(10.^(EbN0dB/10))); % theoretical BER
sim_BER    = N_err/N;

close all
figure
semilogy(EbN0dB,t_BER,'b-','LineWidth',2);
hold on
semilogy(EbN0dB,sim_BER,'b-','LineWidth',2);
axis([0 10 10^-5 0.5])
grid on
legend('Uncoded', 'Hamming 7,4');
xlabel('Eb/No, dB');
ylabel('Bit Error Rate');
title('BER 4-PAM in AWGN with Hamming (7,4) code');
```

QPSK in ISI Channels

The resilience of OFDM systems to frequency-selective fading can be attributed to the cyclic prefix inserted between symbols that allows decomposition of the channel into independent subchannels by use of the fast Fourier transform (FFT). However, a consequence of this “frame” structure of an OFDM symbol is that it becomes important for the receiver to identify the beginning of each new symbol. This is the problem of symbol synchronization.





MATLAB CODE

```
%FOR QPSK
clear
N  = 10^6; % number of bits or symbols
Eb_N0_dB = [0:15]; % multiple Error_b/N0 values
K = 3;

for k_iter = 1:length(Eb_N0_dB)

  % Transmitter
  ip = rand(1,N)>0.5; % generating 1,-1,3,-3 with equal probability
  s = 2*ip-1; % QPSK

  % Channel model, multipath channel
  TAP = 9;
```

```
h_transfer = [0.2 0.9 0.3];

L  = length(h_transfer);


chanOut = conv(s,h_transfer);

n = 1/sqrt(2)*[randn(1,N+length(h_transfer)-1) + j*randn(1,N+length(h_transfer)-1)]; % white gaussian noise, 0dB variance


% Noise addition

y = chanOut + 10^(-Eb_N0_dB(k_iter)/20)*n; % additive white gaussian noise



 %%

 % IFFT


 ifft_sig=ifft(y,16);




% zero forcing equalization

hM = toeplitz([h_transfer([2:end]) zeros(1,2*K+1-L+1)], [ h_transfer([2:-1:1]) zeros(1,2*K+1-L+1) ]);

d  = zeros(1,2*K+1);

d(K+1) = 1;

c_zf  = [inv(hM)*d.'].';

yFilt_zf = conv(y,c_zf);

yFilt_zf = yFilt_zf(K+2:end);

yFilt_zf = conv(yFilt_zf,ones(1,1)); % convolution

ySamp_zf = yFilt_zf(1:1:N);  % sampling at time T


% mmse equalization

hAutoCorr = conv(h_transfer,fliplr(h_transfer));

hM = toeplitz([hAutoCorr([3:end]) zeros(1,2*K+1-L)], [ hAutoCorr([3:end]) zeros(1,2*K+1-L) ]);
```

```
  hM = hM + 1/2*10^(-Eb_N0_dB(k_iter)/10)*eye(2*K+1);
  d  = zeros(1,2*K+1);
  d([-1:1]+K+1) = fliplr(h_transfer);
  c_mmse  = [inv(hM)*d.'].';
  yFilt_mmse = conv(y,c_mmse);
  yFilt_mmse = yFilt_mmse(K+2:end);
  yFilt_mmse = conv(yFilt_mmse,ones(1,1)); % convolution
  ySamp_mmse = yFilt_mmse(1:1:N);  % sampling at time T

  % receiver - hard decision decoding
  ipHat_zf = real(ySamp_zf)>0;
  ipHat_mmse = real(ySamp_mmse)>0;


   %%
   % FFT

   ff_sig=fft(yFilt_mmse,16);

  % counting the errors
  n_err_zf(1,k_iter) = size(find([ip- ipHat_zf]),2);
  n_err_MMSE(1,k_iter) = size(find([ip- ipHat_mmse]),2);
end

simBer_zf = 1e-2.*n_err_zf/N; % simulated ber
simBer_mmse = 1e-2.*n_err_MMSE/N; % simulated ber
theoryBer = 0.5*erfc(sqrt(10.^(Eb_N0_dB/10))); % theoretical ber

% plot
close all
figure
```

```
semilogy(Eb_N0_dB,simBer_zf(1,:),'bs-','Linewidth',2);

hold on

semilogy(Eb_N0_dB,simBer_mmse(1,:),'gd-','Linewidth',2);

axis([0 14 10^-5 0.5])

grid on

legend('zf', 'mmse');

xlabel('SNR');

ylabel('BER');

title('BER for QPSK in ISI');




% Energy histogram

data=[0  1 0 1 1 1 0 0 1 1]; % information


data_NZR=2*data-1; % Data Represented at NZR form for QPSK modulation

s_p_data=reshape(data_NZR,2,length(data)/2);  % S/P convertion of data


br=10.^6; %Let us transmission bit rate  1000000

f=br; % minimum carrier frequency

T=1/br; % bit duration

t=T/99:T/99:T; % Time vector for one bit information



y=[];

y_in=[];

y_qd=[];

for(i=1:length(data)/2)

   y1=s_p_data(1,i)*cos(2*pi*f*t); % inphase component

   y2=s_p_data(2,i)*sin(2*pi*f*t) ;% Quadrature component

   y_in=[y_in y1]; % inphase signal vector
```

```
   y_qd=[y_qd y2]; %quadrature signal vector
   y=[y y1+y2]; % modulated signal vector
end
y_energy = abs(y);
figure;
histogram(y_energy,20)

title('Histogram of the energy of the transmit symbols')
```