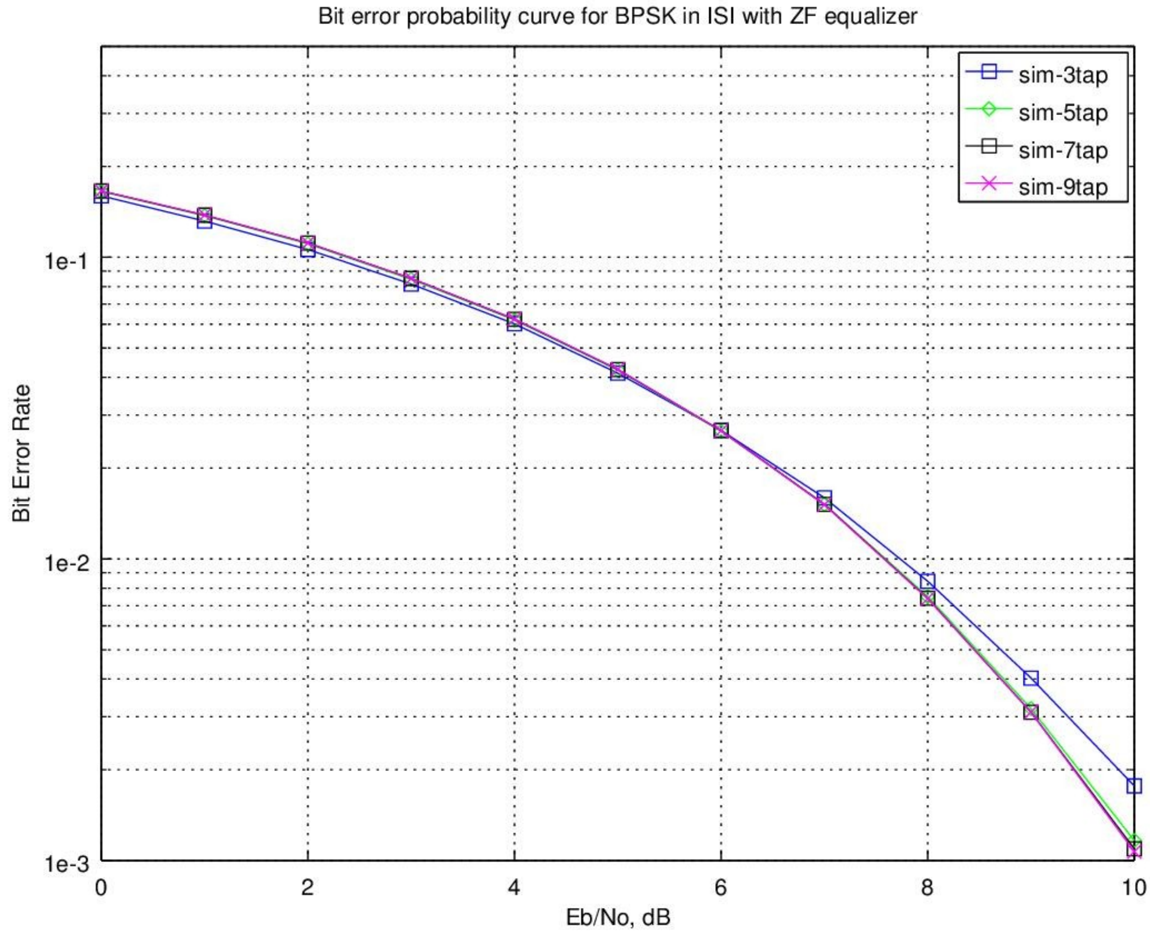
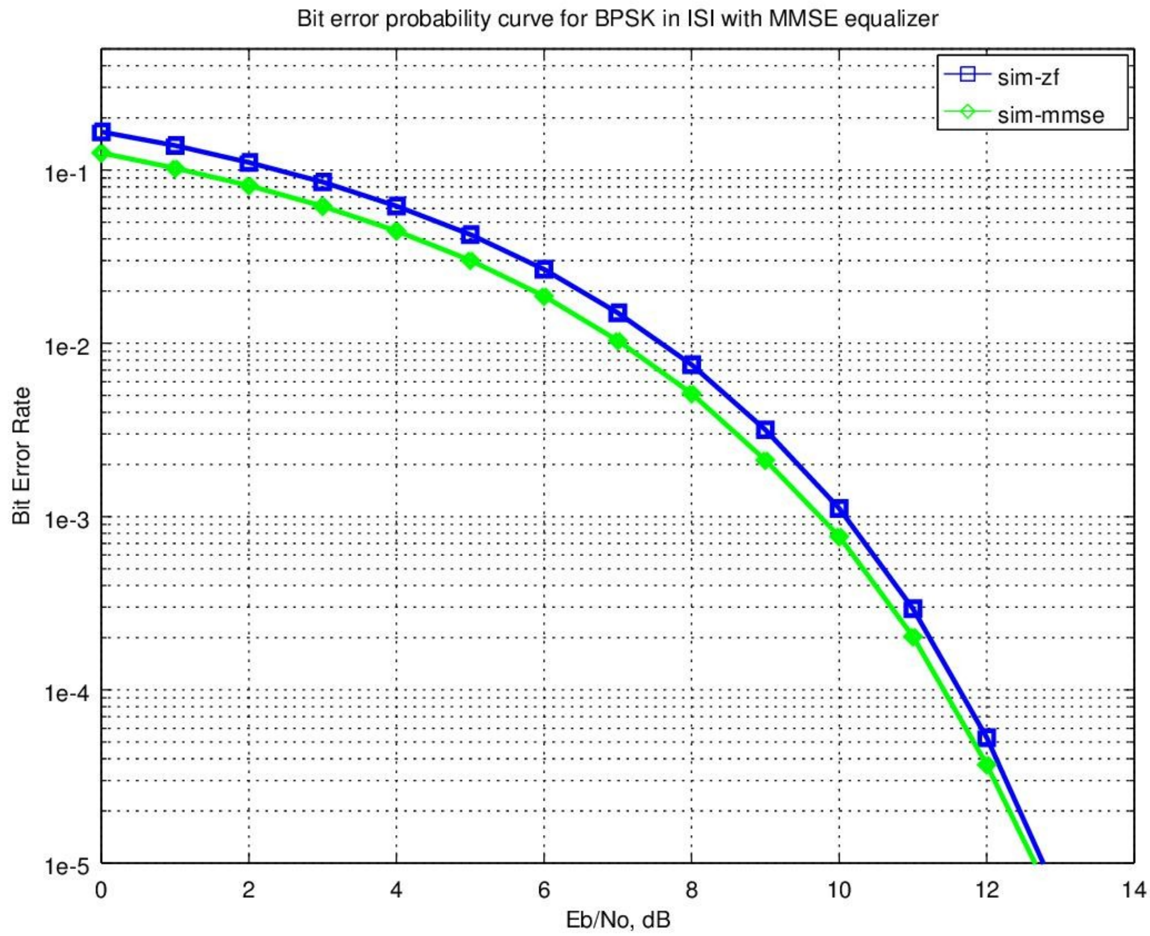
Name: Wan Ahmad Zainie bin Wan Mohamad
ID: ME131135
MET1413 – Assignment #4 – 2013/14 Semester 2

(a) and (b) Run the ZF equalization and MMSE equalization program.





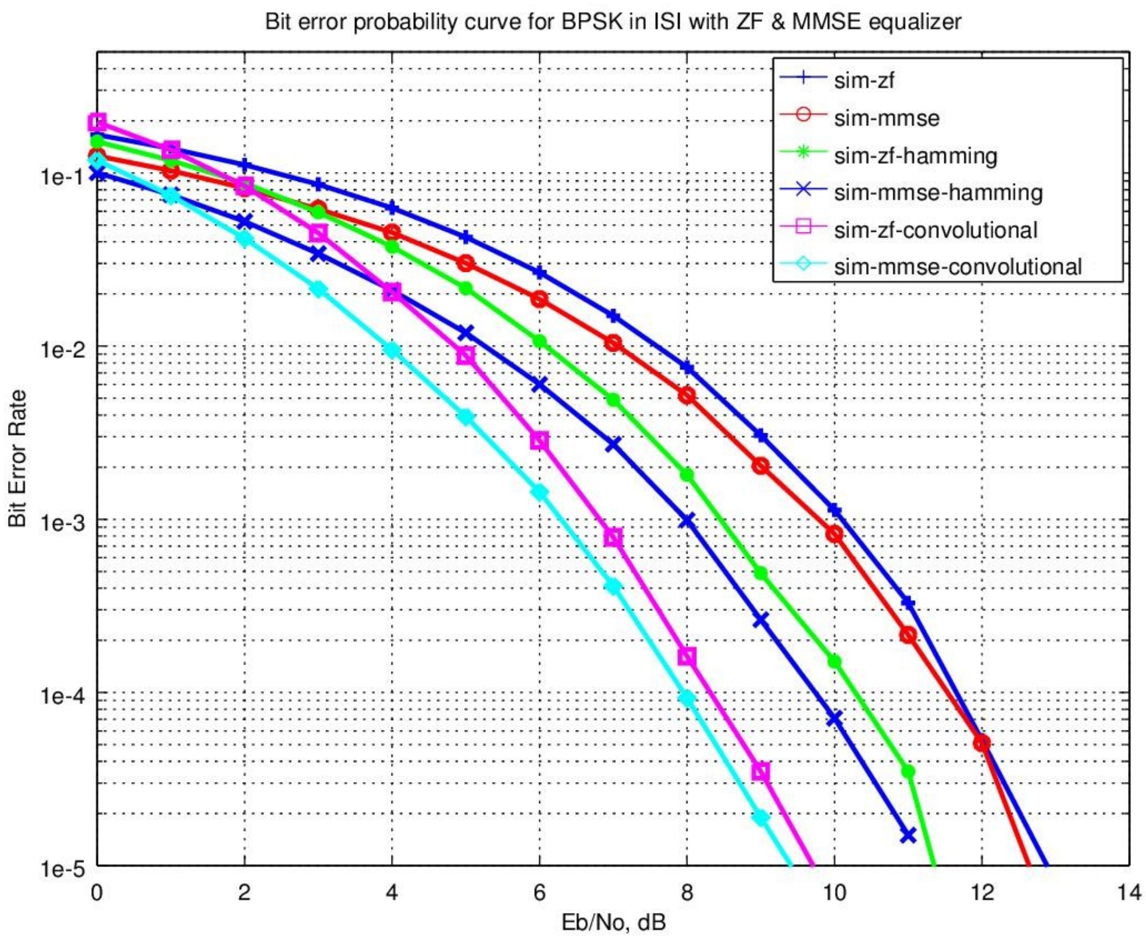
(c) Add channel coding (block coding and convolutional code) to the program (modifications are required).

The modified script's source code is attached at the end of the document. All the scripts are executed in GNU Octave 3.8.1.

The modified script perform the following:

1. Generate random binary sequence of 0's and 1's.
2. (a) without channel coding, skip to 3.
   (b) with channel coding, [ADDED]
      → encode the bits using Hamming (7,4) encoder, OR
      → convolutionally encode the bits using rate ½, generator polynomial [7,5] octal code.
3. BPSK modulation; bit 0 represented as -1 and bit 1 represented as +1.
4. Convolving the symbols with a 3-tap fixed fading channel.
5. Passing them through Additive White Gaussian Noise channel.
6. Computing the the ZF and MMSE equalization filter at the receiver (with 7 taps in length).
7. Demodulation and conversion to bits.
8. (a) without channel coding, skip to 9.
   (b) with channel coding, [ADDED]
      → decode the coded bits using Hamming (7,4) decoder, OR
      → decode the coded bits using Viterbi decoder.
9. Counting the number of bit errors.
10. Repeating for multiple values of Eb/N0, and plot the simulation result.

(d) Determine the effects of the channel coding applied.



Using channel coding improves the performance of the communication system tremendously.

However, there are not much difference in the bit error rate at lower Eb/No, in the region of less than 3 dB. At the lower Eb/No value, there are more chances of multiple received coded bits in errors, and the Hamming coding (in the case of block code) or Viterbi algorithm (in the case of convolutional code) is unable to recover.

(e) Discuss critically on the developed communication system.

Coding is a technique where redundancy is added to the original bit sequence to increase the reliability of the communication. There are 2 (two) types of channel coding:- 1) block coding, and 2) convolutional coding.

A block coding is a scheme where a group of *k* information bits is mapped into *n* coded bits. Such codes are referred to as (*n*,*k*) codes. In this assignment, Hamming (7,4) codes is used, where 4 information bits are mapped into 7 coded bits. The usage of built-in function *encode* and *decode* hides the details of the implementation of the Hamming coding.

And to demonstrate the effect of convolutional code, the system that consist of a simple binary convolutional coding scheme at the transmitter and the associated Viterbi (maximum likelihood) decoding scheme at the receiver is added.

There are three parameters which define the convolutional code:-

1. Rate: Ratio of the number of input bits to the number of output bits. In this assignment, the rate is ½ which means there are 2 (two) output bits for each input bit.
2. Constraint length: The number of delay elements in the convolutional coding. In this assignment, with K = 3 there are two delay elements.
3. Generator polynomial: Wiring of the input sequence with the delay elements to form the output. In this assignment, generator polynomial is $[7,5]_8 = [\ 1\ 1\ 1,\ 1\ 0\ 1\ ]_2$. The output from the $7_8 = 111_2$ arm uses the XOR of the current input, previous input and the previous to previous input. The output from the $5_8 = 101_2$ arm uses the XOR of the current input and the previous to previous input.

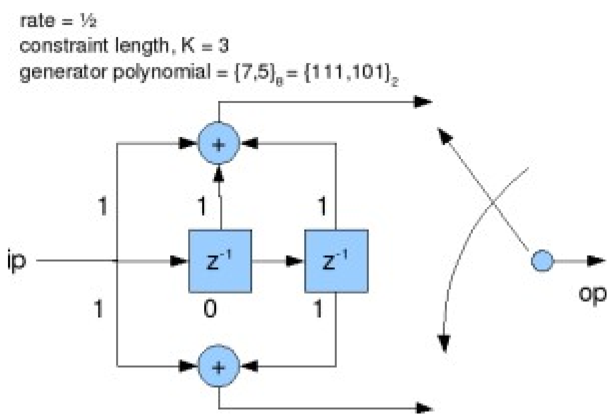


Figure 1: Convolutional code with Rate ½, K=3, Generator Polynomial [7,5] octal.

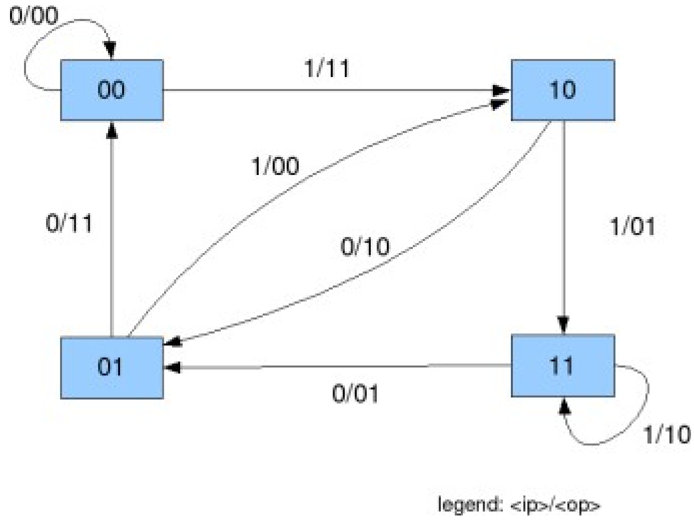


Figure 2: State transition for K=3, rate=½ convolutional code

Decoding the convolutional code using Viterbi algorithm is quite complex. Some of the key assumptions made to reduce the complexity to a more manageable levels are:-

1. Any state can be reached from only 2 (two) possible previous state.
2. Out of the 2 (two) possible states, only 1 (one) of the transition is valid.
3. The errors in the received coded sequence are randomly distributed and the probability of error is small.

Based on the above assumptions, the decoding scheme proceed as follows:-
→ Assume that there are *N* coded bits.
→ Take two coded bits at a time for processing and compute Hamming distance, Branch Metric, Path Metric and Survivor Path Index for $N/2 + K - 1$ times.
→ Let i be the index varying from 1 till $N/2 + K - 1$.

The rest of the developed communication system is same as Assignment #2.

REFERENCES:-

1. http://www.dsplog.com/2009/11/29/ber-bpsk-isi-channel-zero-forcing-equalization/
2. http://www.dsplog.com/2010/01/24/ber-bpsk-isi-channel-mmse-equalization/
3. http://www.downscripts.com/channel-coding-using-hamming-codes_matlab-script.html
4. http://www.dsplog.com/2009/09/29/hamming-74-code-with-hard-decision-decoding/
5. http://www.dsplog.com/2009/01/04/convolutional-code/
6. http://www.dsplog.com/2009/01/04/viterbi/

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% All rights reserved by Krishna Sankar, http://www.dsplog.com
% The file may not be re-distributed without explicit authorization
% from Krishna Sankar.
% Checked for proper operation with Octave Version 3.0.0
% Author        : Krishna Sankar M
% Email         : krishna@dsplog.com
% Version       : 1.0
% Date          : 24 January 2010
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Script for computing the BER for BPSK modulation in 3 tap ISI
% channel. Minimum Mean Square Error (MMSE) equalization with 7 tap
% and the BER computed (and is compared with Zero Forcing equalization)

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

mH = 3; nH = 2^mH-1; kH = nH-mH; % Hamming (7,4)

ref = [0 0 ; 0 1; 1 0  ; 1 1 ];

ipLUT = [ 0   0   0   0;...
          0   0   0   0;...
          1   1   0   0;...
          0   0   1   1 ];

for ii = 1:length(Eb_N0_dB)

   % Transmitter
   ip = rand(1,N)>0.5; % generating 0,1 with equal probability
   s = 2*ip-1; % BPSK modulation 0 -> -1; 1 -> 0

   % Channel model, multipath channel
   nTap = 3;
   ht = [0.2 0.9 0.3];
   L  = length(ht);

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

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

   % Channel coding - block code
   ip_bc = encode(ip,nH,kH,'hamming/binary'); % Hamming coding
   ip_bc = reshape(ip_bc,1,size(ip_bc));
   s_bc = 2*ip_bc-1; % BPSK modulation 0 -> -1; 1 -> 0
   chanOut_bc = conv(s_bc,ht);
   n_bc = 1/sqrt(2)*[randn(1,size(ip_bc,2)+length(ht)-1) + j*randn(1,size(ip_bc,2)+length(ht)-1)]; % white gaussian noise, 0dB variance
   y_bc = chanOut_bc + 10^(-Eb_N0_dB(ii)/20)*n_bc; % additive white gaussian noise

   % Channel coding - convolutional coding, rate - 1/2, generator polynomial - [7,5] octal
   ip_cc1 = mod(conv(ip,[1 1 1 ]),2);
   ip_cc2 = mod(conv(ip,[1 0 1 ]),2);
   ip_cc = [ip_cc1;ip_cc2];
   ip_cc = ip_cc(:).';
   s_cc = 2*ip_cc-1; % BPSK modulation 0 -> -1; 1 -> 0
   chanOut_cc = conv(s_cc,ht);
   n_cc = 1/sqrt(2)*[randn(1,size(ip_cc,2)+length(ht)-1) + j*randn(1,size(ip_cc,2)+length(ht)-1)]; % white gaussian noise, 0dB variance
   y_cc = chanOut_cc + 10^(-Eb_N0_dB(ii)/20)*n_cc; % additive white gaussian noise

   % zero forcing equalization
   hM = toeplitz([ht([2:end]) zeros(1,2*K+1-L+1)], [ ht([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(ht,fliplr(ht));
    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(ii)/10)*eye(2*K+1);
    d  = zeros(1,2*K+1);
    d([-1:1]+K+1) = fliplr(ht);
    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

    % zero forcing equalization - block code
    hM = toeplitz([ht([2:end]) zeros(1,2*K+1-L+1)], [ ht([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_bc = conv(y_bc,c_zf);
    yFilt_zf_bc = yFilt_zf_bc(K+2:end);
    yFilt_zf_bc = conv(yFilt_zf_bc,ones(1,1)); % convolution
    ySamp_zf_bc = yFilt_zf_bc(1:1:size(ip_bc,2));  % sampling at time T

    % zero forcing equalization - convolutional code
    hM = toeplitz([ht([2:end]) zeros(1,2*K+1-L+1)], [ ht([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_cc = conv(y_cc,c_zf);
    yFilt_zf_cc = yFilt_zf_cc(K+2:end);
    yFilt_zf_cc = conv(yFilt_zf_cc,ones(1,1)); % convolution
    ySamp_zf_cc = yFilt_zf_cc(1:1:size(ip_cc,2));  % sampling at time T

    % mmse equalization - block code
    hAutoCorr = conv(ht,fliplr(ht));
    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(ii)/10)*eye(2*K+1);
    d  = zeros(1,2*K+1);
    d([-1:1]+K+1) = fliplr(ht);
    c_mmse  = [inv(hM)*d.'].';
    yFilt_mmse_bc = conv(y_bc,c_mmse);
    yFilt_mmse_bc = yFilt_mmse_bc(K+2:end);
    yFilt_mmse_bc = conv(yFilt_mmse_bc,ones(1,1)); % convolution
    ySamp_mmse_bc = yFilt_mmse_bc(1:1:size(ip_bc,2));  % sampling at time T

    % mmse equalization - convolutional code
    hAutoCorr = conv(ht,fliplr(ht));
    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(ii)/10)*eye(2*K+1);
    d  = zeros(1,2*K+1);
    d([-1:1]+K+1) = fliplr(ht);
    c_mmse  = [inv(hM)*d.'].';
    yFilt_mmse_cc = conv(y_cc,c_mmse);
    yFilt_mmse_cc = yFilt_mmse_cc(K+2:end);
    yFilt_mmse_cc = conv(yFilt_mmse_cc,ones(1,1)); % convolution
    ySamp_mmse_cc = yFilt_mmse_cc(1:1:size(ip_cc,2));  % sampling at time T

    % receiver - hard decision decoding
    ipHat_zf = real(ySamp_zf)>0;
    ipHat_zf_bc = real(ySamp_zf_bc)>0;
    ipHat_zf_bc = decode(ipHat_zf_bc,nH,kH,'hamming/binary');
    ipHat_zf_bc = reshape(ipHat_zf_bc,1,N);
    ipHat_mmse = real(ySamp_mmse)>0;
    ipHat_mmse_bc = real(ySamp_mmse_bc)>0;
    ipHat_mmse_bc = decode(ipHat_mmse_bc,nH,kH,'hamming/binary');
    ipHat_mmse_bc = reshape(ipHat_mmse_bc,1,N);
    ipHat_zf_cc = real(ySamp_zf_cc)>0;
    ipHat_mmse_cc = real(ySamp_mmse_cc)>0;
```

```
    for kk = 1:2
    % Viterbi decoding
    pathMetric  = zeros(4,1);  % path metric
    if (kk == 1)
    survivorPath_v_zf  = zeros(4,length(ySamp_zf_cc)/2); % survivor path
    length_y = length(ySamp_zf_cc)
    else
    survivorPath_v_mmse  = zeros(4,length(ySamp_mmse_cc)/2); % survivor path
    length_y = length(ySamp_mmse_cc)
    endif

    for iii = 1:length_y/2
       if (kk == 1)
       r = ipHat_zf_cc(2*iii-1:2*iii); % taking 2 coded bits
       else
       r = ipHat_mmse_cc(2*iii-1:2*iii); % taking 2 coded bits
       endif

       % computing the Hamming distance between ip coded sequence with [00;01;10;11]
       rv = kron(ones(4,1),r);
       hammingDist = sum(xor(rv,ref),2);

       if (iii == 1) || (iii == 2)
          % branch metric and path metric for state 0
          bm1 = pathMetric(1,1) + hammingDist(1);
          pathMetric_n(1,1)  = bm1;
          survivorPath(1,1)  = 1;

          % branch metric and path metric for state 1
          bm1 = pathMetric(3,1) + hammingDist(3);
          pathMetric_n(2,1) = bm1;
          survivorPath(2,1)  = 3;

          % branch metric and path metric for state 2
          bm1 = pathMetric(1,1) + hammingDist(4);
          pathMetric_n(3,1) = bm1;
          survivorPath(3,1)  = 1;

          % branch metric and path metric for state 3
          bm1 = pathMetric(3,1) + hammingDist(2);
          pathMetric_n(4,1) = bm1;
          survivorPath(4,1)  = 3;

       else
          % branch metric and path metric for state 0
          bm1 = pathMetric(1,1) + hammingDist(1);
          bm2 = pathMetric(2,1) + hammingDist(4);
          [pathMetric_n(1,1) idx] = min([bm1,bm2]);
          survivorPath(1,1)  = idx;

          % branch metric and path metric for state 1
          bm1 = pathMetric(3,1) + hammingDist(3);
          bm2 = pathMetric(4,1) + hammingDist(2);
          [pathMetric_n(2,1) idx] = min([bm1,bm2]);
          survivorPath(2,1)  = idx+2;

          % branch metric and path metric for state 2
          bm1 = pathMetric(1,1) + hammingDist(4);
          bm2 = pathMetric(2,1) + hammingDist(1);
          [pathMetric_n(3,1) idx] = min([bm1,bm2]);
          survivorPath(3,1)  = idx;

          % branch metric and path metric for state 3
          bm1 = pathMetric(3,1) + hammingDist(2);
          bm2 = pathMetric(4,1) + hammingDist(3);
          [pathMetric_n(4,1) idx] = min([bm1,bm2]);
          survivorPath(4,1)  = idx+2;

       end

       pathMetric = pathMetric_n;
```

```
        if (kk == 1)
        survivorPath_v_zf(:,iii) = survivorPath;
        else
        survivorPath_v_mmse(:,iii) = survivorPath;
        endif

    end
    end

    % trace back unit - ZF
    currState = 1;
    ipHat_zf_cc = zeros(1,length(ySamp_zf_cc)/2);
    for jj = length(ySamp_zf_cc)/2:-1:1
       prevState = survivorPath_v_zf(currState,jj);
       ipHat_zf_cc(jj) = ipLUT(currState,prevState);
       currState = prevState;
    end

    % trace back unit - MMSE
    currState = 1;
    ipHat_mmse_cc = zeros(1,length(ySamp_mmse_cc)/2);
    for jj = length(ySamp_mmse_cc)/2:-1:1
       prevState = survivorPath_v_mmse(currState,jj);
       ipHat_mmse_cc(jj) = ipLUT(currState,prevState);
       currState = prevState;
    end

    % counting the errors
    nErr_zf(1,ii) = size(find([ip- ipHat_zf]),2);
    nErr_zf_bc(1,ii) = size(find([ip- ipHat_zf_bc]),2);
    nErr_zf_cc(1,ii) = size(find([ip- ipHat_zf_cc(1:N)]),2);
    nErr_mmse(1,ii) = size(find([ip- ipHat_mmse]),2);
    nErr_mmse_bc(1,ii) = size(find([ip- ipHat_mmse_bc]),2);
    nErr_mmse_cc(1,ii) = size(find([ip- ipHat_mmse_cc(1:N)]),2);

end

simBer_zf = nErr_zf/N; % simulated ber
simBer_zf_bc = nErr_zf_bc/N; % simulated ber
simBer_zf_cc = nErr_zf_cc/N; % simulated ber
simBer_mmse = nErr_mmse/N; % simulated ber
simBer_mmse_bc = nErr_mmse_bc/N; % simulated ber
simBer_mmse_cc = nErr_mmse_cc/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,:),'b+-','Linewidth',2);
hold on
semilogy(Eb_N0_dB,simBer_mmse(1,:),'ro-','Linewidth',2);
hold on
semilogy(Eb_N0_dB,simBer_zf_bc(1,:),'g*-','Linewidth',2);
hold on
semilogy(Eb_N0_dB,simBer_mmse_bc(1,:),'bx-','Linewidth',2);
hold on
semilogy(Eb_N0_dB,simBer_zf_cc(1,:),'ms-','Linewidth',2);
hold on
semilogy(Eb_N0_dB,simBer_mmse_cc(1,:),'cd-','Linewidth',2);
axis([0 14 10^-5 0.5])
grid on
legend('sim-zf', 'sim-mmse', 'sim-zf-hamming', 'sim-mmse-hamming', 'sim-zf-convolutional', 'sim-mmse-convolutional');
xlabel('Eb/No, dB');
ylabel('Bit Error Rate');
title('Bit error probability curve for BPSK in ISI with ZF & MMSE equalizer');
print("q4.jpg");
```