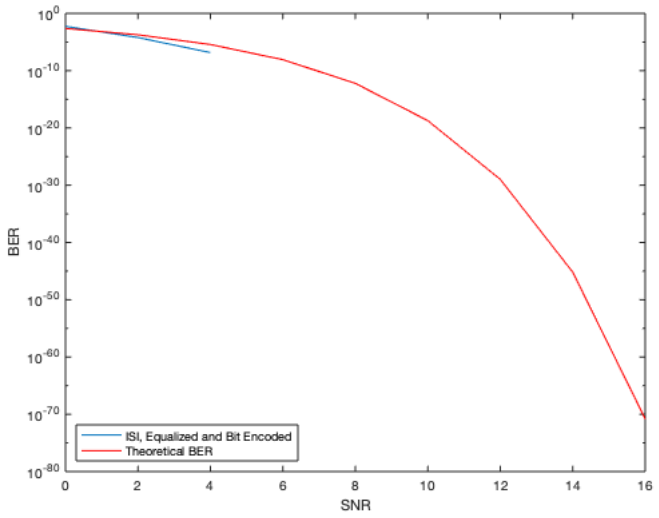
```
clear;close all;clc
numIter = 5000;  % The number of iterations of the simulation. We used 4000 because our code
% is too strong and we need more bits to get observable error
nSym = 1000;    % The number of symbols per packet
SNR_Vec = 0:2:16;
lenSNR = length(SNR_Vec);

M = 16;        % The M-ary number, 2 corresponds to binary modulation

%chan = 1;          % No channel
chan = [1 .2 .4]; % Somewhat invertible channel impulse response, Moderate ISI
%chan = [0.227 0.460 0.688 0.460 0.227]';   % Not so invertible, severe ISI

% Create equalizer object with signal ocnstellation based on M
% We used a Decision Feedback Equalizer instead of linear because it
% produced much better BER
sigConst = qammod(0:M-1,M);
eqlms = dfe(5,3, lms(0.005));
eqlms.SigConst = sigConst;
eqlms.ResetBeforeFiltering = 0;

% Create a vector to store the BER computed during each iteration
berVec = zeros(numIter, lenSNR);

% Run the simulation numIter amount of times
for i = 1:numIter

    bits = randi([0 1],  1, nSym*log2(M));     % Generate random bits
    % New bits must be generated at every iteration
    % To reduce BER, we use convolutional coding. Convolutional coding was
    % chosen to keep simulation time manageable. BCH would take far too
    % long. We managed to achieve 10^(-7) BER with 2 to 3 convolutional
    % coding. 1 to 2 convolutional coding could not.
    trellis = poly2trellis([5 4],[23 35 0; 0 5 13]); % Constraint Lengths 5 and 4 for two input channels, two sets of Generator Matrices for two input
    convbits = convenc(bits, trellis); % apply convolutional coding
    convbitLen = length(convbits); % length after bit encoding
    trainLen = convbitLen/5; % Use the first 20% of bits to train
    % Here we convert the bits into symbols ranging from 0 to M-1
    msg = zeros(1, convbitLen/log2(M));
    msgindex = 1;
    for bitindex = 1:log2(M):(convbitLen)
        msg(msgindex) = bi2de(convbits(bitindex:bitindex+log2(M)-1), 'left-msb');
        msgindex = msgindex+1;
    end


    %for j = 1:lenSNR % one iteration of the simulation at each SNR Value
    for j = 1:3
        tx = qammod(msg,M);  % BPSK modulate the signal
        % Choose whether or not to filter the signal thorugh a channel with
        % ISI
        if isequal(chan,1)
            txChan = tx;
        else
            txChan = filter(chan,1,tx);  % Apply the channel.
        end

        txNoisy = awgn(txChan,SNR_Vec(j)+ 10*log10(log2(M))); % Add AWGN

        eqSig = equalize(eqlms,txNoisy,tx(1:trainLen)); %equalize the signal with trainLen training bits

        rxEq = qamdemod(eqSig,M); % Demodulate

        % Convert symbols back to bits
        rxMSG = zeros(1, convbitLen);
        rxMSGindex = 1;
        for rxEqindex = 1:1:length(rxEq)
            rxMSG(rxMSGindex:rxMSGindex+log2(M)-1) = de2bi(rxEq(rxEqindex), log2(M), 2, 'left-msb');
            rxMSGindex = rxMSGindex + log2(M);
        end

        % decode our convolutional encoding
        deconvbits = vitdec(rxMSG, trellis, 20, 'trunc', 'hard');

        % Compute and store the BER for this iteration
        [~, berVec(i,j)] = biterr(bits(trainLen+1:end), deconvbits(trainLen+1:end)); % We're interested in the BER, which is the 2nd output of BITERR
        % Here we only discard the first trainLen bits since they are used
        % for training

    end  % End SNR iteration
end      % End numIter iteration


% Compute and plot the mean BER
ber = mean(berVec,1);

figure;
semilogy(SNR_Vec, ber)

% Compute the theoretical BER for this scenario
```

```
berTheory = berawgn(SNR_Vec + 10*log10(log2(M)),'psk',log2(M),'nondiff');
hold on
semilogy(SNR_Vec,berTheory,'r');
xlabel("SNR");
ylabel("BER");
legend('ISI, Equalized and Bit Encoded', 'Theoretical BER', 'Location', 'southwest')
```



Published with MATLAB® R2019b