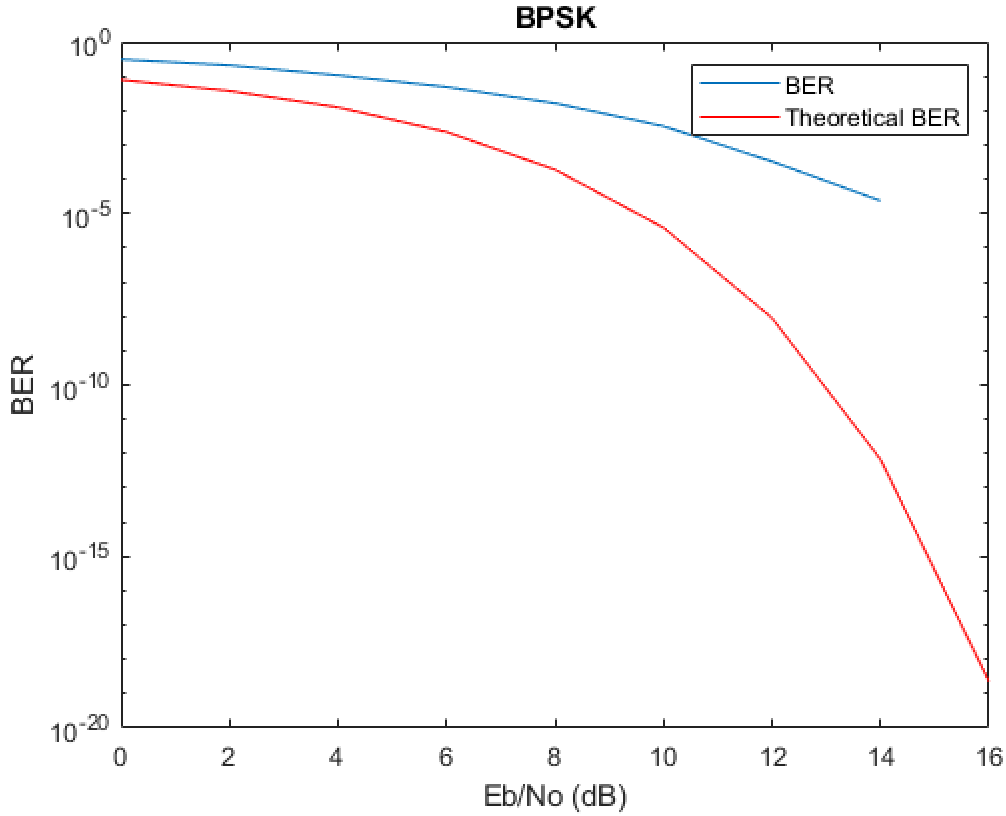
```
%Comms Link
%The goal of this project is to design and simulate an end-to-end
%communications link with coding, modulation and equalization.

%Sources
%source1: https://www.mathworks.com/help/comm/ug/equalize-a-bspk-
signal.html
%source2: https://www.mathworks.com/help/comm/ref/bchenc.html
%source3: https://www.mathworks.com/help/comm/ref/dfe.html
```

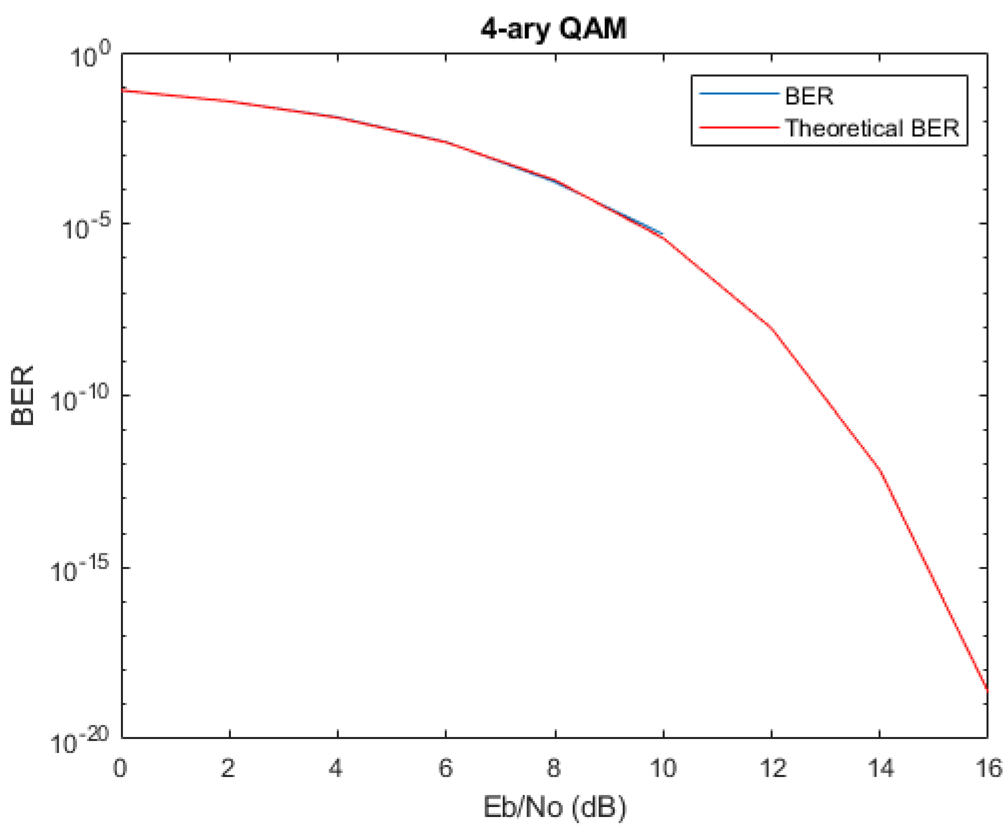
Part 1: Design Equalizer to Achieve Moderate ISI BER of 1e-4

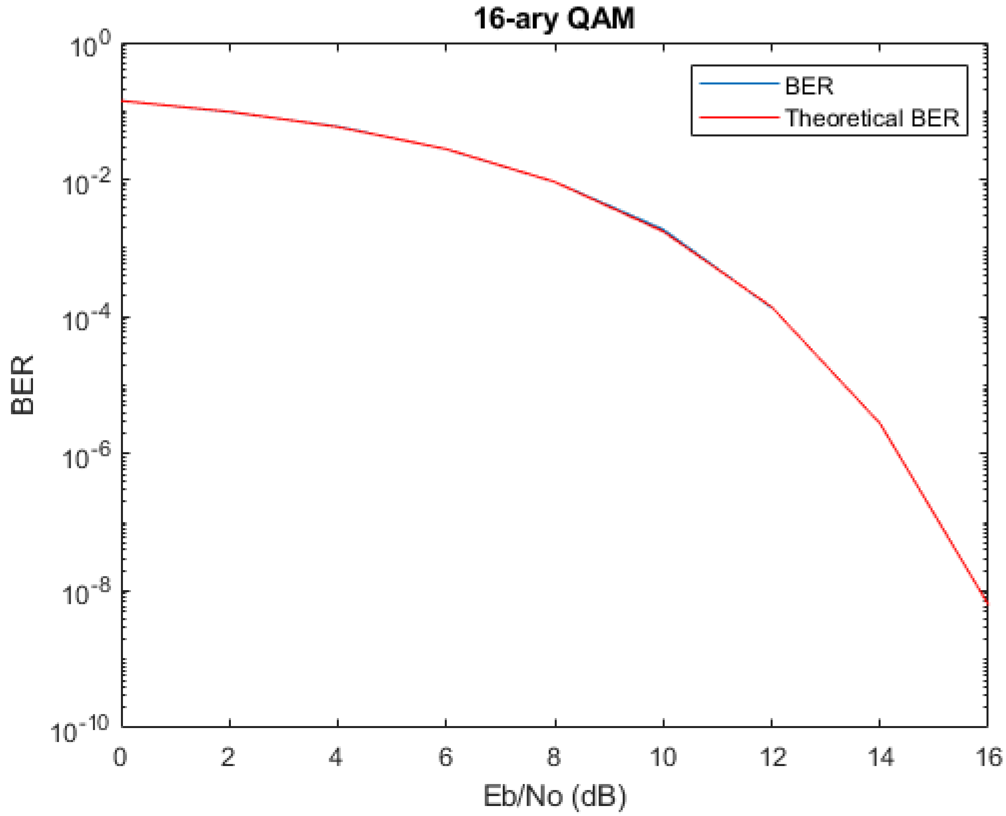
```
clear all;close all;clc

SNR_Vec = 0:2:16;
lenSNR = length(SNR_Vec);
M = [2, 4, 16];     %The M-ary number, 2 corresponds to BPSK
Mlabel = {'BPSK', '4-ary QAM', '16-ary QAM'};
lenM = length(M);

%Parameters
numIter = 100; %The number of iterations of the simulation
nSym = 1000; %The number of symbols per packet
tr = 150;       %The number of training bits

% LOOP 1 through M-ary number
for l = 1:lenM
    Mary=M(l);

    if Mary == 2
        chan = [1 .2 .4]; %Moderate ISI
    else
        chan = 1;
    end

    berVec = zeros(numIter,lenSNR); %Vector to store BER at each
 iteration

    %LOOP 2 through number of iterations
    for i = 1:numIter %Run the simulation numIter amount of times

        bits = randi([0,1], 1, nSym*log2(Mary)); %Generate random bits
 at every iteration

        %Bits to symbols
        if Mary == 2
            msg = bits;
        else
            msg = reshape(bits,[log2(Mary),nSym]);
            msg = (bi2de(msg.','left-msb'))';
```

```
        end

        %LOOP 3 through SNR
        for j = 1:lenSNR %Iterate through SNR values

            %Modulate the signal
            tx = qammod(msg,Mary);

            %Channel
            if isequal(chan,1)
                txChan = tx;
            else
                txChan = filter(chan,1,tx);
            end

            %AWGN
            txNoisy =
 awgn(txChan,SNR_Vec(j)+10*log10(log2(Mary)), 'measured');

            %Differential Feedback Equalizer
            if Mary == 2
                stepsize = 0.01;
                dfeObj = dfe(5,3,lms(stepsize)); %Differential
 feedback object
                dfeObj.SigConst =
 qammod((0:Mary-1)',Mary,'UnitAveragePower',true)';
                dfeObj.ResetBeforeFiltering = 1;
                [txEq,~,e] = equalize(dfeObj,txNoisy,tx(1:floor(tr/
(log2(Mary)))));

                %Initially I tried a linear equalizer, which obtained
 the
                %required 1e-4 ber for BPSK and moderate ISI. In part
 b,
                %however, I needed to improve the BER further and so
                %implemented the differential feedback equalizer
 above.
                %The linear equalizer is below.
                %stepsize = 0.05;
                %tr = 200; %Number of training bits
                %eqlms = lineareq(6,lms(stepsize)); %Linear equalizer
 object
                %trSeq = tx(1:tr); %Training sequence
                %[txEq,~,e] = equalize(eqlms,txNoisy,trSeq);
            else
                txEq = txNoisy;
            end

            %Demodulate the signal
            rx = qamdemod(txEq,Mary);

            %Symbols back to bits
            if Mary == 2
                rxMSG = rx;
```

```
            else
                rxMSG = de2bi(rx.','left-msb')';
                rxMSG = reshape(rxMSG, [1,nSym*log2(Mary)]);
            end

            %Compute and store BER (exclude training bits)
            if isequal(Mary,2)
                [~, berVec(i,j)] = biterr(bits((tr*log2(Mary))+1:end),
 rxMSG((tr*log2(Mary))+1:end));
            else
                [~, berVec(i,j)] = biterr(bits, rxMSG);
            end

        end%End SNR loop
    end%End numIter loop

    ber = mean(berVec,1);

    figure
    semilogy(SNR_Vec, ber)

    if Mary==2
        berTheory = berawgn(SNR_Vec,'psk',2,'nondiff');
    else
        berTheory = berawgn(SNR_Vec,'qam',Mary,'nondiff');
    end

    hold on
    semilogy(SNR_Vec,berTheory,'r')
    legend('BER', 'Theoretical BER')
    xlabel('Eb/No (dB)')
    ylabel('BER')
    title(sprintf('%s',Mlabel{l}))

end%End M-ary loop

%End of Part 1
```


BPSK
BER
Theoretical BER
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
BER
0
2
4
6
8
10
12
14
16
Eb/No (dB)


4-ary QAM
BER
Theoretical BER
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
BER
0
2
4
6
8
10
12
14
16
Eb/No (dB)



Part 2: Implement Modulation Scheme to Reduce BER and Maximize Bit Rate

```
clear all;

numIter = 1000;      %The number of iterations of the simulation
nSym = 1000;         %The number of symbols per packet
SNR = 12;
Mary = 2;            %BPSK
chan = [1 .2 .4];    %Moderate ISI

%These are constants defining the rate of the BCH code. Changing these
 changed the
%bit rate and the BER. The ones below were decided to be the optimal
%combination (simulated with 1000 iterations)
n = 31;
k = 26;
numBCH = 29;
tr = 101; % number of training bits determined by how many bits is
 needed
          %to have 1000 bits at the input of the modulator

%rate 7-4: 132 numBCH, 76 training bits: ber = 0, bitRate = .5280
    %(for 10000 iterations, ber = 3.0*10^-6, bitRate = .5280)
```

```
%rate 15-11: 62 numBCH, 70 training bits: ber = 4.4*10^-6, bitRate
 = .6820
%rate 31-26: 29 numBCH, 101 training bits: ber = 8.0*10^-6, bitRate
 = .7540
%rate 63-57: 14 numBCH, 118 training bits: ber = 1.9*10^-5 , bitRate
 = .7980
%rate 127-120: 7 numBCH, 111 training bits: ber = 3.1*10^-5, bitRate
 = .8400

%rate 31-26 is the optimal scheme to minimize BER and maximize bit
 rate.

%The communications link begins here. Bits are generated, then encoded
 by the n-k BCH scheme.
%The message is then BPSK modulated, passed through the moderate ISI
%channel, passed through the equalizer to reverse the ISI, then
 demodulated, and decoded. The
%bit error rate is calculated by comparing the initial bits to the
 bits
%outputed from the decoder.

berVec = zeros(numIter, 1); %Vector to store BER at each iteration

%LOOP through number of iterations
for i = 1:numIter

    bits = randi([0,1], numBCH, k); %Generate random bits

    %Here a BCH coding scheme is used to encode the data. The
 parameters
    %can be adjusted at the top of the code.
    bchmsg = gf(bits);
    coded = bchenc(bchmsg,n,k);
    encbits = reshape(coded.x,1,numBCH*n);

    msg = [randi([0 1], 1, tr), encbits];
    msg = reshape(msg,log2(Mary),nSym);

    %Modulate the signal
    tx = qammod(msg,Mary,'UnitAveragePower',true,'InputType','bit');

    %Channel
    txChan = filter(chan,1,tx);
    txNoisy = awgn(txChan,SNR+10*log10(log2(Mary)), 'measured'); % Add
 AWGN

    %Differential Feedback Equalizer
    stepsize = 0.01;
    dfeObj = dfe(5,3,lms(stepsize)); %the differential feedback object
 used in the equalizer
    dfeObj.SigConst =
 qammod((0:Mary-1)',Mary,'UnitAveragePower',true)';
    dfeObj.ResetBeforeFiltering = 1;
```

```
    [txEq,~,e] = equalize(dfeObj,txNoisy,tx(1:floor(tr/
(log2(Mary)))));

    %Demodulate
    rxMSG =
 qamdemod(txEq,Mary,'UnitAveragePower',true,'OutputType','bit');

    %Decode the recieved data using the BCH coding scheme
    channeled = reshape(rxMSG(tr+1:end),numBCH,n);
    decoded = bchdec(gf(channeled),n,k);
    decodednotgf = double(decoded.x);
    decbits = reshape(decodednotgf,1,[]);

    %Compute and store BER (exclude training bits)
    [~, berVec(i)] = biterr(bits(:), decbits(:));

end%End numIter loop

%Compute the mean BER and bit rate
ber = mean(berVec,1)
bitRate = k*numBCH/(nSym*log2(Mary))

%End of Part 2
```

```
ber =

   7.9576e-06


bitRate =

    0.7540
```

Published with MATLAB® R2019b