AWGN channel with ISI Simulation (CommTheory recap)

Wireless Communications, Spring 2020 January 29, 2020 Jongoh (Andy) Jeong

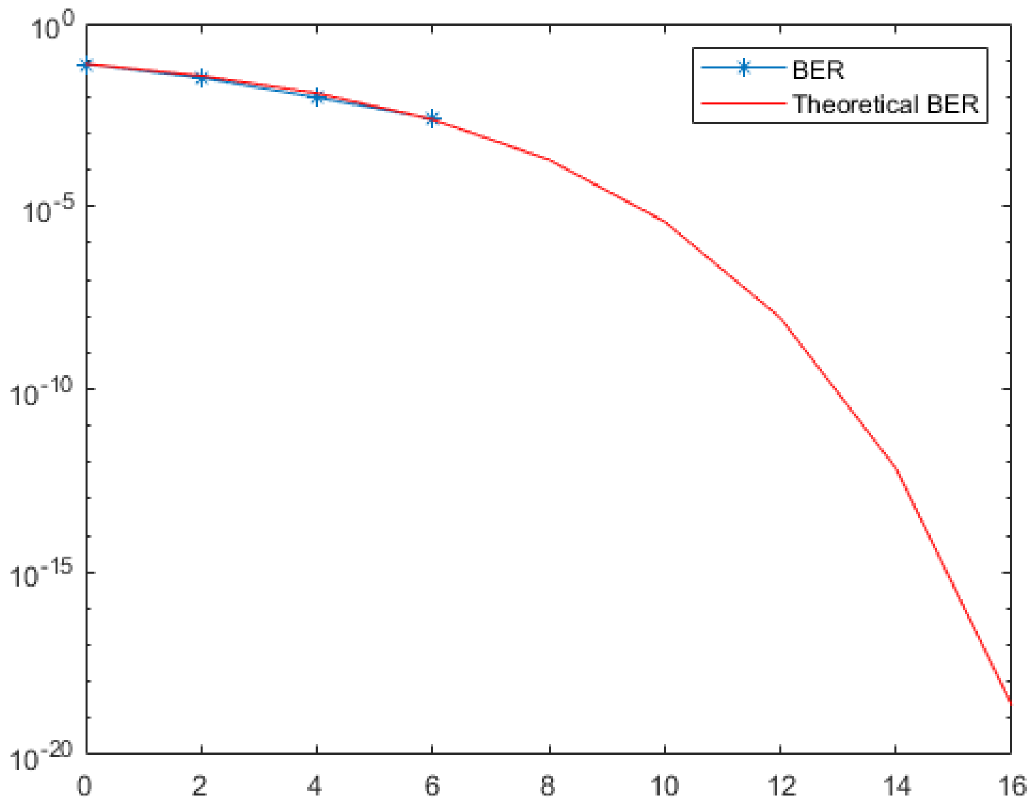
Part I - bit transmission (BPSK) through AWGN channels using linera modulation schemes M-ary PSK, PAM, QAM) and equalizers (Linear, DFE, MLSE) - goal: reach BER of 10e-4 order using equalizers Part II - Encoded transmission using error correcting code at symbol- and bit-level, and convolutional encoding (RS, BCH, Convolutional encoding-Viterbi decoding) - goal: reach BER of 10e-6 order using ECC

Contents

wireless link simulation - skeleton script

```
clear all;close all;clc;

% (code from Fall 2018)
% For the final version of this project, you must use these 3
% parameter. You will likely want to set numIter to 1 while you debug your
% link, and then increase it to get an average BER.
numIter = 1;  % The number of iterations of the simulation
nSym = 1000;    % The number of symbols per packet
SNR_Vec = 0:2:16;
lenSNR = length(SNR_Vec);
M = 2;        % 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
% Time-varying Rayleigh multipath channel, try it if you dare. Or take
% wireless comms.
% ts = 1/1000;
% chan = rayleighchan(ts,1);
% chan.pathDelays = [0 ts 2*ts];
% chan.AvgPathGaindB = [0 5 10];
% chan.StoreHistory = 1; % Uncomment if you want to be able to do plot(chan)
% 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
    % Generate random bits
    bits = randi(2,[nSym*M, 1])-1;
    % New bits must be generated at every iteration
    % If you increase the M-ary number, as you most likely will, you'll need to
    % convert the bits to integers. See the BI2DE function
    % For binary, our MSG signal is simply the bits
    msg = bits;
    for j = 1:lenSNR % one iteration of the simulation at each SNR Value
        tx = qammod(msg,M);  % BPSK modulate the signal
```

```
        if isequal(chan,1)
            txChan = tx;
        elseif isa(chan,'channel.rayleigh')
            reset(chan) % Draw a different channel each iteration
            txChan = filter(chan,tx);
        else
            txChan = filter(chan,1,tx);  % Apply the channel.
        end
        % Convert from EbNo to SNR.
        % Note: Because No = 2*noiseVariance^2, we must add ~3 dB
        % to get SNR (because 10*log10(2) ~= 3).
        txNoisy = awgn(txChan,3+SNR_Vec(j),'measured'); % Add AWGN
        rx = qamdemod(txNoisy,M); % Demodulate
        % Again, if M was a larger number, I'd need to convert my symbols
        % back to bits here.
        rxMSG = rx;
        % Compute and store the BER for this iteration
        [zzz berVec(i,j)] = biterr(msg, rxMSG);
         % We're interested in the BER, which is the 2nd output of BITERR
    end  % End SNR iteration
end      % End numIter iteration
% Compute and plot the mean BER
ber = mean(berVec,1);
semilogy(SNR_Vec, ber,'-*');
% Compute the theoretical BER for this scenario
% THIS IS ONLY VALID FOR BPSK!
% YOU NEED TO CHANGE THE CALL TO BERAWGN FOR DIFF MOD TYPES
% Also note - there is no theoretical BER when you have a multipath channel
berTheory = berawgn(SNR_Vec,'pam',2);
hold on
semilogy(SNR_Vec,berTheory,'r')
legend('BER', 'Theoretical BER')
```



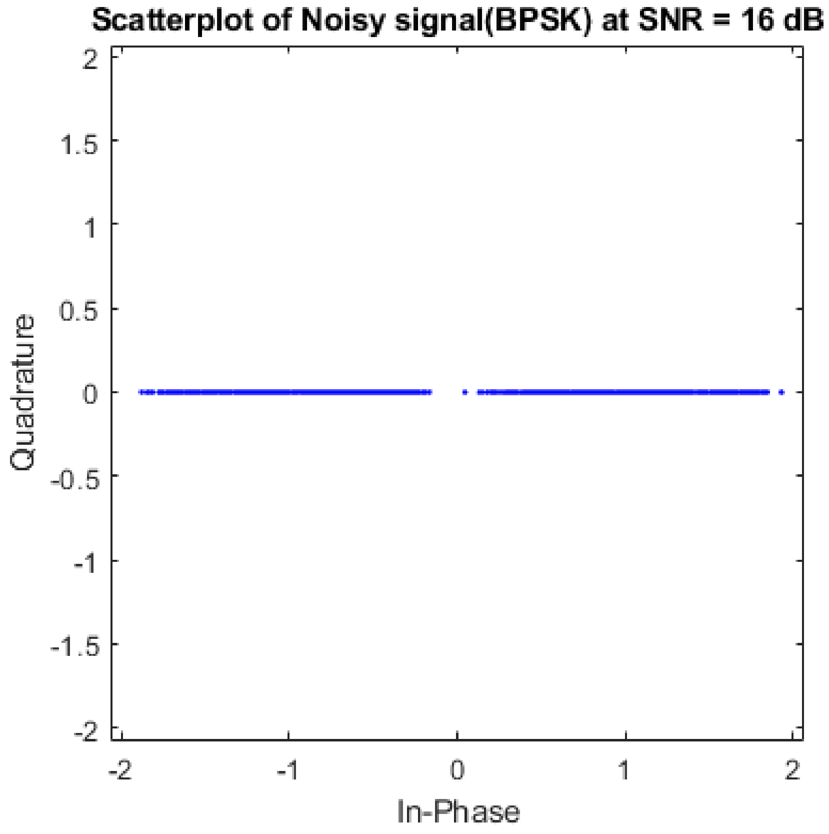
wireless link simulation - Part I - equalizer (linear, DFE, MLSE)

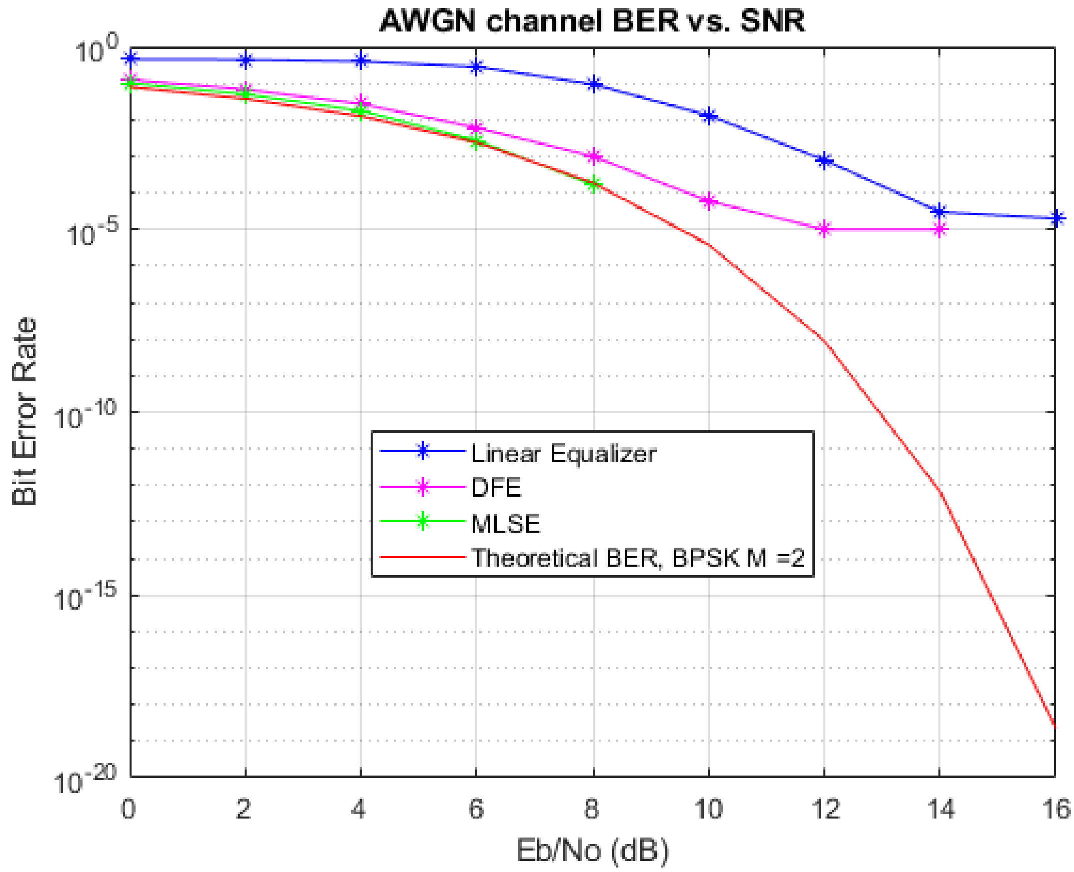
channel with moderate ISI

```
clear all;close all;clc;
rng default;                                % using default RNG

nIter = 100;                                % The number of iterations of the simulation
nSymbols = 1000;                            % symbols processed (max 1000)
snrVector = 0:2:16;                         % Eb/No values (dB)
lenSnrVector = length(snrVector);           % length of SNR vector
M = 2;  % BPSK
channel = [1, 0.2, 0.4];                    % moderate ISI channel
BERVectorLin = zeros(nIter, lenSnrVector);
BERVectorDfe = zeros(nIter, lenSnrVector);
BERVectorMLSE = zeros(nIter, lenSnrVector);
k = log2(M);                                % bits per symbol
nsamp = 1;                                  % sampling rate
% LMS Linear Equalizer
eq1 = lineareq(4, lms(0.1));
eq1.SigConst = qammod((0:M-1)',M)';
eq1.RefTap = 1;
eq1.ResetBeforeFiltering = 0;
% LMS Nonlinear Decision-Feedback Equalizer
dfe = dfe(2, 3, lms(0.01));
dfe.SigConst = qammod((0:M-1)',M)';
dfe.RefTap = 1;
dfe.ResetBeforeFiltering = 0;
% Nonlinear MLSE Equalizer
tblen = 10; % traceback depth for mlse equalizer
constellation = qammod((0:M-1)',M)';
mlse = comm.MLSEEqualizer('TracebackDepth',tblen,'Channel',channel', ...
    'Constellation',constellation);
trainlen = 0.1 * nSymbols;  % training length = 10% of sent symbols
noisySignals = zeros(nSymbols,lenSnrVector); % intiailize for scatterplot

% Run the simulation numIter amount of times
tic;
for i = 1:nIter
    msgBin = randi([0 1],[nSymbols*k, 1]);
    msgDec = bi2de(msgBin);
    for j = 1:lenSnrVector % iteration of the simulation at each SNR Value
        snrdB = snrVector(j) + 10*log10(2) - 10*log10(nsamp); % noise scale by 3dB
        % QAM modulate using gray symbol mapping
        modMsg = qammod(msgDec, M);
        rxMsg = filter(channel,1,modMsg);
        % add AWGN
        rxNoisy = awgn(rxMsg,snrdB,'measured');
        noisySignals(:,j) = rxNoisy; % for scatterplot
        % equalize
        [~, rxNoisyLIN] = equalize(eq1,rxNoisy,modMsg(1:trainlen));
        [~, rxNoisyDFE] = equalize(dfe,rxNoisy,modMsg(1:trainlen));
        rxNoisyMLSE = mlse(rxNoisy);
        % demodulate
        rxMsgDfeDec = qamdemod(rxNoisyDFE,M);
        rxMsgDfeBin = de2bi(rxMsgDfeDec,k);
        rxMsgLinDec = qamdemod(rxNoisyLIN,M);
        rxMsgLinBin = de2bi(rxMsgLinDec,k);
        rxMsgMLSEDec = qamdemod(rxNoisyMLSE,M);
        rxMsgMLSEBin = de2bi(rxMsgMLSEDec,k);
```

```
        % Calculate Bit Error
        [nErrorsLin, BERVectorLin(i,j)] = biterr(msgBin, rxMsgLinBin);
        [nErrorsDfe, BERVectorDfe(i,j)] = biterr(msgBin, rxMsgDfeBin);
        [nErrorsMLSE, BERVectorMLSE(i,j)] = biterr(msgBin, rxMsgMLSEBin);
    end  % End SNR iteration
end      % End numIter iteration
toc;

% graphs
scatterplot(noisySignals(:,9)); title('Scatterplot of Noisy signal(BPSK) at SNR = 16 dB');
% Compute and plot the mean BER
berLin = mean(BERVectorLin, 1);
berDfe = mean(BERVectorDfe, 1);
berMLSE = mean(BERVectorMLSE, 1);
figure(2);
semilogy(snrVector, berLin,'b-*'); hold on;
semilogy(snrVector, berDfe,'m-*'); hold on;
semilogy(snrVector, berMLSE,'g-*'); hold on;
berTheory = berawgn(snrVector,'pam',M); hold on; % BPSK theoretical
semilogy(snrVector,berTheory,'r'); grid on;
legend('Linear Equalizer', 'DFE', 'MLSE', ...
strcat('Theoretical BER, BPSK M = ',int2str(M)),'Location','best');
xlabel('Eb/No (dB)'); ylabel('Bit Error Rate'); title('AWGN channel BER vs. SNR');

% bit rate = samp_rate * (symbols Tx)*(bits per 1 symbol) / total_symbols
bitRate1 = nsamp * ((nSymbols-trainlen)*k) / nSymbols;
bitRate2 = nsamp * ((nSymbols)*k) / nSymbols;
sprintf('Linear and DFE bit rate: %d', bitRate1)
sprintf('MLSE bit rate: %d',bitRate2)
```

```
Elapsed time is 85.982660 seconds.

ans =

    'Linear and DFE bit rate: 9.000000e-01'


ans =

    'MLSE bit rate: 1'
```





wireless link simulation - Part II - BPSK Linear block

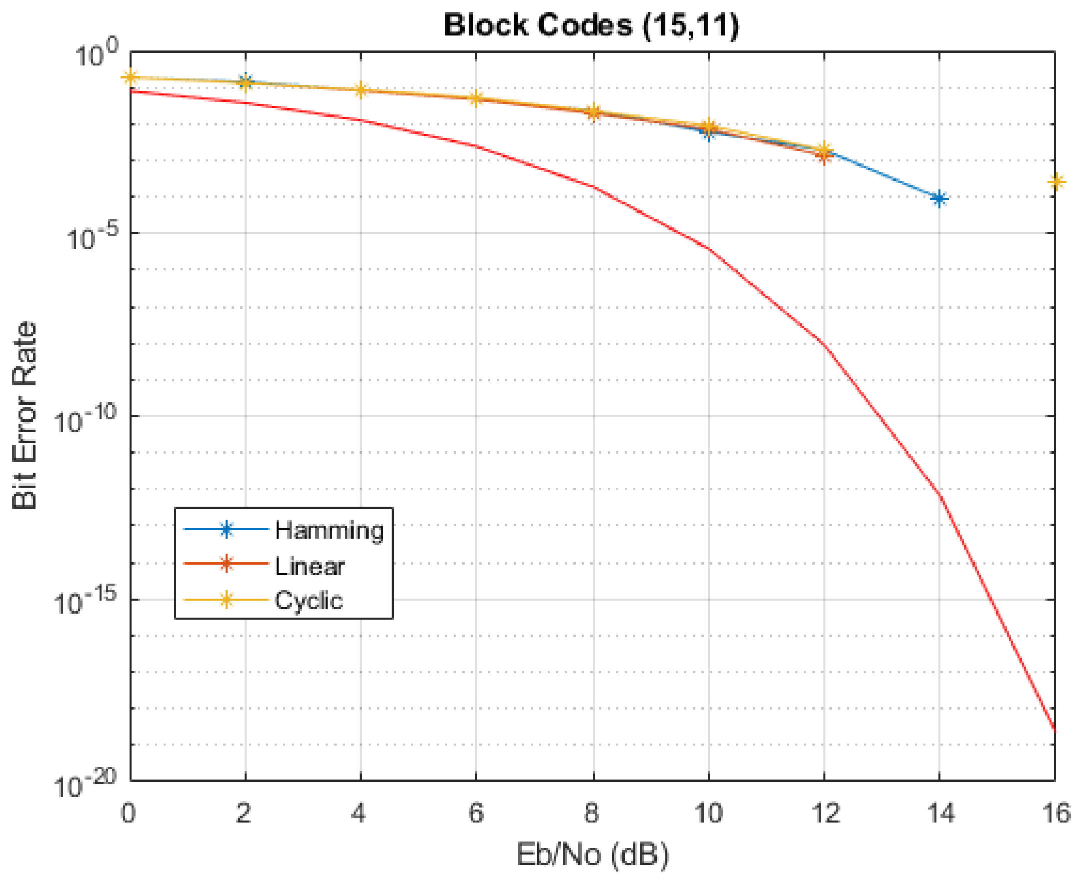
hamming, linear, cyclic Block Codes for BPSK

```
clear all;close all;clc;
```

```
nIter = 1;  % The number of iterations of the simulation
nSymbols = 1e3; % number of bits to process
snrVector = 0:2:16; % Eb/No values (dB)
lenSnrVector = length(snrVector);
M = 2; % BPSK
k = log2(M);
channel = [1, 0.2, 0.4];
% initialize BER vector
BERVector1 = zeros(nIter, lenSnrVector);
BERVector2 = zeros(nIter, lenSnrVector);
BERVector3 = zeros(nIter, lenSnrVector);
% parameters
N = 15; % Codeword length
K = 11; % message length
% N = 7; K = 4;
codeRate = K/N;

% Hamming Binary
prim_poly1 = gfprimdf(N-K); % default
% Linear Binary
pol2 = cyclpoly(N,K);
parmat2 = cyclgen(N,pol2);
genmat2 = gen2par(parmat2);
% Cyclic Binary
gpol3 = cyclpoly(N,K);
trainlen = 0;
tic;
% Run the simulation numIter amount of times
for i = 1:nIter
    % Generate random bits
    data = randi([0 1],nSymbols*K*k,1);                 % Generate binary data
    data = bi2de(data);
    encodedData1 = encode(data,N,K,'hamming/binary',prim_poly1);
    encodedData2 = encode(data,N,K,'linear/binary',genmat2);
    encodedData3 = encode(data,N,K,'cyclic/binary',gpol3);
    for j = 1:lenSnrVector % iteration of the simulation at each SNR Value
          snrdB = snrVector(j) + 10*log10(2) + 10*log10(codeRate);
          modSignal1 = qammod(encodedData1,M);         % BPSK modulate
          modSignal2 = qammod(encodedData2,M);         % BPSK modulate
          modSignal3 = qammod(encodedData3,M);         % BPSK modulate
          % 'InputType','bit'
          modSignalFilt1 = filter(channel,1,modSignal1);
          modSignalFilt2 = filter(channel,1,modSignal2);
          modSignalFilt3 = filter(channel,1,modSignal3);
          receivedSignal1 = awgn(modSignalFilt1,snrdB,'measured');        % Pass through A
WGN channel
          receivedSignal2 = awgn(modSignalFilt2,snrdB,'measured');        % Pass through A
WGN channel
          receivedSignal3 = awgn(modSignalFilt3,snrdB,'measured');        % Pass through A
WGN channel
          demodSignal1 = qamdemod(receivedSignal1,M);  % BSPK demodulate
          demodSignal2 = qamdemod(receivedSignal2,M);  % BSPK demodulate
          demodSignal3 = qamdemod(receivedSignal3,M);  % BSPK demodulate
          % 'OutputType','bit'
          receivedBits1 = decode(demodSignal1,N,K,'hamming/binary',prim_poly1);
          receivedBits2 = decode(demodSignal2,N,K,'linear/binary',genmat2);
          receivedBits3 = decode(demodSignal3,N,K,'cyclic/binary',gpol3);
          receivedBits1 = de2bi(receivedBits1);
          receivedBits2 = de2bi(receivedBits2);
          receivedBits3 = de2bi(receivedBits3);
```

```
            [nErr, BERVector1(i,j)] = biterr(data,receivedBits1);
            [nErr, BERVector2(i,j)] = biterr(data,receivedBits2);
            [nErr, BERVector3(i,j)] = biterr(data,receivedBits3);
    end
end
toc;
% Compute and plot the mean BER
ber1 = mean(BERVector1,1);
ber2 = mean(BERVector2,1);
ber3 = mean(BERVector3,1);
figure;
semilogy(snrVector, ber1,'-*', snrVector, ber2,'-*', snrVector, ber3,'-*');
% Note: No theoretical BER when you have a multipath channel
berTheory = berawgn(snrVector,'pam',M); hold on; % 2-QAM ~ 2-PAM ~ BPSK
semilogy(snrVector,berTheory,'r'); grid on;
legend('Hamming', 'Linear', 'Cyclic','Location','best');
xlabel('Eb/No (dB)'); ylabel('Bit Error Rate'); title('Block Codes (15,11)');
```

```
Elapsed time is 3.232523 seconds.
```

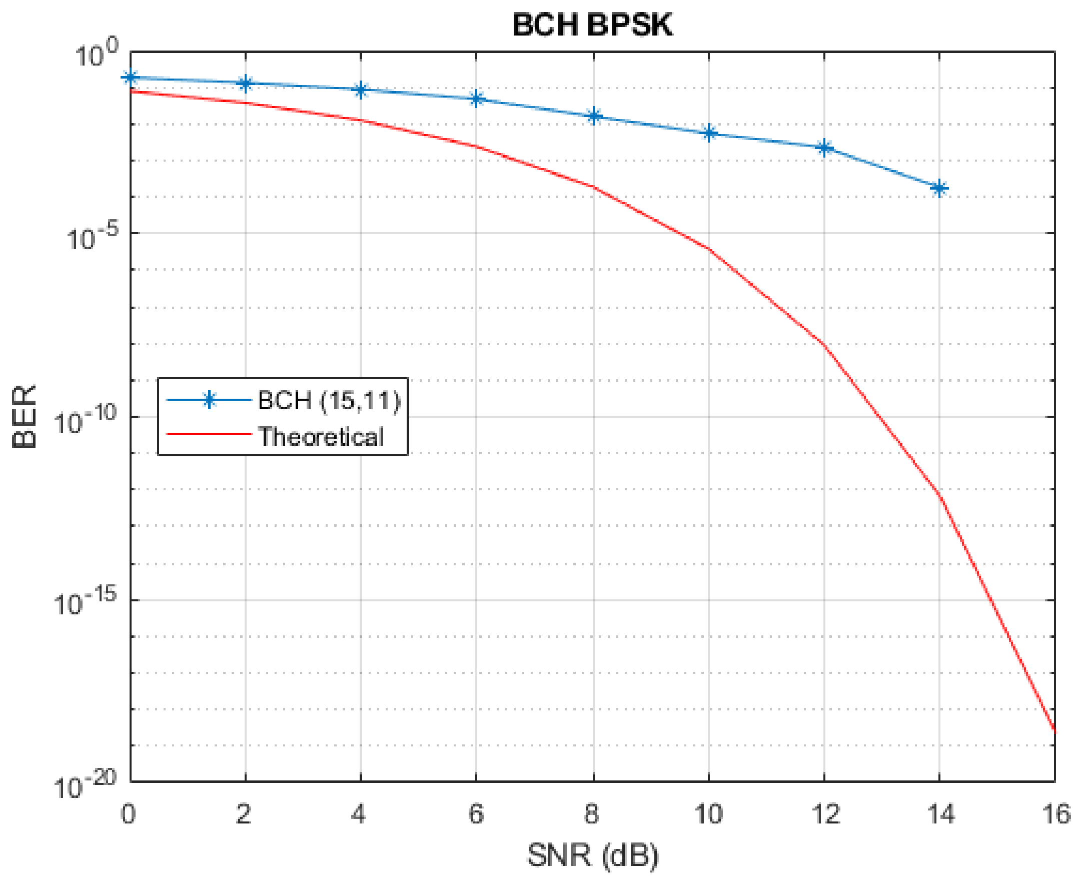


wireless link simulation - Part II - BPSK BCH

```
clear all;close all;clc;

N = 15;
K = 11;
codeRate = K/N;
M = 2; % BPSK
k = log2(M);
nIter = 1;  % The number of iterations of the simulation
nSymbols = 1e3;
```

```
channel = [1, 0.2, 0.4];
snrVector = 0:2:16; % Eb/No values (dB)
lenSnrVector = length(snrVector);
bchEncoder = comm.BCHEncoder(N,K);
bchDecoder = comm.BCHDecoder(N,K);
BERVector = zeros(k, lenSnrVector);
% Run the simulation numIter amount of times
tic;
for i = 1:nIter
    msg = randi([0 1],[nSymbols*K*k,1]);
    data = bi2de(msg);
    encodedData = bchEncoder(data);             % BCH encode data
    for j = 1:lenSnrVector
        snrdB = snrVector(j) + 10*log10(2) + 10*log10(codeRate);
        modSignal = qammod(encodedData,M);          % BPSK modulate
        filtSignal = filter(channel,1,modSignal);  % pass through channel with moderate IS
I
        receivedSignal = awgn(filtSignal,snrdB,'measured');% Add AWGN
        demodSignal = qamdemod(receivedSignal,M);  % BSPK demodulate
        receivedSyms = bchDecoder(demodSignal);    % BCH decode data
        receivedBits = de2bi(receivedSyms);
        [nErrors, BERVector(i,j)] = biterr(data, receivedBits);
    end % end SNR iteration
end
toc;
ber = mean(BERVector,1);
figure;
semilogy(snrVector, ber,'-*'); grid on; hold on;
% Note: No theoretical BER when you have a multipath channel
berTheory = berawgn(snrVector,'pam',M); hold on;
semilogy(snrVector,berTheory,'r');
legend('BCH (15,11)', 'Theoretical','Location','best');
xlabel('Eb/No (dB)'); ylabel('BER'); hold off; grid;
title('BCH BPSK'); xlabel('SNR (dB)'); ylabel('BER'); grid;
```

```
Elapsed time is 2.080113 seconds.
```

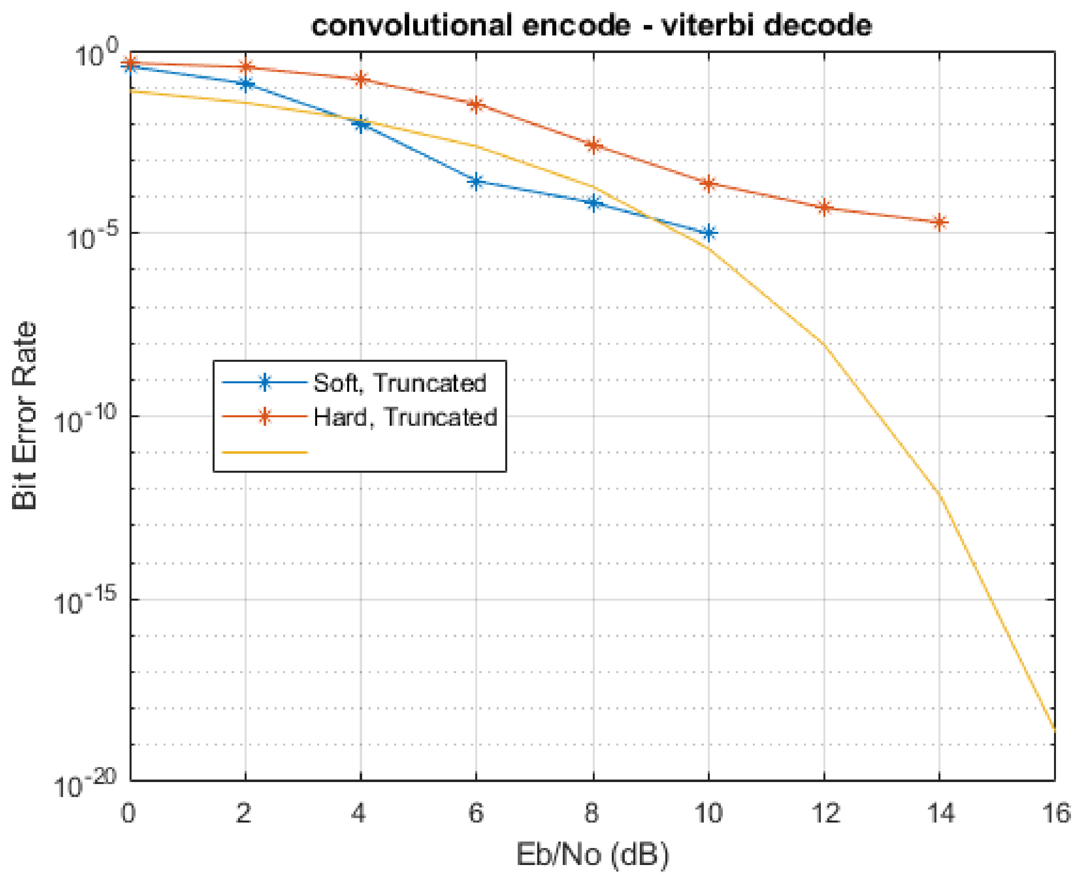


wireless link simulation - Part II - BPSK Conv. code

```
clear all;close all;clc;

M = 2;                  % BPSK
k = log2(M);            % Bits per symbol
snrVector = 0:2:16; % Eb/No values (dB)
nIter = 1e2;                % # of iterations
nSymbols = 1e3;             % Number of symbols per frame
% poly2trellis:
%   K = constraint length
%   [#, #] = generator polynomial in octal base
trellis = poly2trellis(7,[171 133]); % values experimentally chosen
codeRate = 1/2; % since k = 1 (binary) and gp is defined to be 2 (2 shift registers) for t
rellis, rate = 1/2
tbl = 100;                  % traceback depths (higher results in better, but there seems to b
e a limit
channel = [1 0.2 0.4];  % moderate ISI channel
BERVectorSoft = zeros(k, length(snrVector)); % initialize vectors to store BER(soft)
BERVectorHard = zeros(k, length(snrVector));% initialize vectors to store BER(hard)

tic;
for i = 1:nIter
    % Reset the error and bit counters
    [numErrsSoft,numErrsHard,numBits] = deal(0);
    % Generate binary data and convert to symbols
    dataIn = randi([0 1],nSymbols*k,1);
    % Convolutional-encode the data
    dataEnc = convenc(dataIn,trellis);
    for j = 1:length(snrVector)
        % Convert Eb/No to SNR with scaling
        snrdB = snrVector(j) + 10*log10(2) + 10*log10(codeRate);
        txSig = qammod(dataEnc,M,'InputType','bit'); % input rows => integer multiple of k
```

```
 = log2(M)
        txSigFilt = filter(channel,1,txSig); % pass through the channel
        rxSig = awgn(txSigFilt,snrdB,'measured'); % Pass through AWGN channel
        % Demodulate the noisy signal by
                % hard (bit) and soft decision (approx. Log Likelihood Ratio) approaches.
        rxDataHard = qamdemod(rxSig,M,'OutputType','bit');
        rxDataSoft = qamdemod(rxSig,M,'OutputType','approxllr');
        % (1) truncation mode (no delay in decoding)
        % Viterbi decode the demodulated data
        dataHard = vitdec(rxDataHard,trellis,tbl,'trunc','hard');
        dataSoft = vitdec(rxDataSoft,trellis,tbl,'trunc','unquant');
        % Calculate the number of bit errors in the frame. Adjust for the
        % decoding delay, which is equal to the traceback depth.
        [numErrsInFrameHard,BERVectorHard(i,j)] = biterr(dataIn(1:end),dataHard(1:end));
        [numErrsInFrameSoft,BERVectorSoft(i,j)] = biterr(dataIn(1:end),dataSoft(1:end));
    end
end
toc;
figure;
meanbersoft = mean(BERVectorSoft,1);
meanberhard = mean(BERVectorHard,1); % find mean BERs
semilogy(snrVector,meanbersoft,'-*',snrVector, meanberhard,'-*'); hold on;
theor_ber = berawgn(snrVector,'pam',M); % 'pam' for BPSK
semilogy(snrVector, theor_ber);
legend('Soft, Truncated','Hard, Truncated','','location','best');
title('convolutional encode - viterbi decode'); grid on;
xlabel('Eb/No (dB)'); ylabel('Bit Error Rate');
```

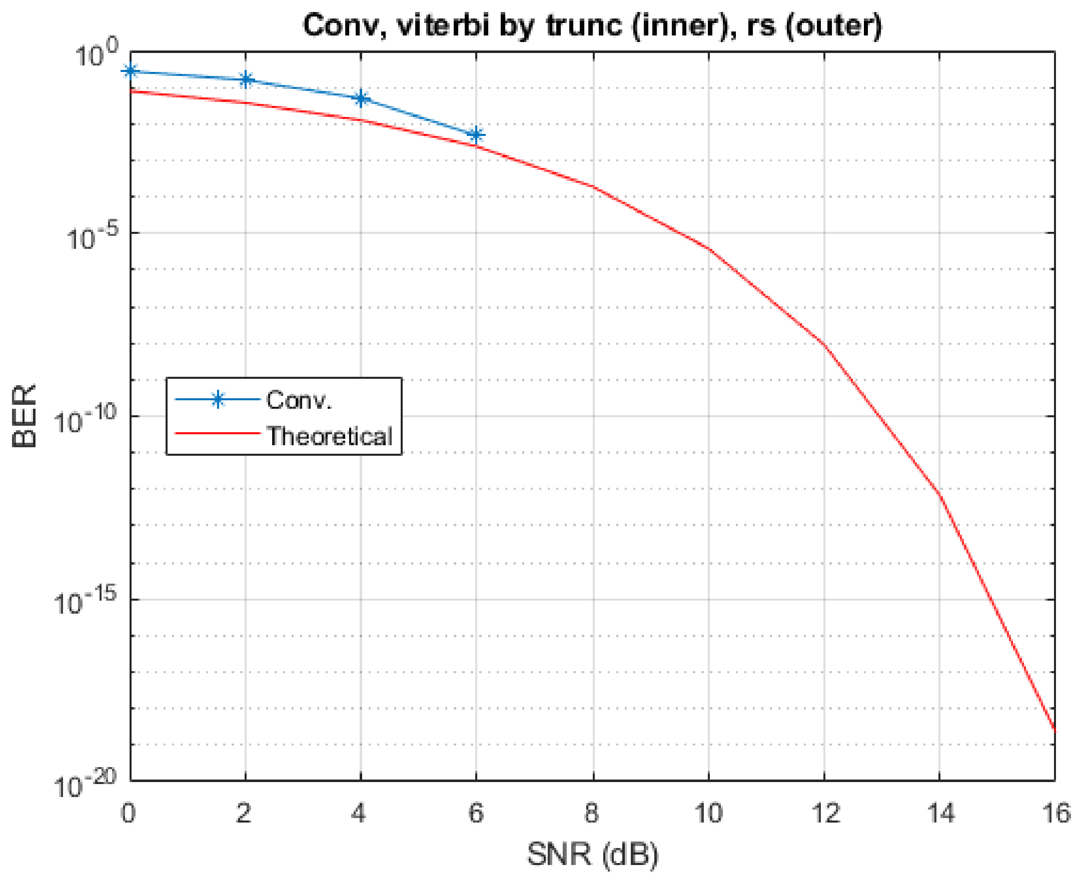
```
Elapsed time is 16.287059 seconds.
```



wireless link simulation - Part II - BPSK RS + Conv. code

```
close all; clear all; clc;

N = 15; % codeword length
K = 11; % message length
codeRateRS = K/N;
nIter = 1;  % The number of iterations of the simulation
nSymbols = 1e3;
channel = [1, 0.2, 0.4]; % moderate ISI Channel
snrVector = 0:2:16; % Eb/No values (dB)
lenSnrVector = length(snrVector);
M = 2; % BPSK
k = log2(M);
% RSEncoder:
%   comm.RSEncoder / comm.RSEncoder(N,K) / comm.RSEncoder(N,K,gp)
%   the form (N,K,gp) gave relatively better results
genpoly = rsgenpoly(N,K);
rsEncoder = comm.RSEncoder(N,K,genpoly,'BitInput',true);
rsDecoder = comm.RSDecoder(N,K,genpoly,'BitInput',true);
trellis = poly2trellis(3, [7 5]); % trellis structure giving good performance
tbl = 100;
codeRate = 1/2;
% Run the simulation numIter amount of times
tic;
for i = 1:nIter
    msg = randi([0 1],[nSymbols*K,1]);
    data = de2bi(msg);
    data = rsEncoder(data);
    data = convenc(data,trellis);
    encodedData = data;
    for j = 1:lenSnrVector
        snrdB = snrVector(j) + 10*log10(2) + 10*log10(codeRateRS) + 10*log10(codeRate);
        modSignal = qammod(encodedData,M,'InputType','bit');         % BPSK modulate
        filtSignal = filter(channel,1,modSignal);  % pass through channel with moderate IS
I
        receivedSignal = awgn(filtSignal,snrdB,'measured');% Add AWGN
        demodSignal = qamdemod(receivedSignal,M,'OutputType','bit','OutputType','approxllr
');  % BSPK demodulate
        receivedSyms = demodSignal;
        % trunc
        receivedSyms = vitdec(receivedSyms,trellis,tbl,'trunc','unquant');
        receivedBits = bi2de(receivedSyms);
        receivedBits = rsDecoder(receivedBits);
        [nErrors, BERVector(i,j)] = biterr(msg, receivedBits);
    end
end
toc;
BERVector(1,:) = mean(BERVector,1);
figure;
semilogy(snrVector, BERVector(1,:),'-*'); grid on; hold on;
berTheory = berawgn(snrVector,'pam',M); hold on;
semilogy(snrVector,berTheory,'r');
legend('Conv.', 'Theoretical','Location','best');
xlabel('Eb/No (dB)'); ylabel('BER'); hold off; grid;
title('Conv, viterbi by trunc (inner), rs (outer)'); xlabel('SNR (dB)'); ylabel('BER'); gr
id;
```

```
Elapsed time is 1.437370 seconds.
```



wireless link simulation - Part II - BPSK BCH + Conv. code

```
close all; clear all; clc;

N = 15; % codeword length
K = 11; % message length
codeRateBCH = K/N;
nIter = 1;  % The number of iterations of the simulation
nSymbols = 5e6;
channel = [1, 0.2, 0.4];
% channel = 1;
snrVector = [0:2:16]; % Eb/No values (dB)
lenSnrVector = length(snrVector);
M = 2; k = log2(M);
% BCHEncoder:
%   comm.BCHEncoder / comm.BCHEncoder(N,K) / comm.BCHEncoder(N,K,gp)
%   the form (N,K) gave relatively better results
bchEncoder = comm.BCHEncoder(N,K);
bchDecoder = comm.BCHDecoder(N,K);
BERVector = zeros(k, lenSnrVector);

trellis = poly2trellis(3, [7 5]);
tbl = 100;
% Run the simulation numIter amount of times
tic;
for i = 1:nIter
        msg = randi([0 1],[nSymbols*k*K,1]);
        data1 = de2bi(msg);
    data2 = bchEncoder(data1);
    data = convenc(data2,trellis);
    encodedData = data;
        for j = 1:lenSnrVector
                snrdB = snrVector(j) + 10*log10(2) + 10*log10(codeRateBCH) + 10*log10(1/2)
```

```
;
                modSignal = qammod(encodedData,M,'InputType','bit');         % BPSK modula
te
                filtSignal = filter(channel,1,modSignal);  % pass through channel with mod
erate ISI
                receivedSignal = awgn(filtSignal,snrdB,'measured');% Add AWGN
                demodSignal = qamdemod(receivedSignal,M,'OutputType','bit','OutputType','a
pproxllr');  % BSPK demodulate

                % trunc
                receivedSyms = vitdec(demodSignal,trellis,tbl,'trunc','unquant');
                receivedBits = bi2de(receivedSyms);
                receivedBits = bchDecoder(receivedBits);
                [nErrors, BERVector(i,j)] = biterr(msg, receivedBits);
        end % end SNR iteration
end               % end niterations
toc;
figure;
semilogy(snrVector, BERVector,'-*'); grid on; hold on;
berTheory = berawgn(snrVector,'pam',M); hold on;
semilogy(snrVector,berTheory,'r');
legend('Conv.', 'Theoretical','Location','best');
xlabel('Eb/No (dB)'); ylabel('BER'); hold off; grid;
title('Conv, Viterbi(inner), BCH (outer)'); xlabel('SNR (dB)'); ylabel('BER'); grid;
```

```
Elapsed time is 1102.048312 seconds.
```