

---

# ECE 300 Communication Theory

## Matlab Project 2 Using viterbi

### Table of Contents

Project Description .....	1
Algorithm Initialization .....	1
Convolutional .....	1
iteration .....	2
Plots .....	4

Zheng Liu, Jing Jiang, Tianshu Ren

## Project Description

Our Goal in part 2 is to Achieve BER of  $10^{-6}$  at 12 dB SNR over moderate ISI channel using whatever means possible.

## Algorithm Initialization

```
clear all;close all;clc

numIter = 5; % 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 = 8; % 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 a vector to store the BER computed during each iteration
berVec = zeros(numIter, lenSNR);
eqlBerVec = zeros(numIter, lenSNR);
% Run the simulation numIter amount of times
```

## Convolutional

```
%trell = poly2trellis([5 4],[23 35 0;0 5 13]); % Define trellis.
trell = poly2trellis(7,[171 133]); %7 is the constraint length.
    171 173 define the system.
traceBack = 32;
```

```
codeRate = 1/2;
```

Below is the hard decision approach we tried. It worked worse than soft

```
%decision. And soft decision is the more complicated case. So only
soft
%decision is shown.
%enc = comm.ConvolutionalEncoder(trel);
%dec =
comm.ViterbiDecoder('TrellisStructure',trel,'InputFormat','Hard','TracebackDepth'
'Continuous'); %Hard
```

## iteration

```
tic
for i = 1:numIter

    % bits = randint(1, nSym*M, [0 1]); % Generate random bits
    bits = randi(2,[codeRate*nSym*log2(M), 1])-1;

    % If you increase the M-ary number, as you most likely will,
    you'll need to
    % convert the bits to integers. See the BIN2DE function
    % For binary, our MSG signal is simply the bits

    msg = bi2de(reshape(bits,log2(M),[]).');
    encodedMSG = convenc(bits,trel); %Encoding using
    trellis generated above.

    for j = 1:lenSNR % one iteration of the simulation at each SNR
        Value

        noise_addition = 10*log10(log2(M)*codeRate);
        %To use soft decision, the variance of the noise is required for
        demodulation.
        noise_var = 10.^(-(SNR_Vec(j) + noise_addition)/10);

        tx =
        gammod(encodedMSG,M,'UnitAveragePower',true,'InputType','bit'); %
        BPSK modulate the signal
        tx2 =
        gammod(bits,M,'UnitAveragePower',true,'InputType','bit');

        if isequal(chan,1)
            txChan = tx;
            txChan2 = tx2;
        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.
            txChan2 = filter(chan,1,tx2);
        end
    end
end
```

```
% Convert from EbNo to SNR.
% Note: Because  $N_0 = 2 \cdot \text{noiseVariance}^2$ , we must add ~3 dB
% to get SNR (because  $10 \cdot \log_{10}(2) \approx 3$ ).
txNoisy =
awgn(txChan,10*log10(log2(M))+SNR_Vec(j),'measured'); % Add AWGN
txNoisy2 =
awgn(txChan2,10*log10(log2(M))+SNR_Vec(j),'measured');

rx = txNoisy;
rx2 = txNoisy2;

% Again, if M was a larger number, I'd need to convert my
symbols
% back to bits here.

rxMSG =
qamdemod(rx2,M,'UnitAveragePower',true,'OutputType','bit');
noEq1MSG =
qamdemod(rx,M,'UnitAveragePower',true,'OutputType','bit');

%rxMSG = reshape(de2bi(rx,log2(M)),[],1);

%Equalizer
mu = 0.001; %step size
trainlen = 200;
n = 8; %number of weights
const = qammod((0:1:M-1),M);
trainSig=tx(1:trainlen);

%LMS decision-feedback equalizer
nfwd = 16;
nfbk = 12;
dfeLMS = dfe(nfwd,nfbk,lms(mu));
dfeLMS.SigConst = const; % Set signal constellation.
dfeLMS.ResetBeforeFiltering = 0;

%RLS decision-feedback equalizer
dfeRLS = dfe(nfwd,nfbk,rls(0.99,0.9));
dfeRLS.SigConst = const; % Set signal constellation.
dfeRLS.ResetBeforeFiltering = 0;

%lms,linear
%trainMSG = reshape(de2bi(tx(1:trainlen),log2(M)),[],1);
linLMS = lineareq(n, lms(mu)); % Create an equalizer object.
linLMS.SigConst = const; % Set signal constellation.
linLMS.ResetBeforeFiltering = 0;

%rls,linear
linRLS = lineareq(n, rls(1,0.1)); % Create an equalizer
object.
linRLS.SigConst = const; % Set signal constellation.
linRLS.ResetBeforeFiltering = 0;
```

```

%Again, linRLS is picked.

%delay = (numRefTap-1)/eqobj.nSampPerSym;
[y,eqlSig] = equalize(linRLS,rx,trainSig); % Equalize.
eqlMSG =
gamdemod(eqlSig,M,'UnitAveragePower',true,'OutputType','bit');
eqlMSG_soft = gamdemod(eqlSig,M,'OutputType','approxllr', ...
    'UnitAveragePower',true,'NoiseVariance',noise_var);
%decodedMSG = dec(eqlMSG_soft);
decodedMSG =
vitdec(eqlMSG_soft,trel,traceBack,'cont','unquant');

% Compute and store the BER for this iteration

rxMSG = bi2de(reshape(rxMSG,log2(M),[]).');
%decodedMSG = bi2de(reshape(decodedMSG,log2(M),[]).');
[zzz, berVec(i,j)] = biterr(msg, rxMSG); % We're interested
in the BER, which is the 2nd output of BITERR

[zzz, eqlBerVec(i,j)] = biterr(bits(trainlen+1:end-traceBack),
decodedMSG(trainlen+traceBack+1:end)); %For Conv Code
[zzz, eqlBerVec(i,j)] = biterr(bits(trainlen+1:end-traceBack),
decodedMSG(trainlen+traceBack+1:end));

Except for getting rid of the training part, we also need to deal with delay between input and output caused
by convolution, which is equal to traceback in this case because the rate is 1/2.

% The best result we got is 8-QAM, the bit rate is improved
while
% keeping the BER below e-6.

Bit_Rate = (log2(M)*nSym - trainlen) / 1000;

end % End SNR iteration
end % End numIter iteration
time = toc;
time

time =

5.9601

```

## Plots

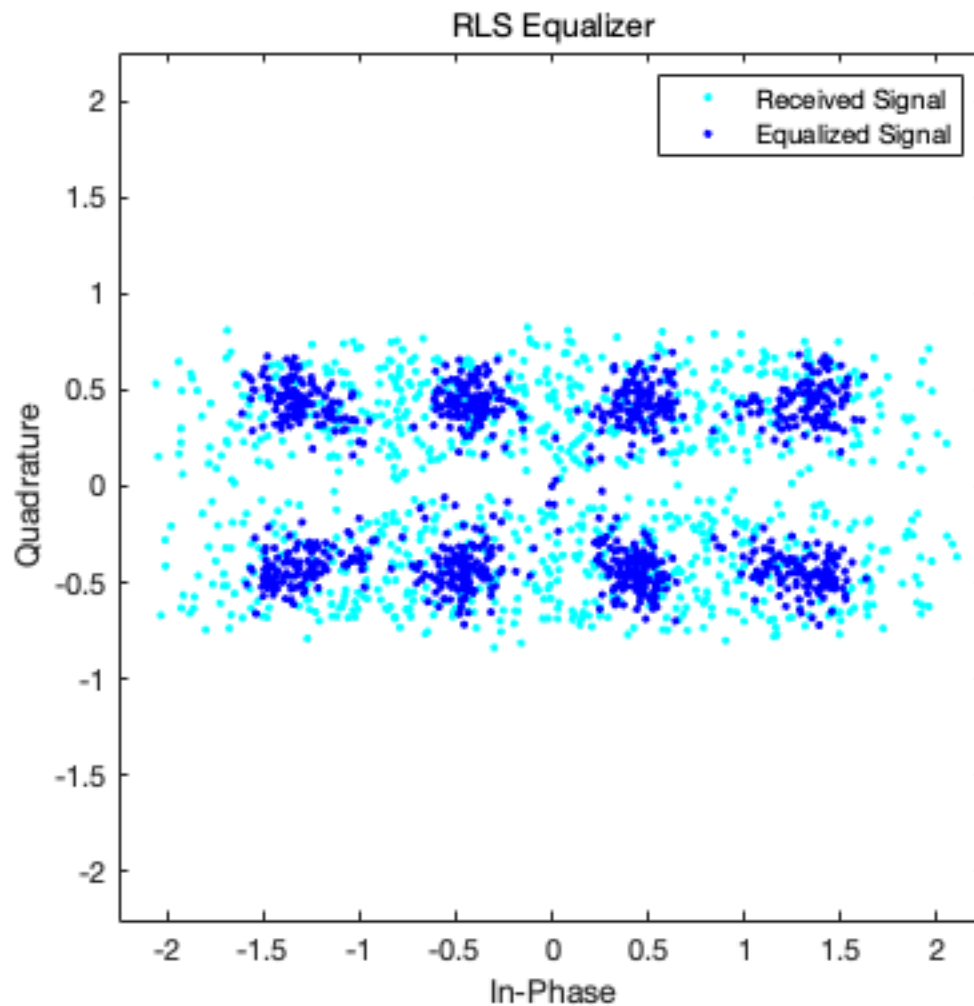
Played with M-ary QAM Scatter plot to see how equalizers work on

```

h = scatterplot(rx,1,0,'c. ');
hold on
scatterplot(y,1,0,'b.',h)
legend('Received Signal','Equalized Signal')
title('RLS Equalizer')

```

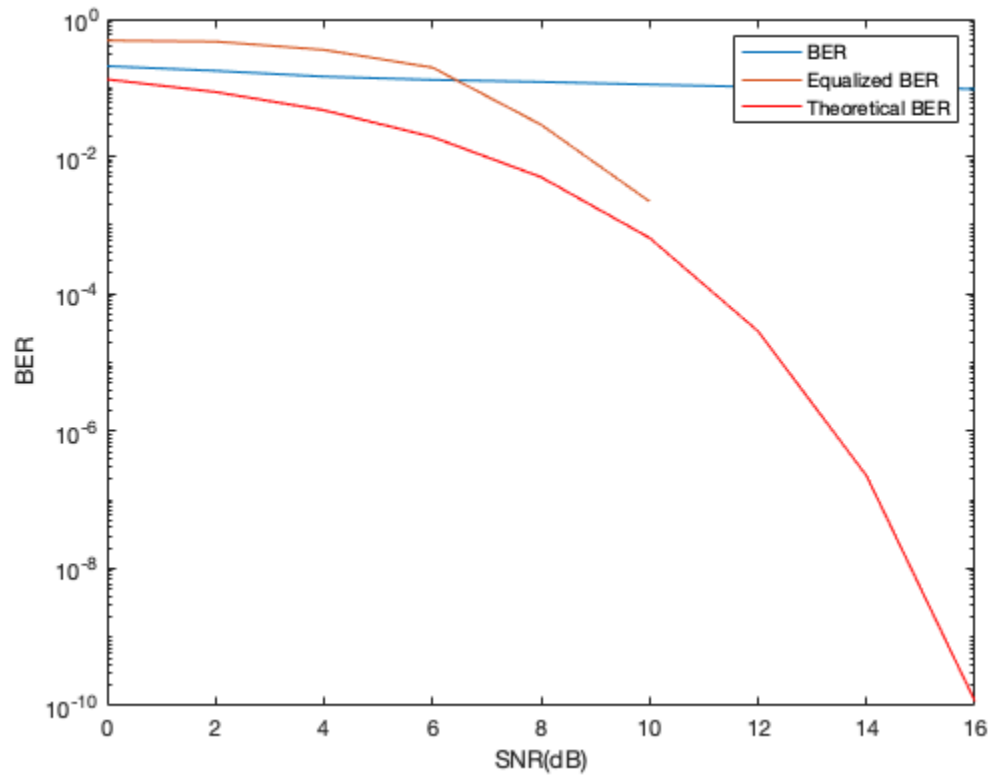
```
hold off  
%
```



Compute and plot the mean BER. The result at 12dB is 0. Which fits the requirement.

```
figure;  
ber = mean(berVec,1);  
eqlBer = mean(eqlBerVec,1);  
semilogy(SNR_Vec, ber);  
hold on  
semilogy(SNR_Vec, eqlBer);  
  
berTheory = berawgn(SNR_Vec, 'qam', M);  
hold on  
semilogy(SNR_Vec, berTheory, 'r')  
xlabel("SNR(dB)")  
ylabel("BER")
```

```
legend('BER', 'Equalized BER', 'Theoretical BER')
```



*Published with MATLAB® R2018b*