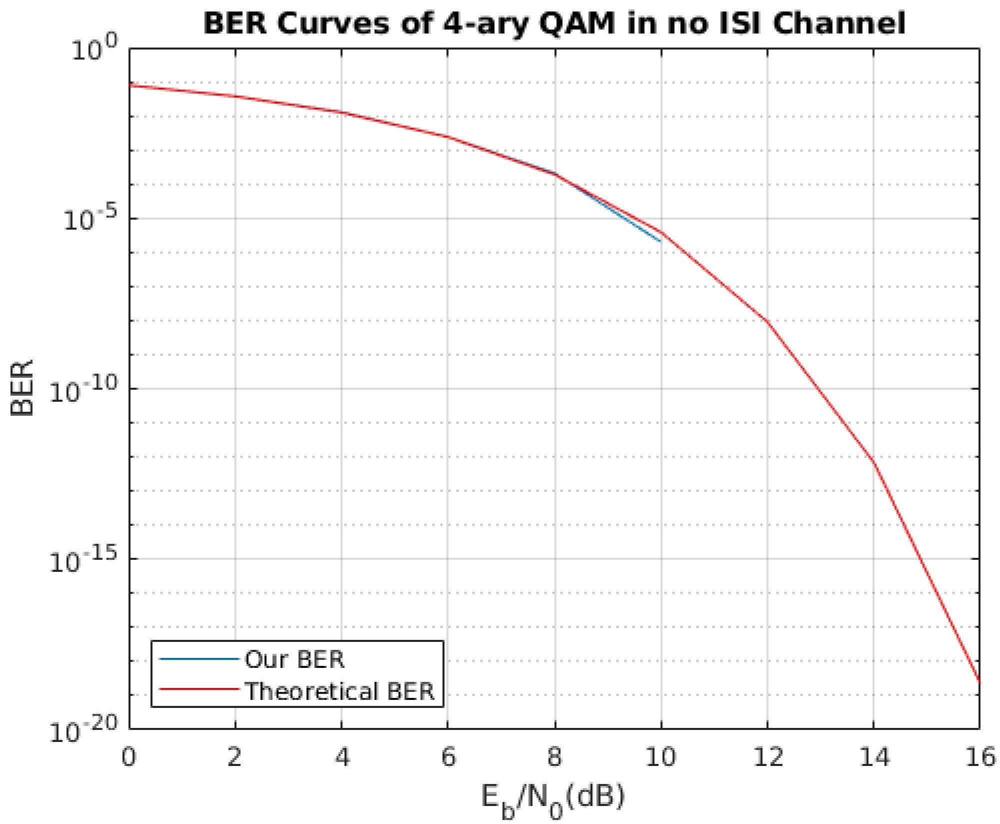
Table of Contents

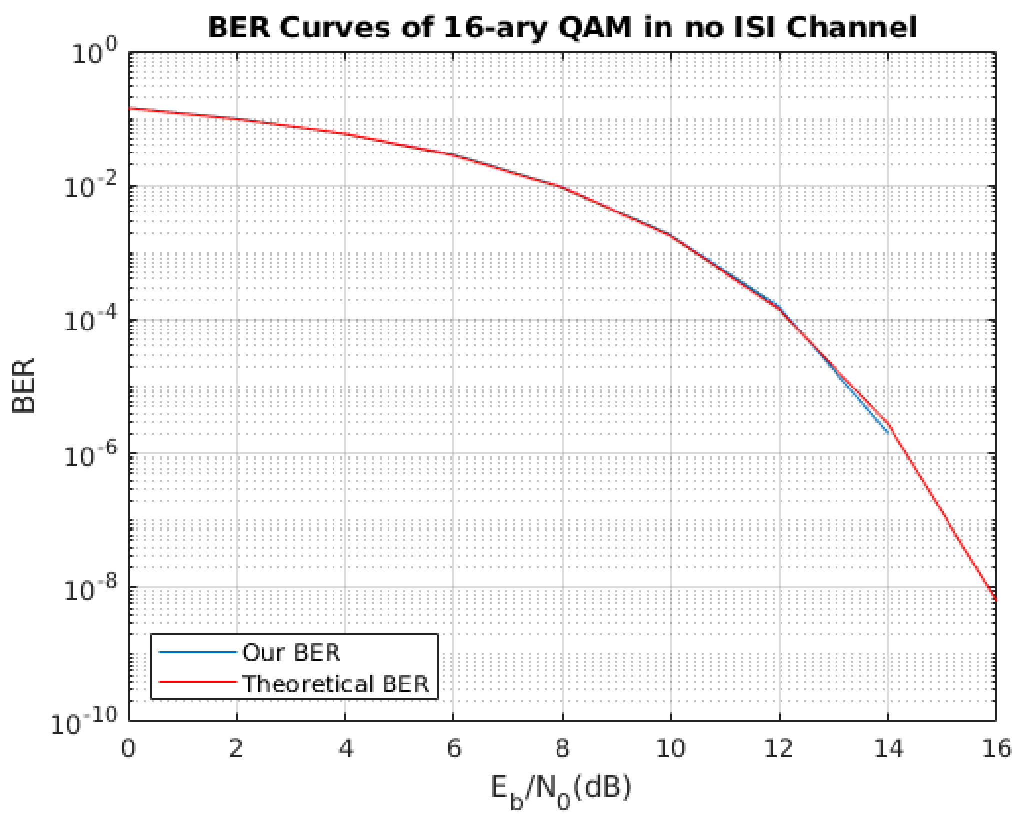

```
%
% Communication Theory Project
% Group: Shifra, Jonny, & Guy
%
% Part 1
```

Section 1: Transmitting over a channel with no ISI.

```
% In this section we show our code pipeline working with no ISI in the
% channel. We produce two ber graphs, one with 4-ary QAM and the other
 with
% 16-ary QAM, showing them aligning with the theoretical BERAWGN
 curve.

% It took us a considerable amount of time to figure out how to scale
 our
% noise. We first believed that the noise addition was a single
 formula,
% yet after observing that our results did not align with the
 theoretical ber,
% we did more research and came with the piecewise equation that can
 be seen in
% section 2. In this section, because M>2, we can only look at the
 relevant
% part of the piecewise equation.

% Parameters:
% We made the number of iterations large so that we could see the ber
 at
% 12 SNR and confirm that it meets the specifications.
numIter = 25;
n_sym = 10000;    % The number of symbols per packet
SNR_Vec = 0:2:16;
len_SNR = length(SNR_Vec);

% The M-ary number. We will run it for 4-ary and 16-ary QAM.
M = [4, 16];

for m = M
    % Create a vector to store the BER computed during each iteration
    berVec = zeros(numIter, len_SNR);
```

```matlab
    % Running the simulation:
    for i = 1:numIter

        % message to transmit
        bits = randi(2,[n_sym*log2(m), 1])-1;
        msg = bits2msg(bits, m);

        for j = 1:len_SNR % one iteration at each SNR Value

            tx = qammod(msg, m); % QAM modulation

            txChan = tx; % in this case we have no channel ISI.

            % Add AWGN
            noise_addition = round(10*log10(log2(m)));
            tx_noisy = awgn(txChan, noise_addition
+SNR_Vec(j), 'measured');

            % de-modulation
            rx = qamdemod(tx_noisy, m);  % QAM
            rx_msg = msg2bits(rx, m);

            % Compute and store the BER for this iteration
            % We're interested in the BER, the 2nd output of BITERR
            [~, berVec(i,j)] = biterr(bits,rx_msg);

        end  % End SNR iteration
    end      % End numIter iteration


    % Compute and plot the mean BER
    ber = mean(berVec,1);
    figure
    semilogy(SNR_Vec, ber, 'DisplayName', 'Our BER')

    % Compute the theoretical BER for this scenario
    berTheory = berawgn(SNR_Vec, 'qam', m); % QAM

    hold on
    semilogy(SNR_Vec, berTheory, 'r', 'DisplayName', 'Theoretical
 BER')
    xlabel('E_b/N_0(dB)');  ylabel('BER');
    legend('Location','southwest')
    grid
    title(['BER Curves of ' num2str(m) '-ary QAM in no ISI Channel']);
end
```


BER Curves of 4-ary QAM in no ISI Channel
BER
E_b/N_0(dB)
Our BER
Theoretical BER
10^0
10^{-5}
10^{-10}
10^{-15}
10^{-20}
0 2 4 6 8 10 12 14 16


BER Curves of 16-ary QAM in no ISI Channel
BER
E_b/N_0(dB)
Our BER
Theoretical BER
10^0
10^{-2}
10^{-4}
10^{-6}
10^{-8}
10^{-10}
0 2 4 6 8 10 12 14 16

Section 2: Transmitting over a channel with moderate ISI

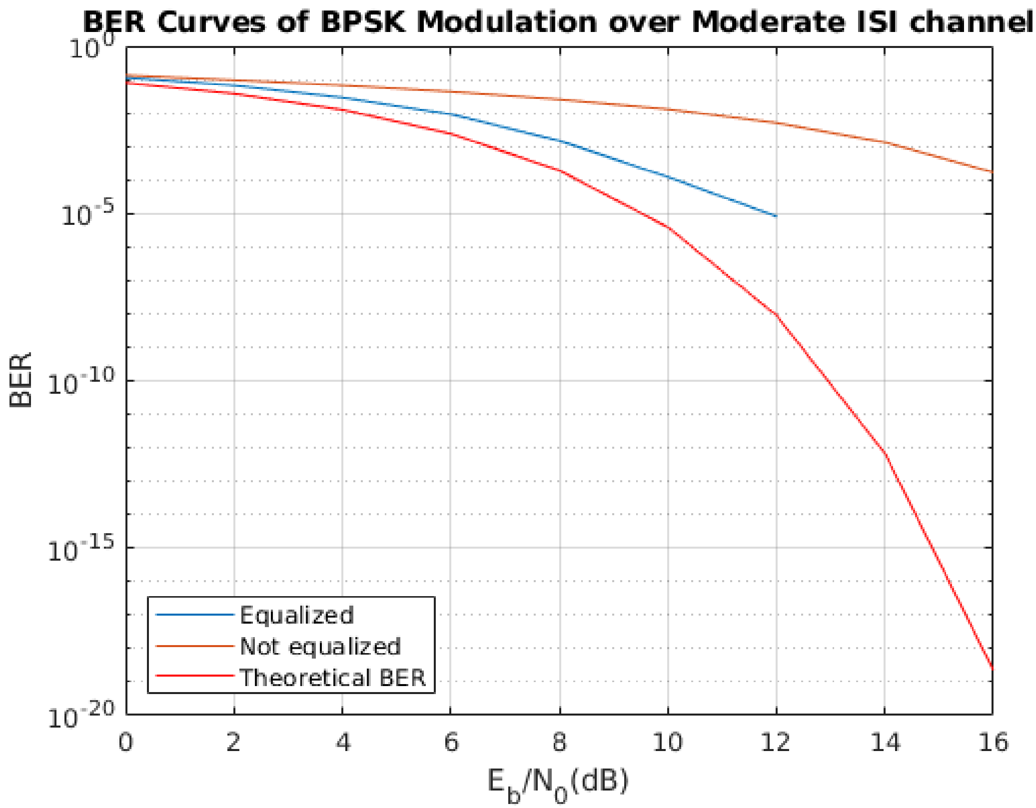
```
% In this section we graph a ber curve for BPSK with moderate ISI.
% To handle the ISI, we used an adaptive filter as an equalizer.
% After trying several combinations of the parameters for the adaptive
% algorithm and equalizer, we decided to write a grid search function
 to find
% the optimal parameters: equalizer weights, training symbol length,
 and
% adaptive filter hyperparameters (stepsize for lms and the forget
 factor
% for rls).

% Although we tried to include reference taps, we were not able to get
 it
% this feature to work (as of yet, at least) and so we have decided to
% avoid it altogether. As such we believe that the results still have
 room
% for improvement, and we hope to utilize this margin of possible
 improvement
% in part 2 of this project.

% In this section we used the optimal parameters found for an rls
 adaptive
% algorithm and a linear equalizer, seeing as they more than satisfy
 the
% requirement for 10^-4 ber at 12 SNR.

% Note that this script is written in the most general sense, so that
 we could
% easily alter our choices for M values, modulation types, adaptive
 algorithms,
% and equalizer objects.


% Parameters:
% We made the number of iterations large so that we could see the ber
 at
% 12 SNR and confirm that it meets the specifications.
numIter = 25;
n_sym = 10000;    % The number of symbols per packet
SNR_Vec = 0:2:16;
len_SNR = length(SNR_Vec);

% The M-ary number. Two corresponds to binary modulation.
M = 2;

% Modulation type
%  - 1 = PAM
%  - 2 = QAM
%  - 3 = PSK
```

```
modulation = 2;

% Channel to use
%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.
% 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 do plot(chan)

% Number of training symbols (max=len(msg)=1000)
numTrain = 175;

% Adaptive Algorithm
%   - 0 = varlms
%   - 1 = lms
%   - 2 = rls
adaptive_algo = 2;

% Equalizer
%   - 0 = lineareq
%   - 1 = dfe
equalize_val = 0;

% equalizer hyperparameters
n_weights = 6;
n_weights_feedback = 7;

%numRefTap = 2;

stepsize = 0.005;
forgetfactor = 1; % between 0 and 1


% Building the equalizer:
% adaptive filter algorithm
if isequal(adaptive_algo, 0)
    adaptive_algo = varlms(stepsize,0.01,0,0.01);
elseif isequal(adaptive_algo, 1)
    adaptive_algo = lms(stepsize);
else
    adaptive_algo = rls(forgetfactor);
end

% equalizer object
if isequal(equalize_val, 0)
    eqobj = lineareq(n_weights, adaptive_algo); % like FIR
    %eqobj.RefTap = numRefTap;
```

```
else
    eqobj = dfe(n_weights, n_weights_feedback, adaptive_algo); % like
 IIR
end


% Create a vector to store the BER computed during each iteration
berVec_no_eq = zeros(numIter, len_SNR);
berVec_eq = zeros(numIter, len_SNR);

% Running the simulation:
for i = 1:numIter

    % message to transmit
    bits = randi(2,[n_sym*log2(M), 1])-1;
    msg = bits2msg(bits, M);

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

        % modulation
        if isequal(modulation, 1)
            tx = pammod(msg, M);  % PAM modulation
        elseif isequal(modulation, 2)
            tx = qammod(msg, M);  % QAM modulation
        else
            tx = pskmod(msg, M);  % PSK modulation
        end

        % Sequence of Training Symbols
        trainseq = tx(1:numTrain);

        % transmit (convolve) through channel
        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


        % Adding AWGN. First need to convert from EbNo to SNR.
        if isequal(M, 2)
            noise_addition = 3;
        else
            noise_addition = round(10*log10(log2(M)));
        end
        tx_noisy = awgn(txChan, noise_addition
+SNR_Vec(j), 'measured');

        yd = equalize(eqobj,tx_noisy,trainseq);
```

```
        % de-modulation
        if isequal(modulation, 1)
            rx = pamdemod(yd, M);  % PAM
        elseif isequal(modulation, 2)
            rx = qamdemod(yd, M);  % QAM
            rx2 = qamdemod(tx_noisy,M);
        else
            rx = pskdemod(yd, M);  % PSK
        end

        rx_msg = msg2bits(rx, M);
        rx_msg2 = msg2bits(rx2,M);

        % Compute and store the BER for this iteration
        % We're interested in the BER, which is the 2nd output of
 BITERR
        [~, berVec_eq(i,j)] = biterr(bits(numTrain:end),
 rx_msg(numTrain:end));
        [~, berVec_no_eq(i,j)] = biterr(bits,rx_msg2);

    end  % End SNR iteration
end      % End numIter iteration


% Compute and plot the mean equalizer BER
ber_eq = mean(berVec_eq,1);
ber_no_eq = mean(berVec_no_eq,1);
figure
semilogy(SNR_Vec, ber_eq, 'DisplayName', 'Equalized')
hold on
semilogy(SNR_Vec, ber_no_eq, 'DisplayName', 'Not equalized')

% Compute the theoretical BER for this scenario
% NOTE: there is no theoretical BER when you have a multipath channel
if isequal(modulation, 1) || (M<4) % if M<4, qam berawgn is pam
 berawng
    berTheory = berawgn(SNR_Vec, 'pam', M); % PAM
elseif isequal(modulation, 2)
    berTheory = berawgn(SNR_Vec, 'qam', M); % QAM
else
    berTheory = berawgn(SNR_Vec, 'psk', M, 'nodiff'); % PSK
end

hold on
semilogy(SNR_Vec, berTheory, 'r', 'DisplayName', 'Theoretical BER')
xlabel('E_b/N_0(dB)');  ylabel('BER');
legend('Location','southwest')
grid
if isequal(M,2)
    title('BER Curves of BPSK Modulation over Moderate ISI channel')
elseif isequal(modulation, 1) % if M<4, qam berawgn is pam berawng
    title(['BER Curves of ' num2str(M) '-ary PAM in Moderate ISI
 channel'])
elseif isequal(modulation, 2)
```

```
    title(['BER Curves of ' num2str(M) '-ary QAM in Moderate ISI
 channel'])
else
    title(['BER Curves of ' num2str(M) '-ary PSK in Moderate ISI
 channel'])
end
```



Section 3: Grid Searching to find the optimal equalizer parameters

```
% In this section, we ran grid search several times, each time
 updating the
% first 2 parameters of the gridSearch, which correspond to the type
 of
% adaptive filter and equalizer object the gridSearch fuction should
 use. The
% overall results can be found after the function call.

% For the sake of time, the grid search function will not run in the
% published matlab code. Instead, we included the results below from
 the four
% iterations that we ran.

adaptive_params = 0:0.005:0.2;
weights = 5:9;
```

```
feedback_weights = 5:9;
trains = 25:25:350;
%[best_adap_val, best_n_weight, best_feedback_weight, best_train_len]
 = ...
%     gridSearch(1, 1, adaptive_params, weights, feedback_weights,
 trains);

% GRID SEARCH RESULTS:
% lms and lineareq: adap_val=0.005, n_weight=8, best_train_len=75
% rls and lineareq: adap_val=1,     n_weight=6, best_train_len=175
% lms and def: adap_val=0.01, n_weight=8,  feedback_weights=5,
 train_len=25
% rls and def: adap_val=0.865, n_weight=6, feedback_weights=5,
 train_len=25
```

Section 4: Helper Functions

```
% Here are the functions we used in the above parts:
% 1. Grid Search - the function used in our equalizer parameter
 search.
% 2. msg2bits and bits2msg - functions used in our bits-message
 conversions.

function [optimal_adaptive_algo, optimal_n_weight, ...
     optimal_feedback_weight, optimal_train_len] = ...
     gridSearch (adaptive_algo, equalizer,
 adaptive_algo_contenders, ...
     n_weight_contenders, feedback_weight_contenders,
 num_train_contenders)
   % Performs a grid search across the specified equalizer
 parameters,
   % assuming BPSK modulation over a channel with moderate ISI,
 returning
   % the optimal values found.

   % hard-coded parameters
   M = 2;
   nSym=3000;
   modulation = 2;
   chan = [1 .2 .4];

   % optimal values found
   optimal_adaptive_algo = -1;
   optimal_n_weight = -1;
   optimal_feedback_weight = -1;
   optimal_train_len = -1;

   best_ber = 1; % best ber rate found so far

   % exahust through all the parameter combinations!
   for cur_adaptive_algo = adaptive_algo_contenders
       for cur_n_weight = n_weight_contenders
           for cur_train = num_train_contenders
```

```
                for cur_fb_weight = feedback_weight_contenders

                    fprintf(['Current run: adaptive: %f, weight: %d,
', ...
                        'feedback_weight: %d, train: %d\n'], ...
                        cur_adaptive_algo, cur_n_weight, ...
                        cur_fb_weight, cur_train);

                    % adaptive filter algorithm
                    if isequal(adaptive_algo, 0)
                        adap_filt =
varlms(cur_adaptive_algo,0.01,0,0.01);
                    elseif isequal(adaptive_algo, 1)
                        adap_filt = lms(cur_adaptive_algo);
                    else
                        adap_filt = rls(cur_adaptive_algo);
                    end

                    % Equalizer Object
                    if isequal(equalizer, 0)
                        eqobj = lineareq(cur_n_weight,adap_filt);
                    else
                        eqobj =
dfe(cur_n_weight,cur_fb_weight,adap_filt);
                    end

                    % message to transmit
                    bits = randi(2,[nSym*log2(M), 1])-1;
                    msg = bits2msg(bits, M);

                    % modulation
                    if isequal(modulation, 1)
                        tx = pammod(msg, M);  % PAM modulation
                    elseif isequal(modulation, 2)
                        tx = qammod(msg, M);  % QAM modulation
                    else
                        tx = pskmod(msg, M);  % PSK modulation
                    end

                    % Sequence of Training Symbols
                    trainseq = tx(1:cur_train);

                    % transmit (convolve) through channel
                    txChan = filter(chan,1,tx);  % Apply the channel.

                    % Convert from EbNo to SNR (hardcoding our SNR
req)
                    txNoisy = awgn(txChan, 3+12, 'measured'); % Add
AWGN

                    yd = equalize(eqobj,txNoisy,trainseq);

                    % de-modulation
                    if isequal(modulation, 1)
```

```
                    rx = pamdemod(yd, M);  % PAM
                elseif isequal(modulation, 2)
                    rx = qamdemod(yd, M);  % QAM
                else
                    rx = pskdemod(yd, M);  % PSK
                end
                rx_msg = msg2bits(rx, M);

                % Compute and store the BER
                [~, ber] = biterr(bits(cur_train:end),
 rx_msg(cur_train:end));

                if ber ~= 0 && ber < best_ber
                    best_ber = ber;
                    optimal_train_len = cur_train;
                    optimal_adaptive_algo = cur_adaptive_algo;
                    optimal_n_weight = cur_n_weight;
                    optimal_feedback_weight = cur_fb_weight;
                    fprintf('New best ber achieved: %e.\n', ber);
                end
            end
        end
    end
end
% return optimal parameter combination
fprintf('Best score: %f\n', best_ber);
fprintf(['Current run: adaptive: %f, weight: %d, ', ...
    'feedback_weight: %d, train: %d\n'], ...
    optimal_adaptive_algo, optimal_n_weight, ...
    optimal_feedback_weight, optimal_train_len);
end


function [msg] = bits2msg(bits, M)
    % Convert the message from bits into the correct integer values
    % based on the inputted M-ary modulation.
    % NOTE: M has to be a multiple of 2.

    % The length of bits that will be converted into decimal.
    len = log2(M);

    msg = zeros(size(bits,1)/len, 1);

    for i = 1:size(bits,1)/len
        msg(i) = bi2de(bits(1+(i-1)*len : 1+(i-1)*len + (len-1))');
    end

end


function [bits] = msg2bits(msg, M)
    % Convert the message from integers into the bit values
    % based on the inputted M-ary modulation.
    % NOTE: M has to be a multiple of 2.
```

```
    % The length of bits that will be converted into decimal.
    len = log2(M);

    bits = zeros(1, size(msg,1)*len);

    for i = 1:size(msg,1)
        bits(1+(i-1)*len:1:1+(i-1)*len + (len-1)) = de2bi(msg(i),
 len);
    end
    bits = bits'; % needed to keep in same format as the rest of the
 code
end
```

Published with MATLAB® R2018b