ECE 132A Project W24

Caden Davis and Nathan Nguyendinh

This MATLAB livescript is meant to simulate building up to a multiple-input multiple-output (MIMO) communication system. We start by simulating a simpler SISO communication system using QPSK modulation with a square-root raised cosine pulse shaping.

Simulation Setup

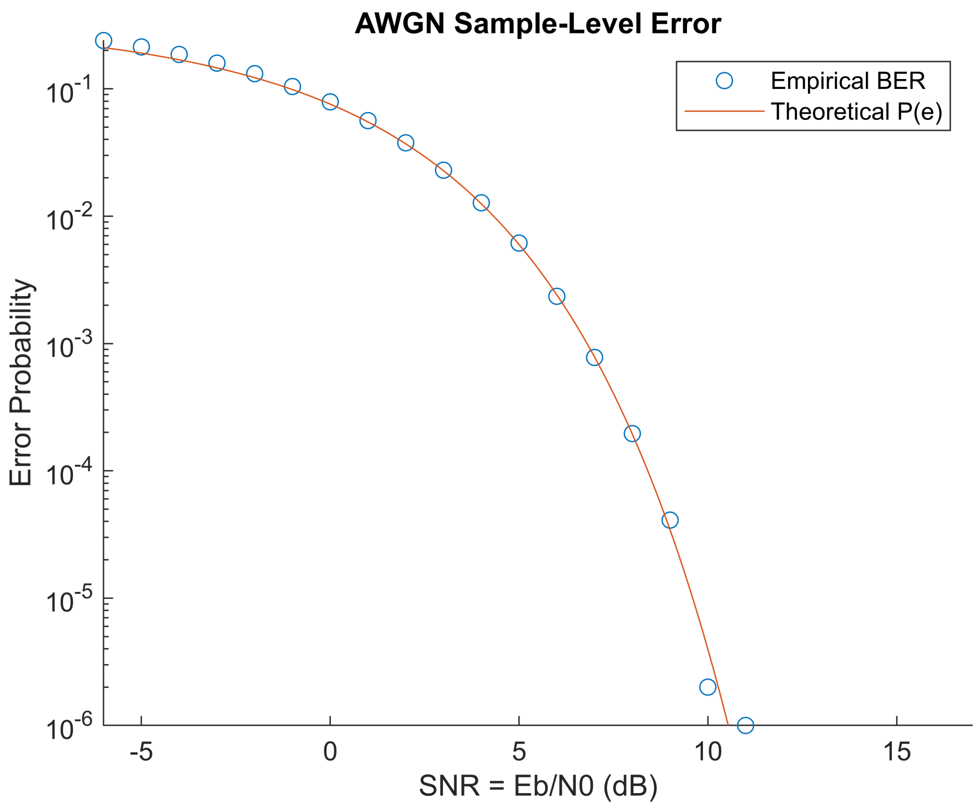
```
Nbits = 1e6; % number of bits
Rs = 1; % symbol rate
Fs = 4; % sampling rate
sps = Fs/Rs; % samples per symbol
EsN0dBs = -3:1:20;
EbN0dBs = EsN0dBs-3;
N0s = 10.^(-EsN0dBs./10);

EsN0dBs_fine = -3:0.1:20; % Use for plotting theoretical P(e)'s
EbN0dBs_fine = EsN0dBs_fine - 3;
N0s_fine = 10.^(-EsN0dBs_fine./10);
% Generate RRC filter
hrrc = rrc((-6:1/sps:6)',0.2,1);
% Tx
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);
```

Single-Input Single-Output (SISO)

AWGN Channel, Sample Level

Now that we have our modulated symbols, we are going to simulate the waveforms travelling through an additive white gaussian noise (AWGN) channel and observe the bit error rates once we demodulate the symbols. Our theoretical bit error should be a function of our SNR, following the equation rho = Q(sqrt(Eb/N0)), P(e) = rho * (1 - 0.5 * rho). This equation comes from the class notes. We use N0 instead of N0/2 because we have both the I and Q channel noise for QPSK. We have plotted both our simulated error from multiple values of the SNR as well as the theoretical error, and we see that they correctly match.

```
% Convolve with RRC
txvec = conv(upsample(symvec,sps),hrrc,'same');
bers1 = zeros(length(N0s), 1);

for i = 1:length(N0s)
    N0 = N0s(i);
    sig2 = N0*Fs/2;
    % Add noise
    noisevec = sqrt(sig2)*complex(randn(size(txvec)),randn(size(txvec)));
    % Rx
    rxvec = txvec + noisevec;
    % MF
    mfout = conv(rxvec, hrrc, 'same'); % matched filter rxvec
    % Downsample to convert back to symbol rate
    mfsamp = mfout(1:sps:end); % sample mfout at every sps-th sample
    [detsymvec,detbitvec] = qpskdetect(mfsamp);
    ser1 = sum(detsymvec~=symvec)/(Nbits/2);
    bers1(i) = sum(xor(detbitvec,bitvec))/Nbits;
end

clf;
figure;
hold on
scatter(EbN0dBs, bers1);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('AWGN Sample-Level Error');
set(gca,'yscale','log')

% Gray-Mapped QPSK BER Formula
Q = qfunc(sqrt(1 ./ N0s_fine));
P_e = Q .* (1 - 0.5 * Q);
% 1 instead of 1/2 because N0 has both
% I and Q dimensions -- N0/2 per dimension
plot(EbN0dBs_fine, P_e);
ylim([1e-6 inf])
xlim([-inf, inf])
legend("Empirical BER", "Theoretical P(e)")

hold off
```

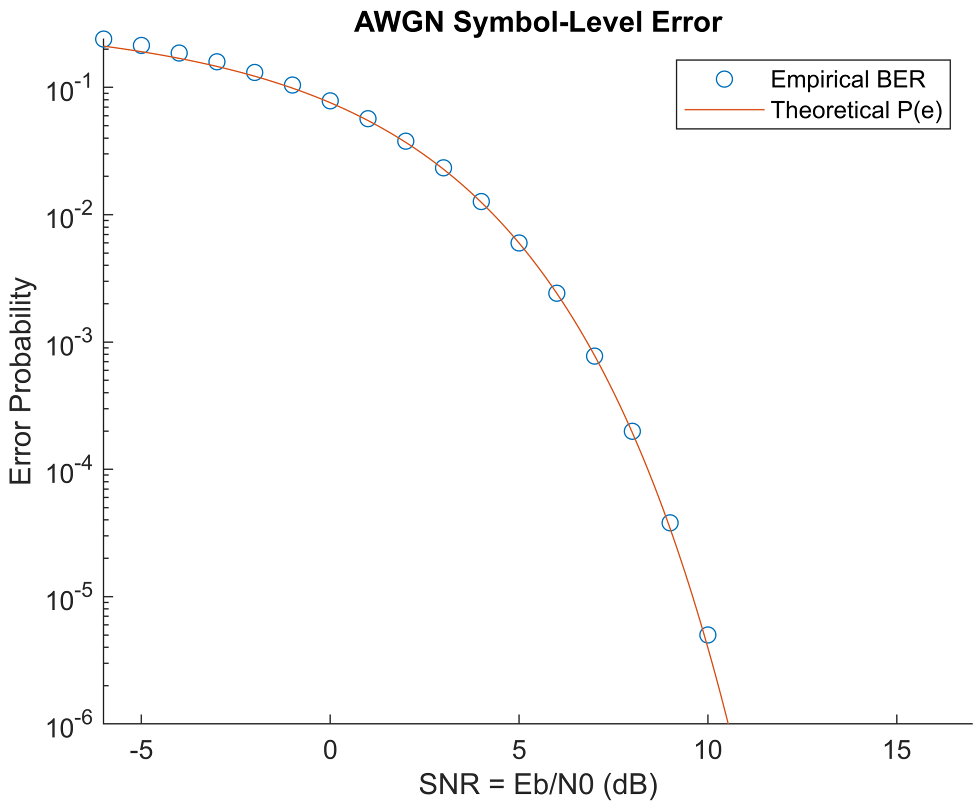


AWGN Channel, Symbol Level

Note that in the previous section, we simulated our transmission on the sample level, but we can also simulate the same transmission on the symbol level instead, as shown below. We can observe that these two representations are equivalent because we get the same simulated result that matches our theoretical equation.

```
bers2 = zeros(length(N0s), 1);

for i = 1:length(N0s)
    N0 = N0s(i);
    symnoise = sqrt(N0/2)*complex(randn(size(symvec)),randn(size(symvec)));
    rxsymvec = symvec + symnoise;
    [detsymvec2,detbitvec2] = qpskdetect(rxsymvec);
    ser2 = sum(detsymvec2~=symvec)/(Nbits/2);
    bers2(i) = sum(xor(detbitvec2,bitvec))/Nbits;
end

clf;
figure;
hold on
scatter(EbN0dBs, bers2);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('AWGN Symbol-Level Error');
```

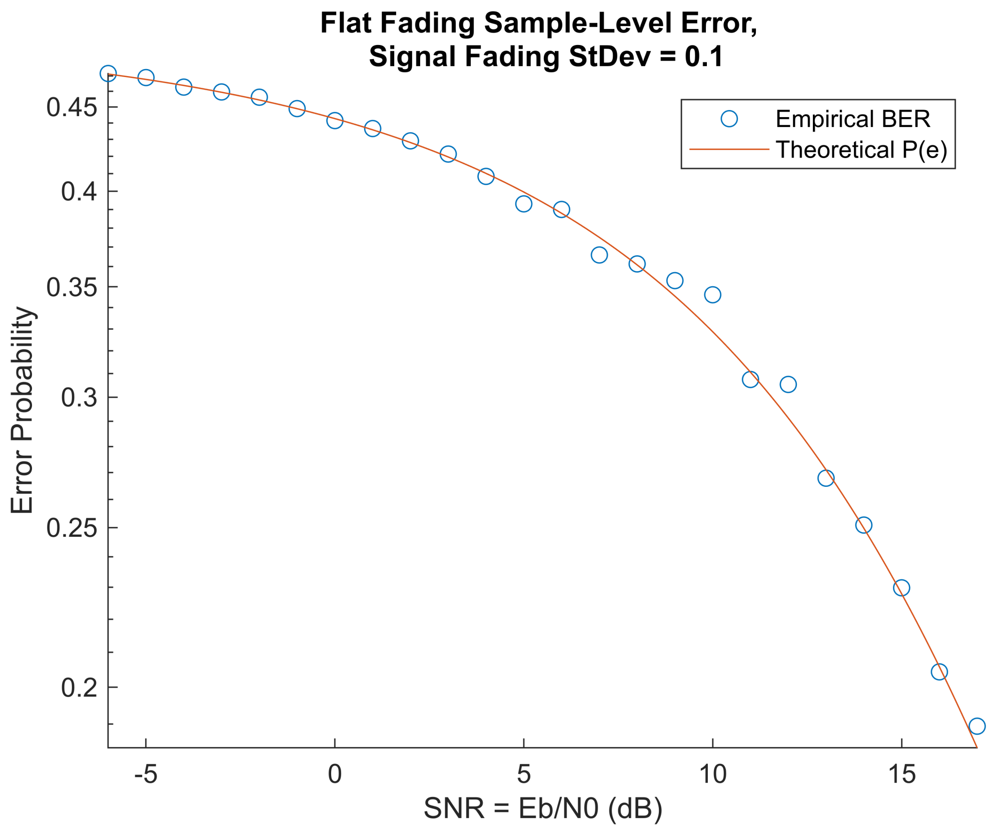
```
set(gca,'yscale','log')
% % compare sample-level to symbol-level sims

% Gray-Mapped QPSK BER Formula
Q = qfunc(sqrt(1 ./ N0s_fine));
P_e = Q .* (1 - 0.5 * Q);
plot(EbN0dBs_fine, P_e);
ylim([1e-6 inf])
xlim([-inf, inf])
legend("Empirical BER", "Theoretical P(e)")

hold off
```

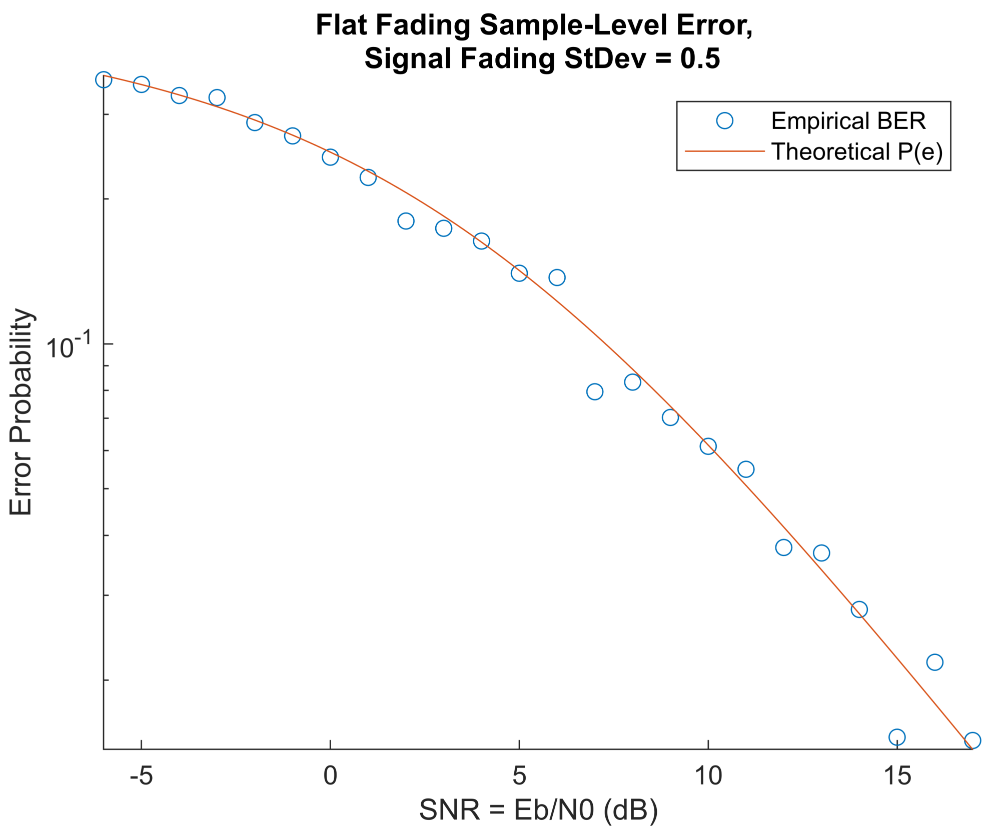


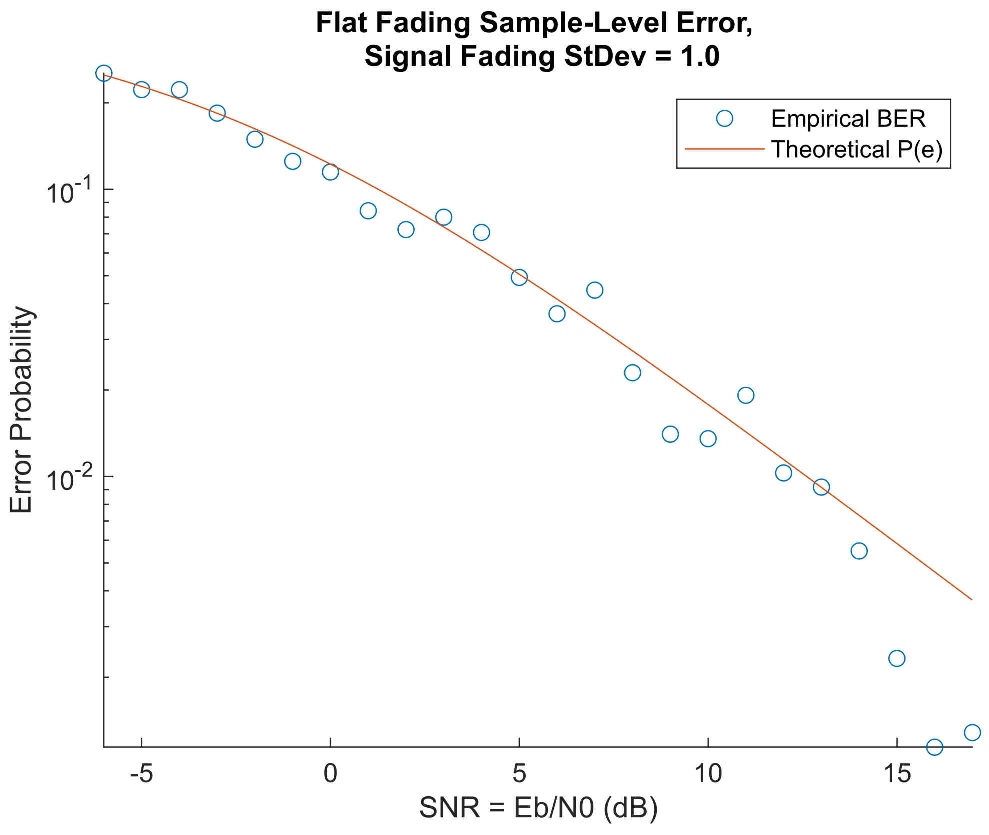
Slow Flat Fading Channel, Sample Level

For the next part of our simulation, we are going to simulate the entire tranmission being affected by the same fading (known as slow fading). We simulate this by multiplying our entire waveform by a complex Gaussian variable with zero mean and a given variance (flat fading). Note that as this fading coefficient's variance *decreases*, our channel becomes worse (for SNR) on average as the fading alpha tends to be closer to a magnitude of zero, but as the variance increases we see more variability in our BER. We still follow the same steps from before of adding noise, using matched filters, and calculating the bit error rate. With the addition of slow fading, our theoretical error rate is given by replacing Eb/N0 in the previous P(e) expression with |alpha|^2 * Eb/N0 where we use the expected value of |alpha|^2 from the complex Gaussian distribution. We can see from the plot that the simulations match up with this. Note that the high variance in our simulations is due to averaging over an insufficient number of fading realizations.

```
Nbits = 1e4;
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

% sample-level simulation
txvec = conv(upsample(symvec,sps),hrrc,'same');
bers3 = zeros(length(N0s), 1);

sigfads = [0.1 0.5 1];
for n = 1:3
    sigfad = sigfads(n);
    for i = 1:length(N0s)
        N0 = N0s(i);
        sig2 = N0*Fs/2;
        tmps = 0;
        tmpb = 0;
        nruns = 100;
        for k = 1:nruns
            noisevec = sqrt(sig2)*complex(randn(size(txvec)),randn(size(txvec)));
            fadingalpha = sigfad*complex(randn(1),rand(1));
            rxvecfad = fadingalpha * txvec + noisevec;
            mfout = conv(rxvecfad*exp(-1j*angle(fadingalpha)),hrrc,'same')/Fs; %
phase compensation
            mfsamp = mfout(1:sps:end);
            [detsymvec,detbitvec] = qpskdetect(mfsamp);
            tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
            tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
        end
        ser3 = tmps/nruns;
        bers3(i) = tmpb/nruns;
    end

    figure;
    hold on
    scatter(EbN0dBs, bers3);
    xlabel('SNR = Eb/N0 (dB)');
    ylabel('Error Probability');
    title(sprintf('Flat Fading Sample-Level Error, \n Signal Fading StDev = %.1f',
sigfad));
    set(gca,'yscale','log')

    avg_SNR = sigfad^2 ./ N0s_fine;
    P_e = 0.5 * (1 - sqrt(avg_SNR ./ (1.5 + avg_SNR)));
    plot(EbN0dBs_fine, P_e);
    ylim([-inf inf])
    xlim([-inf inf])
    legend("Empirical BER", "Theoretical P(e)")

    hold off
end
```



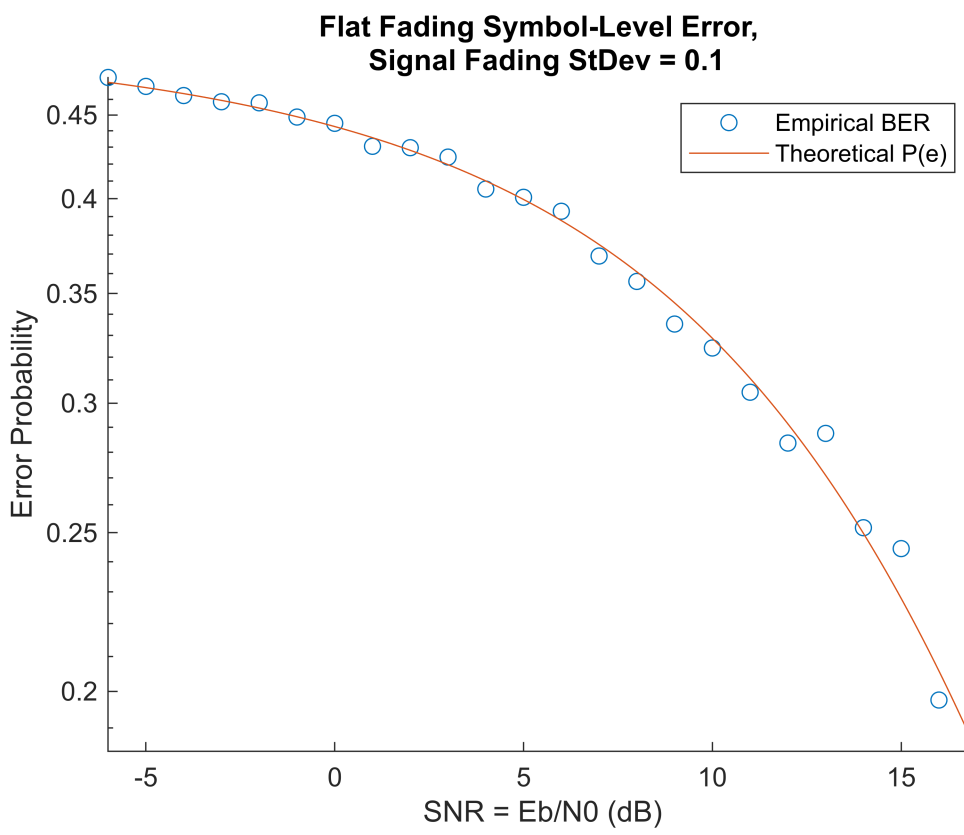

Flat Fading Sample-Level Error,
Signal Fading StDev = 0.5
Empirical BER
Theoretical P(e)
Error Probability
10^{-1}
-5
0
5
10
15
SNR = Eb/N0 (dB)

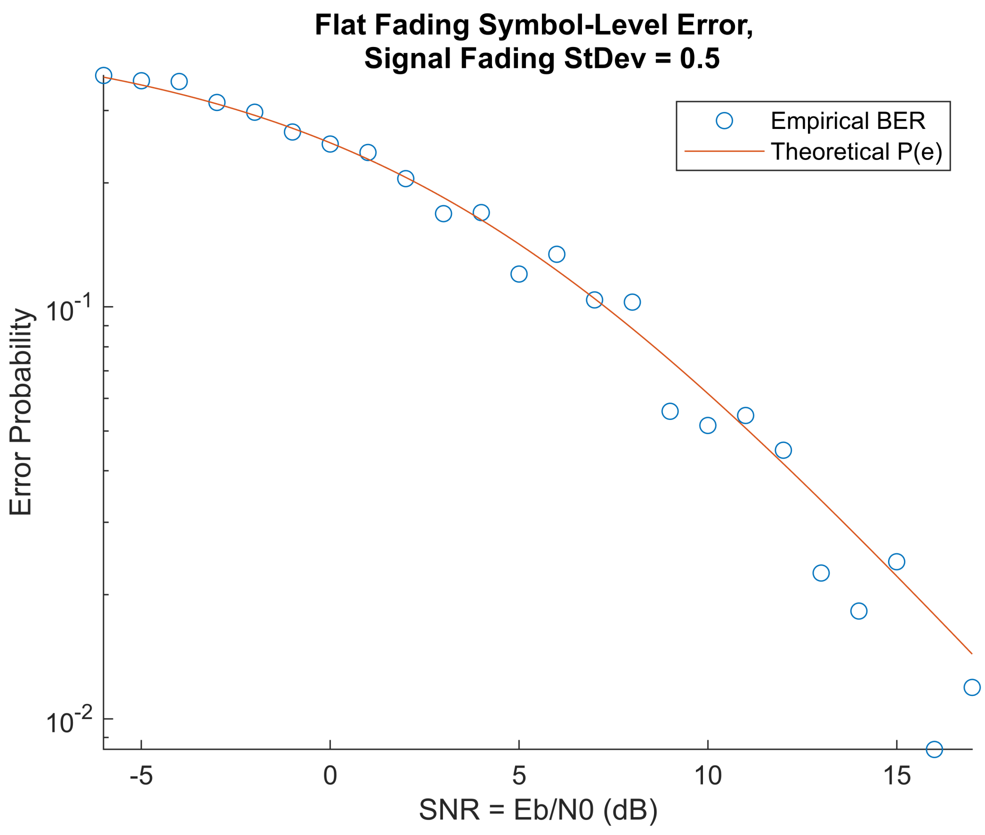

Flat Fading Sample-Level Error,
Signal Fading StDev = 1.0
Empirical BER
Theoretical P(e)
Error Probability
10^{-1}
10^{-2}
-5
0
5
10
15
SNR = Eb/N0 (dB)

Slow Flat Fading Channel, Symbol Level

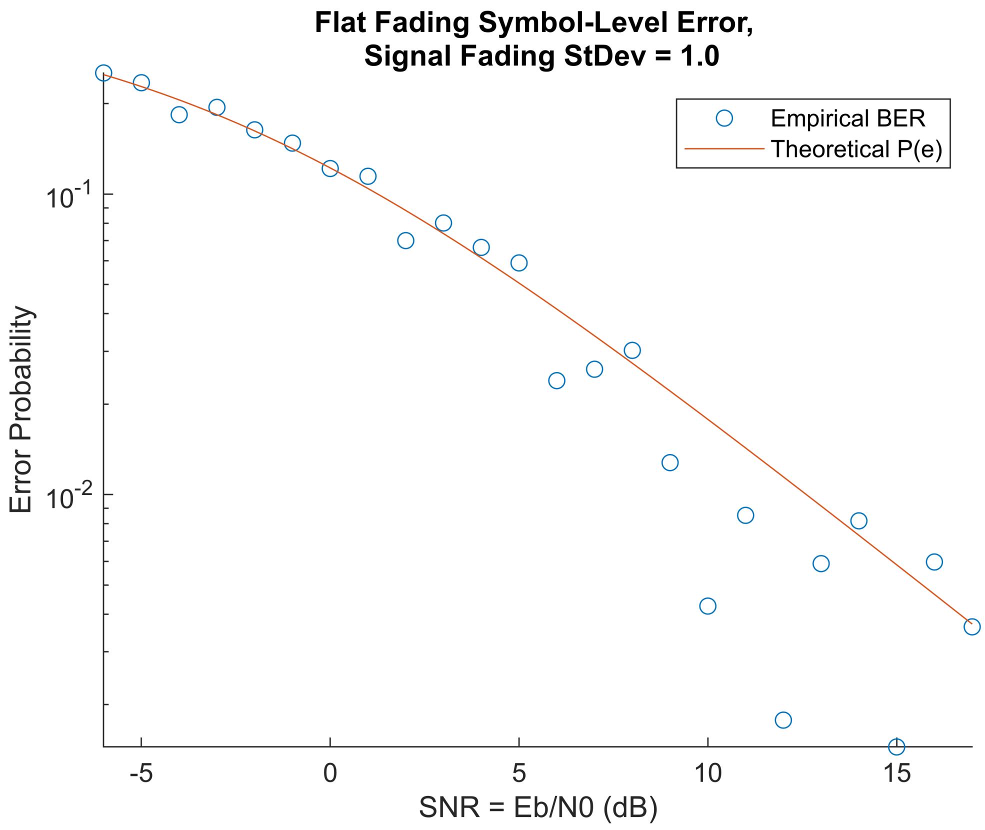
Like with the AWGN section, we can also simulate the transmission with slow fading at the symbol level. As we can see, the symbol level simulation here matches up with the same bit error plot as the sample level simulation from above.

```
Nbits = 1e4;
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

% symbol-level
txvec = conv(upsample(symvec,sps),hrrc,'same');
bers4 = zeros(length(N0s), 1);

sigfads = [0.1 0.5 1];
for n = 1:3
    sigfad = sigfads(n);
    for i = 1:length(N0s)
        N0 = N0s(i);
        sig2 = N0/2; % Remove Fs for symbol level
        tmps = 0;
        tmpb = 0;
        nruns = 100;
        for k = 1:nruns
            symnoise = sqrt(sig2)*complex(randn(size(symvec)),randn(size(symvec)));
            fadingalpha = sigfad*complex(randn(1),rand(1));
            rxsymvecfad = abs(fadingalpha) * symvec + symnoise; % abs because
assume phase compensated
            [detsymvec,detbitvec] = qpskdetect(rxsymvecfad);
            tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
            tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
        end
        ser4 = tmps/nruns;
        bers4(i) = tmpb/nruns;
    end

    figure;
    hold on
    scatter(EbN0dBs, bers4);
    xlabel('SNR = Eb/N0 (dB)');
    ylabel('Error Probability');
    title(sprintf('Flat Fading Symbol-Level Error, \n Signal Fading StDev = %.1f',
sigfad));
    set(gca,'yscale','log')

    avg_SNR = sigfad^2 ./ N0s_fine;
    P_e = 0.5 * (1 - sqrt(avg_SNR ./ (1.5 + avg_SNR)));
    plot(EbN0dBs_fine, P_e);
    ylim([-inf inf])
    xlim([-inf inf])
    legend("Empirical BER", "Theoretical P(e)")

    hold off
end
```





Flat Fading Symbol-Level Error,
Signal Fading StDev = 0.5
Empirical BER
Theoretical P(e)
Error Probability
10^-1
10^-2
-5
0
5
10
15
SNR = Eb/N0 (dB)



Flat Fading Symbol-Level Error,
Signal Fading StDev = 1.0
Empirical BER
Theoretical P(e)
Error Probability
10^-1
10^-2
-5
0
5
10
15
SNR = Eb/N0 (dB)

Multiple-Input Multiple-Output (MIMO)

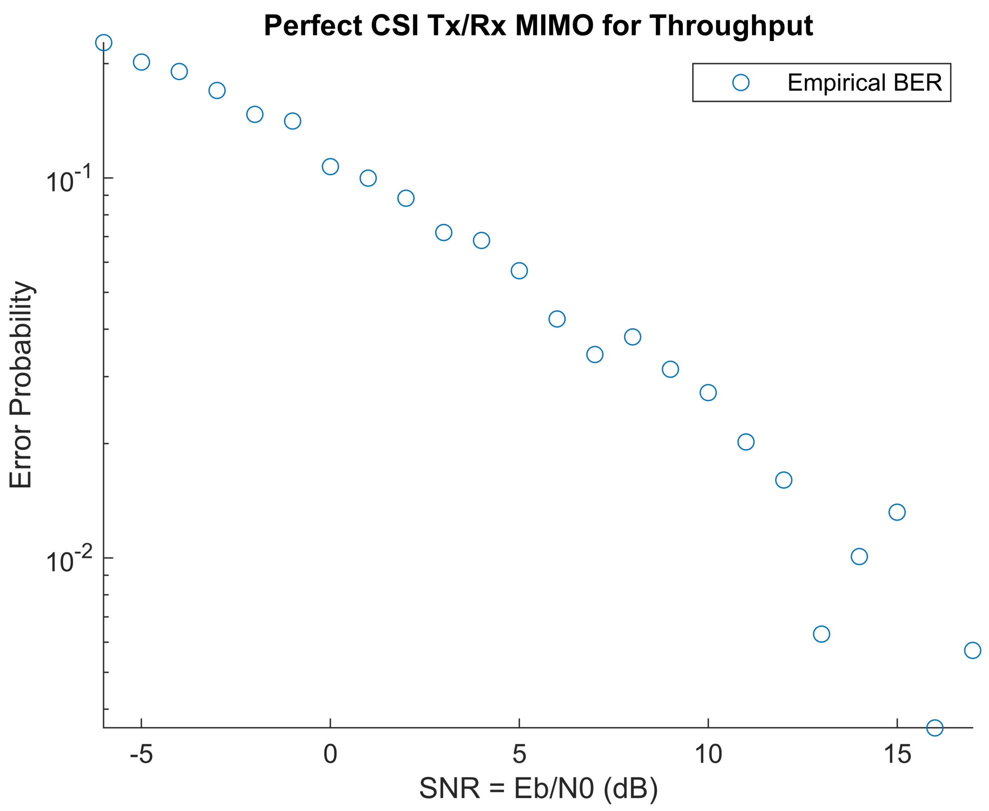
We now transition into simulating a 2x2 MIMO system with the same fading and AWGN conditions from the previous section.

Perfect CSI at Tx and Rx, MIMO for throughput

In this section, we will assume that we already have perfect channel state information at both the transmitter and receiver. This allows us to precode our symbols using the right singular matrix of the SVD (V) of the simulated channel matrix, which we refer to as H in our code. We can then combine/premultiply our received vector by the hermitian transpose of the left singular matrix (U^H) and demodulate to recover our symbols. This process theoretically isolates the two eigenchannels, rendering each spatial stream independent and free of interference. In this simulation, we are utilizing our MIMO system for greatly improved throughput, and we plot the bit error rate. We see the effect of MIMO in our expected value of Eb/N0 now being multiplied by the average squared singular value of the eigenchannel on which each data stream is transmitted. This can be interpreted as the channel gain of the MIMO eigenchannel, which may increase or decrease our SNR on each stream independently.

```
Nbits = 1e4;
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

Nt = 2; Nr = 2; % number of tx and rx antennas
txsyms = reshape(symvec,floor([Nt,length(symvec)/Nt]));
bers5 = zeros(length(N0s), 1);
sigfad = 1;

for i = 1:length(N0s)
    N0 = N0s(i);
    tmps = 0;
    tmpb = 0;
    nruns = 100;
    for k = 1:nruns
        H = sigfad * complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
        [u,s,v] = svd(H);
        xtilde = v * txsyms; % precode txsyms;
        noisemat = sqrt(N0)*complex(randn(size(txsyms)),randn(size(txsyms)));
        ytilde = H*xtilde + noisemat;
        y = ctranspose(u) * ytilde; % postprocess ytilde
        rxvec = y(:);
        [detsymvec,detbitvec] = qpskdetect(rxvec);
        tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
        tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
    end
    ser5 = tmps/nruns;
    bers5(i) = tmpb/nruns;
end
```

```
clf;
figure;
hold on
scatter(EbN0dBs, bers5);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('Perfect CSI Tx/Rx MIMO for Throughput');
set(gca,'yscale','log')
legend("Empirical BER")
ylim([-inf inf])
xlim([-inf inf])
hold off
```

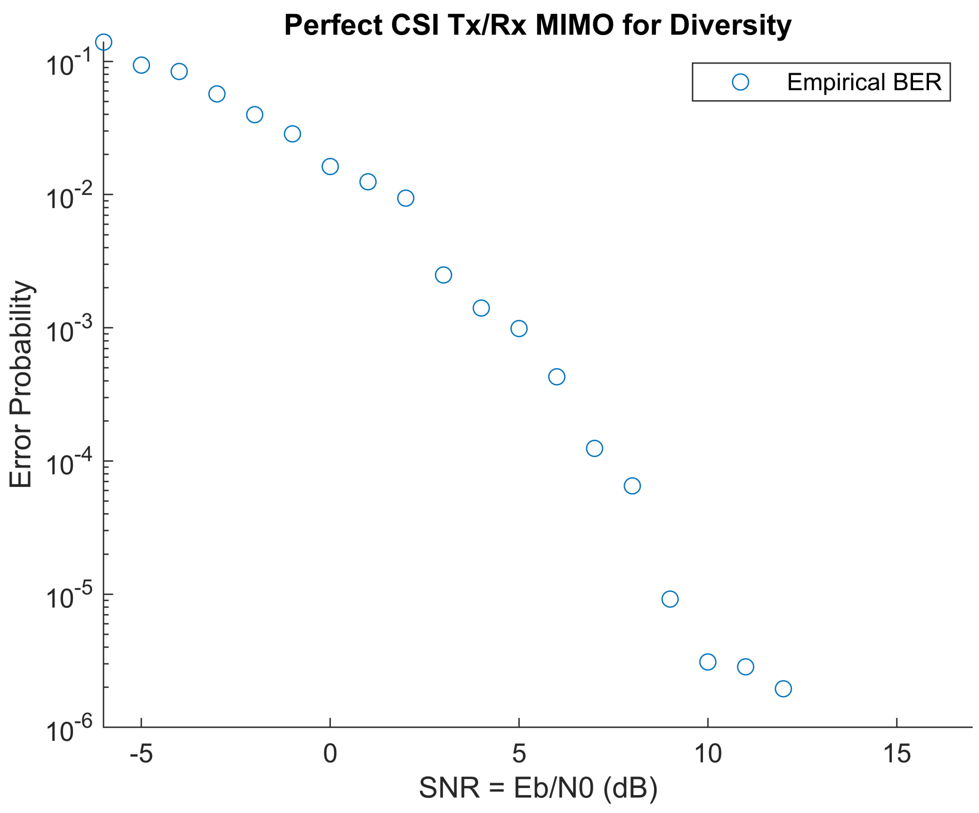


Perfect CSI at Tx and Rx, MIMO for diversity

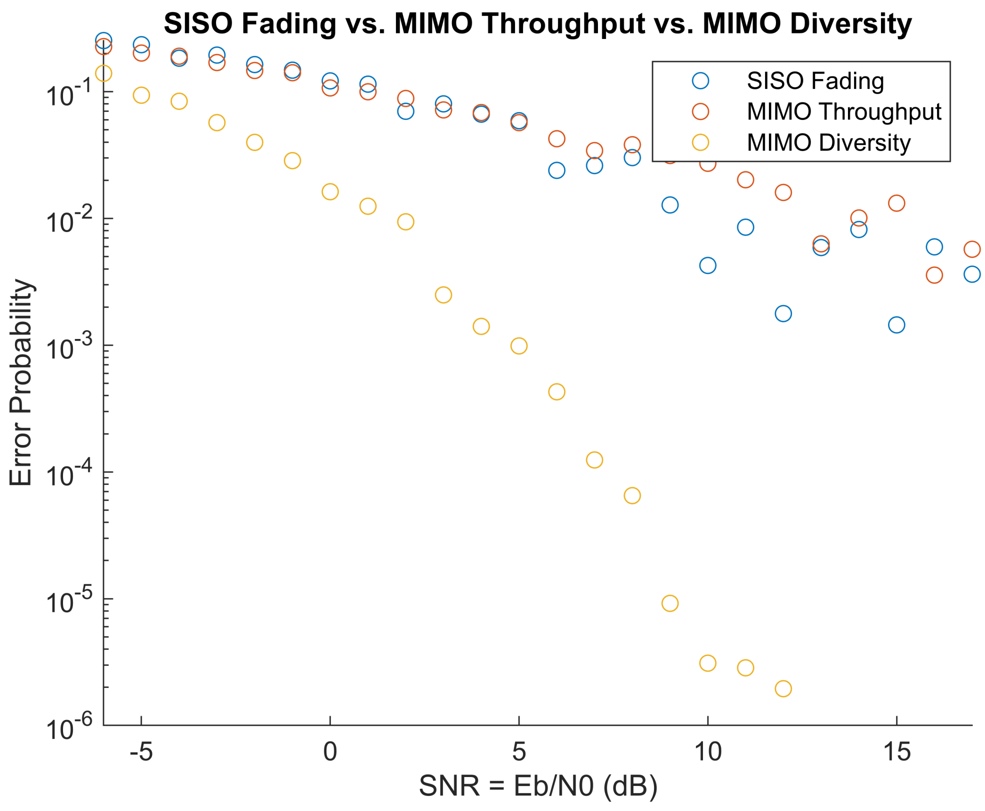
Here we are simulating the same conditions, but we are utilizing our MIMO system for diversity instead. This means that we are sending the same signal on both transmitters such that our bit error rate at the receivers will be lower and the system will be less susceptible to noise. We can see the effects of this as our bit error rates are notably lower here than in the previous section, which comes with the tradeoff of less throughput. For the diversity case, the channel gains of both eigenchannels are now added together to boost the SNR of the single data stream.

```
Nbits = 1e6;
```

```
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

Nt = 2; Nr = 2; % number of tx and rx antennas
txsyms = repmat(symvec,[1,2]).'; % note the difference with line 51!

bers6 = zeros(length(N0s), 1);

for i = 1:length(N0s)
    N0 = N0s(i);
    tmps = 0;
    tmpb = 0;
    nruns = 100;
    for k = 1:nruns
        H = sigfad * complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
        [u,s,v] = svd(H);
        xtilde = v * txsyms; % precode txsyms;
        noisemat = sqrt(N0)*complex(randn(size(txsyms)),randn(size(txsyms)));
        ytilde = H*xtilde + noisemat;
        y = ctranspose(u) * ytilde; % postprocess ytilde
        rxvec = sum(y).';
        [detsymvec,detbitvec] = qpskdetect(rxvec);
        tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
        tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
    end
    ser6 = tmps/nruns;
    bers6(i) = tmpb/nruns;
end

clf;
figure;
hold on
scatter(EbN0dBs, bers6);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('Perfect CSI Tx/Rx MIMO for Diversity');
set(gca,'yscale','log')
legend("Empirical BER")
ylim([1e-6 inf])
xlim([-inf inf])
hold off
```



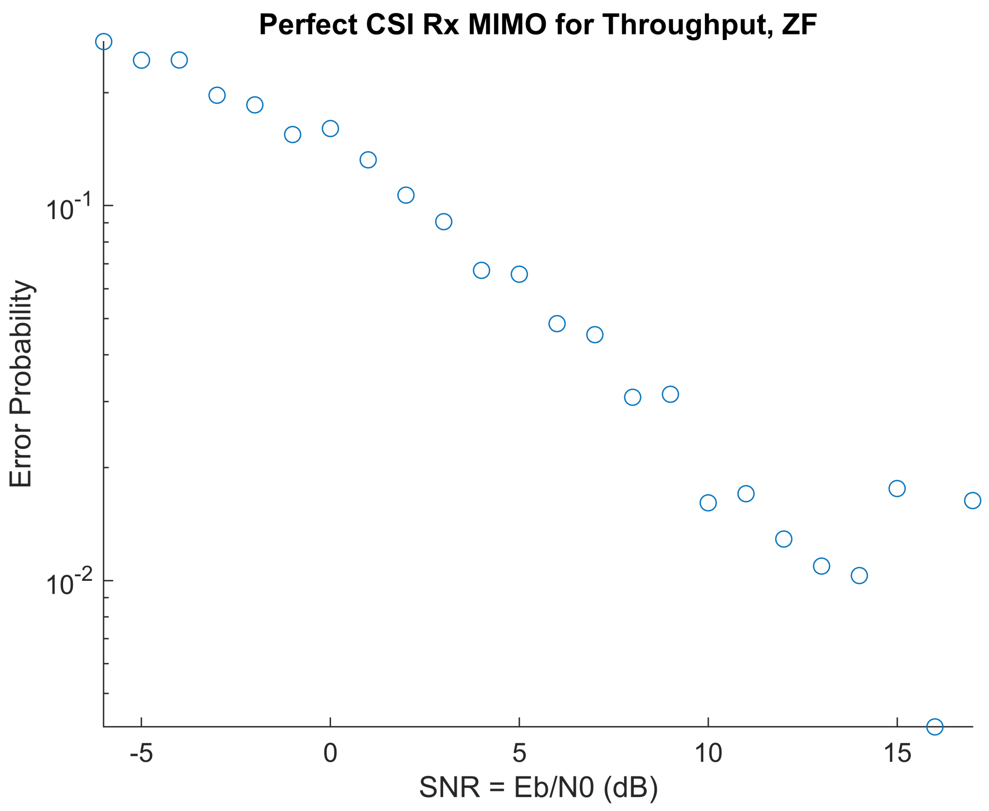
```
% Extra: Comparison of SISO fading vs. MIMO throughput vs. MIMO diversity
clf;
figure;
hold on
scatter(EbN0dBs, bers4);
scatter(EbN0dBs, bers5);
scatter(EbN0dBs, bers6);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('SISO Fading vs. MIMO Throughput vs. MIMO Diversity');
set(gca,'yscale','log')
legend("SISO Fading", "MIMO Throughput", "MIMO Diversity")
ylim([1e-6 inf])
xlim([-inf inf])
hold off
```



Perfect CSI at Rx only, Zero-Forcing, MIMO for throughput

In this section, we are assuming that we only have perfect channel state information at the receiver. Instead of precoding the symbols, here we apply zero-forcing channel equilization at the receiver, and we are utilizing MIMO for throughput. Shown below is the resulting bit error plot from this simulation.

```
Nbits = 1e4;
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

txsyms = reshape(symvec,floor([Nt,length(symvec)/Nt]));
bers7 = zeros(length(N0s), 1);

for i = 1:length(N0s)
    N0 = N0s(i);
    tmps = 0;
    tmpb = 0;
    nruns = 100;
    for k = 1:nruns
        H = complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
        [u,s,v] = svd(H);
        noisemat = sqrt(N0)*complex(randn(size(txsyms)),randn(size(txsyms)));
        y = H*txsyms + noisemat;
        xhatzf = (H' * H) \ (H' * y); % apply ZF channel equalization
        rxvec = xhatzf(:);
        [detsymvec,detbitvec] = qpskdetect(rxvec);
        tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
        tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
    end
    ser7 = tmps/nruns;
    bers7(i) = tmpb/nruns;
end

figure;
scatter(EbN0dBs, bers7);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('Perfect CSI Rx MIMO for Throughput, ZF');
xlim([-inf inf])
set(gca,'yscale','log')
```

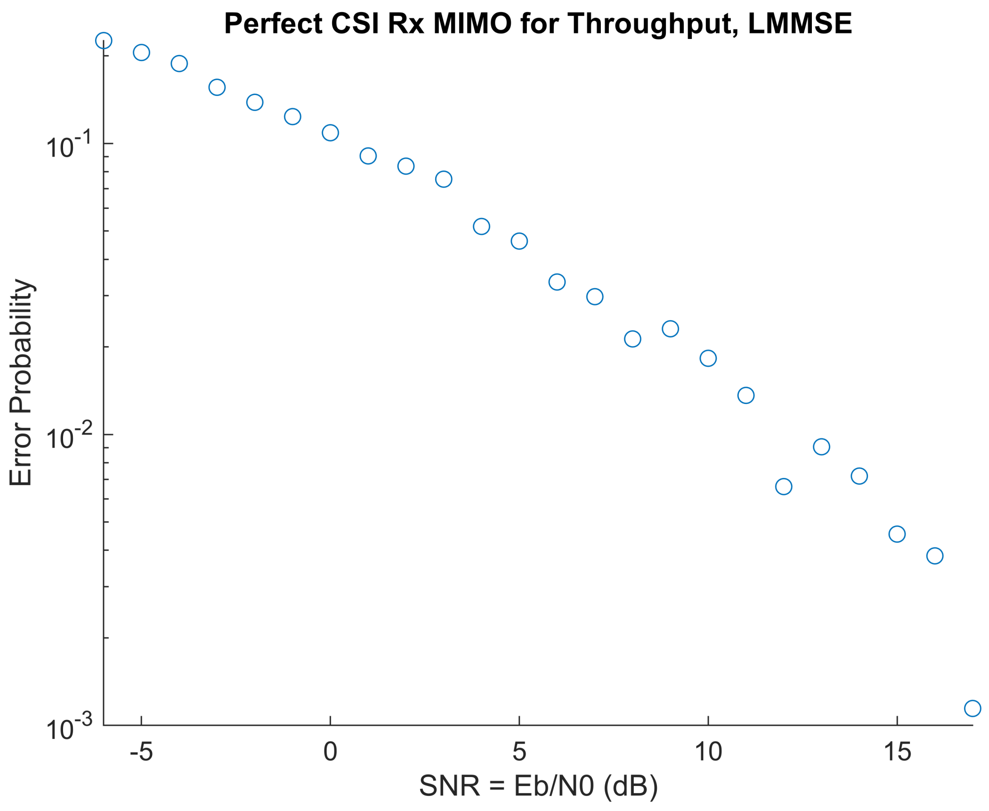


Perfect CSI at Rx only, LMMSE, MIMO for throughput

Here, we are simulating the same conditions as the previous section, but instead of using zero-forcing, we are utilizing linear minimum mean-squared (LMMSE) channel equalization. LMMSE should be less susceptible to noise as it uses the noise term as a regularizer to modify the Zero-Forcing least squares solution, and we can observe that in our bit error plot here that the error rate is lower here than for the zero-forcing.

```
Nbits = 1e4;
```

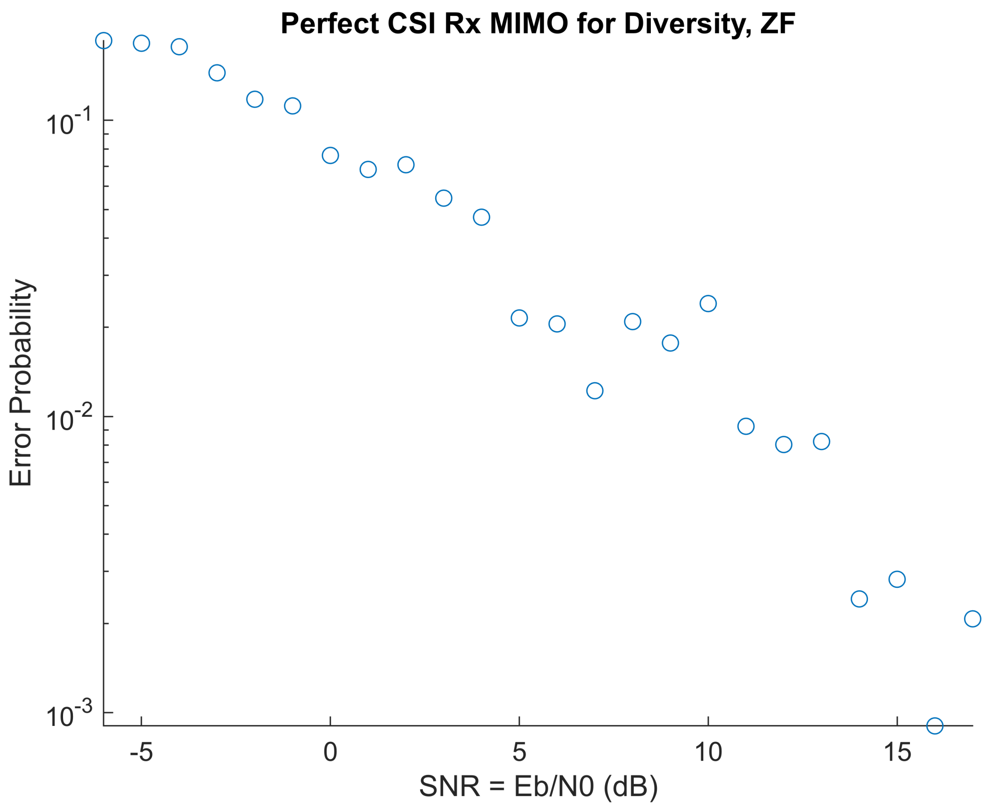
```
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

txsyms = reshape(symvec,floor([Nt,length(symvec)/Nt]));
bers8 = zeros(length(N0s), 1);

for i = 1:length(N0s)
    N0 = N0s(i);
    tmps = 0;
    tmpb = 0;
    nruns = 100;
    for k = 1:nruns
        H = complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
        [u,s,v] = svd(H);
        noisemat = sqrt(N0)*complex(randn(size(txsyms)),randn(size(txsyms)));
        y = H*txsyms + noisemat;
        xhatlmmse = ((H' * H) + (N0 .* eye(size(H,1)))) \ (H' * y); % apply LMMSE
channel equalization
        rxvec = xhatlmmse(:);
        [detsymvec,detbitvec] = qpskdetect(rxvec);
        tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
        tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
    end
    ser8 = tmps/nruns;
    bers8(i) = tmpb/nruns;
end

figure;
scatter(EbN0dBs, bers8);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('Perfect CSI Rx MIMO for Throughput, LMMSE');
xlim([-inf inf])
set(gca,'yscale','log')
```



Perfect CSI at Rx only, Zero-Forcing, MIMO for diversity

The change we are making for this section is that instead of using MIMO for throughput, we are now simulating the zero-forcing equalization when using MIMO for diversity. As expected, we see that our bit error plot the error is lower than when simulating with MIMO for throughput due to utilizing both channels for one set of symbols to enhance the SNR.

```
Nbits = 1e5;
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

txsyms = repmat(symvec,[1,2]).';
bers9 = zeros(length(N0s), 1);

for i = 1:length(N0s)
    N0 = N0s(i);
    tmps = 0;
    tmpb = 0;
    nruns = 100;
    for k = 1:nruns
        H = complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
        [u,s,v] = svd(H);
        noisemat = sqrt(N0)*complex(randn(size(txsyms)),randn(size(txsyms)));
        y = H*txsyms + noisemat;
        xhatzf = (H' * H) \ (H' * y);
        rxvec = sum(xhatzf).';
        [detsymvec,detbitvec] = qpskdetect(rxvec);
        tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
        tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
    end
    ser9 = tmps/nruns;
    bers9(i) = tmpb/nruns;
end

figure;
scatter(EbN0dBs, bers9);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('Perfect CSI Rx MIMO for Diversity, ZF');
xlim([-inf inf])
set(gca,'yscale','log')
```

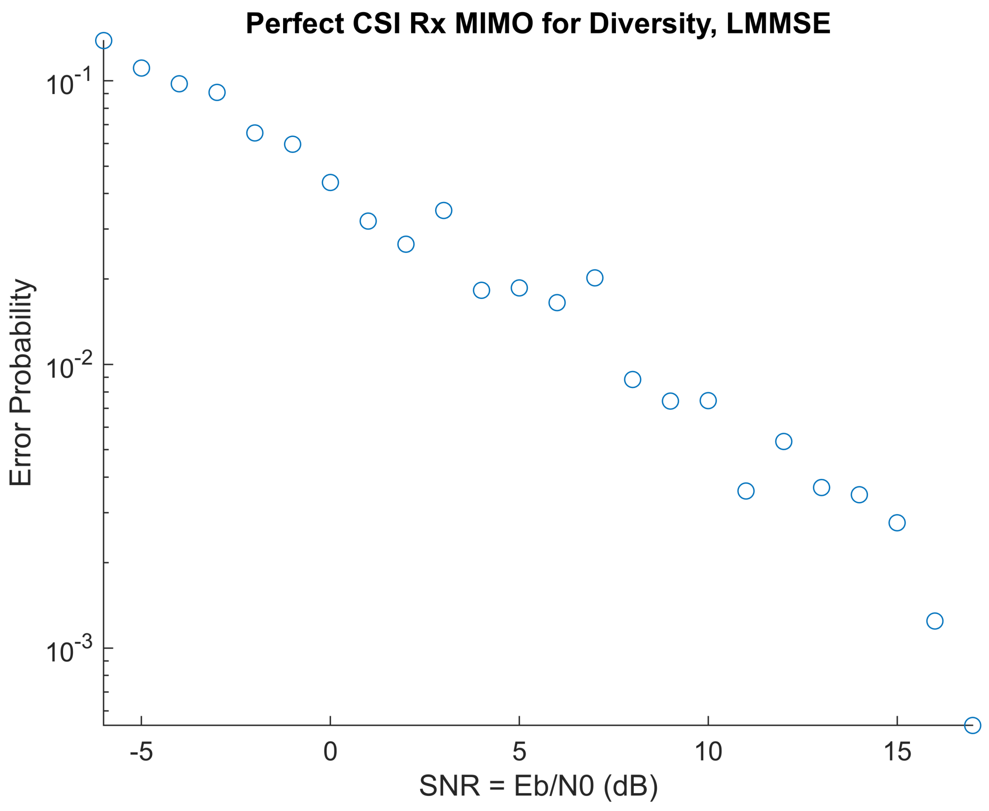


Perfect CSI at Rx only, LMMSE, MIMO for diversity

As before, we are testing our LMMSE equalization but are now using MIMO for diversity. We observe the expected behavior in our bit error plot that our bit error rate is lower than with MIMO for throughput, showcasing the tradeoff we are making between throughput and error rate.

```
Nbits = 1e5;
```

```
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

txsyms = repmat(symvec,[1,2]).';
bers10 = zeros(length(N0s), 1);

for i = 1:length(N0s)
    N0 = N0s(i);
    tmps = 0;
    tmpb = 0;
    nruns = 100;
    for k = 1:nruns
        H = complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
        [u,s,v] = svd(H);
        noisemat = sqrt(N0)*complex(randn(size(txsyms)),randn(size(txsyms)));
        y = H*txsyms + noisemat;
        xhatlmmse = ((H' * H) + (N0 .* eye(size(H,1)))) \ (H' * y);
        rxvec = sum(xhatlmmse).';
        [detsymvec,detbitvec] = qpskdetect(rxvec);
        tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
        tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
    end
    ser10 = tmps/nruns;
    bers10(i) = tmpb/nruns;
end

figure;
scatter(EbN0dBs, bers10);
xlabel('SNR = Eb/N0 (dB)');
ylabel('Error Probability');
title('Perfect CSI Rx MIMO for Diversity, LMMSE');
xlim([-inf inf])
set(gca,'yscale','log')
```



Channel Estimation with Pilots

Here, we demonstrate the use of pilot symbols to estimate the MIMO channel matrix. This enables previously shown techniques such as ZF or LMMSE as well as full SVD-based MIMO if the receiver feeds its channel estimate back to the transmitter. We implement the technique suggested by Prof. Lorenzelli in class as well as the Least Squares, Tikohonov / Regularized Least Squares, and Linear Minimum Mean Square Error methods for channel estimation. We compute and compare the mean square error of each estimate, then plot the BER of the LMMSE-equalized system using the estimated channel. We observe that for Npilots = 1000 symbols, the BER is very similar to the perfect RX CSI LMMSE with MIMO for throughput plot above. This is likely due to our channel estimate being highly accurate since a large number of pilot symbols were used. When we decrease the number of pilot symbols, our channel estimate has a larger error and thus the equalized signal has a worse BER, though this largely gets averaged out on the graph over our high number of iterations.

```
% use pilots (QPSK symbols, but different from info symbols)
% detect using estimated channel
Nt = 2; Nr = 2;
H = complex(randn(Nr,Nt),randn(Nr,Nt)) % channel matrix
```

```
H = 2×2 complex
  -1.0853 - 0.9746i   0.0899 + 0.5874i
   2.3922 + 0.2348i  -1.2778 - 1.5170i
```

```
Npilots = 1e3; % number of pilot symbols to send
pilotbitvec = randi([0,1],[2*Npilots,1]);
pilotsymvec = complex(2*pilotbitvec(1:2:end)-1,2*pilotbitvec(2:2:end)-1)/sqrt(2);
```

```
pilotsyms = reshape(pilotsymvec,floor([Nt,length(pilotsymvec)/Nt]));
noisemat = sqrt(N0)*complex(randn(size(pilotsyms)),randn(size(pilotsyms)));
y_pilots = H*pilotsyms + noisemat;

% Use Lorenzelli's Suggested Way (stacking & reshaping):
X = zeros(Npilots, Nt*Nr);
for i = 1:Npilots/2
    X(2*i-1:2*i, :) = [pilotsymvec(2*i-1), pilotsymvec(2*i), 0, 0; 0, 0,
pilotsymvec(2*i-1), pilotsymvec(2*i)];
end
h = pinv(X) * y_pilots(:); % ZF (Least Squares) Solution to Invert
H_hat_Lorenzelli = reshape(h, 2, 2).'
```

```
H_hat_Lorenzelli = 2×2 complex
  -1.0822 - 0.9674i   0.0864 + 0.5946i
   2.3956 + 0.2385i  -1.2810 - 1.5227i
```

```
channel_estim_Lorenzelli_MSE = norm(H - H_hat_Lorenzelli, 'fro')
```

```
channel_estim_Lorenzelli_MSE = 0.0139
```

```
% We observe that this is equivalent to the LS solution below

% Least-Squares Channel Estimate
H_hat_ls = y_pilots * pinv(pilotsyms)
```

```
H_hat_ls = 2×2 complex
  -1.0822 - 0.9674i   0.0864 + 0.5946i
   2.3956 + 0.2385i  -1.2810 - 1.5227i
```

```
channel_estim_LS_MSE = norm(H - H_hat_ls, 'fro')
```

```
channel_estim_LS_MSE = 0.0139
```

```

% Tikhonov (Regularized Least-Squares) Channel Estimate
lambda = 0.1;
H_hat_Tikhonov = (inv(conj(pilotsyms) * pilotsyms.' + lambda * eye(Nr,Nt)) *
conj(pilotsyms) * y_pilots.').'
```

```
H_hat_Tikhonov = 2×2 complex
  -1.0819 - 0.9672i   0.0863 + 0.5944i
   2.3951 + 0.2384i  -1.2808 - 1.5224i
```

```
channel_estim_Tikhonov_MSE = norm(H - H_hat_Tikhonov, 'fro')
```

```
channel_estim_Tikhonov_MSE = 0.0137
```

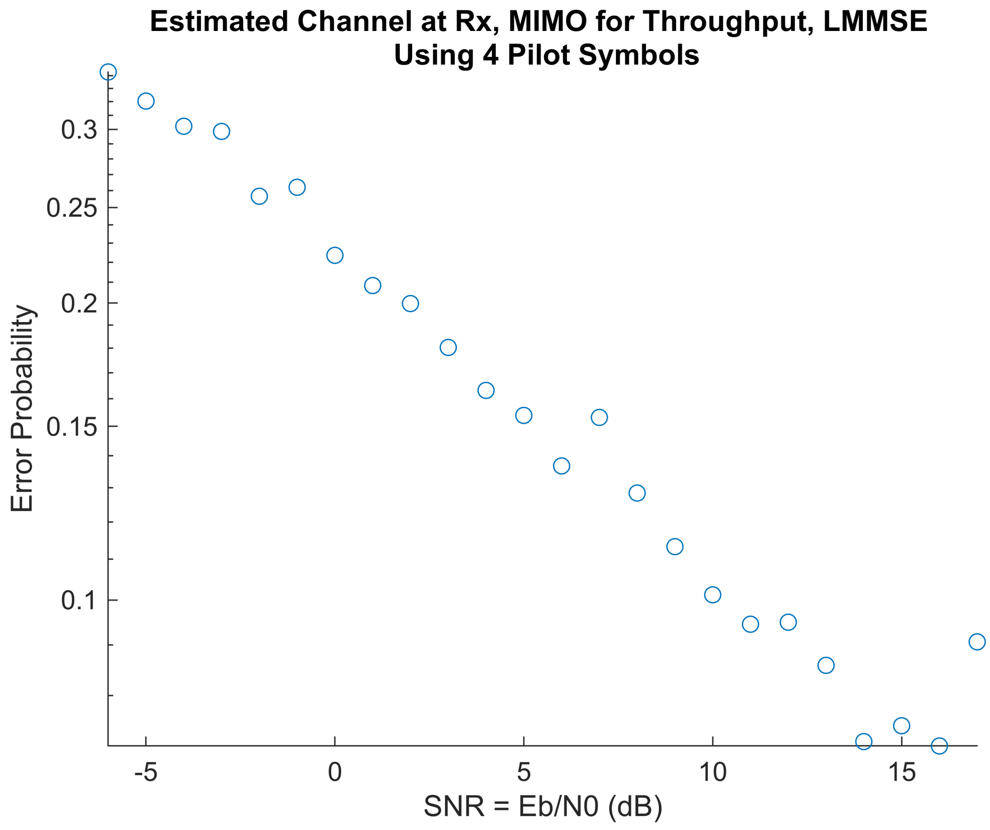
```

% LMMSE Channel Estimate
H_hat_MMSE = (inv(pilotsyms * pilotsyms' + N0 * eye(Nt, Nt)) * pilotsyms *
y_pilots')'
```

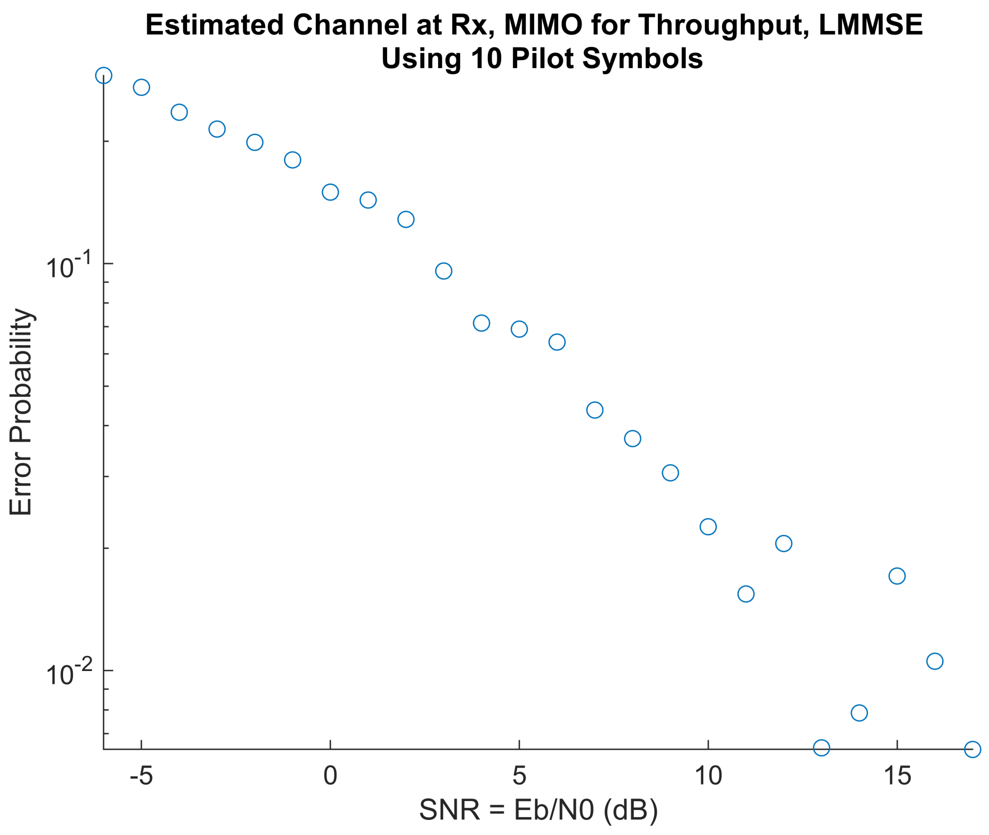
```
H_hat_MMSE = 2×2 complex
  -1.0821 - 0.9674i   0.0864 + 0.5945i
   2.3955 + 0.2384i  -1.2810 - 1.5227i
```

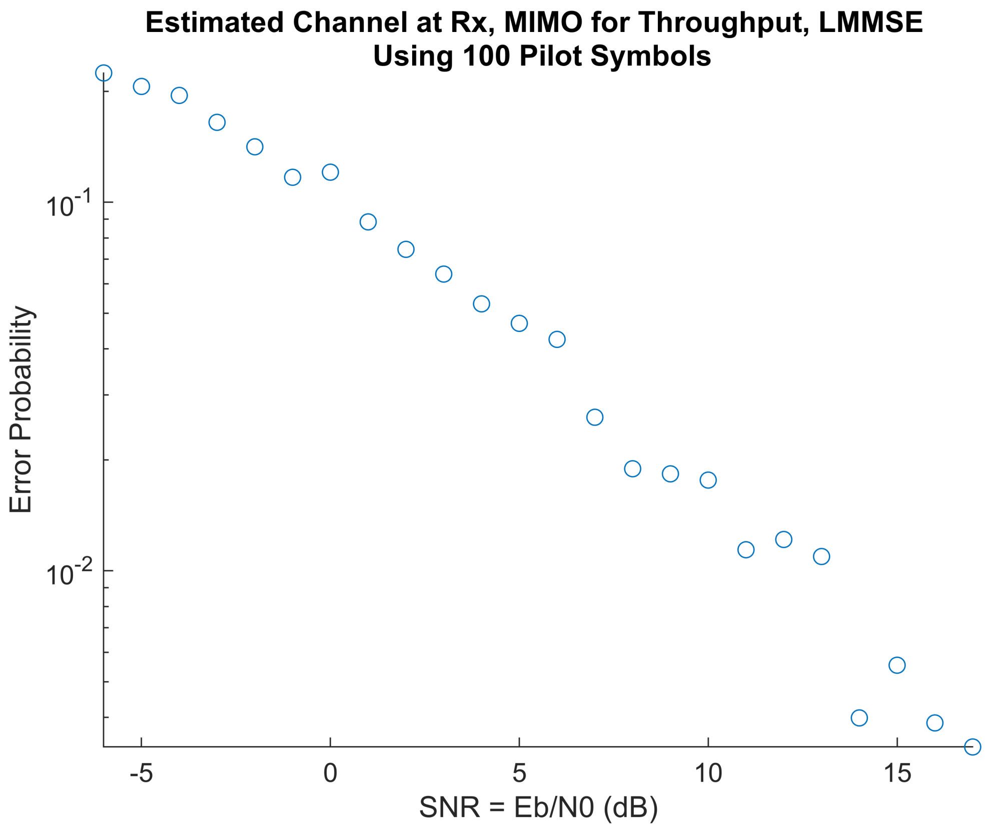
```
channel_estim_MMSE_MSE = norm(H - H_hat_MMSE, 'fro')
```

```
channel_estim_MMSE_MSE = 0.0139
```

```


% LMMSE Equalize with Estimated Channel

Nbits = 1e4;
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

txsyms = reshape(symvec,floor([Nt,length(symvec)/Nt]));
Npilots_arr = [4e0 1e1 1e2];
for n = 1:3
    Npilots = Npilots_arr(n);
    bers11 = zeros(length(N0s), 1);
    for j = 1:length(N0s)
        N0 = N0s(j);
        tmps = 0;
        tmpb = 0;
        nruns = 100;
        for k = 1:nruns
            H = complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
            pilotbitvec = randi([0,1],[2*Npilots,1]);
            pilotsymvec =
complex(2*pilotbitvec(1:2:end)-1,2*pilotbitvec(2:2:end)-1)/sqrt(2);
            pilotsyms = reshape(pilotsymvec,floor([Nt,length(pilotsymvec)/Nt]));
            noisemat =
sqrt(N0)*complex(randn(size(pilotsyms)),randn(size(pilotsyms)));
            y_pilots = H*pilotsyms + noisemat;

            % Use Lorenzelli's Suggested Way:
            X = zeros(Npilots, Nt*Nr);
            for i = 1:Npilots/2
                X(2*i-1:2*i, :) = [pilotsymvec(2*i-1), pilotsymvec(2*i), 0, 0; 0,
0, pilotsymvec(2*i-1), pilotsymvec(2*i)];
            end
            h = pinv(X) * y_pilots(:);
            H_hat_Lorenzelli = reshape(h, 2, 2).';

            noisemat = sqrt(N0)*complex(randn(size(txsyms)),randn(size(txsyms)));
            y = H*txsyms + noisemat;
            xhatlmmse = ((H_hat_Lorenzelli' * H_hat_Lorenzelli) + (N0 .*
eye(size(H_hat_Lorenzelli,1)))) \ (H_hat_Lorenzelli' * y); % apply LMMSE channel
equalization
            rxvec = xhatlmmse(:);
            [detsymvec,detbitvec] = qpskdetect(rxvec);
            tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
            tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
        end
        ser11 = tmps/nruns;
        bers11(j) = tmpb/nruns;
    end

    figure;
    scatter(EbN0dBs, bers11);
    xlabel('SNR = Eb/N0 (dB)');
    ylabel('Error Probability');
    title('Estimated Channel at Rx, MIMO for Throughput, LMMSE');
    title(sprintf('Estimated Channel at Rx, MIMO for Throughput, LMMSE \n Using %d
Pilot Symbols', Npilots_arr(n)))
    xlim([-inf inf])
    set(gca,'yscale','log')
end
```





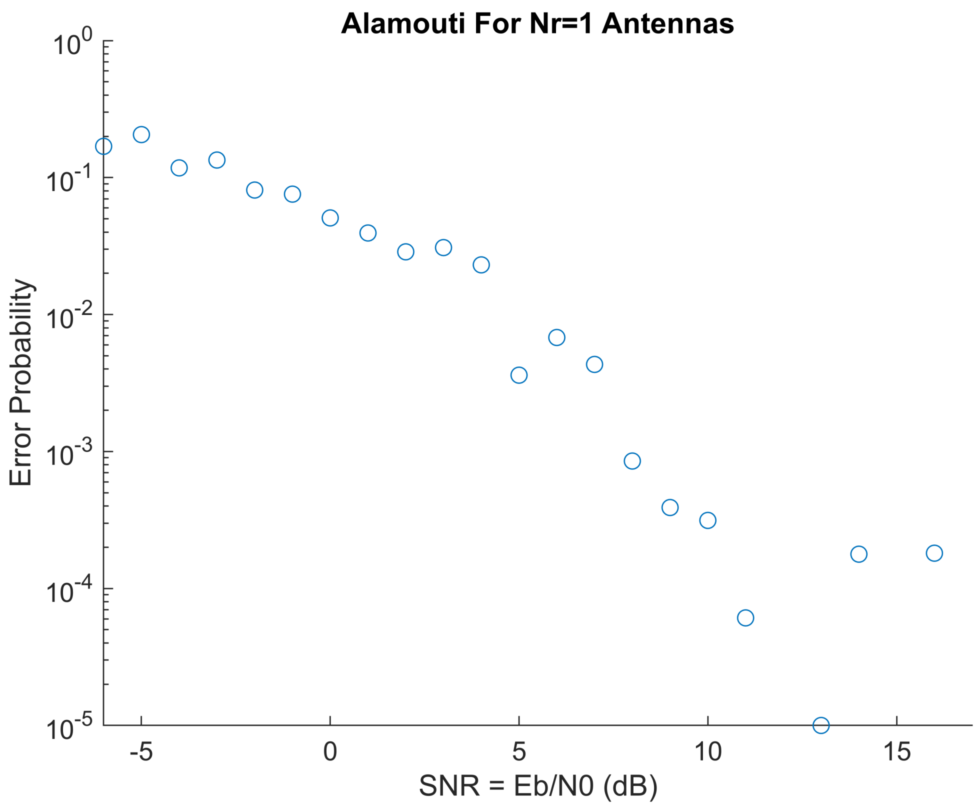


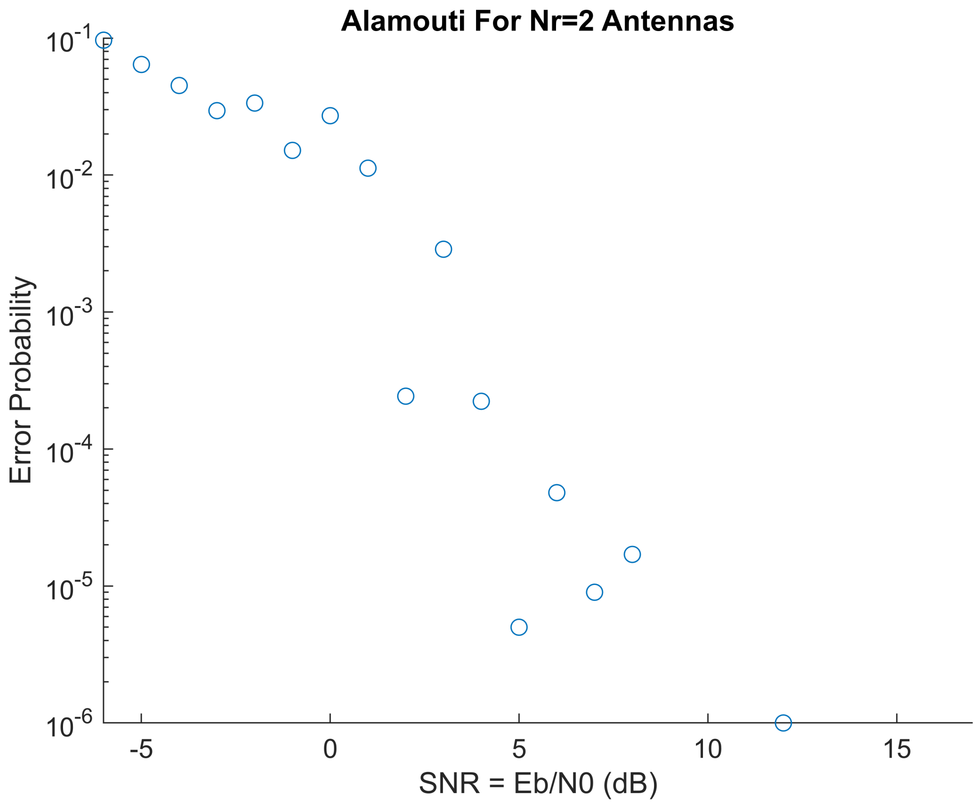
Alamouti Space-Time Code (Nt = 2)

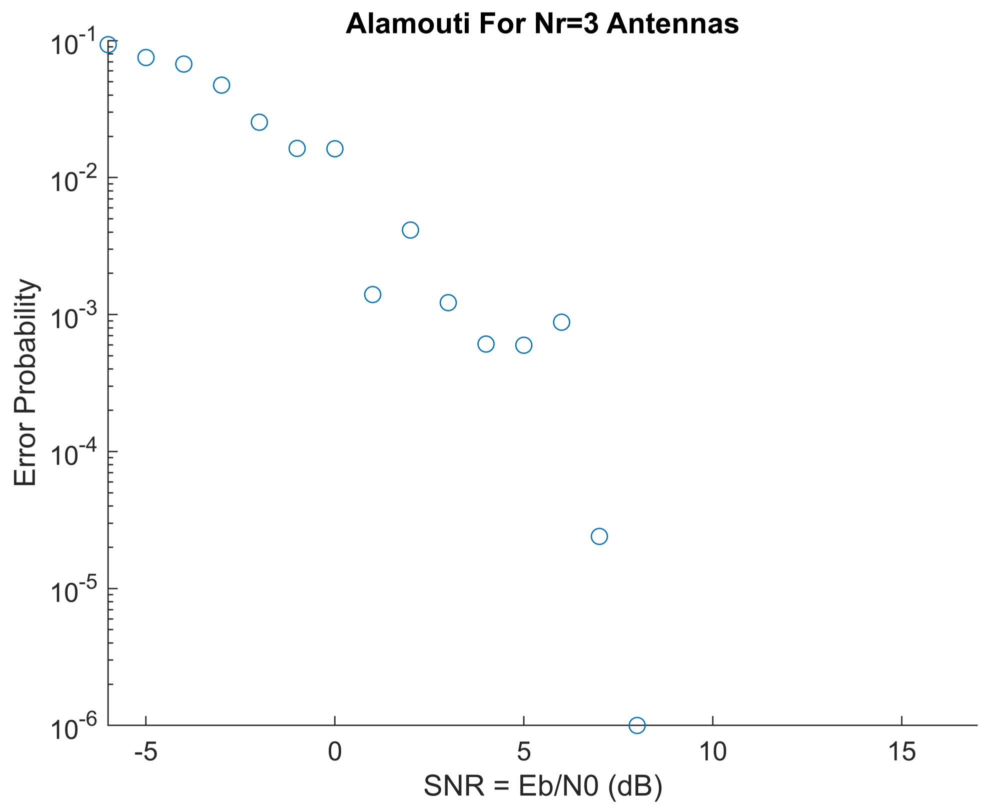
Here we demonstrate the Alamouti code technique for MIMO systems with two transmit antennas and a variable number of receive antennas. The Alamouti code is a scheme which provides diversity gain to maximize SNR without requiring explicit channel knowledge (CSI) at the transmitter. Over both antennas, we send [s0 s1] in time instant one and [-s1* s0*] in time instant two, enabling a signal processing technique for the effective recovery of s0 and s1 with large diversity gain. We observe that the BER decreases more rapidly towards zero as the number of receive antennas increases.

```
Nbits = 1e5;
% generate random bits
bitvec = randi([0,1],[Nbits,1]);
% QPSK modulation
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

Nt = 2;
sig_fad = 1;
bers12 = zeros(length(N0s), 3);

txsyms = zeros(2,length(symvec));
for ii = 1:length(symvec)/2
    txsyms(:,(ii-1)*2+1) = symvec((ii-1)*2+(1:2));
    txsyms(:,(ii-1)*2+2) = [-conj(symvec((ii-1)*2+2));conj(symvec((ii-1)*2+1))];
end

for Nr_new = 1:3
    for i = 1:length(N0s)
        N0 = N0s(i);
        tmps = 0;
        tmpb = 0;
        nruns = 10;
        for k = 1:nruns
            H = sig_fad * complex(randn(Nr_new,Nt),randn(Nr_new,Nt)); % channel
matrix
            noisemat =
sqrt(N0)*complex(randn(size(H*txsyms)),randn(size(H*txsyms)));
            y = H*txsyms + noisemat;
            yala = zeros(4, length(symvec)/2);
            % Ensure the code works for both Nr = 1 and Nr = 2, 3, ...
            for ii = 1:length(symvec)/2
                tmp = y(:, (ii-1)*2+(1:2));
                if Nr_new == 1
                    yala(:, ii) = [tmp(1, 1); conj(tmp(1, 2)); 0; 0]; % Handle Nr =
1 separately
                else
                    yala(:, ii) = [tmp(1, 1); conj(tmp(1, 2)); tmp(2, 1);
conj(tmp(2, 2))];
                end
            end

            if Nr_new == 1
```

```
                Hala = [H(1, 1), H(1, 2); conj(H(1, 2)), -conj(H(1, 1))];
            else
                Hala = [H(1, 1), H(1, 2); conj(H(1, 2)), -conj(H(1, 1)); H(2, 1),
H(2, 2); conj(H(2, 2)), -conj(H(2, 1))];
            end

            if Nr_new == 1
                tmp1 = Hala'*yala(1:2,:);
            else
                tmp1 = Hala'*yala;
            end
            rxvec = reshape(tmp1,[1,length(symvec)]).';
            [detsymvec,detbitvec] = qpskdetect(rxvec);
            tmps = tmps + sum(detsymvec~=symvec)/(Nbits/2);
            tmpb = tmpb + sum(xor(detbitvec,bitvec))/Nbits;
        end
        ser12 = tmps/nruns;
        bers12(i, Nr_new) = tmpb/nruns;
    end

    figure;
    hold on
    scatter(EbN0dBs, bers12(:,Nr_new));
    xlabel('SNR = Eb/N0 (dB)');
    ylabel('Error Probability');
    title(sprintf('Alamouti For Nr=%d Antennas', Nr_new));
    set(gca,'yscale','log')
    xlim([-inf inf])
    hold off
end
```


Alamouti For Nr=1 Antennas
Error Probability
SNR = Eb/N0 (dB)
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
-5
0
5
10
15


Alamouti For Nr=2 Antennas
Error Probability
SNR = Eb/N0 (dB)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
-5
0
5
10
15


Alamouti For Nr=3 Antennas
Error Probability
SNR = Eb/N0 (dB)
10^-1
10^-2
10^-3
10^-4
10^-5
10^-6
-5
0
5
10
15