SISO Communication System using QPSK Modulation

In QPSK, each symbol is represented by 2 bits: the first bit (or odd bits) corresponds to the real part, and the second bit (or even bits) corresponds to the imaginary part of the symbol in 'symvec'. Each symbol energy Es is normalized to 1, and the Gray-coded symbols are as follows:

$11 \rightarrow \frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}$

$01 \rightarrow -\frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}}$

$11 \rightarrow -\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}$

$10 \rightarrow \frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}$

Gray mapping is used to minimize bit error rate by ensuring only one bit change between adjacent symbols.

Since each symbol is represented by 2 bits in QPSK modulation, the relationshop between symbol energy E_s and bit energy E_b is given by $E_s = kE_b$, where k is the number of bits per symbol. For QPSK, k can be derived from $\log_2 M$, where M=4 is the number of distinct symbols. This gives the relation $E_s = 2E_b$.

When converting from $\frac{E_s}{N_0}(\text{dB})$ to $\frac{E_b}{N_0}(\text{dB})$, we use the formula:

$$\frac{E_s}{N_0}(\text{dB}) = \frac{E_b}{N_0}(\text{dB}) + 10\log_{10}(k)$$

For QPSK, k=2. $10\log_{10}(2) \approx 3(\text{dB})$

Thus,

$$\frac{E_b}{N_0}(\text{dB}) = \frac{E_s}{N_0}(\text{dB}) - 3(\text{dB})$$

Simulation Setup

```
M=4;               % QPSK
m=2;               % represent each symbol with 2 bits
Es=1;              % symbol energy
Nbits = 1e6;       % number of bits
Rs = 1;            % symbol rate
Fs = 4;            % sampling rate
sps = Fs/Rs;       % samples per symbol
EsN0dB = -3:1:20;
EbN0dB = EsN0dB-3;
N0 = 10.^(-EsN0dB/10);
hrrc = rrc((-6:1/sps:6)',0.2,1); % Generate square-root raised cosine(rrc)
pulse shaping filter
```

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

AWGN Channel, Sample Level

Now, we are passing our QPSK-modulated waveform through a root-raised-cosine (RRC) pulse shaping filter and then through an additive white Gaussian noise (AWGN) channel. We are sampling the signal over a range of SNR values from -3 dB to 20 dB. Our goal is to compare the bit error rate (BER) obtained from demodulation with the theoretical BER for Gray-mapped QPSK. The BER for QPSK is given by the following expression:

$$\rho = Q\left(\sqrt{\frac{E_s}{N_0}}\right)$$

$$\mathrm{BER}_{\mathrm{QPSK}} = \rho \cdot \left(1 - \frac{1}{2}\rho\right)$$

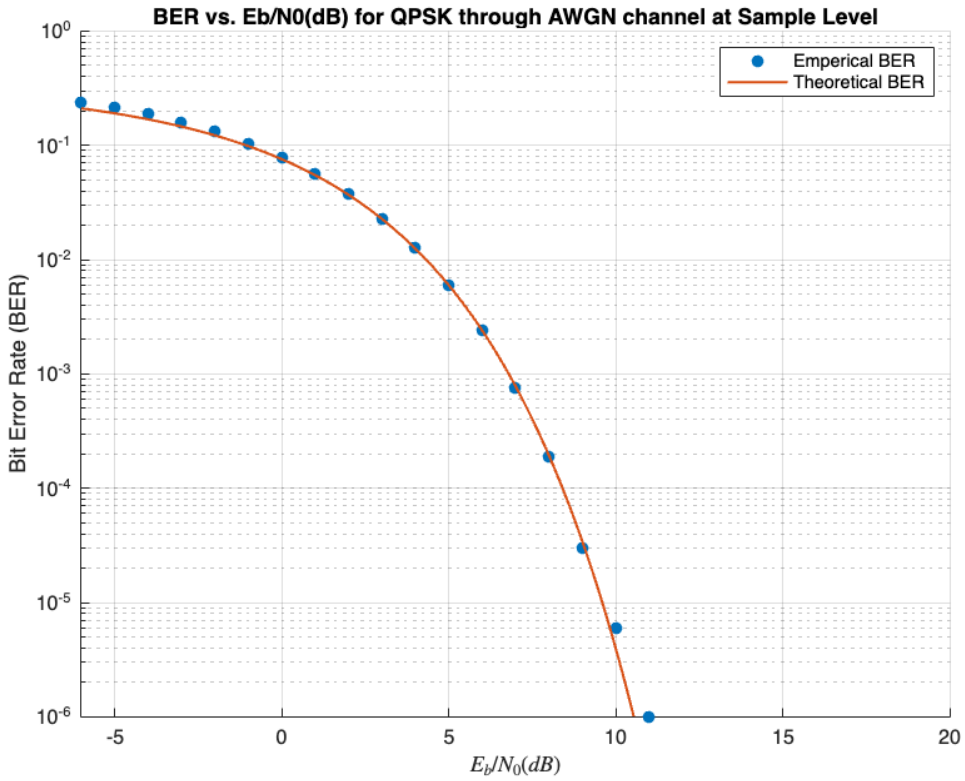
The transmitted signal 'txvec' is obtained by upsampling the QPSK complex symbol vector 'symvec' and then convolving it with the RRC filter 'hrrc'.

The variable 'sig2' represents the noise power, σ^2. This value changes with SNR because N_0 (the noise power spectral density) is dependent on SNR. In the equation, $\frac{N_0}{2}$ is used instead of N_0 since the noise power is equally distributed between the real and imaginary components of the signal for QPSK modulation.

The line "noisevec = sqrt(sig2)*complex(randn(size(txvec)),randn(size(txvec)))" generates complex noise with a standard normal distribution (mean=0, standard deviation=1). The noise is then scaled by 'sqrt(sig2)' to adjust for different SNR levels, ensuring that the noise power corresponds to the desired SNR.

```
txvec = conv(upsample(symvec,sps),hrrc,'same');

% AWGN channel
% sample level
BER=zeros(1,length(EsN0dB));
for i=1:length(EsN0dB)

    sig2 = (N0(i)/2)*Fs;
    noisevec = sqrt(sig2)*complex(randn(size(txvec)),randn(size(txvec)));
    % Rx
    rxvec = txvec + noisevec; % length=(2e6x1)

    % Pass the received signal rxvec through a match filter, hrrc(T-t),
    matched_filter=conv(rxvec,hrrc,'same');            % y(t)=rxvex*hrrc
    matched_filter_sample=matched_filter(1:sps:end);   % downsample y(t)

    % Demodulation
    [decoded_syms,decoded_bits]=qpskdetect(matched_filter_sample);
    BER(i)=sum(xor(bitvec,decoded_bits))/Nbits;
end
```

```
% theoretical BER for gray-mapping QPSK
EsN0dB2 = -3:0.1:20;
EbN0dB2 = EsN0dB2 - 3;
N02 = 10.^(-EsN0dB2/10);
BER_th=qfunc(sqrt(Es./N02)).*(1-0.5.*qfunc(sqrt(Es./N02)));

clf;
figure;
scatter(EbN0dB,BER,'filled')
set(gca,'yscale','log')
hold on
plot(EbN0dB2,BER_th,'LineWidth',1.3)
xlim([-6 20])
ylim([10e-7 1])
grid on
hold off
xlabel('$E_b/N_0 (dB)$', 'Interpreter', 'LaTex');
ylabel('Bit Error Rate (BER)');
title('BER vs. Eb/N0(dB) for QPSK through AWGN channel at Sample Level')
legend('Emperical BER','Theoretical BER')
```



AWGN Channel, Symbol Level

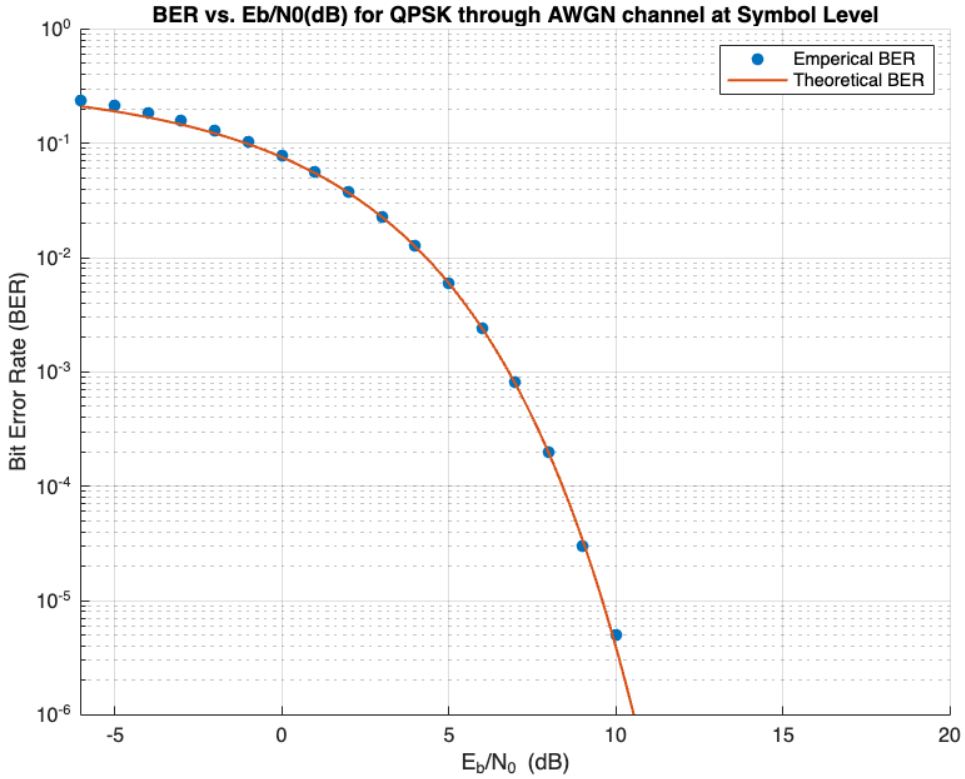
Compare sample level simulation to symbol level simulation.

The results show that the performance is consistent across both symbol level and sample level simulations.

```
BER2=zeros(1,length(EsN0dB));
```

```
Warning: Graphics timeout occurred. To share details of this issue with MathWorks technical
support, please include that this is an unresponsive graphics client with your service request.
```

```
SER2=zeros(1,length(EsN0dB)); % symbol error rate

% symbol level
for i=1:length(EsN0dB)

    sig2 = N0(i)/2;  % notice that we don't have to multiply noise power
with Fs here
    symnoise = sqrt(sig2)*complex(randn(size(symvec)),randn(size(symvec)));
    % Rx
    rxsymvec = symvec + symnoise; % length=(0.5e6x1)

    % Demodulation
    [decoded_syms2,decoded_bits2] = qpskdetect(rxsymvec);

    SER2(i) = sum(decoded_syms2~=symvec)/(Nbits/2);
    BER2(i) = sum(xor(decoded_bits2,bitvec))/Nbits;
end

% theoretical BER for gray-mapping QPSK
EsN0dB2 = -3:0.1:20;
EbN0dB2 = EsN0dB2 - 3;
N02 = 10.^(-EsN0dB2/10);
BER_th=qfunc(sqrt(Es./N02)).*(1-0.5.*qfunc(sqrt(Es./N02)));

figure;
scatter(EbN0dB,BER2,'filled')
set(gca,'yscale','log')
hold on
plot(EbN0dB2,BER_th,'LineWidth',1.3)
xlim([-6 20])
ylim([10e-7 1])
grid on
hold off
xlabel('E_b/N_0 (dB)', 'Interpreter', 'tex');
ylabel('Bit Error Rate (BER)');
title('BER vs. Eb/N0(dB) for QPSK through AWGN channel at Symbol Level')
legend('Emperical BER','Theoretical BER')
```



Impulse Response of RRC Filter

'rrc' is the root-raised-cosine (RRC) pulse shaping filter function.

In the following equation, the roll-off factor is denoted by β, which corresponds to α as used in this project.

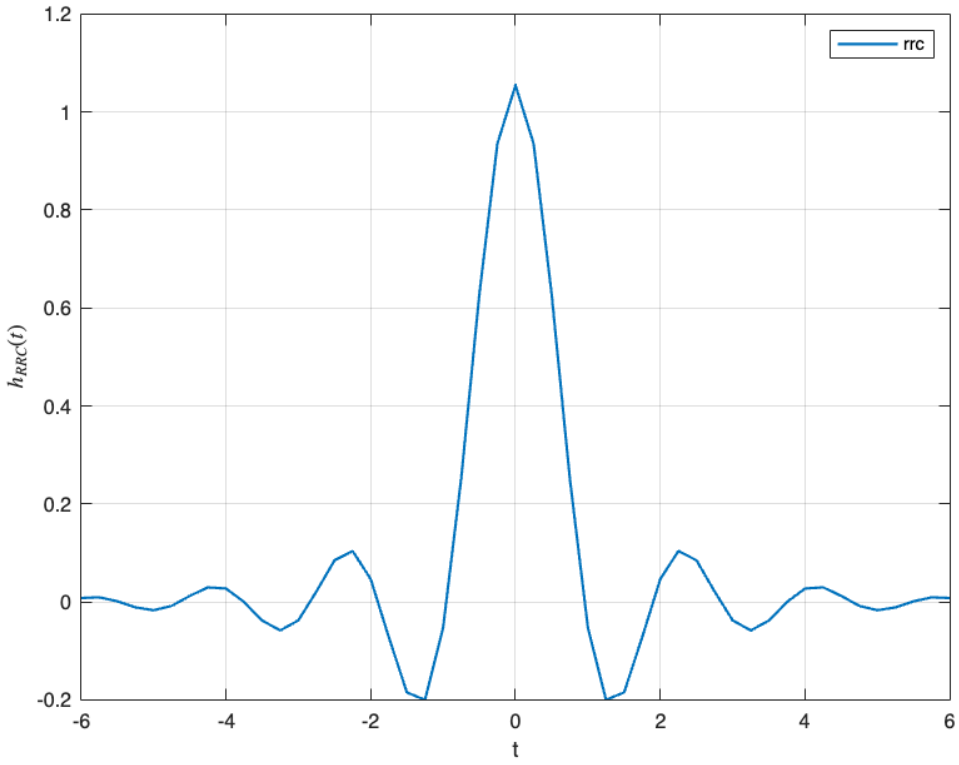
$T_s = \frac{1}{R_s}$.

$$h(t) = \begin{cases} \frac{1}{T_s}\left(1+\beta(\frac{4}{\pi}-1)\right), & t=0 \\ \frac{\beta}{T_s\sqrt{2}}\left[\left(1+\frac{2}{\pi}\right)\sin\left(\frac{\pi}{4\beta}\right)+\left(1-\frac{2}{\pi}\right)\cos\left(\frac{\pi}{4\beta}\right)\right], & t=\pm\frac{T_s}{4\beta} \\ \frac{1}{T_s}\frac{\sin\left[\pi\frac{t}{T_s}(1-\beta)\right]+4\beta\frac{t}{T_s}\cos\left[\pi\frac{t}{T_s}(1+\beta)\right]}{\pi\frac{t}{T_s}\left[1-\left(4\beta\frac{t}{T_s}\right)^2\right]}, & \text{otherwise} \end{cases},$$

Equation1. Impulse Response of RRC Filter

```
t=(-6:1/sps:6)';
alpha=0.2;
T=1/Rs;
hrrc=rrc(t,alpha,T);
```

```
figure;
plot(t,hrrc,'LineWidth',1.3)
grid on
xlabel('t')
ylabel('$h_{RRC}(t)$', 'Interpreter', 'latex')
legend('rrc')
```



Effect of RRC Filter Roll-off Factor α on BER

Roll-factor α of RRC filter measures the additional bandwidth that the filter occupies beyond the Nyquist bandwidth in frequency domain. The Nyquist bandwidth is defined as $B_{\text{Nyquist}} = \frac{1}{2T}$, where $T = \frac{1}{R_s}$.

Although an RRC filter with a higher roll-off factor α requires more bandwidth, it has a smoother impulse response in the time domain (Figure 1), which reduces overlap between adjacent symbols and better prevents intersymbol interference (ISI). In contrast, an RRC filter with a lower roll-off factor α uses less bandwidth but has a sharper, more jittery impulse response, leading to higher BER due to increased ISI.

Thus, there's a trade-off between bandwidth and BER:

Higher roll-off factors improve BER at the cost of additional bandwidth, while lower roll-off factors conserve bandwidth but can result in higher BER due to greater ISI.

From the plot 'Impact of RRC Roll-off Factor on BER,' we observe that a roll-off factor of α = 0 results in a higher BER compared to α = 1. The result is consistent with the expectation based on our analysis.

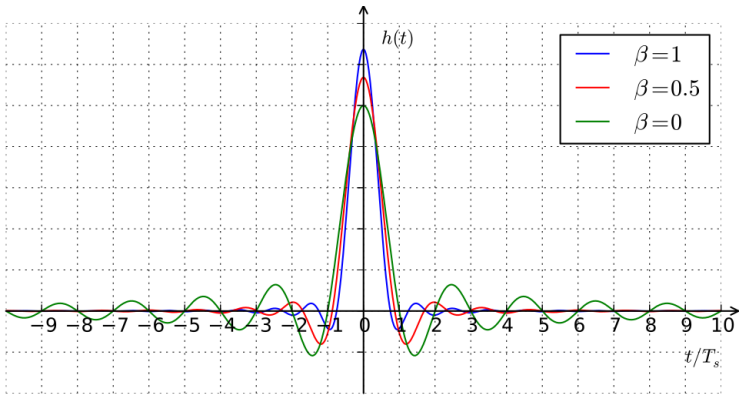


Figure 1. Impulse response of RRC filter in time domain (*Wikipedia: Root-raised-cosine filter, https://en.wikipedia.org/wiki/Root-raised-cosine_filter)*

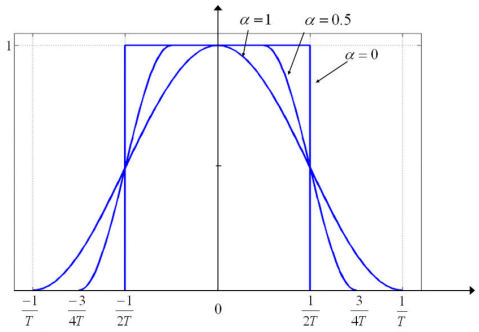
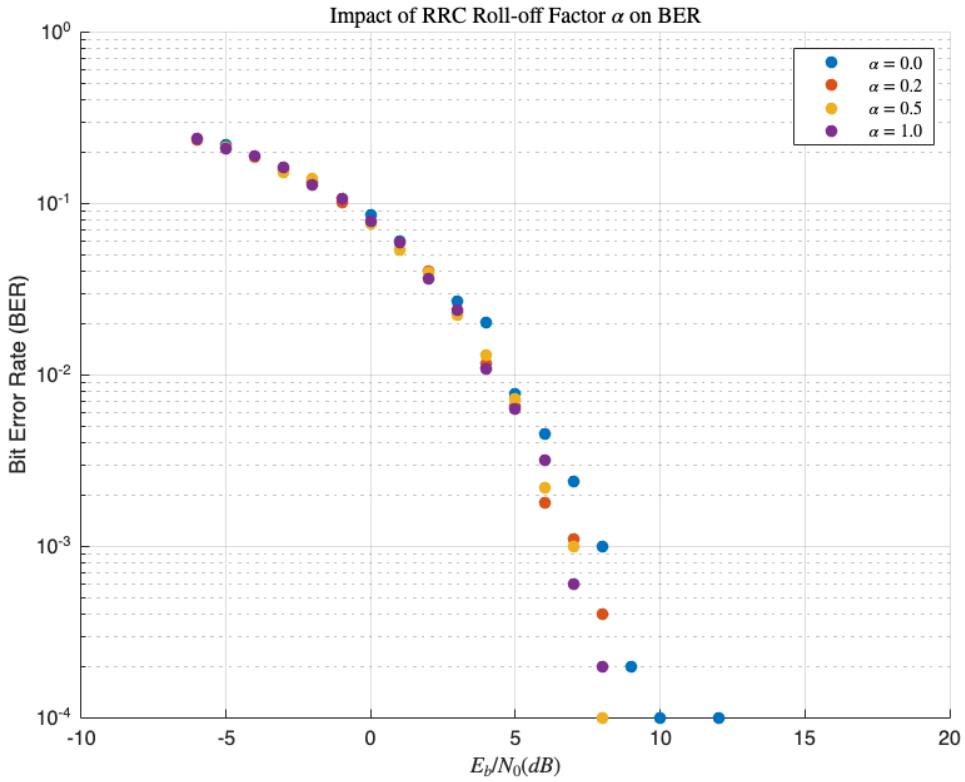


Figure 2. Impulse response of RRC filter in frequency domain *(Navipedia: Root-raised-cosine signals, https://gssc.esa.int/navipedia/index.php/Square-Root_Raised_Cosine_Signals_%28SRRC%29)*

```
alpha=[0 0.2 0.5 1];
for k=1:length(alpha)
    hrrc = rrc((-6:1/sps:6)',alpha(k),1);

    % Tx
    bitvec = randi([0,1],[Nbits,1]); % Generate random bits
    symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);
    txvec = conv(upsample(symvec,sps),hrrc,'same');

    BER=zeros(1,length(EsN0dB));
    for i=1:length(EsN0dB)

        sig2 = (N0(i)/2)*Fs;
        noisevec =
sqrt(sig2)*complex(randn(size(txvec)),randn(size(txvec)));
        % Rx
        rxvec = txvec + noisevec; % length=(2e6x1)

        matched_filter=conv(rxvec,hrrc,'same');            % y(t)=rxvex*hrrc
        matched_filter_sample=matched_filter(1:sps:end);   % downsample y(t)

        % Demodulation
```

```
        [syms,decoded_bits]=qpskdetect(matched_filter_sample);
        BER(i)=sum(xor(bitvec,decoded_bits))/Nbits;
    end
    scatter(EbN0dB,BER,'filled','DisplayName',sprintf('$\\alpha=%.1f$',
alpha(k)));
    hold on
end

set(gca,'yscale','log')
grid on
hold off
xlabel('$E_b/N_0 (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('Impact of RRC Roll-off Factor $\alpha$ on BER', 'Interpreter',
'latex');
legend('Interpreter', 'latex')
```



Slow Flat Fading Channel (Sample Level Simulation)

When a signal is transmitted from a transmitter to a receiver, it often travels through multiple paths, a phenomenon known as multipath propagation. Each path may introduce different levels of attenuation and phase shifts. When these signals converge at the receiver, their alignment in amplitude and phase can lead

to either constructive or destructive interference, resulting in fluctuations in signal strength over time called "fading." This effect can be quantified by the fading channel coefficient, h.

Fading Channel Coefficient: $h = a \cdot e^{-j\theta} = x + \mathrm{jy}$, which $a = |h|$ is the amplitude and $\theta = \angle h$ is the phase of fading channel coefficient.

$a = \sqrt{x^2 + y^2}$

$\theta = \tan^{-1}\left(\frac{y}{x}\right)$

Flat Fading: A flat fading channel exhibits uniform attenuation across the entire bandwidth of the transmitted signal, affecting all frequency components equally. This means the signal is transmitted without distortion, though it may be uniformly attenuated or amplified. (Distortion occurs when different frequency components of the transmitted signal experience different levels of attenuation and phase shift in frequency domain, altering the signal's original shape.)

Slow Fading: Fading channel coefficient, h, varies slowly over time. Happens when obstacles or transimtter and receiver either move slowly or remain static.

'sigfad': $|h|$

'fadingalpha': $h = |h| \cdot e^{-j\theta}$

fadingalpha = sigfad(n)*complex(randn(1),rand(1));

In the above code, we generate a random fading channel coefficient h with its amplitude 'sigfad' Ihl. We then multiply Ihl by a randomly generated complex number, where both the real and imaginary parts follow a standard normal distribution, ensuring the phase is uniformly distributed over $-\pi$ to π.

Suppose we have a complex $Z = X + \mathrm{jY}$, $X \sim N(0,1)$ and $Y \sim N(0,1)$, and X is independent of Y

$$f_{XY}(x,y) = f_X(x) \cdot f_Y(y) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}} \cdot \frac{1}{\sqrt{2\pi}} e^{\frac{-y^2}{2}} = \frac{1}{2\pi} e^{\frac{-(x^2+y^2)}{2}}$$

$$f_{A\Theta}(a,\theta) = \frac{a}{2\pi} e^{\frac{-a^2}{2}}$$

$f_A(a) = \int_{-\pi}^{\pi} \frac{a}{2\pi} e^{\frac{-a^2}{2}} d\theta = 2a \cdot e^{-a^2}$ ...Rayleigh distribution

$f_\Theta(\theta) = \int_0^{\infty} \frac{a}{2\pi} e^{\frac{-a^2}{2}} \mathrm{da} = \frac{1}{2\pi}$...uniform distribution, $[-\pi, \pi]$

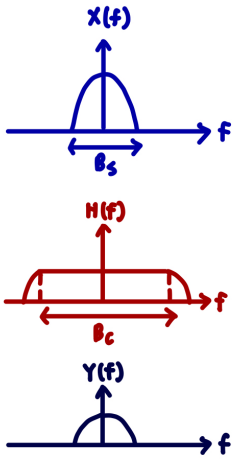
Coherence Bandwidth:

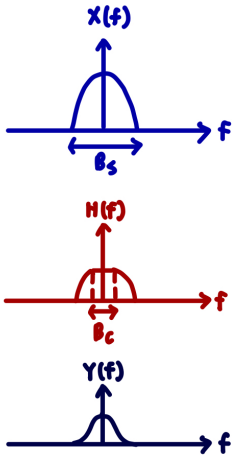
B_c: coherence bandwidth (flat portion of channel bandwidth)

B_s: signal bandwidth

(a) $B_s < B_c$ ⟶ Flat fading without distortion

(b) $B_s > B_c$ ⟶ Frequency selective channel with distortion





In frequency domain, we have $|Y| = |X| \cdot |H|$, where $|X|$ and $|Y|$ are transmitted and received signals, and $|H|$ is the impulse response of the channel. In case (a), the signal bandwidth B_s is within the flat portion of channel bandwidth B_c, so all frequency components of the signal are attenuated equally as they pass through the channel. As a result, no distortion occurs. In case (b), since signal bandwidth B_s is wider than the channel bandwidth B_c, the transmitted signal gets distorted.

Phase Compensation:

mfout = conv(rxvecfad*exp(-1i*angle(fadingalpha)),hrrc,'same')/Fs;

In this step, we correct the phase shift introduced by channel fading during transmission. This process, known as phase compensation, involves applying the inverse of the phase shift caused by the channel to the received signal.

Compare experimental result with theoretical value:

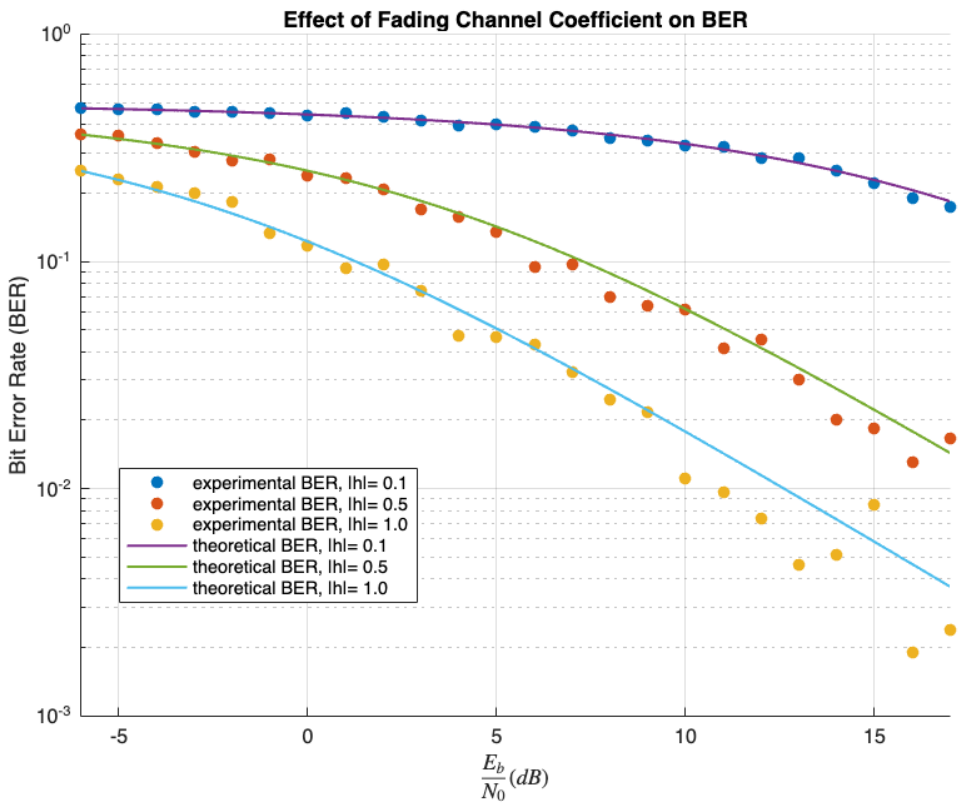
SNR affected by fading channel: $\mathrm{SNR}_{\mathrm{fading}} = \dfrac{|h|^2 \cdot E_s}{N_0} = |h|^2 \cdot \mathrm{SNR}$

Theoretical value of BER with fading channel: $\mathrm{BER}_{\mathrm{thm}} = \dfrac{1}{2} \cdot \left(1 - \sqrt{\dfrac{\mathrm{SNR}_{\mathrm{fading}}}{1.5 + \mathrm{SNR}_{\mathrm{fading}}}}\right)$

```
%% Flat fading
% sample-level simulation
clf;
figure;
Nbits=1e3;
Rs=1;
T=1/Rs;
Fs=4;
sps=Fs/Rs;
```

```
alpha=0.2;  % roll-off factor
hrrc=rrc((-6:1/sps:6)',alpha,T);

EsN0dB = -3:1:20;
EbN0dB = EsN0dB-3;
N0 = 10.^(-EsN0dB/10);
EsN0dB2 = -3:0.1:20;
EbN0dB2 = EsN0dB2-3;
N02 = 10.^(-EsN0dB2/10);

bitvec=randi([0,1],[Nbits,1]);
symvec=(1/sqrt(2))*complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1);
txvec = conv(upsample(symvec,sps),hrrc,'same');

sigfad = [0.1 0.5 1];   % |h|: amplitude of fading channel coefficent

ser3=zeros(1,length(EsN0dB));
ber3=zeros(1,length(EsN0dB));

nruns = 100;
for n=1:length(sigfad)

    for i=1:length(EsN0dB)
        sig2 = N0(i)*Fs/2;
        total_symbol_error = 0;
        total_bit_error = 0;
        for k = 1:nruns
            noisevec =
sqrt(sig2)*complex(randn(size(txvec)),randn(size(txvec)));
            fadingalpha = sigfad(n)*complex(randn(1),rand(1));  % h=|h|
*exp(i*theta), theta=phase of h
            rxvecfad = fadingalpha * txvec + noisevec;
            mfout =
conv(rxvecfad*exp(-1i*angle(fadingalpha)),hrrc,'same')/Fs; % phase
compensation
            mfsamp = mfout(1:sps:end);
            [detsymvec,detbitvec] = qpskdetect(mfsamp);
            total_symbol_error = total_symbol_error +
sum(detsymvec~=symvec)/(Nbits/2);
            total_bit_error = total_bit_error + sum(xor(detbitvec,bitvec))/
Nbits;
        end
        ser3(i) = total_symbol_error/nruns;
        ber3(i) = total_bit_error/nruns;
    end

    scatter(EbN0dB,ber3,'filled','DisplayName',sprintf('experimental BER, |
h|= %.1f',sigfad(n)));
    hold on
end
```

```
% theoretical BER for QPSK modulation with flat fading channel
for n=1:length(sigfad)
    SNR_fad=sigfad(n)^2*Es./N02; % SNRs affected by fading channel
    BER_th=0.5*(1-sqrt(SNR_fad./(1.5+SNR_fad)));
    plot(EbN0dB2,BER_th,'LineWidth',1.3,'DisplayName',sprintf('theoretical
BER, |h|= %.1f',sigfad(n)))
end


set(gca,'yscale','log')
hold off
xlim([-6 17])
ylim([1e-3 1])
grid on

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('Effect of Fading Channel Coefficient on BER')
legend("Position", [0.1641 0.2503 0.2661, 0.1583]);
```



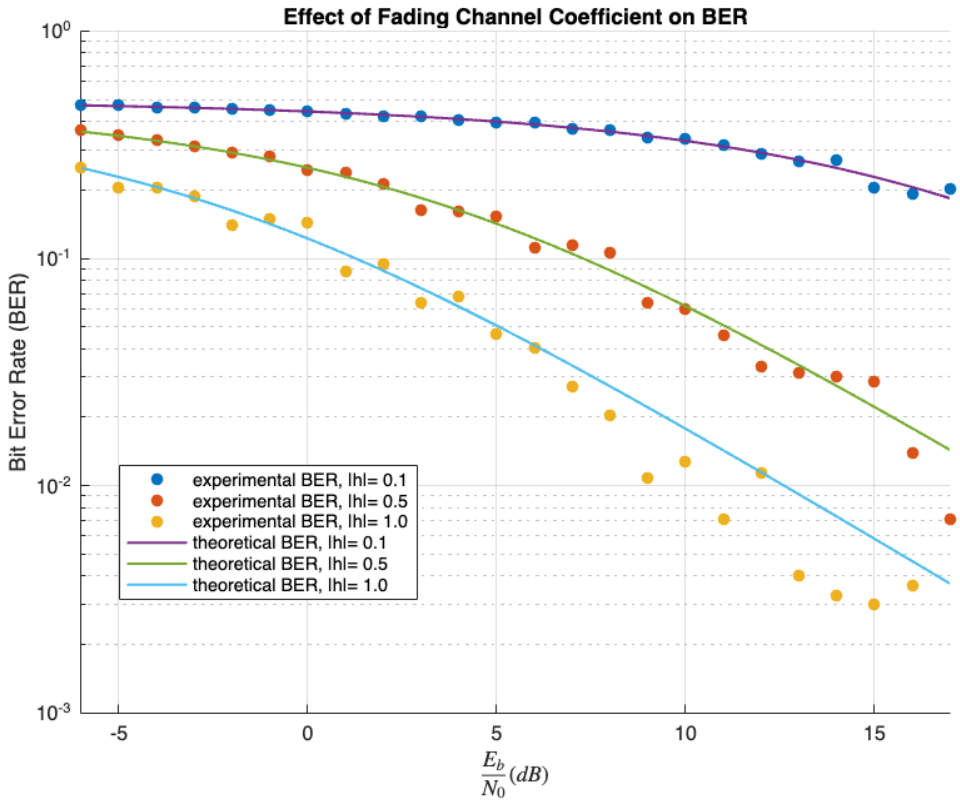
Slow Flat Fading Channel (Symbol Level Simulation)

Compare sample level simulation to symbol level simulation.

The results show that the performance is consistent across both symbol level and sample level simulations.

```
%% Flat fading
% symbol-level simulation
clf;
figure;

bitvec=randi([0,1],[Nbits,1]);
symvec=(1/sqrt(2))*complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1);

sigfad = [0.1 0.5 1];   % |h|: amplitude of fading channel coefficent

ser4=zeros(1,length(EsN0dB));
ber4=zeros(1,length(EsN0dB));

nruns = 100;
for n=1:length(sigfad)

    for i=1:length(EsN0dB)
        sig2 = N0(i)/2;
        total_symbol_error = 0;
        total_bit_error = 0;
        for k = 1:nruns
            noisevec =
sqrt(sig2)*complex(randn(size(symvec)),randn(size(symvec)));
            fadingalpha = sigfad(n)*complex(randn(1),rand(1));  % h=|h|
*exp(i*theta), theta=phase of h
            rxvecfad = fadingalpha * symvec + noisevec;
            % Phase compensation
            rxvecfad = rxvecfad*exp(-1i*angle(fadingalpha));
            [detsymvec,detbitvec] = qpskdetect(rxvecfad);
            total_symbol_error = total_symbol_error +
sum(detsymvec~=symvec)/(Nbits/2);
            total_bit_error = total_bit_error + sum(xor(detbitvec,bitvec))/
Nbits;
        end
        ser4(i) = total_symbol_error/nruns;
        ber4(i) = total_bit_error/nruns;
    end

    scatter(EbN0dB,ber4,'filled','DisplayName',sprintf('experimental BER, |
h|= %.1f',sigfad(n)));
    hold on
end

% theoretical BER for QPSK modulation with flat fading channel
for n=1:length(sigfad)
    SNR_fad=sigfad(n)^2*Es./N02; % SNRs affected by fading channel
    BER_th=0.5*(1-sqrt(SNR_fad./(1.5+SNR_fad)));
    plot(EbN0dB2,BER_th,'LineWidth',1.3,'DisplayName',sprintf('theoretical
BER, |h|= %.1f',sigfad(n)))
end
```

```
set(gca,'yscale','log')
hold off
xlim([-6 17])
ylim([1e-3 1])
grid on

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('Effect of Fading Channel Coefficient on BER')
legend("Position", [0.1641 0.2503 0.2661, 0.1583]);
```



2x2 MIMO System

Perfect CSI at Rx only: Zero-Forcing (ZF) Equalizer

Perfect channel state information (CSI): The receiver knows the exact channel conditions (ex: channel matrix H)

Suppose the received signal is represented as: $y = \mathrm{H}x + n$, where x and y are the transimtted and received signal, n is the Gaussian noise, and H is the channel matrix containing channel fading coefficients h_{ij} of each pair of Tx/Rx channel.

Now we want to estimate the transmit signal x with Zero Forcing (ZF) equalizer: $\hat{x}_{\mathrm{ZF}} = (H^H H)^{-1} H^H y$,

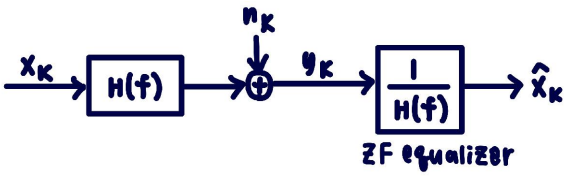
$W_{\mathrm{ZF}} = (H^H H)^{-1} H^H$

ZF equalizer applies the inverse frequency response of the channel to the received signal y to recover the transmitted signal x.

In the frequency domain, the Zero Forcing equalizer aims to recover the transmitted signal $X(f)$ from the received signal $Y(f)$ by applying an inverse filter $\frac{1}{H(f)}$:

$\hat{X}(f) = \frac{1}{H(f)} Y(f)$, where $Y(f) = X(f)H(f) + \text{noise}$ is the received signal

$\hat{X}(f) = \frac{1}{H(f)} \{X(f)H(f) + \text{noise}\} = X(f) + \frac{1}{H(f)} \cdot \text{noise}$

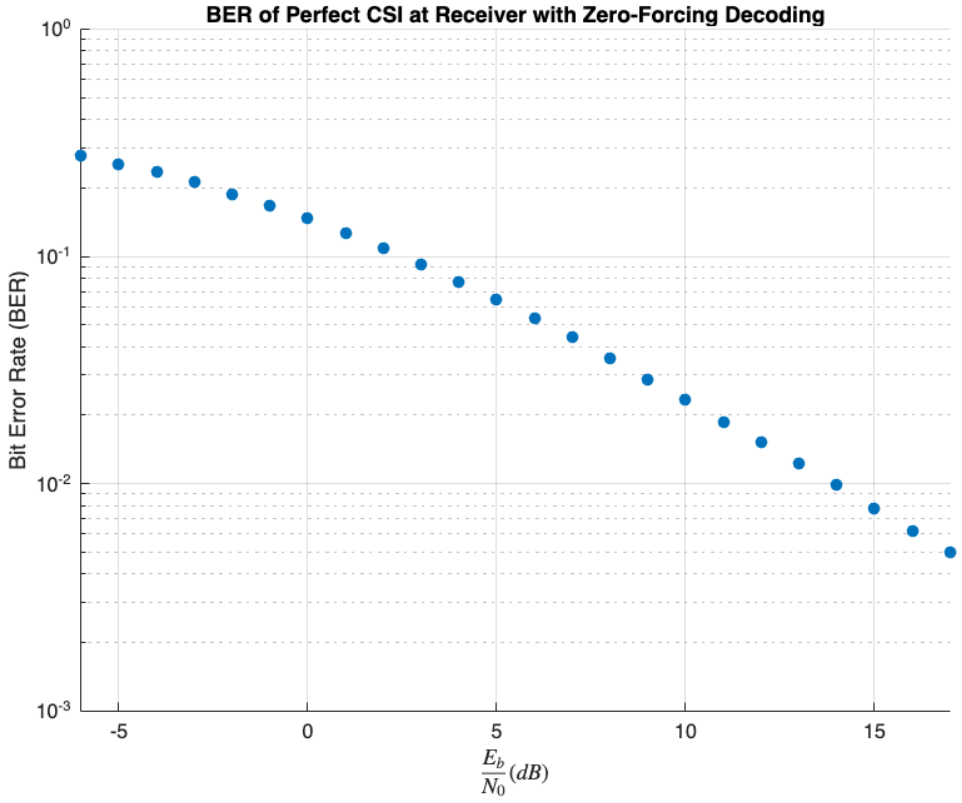


Limitations of ZF equalizer:

The ZF equalizer can amplify noise significantly, especially when the channel response $H(f)$ is very small or approaches zero (deep fades or nulls). In these conditions, the term $\frac{1}{H(f)}$ becomes very large, leading to substantial noise amplification. This effect is exacerbated at low SNR, where the noise level is already high, further degrading the system performance.

```
% Perfect CSI at Rx only
% Zero Forcing (ZF) MIMO receiver
clf;

Nbits=1e4;
Es=1;
Rs=1;
Fs=4;
T=1/Rs;
sps=Fs/Rs;

EsN0dB = -3:1:20;
EbN0dB = EsN0dB-3;
N0 = 10.^(-EsN0dB/10);

Nt = 2; Nr = 2; % number of tx and rx antennas
sigfad = 1;

bitvec = randi([0,1],[Nbits,1]);
```

```
symvec = (1/sqrt(2))*complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1); %
(1x5000)
txsyms = reshape(symvec,floor([Nt,length(symvec)/Nt]));  % X=[x1 x2]'
numCol = length(txsyms(1,:));
ser5=zeros(1,length(EsN0dB));
ber5=zeros(1,length(EsN0dB));
nruns=100;

for i=1:length(EsN0dB)
    total_symbol_error=0;
    total_bit_error=0;
    for k=1:nruns
        y=zeros(size(txsyms));
        x_ZF=zeros(size(txsyms));
        for col=1:numCol
            noise = sqrt(N0(i))*complex(randn(Nr,1),randn(Nr,1)); % (2x1)
            H = sigfad*complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
(2x2)
            y(:,col)=H*txsyms(:,col)+noise;

            % Use Zero Forcing estimation to recover transmit symbols
            x_ZF(:,col)=(H'*H)\(H'*y(:,col));
        end
        x_ZF=reshape(x_ZF,[1,length(symvec)]);
        [decoded_symbol_vector,decoded_bit_vector]=qpskdetect(x_ZF);
        total_symbol_error =
total_symbol_error+sum(decoded_symbol_vector~=symvec')/(Nbits/2);
        total_bit_error =
total_bit_error+sum(xor(bitvec,decoded_bit_vector))/Nbits;
    end
    ser5(i)=total_symbol_error/nruns;
    ber5(i)=total_bit_error/nruns;
end


figure;
scatter(EbN0dB,ber5,'filled');
set(gca,'yscale','log')
xlim([-6 17])
ylim([1e-3 1])
grid on

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('BER of Perfect CSI at Receiver with Zero-Forcing Decoding');
```



Perfect CSI at Rx only: Linear Minimum Mean-Square-Error (LMMSE) Equalizer

Now, we estimate transmit signal x with Linear Minimize Mean Square Error (LMMSE) equalizer:

$$\hat{x}_{\text{LMMSE}} = \left(H^H H + \frac{1}{\text{SNR}} I\right)^{-1} H^H y$$

$$W_{\text{LMMSE}} = \left(H^H H + \frac{1}{\text{SNR}} I\right)^{-1} H^H, \frac{1}{\text{SNR}} = \frac{\sigma_n^2}{\sigma_x^2}$$

The LMMSE equalizer balances the trade-off between recovering the transmitted signal and mitigating noise amplification. It scales the noise term based on SNR, which helps to reduce noise term compared to ZF equalizer, particularly in low SNR conditions.

(1) At high SNR, LMMSE equalizer performs similar as ZF equalizer: $\frac{1}{\text{SNR}} \to 0,\ W_{\text{LMMSE}} \approx W_{\text{ZF}} = (H^H H)^{-1} H^H$

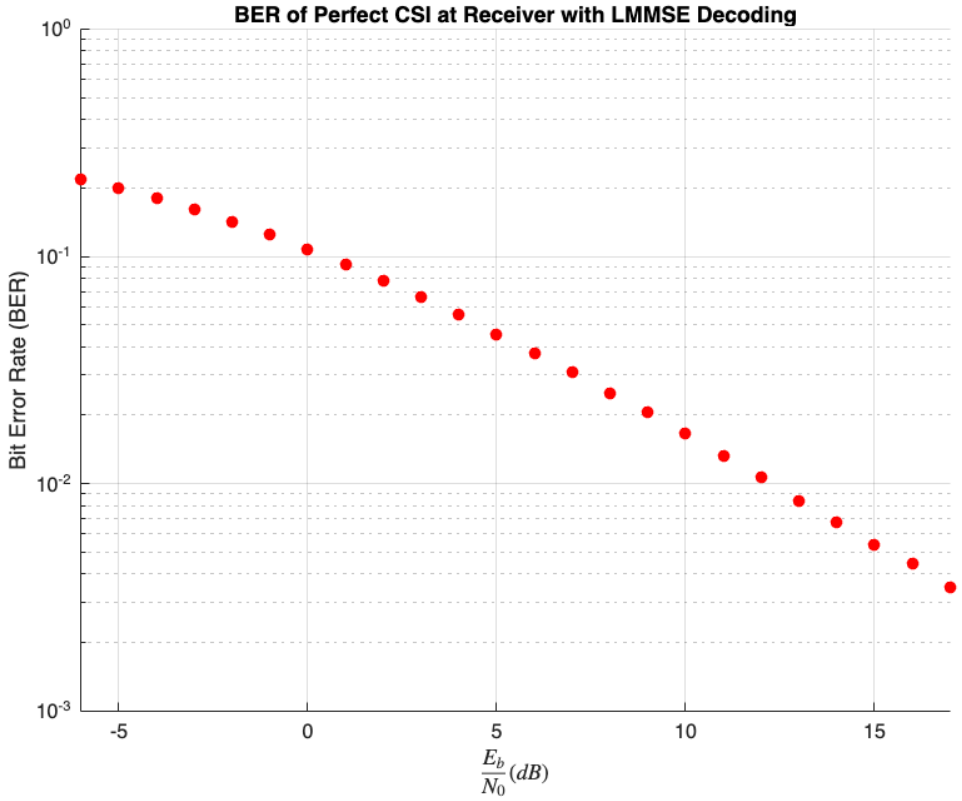
(2) At low SNR, $W_{\text{LMMSE}} \approx \left(\frac{1}{\text{SNR}} I\right)^{-1} H^H = \text{SNR} \cdot H^H$

$$\hat{x}_{\text{LMMSE}} = W_{\text{LMMSE}} \cdot y = (\text{SNR} \cdot H^H) \cdot (H \cdot x + \text{noise}) = \text{SNR} \cdot x + \text{SNR} \cdot H^H \cdot \text{noise}$$

Appying the LMMSE equalizer scales down the noise term at low SNR. While this scaling also reduces the amplitude of the recovered transmitted signal x, it significantly lowers the bit error rate by reducing the noise amplification that occurs at low SNR.

```
% convert SNR to linear scale
```

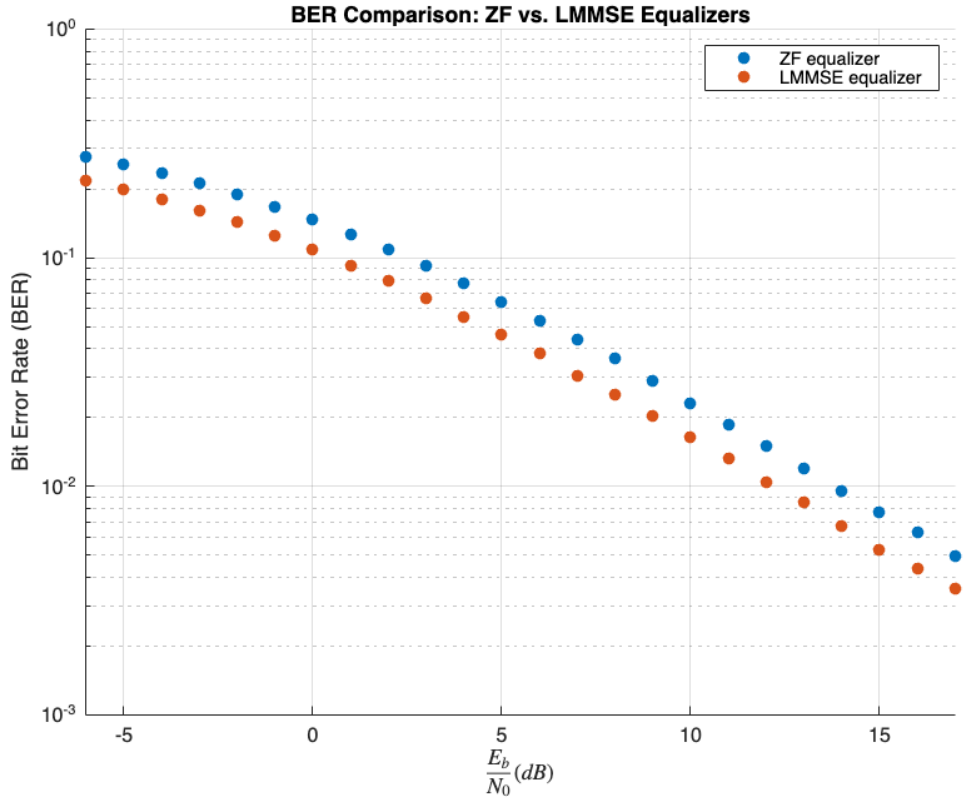
```
SNR=10.^(EsN0dB./10);
ser6=zeros(1,length(EsN0dB));
ber6=zeros(1,length(EsN0dB));
nruns=100;

for i=1:length(EsN0dB)
    total_symbol_error=0;
    total_bit_error=0;
    for k=1:nruns
        y=zeros(size(txsyms));
        x_ZF=zeros(size(txsyms));
        for col=1:numCol
            noise = sqrt(N0(i))*complex(randn(Nr,1),randn(Nr,1)); % (2x1)
            H = sigfad*complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
(2x2)
            y(:,col)=H*txsyms(:,col)+noise;

            % Use Zero Forcing estimation to recover transmit symbols
            x_ZF(:,col)=(H'*H+(1/SNR(i))*eye(Nt))\(H'*y(:,col));
        end
        x_ZF=reshape(x_ZF,[1,length(symvec)]);
        [decoded_symbol_vector,decoded_bit_vector]=qpskdetect(x_ZF);
        total_symbol_error =
total_symbol_error+sum(decoded_symbol_vector~=symvec')/(Nbits/2);
        total_bit_error =
total_bit_error+sum(xor(bitvec,decoded_bit_vector))/Nbits;
    end
    ser6(i)=total_symbol_error/nruns;
    ber6(i)=total_bit_error/nruns;
end


figure;
scatter(EbN0dB,ber6,'filled','r');
set(gca,'yscale','log')
xlim([-6 17])
ylim([1e-3 1])
grid on

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('BER of Perfect CSI at Receiver with LMMSE Decoding');
```



Comparison of BER for ZF and LMMSE Equalizers

The result below shows that the LMMSE equalizer generally has a lower SNR than the ZF equalizer due to its noise scaling. At very high SNR, where the noise becomes extremely small, both equalizers are expected to perform similarly, resulting in converging BERs.

```
load('/Users/tracychiayu/Documents/132A project/data/BER_ZF_equalizer.mat');
load('/Users/tracychiayu/Documents/132A project/data/
BER_LMMSE_equalizer.mat');

figure;
scatter(EbN0dB,ber5,'filled','DisplayName','ZF equalizer');
hold on
scatter(EbN0dB,ber6,'filled','DisplayName','LMMSE equalizer');
set(gca,'yscale','log')
xlim([-6 17])
ylim([1e-3 1])
grid on
hold off

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('BER Comparison: ZF vs. LMMSE Equalizers');
legend;
```



Perfect CSI at Tx and Rx: MIMO for Throughput

In a MIMO system designed for throughput, the goal is to increase the data rate by transmitting different data streams simultaneously through multiple independent channels. By performing Singular Value Decomposition (SVD) on the channel matrix H, we can decompose H into $H = U\Sigma V^H$, where U and V are unitary matrices, and Σ is a diagonal matrix containing the singular values of H. The matrix V is used to precode the transmitted signals, transforming the original MIMO system into a set of parallel independent channels with channel coeffient σ_i. This enables simultaneous transmission of multiple data streams through parallel channels, increasing the system's data rate. The receiver then uses the matrix U^H to post-process the received signal $\tilde{y}$, allowing signal decoding.

Singular Value Decomposition (SVD): $H = U\Sigma V^H$

U is orthonormal and unitary. Suppose $U = [u_1 \dots u_r]$ where u_i is column vector. $u_i \cdot u_j = 0$ if $i \neq j$ and $u_i \cdot u_j = 1$ if $i = j$. $U^H U = UU^H = I$

Σ is a diagonal matrix. For 2x2 MIMO system, $\Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix}$ where $\sigma_1 \geq \sigma_2 \geq 0$ and $\sigma_i's$ are singular values (non-negative real number) of H

V is the unitary matrix, where $V^H V = VV^H = I$ and $V^{-1} = V^H$

Now, we want to transform the orginal MIMO system $\bar{y} = H\bar{x} + \bar{n}$ into a set of parallel indepedent channels.

Process:

1. Precode the transmitted signal $\bar{x}$ using V matrix from $\text{svd}(H)$: $\tilde{x} = V\bar{x}$
2. Pass the precode signal $\tilde{x}$ through the channel. The receiced signal is denoted as $\tilde{y} = H\tilde{x} + \text{noise}$
3. Post-processing received signal $\tilde{y}$ by U^H:

$$\tilde{y} = H\tilde{x} + \text{noise} = (U\Sigma V^H)(V\bar{x}) + \text{noise} = U\Sigma\bar{x} + \text{noise}$$

$$U^H\tilde{y} = U^H U\Sigma\bar{x} + U^H \cdot \text{noise}$$

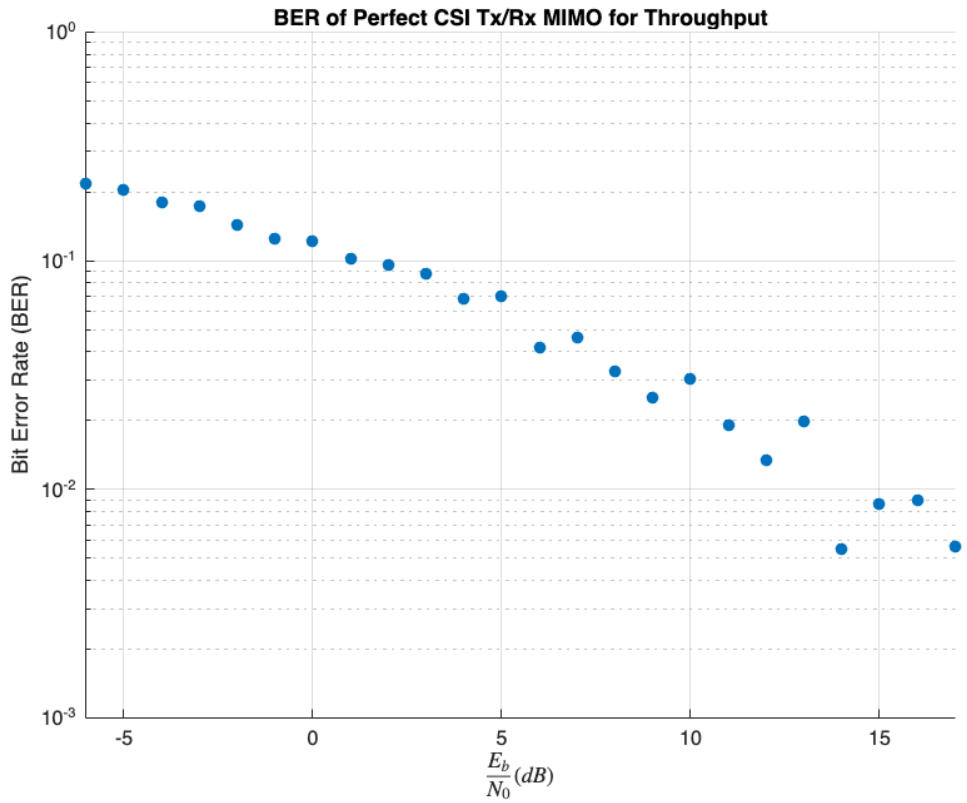
$$\bar{y} = \Sigma\bar{x} + U^H \cdot \text{noise}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} = \begin{bmatrix} \sigma_1 x_1 + n_1 \\ \sigma_2 x_2 + n_2 \end{bmatrix}$$

By precoding the transmitted signal $\bar{x}$ using the matrix V from the $\text{svd}(H)$, we transform the original MIMO system into a set of parallel independent channels.

```
Nbits=1e4;
EsN0dB = -3:1:20;
EbN0dB = EsN0dB-3;
N0 = 10.^(-EsN0dB/10);

Nt = 2; Nr = 2; % number of tx and rx antennas
sigfad = 1;     % |h|
bitvec=randi([0,1],[Nbits,1]);
symvec=complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2); % 1x5000
txsyms = reshape(symvec,floor([Nt,length(symvec)/Nt]));          % 2x2500
numCol = length(txsyms(1,:)); % 2500

% perfect CSI at Tx and Rx - MIMO for throughput
nrun=100;
ser7=zeros(1,length(EsN0dB));
ber7=zeros(1,length(EsN0dB));
for i=1:length(EsN0dB)
    total_bit_error=0;
    total_symbol_error=0;
    for k=1:nruns
        H = sigfad*complex(randn(Nr,Nt),randn(Nr,Nt)); % channel matrix
        [U,S,V] = svd(H);
        rxsyms=zeros(size(txsyms));
        for col=1:numCol

            noise=sqrt(N0(i))*complex(randn(Nr,1),randn(Nr,1));
            xtilde = V*txsyms(:,col) ; % precode txsyms
            ytilde = H*xtilde+noise;
```

```
            rxsyms(:,col)=U'*ytilde; % post-processing

        end

        rxsyms=reshape(rxsyms,[1,length(symvec)]);
        [decoded_symbol_vector,decoded_bit_vector]=qpskdetect(rxsyms);

total_symbol_error=total_symbol_error+sum(decoded_symbol_vector~=symvec')/
(Nbits/2);
        total_bit_error=total_bit_error+sum(xor(decoded_bit_vector,bitvec))/
Nbits;
    end
    ser7(i)=total_symbol_error/nruns;
    ber7(i)=total_bit_error/nruns;
end

figure;
scatter(EbN0dB,ber7,'filled');
set(gca,'yscale','log')
xlim([-6 17])
ylim([1e-3 1])
grid on

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('BER of Perfect CSI Tx/Rx MIMO for Throughput');
```



Perfect CSI at Tx and Rx: MIMO for Diversity

In the previous section, we increased the data rate by transmitting different data streams through a MIMO system. Now, we aim to enhance reliability by transmitting the same data across multiple independent channels, each experiencing different fading effects. By summing the signals received from these channels, the likelihood of the transmitted signal undergoing deep fade is reduced, thereby lowering the bit error rate.

txsyms = repmat(symvec,[1,2]).';

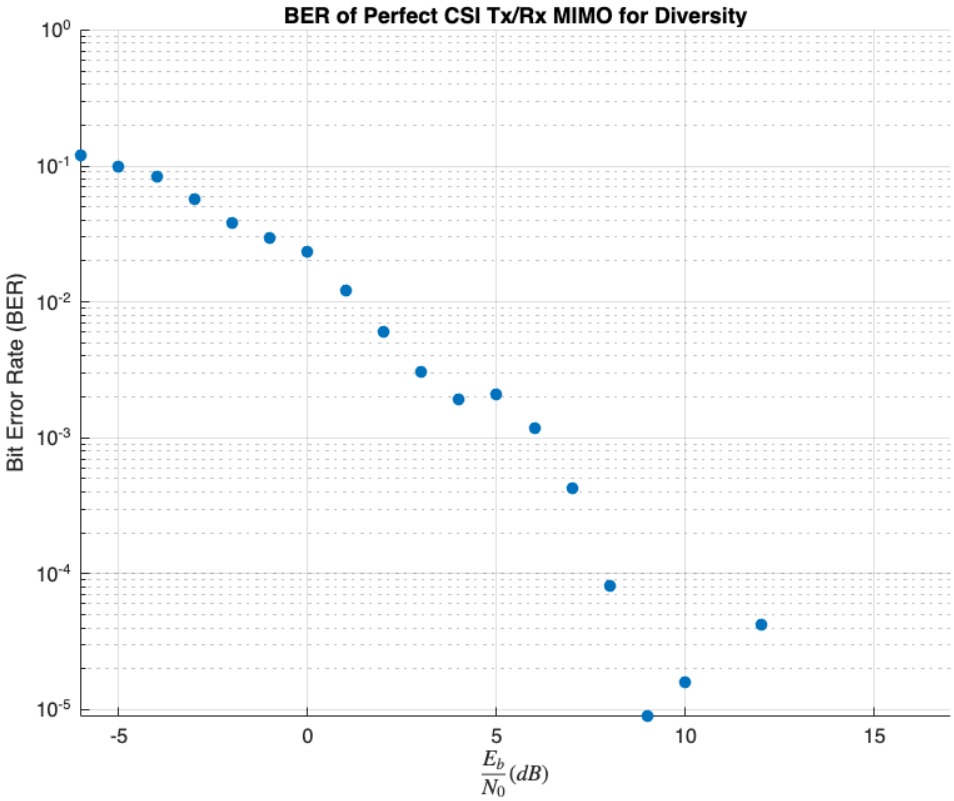
We duplicate the 5000x1 symbol vector 'symvec' to create a 2x5000 transmitted matrix 'txsyms', which is then sent through two independent channels.

rxvec=sum(y).';

'rxvec' is a 5000x1 vector that sums the received signals for each duplicated symbol across multiple channels, which helps enhance the signal and reduce the impact of deep fading on each symbol.

```
% perfect CSI at Tx and Rx - MIMO for diversity
Nbits=1e4;
bitvec=randi([0,1],[Nbits,1]);
symvec=complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2); % 5000x1
txsyms = repmat(symvec,[1,2]).';                                 % 2x5000
replicate the transmitted symbols
numCol=length(symvec);

ser8=zeros(1,length(EsN0dB));
ber8=zeros(1,length(EsN0dB));
nruns=100;
for i=1:length(EsN0dB)
    total_bit_error=0;
    total_symbol_error=0;
    for k=1:nruns
        H=sigfad*complex(randn(Nt,Nr),randn(Nt,Nr));
        [U,S,V]=svd(H);
        y=zeros(size(txsyms));
        for col=1:numCol
            noise=sqrt(N0(i))*complex(randn(Nr,1),randn(Nr,1));
            xtilde=V*txsyms(:,col);  % precode txsyms
            ytilde=H*xtilde+noise;
            y(:,col)=U'*ytilde;      % post-process ytilde
        end
        rxvec=sum(y).';  % 5000x1
        [decoded_symbol_vector,decoded_bit_vector]=qpskdetect(rxvec);
        total_bit_error=total_bit_error+sum(xor(bitvec,decoded_bit_vector))/
Nbits;

total_symbol_error=total_symbol_error+sum(symvec~=decoded_symbol_vector)/
(Nbits/2);
    end
    ser8(i) = total_symbol_error/nruns;
```

```
    ber8(i) = total_bit_error/nruns;
end

figure;
scatter(EbN0dB,ber8,'filled');
set(gca,'yscale','log')
xlim([-6 17])
ylim([0 1])
grid on

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('BER of Perfect CSI Tx/Rx MIMO for Diversity');
```

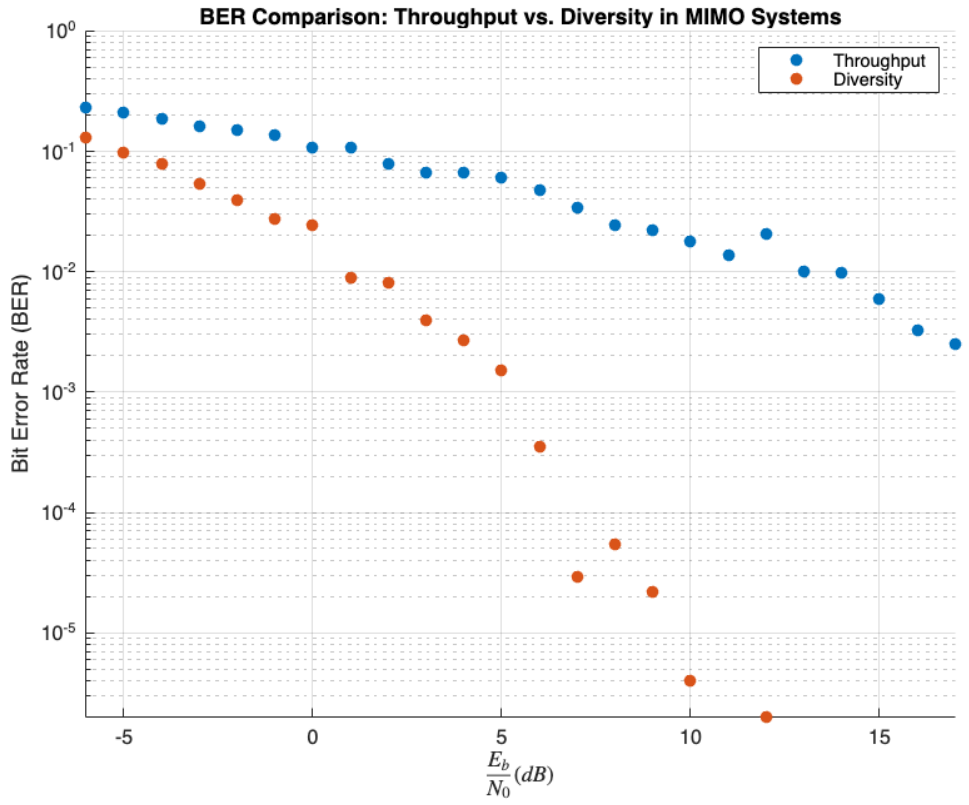


Comparison of BER for Throughput and Diversity in MIMO Systems

The result below shows that MIMO for diversity achieves a lower BER than MIMO for throughput, as it enhances liability by transmitting the same data across multiple independent channels and mitigating deep fading. Conversely, MIMO for throughput increases data rate by sending different data streams through multiple channels but may result in higher BER due to potential deep fading in any channel. Thus, choosing between these approaches involves balancing higher data rates with transmission reliability.

```
load('/Users/tracychiayu/Documents/132A project/data/
BER_of_perfect_csi_MIMO_for_throughput.mat');
load('/Users/tracychiayu/Documents/132A project/data/
BER_of_perfect_csi_MIMO_for_diversity.mat');
```

```
figure;
scatter(EbN0dB,ber7,'filled','DisplayName','Throughput');
hold on;
scatter(EbN0dB,ber8,'filled','DisplayName','Diversity');
set(gca,'yscale','log')
xlim([-6 17])
ylim([0 1])
grid on
hold off;

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('BER Comparison: Throughput vs. Diversity in MIMO Systems');
legend;
```



No CSI at Tx or Rx: Channel Estimation using Pilot Vectors

In this senario, signals are transmitted through a MIMO system without prior knowledge of the channel conditions. To estimate the channel matrix H, we transmit $N_p = 1000$ pilot vectors p_i from the transmitter, where $i = 0, \ldots, N_p - 1$ and $p_i \in C^{2x1}$. The received signal is modeled as $y_i = H \cdot p_i + n$. We then rewrite this as $y_i = P_i \cdot h + n$, where P_i is a 2x4 matrix derived from p_i, and h is an unknown 4x1 vector. By stacking all y_i and

P_i together, we form a $2N_p$x1 vector y and a $2N_p$x4 matrix P repectively. The vector h is then estimated using LMMSE: $h = \left(P^H \cdot P + \frac{1}{\text{SNR}} I\right)^{-1} P^H \cdot y$

The code demonstrates that the norm between the actual channel matrix H and the estimated H is around 0.01, indicating a highly accurate estimation.

Equation Derivation for i^{th} Pilot Vector:

$$y_i = H \cdot p_i + n$$

$$\begin{bmatrix} y_{1i} \\ y_{2i} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} p_{1i} \\ p_{2i} \end{bmatrix} + \begin{bmatrix} n_{1i} \\ n_{2i} \end{bmatrix} = \begin{bmatrix} h_{11}p_{1i} + h_{12}p_{2i} + n_{1i} \\ h_{21}p_{1i} + h_{22}p_{2i} + n_{2i} \end{bmatrix}$$

$$= \begin{bmatrix} p_{1i} & p_{2i} & 0 & 0 \\ 0 & 0 & p_{1i} & p_{2i} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{21} \\ h_{22} \end{bmatrix} + \begin{bmatrix} n_{1i} \\ n_{2i} \end{bmatrix}$$

$$= P \cdot h + n$$

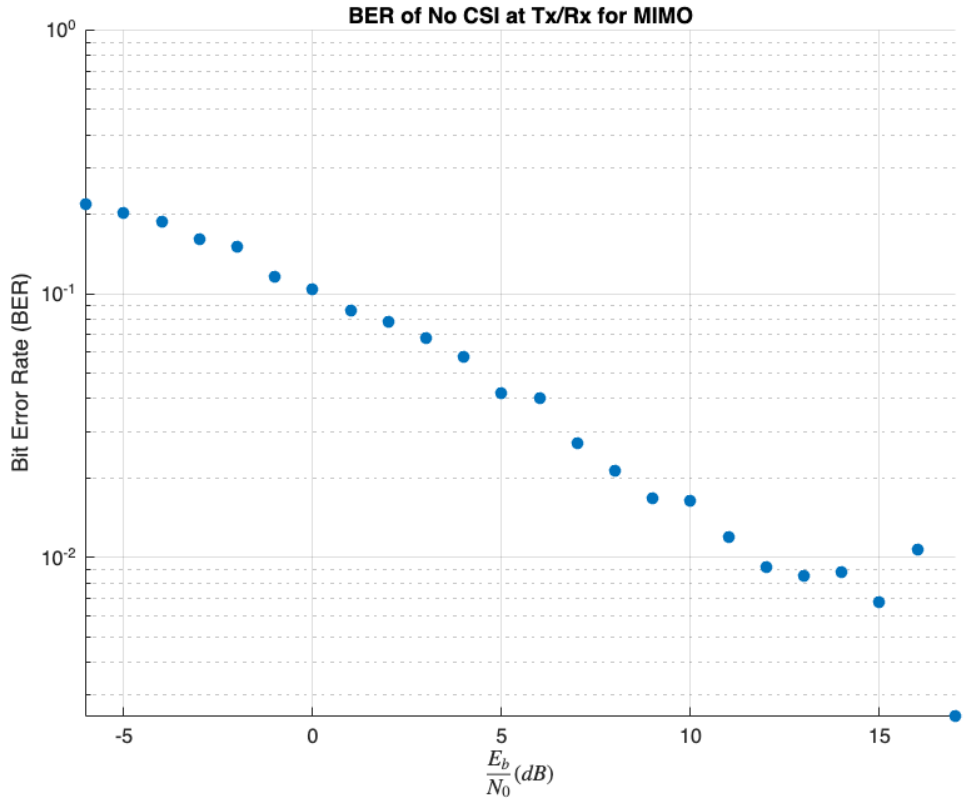
```
EsN0dB = -3:1:20;
EbN0dB = EsN0dB-3;
N0 = 10.^(-EsN0dB/10);
SNR=10.^(EsN0dB./10);

Nt=2; Nr=2;
sigfad=1;

% transmitted data using QPSK modulation
Nbits=1e4;
bitvec=randi([0,1],[1,Nbits]);                                    % 1x10000
symvec=complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2); % 1x5000
txsyms=reshape(symvec,floor([Nt,length(symvec)/Nt]));             % 2x2500

ber9=zeros(1,length(EsN0dB));
ser9=zeros(1,length(EsN0dB));
for i=1:length(EsN0dB)
    total_bit_error=0;
    total_symbol_error=0;
    nruns=100;
    for k=1:nruns

        H=sigfad*complex(randn(Nt,Nr),randn(Nt,Nr));
        % Channel Estimation
        Npilots=1e3; % number of (2x1) pilot vectors
        pilot_bits=randi([0,1],[1,2*(Nt*Npilots)]);
```

```
pilot_symbols=complex(2*pilot_bits(1:2:end)-1,2*pilot_bits(2:2:end)-1)/
sqrt(2); % 1x2000
        pilot_symbols=reshape(pilot_symbols,floor([Nt,length(pilot_symbols)/
Nt]));      % 2x1000

        P=zeros(2*Npilots,4);
        for Np=1:Npilots % go through each pilot vector
            P((2*Np-1):2*Np,:)=[pilot_symbols(:,Np).' 0 0; 0 0
pilot_symbols(:,Np).'];
        end

        noise_pilot=sqrt(N0(i))*complex(randn(2,Npilots),randn(2,Npilots));
% 2x1000
        y_pilot=H*pilot_symbols+noise_pilot;  % 2x1000
        y_pilot=reshape(y_pilot,[2*Npilots,1]);
        h=(P'*P+(1/SNR(i))*eye(4))\(P'*y_pilot);
        estimated_H=reshape(h,[Nt,Nr]).';

        % Transmit actual data
        noise=sqrt(N0(i))*complex(randn(size(txsyms)),randn(size(txsyms)));
        y=H*txsyms+noise;
        x_LMMSE=(estimated_H'*estimated_H+(1/SNR(i))*eye(2))\
(estimated_H'*y); % 2x2500
        x_LMMSE=reshape(x_LMMSE,[1,length(symvec)]); % 1x5000

        [decoded_symbol_vector,decoded_bit_vector]=qpskdetect(x_LMMSE);

total_bit_error=total_bit_error+sum(xor(bitvec',decoded_bit_vector))/Nbits;

total_symbol_error=total_symbol_error+sum(symvec~=decoded_symbol_vector)/
(Nbits/2);
    end
    ber9(i)=total_bit_error/nruns;
    ser9(i)=total_symbol_error/nruns;
end
norm(H-estimated_H,'fro')
```

```
ans = 0.0088
```

```
figure;
scatter(EbN0dB,ber9,'filled');
set(gca,'yscale','log')
xlim([-6 17])
ylim([0 1])
grid on

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
```

```
title('BER of No CSI at Tx/Rx for MIMO with');
```



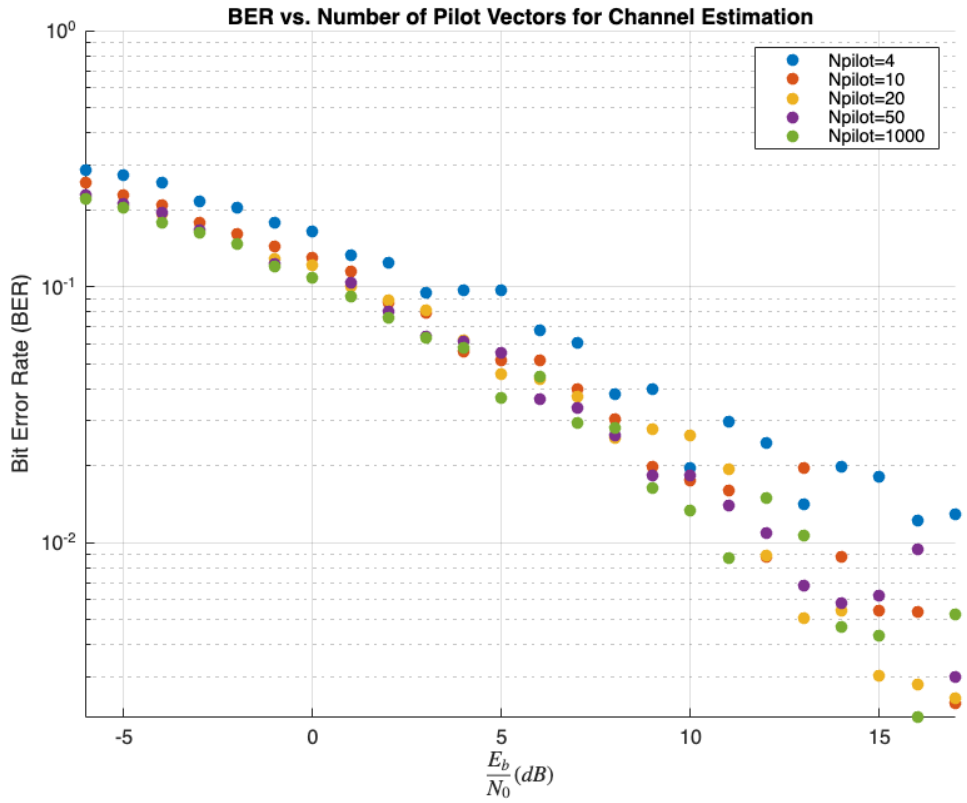
Comparison of BER with Different Number of Pilot Vectors

From the equation $y_i = P_i \cdot h + n$, we see that a minimum of 4 pilot vectors is required to estimate the channel matrix H. The plot below shows that the BER for Npilot=10, 20, 50, and 1000 are quite similar. To balance channel estimation accuracy with the overhead of pilot symbols, choosing Npilot=20 is a practical compromise.

```
% transmitted data using QPSK modulation
Nbits=1e4;
bitvec=randi([0,1],[1,Nbits]);                                    % 1x10000
symvec=complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2); % 1x5000
txsyms=reshape(symvec,floor([Nt,length(symvec)/Nt]));             % 2x2500

close all;
figure;
number_of_pilot_vectors=[4 10 20 50 1000];
for n=1:length(number_of_pilot_vectors)
    Npilots=number_of_pilot_vectors(n); % number of (2x1) pilot vectors
    ber9=zeros(1,length(EsN0dB));
    ser9=zeros(1,length(EsN0dB));
    for i=1:length(EsN0dB)
        total_bit_error=0;
        total_symbol_error=0;
        nruns=100;
```

```
        for k=1:nruns

            H=sigfad*complex(randn(Nt,Nr),randn(Nt,Nr));
            % Channel Estimation
            pilot_bits=randi([0,1],[1,2*(Nt*Npilots)]);

pilot_symbols=complex(2*pilot_bits(1:2:end)-1,2*pilot_bits(2:2:end)-1)/
sqrt(2); % 1x2000

pilot_symbols=reshape(pilot_symbols,floor([Nt,length(pilot_symbols)/
Nt]));      % 2x1000

            P=zeros(2*Npilots,4);
            for Np=1:Npilots % go through each pilot vector
                P((2*Np-1):2*Np,:)=[pilot_symbols(:,Np).' 0 0; 0 0
pilot_symbols(:,Np).'];
            end


noise_pilot=sqrt(N0(i))*complex(randn(2,Npilots),randn(2,Npilots)); % 2x1000
            y_pilot=H*pilot_symbols+noise_pilot;  % 2x1000
            y_pilot=reshape(y_pilot,[2*Npilots,1]);
            h=(P'*P+(1/SNR(i))*eye(4))\(P'*y_pilot);
            estimated_H=reshape(h,[Nt,Nr]).';

            % Transmit actual data

noise=sqrt(N0(i))*complex(randn(size(txsyms)),randn(size(txsyms)));
            y=H*txsyms+noise;
            x_LMMSE=(estimated_H'*estimated_H+(1/SNR(i))*eye(2))\
(estimated_H'*y); % 2x2500
            x_LMMSE=reshape(x_LMMSE,[1,length(symvec)]); % 1x5000

            [decoded_symbol_vector,decoded_bit_vector]=qpskdetect(x_LMMSE);

total_bit_error=total_bit_error+sum(xor(bitvec',decoded_bit_vector))/Nbits;

total_symbol_error=total_symbol_error+sum(symvec~=decoded_symbol_vector)/
(Nbits/2);
        end
        ber9(i)=total_bit_error/nruns;
        ser9(i)=total_symbol_error/nruns;
    end

scatter(EbN0dB,ber9,'filled','DisplayName',sprintf('Npilot=%d',number_of_pil
ot_vectors(n)));
    hold on
end

set(gca,'yscale','log')
```

```
xlim([-6 17])
ylim([0 1])
grid on
hold off

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
title('BER vs. Number of Pilot Vectors for Channel Estimation');
legend;
```



Perfect CSI at Rx: Alamouti Code for $N_t = 2$ MIMO System

The Alamouti code is a space-time block code used for MIMO system with $N_t = 2$ transmitters and N_r receivers. Its main advantage is that it achieves full diversity gain with relatively low complexity. Additionally, it does not require channel state information at the transmitter, which simplifies the implementation. However, the Alamouti code is restricted to 2 transmitters, which limits its ability to provide higher spatial diversity.

Alamouti Encoding:

For simplicity, we first explain the case with $N_r = 1$ which the channel matrix can be denoted as $H = \begin{bmatrix} h_0 & h_1 \end{bmatrix}$.

Suppose we're transmitting 5000 symbols, $\text{symvec} = \begin{bmatrix} s_0 & s_1 & \dots & s_{4999} \end{bmatrix}^T$, with 2 symbols transmitted at a time.

The transmitters send symbols $\begin{bmatrix} s_0 \\ s_1 \end{bmatrix}$ on the two antennas at t=0, and $\begin{bmatrix} -s_1^* \\ s_0^* \end{bmatrix}$ at t=1. This pattern is repeated for the rest of the symbols.

Now, we transform 'symvec' into a transmitted symbol matrix 'txsyms' using Alaouti encoding. Each column of txsyms represents time instant from t=0 to t=4999.

$$\text{txsyms} = \begin{bmatrix} s_0 & -s_1^* & s_2 & -s_3^* & \dots & -s_{4999}^* \\ s_1 & s_0^* & s_3 & s_2^* & \dots & s_{4998}^* \end{bmatrix}$$

We then transmit this symbol matrix though the channels, where
$y = H \cdot \text{txsyms} + \text{noise} = \begin{bmatrix} y_0(0) & y_0(1) & \dots & y_0(4999) \end{bmatrix}$ is the received signals.

$y_r(t)$ is denoted as the signal received at the $(r+1)^{\text{th}}$ receiver at $t = 0$.

In absence of noise,

$y_0(0) = h_0 s_0 + h_0 s_0$

$y_0(1) = h_1 s_0^* - h_0 s_1^*$

This can be rewritten as $y_{\text{ala}} = H_{\text{ala}} \cdot s \rightarrow \begin{bmatrix} y_0(0) & \dots & y_0(4998) \\ y_0(1)^* & \dots & y_0(4999)^* \end{bmatrix} = \begin{bmatrix} h_0 & h_1 \\ h_1^* & -h_0^* \end{bmatrix} \begin{bmatrix} s_0 & \dots & s_{4998} \\ s_1 & \dots & s_{4999} \end{bmatrix}$

To recover symbol matrix 's', we multiply both sides by ${H_{\text{ala}}}^H$, resulting in ${H_{\text{ala}}}^H \cdot y = \left(|h_0|^2 + |h_1|^2\right) \cdot s$.

The symbols are then decoded as $s = \frac{1}{|h_0|^2 + |h_1|^2} \cdot {H_{\text{ala}}}^H \cdot y$

For $N_r = 2,\ H = \begin{bmatrix} h_{00} & h_{01} \\ h_{10} & h_{11} \end{bmatrix}$

$$y = H \cdot \text{txsyms} + \text{noise} = \begin{bmatrix} y_0(0) & y_0(1) & \dots & y_0(4999) \\ y_1(0) & y_1(1) & \dots & y_1(4999) \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} \\ h_{01}^* & -h_{00}^* \\ h_{10} & h_{11} \\ h_{11}^* & -h_{10}^* \end{bmatrix} \begin{bmatrix} s_0 & \dots & s_{4998} \\ s_1 & \dots & s_{4999} \end{bmatrix}$$

$$y_{\text{ala}} = H_{\text{ala}} \cdot s \rightarrow \begin{bmatrix} y_0(0) & \dots & y_0(4998) \\ y_0(1)^* & \dots & y_0(4999)^* \\ y_1(0) & \dots & y_1(4998) \\ y_1(1)^* & \dots & y_1(4999)^* \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} \\ h_{01}^* & -h_{00}^* \\ h_{10} & h_{11} \\ h_{11}^* & -h_{10}^* \end{bmatrix} \begin{bmatrix} s_0 & \dots & s_{4998} \\ s_1 & \dots & s_{4999} \end{bmatrix}$$

$$s = \frac{1}{|h_{00}|^2 + |h_{01}|^2 + |h_{10}|^2 + |h_{11}|^2} \cdot {H_{\text{ala}}}^H \cdot y$$

This process can be extended to the case where $N_r > 2$

The results show that MIMO system with $N_r = 3$ receivers performs better than system with $N_r = 1$ receiver. With more receive antennas, the system can receive multiple copies of the transmitted signal, each experiencing different channel conditions. This spatial diversity helps in averaging out the effects of fading, improving overall signal quality and reducing the BER.

```
% Alamouti (Nt = 2)
EsN0dB = -3:1:20;
EbN0dB = EsN0dB-3;
N0 = 10.^(-EsN0dB/10);

Nt=2;
num_Rx=[1 2 3];
Nbits=1e4;
sigfad=1;
bitvec = randi([0,1],[Nbits,1]);
symvec = complex(2*bitvec(1:2:end)-1,2*bitvec(2:2:end)-1)/sqrt(2);

txsyms = zeros(Nt,length(symvec)); % 2x5000
for k = 1:length(symvec)/2
    txsyms(:,2*k-1) = symvec((2*k-1):2*k);
    txsyms(:,2*k) = [-conj(symvec(2*k));conj(symvec(2*k-1))];
end

nrun=100;

for r=1:length(num_Rx)
    Nr=num_Rx(r);
    ber10=zeros(1,length(EsN0dB));
    ser10=zeros(1,length(EsN0dB));
    for i=1:length(EsN0dB)
        total_bit_error=0;
        total_symbol_error=0;
        for k=1:nrun
            H=sigfad*complex(randn(Nr,Nt),randn(Nr,Nt));

noise=N0(i)*complex(randn(Nr,length(symvec)),randn(Nr,length(symvec)));
            y=H*txsyms+noise;

            if Nr==1

                yala=zeros(2*Nr,length(symvec)/2);
                for n=1:length(symvec)/2
                    tmp=y(:,(2*n-1):2*n);
                    yala(:,n)=[tmp(1,1); conj(tmp(1,2))];
                end
                Hala=[H(1,1) H(1,2); conj(H(1,2)) -conj(H(1,1))]; % 2x2
```

```
            elseif Nr==2

                yala=zeros(2*Nr,length(symvec)/2);
                for n=1:length(symvec)/2
                    tmp=y(:,(2*n-1):2*n);
                    yala(:,n)=[tmp(1,1); conj(tmp(1,2)); tmp(2,1);
conj(tmp(2,2));];
                end
                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))]; % 4x2

            elseif Nr==3

                yala=zeros(2*Nr,length(symvec)/2);
                for n=1:length(symvec)/2
                    tmp=y(:,(2*n-1):2*n);
                    yala(:,n)=[tmp(1,1); conj(tmp(1,2)); tmp(2,1);
conj(tmp(2,2)); tmp(3,1); conj(tmp(3,2))];
                end
                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));
                      H(3,1) H(3,2); conj(H(3,2)) -conj(H(3,1))]; % 4x2

            end

            s=(Hala'*yala)./norm(H,'fro')^2;
            rxvec=reshape(s,[1,length(symvec)]).';
            [decoded_symbol_vector,decoded_bit_vector]=qpskdetect(rxvec);

total_bit_error=total_bit_error+sum(xor(bitvec,decoded_bit_vector))/Nbits;

total_symbol_error=total_symbol_error+sum(symvec~=decoded_symbol_vector)/
(Nbits/2);
        end
        ber10(i)=total_bit_error/nruns;
        ser10(i)=total_symbol_error/nruns;
    end
    scatter(EbN0dB,ber10,'filled','DisplayName',sprintf('Nr=%d',Nr));
    hold on
end

set(gca,'yscale','log')
xlim([-6 17])
ylim([0 1])
grid on
hold off

xlabel('$\frac{E_b}{N_0} (dB)$', 'Interpreter', 'latex');
ylabel('Bit Error Rate (BER)');
```

```
title('BER vs. Number of Receivers in Nrx2 MIMO with Perfect CSI at Rx');
legend;
```