EE5141 Wireless Communication

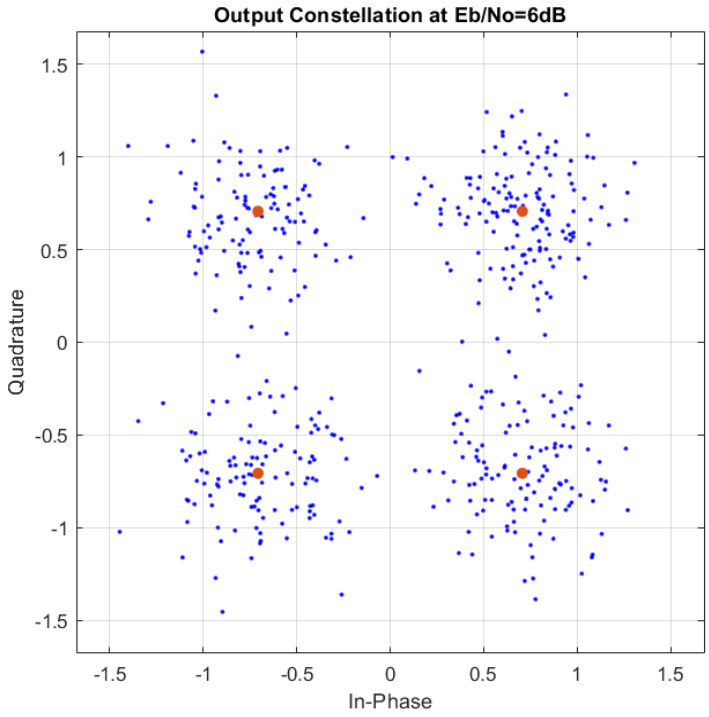
Computer Assignment 2

K R Srinivas (EE18B136)

The task is to build a QPSK modem and study the Bit Error Rate (BER) and Symbol Error Rate (SER) performance using Monte-Carlo simulations, as given in the Figure. Generate a random sequence of about 512 QPSK symbols (1024 bits) and apply pulse shaping with an SRRC pulse (roll off (alpha) = 0.35).

Bits	00	01	10	11
QPSK Symbol	$e^{j\frac{\pi}{4}}$	$e^{j\frac{3\pi}{4}}$	$e^{-j\frac{\pi}{4}}$	$e^{-j\frac{3\pi}{4}}$

A. Visualize received symbols for the case Eb/No = 6dB (in the presence of AWGN)

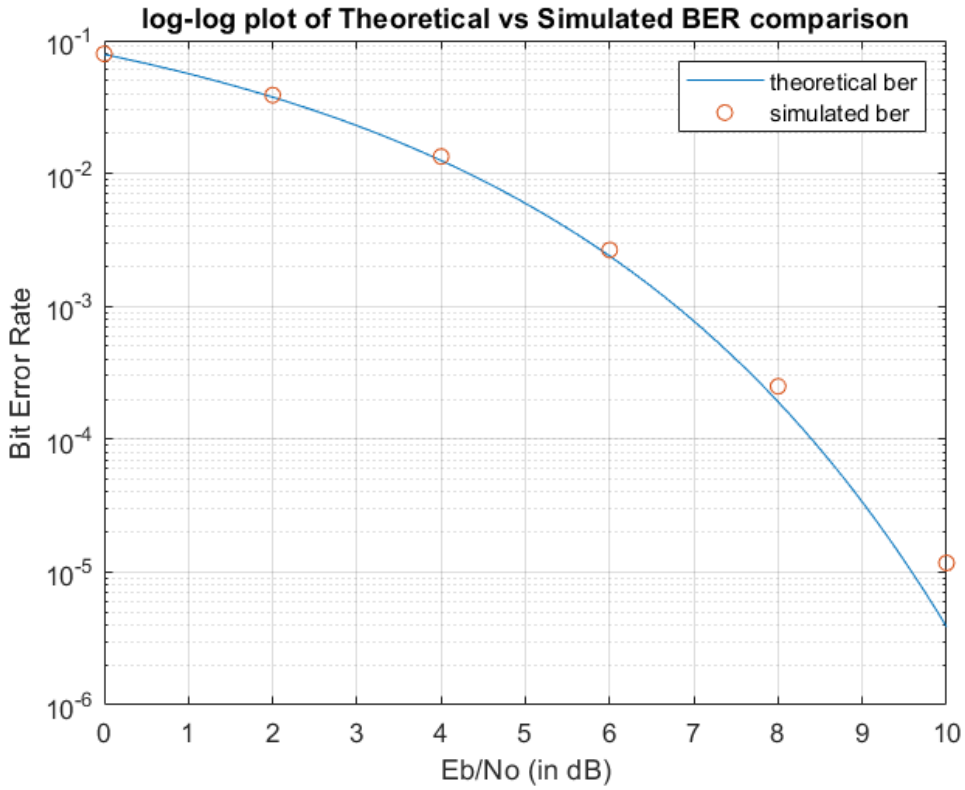


For Eb/No = 6dB, we generate the noise, convolve it with a SRRC signal and add it to the RC pulse shaped signal. Then, optimally sampling the resulting signal, the received symbols are plotted above.

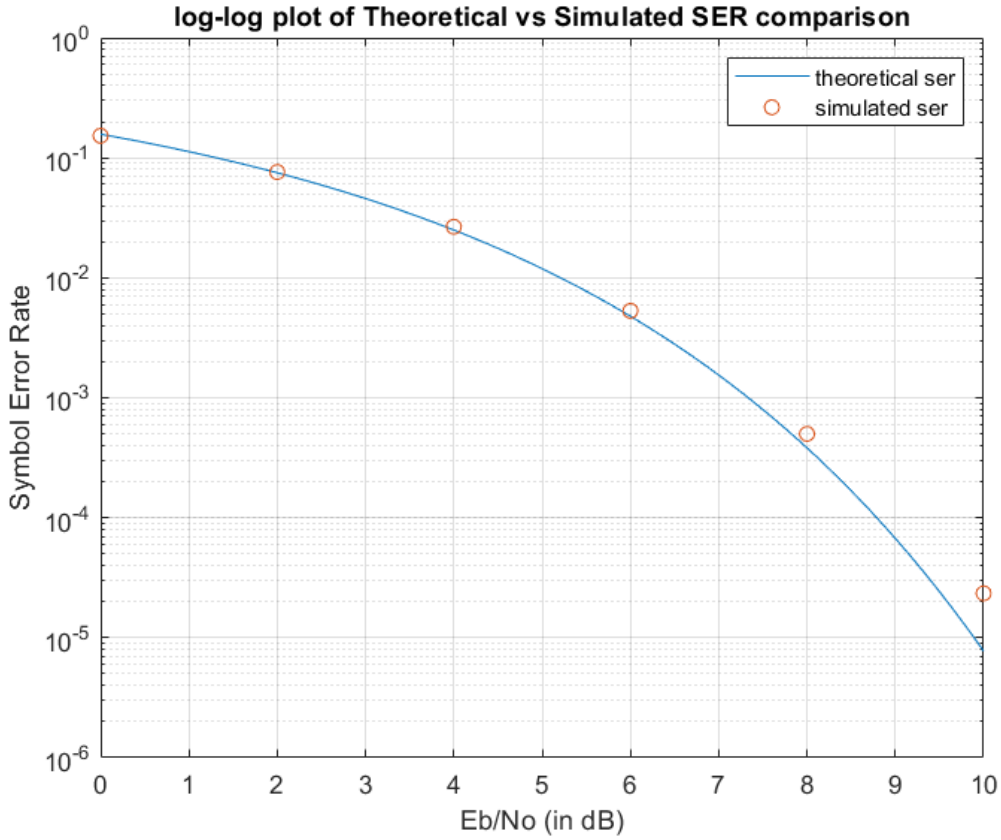
The received signal is plotted in blue and the original location of the reference symbol is plotted in red. The symbols should ideally have been limited to 4 specific points mentioned in the table. But the presence awgn noise distorts and hence scatters the transmitted symbol.

ML decoding is used to predict the transmitted symbol from this constellation and compute the corresponding BER and SER.To analyze the effect of noise level on BER and SER we vary Eb/No given constant Tx signal power. The above procedure is repeated 500 times and the BER and the SER averages are calculated.

B. SER and BER Plots (Analytical vs Simulation) - (C,D)



$$BER = Q\left(\sqrt{\frac{2Eb}{No}}\right)$$



$$SER = 2 * BER$$

We can see that the theoretical BER and SER match the simulations. We observe that both BER and SER decrease with increase in Eb/No value.The BER/SER deviates at higher SNR's as there are fewer errors made and the sample size is small.

SELF DECLARATION

I certify that this assignment submission is my own work and not obtained from any other source.

K. R. Srinivas

1 A

```
%% EE5141 Wireless Communication : Computer Assignment 2
%  Name : K.R.Srinivas
%  Roll No : EE18B136

clear all;
close all;

alpha = 0.35;
sym_rate=25e3;
oversampling=8;
t_bnd = 5/sym_rate;
t_res = 1/(oversampling*sym_rate);
ebno = 6;

ind = 1:1:4;
QPSK_constellation = exp((2.*ind - 1)*pi*1j/4);

ip_bits = randi([0 1], 1, 1024);
tx_sym = qpsk_mod(ip_bits);

[tx_signal,t_arr] = RC_pulse_shaping(tx_sym, t_bnd, t_res, sym_rate, alpha);

rx_signal = awgn_channel(tx_signal, ebno, t_bnd, t_res, sym_rate, alpha);

rx_sym = opt_sample_RC(rx_signal, t_arr*sym_rate, size(tx_sym,2), t_res);
scatterplot(rx_sym);
hold on ;
scatter(real(QPSK_constellation), imag(QPSK_constellation), 'filled');

title("Output Constellation at Eb/No=6dB");
grid on ;
```

2 B

```
close all ;
clear all ;

alpha = 0.35;
sym_rate=25e3;
oversampling=8;
t_bnd = 5/sym_rate;
t_res = 1/(oversampling*sym_rate);
ebno = (0:2:10);
```

```
ber_sim = zeros(size(ebno));
ser_sim = zeros(size(ebno));

for i=1:size(ebno,2)
    [~, ber_sim(i), ~, ser_sim(i)] = monte_carlo_func(ebno(i), alpha, sym_rate, t_bnd, t_res
end

ebno_fine = (0:0.1:10);
ber_theory = normcdf(-sqrt(2.*(10.^(ebno_fine./10))));
ser_theory = 2*ber_theory;

figure;
semilogy(ebno_fine,ber_theory);
hold on;
scatter(ebno,ber_sim);
legend("theoretical ber","simulated ber");
title("log-log plot of Theoretical vs Simulated BER comparison");
xlabel("Eb/No (in dB)");
ylabel("Bit Error Rate");
grid on;

figure;
semilogy(ebno_fine,ser_theory);
hold on;
scatter(ebno,ser_sim);
legend("theoretical ser","simulated ser");
title("log-log plot of Theoretical vs Simulated SER comparison");
xlabel("Eb/No (in dB)");
ylabel("Symbol Error Rate");
grid on;

function [bit_errors, ber, sym_errors, ser] = monte_carlo_func(ebno, alpha, sym_rate, t_bnd
                                                t_res)
    burst_len = 500;

    bit_err_arr = zeros(1,burst_len);
    sym_err_arr = zeros(1,burst_len);
    ber_arr = zeros(1,burst_len);
    ser_arr = zeros(1,burst_len);

    for i=1:burst_len
        ip_bits = randi([0 1], 1, 1024);
        tx_sym = qpsk_mod(ip_bits);

        [tx_signal,t_arr] = RC_pulse_shaping(tx_sym, t_bnd, t_res, sym_rate, alpha);
```

```
        rx_signal = awgn_channel(tx_signal, ebno, t_bnd, t_res, sym_rate, alpha);

        rx_sym = opt_sample_RC(rx_signal, t_arr*sym_rate, size(tx_sym,2), t_res);

        [op_bits, op_sym] = ml_demod(rx_sym);
        [bit_err_arr(i), ber_arr(i)] = biterr(ip_bits,op_bits);
        [sym_err_arr(i), ser_arr(i)] = symerr(tx_sym,op_sym);
    end
    bit_errors = mean(bit_err_arr);
    ber = mean(ber_arr);
    sym_errors = mean(sym_err_arr);
    ser = mean(ser_arr);

end

function symbols = qpsk_mod(bits)

    symbols = zeros(1,size(bits,2)/2);
    bit_to_buff = @(x,y) 2*x + y + 1;
    buff_to_sym = [exp(j*pi/4), exp(j*3*pi/4), exp(-j*pi/4), exp(-j*3*pi/4)];

    for i=1:size(symbols,2)
        symbols(i) = buff_to_sym(bit_to_buff(bits(2*i-1),bits(2*i)));
    end
end

function [x, t_arr_rc] = RC_pulse_shaping(ip_sym, t_bnd, t_res, sym_rate, alpha)

    T = 1/sym_rate;

    t_arr_srrc = (-t_bnd : t_res : t_bnd);
    srrc_arr = srrc_gen(t_arr_srrc, alpha, sym_rate);

    rc_arr = conv(srrc_arr,srrc_arr)*(t_res*sym_rate);

    t_arr_rc = (-2*t_bnd : t_res : (size(ip_sym,2)*T+t_bnd));
    rc_arr = [rc_arr', zeros(1,floor((size(ip_sym,2)*T - t_bnd)/t_res))];
    len = size(rc_arr,2);

    x = zeros(size(t_arr_rc));
    for i = 1:size(ip_sym')
        shift_amt = floor((i-1)*T/t_res);
        %x = x + ip_sym(i)*circshift(rc_arr,(i-1)*T/t_res);
        x = x + ip_sym(i)*[rc_arr(len-shift_amt+1:len), rc_arr(1:len-shift_amt)];
    end
```

```
end

function [op_signal] = awgn_channel(ip_signal, ebno, t_bnd, t_res, sym_rate, alpha)

    snr = ebno + 3;
    snr_mag = 10^(snr/10);

    wg = wgn(size(ip_signal,2),1,0,'complex')';

    t_arr_srrc = (-t_bnd : t_res : t_bnd);
    srrc_arr = srrc_gen(t_arr_srrc, alpha, sym_rate);
    wg_bl = conv(wg,srrc_arr,'same')*(t_res*sym_rate);

    %Ps = sum(abs(ip_signal).^2)/size(ip_signal,2);
    Pn = sum(abs(wg_bl).^2)/size(wg_bl,2);
    beta = sqrt(1 / (Pn * snr_mag));

    op_signal = ip_signal + (beta*wg_bl);

end

function [x_sym] = opt_sample_RC(x_arr, t_arr, n, t_res)

    x_sym = zeros(1,n);
    for i=1:n
        ind = find(abs(t_arr - (i-1)) < t_res/2);
        x_sym(i) = x_arr(ind);
    end
end

function [op_bits, op_sym] = ml_demod(ip_symbols)

    op_bits = zeros(1,2*size(ip_symbols,2));
    op_sym = zeros(1,size(ip_symbols,2));

    for i=1:size(ip_symbols,2)
        if (real(ip_symbols(i))>=0 && imag(ip_symbols(i))>=0)
            op_bits(2*i-1) = 0;
            op_bits(2*i) = 0;
            op_sym(i) = exp(j*pi/4);
        elseif (real(ip_symbols(i))<0 && imag(ip_symbols(i))>0)
            op_bits(2*i-1) = 0;
            op_bits(2*i) = 1;
            op_sym(i) = exp(j*3*pi/4);
        elseif (real(ip_symbols(i))>=0 && imag(ip_symbols(i))<=0)
```

```
            op_bits(2*i-1) = 1;
            op_bits(2*i) = 0;
            op_sym(i) = exp(-j*pi/4);
        else
            op_bits(2*i-1) = 1;
            op_bits(2*i) = 1;
            op_sym(i) = exp(-j*3*pi/4);
        end
    end
end
```