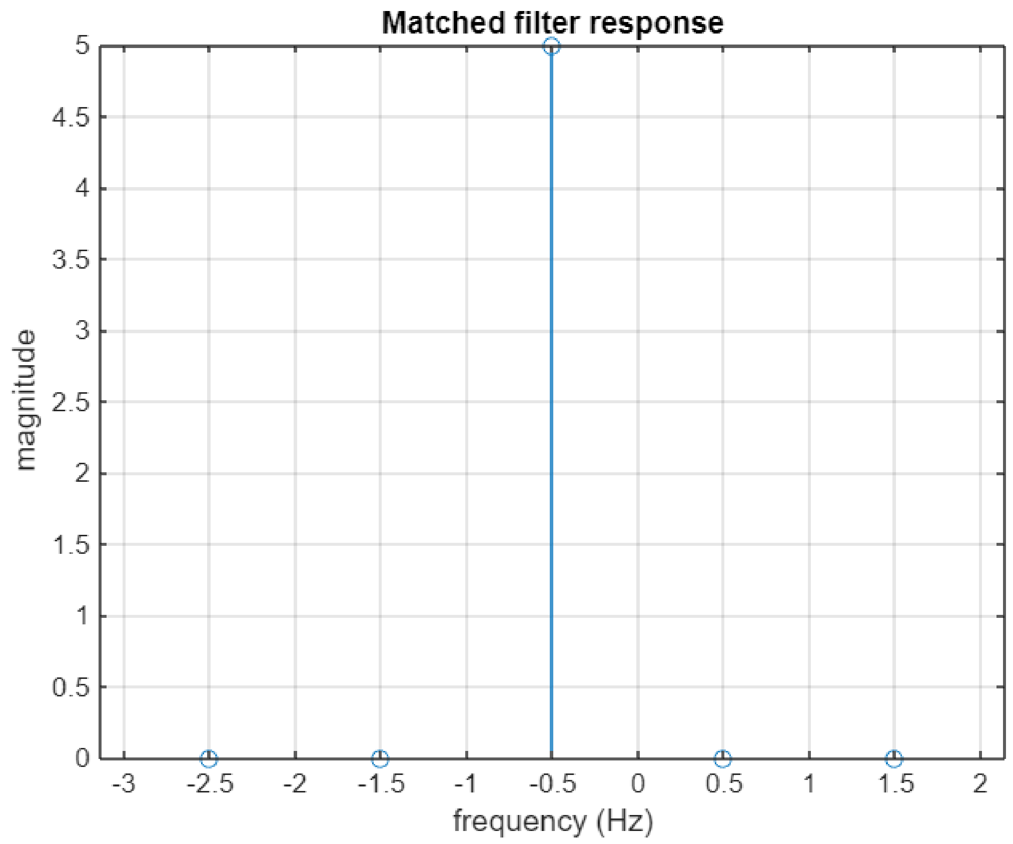
Communication System Lab 3

By: Saurabh Kumar (SC22B146)

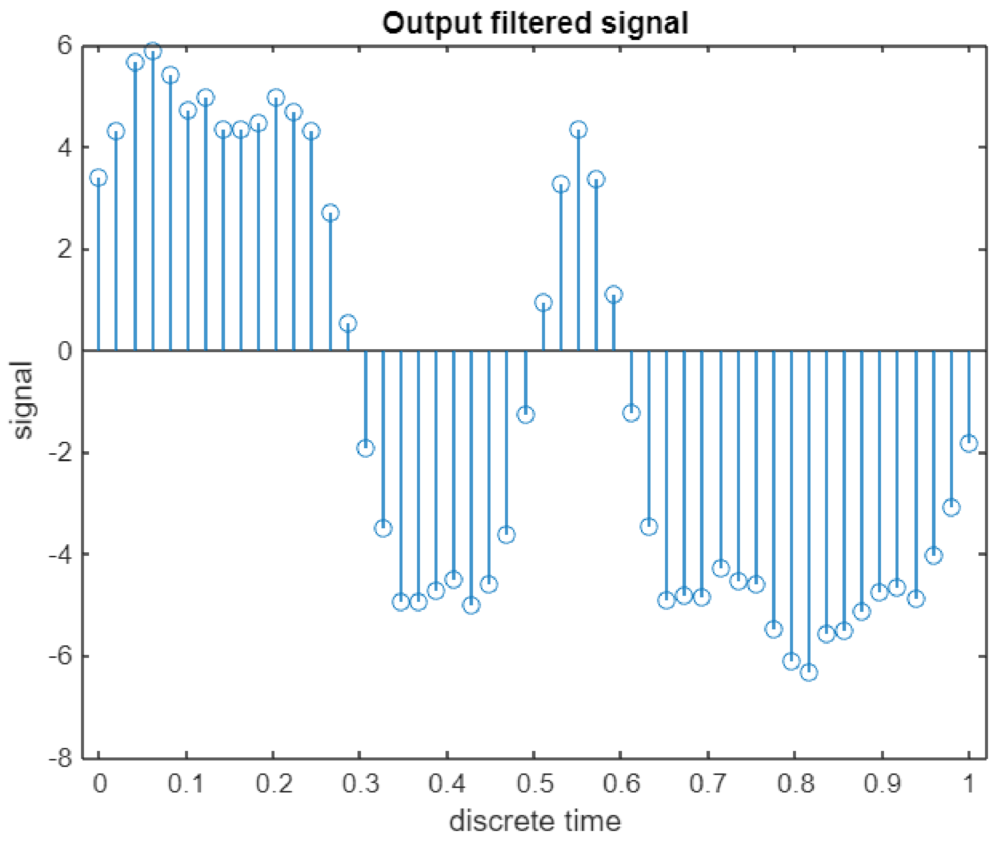
3.3 Matched filtering

1. At the output of the ideal channel, implement the matched filter for the pulse shape that you have used. Plot the magnitude response of the filter that you have implemented.

```
% bits and signal generation
N = 10;
source = rand(1, N) > 0.5;
T = 0.1; fs1 = 50; n = T*fs1;
signal = zeros(1,n*N);
for i=1:(length(source)-1)
    if i==1
       signal(1:n) = repelem(source(1),n);
    else
       signal((i-1)*n+1:(i)*n) = repelem(source(i),n);
    end
end
signal(signal==0) = -1;

% channel and received signal
fs = 160; fc = 20;
channel = firpm(200, [0 fc fc+10, fs/2]/(fs/2), [1 1 0 0]);
output_signal = conv(signal, channel, 'same');
noise_var = 0.05;
noise = sqrt(noise_var) * randn(1, length(output_signal));
output_signal_noisy = output_signal + noise;

% matched filter
filter_matched = fliplr(ones(1,n));
f = -(length(filter_matched)/2):(length(filter_matched)/2-1);
figure;
stem(f,fftshift(abs(fft(filter_matched))));
title("Matched filter response");
xlabel("frequency (Hz)");
ylabel("magnitude");
grid on;
```



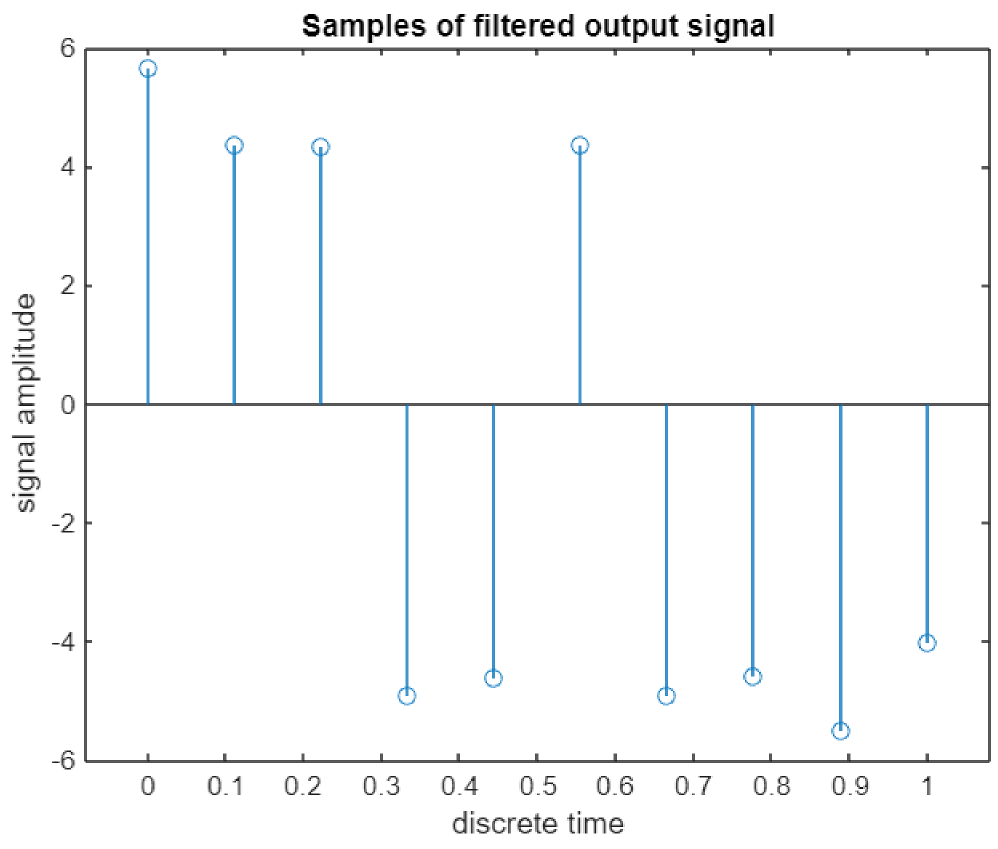
```
% filtered signal
output_signal_filtered = conv(output_signal_noisy, filter_matched, 'same');
figure;
t = linspace(0,1,N*0.1*50);
stem(t,output_signal_filtered);
title("Output filtered signal");
xlabel("discrete time");
ylabel("signal");
```



Inference: Here, the matched filter is implemented using 'fliplr' function which flips and shifts the input signal. The signal is then simulted to pass through the channel and the matched filter.

2. Obtain samples from the output of the matched filter; in this set of tasks the sampling times can be manually set by inspection of the channel output signal.

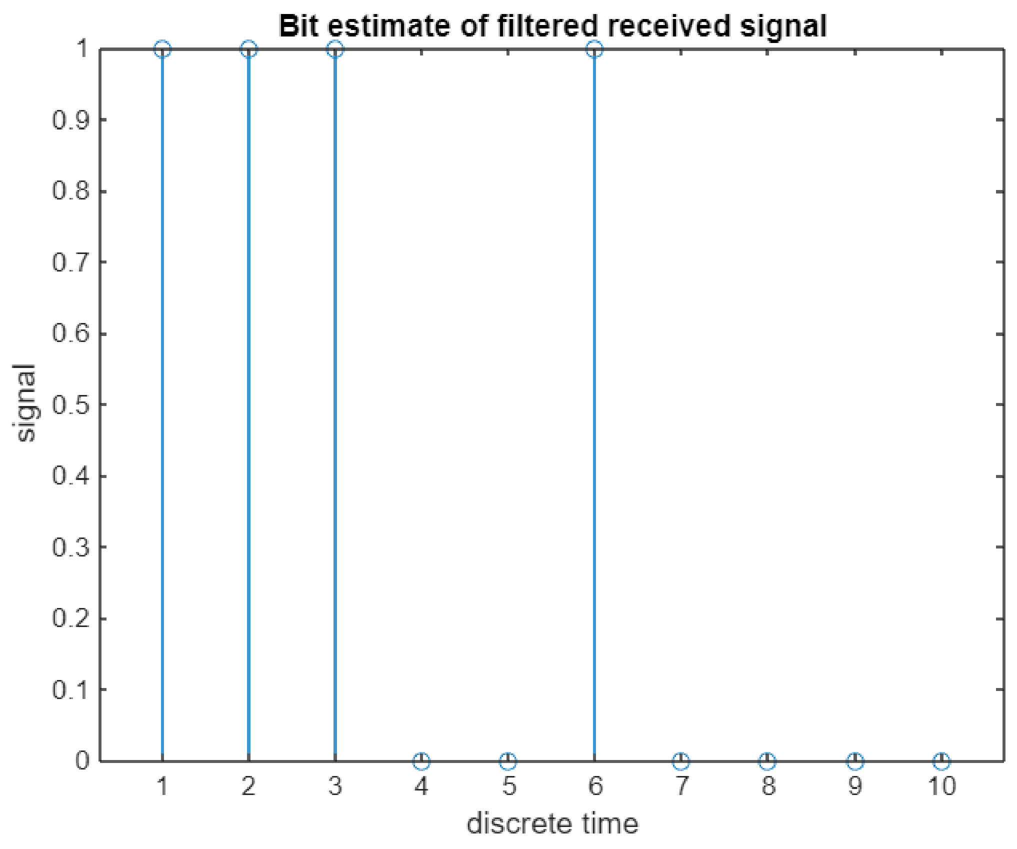
```
n=1:N;
sampling_times = (n-0.5)*T;
samples_filtered = output_signal_filtered(round(sampling_times*50));
figure;
t = linspace(0,1,N);
stem(t,samples_filtered);
title("Samples of filtered output signal");
xlabel("discrete time");
ylabel("signal amplitude");
```



Inference: The sample of the output is taken by sampling the received signal at the mid-Ts.

3. Implement a decision making device - a thresholder that will convert the samples to 0 or 1, and obtain the bit estimates at the output of the decision device.

```
% thresholding sampled filtered signal
threshold = 0;
bit_estimate_filtered = zeros(1,N);
for i = 1:length(samples_filtered)
    bit_estimate_filtered(i) = samples_filtered(i) > threshold;
end

figure;
stem(bit_estimate_filtered);
title("Bit estimate of filtered received signal");
xlabel("discrete time");
ylabel("signal");
```



```
BER = sum(bit_estimate_filtered ~= source)/length(source)*100;
disp(['BER is ', num2str(BER), '.']);
```

```
BER is 0.
```

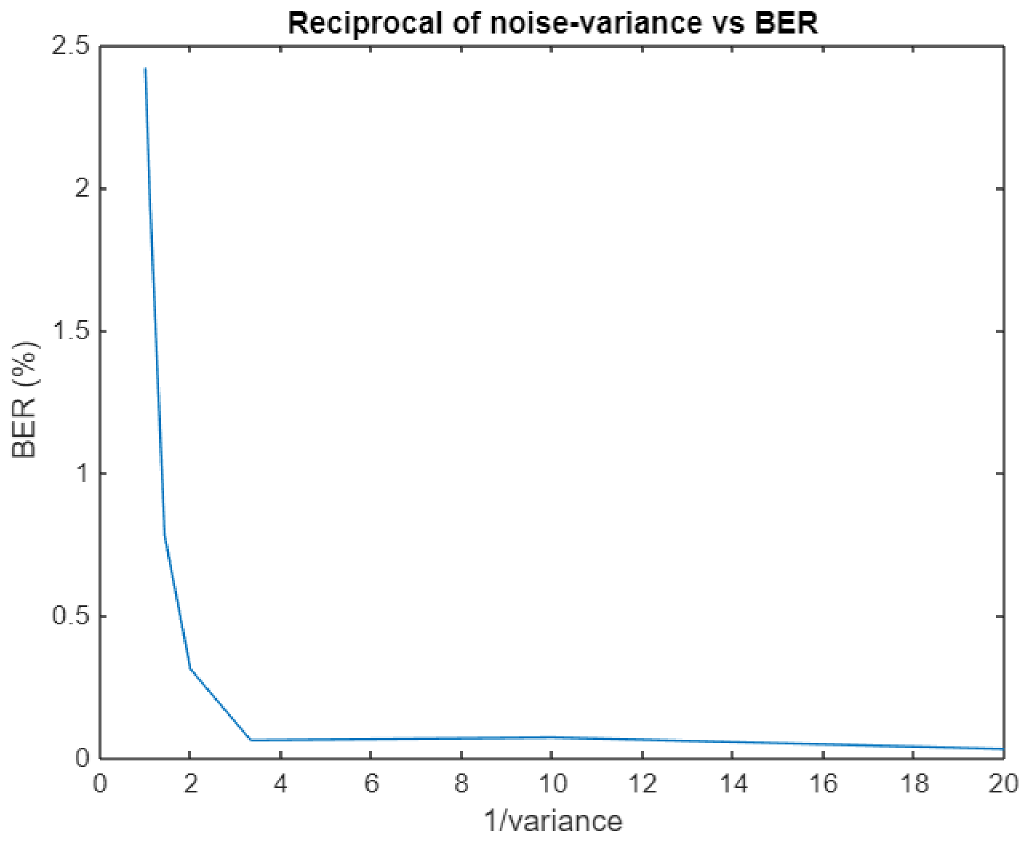
Inference: The bit estimate is generated by comparing the incoming signal with a threshold value (here, 0).

4. Plot the bit error rate (using the additive noise model as was implemented in the previous task) as a function of the reciprocal of the noise-variance. For making this plot, you have to consider different values of the noise variance. For each value of the noise variance, generate the bit error rate value for 1000 bits, 10 times. Take the average of the 10 bit error rates as the bit error rate for that noise variance. Repeat for all values of the noise variance. Consider noise-variance values of 0.05, 0.1, 0.3, 0.5, 0.7, 0.9, 1.

```
BER_matrix_filtered = zeros(1,7);
N = 1000;
k = 1; % loop iterator

for noise_var = [0.05,0.1,0.3,0.5,0.7,0.9,1]
    BER_sum = 0;
    for l = 1:10
        % bits and signal generation
        source = rand(1, N) > 0.5;
        T = 0.1; fs1 = 50; n = T*fs1;
        signal = zeros(1,n*N);
        for i=1:(length(source)-1)
            if i==1
               signal(1:n) = repelem(source(1),n);
            else
               signal((i-1)*n+1:(i)*n) = repelem(source(i),n);
            end
        end
        signal(signal==0) = -1;

        % channel and received signal
        fs = 160; fc = 20;
        channel = firpm(200, [0 fc fc+10, fs/2]/(fs/2), [1 1 0 0]);
        output_signal = conv(signal, channel, 'same');
        noise = sqrt(noise_var) * randn(1, length(output_signal));
        output_signal_noisy = output_signal + noise;

        % matched filter
        filter_matched = fliplr(ones(1,n));
        output_signal_filtered = conv(output_signal_noisy, filter_matched,
'same');

        % sampling receiver signal
        n=1:N;
        sampling_times = (n-0.5)*T;
        samples_filtered = output_signal_filtered(round(sampling_times*50));

        % bit estimation
        bit_estimate_filtered = zeros(1,N);
        for i = 1:length(samples_filtered)
            bit_estimate_filtered(i) = samples_filtered(i) > 0;
        end

        % BER calculation
        BER_sum = BER_sum + sum(bit_estimate_filtered ~= source)/
length(source)*100;
    end
    BER_matrix_filtered(k) = BER_sum/l;
    k = k+1;
end
disp(['BERs are: ', num2str(BER_matrix_filtered)]);
```

```
BERs are: 0.03        0.07        0.06        0.31        0.78        1.93        2.42
```

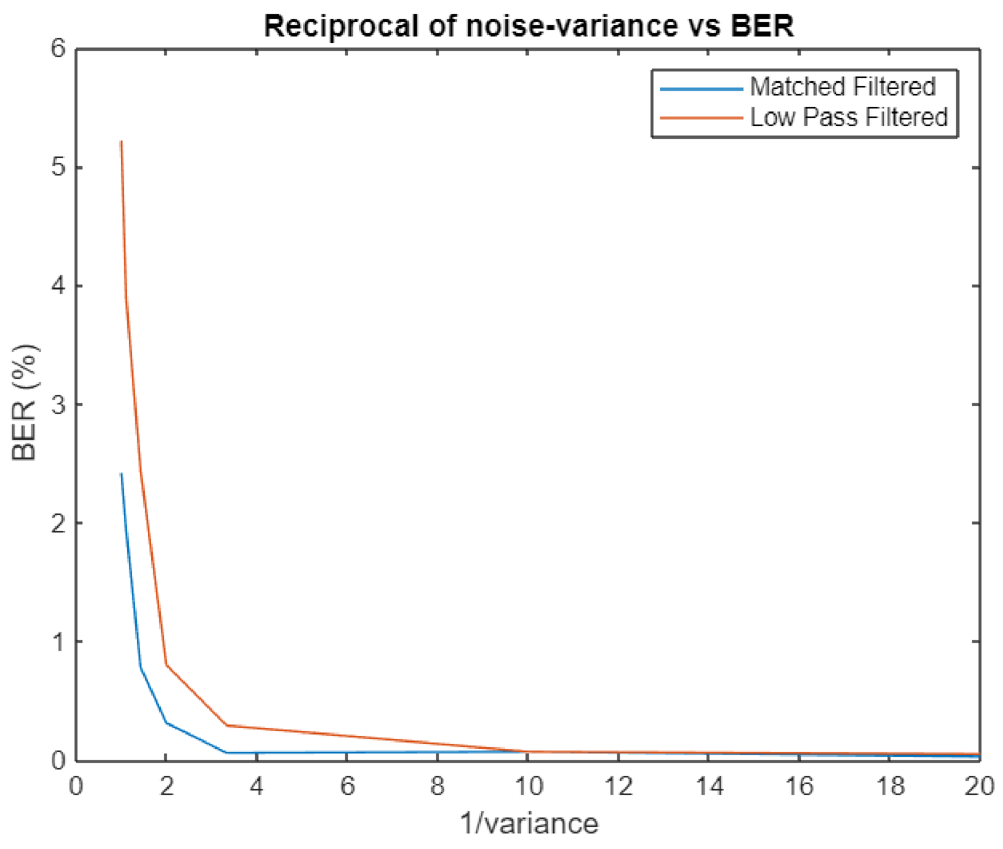
```
figure;
plot(1./[0.05,0.1,0.3,0.5,0.7,0.9,1], BER_matrix_filtered);
title("Reciprocal of noise-variance vs BER");
xlabel("1/variance");
ylabel("BER (%)");
```



Inference: Low BERs are observed.

5. On the same plot draw the BER curve that you had obtained in the previous task for the low pass receive filter. Compare and comment on the two BER curves

```
% Using low-pass filter
BER_matrix = zeros(1,7);
N=1000;
k = 1;

for noise_var = [0.05,0.1,0.3,0.5,0.7,0.9,1]
    BER_sum = 0;
    for l = 1:10
        % bits and signal generation
        source = rand(1, N) > 0.5;
        T = 0.1; fs1 = 50; n = T*fs1;
        signal = zeros(1,n*N);
        for i=1:(length(source)-1)
            if i==1
               signal(1:n) = repelem(source(1),n);
            else
               signal((i-1)*n+1:(i)*n) = repelem(source(i),n);
            end
        end
        signal(signal==0) = -1;

        % channel and received signal
        fs = 160; fc = 20;
        channel = firpm(200, [0 fc fc+10, fs/2]/(fs/2), [1 1 0 0]);
        output_signal = conv(signal, channel, 'same');
        noise = sqrt(noise_var) * randn(1, length(output_signal));
        output_signal_noisy = output_signal + noise;

        % low-pass filtering
        fs = 100;
        filter_receive = firpm(200, [0, 20, 25, fs/2]/(fs/2), [1, 1, 0, 0]);
        output_signal_filtered = conv(output_signal_noisy, filter_receive,
'same');

        % sampling receiver signal
        n=1:N;
        sampling_times = (n-0.5)*0.1;
        samples_noisy = output_signal_filtered(round(sampling_times*50));

        % bit estimation
        bit_estimate_noisy = zeros(1,N);
        for i = 1:length(samples_noisy)
            bit_estimate_noisy(i) = samples_noisy(i) > 0;
        end

        % BER calculation
        BER_sum = BER_sum + sum(bit_estimate_noisy ~= source)/
length(source)*100;
    end
    BER_matrix(k) = BER_sum/l;
    k = k+1;
end

hold on
plot(1./[0.05,0.1,0.3,0.5,0.7,0.9,1], BER_matrix); % for low-passed signal
legend('Matched Filtered', 'Low Pass Filtered');
```



Inference: The matched-filtered signal at the receiver shows comparatively less BER. Hence, match filtering improves the SNR.