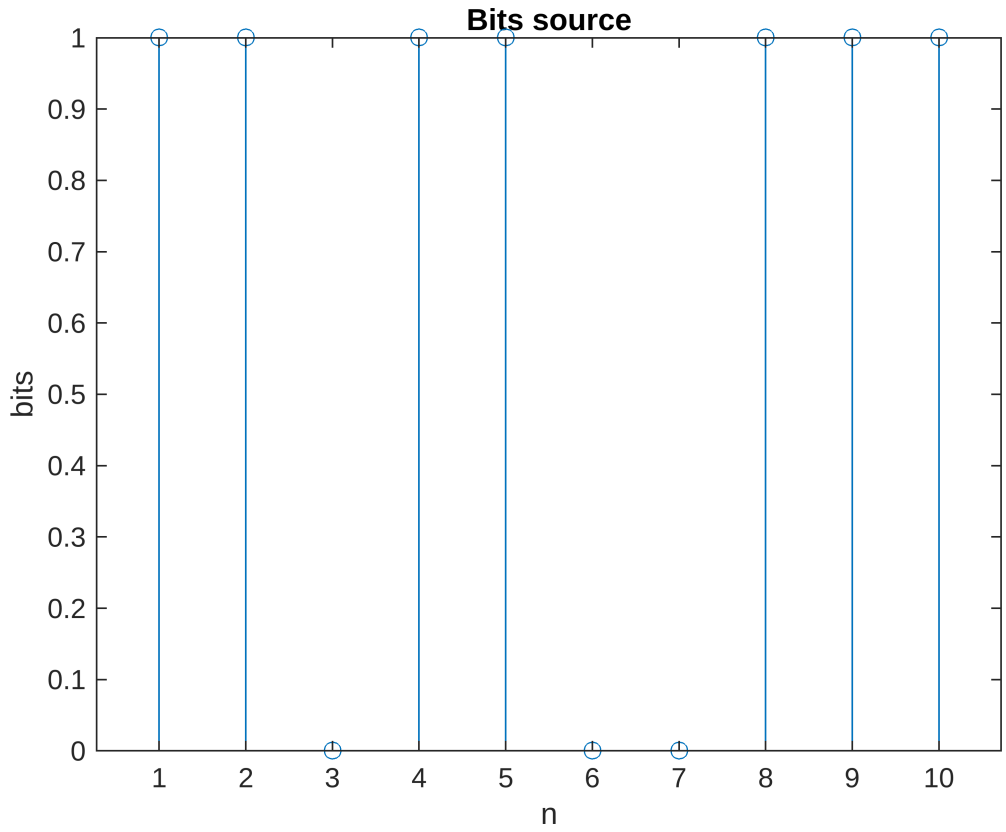
Communication System Lab 3

By: Saurabh Kumar (SC22B146)

3.1 Simulation of a Baseband Communication System

1. Generation of bits: The source is assumed to produce a stream of bits. The bits are usually modelled as an independent and identically distributed random process with the probability of a bit being 1 being 0.5. Generate a random sequence of bits of length N (input) satisfying this property.

```
rng("default");
N = 10;
source = rand(1, N) > 0.5;
figure;
stem(source);
title("Bits source");
xlabel("n");
ylabel("bits");
```

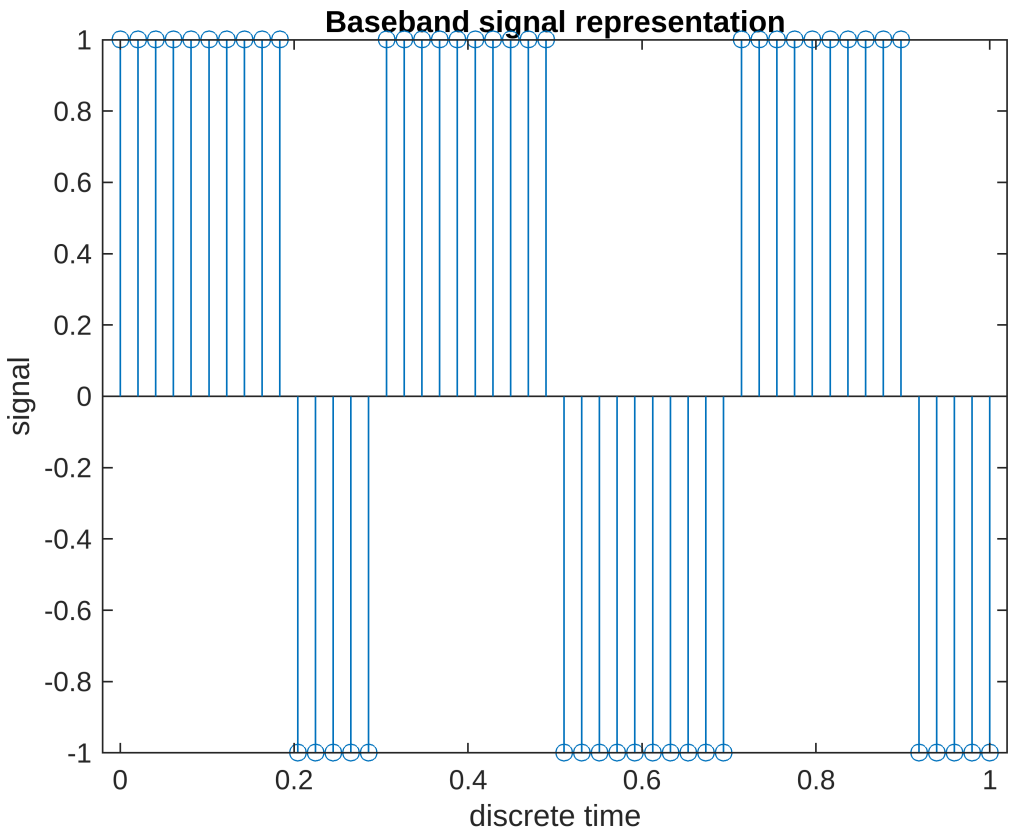


Inference: Random bits are generated and are compared with a threshold to generate the source bits.

2. The above bit sequence is then converted to a baseband signal. The baseband signal is obtained by converting 0 and 1 into appropriate pulses of duration T = 0.1s. For a bit sequence with N = 10, obtain a baseband signal with 0 represented using a level of −1 and 1 using a level of 1. Note that all continuous time

signals have to be represented by their sampled counterparts. Clearly state the sampling rate that you have used. Plot the sampled version of the continuous time signal that you have obtained.

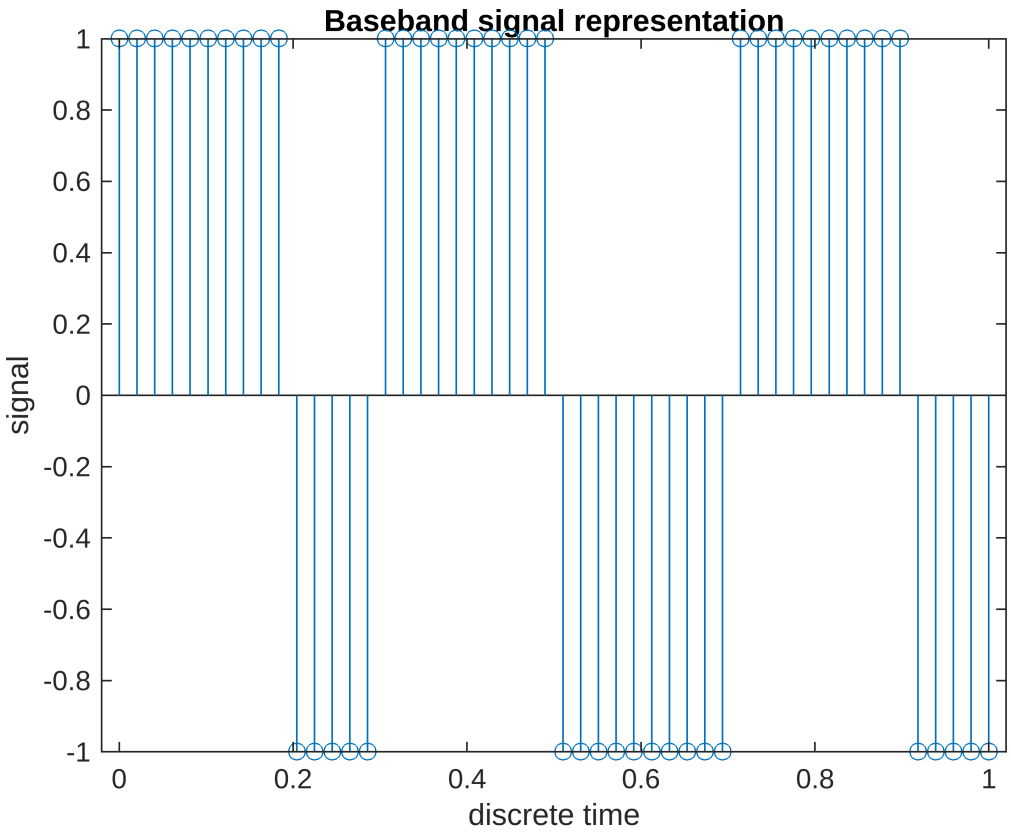
```
T = 0.1;
fs = 50;
n = T*fs;
signal = zeros(1,n*N);
for i=1:(length(source)-1)
    if i==1
       signal(1:n) = repelem(source(1),n);
    else
       signal((i-1)*n+1:(i)*n) = repelem(source(i),n);
    end
end

% Make '0' bits as '-1'
signal(signal==0) = -1;
figure;
t = linspace(0,N*T,N*fs*T);
stem(t,signal);
title("Baseband signal representation");
xlabel("discrete time");
ylabel("signal");
```



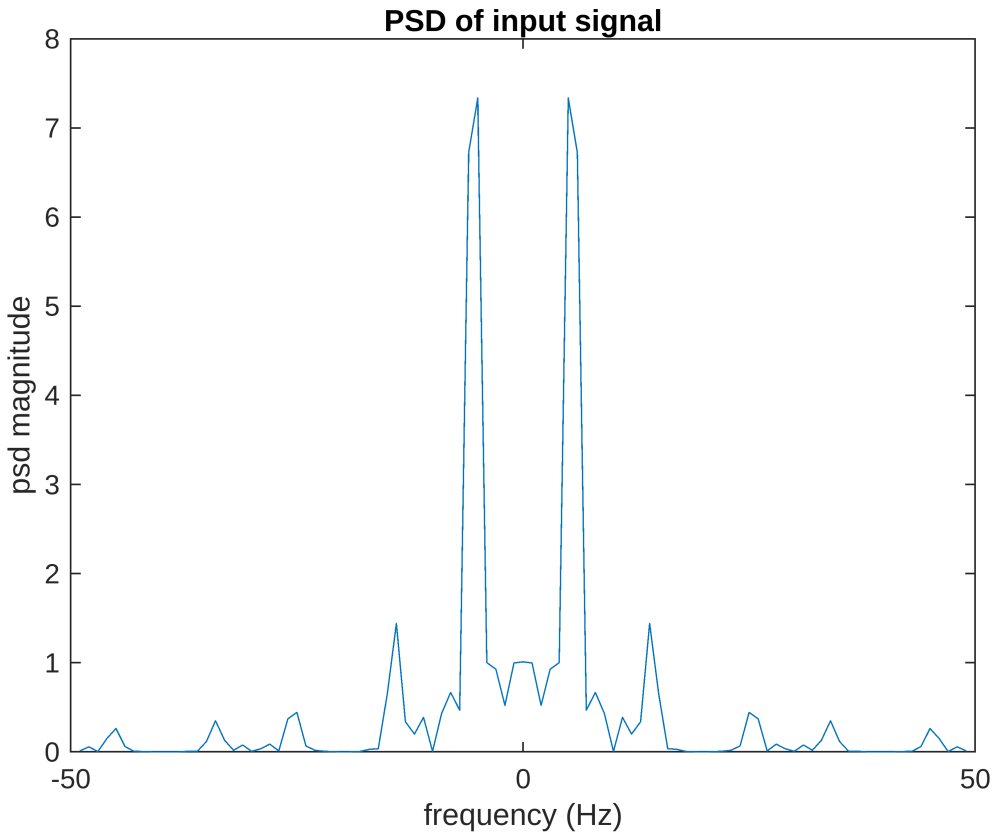
Inference: To get signal representation of the bit source, the bit source is repeated as specified by the sampling time, resembling a sampled signal.

3. Plot the power spectrum of the baseband signal for N = 1000. Note that the power spectrum should be plotted with respect to the continuous time frequency (Hint: carefully think about energy vs. power). (Opt: Along with this plot that you have obtained, plot (i.e. use hold on in Matlab) the PSD that we have obtained from theory in class - compare and contrast the measured spectrum and spectrum obtained from theory.)

```
rng("default");
N = 10;
source = rand(1, N) > 0.5;
T = 0.1;
fs = 50;
n = T*fs;
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;
figure;
t = linspace(0,N*T,N*fs*T);
stem(t,signal);
title("Baseband signal representation");
xlabel("discrete time");
ylabel("signal");
```



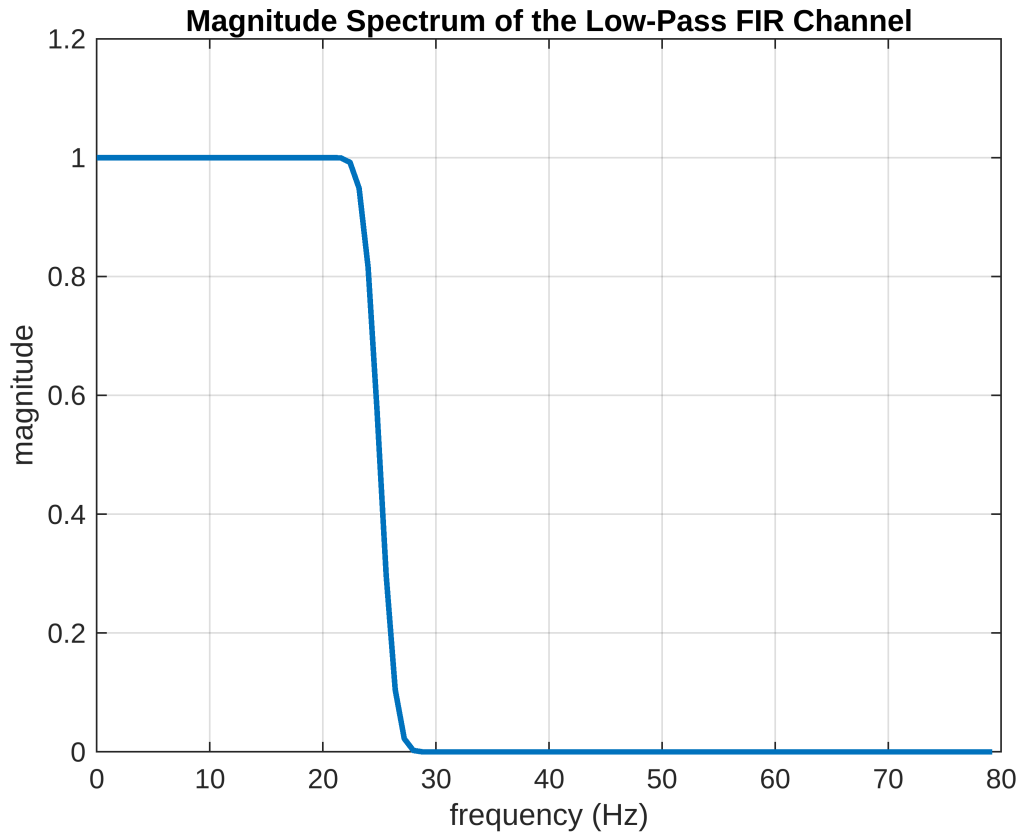

```
[auto_corr,lags] = xcorr(signal);
psd = fftshift(fft(auto_corr))/length(auto_corr);
figure;
plot(lags,abs(psd));
title("PSD of input signal");
xlabel("frequency (Hz)");
ylabel("psd magnitude");
```



Inference: PSD is calculated by taking autocorrelation and then fft of the signal.

4. For modelling a baseband channel use any FIR filter design technique to obtain a filter response that corresponds to a channel with passband edge at 20Hz and a gain of g in the passband. Plot the magnitude spectrum of the channel that you are using - clearly labelling the axes for g = 1.

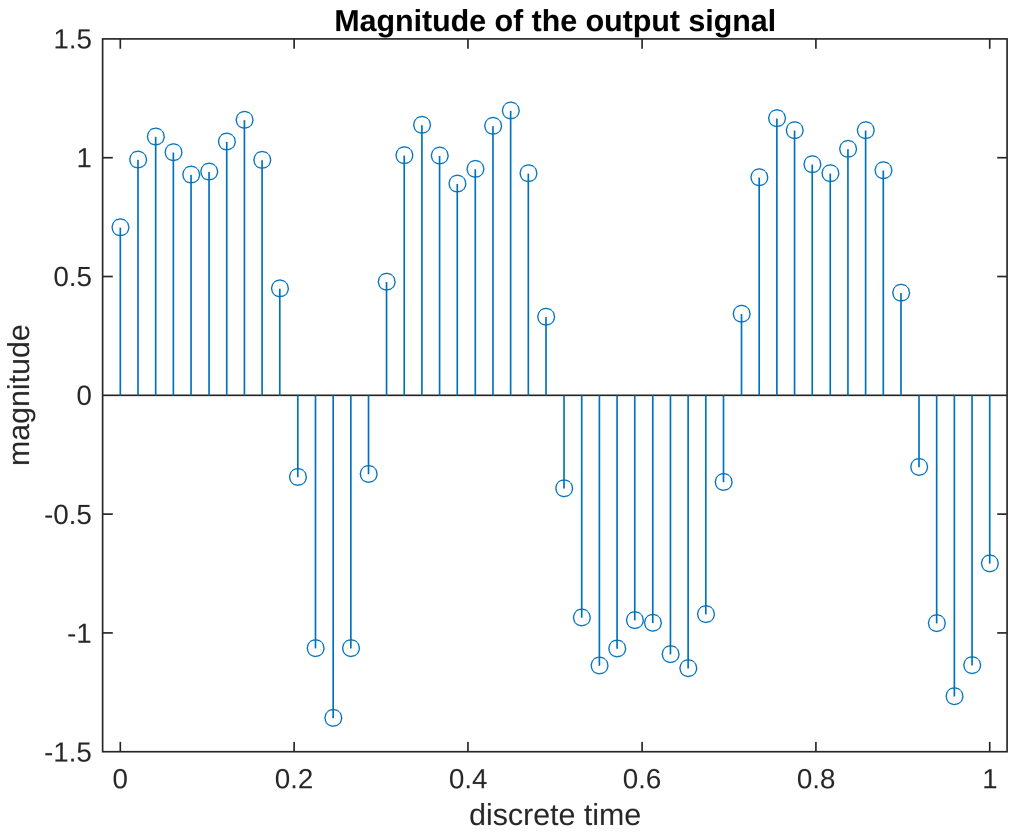
```
fs = 160;
fc = 20;
g = 1;
% low-pass FIR filter
channel = firpm(200, [0 fc fc+10, fs/2]/(fs/2), [g g 0 0]);
[H, f] = freqz(channel, 1, 100, fs);

% Plot the response
figure;
plot(f, abs(H), 'LineWidth', 2);
title('Magnitude Spectrum of the Low-Pass FIR Channel');
grid on;
xlabel('frequency (Hz)');
ylabel('magnitude');
```



Inference: Low-Pass filter is obtained using 'firpm' function as specified by gain, sampling frequency and cutoff frequencies..

5. Simulate sending the baseband signal through the above channel. Plot the output signal from the channel.

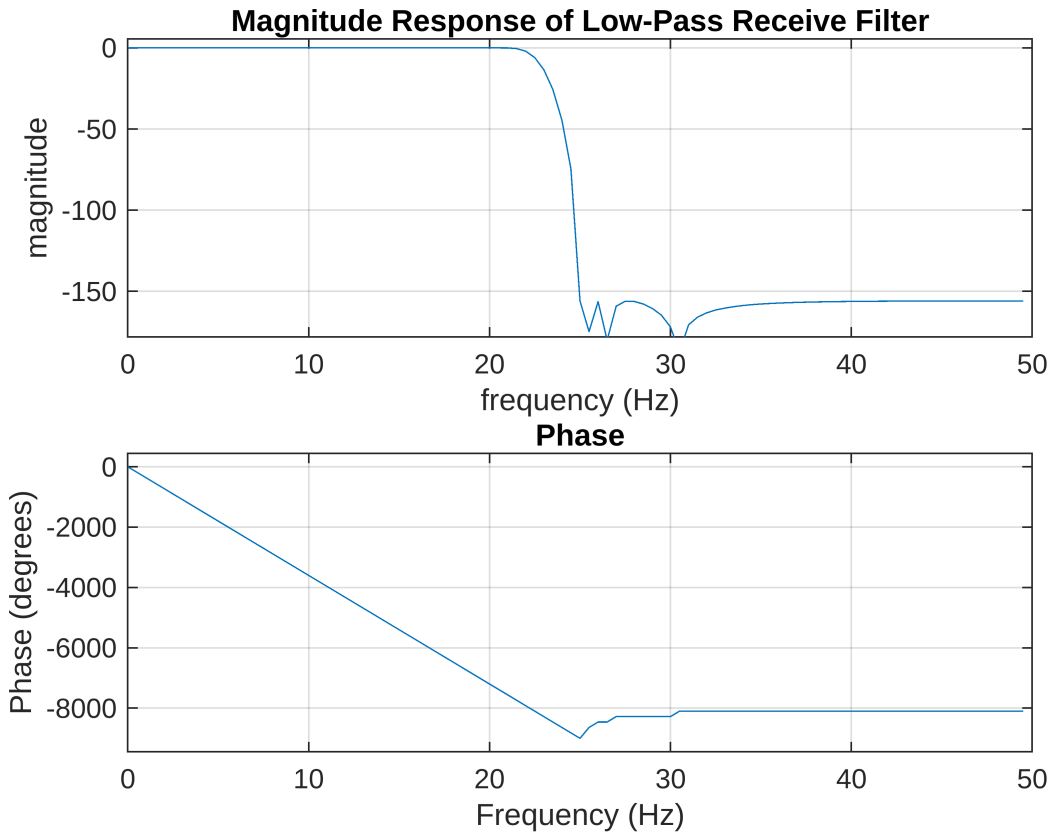
```
output_signal = conv(signal, channel, 'same');

figure;
t = linspace(0,1,50);
stem(t,output_signal);
title('Magnitude of the output signal');
xlabel('discrete time');
ylabel('magnitude');
```



Inference: Signal is passed through the channel, and the received signal is obtained using convolution of the signal and the channel.

6. At the output of the channel, implement a low pass receive filter. Plot the magnitude response of the filter that you have implemented. What are the specifications of the passband and stopband for this filter? Why did you choose those specifications?

```
fs = 100;
passband_edge = 20;
stopband_edge = 25;

filter_receive = firpm(200, [0, passband_edge, stopband_edge, fs/2]/(fs/2),
[1, 1, 0, 0]);

figure;
freqz(filter_receive, 1, 100, fs);
title('Magnitude Response of Low-Pass Receive Filter');
xlabel('frequency (Hz)');
ylabel('magnitude');
```

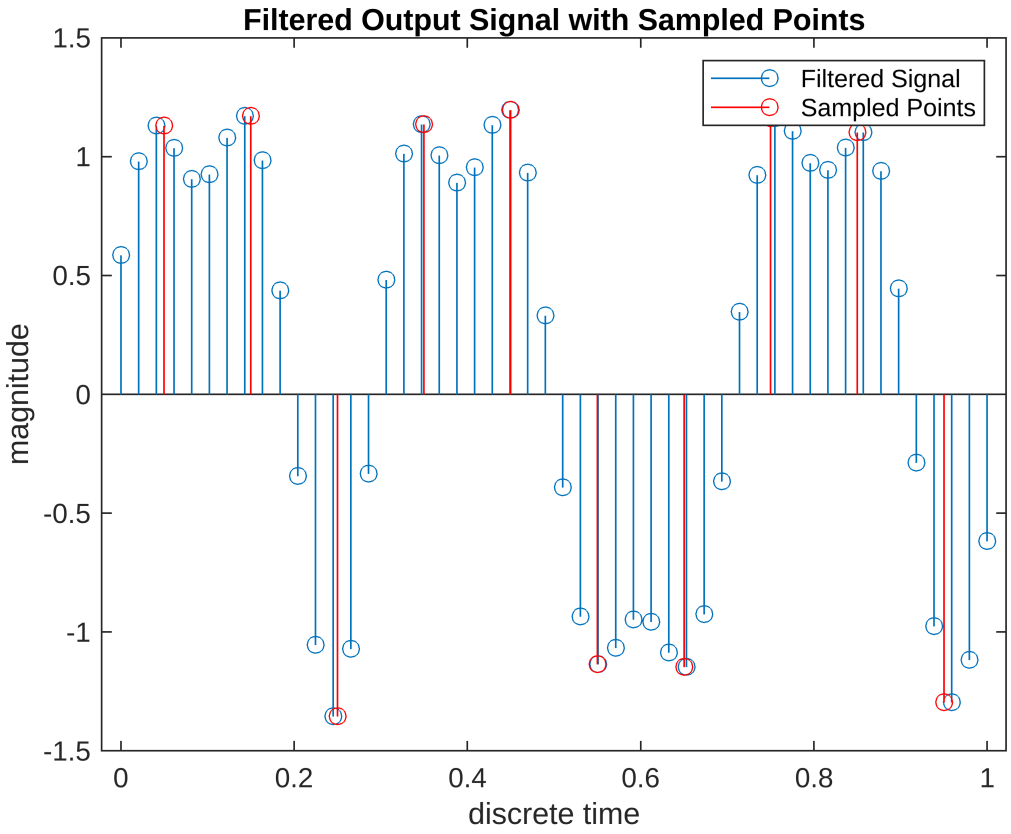


Inference: The passband edge at 20 Hz ensures that the majority of the baseband signal (which lies below 20 Hz) passes through without significant attenuation. The filter starts attenuating frequencies above 25 Hz. This is chosen to reject any high-frequency noise or components that were introduced by the channel, while still ensuring that the filter does not significantly affect the baseband signal.

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

```
output_signal_filtered = conv(output_signal, filter_receive,'same');

figure;
t = linspace(0,1,50);
stem(t,output_signal_filtered);
title('Time response of the filtered output signal');
xlabel('discrete time');
ylabel('magnitude');

N=10;
n=1:N;
sampling_times = (n-0.5)*0.1;
samples = output_signal_filtered(round(sampling_times*50));

hold on;
stem(sampling_times, samples, 'r');
```

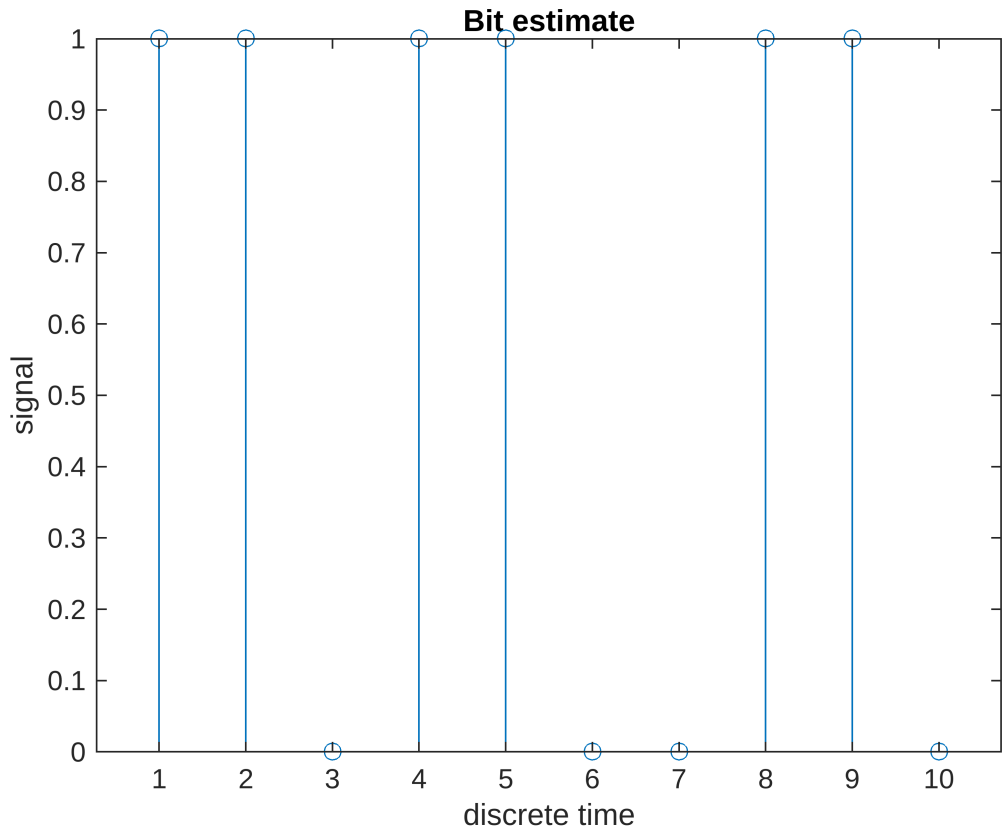
```
legend('Filtered Signal', 'Sampled Points');
title('Filtered Output Signal with Sampled Points');
```



Inference: The received signal is passed through the Low-Pass filter, and the output signal is sampled at the mid-Ts.

8. 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 decision device
threshold = 0;

bit_estimate = zeros(1,N);
for i = 1:length(samples)
    bit_estimate(i) = samples(i) > threshold;
end

figure;
stem(bit_estimate);
title("Bit estimate");
xlabel("discrete time");
ylabel("signal");
```



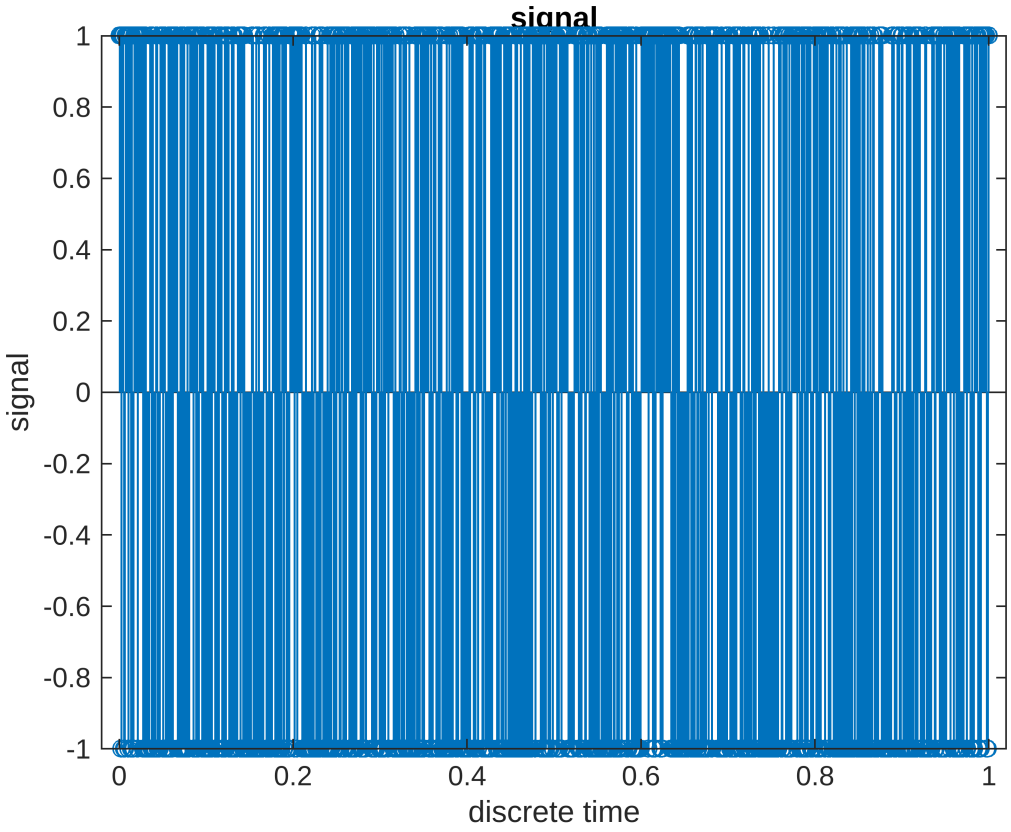
Inference: The sampled signal is compared with a threshold values (here, 0), to decide the received bits.

3.2 Bit error rate

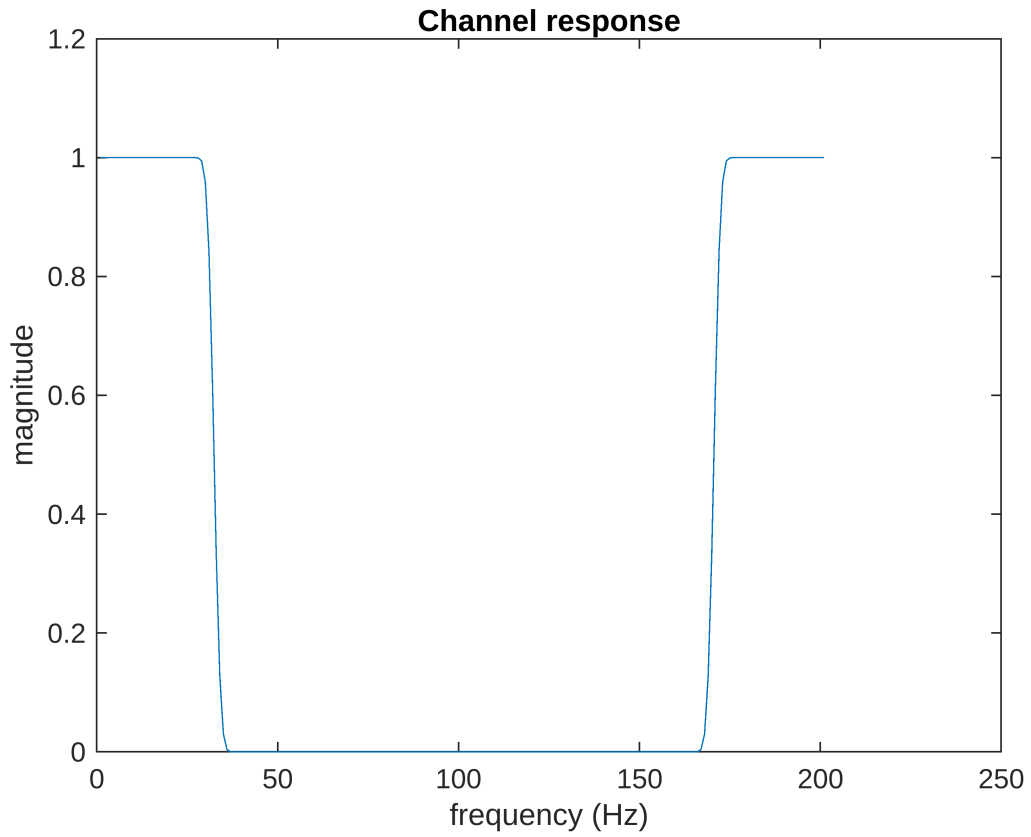
1. Now implement the additive noise model for the channel so that the channel introduces additive white Gaussian noise with variance σ^2 . For this set of tasks use σ^2 = 0.1.

```
% signal and source generation
rng("default");
N = 1000;
source = rand(1, N) > 0.5;
T = 0.1;
fs = 50;
n = T*fs;
signal = zeros(1,N*T*fs);
for i=1:(length(source))
    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;

figure;
```

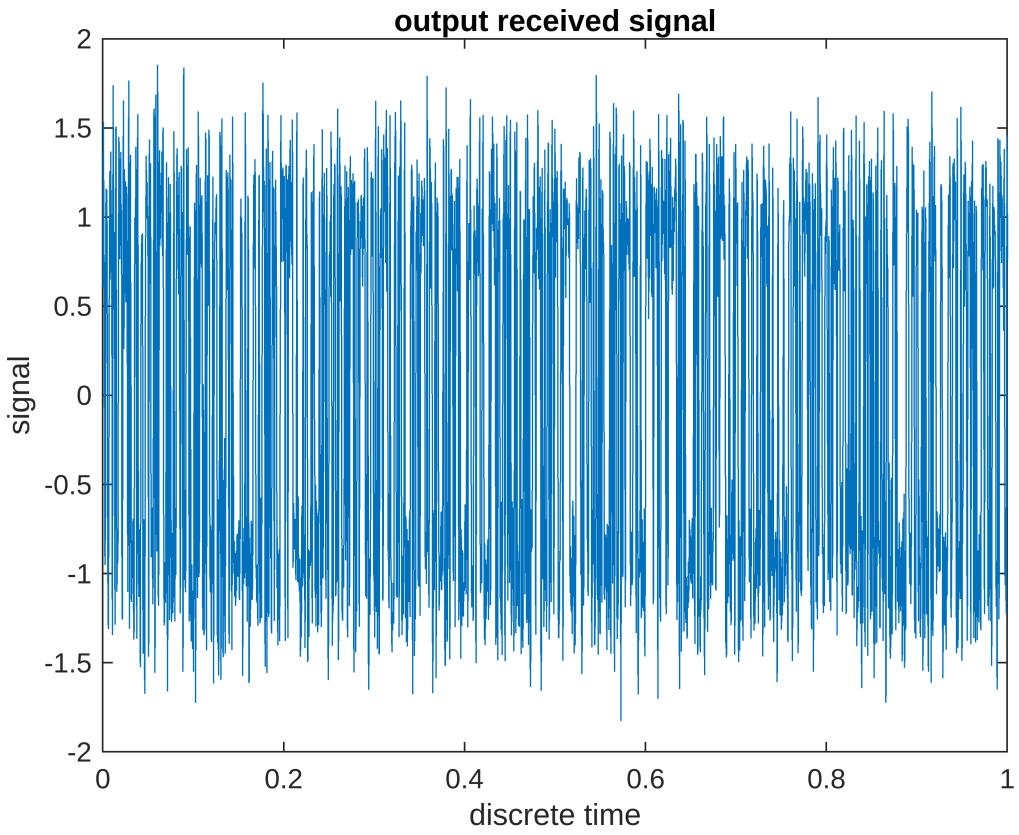
```
t = linspace(0,1,N*T*fs);
stem(t,signal);
title("signal");
xlabel("discrete time");
ylabel("signal");
```



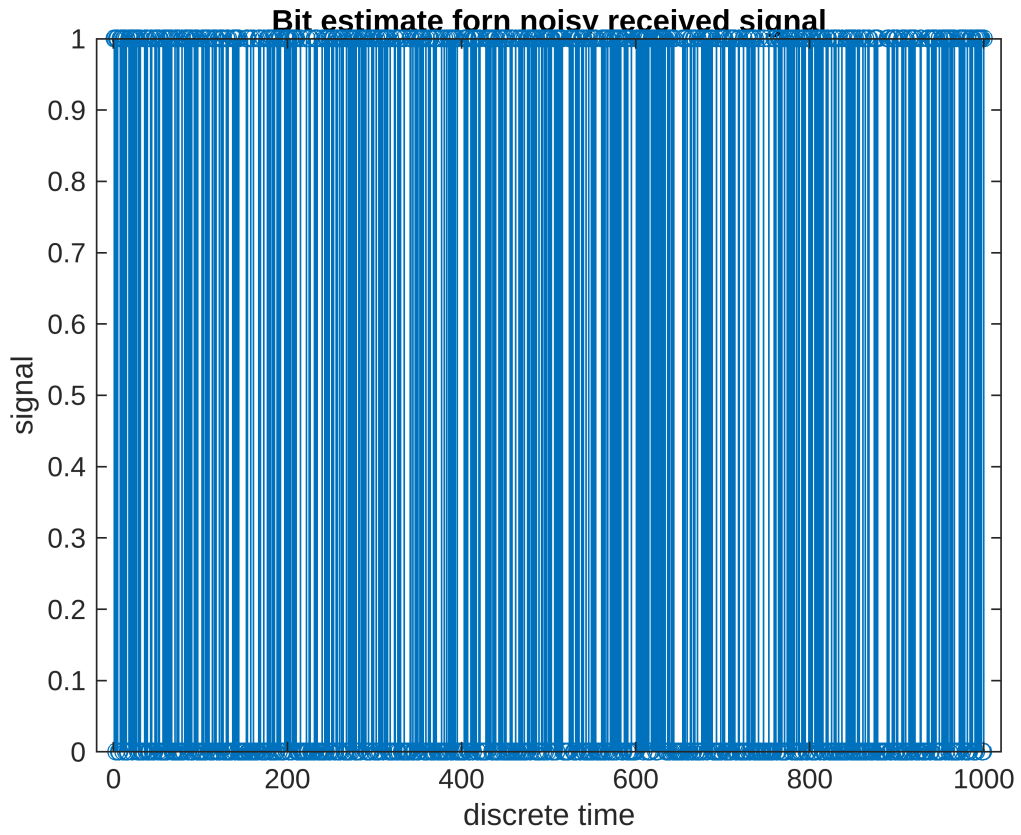

```
% low-pass channel
fs = 160; fc = 20; g = 1;
channel = firpm(200, [0 fc fc+10, fs/2]/(fs/2), [g g 0 0]);
figure;
plot(abs(fft(channel)));
title("Channel response");
xlabel("frequency (Hz)");
ylabel("magnitude");
```




```
% received signal and noise
output_signal = conv(signal, channel, 'same');

noise_var = 0.1;
noise_stddev = sqrt(noise_var);

noise = noise_stddev * randn(1,length(output_signal));

output_signal_noisy = output_signal + noise;

% low-pass filtering
fs = 100;
passband_edge = 20;
stopband_edge = 25;
filter_receive = firpm(200, [0, passband_edge, stopband_edge, fs/2]/(fs/2),
[1, 1, 0, 0]);

output_signal_filtered = conv(output_signal_noisy, filter_receive, 'same');

figure;
t = linspace(0,1,N*0.1*50);
plot(t, output_signal_filtered);
title("output received signal");
xlabel("discrete time");
ylabel("signal");
```




```
% sampling output noisy signal
n=1:N;
sampling_times = (n-0.5)*0.1;
samples_noisy = output_signal_noisy(round(sampling_times*50));

% thresholding sampled signal
threshold = 0;
bit_estimate_noisy = zeros(1,N);
for i = 1:length(samples_noisy)
    bit_estimate_noisy(i) = samples_noisy(i) > threshold;
end

figure;
stem(bit_estimate_noisy);
title("Bit estimate forn noisy received signal");
xlabel("discrete time");
ylabel("signal");
```



Inference: Here, the signal is simulated is to pass through the noise with an added noise and is received at the reciever.

2. By comparing the input bits and the bits at the output of the decision block, obtain the BER - note that the BER is a function of the noise variance.

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

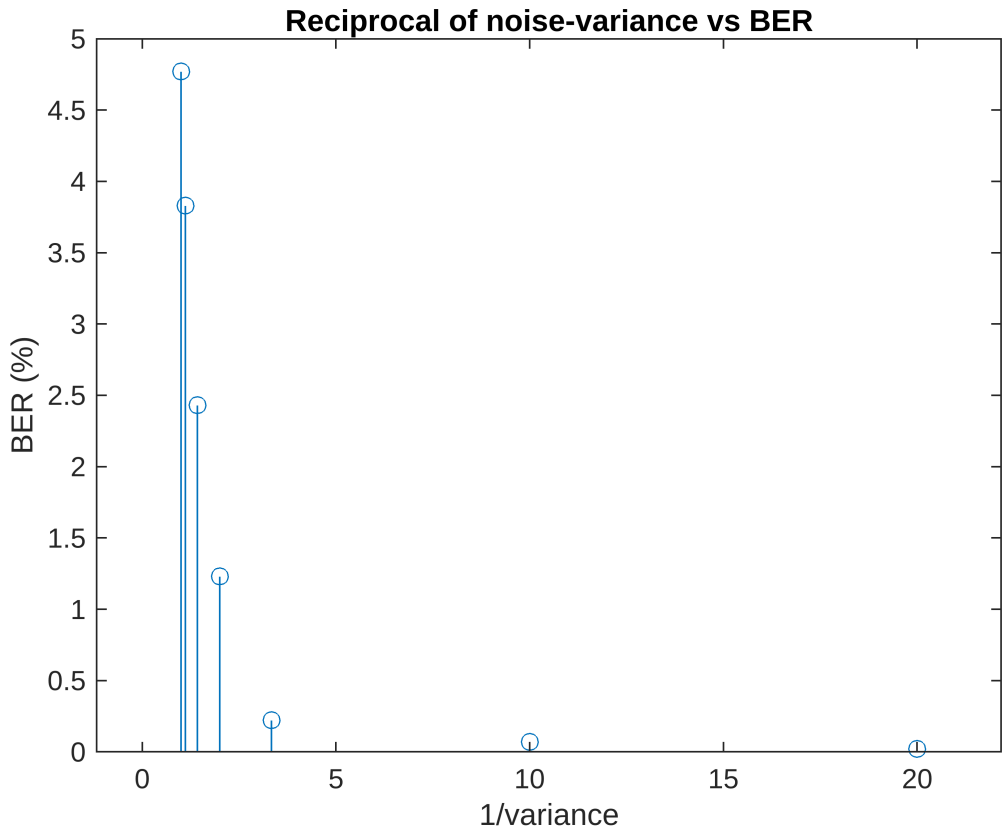
```
BER is 0%.
```

3. Plot the bit error rate as a function of the reciprocal of 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 = 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

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

```
ylabel("BER (%)");
```



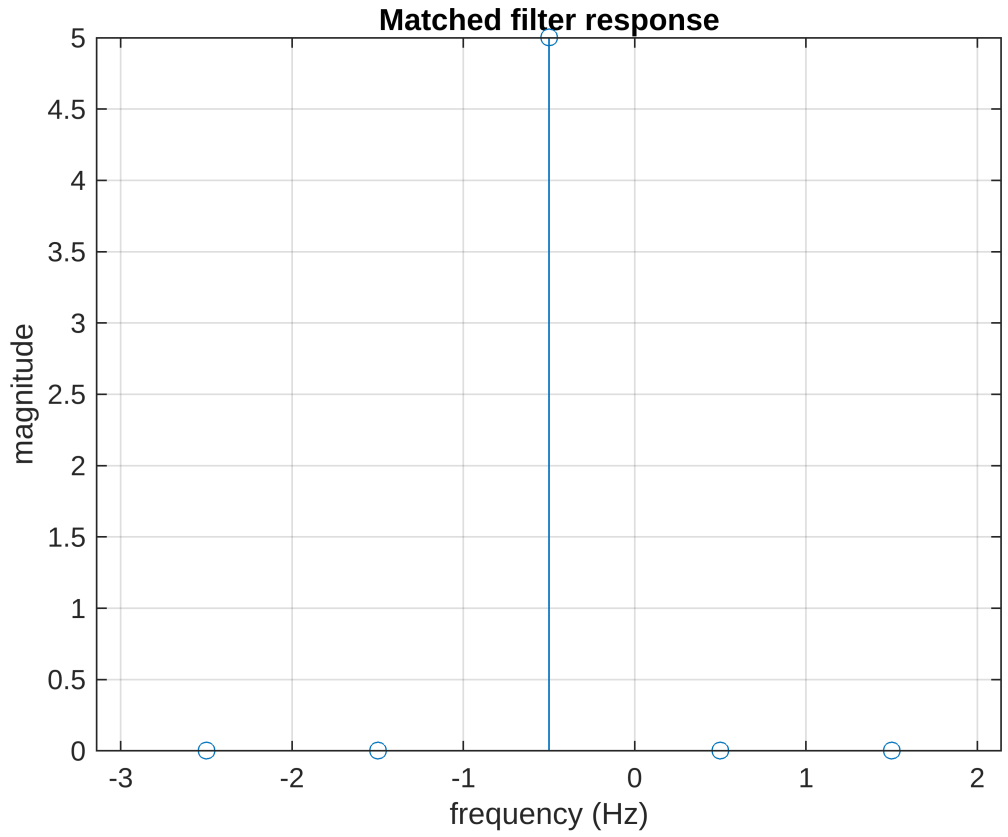
Inference: The BER vs. 1/sigma^2 plot shows the decreaseing exponential curve for the signal. This means, as the BER increases as the noise-variance increases.

3.3 Matched filtering

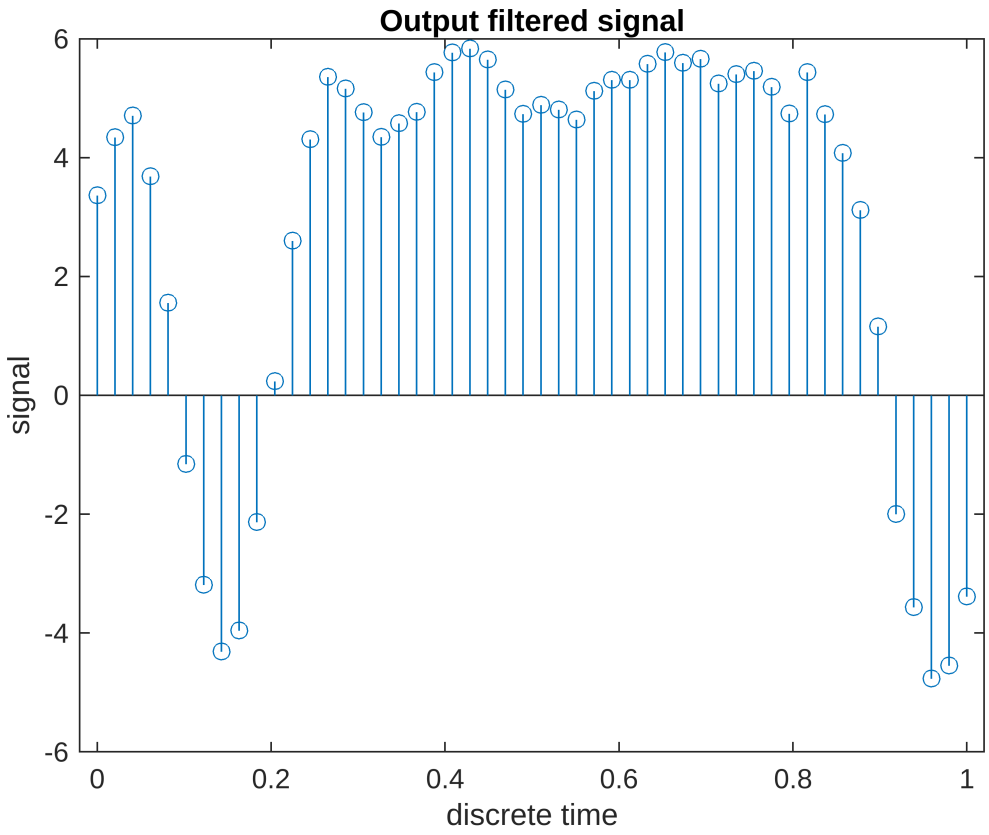
1. At the output of the ideal channel, implement the matched filter for the pulse shape that you have used. Plot the magnitude response of the filter that you have implemented.

```
% 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;
```



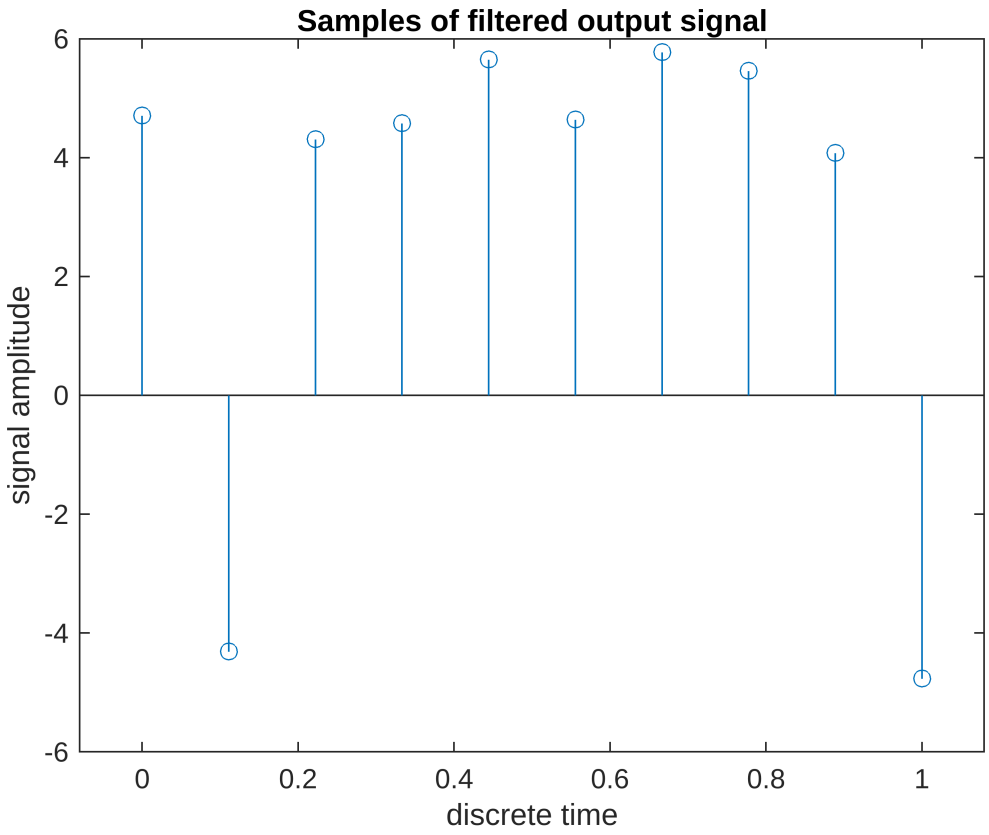

```
% filtered signal
output_signal_filtered = conv(output_signal_noisy, filter_matched, 'same');
figure;
t = linspace(0,1,N*0.1*50);
stem(t,output_signal_filtered);
title("Output filtered signal");
xlabel("discrete time");
ylabel("signal");
```



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.

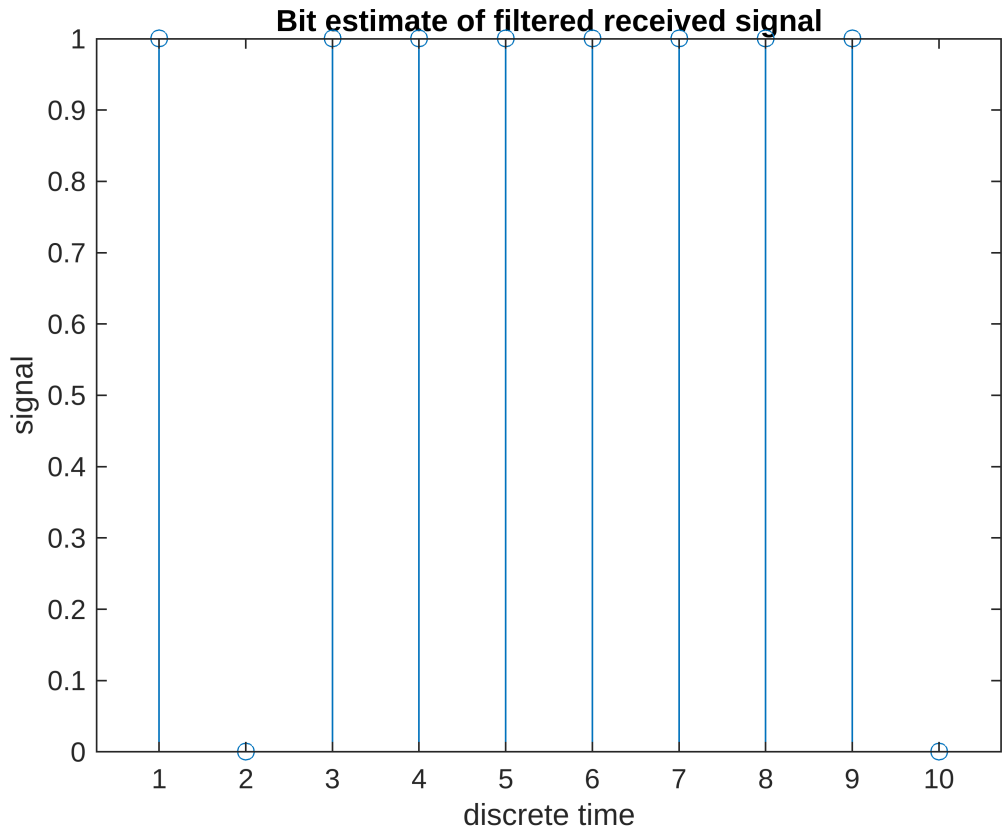
```
n=1:N;
sampling_times = (n-0.5)*T;
samples_filtered = output_signal_filtered(round(sampling_times*50));
figure;
t = linspace(0,1,N);
stem(t,samples_filtered);
title("Samples of filtered output signal");
xlabel("discrete time");
ylabel("signal amplitude");
```



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).

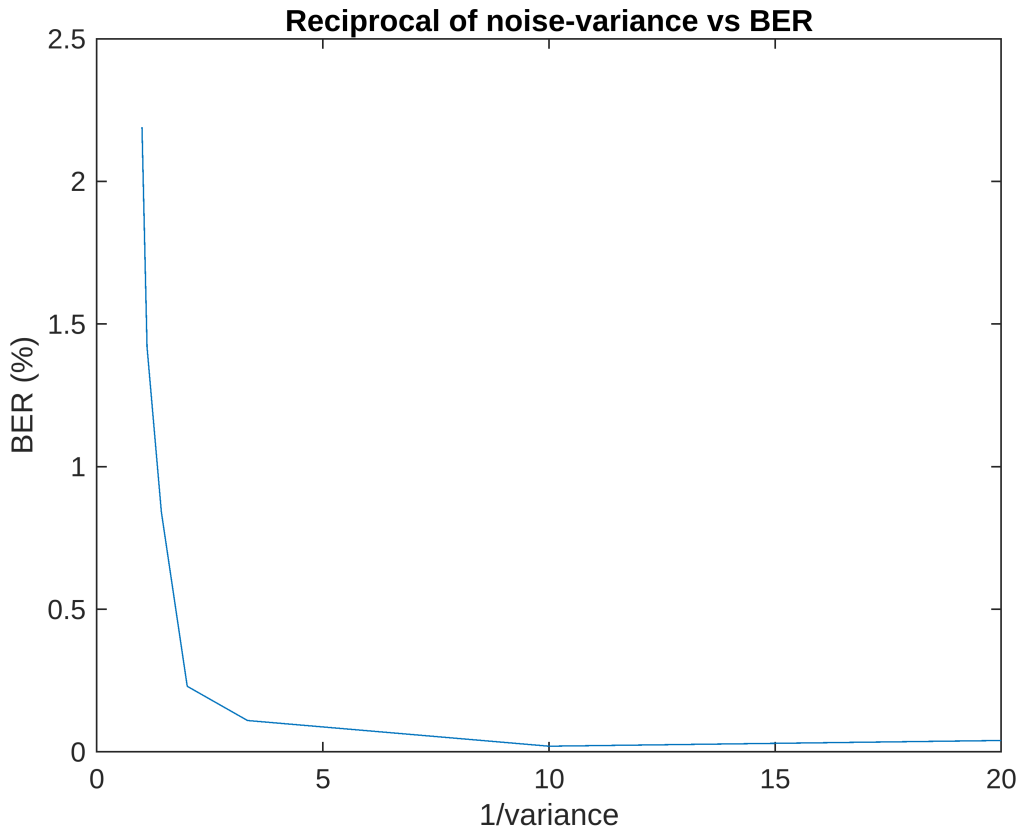
4. Plot the bit error rate (using the additive noise model as was implemented in the previous task) as a function of the reciprocal of the noise-variance. For making this plot, you have to consider different values of the noise variance. For each value of the noise variance, generate the bit error rate 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: 0.04       0.02       0.11       0.23       0.84       1.42       2.19
```

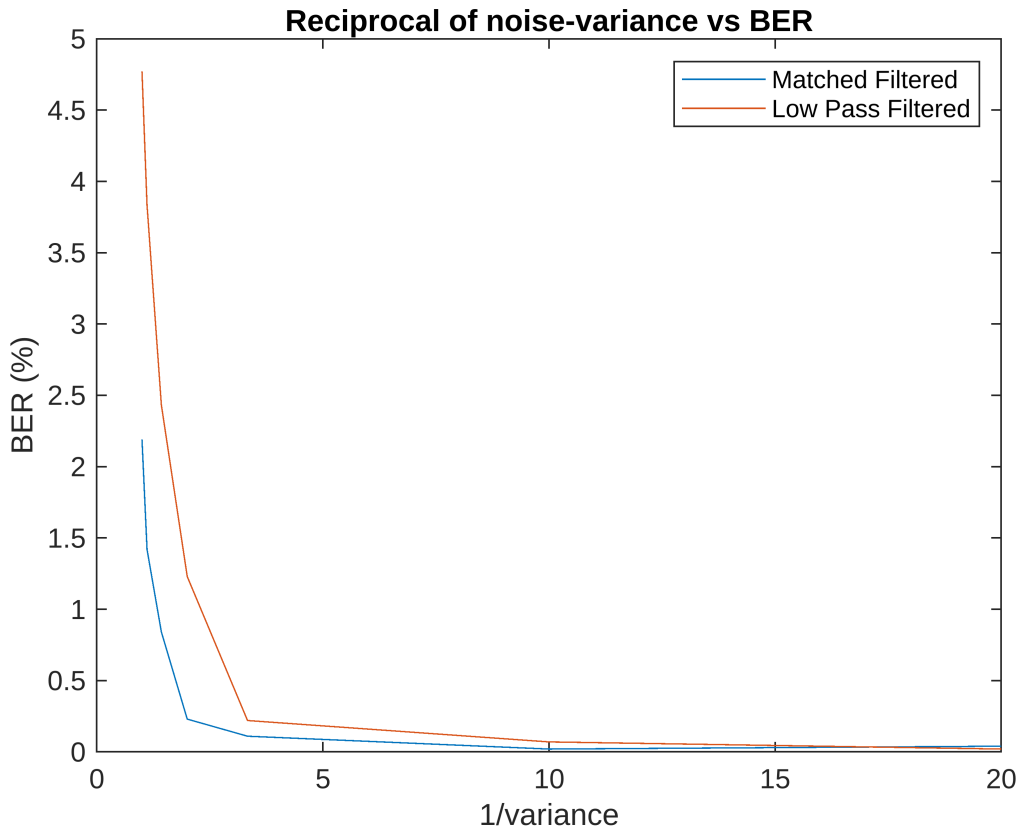
```
figure;
plot(1./[0.05,0.1,0.3,0.5,0.7,0.9,1], BER_matrix_filtered);
title("Reciprocal of noise-variance vs BER");
xlabel("1/variance");
ylabel("BER (%)");
```



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

```
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.