**Department of Computer Engineering**



**Cairo University**

**Faculty of Engineering**

**ELC 325B – Spring 2023**

**Digital Communications**

**Assignment #2**

**Filters**

**Submitted to**

Dr. Mai

Dr.Hala

Eng.Mohamed Khaled

**Submitted by**

|  |  |  |
| --- | --- | --- |
| **Name** | **Sec** | **BN** |
| **Sara Bisheer Fekry** | **1** | **18** |
| **Menna Mohamed Abdelbaset** | **2** | **26** |

Contents

[Part I: Solve the following question: 3](#_Toc16170)

[Hand Analysis 3](#_Toc2436)

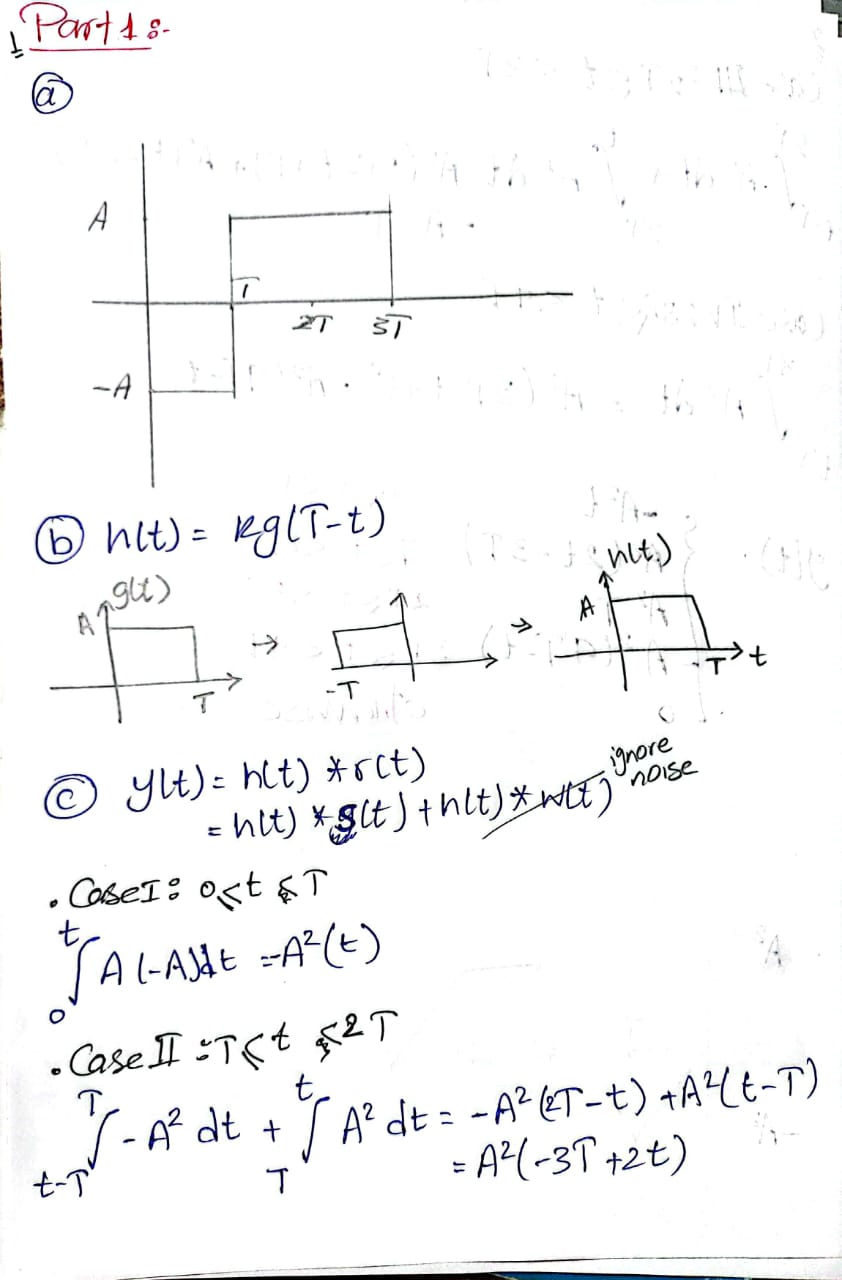
[Part II: Simulation: 6](#_Toc14945)

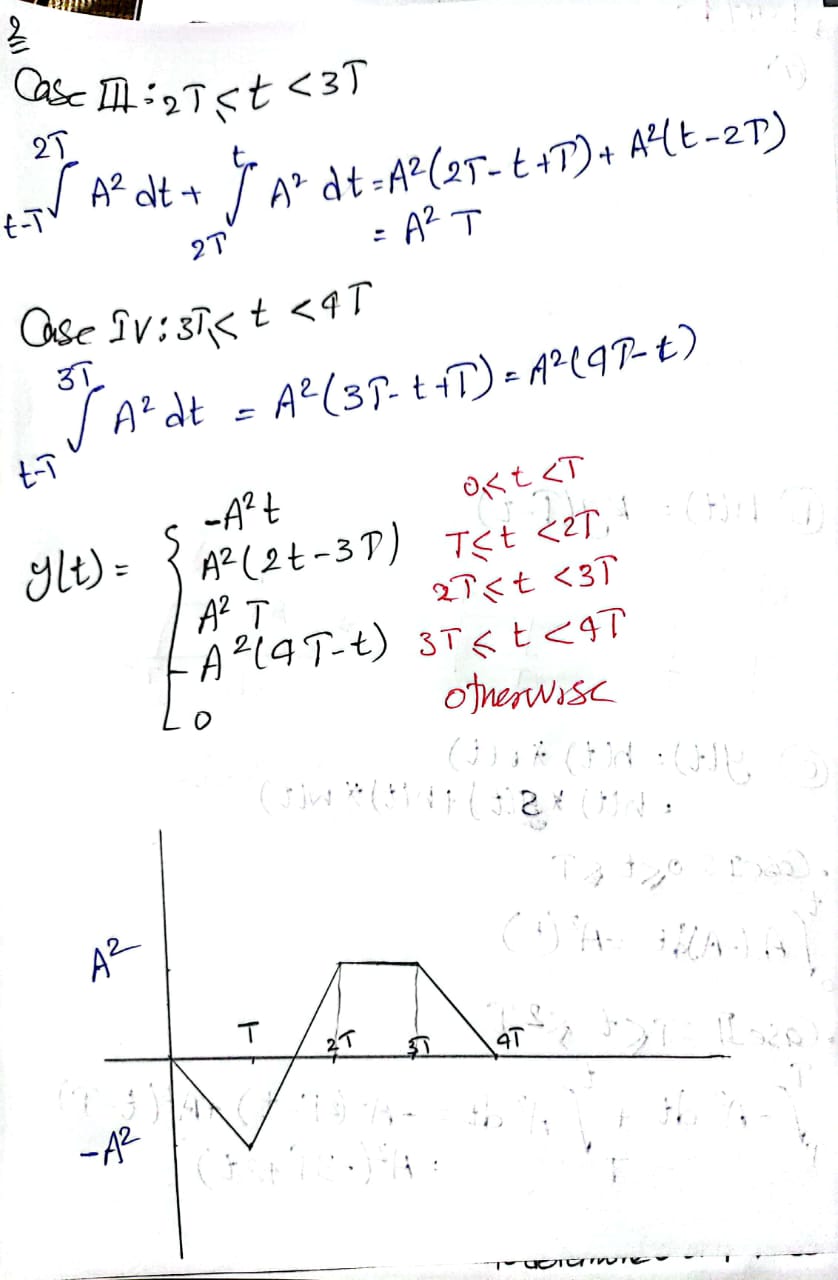
[Hand Analysis 6](#_Toc28281)

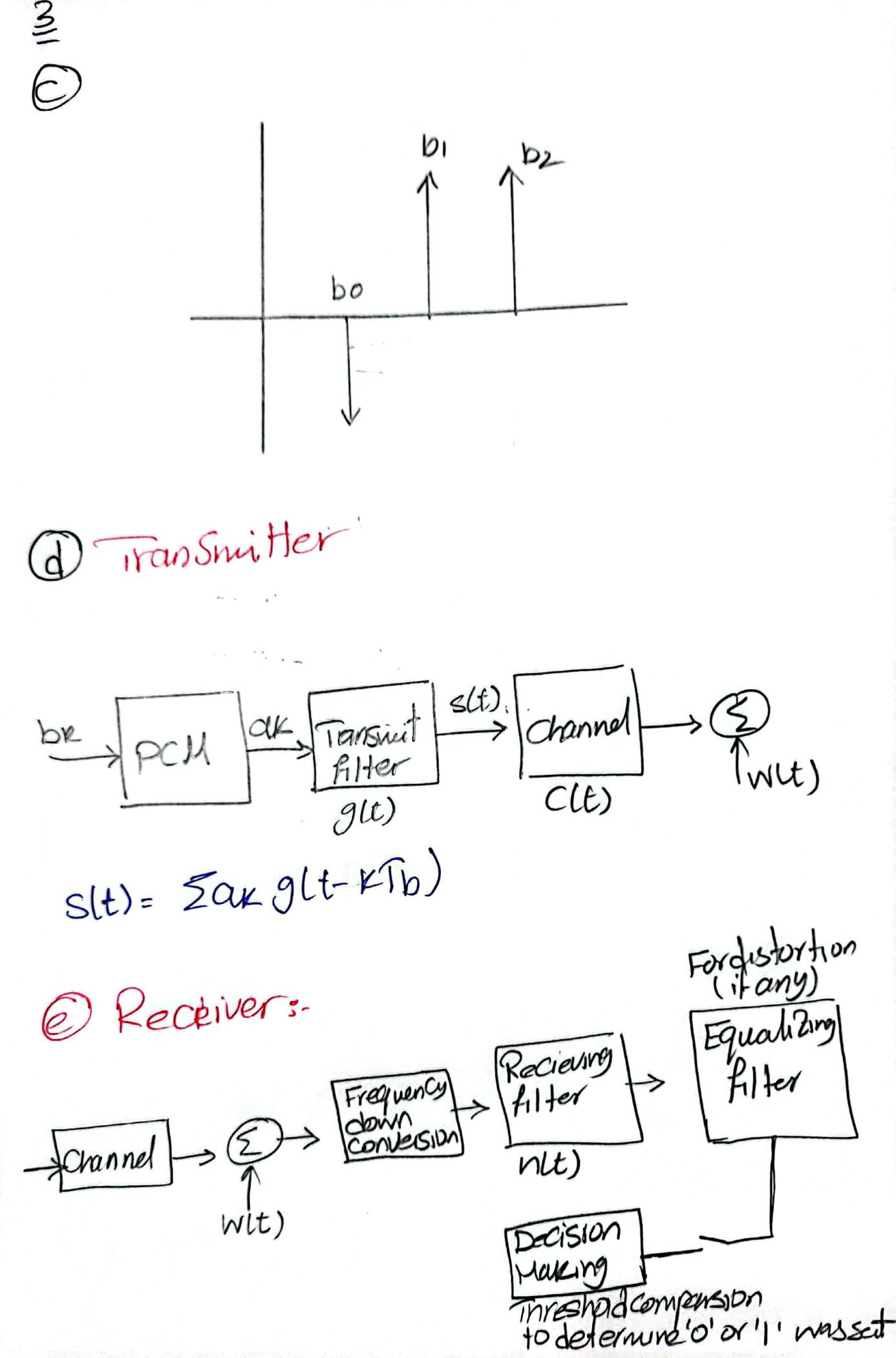
[Figures With Comment 12](#_Toc23251)

# **Part I: Solve the following question:**

# **Hand Analysis**

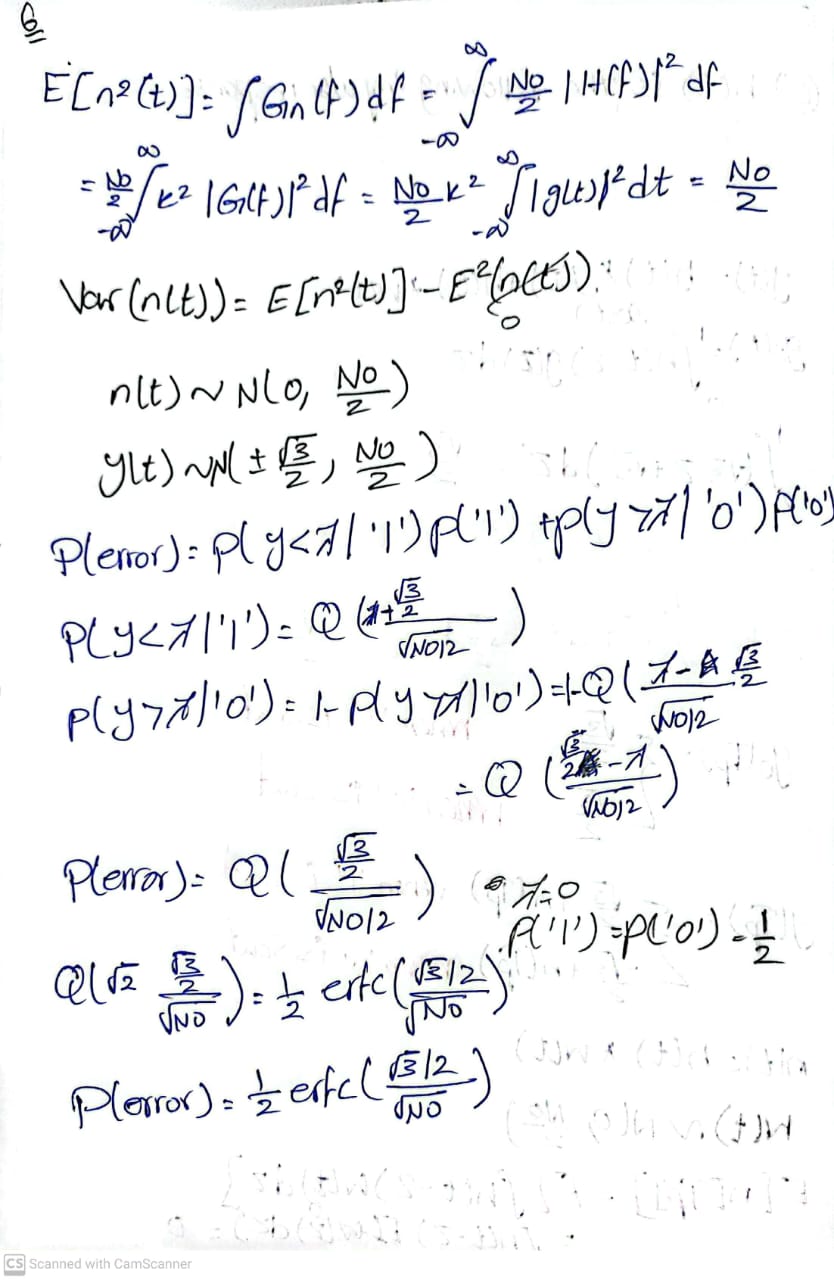
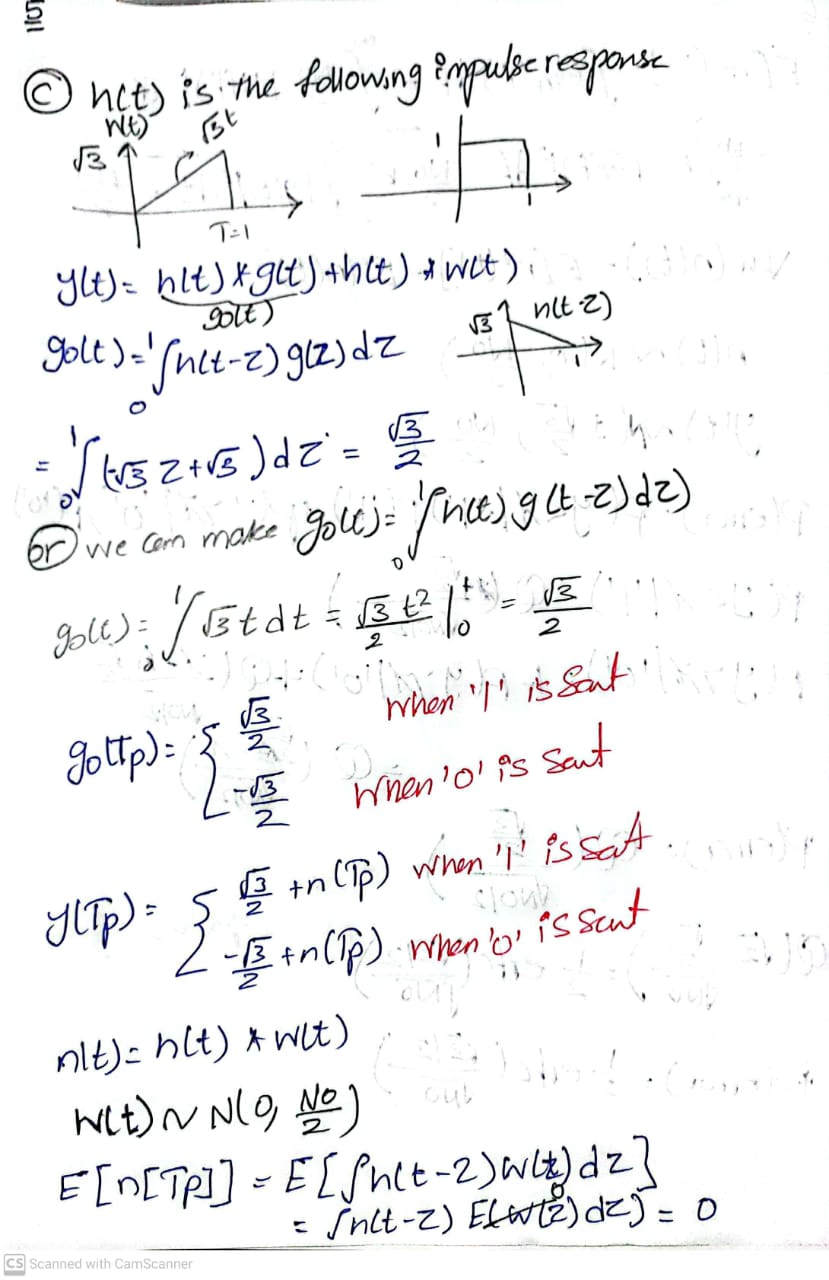
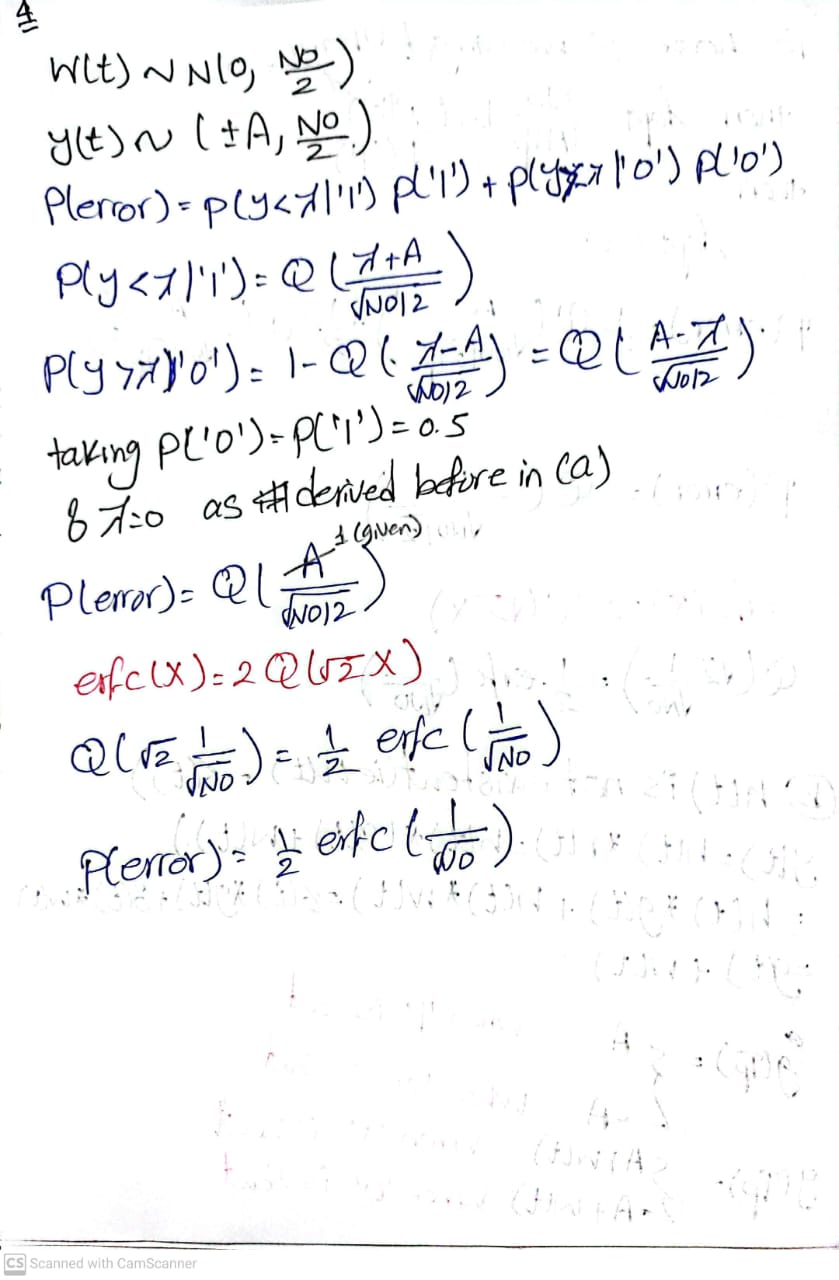
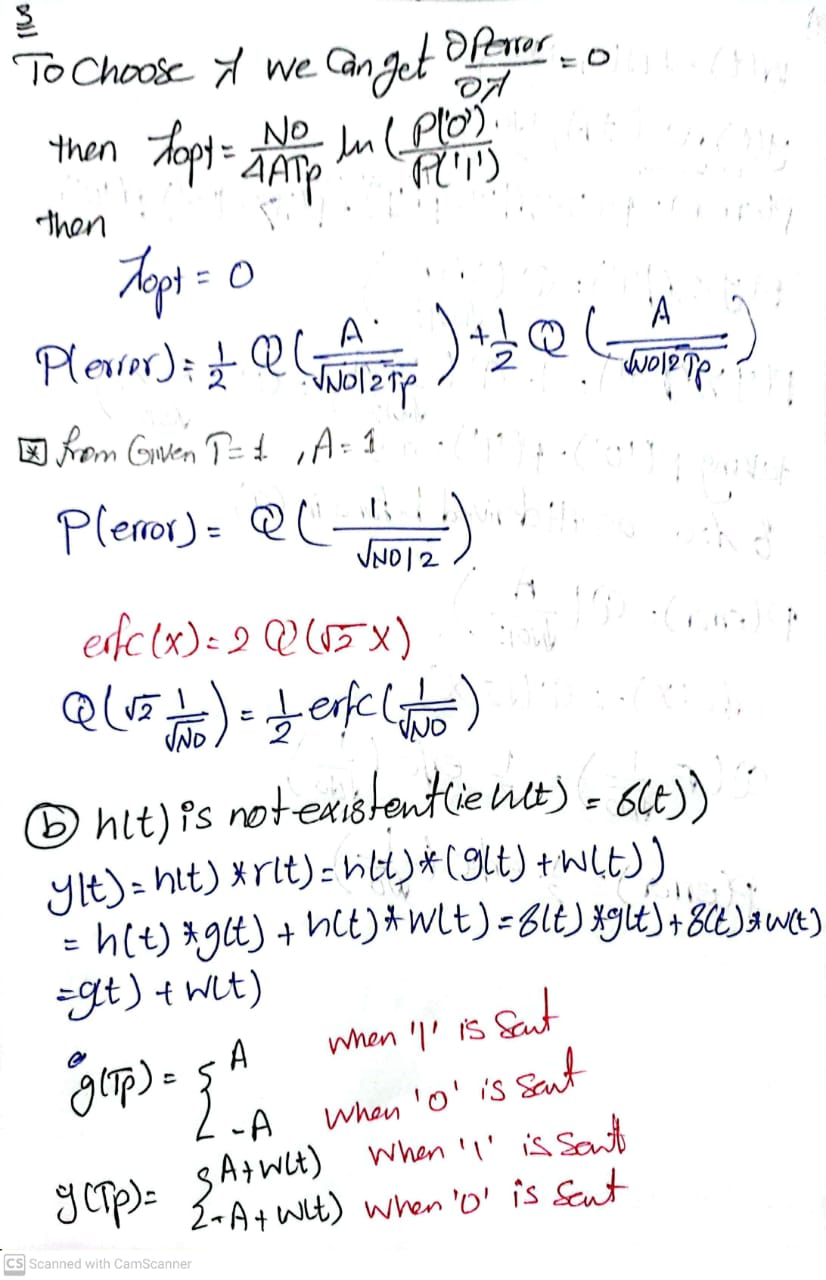
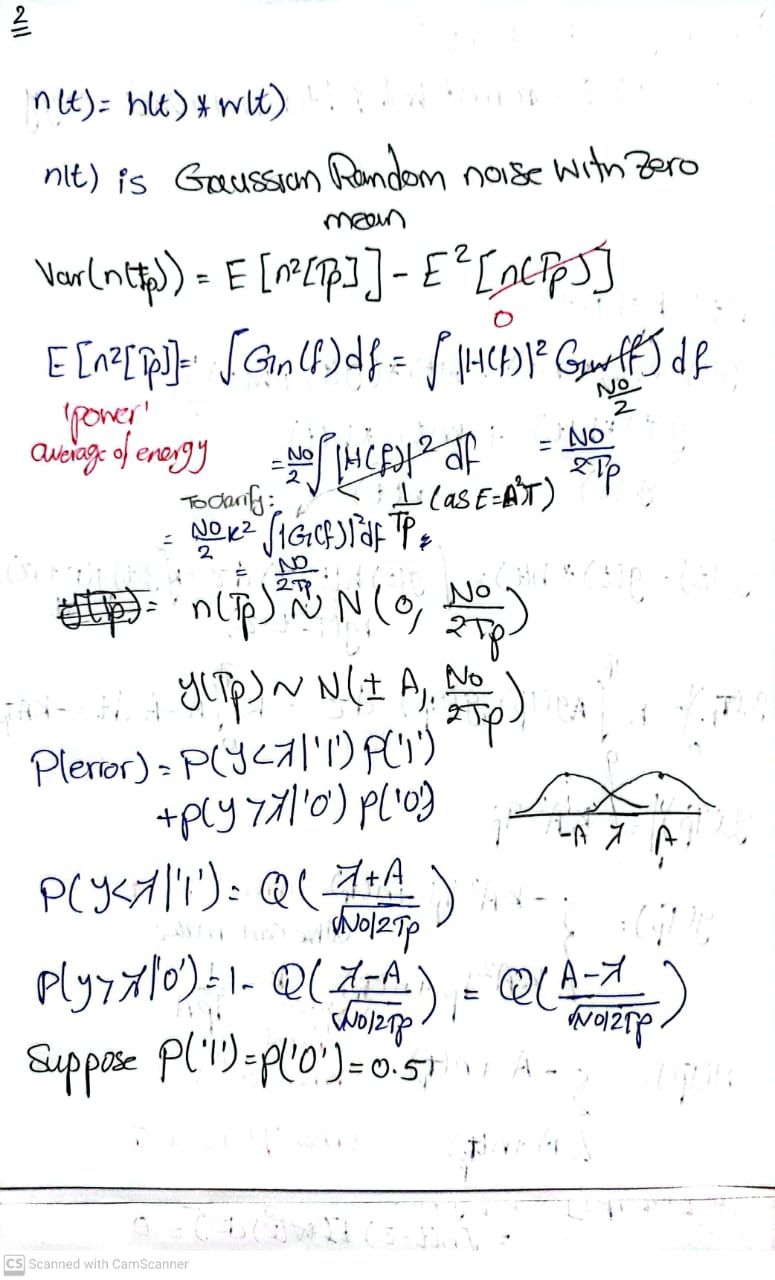




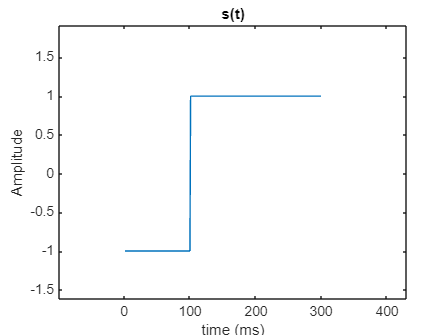


# **Part II: Simulation:**

# part2_2**Hand Analysis**



# **Figures With Comment**

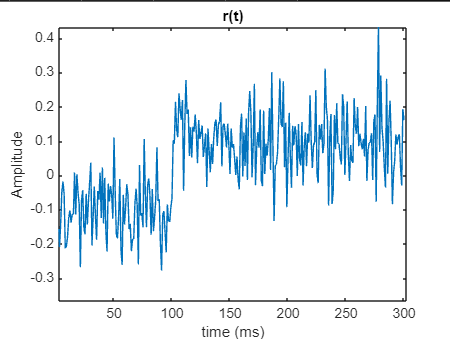
Input = [0 ,1 ,1]

SNR = 20

Number of samples = 100

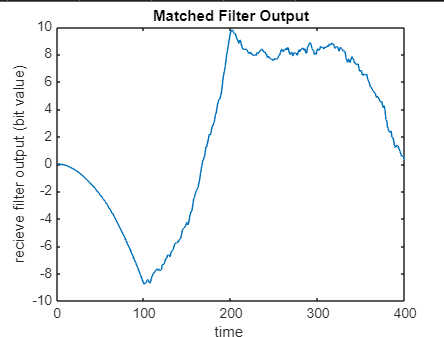
This figure represents the output of the binary source as a series of 0’s and 1’s

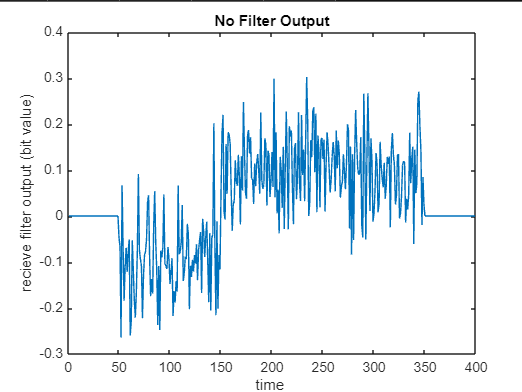
Note The x\_axis is multiple of 100 as the number of samples is 100

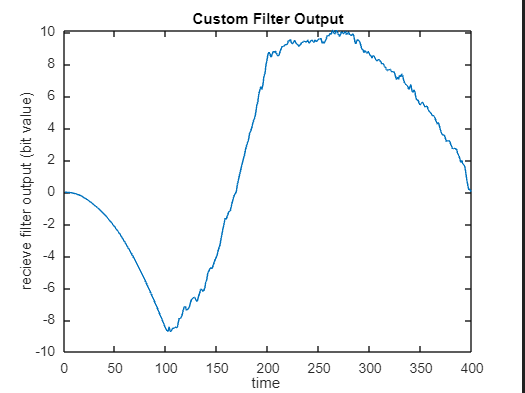
This figure represents the input after adding noise to it

Amplitude is divided by square root of sample number (sample\_number = 100) for normalization the bit energy

#### Output After Filters







#### Comment

No filter has the highest effect of noise compared to the two other filters

Output of the matched filter and this custom filter are smooth and when we increases the number of bit the matched filter appears to be more smooth slightly.

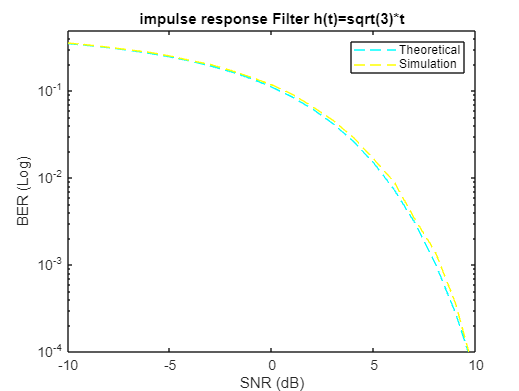
#### BER Theoretical and Simulation For The Three Filters

#### Matched Filter

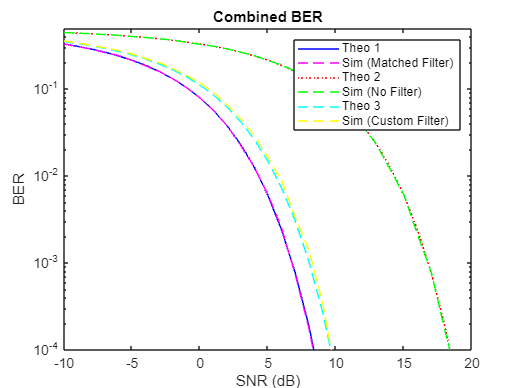
#### No Filter

#### 

#### Custom Filter



**Combined BER**



**Comment**

After plotting theoretical and simulated BER , It shows that the simulation of each filter almost follows the theoretical of its filter

**6 ) Is the BER an increasing or a decreasing function of  E/No? Why?**

Answer

It is a decreasing function as if the energy of the signal compared the energy of noise is high then the noise will not affect the signal so much and BER will decrease

so the more the E/No is the less the BER

**6 ) Which case has the lowest BER? Why?**

Answer

The first one with the matched filter.

The scenario with the matched filter achieves the lowest BER. This is because the matched filter is specifically designed to maximize the peak SNR of the signal.

Consequently, this minimizes the probability of errors, leading to a lower BER.

**Code**

clear;

close all;

samples\_number = 100;

bits = [0,1,1,0,1,1,0,1];

bits\_number = length(bits);

% Pulse Shape

[input] = pulse\_shape(bits\_number,samples\_number,bits);

s = reshape(input.', [], 1);

figure;

plot(s);

title('s(t)');

xlabel('time (ms)');

ylabel('Amplitude');

% Channel AWGN

% Generate Noise To Add

E = 1;

snr\_range = -10:1:20;

snr = 10 ^(snr\_range(30)/10);

[input\_with\_noise] = add\_noise(bits\_number,samples\_number,input,E,snr);

figure;

plot(input\_with\_noise);

title('r(t)');

xlabel('time (ms)');

ylabel('Amplitude');

% filters definations

delta\_filter = zeros(1,samples\_number);

delta\_filter(samples\_number/2)=1;

t = 0 : 1 : samples\_number -1;

tri\_filter = (sqrt(3)/samples\_number)\*t;

matched\_filter = ones(1,samples\_number);

filter ={matched\_filter,delta\_filter,tri\_filter};

output = {0,0,0};

for k=1 : 3

output{k} = conv(input\_with\_noise,filter{k});

end

% show output of each filter

figure;

plot(output{1});

title('Matched Filter Output');

xlabel('time ');

ylabel('recieve filter output (bit value)');

hold on ;

figure;

plot(output{2});

title('No Filter Output');

xlabel('time ');

ylabel('recieve filter output (bit value)');

hold on ;

figure;

plot(output{3});

title('Custom Filter Output');

xlabel('time ');

ylabel('recieve filter output (bit value)');

hold on ;

% sample the output to get stream of bits

for i=0:bits\_number-1

output\_1\_samples = sample(output{1},bits\_number,samples\_number);

output\_2\_samples = sample(output{2},bits\_number,samples\_number);

output\_3\_samples = sample(output{3},bits\_number,samples\_number);

end

% disp(output\_1\_samples)

% disp(output\_2\_samples)

% disp(output\_3\_samples)

% calculate accuracy of each filter

err\_prob\_1 = sum(output\_1\_samples ~= bits);

BER\_1 = err\_prob\_1/bits\_number;

err\_prob\_2 = sum(output\_2\_samples ~= bits);

BER\_2 = err\_prob\_2/bits\_number;

err\_prob\_3 = sum(output\_3\_samples ~= bits);

BER\_3 = err\_prob\_3/bits\_number;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% calculate BER for different SNR

bits\_number = 100000;

samples\_number = 10;

indices = randperm(bits\_number, 1);

bits= ones(bits\_number,1);

bits(indices)=0;

delta\_filter = zeros(1,samples\_number);

delta\_filter(samples\_number/2)=1;

t = 0 : 1 : samples\_number -1;

tri\_filter = (sqrt(3)/samples\_number)\*t;

matched\_filter = ones(1,samples\_number);

filter ={matched\_filter,delta\_filter,tri\_filter};

input =pulse\_shape(bits\_number,samples\_number,bits);

snr\_range = -10:1:20;

% Preallocate arrays to store BER simulations

BER\_sim\_1 = zeros (length(snr\_range),1);

BER\_sim\_2 = zeros (length(snr\_range),1);

BER\_sim\_3 = zeros (length(snr\_range),1);

BER\_theo\_1 = zeros(length(snr\_range),1);

BER\_theo\_2 = zeros(length(snr\_range),1);

BER\_theo\_3 = zeros(length(snr\_range),1);

for i = 1:length(snr\_range)

snr = 10 ^(snr\_range(i)/10);

input\_with\_noise = add\_noise(bits\_number,samples\_number,input,E,snr);

for k = 1:3

output{k} = conv(input\_with\_noise, filter{k}); % Consider using 'same' to maintain dimensionality

end

% Extracting the middle point for each bit period after convolution

output\_1\_samples = sample(output{1},bits\_number,samples\_number);

output\_2\_samples = sample(output{2},bits\_number,samples\_number);

output\_3\_samples = sample(output{3},bits\_number,samples\_number);

% disp(size(bits));

% Calculate errors and BER for each filter

err\_prob\_1 = sum(output\_1\_samples.' ~= bits);

BER\_sim\_1(i) = err\_prob\_1 / bits\_number;

err\_prob\_2 = sum(output\_2\_samples.' ~= bits);

BER\_sim\_2(i) = err\_prob\_2 / bits\_number;

err\_prob\_3 = sum(output\_3\_samples.' ~= bits);

% disp(BER\_sim\_1)

BER\_sim\_3(i) = err\_prob\_3 / bits\_number;

BER\_theo\_1(i)=0.5\*erfc(sqrt(snr));

BER\_theo\_2(i)=0.5\*erfc(sqrt(snr/samples\_number));

BER\_theo\_3(i)=0.5\*erfc(((sqrt(3)/(2))\*sqrt(snr)));

end

% Update plot commands to reflect all data

figure;

semilogy(snr\_range, BER\_theo\_1, 'b-');

hold on;

semilogy(snr\_range, BER\_sim\_1, 'm--');

semilogy(snr\_range, BER\_theo\_2, 'r:');

semilogy(snr\_range, BER\_sim\_2, 'g--');

semilogy(snr\_range, BER\_theo\_3, 'c--');

semilogy(snr\_range, BER\_sim\_3, 'y--');

hold off;

ylim([10^-4 0.5]);

xlabel('SNR (dB)');

ylabel('BER');

legend('Theo 1', 'Sim (Matched Filter)', 'Theo 2', 'Sim (No Filter)', 'Theo 3', 'Sim (Custom Filter)');

title('Combined BER');

figure;

semilogy(snr\_range, BER\_theo\_1, 'b-');

hold on;

semilogy(snr\_range, BER\_sim\_1, 'm--');

hold off;

title('Matched Filter');

ylim([10^-4 0.5]);

xlabel('SNR (dB)');

ylabel('BER (Log)');

legend('Theoretical', 'Simulation');

figure;

semilogy(snr\_range, BER\_theo\_2, 'r:');

hold on;

semilogy(snr\_range, BER\_sim\_2, 'g--');

hold off;

title('No Filter');

ylim([10^-4 0.5]);

xlabel('SNR (dB)');

ylabel('BER (Log)');

legend('Theoretical', 'Simulation');

figure;

semilogy(snr\_range, BER\_theo\_3, 'c--');

hold on;

semilogy(snr\_range, BER\_sim\_3, 'y--');

hold off;

title('impulse response Filter h(t)=sqrt(3)\*t');

ylim([10^-4 0.5]);

xlabel('SNR (dB)');

ylabel('BER (Log)');

legend('Theoretical', 'Simulation');

function [input] = pulse\_shape(bits\_number,samples\_number,bits)

input = ones(bits\_number,samples\_number);

for i=1 : bits\_number

if bits(i) == 0

input(i,:) = -input(i,:);

end

end

end

function [input\_with\_noise] = add\_noise(bits\_number,samples\_number,input,E,snr)

sigma = sqrt(E/(2.0\*snr));

noise = normrnd(0,sigma,[1,bits\_number\*samples\_number]);

input\_with\_noise = input/sqrt(samples\_number);

% add noise to input

for i=1 : bits\_number

input\_with\_noise(i,:) = input\_with\_noise(i,:) + noise((samples\_number)\*(i-1)+1:(samples\_number)\*(i));

end

input\_with\_noise = reshape(input\_with\_noise.', [], 1);

end

function [samples]=sample(output,bits\_number,samples\_number)

samples = ones(1,bits\_number);

for i=0:bits\_number-1

samples(i+1) = (output((samples\_number - 1) + samples\_number \* i+1)) > 0;

end

end