Cairo University

Faculty of Engineering
Cairo University

Digital Communication

Assignment Two

Matched Filter

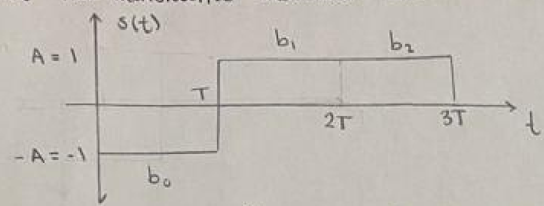
STUDENTS INFORMATION

Name	Section	Bench Number
Yasmine Ashraf Ghanem	2	37
Yasmin Abdullah Nasser	2	38

PART I : HAND ANALYSIS

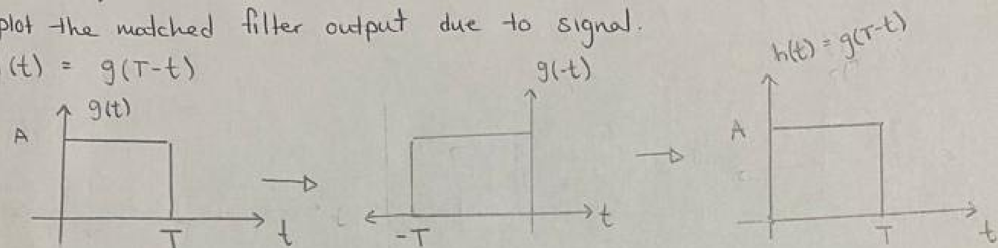
Part 1:

(a) plot the transmitted baseband waveform s(t) for b_0 = '0', b_1 = '1', b_2 = '1'

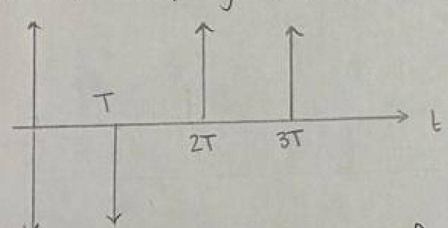


(b) plot the matched filter output due to signal.

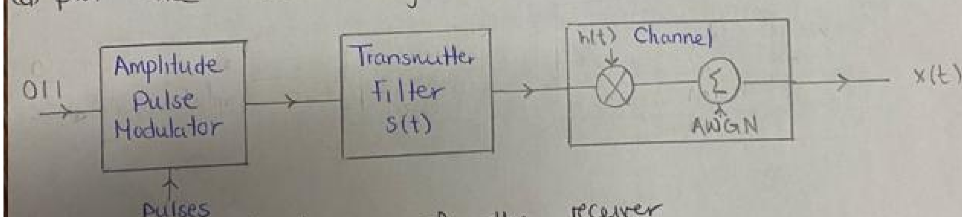
$h(t) = g(T-t)$



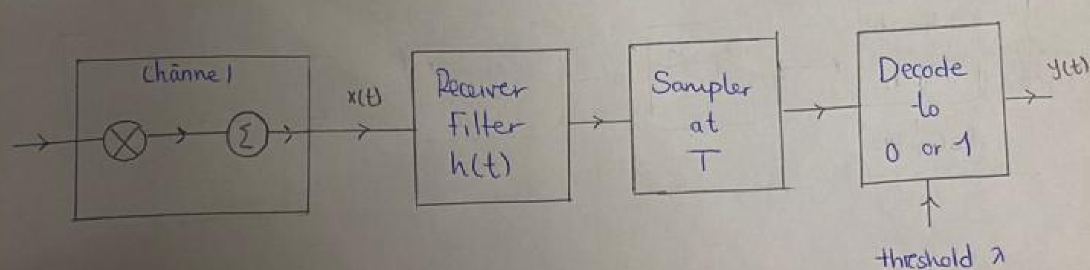
(c) mark the sampling instants to detect b_0, b_1, b_2.



(d) plot the block diagram of the transmitter.



(e) plot the block diagram of the receiver

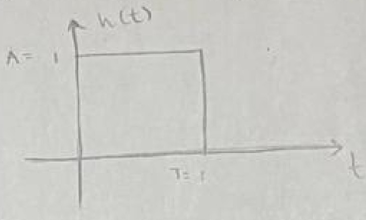


PART II : SIMULATION

1) Derive the probability of error in the three mentioned cases.

Part II: Derive the probability of error for:

(a) the receiver filter $h(t)$ is a matched filter with unit energy



$P(e) = P(1)\, P(\hat{1}|0) + P(0)\, P(\hat{0}|1)$

→ symmetric $P(0) = P(1) = \frac{1}{2}$

$P(e) = \frac{1}{2}\left(P(\hat{1}|0) + P(\hat{0}|1)\right)$

→ matched filter $h(t) = g(T-t)$

$r(t) = g(t) + w(t)$ → AWGN $(N_0/2)$ → σ

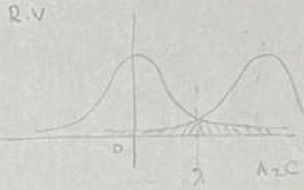
$y(t) = r(t) * h(t) = g(t) * h(t) + w(t) * h(t)$

$= \int_0^T g(\tau) h(t-\tau)\, d\tau + \int_{-\infty}^{\infty} w(\tau) h(t-\tau)\, d\tau$

→ we sample @ T $= \underbrace{\int_0^1 A C \hat{h}(t-\tau)\, d\tau}_{\text{Constant} = A^2 C = C} + \underbrace{\int_{-\infty}^{\infty} w(\tau) h(t-\tau)\, d\tau}_{R.V}$

$y(T) = C + \int_0^1 w(\tau)\, d\tau$ → R.V, gaussian distribution

$y(T) = C + n(T)$ → noise at sample



$\boxed{f_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma}}}$ → $f_y(y|'0') = \frac{1}{\sqrt{2\pi \frac{N_0}{2}}} e^{-\frac{(y-(-1))^2}{2 N_0/2}}$ (-1)

$f_y(y|'0') = \frac{1}{\sqrt{\pi N_0}} e^{\frac{-(y+1)^2}{N_0}}$

$\boxed{\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^{\infty} e^{-z^2} dz}$

$P(\hat{1}|0) = P(y(T) > \lambda \,|\, 0)$

$= \int_\lambda^\infty f_y(y|0)\, dy = \int_\lambda^\infty \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y+1)^2}{N_0}} dy$

① let $z = \frac{y+1}{\sqrt{N_0}}$ $dz = \frac{dy}{\sqrt{N_0}}$ → $\frac{1}{\sqrt{\pi}} \int_{\frac{\lambda+1}{N_0}}^{\infty} e^{-z^2} \frac{dy}{\sqrt{N_0}} = \frac{1}{2} \frac{2}{\sqrt{\pi}} \int_\lambda^\infty e^{-z^2} dz$

∴ $P(\hat{1}|0) = \frac{1}{2} \text{erfc}\left(\frac{\lambda+1}{N_0}\right)$

$P(\hat{0}|1) = P(y(T) < \lambda \,|\, 1) = \int_{-\infty}^{\lambda} f_y(y|1)\, dy = \int_{-\infty}^{\lambda} \frac{1}{\sqrt{\pi N_0}} \cdot e^{-\frac{(y-1)^2}{N_0}}$ (1)

② let $z = \frac{y-1}{\sqrt{N_0}}$ $dz = \frac{dy}{\sqrt{N_0}}$ → $\frac{1}{\sqrt{\pi}} \int_{-\infty}^{\frac{\lambda+1}{N_0}} e^{-z^2} dz = 1 - \text{erfc}\left(\frac{\lambda-1}{\sqrt{N_0}}\right)$ → symmetry

from ① & ②

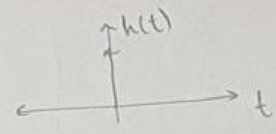
$P(e) = \frac{1}{2} \times 2\, (P(1|0)) = \frac{1}{2} \text{erfc}\left(\frac{\lambda-1}{\sqrt{N_0}}\right)$

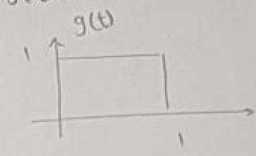
OR

$P(e) = \frac{1}{2} \text{erfc}\left(\frac{\lambda+1}{\sqrt{N_0}}\right)$

(b) the receiver filter $h(t)$ is non existent $(h(t) = \delta(t))$

$P(e) = \frac{1}{2}(P(1|0) + P(0|1))$





filter $h(t) = \delta(t)$

$r(t) = g(t) + w(t) \rightarrow$ AWGN

$y(t) = r(t) * h(t) = g(t) * h(t) + w(t) * h(t)$

$y(t) = g(t) * h(t) + \int_0^1 w(t)$

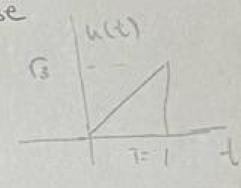
$\rightarrow$ as we sample at T

$y(T) = \underbrace{CA}_{\text{constant}} + \underbrace{n(T)}_{\rightarrow R.V} = \pm 1 + n(T)$

$\rightarrow$ using same derivations used in (a)

$\therefore P(e) = \frac{1}{2}\operatorname{erfc}\left(\frac{\lambda - 1}{\sqrt{N_0}}\right)$ OR $P(e) = \frac{1}{2}\operatorname{erfc}\left(\frac{\lambda + 1}{\sqrt{N_0}}\right)$

(c) the receiver filter $h(t)$ has the impulse response



$P(e) = \frac{1}{2}(P(1|0) + P(0|1))$

filter $h(t) = \sqrt{3}\,t$

$r(t) = g(t) + w(t) \rightarrow$ AWGN

$y(t) = r(t) * h(t) = g(t) * h(t) + w(t) * h(t)$

$= \int_0^1 g(\tau)\, h(t-\tau)\, d\tau + \int_0^1 w(\tau)\, h(t-\tau)\, d\tau$

$= \int_0^1 \sqrt{3}\,t + n(t) = \frac{\sqrt{3}}{2} t^2 \Big|_0^1 + n(t) = \pm\frac{\sqrt{3}}{2} + n(t)$ $\rightarrow$ gaussian R.V

$y(T) = \pm\frac{\sqrt{3}}{2} + n(T)$

$f_y(y|0) = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y+\sqrt{3}/2)^2}{2 N_0/2}} = \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y+\sqrt{3}/2)^2}{N_0}}$

$P(\hat{1}|0) = P(y > \lambda | 0) = \int_\lambda^\infty f_y(y|0)\, dy = \int_\lambda^\infty \frac{1}{\sqrt{\pi N_0}} e^{-\frac{(y+\sqrt{3}/2)^2}{N_0}} dy$

$\Rightarrow$ transform

$P(\hat{1}|0) = \frac{1}{2}\frac{2}{\sqrt{\pi}} \int_{\frac{\lambda+\sqrt{3}/2}{\sqrt{N_0}}}^\infty e^{-z^2}\, dz = \frac{1}{2}\operatorname{erfc}\left(\frac{\lambda + \sqrt{3}/2}{\sqrt{N_0}}\right)$

$\rightarrow$ same for:

$P(\hat{0}|1) = P(y < \lambda | 1) = \frac{1}{2}\operatorname{erfc}\left(\frac{\lambda - \sqrt{3}/2}{\sqrt{N_0}}\right)$

due to symmetry
$\rightarrow$ both are equal

$P(e) = \frac{1}{2} \times 2\,(P(\hat{1}|0))$
OR $P(e) = \frac{1}{2} \times 2\, P(\hat{0}|1)$

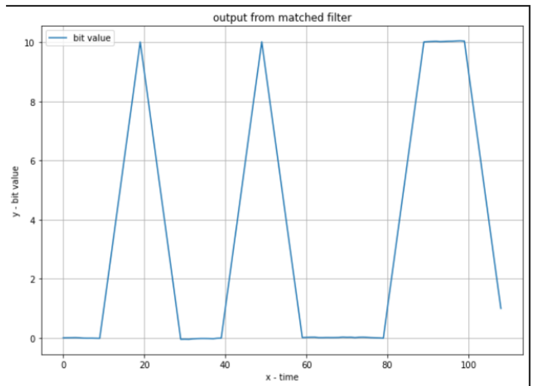
$P(e) = \frac{1}{2}\operatorname{erfc}\left(\frac{\lambda + \sqrt{3}/2}{\sqrt{N_0}}\right)$

2) Write the code that simulates the system
In CODE section

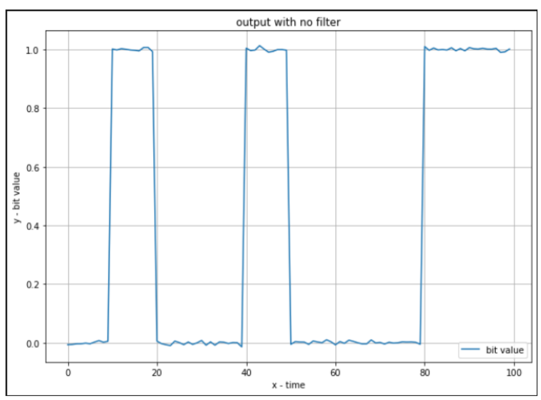
3) Plot the output of the receive filter for the three mentioned cases

These outputs were a result of a small sample of bits (10 bits) as the large number of bits wasn't showing in the graphs.

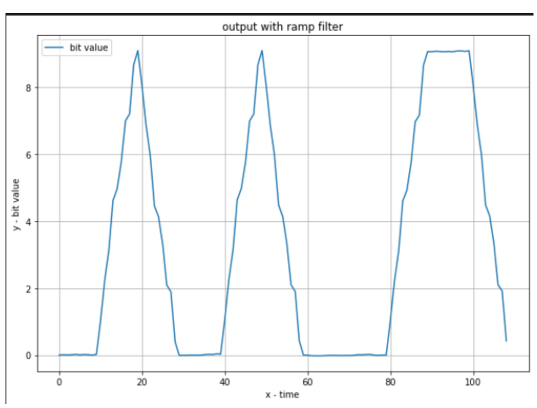
a) Matched Filter



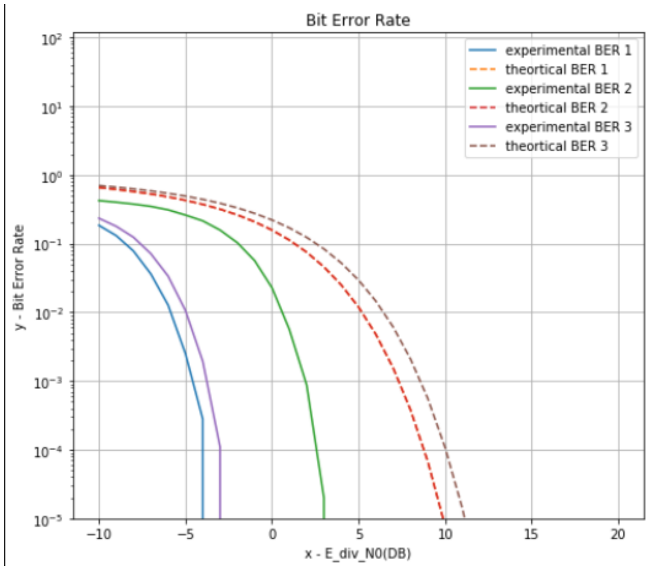
b) No Filter



c) Ramp Filter



4) Plot the theoretical and simulated Bit Error Rate (BER) Vs E/No for the three mentioned cases.



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

The BER is a decreasing function as shown in the graph above as when E/No decreases the No increases which increases the noise power but SNR which is the signal to the noise ratio decreases which will result in a higher probability of flipping the bit from -A to higher than zero and A to less than zero as signal after noise.

6) Which case has the lowest BER? Why?

Matched filter has the lowest BER which shows that it is the optimum receiving filter having the max signal to noise ratio with impulse response h(t) = kg (T - t) which achieves maximum efficiency and hence lowest bit error rate.

CODE

```
# GENERAL FUNCTIONS
def generatRandomBits():
    # generate a random number with number_of_bits passed
    return np.random.choice([0, 1], size=(number_of_bits))

def convertToPulses(generated_bits):
    generated_bits[generated_bits == 0] = -1

    signal = np.repeat(generated_bits, fs)

    return signal

def addWhiteGaussianNoise(generated_bits, sigma):

            generated_noise   =   np.random.normal(loc=0,   scale=sigma,
size=fs*number_of_bits)

    scaled_samples_with_noise = np.ones((generated_bits.shape[0], fs))
    scaled_samples = np.ones((generated_bits.shape[0], fs))
    for i in range(generated_bits.shape[0]):
        scaled_samples[i, :] *= generated_bits[i]
        scaled_samples_with_noise[i, :] *= generated_bits[i]
        scaled_samples_with_noise[i, :] += generated_noise[i*fs:(i+1)*fs]
    return scaled_samples, scaled_samples_with_noise

def convolute(scaled_samples_with_noise, received_filter):

    convolution_result_sampled_Tp = np.zeros(number_of_bits)

    if(received_filter is None):
        convolution_result = scaled_samples_with_noise.flatten()
    else:
                                                 convolution_result      =
np.convolve(scaled_samples_with_noise.flatten(), received_filter)
    for i in range(number_of_bits):
         convolution_result_sampled_Tp[i] = convolution_result[(fs - 1) +
fs * i]
```

```python
    return convolution_result, convolution_result_sampled_Tp

def calculateProbabilityOfError(true_bits, convolution_result, Z):

    # applying threshold to sample output with lambda = 0
    received_samples = np.ones(true_bits.shape[0])
    received_samples += (-2 * (convolution_result < 0))

    # calculate simulation probability of error
    simulation = np.sum(received_samples != true_bits)
    simulation /= true_bits.shape[0]

    # calculate theoretical probability of error
    theoretical = math.erfc(Z)
```

```python
# CASE ONE :  The receive filter h(t) is a matched filter with unit energy

received_filter_matched = np.ones(fs)
for E_div_N0_db in range(-10, 21):
    E_div_N0 = 10 ** (E_div_N0_db/10)
    transmitted_samples, received_samples =
addWhiteGaussianNoise(generated_bits, 1/(2*E_div_N0))
    filtered_samples, filtered_bits = convolute(received_samples,
received_filter_matched)

# plot output of the received filter
plt.figure(figsize=(10,7))
plt.plot(range(0, filtered_samples.flatten().shape[0]),
filtered_samples.flatten(), label = "bit value")

plt.xlabel('x - time')
plt.ylabel('y - bit value')
plt.title('output from matched filter')

plt.legend()
plt.grid()
plt.show()
# CASE TWO: The receive filter h(t) is not existent (i.e. h(t) = δ(t))
```

```python
received_filter_matched = None

for E_div_N0_db in range(-10, 21):
    E_div_N0 = 10 ** (E_div_N0_db/10)
    transmitted_samples, received_samples = 
addWhiteGaussianNoise(generated_bits, 1/(2*E_div_N0))
    filtered_samples, filtered_bits = convolute(received_samples, 
received_filter_matched)

# plot output of the received filter
plt.figure(figsize=(10,7))
plt.plot(range(0, filtered_samples.flatten().shape[0]), 
filtered_samples.flatten(), label = "bit value")

plt.xlabel('x - time')
plt.ylabel('y - bit value')
plt.title('output with no filter')
plt.legend()
plt.grid()
plt.show()
```

```python
# CASE THREE:  The receive filter is a ramp with h(t) = √3t
received_filter_ramp = np.random.uniform(low=0, high=3**0.5, 
size=number_of_bits)
E = 1
for E_div_N0_db in range(-10, 21):
    E_div_N0 = 10 ** (E_div_N0_db/10)
    transmitted_samples, received_samples = 
addWhiteGaussianNoise(generated_bits, E/(2*E_div_N0))
    filtered_samples, filtered_bits = convolute(received_samples, 
received_filter_ramp)

#plotting
plt.figure(figsize=(10,7))
plt.plot(range(0, filtered_samples.flatten().shape[0]), 
filtered_samples.flatten(), label = "bit value")

plt.xlabel('x - time')
plt.ylabel('y - bit value')
```

```
plt.title('output with ramp filter')
plt.legend()
plt.grid()
plt.show()
```

```
# CONSTANTS TO CALCULATE THE BIT ERROR RATE
fs = 20
number_of_bits = 10**5
```

```
# CALCULATE BIT ERROR RATE FOR THE THREE CASES
generated_bits = generatRandomBits()

# receive with matched filter
received_filter_matched = np.ones(fs)
BER_simulation_1 = []
BER_theortical_1 = []

# receive with no filter
received_filter_empty = None
BER_simulation_2 = []
BER_theortical_2 = []

# receive with ramp filter
received_filter_ramp = np.random.uniform(low=0, high=3**0.5, size=fs)
BER_simulation_3 = []
BER_theortical_3 = []

for E_div_N0_db in range(-10, 21):
    E_div_N0 = 10 ** (E_div_N0_db/10)
    transmitted_samples, received_samples =
addWhiteGaussianNoise(generated_bits, 1/(2*E_div_N0))
    filtered_samples1, filtered_bits1 = convolute(received_samples,
received_filter_matched)
    filtered_samples2, filtered_bits2 = convolute(received_samples,
received_filter_empty)
    filtered_samples3, filtered_bits3 = convolute(received_samples,
received_filter_ramp)
    theoretical_BER, simulation_BER =
calculateProbabilityOfError(generated_bits, filtered_bits1, E_div_N0 **
0.5)
```

```
    BER_simulation_1.append(simulation_BER)
    BER_theortical_1.append(theoretical_BER)

    theoretical_BER, simulation_BER =
calculateProbabilityOfError(generated_bits, filtered_bits2, E_div_N0 **
0.5)
    BER_simulation_2.append(simulation_BER)
    BER_theortical_2.append(theoretical_BER)

    theoretical_BER, simulation_BER =
calculateProbabilityOfError(generated_bits, filtered_bits3, (3**0.5/2) *
E_div_N0 ** 0.5)
    BER_simulation_3.append(simulation_BER)
    BER_theortical_3.append(theoretical_BER)
```

```
# PLOTTING
plt.figure(figsize=(8,7))
plt.plot(range(-10, 21), BER_simulation_1, label = "simulation BER 1")
plt.plot(range(-10, 21), BER_theortical_1, "--", label = "theoretical BER
1")

plt.plot(range(-10, 21), BER_simulation_2, label = "simulation BER 2")
plt.plot(range(-10, 21), BER_theortical_2, "--", label = "theoretical BER
2")

plt.plot(range(-10, 21), BER_simulation_3, label = "simulation BER 3")
plt.plot(range(-10, 21), BER_theortical_3, "--", label = "theoretical BER
3")

plt.xlabel('x - E_div_N0(DB)')
plt.ylabel('y - Bit Error Rate')
plt.yscale('log')
plt.ylim(10**(-5))
plt.title('Bit Error Rate')
plt.legend()
plt.grid()
plt.show()
```