Assignment 2 [DC]

Name: علياء عبدالعزيز على عبدالله

Section: 1

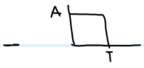
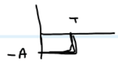
Bench Number: 26

Part 1

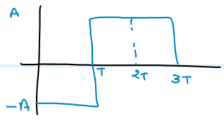
✍️ Hand Analysis

Math

Part ①

 1 was sent  "0" was sent , $g(t)$ input pulse , $r(t)$ received from channel

a)

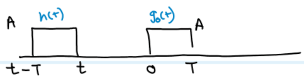


b) $h(t) = K\,g(T-t) = g(T-t)$ "let $K=1$" $= g_{\text{"1"}}(t) = \begin{cases} A & 0 \le t \le T \\ 0 & \text{otherwise} \end{cases} = A \cdot \text{rect}\left(\frac{t - T/2}{T}\right)$

$\therefore\ y(t) = g(t) * h(t) + w(t) * h(t)$

$= g_o(t) + n(t)$ due to 1 pulse

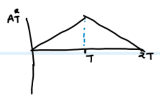
→ when "1" was sent



— ① for $0 < t < T$: $g_o(t) = \int_0^t A^2\,dt = A^2 t$

— ② for $T < t < 2T$: $g_o(t) = \int_{t-T}^{T} A^2\,dt = A^2\,[2T - t]$

$g_o(t) = \begin{cases} A^2 t & 0 < t < T \\ A^2\,[2T - t] & T < t < 2T \end{cases}$

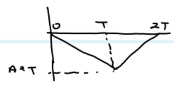


$y(t) = g_o(t)$ ignoring noise when "1" sent

→ when "0" sent

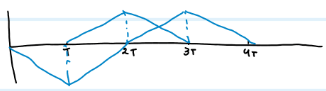
$\therefore\ g_{\text{"0"}}(t) = -g_{\text{"1"}}(t)$

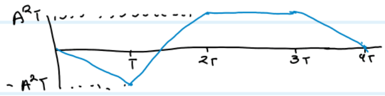
$\therefore\ g_{o_{\text{"0"}}}(t) = -g_{\text{"1"}}(t) * h(t) = -g_{o_{\text{"1"}}}(t) = \begin{cases} -A^2 t & 0 < t < T \\ -A^2[2T - t] & T < t < 2T \end{cases}$



→ Final Output Signal ignoring noise

$y(t)$

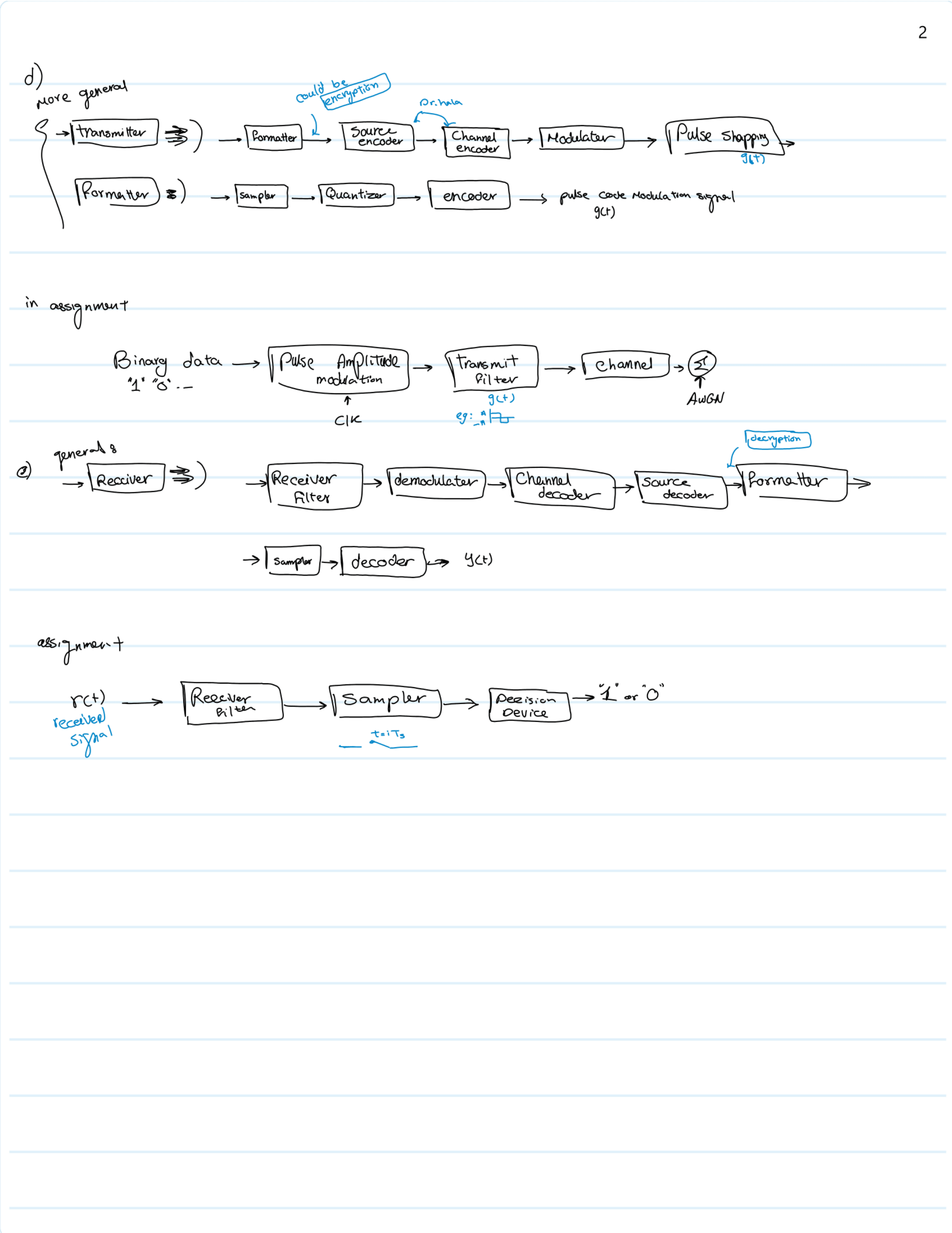




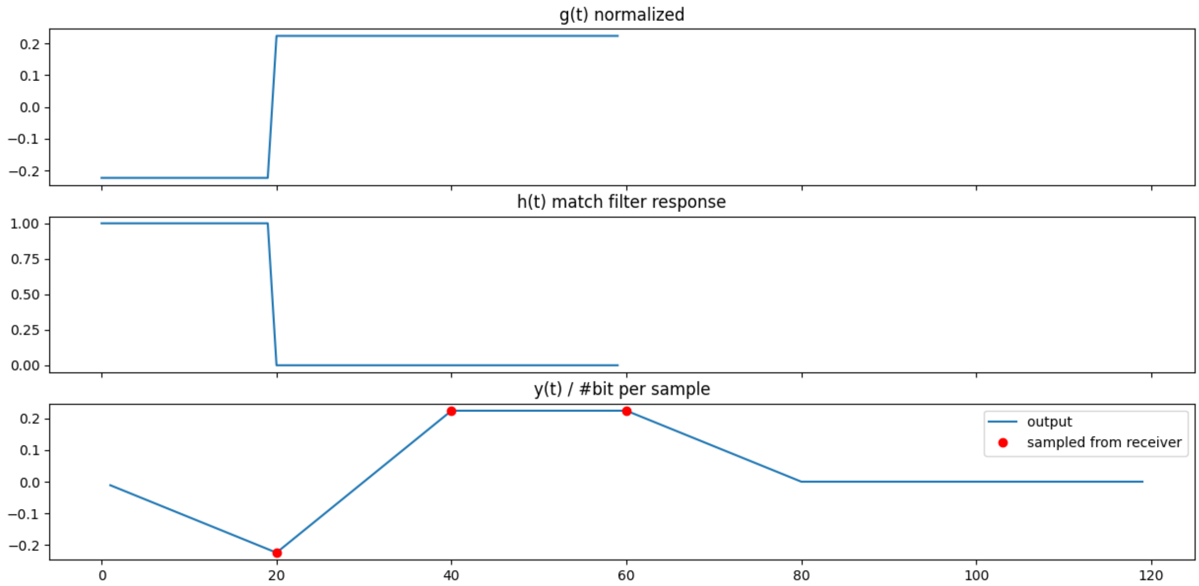
c) will sample at $T, 2T, 3T$

Output after sampling

Blocks



🤖 Simulation output



😎 Code

Constants

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import convolve, fftconvolve
from scipy.special import erfc
```

```


num_bits = 100              # number of bits will send
num_samples_per_bit = 20    # number of sample to convert it to cont
Ts = 1                      # discrete sampling time
A = 1                       # amplitude of signal
g1_t = A * np.ones(num_samples_per_bit) # pulse shape when "1" was s
g1_t_norm = np.linalg.norm(g1_t, axis=0) # for normalizaton 1/sqrt(n
print(g1_t_norm) #4.47213595499958
```

Functions

```
def generate_bipolar_NRT_sig(data: np.ndarray, num_samples_per_bit =
    """
    parameters:
        data: binary array of shape num_bits
    return:
        time: numpy array represent time axis
        g_t : output signal from transmitter
```

```
    """
    g_t = np.repeat(data, num_samples_per_bit) # repeat every data p
    g_t = ((g_t * 2) - 1) # convert 0 -> -A | 1 -> A
    return g_t / g1_t_norm



def generate_match_filter(g1_t: np.ndarray, num_bits = num_bits, num_
    # g1_t : impulse shape when sending "1" must be in shape (num_sa
    h_t = g1_t[::-1] # just reverse it
    h_t = np.concatenate([h_t, np.zeros((num_bits - 1) * num_samples_
    return h_t
```

Simulation

```
data = np.array([0, 1, 1])
g_t = generate_bipolar_NRT_sig(data)
h_t = generate_match_filter(np.ones(num_samples_per_bit) * A, 3)
y_t = convolve(g_t, h_t) # ignoring noise return the same shape of g
```

Plotting

```
fig, axs = plt.subplots(3, 1, figsize=(15, 7), sharex=True)
clock = (np.arange(3) + 1) * num_samples_per_bit # for sampling

axs[0].set_title("g(t) normalized")
axs[0].plot(g_t)

axs[1].set_title("h(t) match filter response")
axs[1].plot(h_t)



axs[2].set_title("y(t) / #bit per sample")
axs[2].plot(np.arange(1, len(y_t)+1), y_t / num_samples_per_bit, lab
axs[2].plot(clock, y_t[clock-1] / num_samples_per_bit, "ro", label =

plt.legend()
plt.show()
```

Part 2

✍️ Hand Analysis

part I

$T = 1$, $w(t) \sim N(0, \frac{N_0}{2})$ $g(t) =$ or

a) Matched filter

— $h(t) = K g(T-t)$ / Energy of filter $= 1 = \frac{N_0}{2}$ $\Rightarrow N_0 = 2$

$= g_{"1"}(t) = 1$ for $0 < t < 1$

— $y(t) = r(t) * h(t) = \underbrace{g(t) * h(t)}_{g_0(t)} + \underbrace{w(t) * h(t)}_{n(t)}$

$$g_0(T) = \int_{-\infty}^{\infty} g(\tau) \cdot h(T-\tau)\, d\tau = \int_0^T g(\tau) \cdot g_{"1"}(\tau)\, d\tau = \begin{cases} 1 & \text{"1" was sent} \\ -1 & \text{"0" was sent} \end{cases} = \begin{cases} A^2 T \\ -A^2 T \end{cases}$$

$$\text{So } y(t) = \begin{cases} \cancel{1}\, A^2T + n(t) & \text{"1" was sent} \\ -\cancel{1}\, A^2T + n(t) & \text{"0" was sent} \end{cases}$$

$$— E(n(T)) = E\left(\int_{-\infty}^{\infty} w(\tau)\, h(T-\tau)\, d\tau\right) = \int_{-\infty}^{\infty} E[\underbrace{w(\tau)}_{\to 0}]\, E[h(T-\tau)]\, d\tau = 0$$

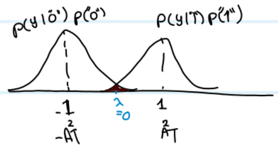
$$Var[n(T)] = E[n^2(T)] - (E[\underbrace{n(T)}_{\to 0}])^2 = R_n(0) = \int_{-\infty}^{\infty} Q_n(F)\, dF = \int_{-\infty}^{\infty} Q_w(F)\, |H(F)|^2\, dF$$

$$= \frac{N_0}{2} \int_{0}^{T} |h(t)|^2\, dt$$

$$= \frac{N_0}{2} A^2 \cdot T$$

$p(y|"1") \sim N(A^2T, \frac{N_0}{2}A^2T)$ "1" was sent

$p(y|"0") \sim N(-A^2T, \frac{N_0}{2}A^2T)$ "0" was sent



By symmetry (same variance) $\lambda = 0$

$- P(\text{error}) = P(\text{err} \mid \text{"1"})\, P(\text{"1"}) + P(\text{err} \mid \text{"0"})\, P(\text{"0"})$

$\because P(\text{"1"}) = P(\text{"0"}) = 0.5$

$\because P(y = \text{"1"} \mid \text{"0"}) = P(\text{error} \mid \text{"0"}) = P(y > 0 \mid \text{"0"})$

$\because$ By symmetry $P(\text{error} \mid \text{"1"}) = P(\text{error} \mid \text{"0"})$

$\therefore P(\text{error}) = P(\text{error} \mid \text{"0"}) = P(y > 0 \mid 0) = P_r\left(z > \dfrac{0 + A^2 T}{\sqrt{\frac{N_0}{2} A^2 T}}\right)$

$\because E = g_0(T)$

$= Q\left(\dfrac{\sqrt{2}.A.\sqrt{T}}{\sqrt{N_0}}\right) = \dfrac{1}{2}\,\text{erfc}\left(A\sqrt{\dfrac{T}{N_0}}\right)$

$= \dfrac{1}{2}\,\text{erfc}\left(\sqrt{\dfrac{E}{N_0}}\right)$

$\text{erfc}(x) = 2Q(\sqrt{2}x)$

$Q(x) = \dfrac{1}{2}\,\text{erfc}\left(\dfrac{1}{\sqrt{2}}x\right)$

---

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

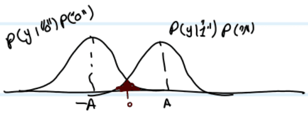
$- \because y(t) = \underbrace{g(t) * h(t)}_{g_0(t)} + \underbrace{w(t) * h(t)}_{n(t)}$

$- g_0(t) = g(t) * h(t) = g(t) * \delta(t) = g(t) = \begin{cases} A & \text{"1" was sent} \\ -A & \text{"0" was sent} \end{cases}$ for $0 < t < 1$

$- n(t) = w(t) * \delta(t) = w(t) \sim N(0, \frac{N_0}{2})$

$P(y \mid \text{"1"}) \sim N(A, \frac{N_0}{2})$ "1" was sent

$- P(y \mid \text{"0"}) \sim N(-A, \frac{N_0}{2})$ "0" was sent



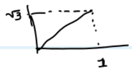
By symmetry $\lambda = 0$

$\because P(\text{"1"}) = P(\text{"0"}) = \frac{1}{2}$, $\because$ By symmetry $P(\text{err} \mid \text{"1"}) = P(\text{err} \mid \text{"0"})$

So $P(\text{error}) = P(\text{err} \mid 1)\, P(1) + P(\text{err} \mid 0)\, P(0)$

$= P(\text{err} \mid 0) = P(y > \overset{\lambda}{0} \mid 0) = P\left(z > \dfrac{0 + A}{\sqrt{N_0/2}}\right) = Q\left(\dfrac{A\sqrt{2}}{\sqrt{N_0}}\right) = \dfrac{1}{2}\,\text{erfc}\left(\dfrac{A}{\sqrt{N_0}}\right) = \dfrac{1}{2}\,\text{erfc}\left(\dfrac{E}{\sqrt{N_0}}\right)$

c) $h(t) = \begin{cases} \frac{\sqrt{3}}{T} t & 0 < t < T \\ 0 & \text{otherwise} \end{cases}$



$y(t) = \underbrace{g(t) * h(t)}_{g_0(t)} + \underbrace{w(t) * h(t)}_{n(t)}$

$- g_{0_{"1"}}(T) = \int_{-\infty}^{\infty} g_{"1"}(\tau)\, h(T-\tau)\, d\tau = \text{Area}(\ldots)$

$= \frac{\sqrt{3}}{2} TA \Rightarrow$ "1" was sent

$g_0(T) = \begin{cases} \frac{\sqrt{3}}{2} AT & \text{"1" was sent} \\ -\frac{\sqrt{3}}{2} AT & \text{"0" was sent} \end{cases}$

since $g_{"0"}(T) = -g_{"1"}(T) = -A$

so $g_{0_{"0"}}(T) = g_{0_{"1"}}(T)$

$- n(T) = \int_{-\infty}^{\infty} w(\tau)\, h(T-\tau)\, d\tau$

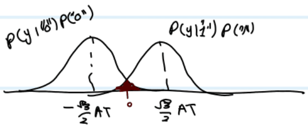
$E(n(T)) = E\left(\int_{-\infty}^{\infty} w(\tau)\, h(T-\tau)\, d\tau\right) = \int_{-\infty}^{\infty} E(\underbrace{w(\tau)}_{\to 0}) E(h(T-\tau))\, d\tau) = 0$

$Var(n(t)) = E(n(t)^2) - \underbrace{E(n(t))^2}_{\to 0} = E(n(t)^2) = R_n(0) =$

$= \int_{-\infty}^{\infty} G_n(f)\, df = \int_{-\infty}^{\infty} G_w(f)\, |H(f)|^2 df$

$= \frac{N_0}{2} \int_{-\infty}^{\infty} |h(t)|^2 dt = \frac{N_0}{2} \frac{3}{T^2} \int_0^T t^2 dt = \frac{N_0}{2} T$

so $p(y \mid \text{"1"}) \sim N\left(\frac{\sqrt{3}}{2} A.T, \frac{N_0 T}{2}\right)$, $p(y \mid \text{"0"}) \sim N\left(-\frac{\sqrt{3}}{2} A.T, \frac{N_0 T}{2}\right)$



By symmetry $\lambda = 0$

$\because p(\text{"1"}) = p(\text{"0"}) = \frac{1}{2}$, & By symmetry $p(err \mid \text{"1"}) = p(err \mid \text{"0"})$

so $p(error) = p(err \mid 1)\, p(1) + p(err \mid 0)\, p(0)$

$= p(err \mid 0) = p(y > \overset{\lambda}{0} \mid 0) = p\left(z > \frac{0 + \frac{\sqrt{3}}{2} AT}{\sqrt{\frac{N_0 T}{2}}}\right) = Q\left(\frac{\sqrt{3}\, A\sqrt{T}}{\sqrt{2}\sqrt{N_0}}\right) = \frac{1}{2} \operatorname{erfc}\left(\frac{\sqrt{3}\, A\sqrt{T}}{2} \cdot \frac{1}{\sqrt{N_0}}\right)$

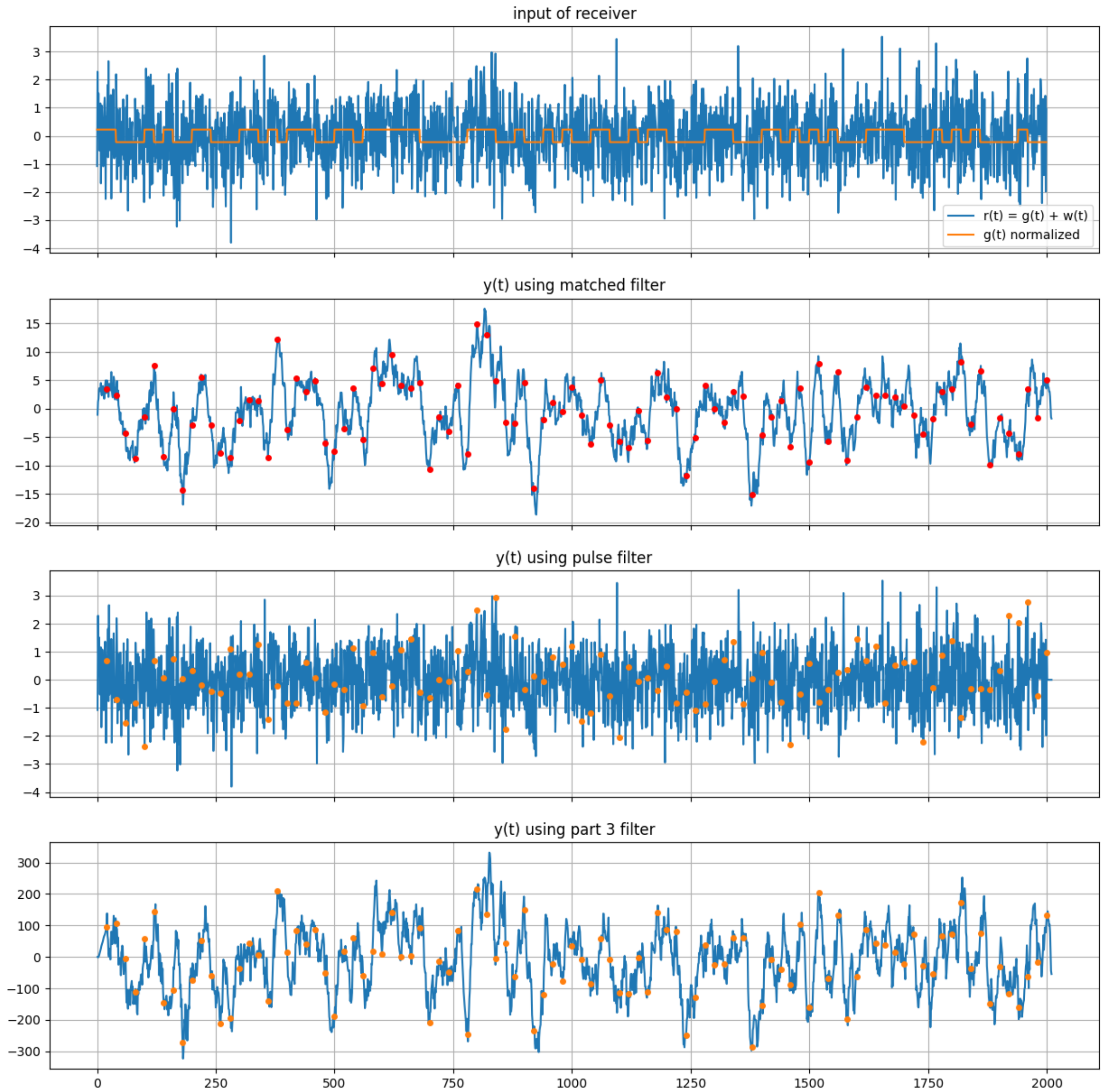
$= \frac{1}{2} \operatorname{erfc}\left(\frac{E}{\sqrt{T N_0}}\right)$

Final Answer (written in latex ❤)

$$\text{Matched Filter}$$
$$P(\text{error}) = 0.5 * \text{erfc}(A\sqrt{\frac{T}{N_o}}) = 0.5 * \text{erfc}(\sqrt{\frac{E}{N_o}})$$
$$E = A^2 * T$$
$$\text{No Filter}$$
$$P(\text{error}) = 0.5 * \text{erfc}(\frac{A}{\sqrt{N_o}}) = 0.5 * \text{erfc}(\frac{E}{\sqrt{N_o}})$$
$$E = A$$
$$\text{Ramp Filter}$$
$$P(\text{error}) = 0.5 * \text{erfc}(\frac{\sqrt{3}A}{2}.\sqrt{\frac{T}{N_o}}) = 0.5 * \text{erfc}(\frac{E}{\sqrt{TN_o}})$$
$$E = \frac{\sqrt{3}}{2}.A.T$$

🤖 Simulation

output from filters + comment



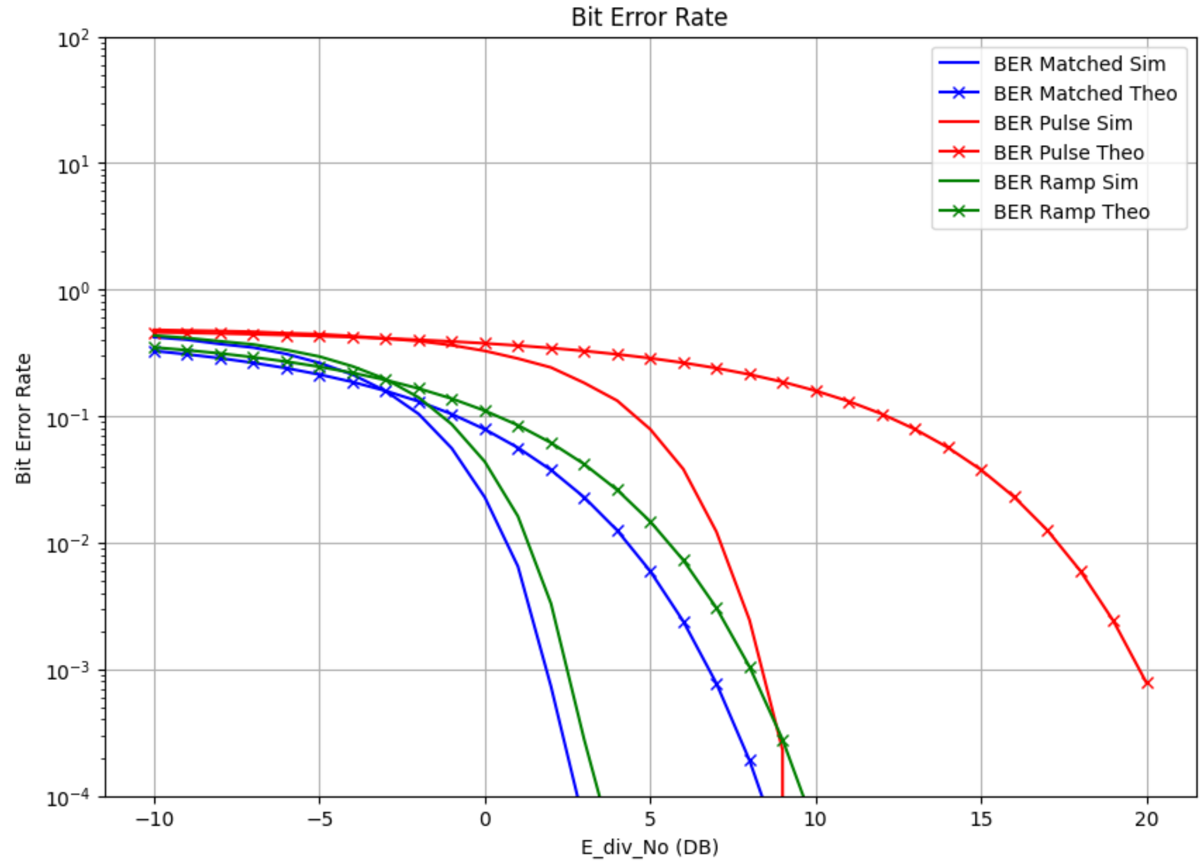
x-axis ⇒ Time * samples per bit

without any needed calculation for BER it's obvious Matched Filter work the best and no Filter (pulse) work worse

BER (Simulation / Comment on problem 😢)

I know I kind of missed up with theoretical calculation due to upscaling I tried to look on energy on how it change the equation during upscaling (still don't what I mess)

the theoretical distribution ≈ the simulated but shifted



Comments on Q5, Q6

Q5) Is the BER an increasing or a decreasing function of *E/No*? Why?

answer simply ⇒ BER is **decreasing** function of E/N_o

as we increase E/N_o we increase transmitted signal energy related to noise energy

if energy of noise close to transmitted signal energy ⇒ $E/N_o \approx 1$ ⇒ noise will destroy signal easy

if energy of noise smaller to transmitted signal energy ⇒ signal won't feal noise

Q6) Which case has the lowest BER? Why?

answer simply ⇒ **Matched Filter**

as matched filter multiply high values in signal by high values in filter than in thresholding
step then you can easily detect it is high value same thing done in low values

but other filters don't multiply signal values with suitable values.

😎 Code

Generation Functions

functions that generate different type of filters, AWGN

```
def generate_match_filter(g1_t: np.ndarray, num_bits = num_bits, num_
    # g1_t : impulse shape when sending "1" must be in shape (num_sa
    h_t = g1_t[::-1] # just reverse it
    h_t = np.concatenate([h_t, np.zeros((num_bits - 1) * num_samples_
    return h_t

def generate_pulse_filter(num_bits = num_bits, num_samples_per_bit = 
    h_t = np.zeros((num_bits * num_samples_per_bit)) # just reverse 
    h_t[0] = 1
    return h_t

def generate_filter_p3(num_bits = num_bits, num_samples_per_bit = nu
    h_t = np.arange(0, num_samples_per_bit, dtype=np.float64)
    h_t *= np.sqrt(3)
    h_t = np.concatenate([h_t, np.zeros((num_bits - 1) * num_samples_
    return h_t



def generate_AWGN(No: int, num_bits = num_bits, num_samples_per_bit 
    noise = np.random.normal(0, No/2, size=(num_bits * num_samples_p
    return noise
```

Functions (More done)

```
# channel jop
def generate_random_signal(num_bits,  No = 2, num_samples_per_bit = 
    data =  np.random.choice([0, 1], size=(num_bits), p=[1./2, 1./2]
    g_t = generate_bipolar_NRT_sig(data, num_samples_per_bit)
    w_t = generate_AWGN(No, num_bits, num_samples_per_bit)

    return data, g_t, w_t

# receiver jop
def get_reciver_ouput(r_t, filter_type = "match", num_samples_per_bi

    h_t = None

    if filter_type == "match":
        h_t = generate_match_filter(g1_t, num_bits, num_samples_per_
```

```
    elif filter_type == "pulse":
        h_t = generate_pulse_filter(num_bits, num_samples_per_bit)
    else:
        h_t = generate_filter_p3(num_bits, num_samples_per_bit)

    y_t = convolve(r_t, h_t)


    return h_t, y_t
```

Drawing output of each Matched Filter

1. **calculation of random signal and output**

```
data, g_t, w_t = generate_random_signal(num_bits, No = 2)
r_t = g_t + w_t
h_t_match, y_t_match = get_reciver_ouput(r_t, "match")
h_t_pulse, y_t_pulse = get_reciver_ouput(r_t, "pulse")
h_t_p3, y_t_p3 = get_reciver_ouput(r_t, "p3")
```

2. **sampling**

```
clock = (np.arange(num_bits) + 1) * num_samples_per_bit
y_t_match_sampled = y_t_match[clock-1]
y_t_pulse_sampled = y_t_pulse[clock-1]
y_t_p3_sampled = y_t_p3[clock-1]
```

3. **plotting**

```
fig, axs = plt.subplots(4, 1, figsize=(15, 15), sharex=True)
show_bound = len(g_t) + 10



axs[0].set_title("input of receiver")
axs[0].plot(r_t, label="r(t) = g(t) + w(t)")
axs[0].plot(g_t, label="g(t) normalized")
axs[0].grid(True)
axs[0].legend()

axs[1].set_title("y(t) using matched filter")
axs[1].plot(np.arange(1, show_bound + 1), y_t_match[:show_bound]) #
axs[1].plot(clock, y_t_match_sampled, "ro", markersize=4)
```

```
axs[1].grid(True)


axs[2].set_title("y(t) using pulse filter")
axs[2].plot(np.arange(1, show_bound + 1), y_t_pulse[:show_bound] ) #
axs[2].plot(clock, y_t_pulse_sampled, "o", markersize=4)
axs[2].grid(True)


axs[3].set_title("y(t) using part 3 filter")
axs[3].plot(np.arange(1, show_bound + 1), y_t_p3[:show_bound]) # con
axs[3].plot(clock, y_t_p3_sampled, "o", markersize=4)
axs[3].grid(True)

plt.show()
```

BER Calculation & Plotting

Functions

```
def calc_sim_BER(data, sampled_received):
    error_prob = np.sum(data != sampled_received)
    error_prob /= data.shape[0]
    return error_prob

def calc_theo_BER(A, No, T, filter_type="match"):
    if filter_type == "match":
        return 0.5 * erfc(  A * np.sqrt(T / No ) )
    elif filter_type == "pulse":
        return 0.5 * erfc(  A / np.sqrt(No )  )
    else:
        return 0.5 * erfc( (np.sqrt(3) / 2) * A * ( np.sqrt(T / No))
```

Calculations

```
num_bits = 10**5

BER_simulated_match = []
BER_theortical_match = []

BER_simulated_pulse = []
BER_theortical_pulse = []
```

```
BER_simulated_p3 = []
BER_theortical_p3 = []

E = 1

data =  np.random.choice([0, 1], size=(num_bits), p=[1./2, 1./2])
g_t = generate_bipolar_NRT_sig(data)
AA = 1 / g1_t_norm            # amplitude after upsampling normaliza
TT = num_samples_per_bit      # sampling time after upsampling

for E_div_No_db in range(-10, 21):
    E_div_No = 10 ** (E_div_No_db/10)
    No = E / E_div_No

    w_t = generate_AWGN(No, num_bits, num_samples_per_bit)
    r_t = g_t + w_t

    h_t_match, y_t_match = get_reciver_ouput(r_t, "match", num_sampl
    h_t_pulse, y_t_pulse = get_reciver_ouput(r_t, "pulse", num_sampl
    h_t_p3, y_t_p3 = get_reciver_ouput(r_t, "p3", num_samples_per_bi

    # sampling
    clock = (np.arange(num_bits) + 1) * num_samples_per_bit
    y_t_match_sampled = (y_t_match[clock-1] > 0).astype(int)
    y_t_pulse_sampled = (y_t_pulse[clock-1] > 0).astype(int)
    y_t_p3_sampled = (y_t_p3[clock-1] > 0).astype(int)


    # print(f"========= for E/No = {E_div_No_db} =======")
    # print(y_t_match_sampled, calc_sim_BER(data, y_t_match_sampled)
    # print(y_t_pulse_sampled, calc_sim_BER(data, y_t_pulse_sampled)
    # print(y_t_p3_sampled, calc_sim_BER(data, y_t_p3_sampled), calc_
    BER_simulated_match.append(calc_sim_BER(data, y_t_match_sampled)
    BER_theortical_match.append(calc_theo_BER(AA, No, TT, "match"))


    BER_simulated_pulse.append(calc_sim_BER(data, y_t_pulse_sampled)
    BER_theortical_pulse.append(calc_theo_BER(AA, No, TT, "pulse"))

    BER_simulated_p3.append(calc_sim_BER(data, y_t_p3_sampled))
    BER_theortical_p3.append(calc_theo_BER(AA, No, TT,  "p3"))
```

Plotting

```
plt.figure(figsize=(10, 7))
plt.plot(range(-10, 21), BER_simulated_match, 'b', label="BER Matche
plt.plot(range(-10, 21), BER_theortical_match, 'x-b', label="BER Mat


plt.plot(range(-10, 21), BER_simulated_pulse, 'r',  label="BER Pulse
plt.plot(range(-10, 21), BER_theortical_pulse, 'x-r', label="BER Pul

plt.plot(range(-10, 21), BER_simulated_p3, 'g',  label="BER Ramp Sim
plt.plot(range(-10, 21), BER_theortical_p3, 'x-g', label="BER Ramp T

plt.xlabel('E_div_No (DB)')
plt.ylabel('Bit Error Rate')
plt.yscale('log')
plt.ylim(10**(-4), 10**2)
plt.title('Bit Error Rate')
plt.legend()
plt.grid()
plt.show()
```