Cairo University
Faculty of Engineering

Department of Computer Engineering

ELC 325B – Spring 2023

Digital Communications

Assignment #2

Submitted to

Dr. Mai

Dr. Hala

Eng. Mohamed Khaled

Submitted by

Name	Sec	BN	Code
Abdelrahman Hamdy Ahmed	1	36	9202833
Zeyad Tarek Khairy	1	28	9202588

Contents

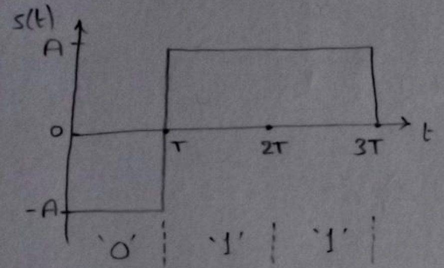
Hand Analysis – Part 1

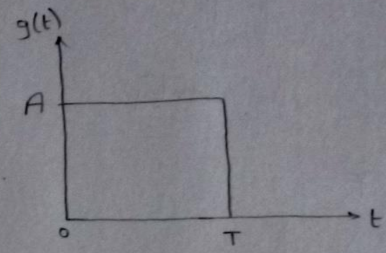
Part 1 DC Assignment 2

'1' & '0' are represented by a positive and a negative pulse

a) Transmitted baseband waveform $s(t)$

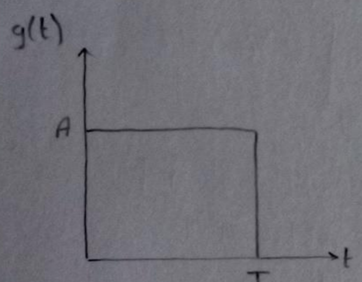
Bit Sequence $\Rightarrow$ $b_0 = $ '0', $b_1 = $ '1', $b_2 = $ '1'

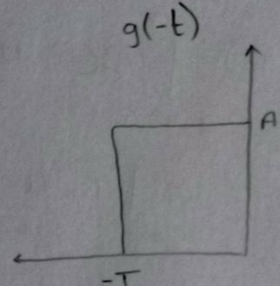


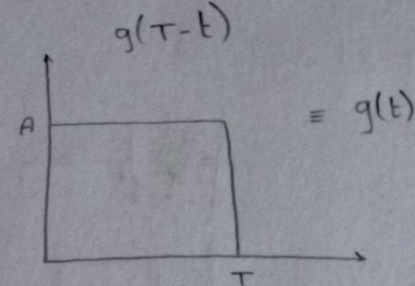


b) Matched Filter output due to signal only (Ignore noise)

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







$\equiv g(t)$

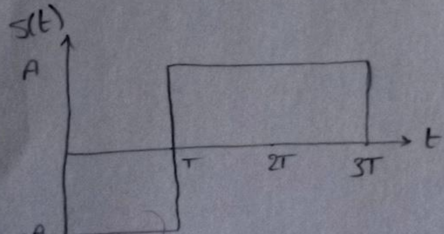
$r(t) = s(t) + \cancel{w(t)}$ $\quad 0 \le t \le T$ $\quad$ [Input]

$\hookrightarrow$ Ignored $\rightarrow r(t) = s(t)$

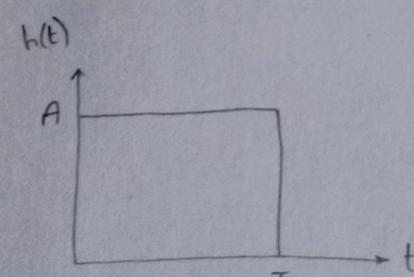
The Filter Output $y(t) = r(t) * h(t)$

$y(t) = \underbrace{s(t) * h(t)}_{g_o(t)} \rightarrow g_o(t) = s(t) * h(t) \rightarrow s(t) * g(T-t)$

- Our goal is to apply convolution to get the matched filter output



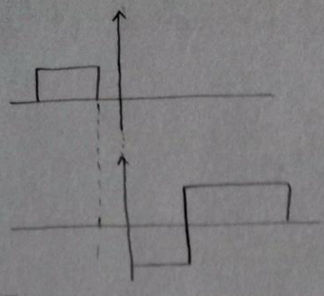
$\circledast$



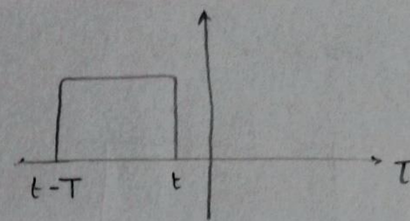
By considering all regions :

$$y(t) = r(t) * h(t) = \int_{-\infty}^{\infty} r(\tau)h(t-\tau)\, d\tau$$

1 $t < 0$

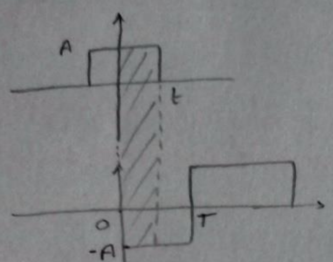


$h(t-\tau)$



$\boxed{y(t) = 0}$ No overlap

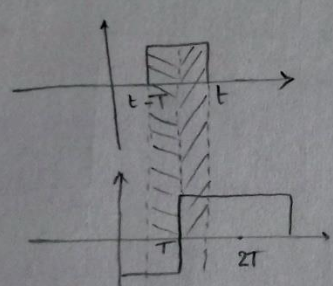
2 $0 < t < T$



$$y(t) = \int_{0}^{t} (A)(-A)\, d\tau$$

$$= -A^2 [\tau]_0^t = \boxed{-A^2 t}$$

3 $T < t < 2T$



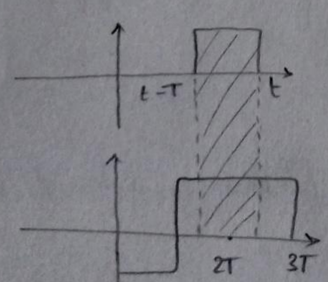
$$y(t) = \int_{t-T}^{T} (A)(-A)\, d\tau + \int_{T}^{t} (A)(A)\, d\tau$$

$$= -A^2 [T - (t-T)] + A^2 [t - T]$$

$$= A^2 t - 2A^2 T + A^2 t - A^2 T$$

$$= \boxed{A^2 [2t - 3T]}$$

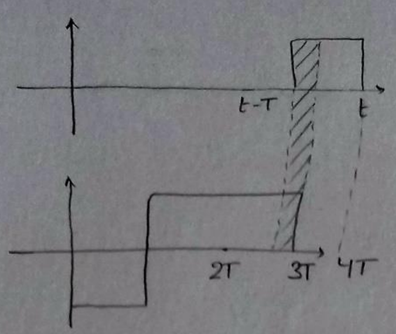
4 $2T < t < 3T$



$$y(t) = \int_{t-T}^{t} (A)(A)\, d\tau$$

$$= A^2 [t - (t-T)] = \boxed{A^2 T}$$

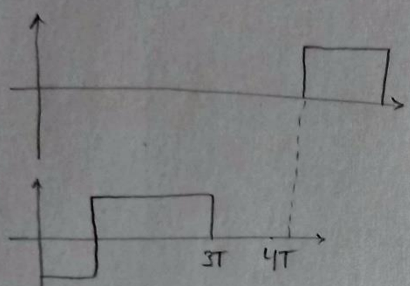
5 $3T < t < 4T$



$$y(t) = \int_{t-T}^{3T} (A)(A)\,d\tau + \int_{3T}^{t} \cancel{(A)(0)\,d\tau} \rightarrow 0$$

$$y(t) = A^2\,[\,3T - (t-T)\,]$$
$$= \boxed{A^2\,[\,4T - t\,]}$$

6 $t > 4T$



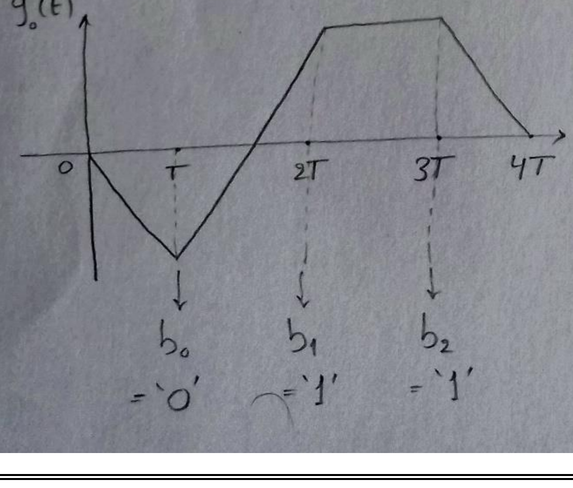
$\boxed{y(t) = 0}$

No overlap

- Final Output

$$y(t) = [g_o(t)] = \begin{cases} 0 & , \quad t < 0 \\ -A^2 t & , \quad 0 \le t < T \\ A^2[2t - 3T] & , \quad T \le t < 2T \\ A^2 T & , \quad 2T \le t < 3T \\ A^2[4T - t] & , \quad 3T \le t < 4T \\ 0 & , \quad t \geqslant 4T \end{cases}$$

t	0	T	$2T$	$3T$	$4T$
$y(t)$	0	$-A^2T$	A^2T	A^2T	0



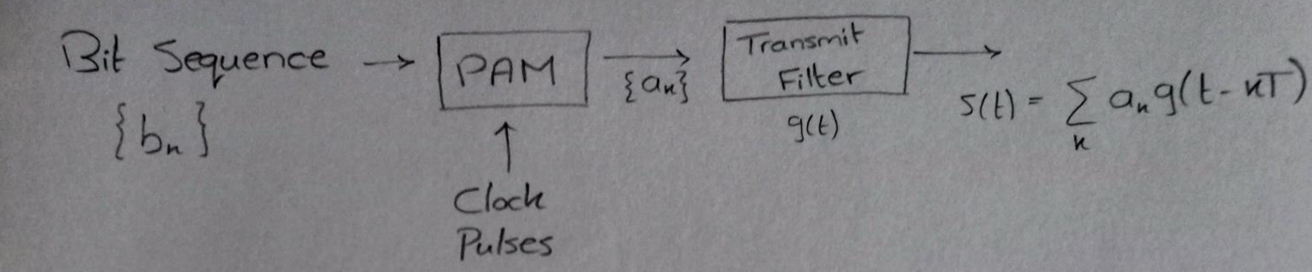
c) Mark the Sampling instants

b_0 @ $t = T$ -ve Amplitude

b_1 @ $t = 2T$ +ve Amplitude

b_2 @ $t = 3T$ +ve Amplitude

d) Transmitter Block Diagram



$\{b_n\}$: The input is binary bits, each of duration T_b

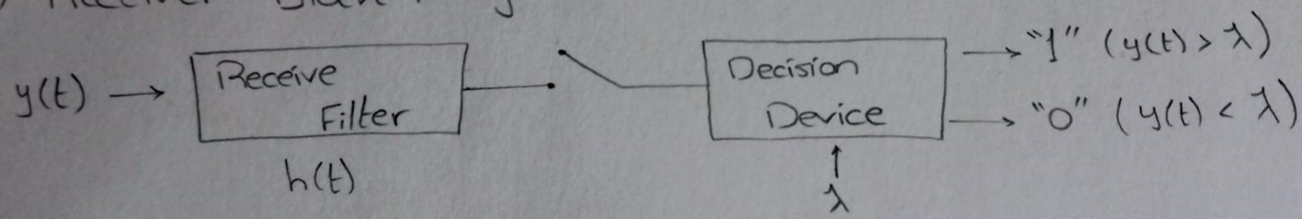
$\{a_n\}$: Sequence of amplitude - modulated short pulses

binary PAM $\rightarrow$ $a_n = \pm 1$

$s(t)$: Sequence of pulse shaped symbols

PAM : Pulse Amplitude Modulator

e) Receiver Block Diagram



$y(t)$: Output of the receiver Filter. It is sampled at $t_i = iT_b$ Synchronously with the transmitter

The Decision device finally decides, based on a threshold λ, whether the sample is "1" or "0"

$$y(t) = s(t) + \underbrace{w(t)}_{\text{AWGN by the channel}}$$

AWGN : Additive White Gaussian Noise

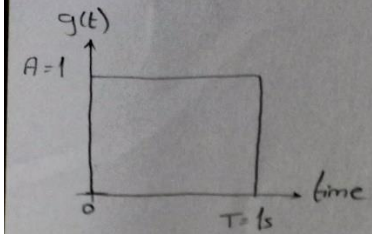
Hand Analysis – Part 2

Part 2

Binary Source → Pulse Shape → channel → Receive Filter $h(t)$ → Sample at T → Decode to '0' & '1'

'0' & '1'

↑ AWGN

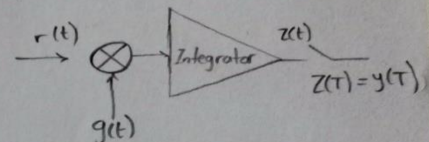


$1 \rightarrow g(t)$

$0 \rightarrow -g(t)$

- Ideal channel
[Impulse response → $\delta(t)$]
- Noise is AWGN with zero mean and $\frac{N_0}{2}$ variance

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



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

$$y(t) = r(t) * h(t) = \int_{-\infty}^{\infty} r(\tau) h(t-\tau)\, d\tau$$

$$y(T) = \int_{-\infty}^{\infty} r(\tau) h(T-\tau)\, d\tau = \int_{-\infty}^{\infty} r(\tau) g(\tau)\, d\tau$$

$\lambda \rightarrow$ Threshold

- Polar NRZ Signaling

$$r(t) = \begin{cases} +A + w(t) & \text{for bit '1'} \quad 0 \le t \le T_b \\ -A + w(t) & \text{for bit '0'} \quad 0 \le t \le T_b \end{cases}$$

$$\Rightarrow y(T_b) = \int_{-\infty}^{\infty} r(\tau) g(\tau)\, d\tau = \int_{0}^{T_b} kAr(t)\, dt = \int_{0}^{T_b} \frac{r(t)}{T_b}\, dt \qquad kAT_b = 1$$

$$\Rightarrow y = y(T_b) = \pm A + n(t) \qquad \text{where } n(t) = \frac{1}{T_b}\int_{0}^{T_b} w(t)\, d\tau$$

$n(t)$ is Gaussian distributed, with zero mean and variance $\sigma^2 = \frac{1}{T_b}\frac{N_0}{2}$

$y(T_b)$ is Gaussian distributed, with $\pm A$ mean & variance $\sigma^2 = \frac{1}{T_b}\frac{N_0}{2}$

$$P(y|\text{'0'}) = \frac{1}{\sqrt{\pi N_0/T_b}}\, e^{-\frac{(y+A)^2}{N_0/T_b}} \qquad P(e|\text{'0'}) = P\{y > \lambda | \text{'0'}\} = \int_{\lambda}^{\infty} p(y|\text{'0'})\, dy$$

$$P(y|\text{'1'}) = \frac{1}{\sqrt{\pi N_0/T_b}}\, e^{-\frac{(y-A)^2}{N_0/T_b}} \qquad P(e|\text{'1'}) = P\{y < \lambda | \text{'1'}\} = \int_{-\infty}^{\lambda} p(y|\text{'1'})\, dy$$

Probability of Error if '0' was transmitted

$$P(e|'0') = \frac{1}{\sqrt{\pi N_0/T_b}} \int_{\lambda}^{\infty} e^{-\frac{(y+A)^2}{N_0/T_b}} dy$$

$$= \frac{1}{\sqrt{\pi}} \int_{\frac{\lambda+A}{\sqrt{N_0/T_b}}}^{\infty} e^{-z^2} dz \quad \Leftarrow \quad z = \frac{y+A}{\sqrt{N_0/T_b}}$$

$$= \boxed{\frac{1}{2} \operatorname{erfc}\left(\frac{\lambda+A}{\sqrt{N_0/T_b}}\right)}$$

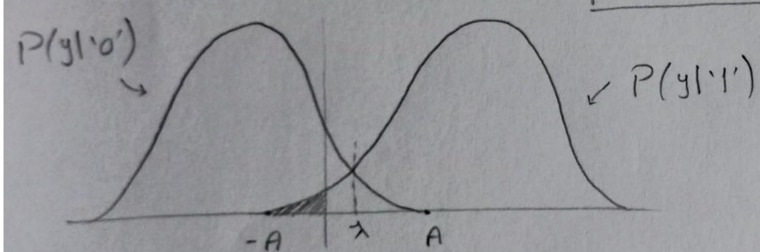
- $\operatorname{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_{x}^{\infty} e^{-t^2} dt$
- $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_{0}^{x} e^{-t^2} dt$
- $Q(x) = \frac{1}{\sqrt{2\pi}} \int_{x}^{\infty} e^{-\frac{t^2}{2}} dt$

Probability of Error if '1' was transmitted

$$P(e|'1') = \frac{1}{\sqrt{\pi N_0/T_b}} \int_{-\infty}^{\lambda} e^{-\frac{(y-A)^2}{N_0/T_b}} dy$$

$$= \frac{1}{\sqrt{\pi}} \int_{\frac{-\lambda+A}{\sqrt{N_0/T_b}}}^{\infty} e^{-z^2} dz \quad \Leftarrow \quad z = -\frac{y-A}{\sqrt{N_0/T_b}}$$

$$= \boxed{\frac{1}{2} \operatorname{erfc}\left(\frac{-\lambda+A}{\sqrt{N_0/T_b}}\right)}$$



$\lambda = 0$ ↳ optimal

Average Error Probability $P(e) = P(e|'0')P('0') + P(e|'1')P('1') = f(\lambda)$

In order to minimize the average error probability, λ should be optimally chosen

$$\lambda_{opt} = \frac{N_0}{4AT_b} \ln\left(\frac{P('0')}{P('1')}\right)$$

<u>Special Case</u> If $P('0') = P('1') = 0.5$, then $\lambda_{opt} = 0$

$$P(e) = P(e|'0') = P(e|'1') = \frac{1}{2}\operatorname{erfc}\left(\frac{A}{\sqrt{N_0/T_b}}\right) = \boxed{\frac{1}{2}\operatorname{erfc}\left(\sqrt{\frac{E_b}{N_0}}\right)}$$

✓ $A = 1$

✓ $E_b = 1$

Case b) $h(t)$ is not existent (i.e $h(t) = \delta(t)$)

$$y(t) = r(t) * \underbrace{h(t)}_{\delta(t)} \rightarrow y(t) = r(t) * \delta(t) = r(t)$$

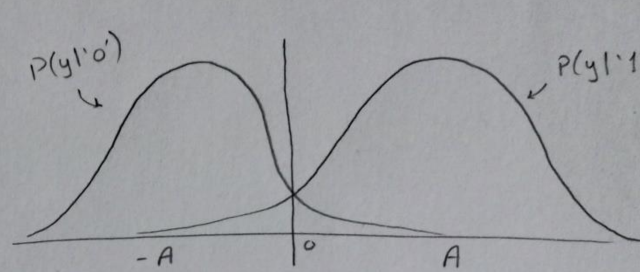
$$y(t) = r(t) = g(t) + \omega(t) \Rightarrow y(T) = g(T) + \omega(T) = \pm A + \omega(T)$$

Mean of $y(T) \rightarrow E[y(T)] = \pm A + \underbrace{E[\cancel{\omega(T)}]}_{\rightarrow 0} = \pm A$

Variance of $y(T) \rightarrow \sigma^2(y(T)) = \sigma^2(\omega(T)) = \frac{N_0}{2}$

Probability of error follows a normal distribution

$$P(y|`0') \sim N(-A, \tfrac{N_0}{2})$$
$$P(y|`1') \sim N(A, \tfrac{N_0}{2})$$



$$P(e|`0') = \int_0^\infty P(y|`0')\,dy = \int_{\frac{A}{\sqrt{\frac{N_0}{2}}}}^\infty f(z)\,dz$$

$\lambda = 0 \rightarrow$ Symmetry

$Z \sim N(0,1)$

$$= Q\left(\frac{A}{\sqrt{\frac{N_0}{2}}}\right)$$

We expect $p(e|`0')$ to be the same as $P(e|`1')$ due to symmetry

$$P(e|`1') = \int_{-\infty}^{0} P(y|`1')\,dy = \int_{-\infty}^{-\frac{A}{\sqrt{\frac{N_0}{2}}}} f(z)\,dz = Q\left(\frac{A}{\sqrt{\frac{N_0}{2}}}\right)$$

By assuming $P(`1') = P(`0') = 0.5$

$$P(e) = P(e|`1')P(`1') + P(e|`0')P(`0')$$

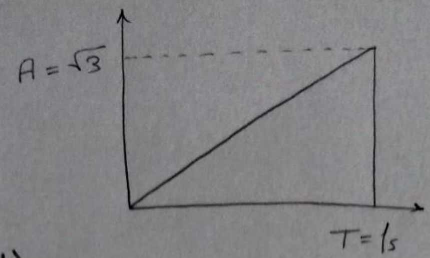
$P(e|`1') = P(e|`0')$

$$= Q\left(\frac{A}{\sqrt{\frac{N_0}{2}}}\right)[0.5 + 0.5] = \boxed{Q\left(\frac{A}{\sqrt{\frac{N_0}{2}}}\right)}$$

$$\Rightarrow \frac{1}{2}\,\text{erfc}\left(\frac{A}{\sqrt{N_0}}\right)$$

Convert from $Q \rightarrow$ erfc
Multiply by $\frac{1}{2}$ & divide by $\sqrt{2}$ inside the paranthesis

Case c) $h(t)$ has the following response



$r(t) = g(t) + w(t)$

The filter output $\rightarrow$ $y(t) = r(t) * h(t)$

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

$$y(t) = g_o(t) + n(t)$$

$$g_o(t) = g(t) * h(t) = \int_{-\infty}^{\infty} g(\tau)\, h(t-\tau)\, d\tau$$

$$g_o(T) = \int_0^T g(\tau)\, h(T-\tau)\, d\tau = \pm\sqrt{3}A \int_0^T (T-\tau)\, d\tau$$

$$g_o(T) = \pm\sqrt{3}A\left[T\tau - \frac{\tau^2}{2}\right]_0^T = \pm\sqrt{3}A\left[T^2 - \frac{T^2}{2}\right] = \pm\frac{\sqrt{3}}{2}AT^2$$

By linearity $\rightarrow$ The expectation of the random gaussian noise convolved with the filter $h(t)$ is zero $\rightarrow$ $E[n(t)] = 0$

Variance $\rightarrow$ $var[n(t)] = E[n(t)^2] - \underbrace{E[n(t)]^2}_{\rightarrow 0} = E[n(t)^2]$

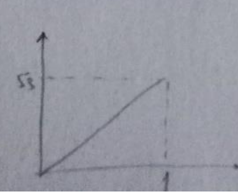
$$E[n(T)^2] = \int_{-\infty}^{\infty} S_N(y)\, dy \qquad n(t) = w(t) * h(t) \quad \text{Recall}$$

$$\cdot\, S_N(y) = S_w(y)\,|H(y)|^2 \qquad S_w(y) = \frac{N_o}{2} \quad (\text{Known})$$

$$E[n(T)^2] = \int_{-\infty}^{\infty} \frac{N_o}{2}\,|H(y)|^2\, dy = \frac{N_o}{2}\int_{-\infty}^{\infty} |H(y)|^2\, dy$$

$$= \frac{N_o}{2}\int_{-\infty}^{\infty} |h(t)|^2\, dt = \frac{N_o}{2}\int_0^T 3t^2\, dt = \frac{N_o}{2}\left[\frac{3t^3}{3}\right]_0^T$$

$$= \boxed{\frac{N_o}{2}T^3} \qquad h(t) = \sqrt{3}t$$



Assuming $P(`0') = P(`1') = 0.5$

$$P(y|`0') \sim N\left(\frac{-\sqrt{3}}{2} AT^2, \frac{N_0}{2} T^3\right)$$

$$P(y|`1') \sim N\left(\frac{\sqrt{3}}{2} AT^2, \frac{N_0}{2} T^3\right)$$

Using the same approach as the previous case

$$P(e|`0') = P(e|`1') = Q\left(\frac{\frac{\sqrt{3}}{2} AT^2}{\sqrt{\frac{N_0}{2} T^3}}\right)$$

$$= Q\left(\frac{\sqrt{3}\, AT^2}{\frac{2\sqrt{N_0 T^3}}{\sqrt{2}}}\right) = Q\left(\frac{\sqrt{3}\, AT^2}{\sqrt{2}\sqrt{N_0 T^3}}\right)$$

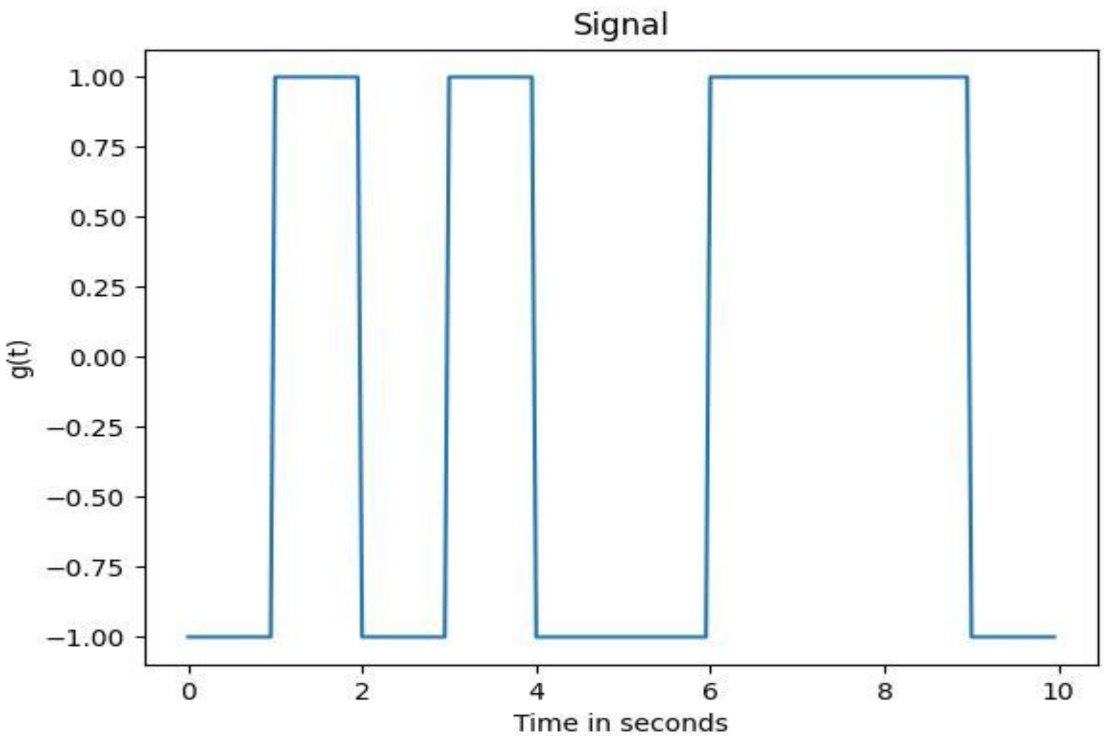
$$P(e) = P(e|`0')P(`0') + P(e|`1')P(`1')$$

$$= \boxed{Q\left(\frac{\sqrt{3}\, AT^2}{\sqrt{2 N_0 T^3}}\right)}$$

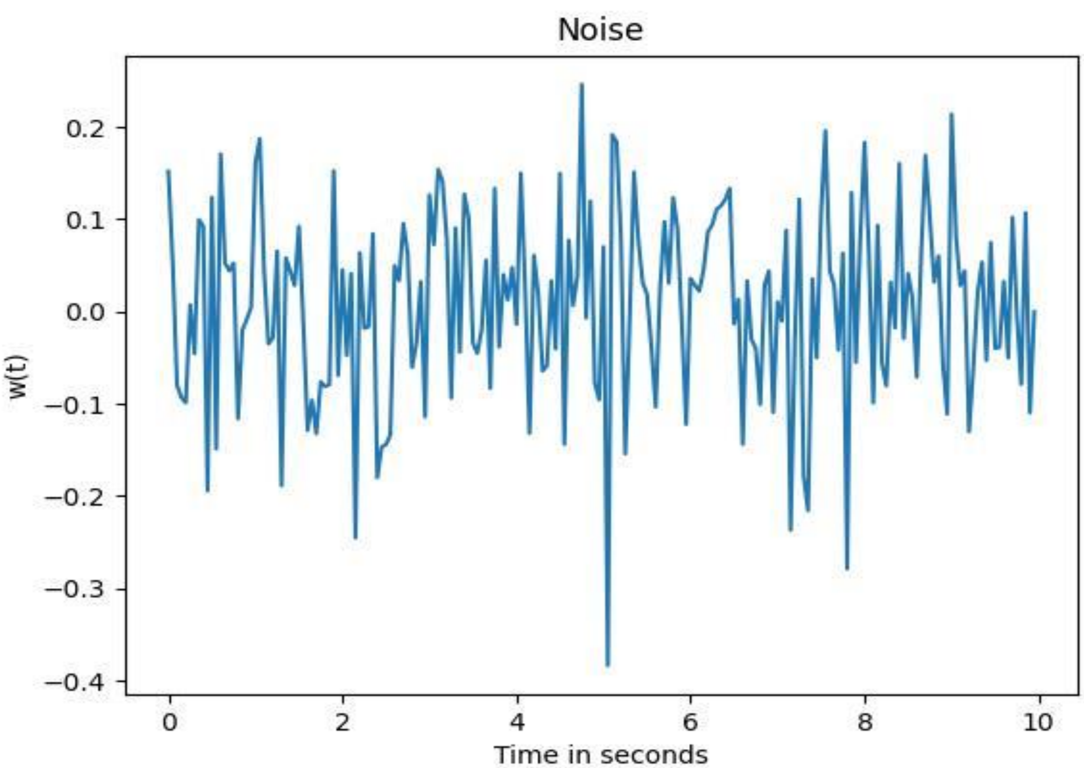
$$\Rightarrow \frac{1}{2} \operatorname{erfc}\left(\frac{\sqrt{3}\, AT^2}{2\sqrt{N_0 T^3}}\right)$$

$$= \frac{1}{2} \operatorname{erfc}\left(\frac{\sqrt{3}\, \frac{A}{2}\, T^2}{\sqrt{N_0}\, T^{1.5}}\right) = \boxed{\frac{1}{2} \operatorname{erfc}\left(\frac{A}{2} \cdot \sqrt{\frac{3T}{N_0}}\right)}$$

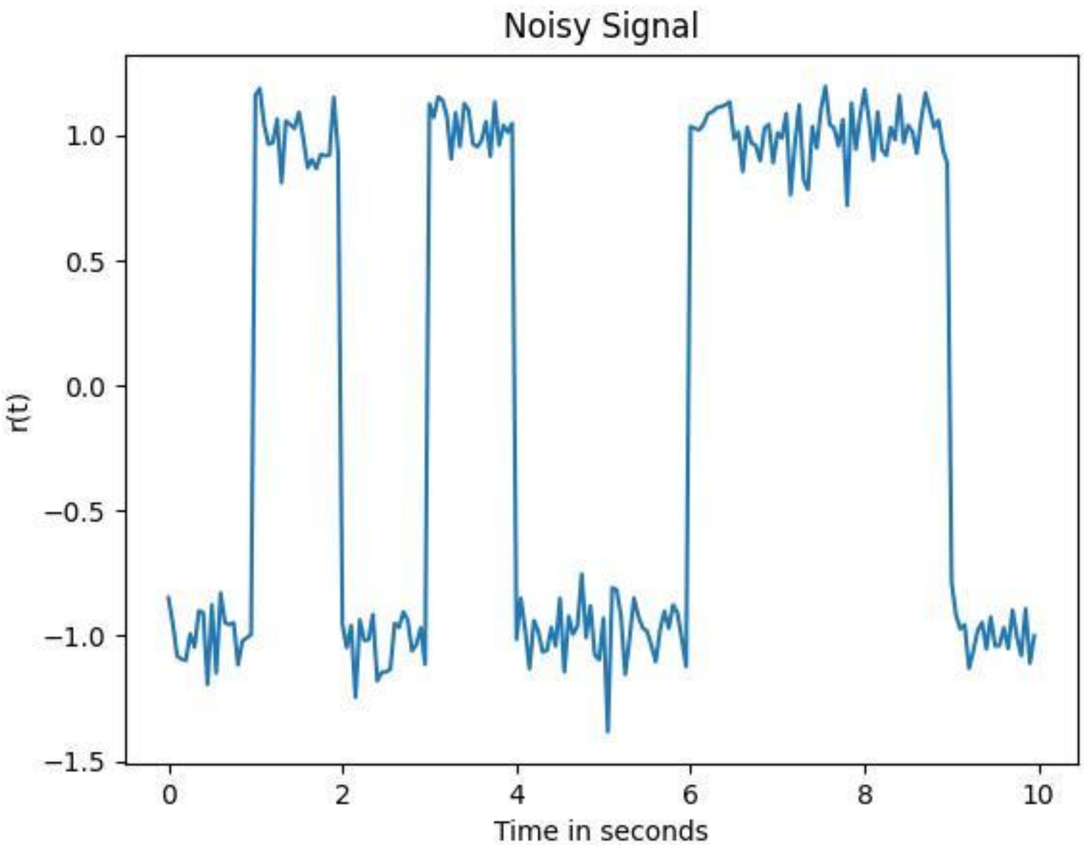
Signal



Noise



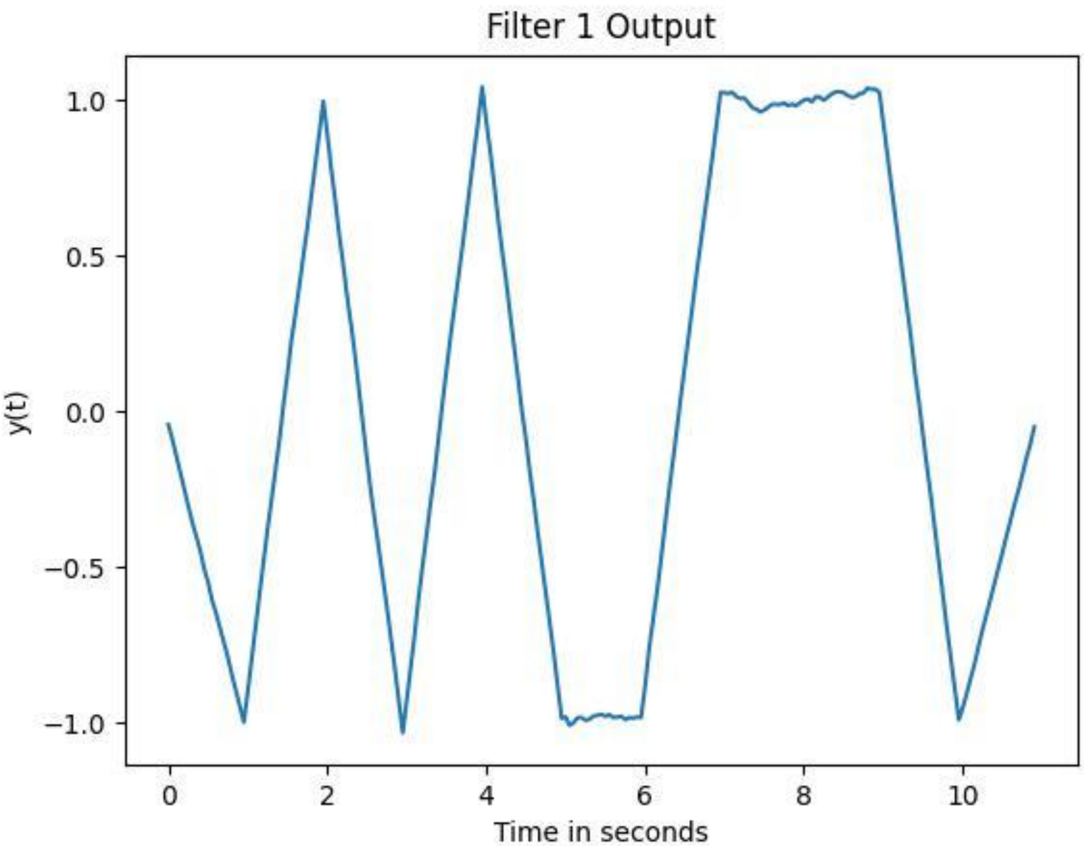
Noisy Signal



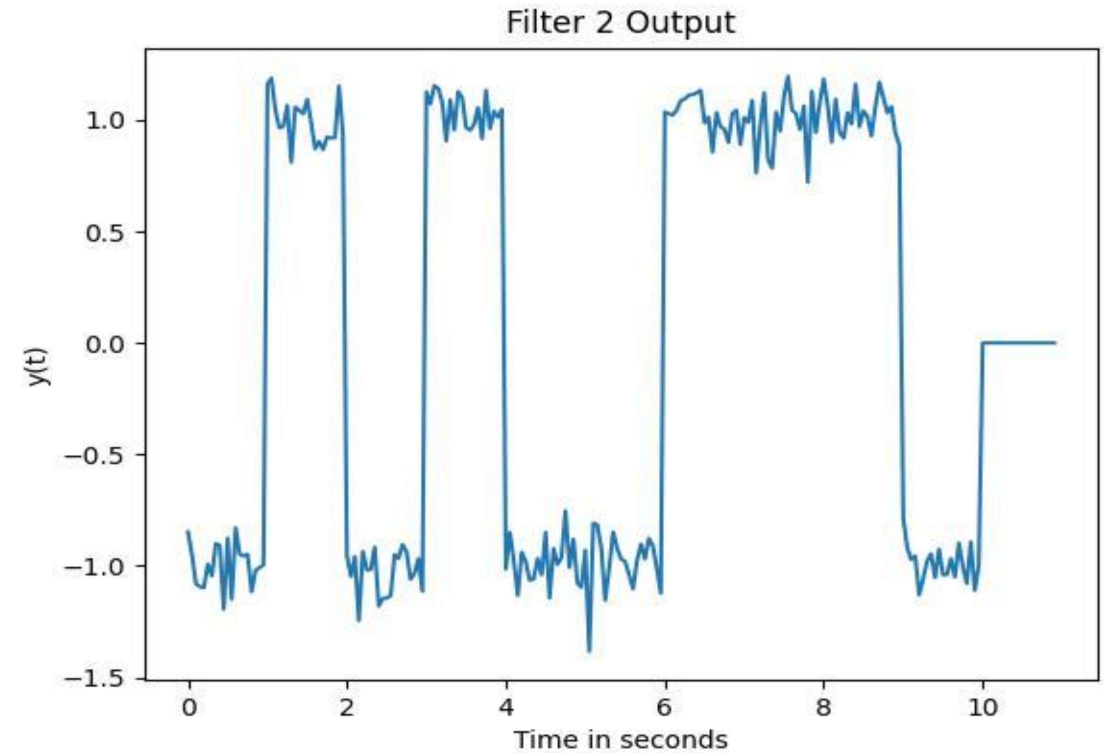
Filter outputs

Number of bits = 10 and one pulse of duration 1 has 20 samples

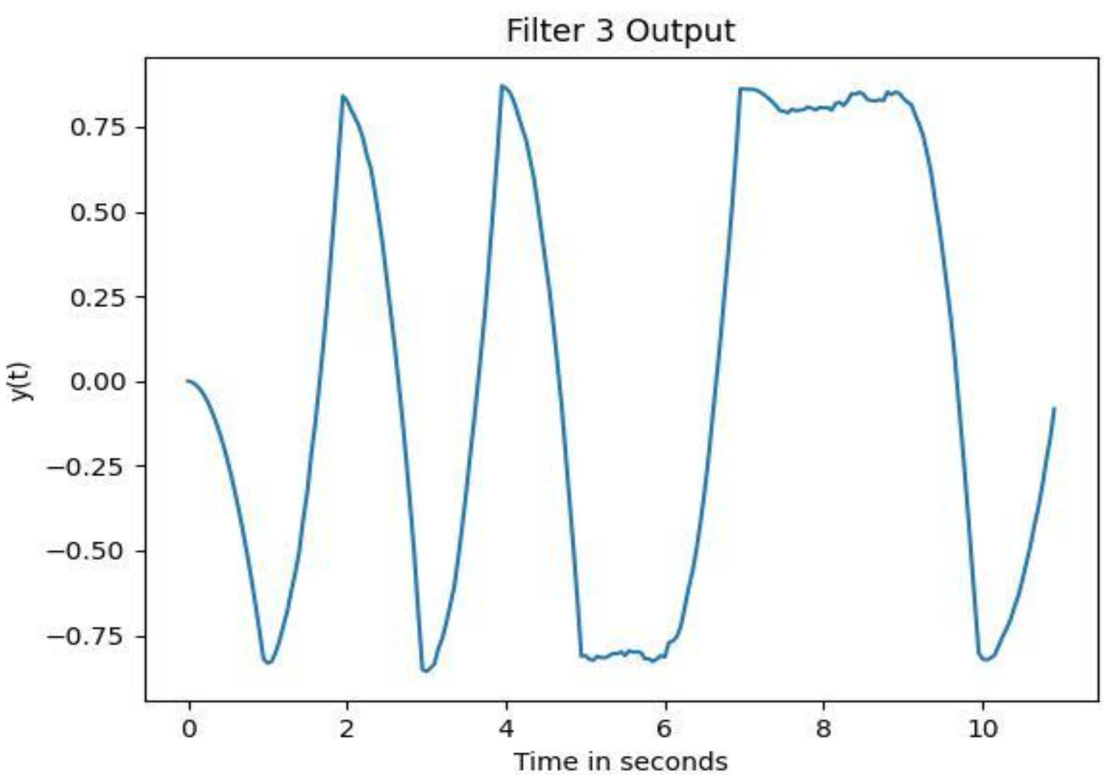
Filter 1 output



Filter 2 output

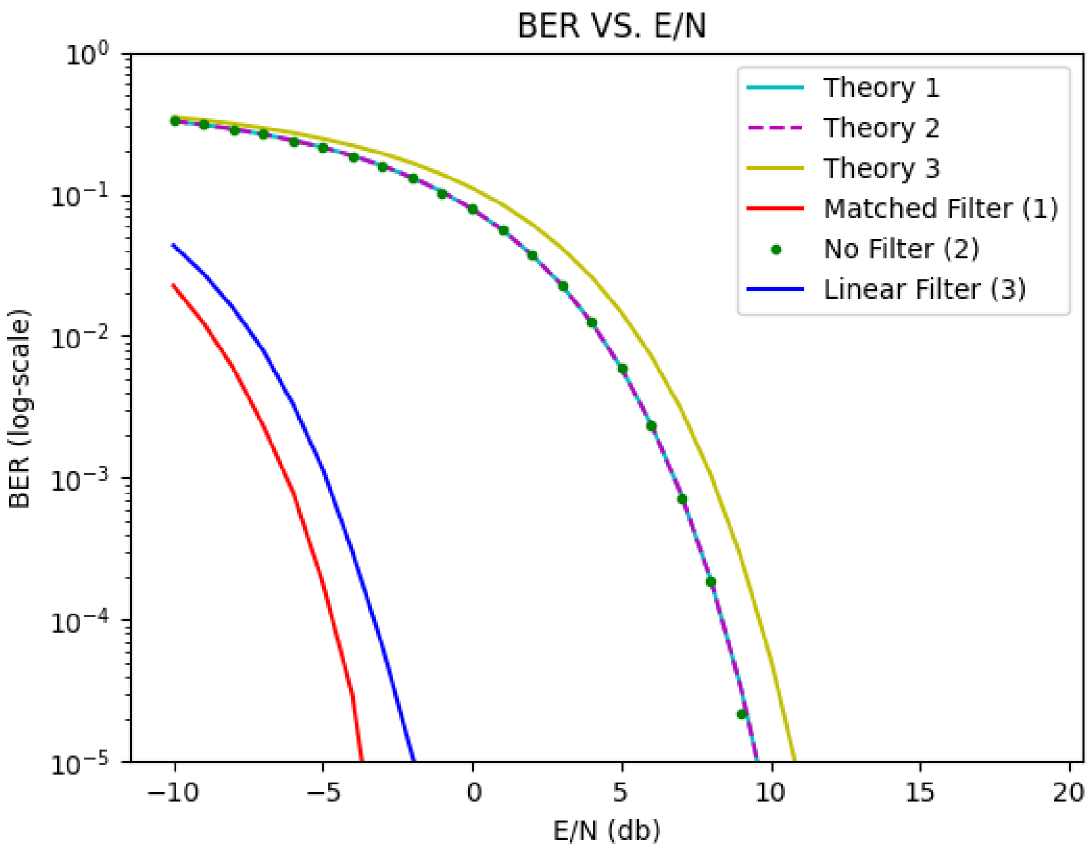


Filter 3 output



Theoretical and simulated Bit Error Rate (BER) Vs E/No:

Number of bits = 1000000 and one pulse of duration 1 has 20 samples




```
"""
The Bit Error Rate (BER) decreases as the energy per bit (E) relative to the noise power
spectral density (No) increases, which is evident in the plot. This can be explained in
various ways:

Using the theoretical expression for BER and noting that the Q function is a decreasing
function, it is apparent that Q(a * sqrt(E/No)) for all the cases mentioned in the problem.
Therefore, as sqrt is an increasing function, it can be concluded that BER is a decreasing
function of E/No.
"""

"""
The case that uses a matched filter has the lowest BER because it employs a filter that is
specifically designed to minimize the probability of error by matching the filter to the
pulse. By doing so, the matched filter maximizes the peak pulse SNR at the sampling instant,
which helps reduce the probability of error.

It is important to note that in the theoretical case, using a filter or not yields the same
expression due to our assumptions on variance and PSD.
"""
```

```python
import numpy as np
from math import erfc, sqrt
import matplotlib.pyplot as plt

STEP = 1.0 / 20
PERIOD = int(1 / STEP)

LAMBDA = 0  # THRESHOLD

SIGMA_NOISE = 0.1

T = 1
A = 1
E = 1


def draw(t, signal, xLabel, yLabel, title):
    """
    This function plots a given signal against the
corresponding time axis, with specified labels for the x-
axis and y-axis, and a title. The plot is displayed and
saved as a JPG file using the provided title.
    """
    plt.plot(t, signal)
    plt.xlabel(xLabel)
    plt.ylabel(yLabel)
    plt.title(title)
    plt.savefig(title + '.jpg')
    plt.show()


def sample(bitstream, signal, n):
    """
    This function samples a given signal at a specified
interval and compares the samples with a threshold value. It
```

```
reconstructs a binary bitstream based on the sample
comparison, calculates the Bit Error Rate (BER), and returns
the reconstructed bitstream.
    """
    samples = np.array([signal[PERIOD - 1 + i * PERIOD] for
i in range(n)])
    result = (samples > LAMBDA)
    print("Reconstructed Bitstram:", result)
    print("Total number of bits:", n)
    print("Received Wrong:", np.sum(bitstream != result))
    print("BER:", np.sum(bitstream != result) / n)
    return result


def generateSignal(bitstream):
    """
    This function generates a signal waveform based on a
given binary bitstream using Pulse Amplitude Modulation
(PAM) with a specified pulse shape. The function applies the
pulse shape along the symbol's interval and returns the
resulting signal.
    """
    # Pulse Amplitude Modulation (PAM)
    P = int(T / STEP)
    pulse = np.ones(P) * A

    pulseLength = len(pulse)
    inputLength = len(bitstream)

    # Generate Signal
    signal = np.zeros(inputLength * pulseLength)

    # Apply the pulse shape along the symbol's interval
    for i in range(inputLength):
```

```
        signal[(i * pulseLength):((i + 1) * pulseLength)] =
pulse if bitstream[i] == 1 else pulse * -1

    return signal


def getFilters():
    """
    This function retrieves three different filters for the
given communication system.
    Case 1: Matched Filter with unit energy.
    Case 2: Non-existent filter (delta function).
    Case 3: Filter with a specific impulse response.
    The function returns a list containing the three
filters.
    """
    # CASE 1
    # h(t) is a Matched Filter with unit energy
    filter1 = np.ones(int(1 / STEP))
    filterLength = len(filter1)
    W = np.zeros(int(1 / STEP) - filterLength)
    filter1 = np.concatenate((filter1, W))

    # CASE 2
    # h(t) is non-existent [h(t) = delta(t)]
    filter2 = np.ones(1)
    filterLength = len(filter2)
    W = np.zeros(int(1 / STEP) - filterLength)
    filter2 = np.concatenate((filter2, W))

    # CASE 3
    # h(t) has the following impulse response
    filter3 = np.sqrt(3) * np.arange(0, 1, STEP)
    filterLength = len(filter3)
    W = np.zeros(int(1 / STEP) - filterLength)
```

```
    filter3 = np.concatenate((filter3, W))

    return [filter1, filter2, filter3]


def applyReceiveFilter(signal, filter, num):
    """
    This function applies a receive filter to a given signal
using convolution. The resulting filtered signal is
returned. If the filter is the first or third filter, the
filtered signal is multiplied by the step size.
    """
    # Apply Convolution (y(t) = r(t) * h(t)) where r(t) =
g(t) + w(t)
    result = np.convolve(signal, filter)

    # For the first & third filters, we'll need to multiply
by the step
    # This won't be needed for the non-existent filter
(Delta)
    if (num != 2):
        result = result * STEP

    # Return the receive filter result
    return result

# variance is the noise's variance and f is the filter's
E_N. (1 or 2 or 3)


def BER(var, g, filter, num, bitstream, n):
    """
    This function calculates the Bit Error Rate (BER) for a
given communication system. It generates additive white
Gaussian noise (AWGN) and adds it to the signal. The signal
```

```
is then passed through a receive filter, sampled, and
compared with the original bitstream to compute the BER. The
BER value is returned.
    """
    # Generate AWGN Noise
    signalLength = len(g)
    w = np.random.normal(0, var, signalLength)

    # Add the randomly generated noise to the signal
    r = g + w

    # Apply the receive filter
    y = applyReceiveFilter(r, filter, num)

    # Sample the output signal
    samplingOutput = sample(bitstream, y, n)

    # Compute the sampling error in the generated bitstream
& return it
    return np.sum(bitstream != samplingOutput) / n


def Q(x):
    """
    This function returns the value of the Q function for a
given input x, which is defined as half the complementary
error function (erfc) of x divided by the square root of 2.
    """
    return 0.5 * erfc(x / sqrt(2))


def filtersBER(filters):
    """
    This function calculates and plots the Bit Error Rate
(BER) for different filters in a communication system. It
```

```
generates a random bitstream, generates a signal based on
the bitstream, and computes the theoretical and simulated
BERs for each filter under different E/N0 values. The BER
values are plotted on a log-scale graph along with the
theoretical BER values.
    """
    # Set the bitstream length
    n = 1000000

    # Input Bitstream
    bitstream = np.random.randint(0, 2, n)

    # Generate Signal
    signal = generateSignal(bitstream)

    # E/N0
    E_N = np.arange(-10, 20, 1)
    N = 1 / (10 ** (E_N / 10))

    # Compute the variance
    variance = np.sqrt(N / 2)

    # BER for each filter
    numberOfFilters = len(filters)

    BERs_th = []
    BERs_th.append([Q(1 / var) for var in variance])
    BERs_th.append([Q(1 / var) for var in variance])
    BERs_th.append([Q((np.sqrt(3)/2) * (1 / var)) for var in
variance])

    # Plot Theoretical BER
    plt.semilogy(E_N, BERs_th[0], 'c')
    plt.semilogy(E_N, BERs_th[1], 'm--')
```

```
    plt.semilogy(E_N, BERs_th[2], 'y')

    BERs = []
    for i in range(numberOfFilters):
        BERs.append([BER(var, signal, filters[i], i + 1,
bitstream, n)
                    for var in variance])

    # Plot the Simulated BER
    plt.semilogy(E_N, BERs[0], 'r')
    plt.semilogy(E_N, BERs[1], 'g.')
    plt.semilogy(E_N, BERs[2], 'b')

    # Labels/Legend/YLim
    plt.xlabel('E/N (db)')
    plt.ylabel('BER (log-scale)')
    plt.title(' BER VS. E/N')
    plt.legend(['Theory 1', 'Theory 2', 'Theory 3',
                'Matched Filter (1)', 'No Filter (2)',
'Linear Filter (3)'])
    plt.ylim([10 / n, 1])
    plt.savefig('./BitErrorRate.png')
    plt.show()


def run():
    # Define n
    n = 10

    # Define the range t
    t = np.arange(0, n, STEP)

    # Generate a 1D array of random binary values
    bitstream = np.random.randint(0, 2, n)
```

```
    # Generate the signal from our bitstream
    signal = generateSignal(bitstream)

    # Plot the Signal
    draw(t, signal, "Time in seconds", "g(t)", "Signal")

    # Generate random Noise (AWGN)
    # Additive White Gaussian Noise
    signalLength = len(signal)
    noise = np.random.normal(0, SIGMA_NOISE, signalLength)

    # Plot the Noise
    draw(t, noise, "Time in seconds", "w(t)", "Noise")

    # Add the random generated noise to the signal
    noisySignal = signal + noise

    # Plot the Noisy Signal
    draw(t, noisySignal, "Time in seconds", "r(t)", "Noisy
Signal")

    # Generate the receive filters for all 3 cases
    filters = getFilters()

    # Receive Filters
    result1 = applyReceiveFilter(noisySignal, filters[0], 1)
    result2 = applyReceiveFilter(noisySignal, filters[1], 2)
    result3 = applyReceiveFilter(noisySignal, filters[2], 3)

    # PLOT: CASE 1
    t = np.arange(0, len(result1) * STEP, STEP)
    draw(t, result1, "Time in seconds", "y(t)", "Filter 1
Output")
    sample(bitstream, result1, n)
```

```python
    # PLOT: CASE 2
    t = np.arange(0, len(result2) * STEP, STEP)
    draw(t, result2, "Time in seconds", "y(t)", "Filter 2 
Output")
    sample(bitstream, result2, n)

    # PLOT: CASE 3
    t = np.arange(0, len(result3) * STEP, STEP)
    draw(t, result3, "Time in seconds", "y(t)", "Filter 3 
Output")
    sample(bitstream, result3, n)

    # Compute the BER for each filter h(t)
    filtersBER(filters)


if __name__ == '__main__':
    run()

"""
Q(5)

The Bit Error Rate (BER) decreases as the energy per bit (E) 
relative to the noise power spectral density (No) increases, 
which is evident in the plot. This can be explained in 
various ways:

Using the theoretical expression for BER and noting that the 
Q function is a decreasing function, it is apparent that Q(a 
* sqrt(E/No)) for all the cases mentioned in the problem. 
Therefore, as sqrt is an increasing function, it can be 
concluded that BER is a decreasing function of E/No.
"""
```

```
"""
Q(6)

The case that uses a matched filter has the lowest BER
because it employs a filter that is specifically designed to
minimize the probability of error by matching the filter to
the pulse. By doing so, the matched filter maximizes the
peak pulse SNR at the sampling instant, which helps reduce
the probability of error.

It is important to note that in the theoretical case, using
a filter or not yields the same expression due to our
assumptions on variance and PSD.
"""
```