Cairo University
Faculty of

Department of Computer Engineering

ELC 3252 – Spring 2025

Digital Communications

Assignment #2

Matched Filters

Submitted to

Dr. Mai Badawi

Dr. Hala

Eng. Mohamed Khaled

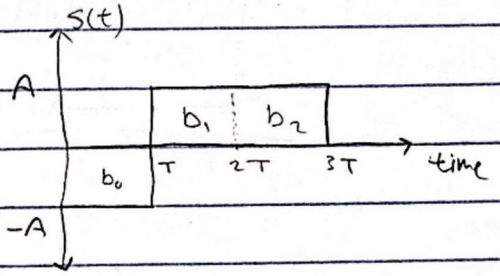
Submitted by

Name	Sec	BN
Ahmed Hamed Gaber	1	3
Somia Saad El-Shemy	1	26

Contents

Part 1 (Hand Analysis)

a) baseband plot for "all" sequence



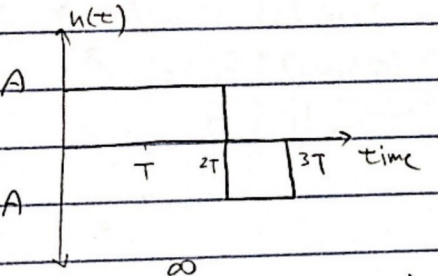
b,c) Matched filter output plot due to signal only

$y(t) = s(t) * h(t)$ $\quad h(t) = s(T-t)$

flip around line of symmetry

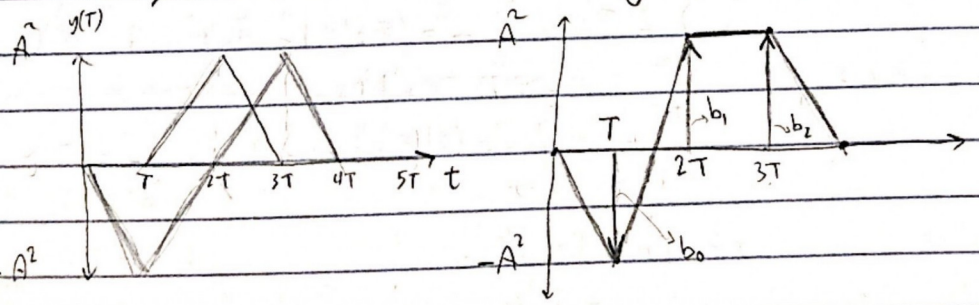
$h(T-t) = s(T-(T-t))$

$= s(t)$

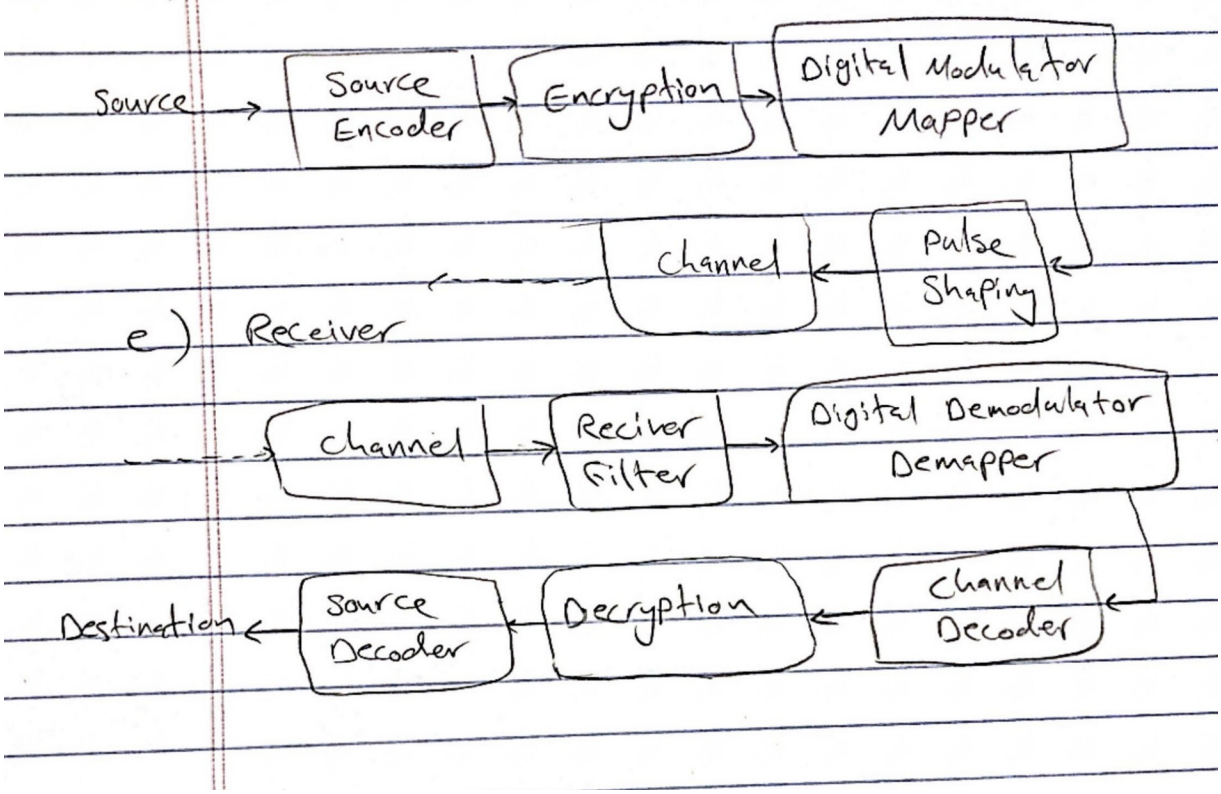


$$y(t) = \int_{-\infty}^{\infty} s(\tau)h(t-\tau)\,d\tau, \quad y(T) = \int_{-\infty}^{\infty} s(\tau)^2\,d\tau$$

→ to make calculations easy we will compute this for every bit



d) Transmitter



e) Receiver

Part 2 (Hand Analysis)

* Part 2: Simulation

$g(t)\begin{cases} -A & '0' \\ A & '1' \end{cases}$ $\quad A=1,\ T=1$

a) $y(t) = r(t) * h(t)$

$$y(T) = \begin{cases} -A^2T + n(T) & '0' = n(T) - 1 \\ A^2T + n(T) & '1' = n(T) + 1 \end{cases}$$

$n(T) = w(t) * h(t)$

$\mu_y = E[y(T)] = E[g_0(T)] + E(n(T))$

$$E[g_0(t)] = \begin{cases} -1 & '0' \\ 1 & '1' \end{cases}$$

$E[n(T)] = E[\int w(\tau)h(T-\tau)d\tau]$

$= E[\int w(\tau)g(\tau)d\tau] = E[\int_0^T A\,w(\tau)d\tau]$

$= E[\int_0^1 w(\tau)d\tau] = 0 \qquad E[w(\tau)] = 0$

$$\mu_y = \begin{cases} -1 & '0' \\ 1 & '1' \end{cases}$$

$\sigma_y^2 = var[y(T)] = E[(g_0(T) + n(T) - \mu_y)^2]$

$= E[n(T)^2] = P_n$ "power of noise"

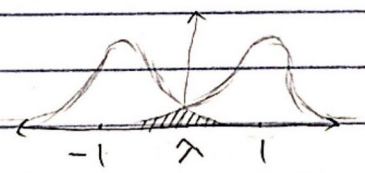
$\sigma_y^2 = \frac{N_0}{2}\int_{-\infty}^{\infty}|H(f)|^2 df = \frac{N_0}{2}\int_0^T |h(t)|^2 dt$

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

$P(y) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(y-\mu_y)^2}{2\sigma^2}\right)$

$P(y|0) = \frac{1}{\sqrt{\pi N_0}}\exp\left(-\frac{(y+1)^2}{N_0}\right)$

$P(y|1) = \frac{1}{\sqrt{\pi N_0}}\exp\left(-\frac{(y-1)^2}{N_0}\right)$



$\lambda = \frac{1-1}{2} = 0$

$$P(e|0) = \int_{?}^{\infty} p(y|0)\,dy = \int_{0}^{\infty} \frac{1}{\sqrt{\pi N_0}} \exp\left(-\frac{(y+1)^2}{N_0}\right) dy$$

let $z = \frac{y+1}{\sqrt{N_0}}$ $\quad dz = \frac{1}{\sqrt{N_0}}\,dy$ $\quad dy = \sqrt{N_0}\,dz$

$$P(e|0) = \frac{1}{\sqrt{\pi N_0}} \int_{\frac{1}{\sqrt{N_0}}}^{\infty} \exp(-z^2)\sqrt{N_0}\,dz$$

$$= \frac{1}{\sqrt{\pi}} \int_{\frac{1}{\sqrt{N_0}}}^{\infty} \exp(-z^2)\,dz$$

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

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

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

$$= \boxed{\frac{1}{2}\operatorname{erfc}\left(\frac{1}{\sqrt{N_0}}\right)} \quad \#\#$$

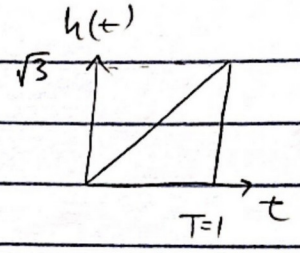
b) $h(t) = \delta(t)$ $\quad y(t) = r(t) * \delta(t)$

$$y(T) = \begin{cases} \to n(T) - 1 \\ \to n(T) + 1 \end{cases}$$

It is the same as (a)

$$\boxed{P(e) = \frac{1}{2}\operatorname{erfc}\left(\frac{1}{\sqrt{N_0}}\right) \quad \#\#}$$

c) $y(t) = r(t) * h(t)$



$$y_0(T) = \int_0^T h(\tau)\, y(T-\tau)\, d\tau$$

$$= \int_0^T \frac{\sqrt{3}}{2} A\tau\, d\tau = \frac{\sqrt{3}}{2} AT = \frac{\sqrt{3}}{2}$$

$$y(T) = \begin{cases} -\frac{\sqrt{3}}{2} + n(T) & (0) \\ \frac{\sqrt{3}}{2} + n(T) & (1) \end{cases} \qquad \mu_y = \begin{cases} -\frac{\sqrt{3}}{2} & (0) \\ \frac{\sqrt{3}}{2} & (1) \end{cases}$$

$$\sigma_y^2 = \frac{N_0}{2} \int_0^T |h(t)|^2 dt = \frac{N_0}{2} \int_0^T (\sqrt{3}\, t)^2 dt = \frac{N_0}{2} \int_0^T 3t^2 dt$$

$$= \frac{N_0}{2} 3 \cdot \frac{1}{3} t^3 \Big|_0^T = \frac{N_0}{2} T^3 = \frac{N_0}{2}$$

$$P(e) = P(e|0) \qquad P(y|0) = \frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{(y + \sqrt{3}/2)^2}{N_0} \right)$$

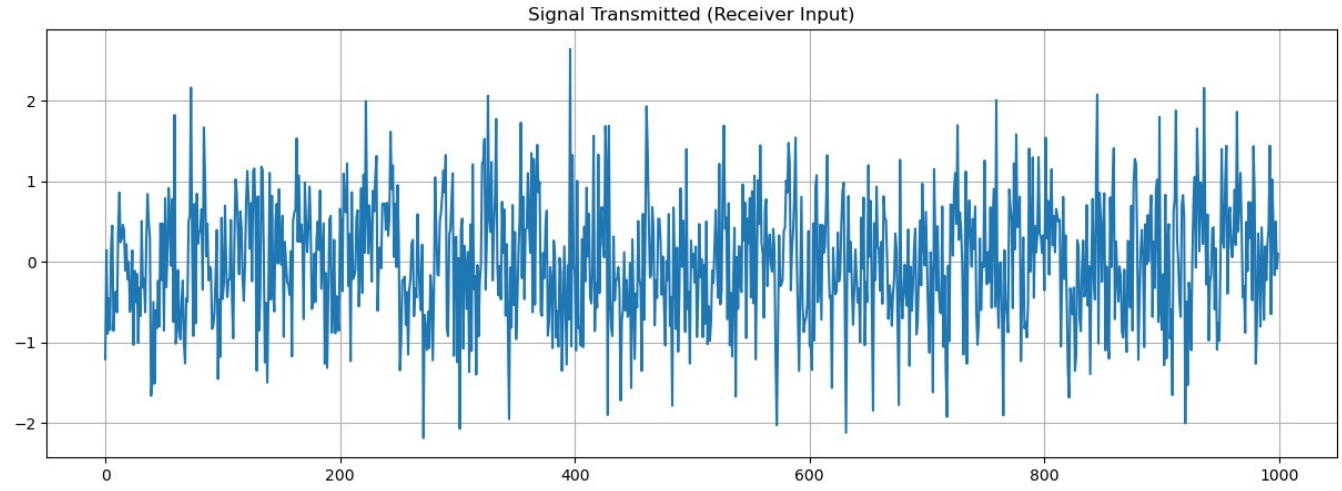
$$P(e|0) = \int_0^\infty P(y|0)\, dy \qquad z = \frac{y + \frac{\sqrt{3}}{2}}{\sqrt{N_0}} \qquad dz = \frac{dy}{\sqrt{N_0}} \qquad dy = \sqrt{N_0}\, dz$$

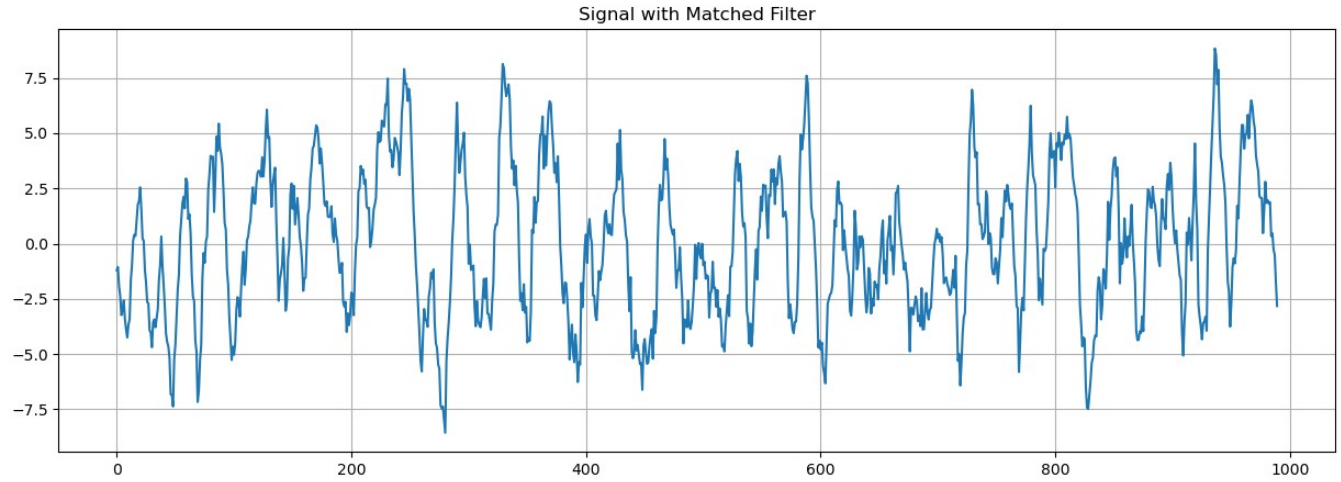
$$P(e) = \frac{1}{\sqrt{\pi N_0}} \int_{\frac{\sqrt{3}}{2\sqrt{N_0}}}^\infty \exp(-z^2) \sqrt{N_0}\, dz = \frac{1}{\sqrt{\pi}} \int_{\frac{\sqrt{3}}{2\sqrt{N}}}^\infty \exp(-z^2)\, dz$$

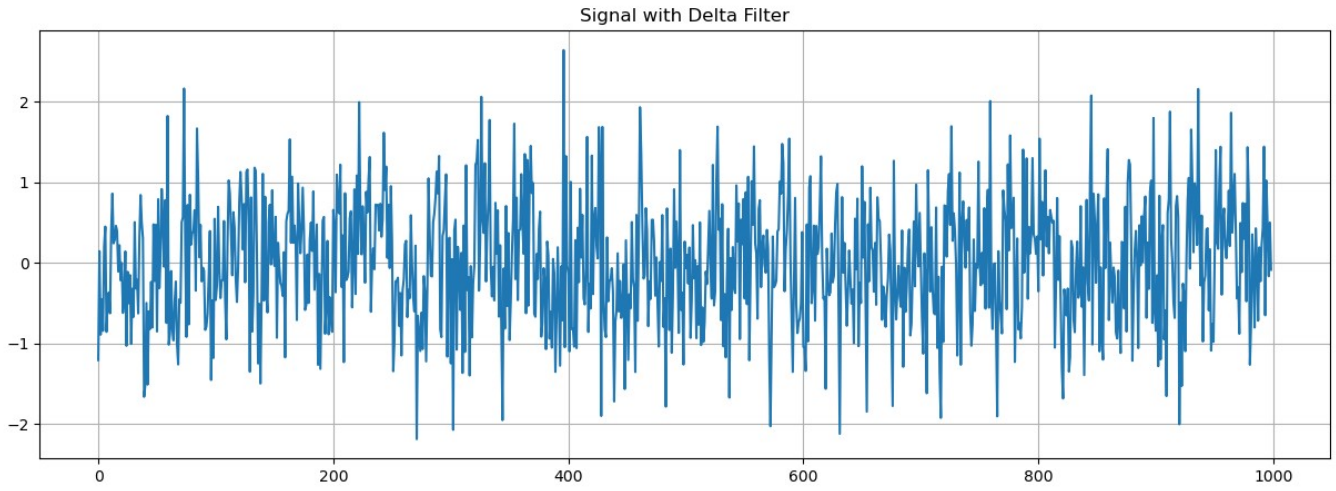
$$\boxed{P(e) = \frac{1}{2} \operatorname{erfc}\left(\frac{\sqrt{3}}{2\sqrt{N_0}} \right)} \quad \#$$

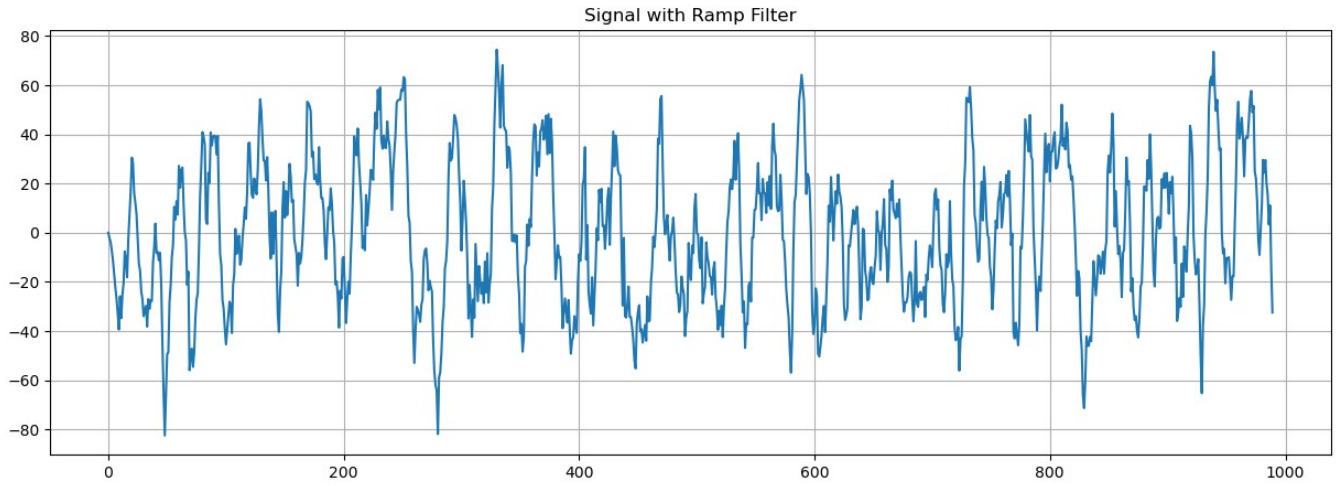
Part 2 (Simulation)

Output of Receive Filters





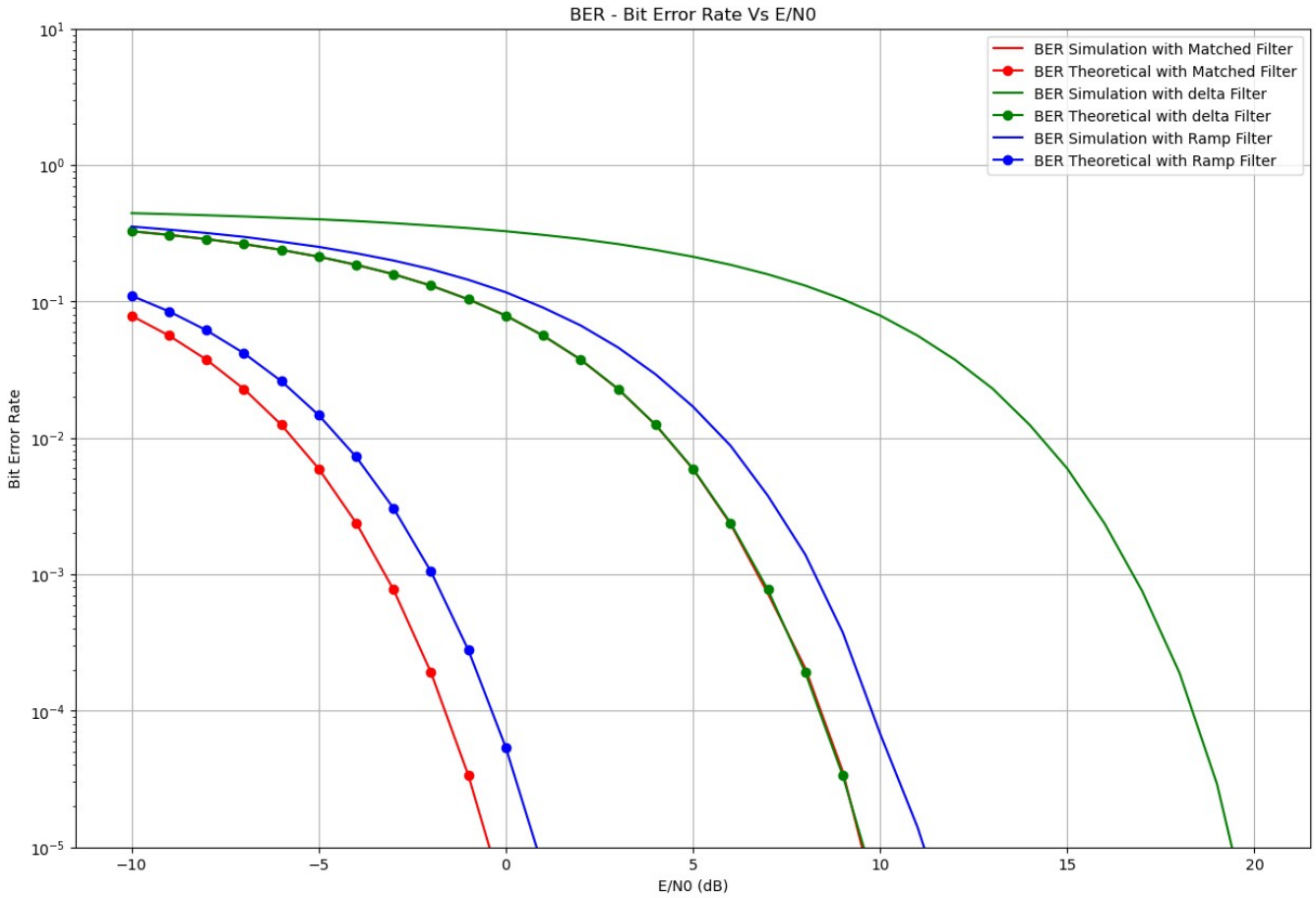




Comment:

The plots shows that matched filters give the best results, and delta (pulse) filters give the worst results.

BER vs E/N0



Comment:

It seems that simulated values are the same as theoretical but shifted right. Simulated values are enhanced when increasing number of bits.

Q5 Answer

It is obvious that BER is a **decreasing** function of E/N0, because increasing E/N0 means that signal energy is bigger compared to noise energy, which means that probability of error will decrease.

Q6 Answer

It is obvious that it is **Matched Filters**, because matched filters multiply each value in the signal with a suitable value, high value with a high value and low value with a low value, which makes the original signal preserved, therefore resulting in lower BER values.

Code (Python)

Imports and global variables

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

number_bits = 100
samples_per_bit = 10
sampling_indices = np.arange(1, number_bits + 1) * samples_per_bit - 1

N0 = 1
E = 1
A = 1

ber_simulated_with_matched_filter = []
ber_theortical_with_matched_filter = []

ber_simulated_with_delta_filter = []
ber_theortical_with_delta_filter = []

ber_simulated_with_ramp_filter = []
ber_theortical_with_ramp_filter = []
```

Functions

```python
def generate_data():
    return np.random.choice([0, 1], size=number_bits)

def generate_pulse(data=None):
    if data is None:
        data = generate_data()
    data = np.where(data == 0, -1, data)
    pulse_stream = np.repeat(data, samples_per_bit)
    single_pulse = np.linalg.norm(A * np.ones(samples_per_bit), axis=0)
    return pulse_stream / single_pulse

def generate_noise(N0=N0):
    segma = np.sqrt(N0 / 2)
    return np.random.normal(loc=0, scale=segma, size=(number_bits * samples_per_bit))

def generate_filter(type):
    base = np.zeros(number_bits * samples_per_bit)
    if type == "ramp":
        base[:samples_per_bit] = np.sqrt(3) * np.arange(0, samples_per_bit, dtype=np.float64)
    elif type == "delta":
        base[0] = A
    elif type == "matched":
        base[:samples_per_bit] = A
    return base

def transmit_over_channel(signal, noise):
    return signal + noise

def filter_receive(signal, filter_type):
    filter = generate_filter(filter_type)
    return convolve(signal, filter)

def sample_signal(signal):
    return (signal[sampling_indices] > 0).astype(int)

def calculate_theoretical_ber(N0, filter_type):
    if filter_type == "ramp":
        return 0.5 * erfc(A * np.sqrt((3 * samples_per_bit) / (4 * N0)))
    elif filter_type == "delta":
        return 0.5 * erfc(A * np.sqrt(1 / N0))
    elif filter_type == "matched":
        return 0.5 * erfc(A * np.sqrt(samples_per_bit / N0))
    return 0

def calculate_simulated_ber(bits, received_sampled_bits):
    errors = np.sum(bits != received_sampled_bits)
    return errors / bits.shape[0]
```

Simulation & Ploting

```python
def plot_signal(title, signal, trim=True):
    if trim:
        signal = signal[:np.argmax(signal[::-1] != 0)]
    plt.figure(figsize=(15, 5))
    plt.title(title)
    plt.plot(signal)
    plt.grid(True)
    plt.show()

def plot_error():
    plt.figure(figsize=(15, 10))
    plt.title('BER - Bit Error Rate Vs E/N0')

    plt.plot(range(-10, 21), ber_simulated_with_matched_filter, 'r', label="BER Simulation with Matched Filter")
    plt.plot(range(-10, 21), ber_theortical_with_matched_filter, 'o-r', label="BER Theoretical with Matched Filter")

    plt.plot(range(-10, 21), ber_simulated_with_delta_filter, 'g',  label="BER Simulation with delta Filter")
    plt.plot(range(-10, 21), ber_theortical_with_delta_filter, 'o-g', label="BER Theoretical with delta Filter")

    plt.plot(range(-10, 21), ber_simulated_with_ramp_filter, 'b',  label="BER Simulation with Ramp Filter")
    plt.plot(range(-10, 21), ber_theortical_with_ramp_filter, 'o-b', label="BER Theoretical with Ramp Filter")

    plt.xlabel('E/N0 (dB)')
    plt.ylabel('Bit Error Rate')
    plt.yscale('log')
    plt.ylim(10**(-5), 10)
    plt.grid()
    plt.legend()
    plt.show()
```

```python
signal_transmitted = transmit_over_channel(generate_pulse(), generate_noise())
signal_with_matched_filter = filter_receive(signal_transmitted, "matched")
signal_with_delta_filter = filter_receive(signal_transmitted, "delta")
signal_with_ramp_filter = filter_receive(signal_transmitted, "ramp")
signal_with_matched_filter_sampled = sample_signal(signal_with_matched_filter)
signal_with_delta_filter_sampled = sample_signal(signal_with_delta_filter)
signal_with_ramp_filter_sampled = sample_signal(signal_with_ramp_filter)
plot_signal("Signal Transmitted (Receiver Input)", signal_transmitted, trim=False)
plot_signal("Signal with Matched Filter", signal_with_matched_filter)
plot_signal("Signal with Delta Filter", signal_with_delta_filter)
plot_signal("Signal with Ramp Filter", signal_with_ramp_filter)
```

BER Error

```python
number_bits = 100000
sampling_indices = np.arange(1, number_bits + 1) * samples_per_bit - 1
data = generate_data()
signal = generate_pulse(data)

ber_simulated_with_matched_filter = []
ber_theortical_with_matched_filter = []

ber_simulated_with_delta_filter = []
ber_theortical_with_delta_filter = []

ber_simulated_with_ramp_filter = []
ber_theortical_with_ramp_filter = []

for EN0 in range(-10, 21):
    EN0 = 10 ** (EN0/10)
    N0 = E / EN0

    signal_transmitted = transmit_over_channel(signal, generate_noise(N0))
    signal_with_matched_filter = filter_receive(signal_transmitted, "matched")
    signal_with_delta_filter = filter_receive(signal_transmitted, "delta")
    signal_with_ramp_filter = filter_receive(signal_transmitted, "ramp")
    signal_with_matched_filter_sampled = sample_signal(signal_with_matched_filter)
    signal_with_delta_filter_sampled = sample_signal(signal_with_delta_filter)
    signal_with_ramp_filter_sampled = sample_signal(signal_with_ramp_filter)

    ber_simulated_with_matched_filter.append(calculate_simulated_ber(data, signal_with_matched_filter_sampled))
    ber_theortical_with_matched_filter.append(calculate_theoretical_ber(N0, "matched"))

    ber_simulated_with_delta_filter.append(calculate_simulated_ber(data, signal_with_delta_filter_sampled))
    ber_theortical_with_delta_filter.append(calculate_theoretical_ber(N0, "delta"))

    ber_simulated_with_ramp_filter.append(calculate_simulated_ber(data, signal_with_ramp_filter_sampled))
    ber_theortical_with_ramp_filter.append(calculate_theoretical_ber(N0, "ramp"))

plot_error()
```