Fathan Ananta Nur
236-D8749

44135 Applied Information Theory
Assignment 5

Perform computer simulations of digital communications using (7,4)-Hamming codes (1-bit error-correcting codes).
Compare the probabilities of incorrect decoding for same p.

These experiments below will use these parameters.

```
Length (N) = 1000000
Parameter c and t = 0.49999
Initial value = 0.1782612
Initial error value = 0.5673244
Initial counter correct decoding = 0
Initial counter incorrect decoding = 0
Initial counter error probability (before decoding) = 0
Initial counter error probability (after decoding) = 0
```

1. Memoryless errors (skew Bernoulli map)
   In this experiment, the additional parameter below will be used:

```
List of p = {0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.49999}
```

The experiment will yield results for both computed and theoretical values of the probability of incorrect decoding, also bit error probability before decoding and after decoding. Those values can be obtained based on the parameters described above and increments in the value of p. The theoretical values on probability of incorrect decoding will be calculated using the following formula.

$$P_I = 1 - (7p \cdot (1-p)^6 + (1-p)^7)$$

The experiment was conducted in a Python environment, and the results are displayed in the following table.

Table 1. Probability of Incorrect Decoding and Bit Error Probability on Memoryless Source Bernoulli Map

p	probability of incorrect decoding		bit error probability	
	computed	theoretical	before decoding	after decoding
0.05	0.04430	0.04438	0.04985	0.01940
0.1	0.14986	0.14969	0.10011	0.06694
0.15	0.28354	0.28342	0.15016	0.12906
0.2	0.42249	0.42328	0.19988	0.19584
0.25	0.55557	0.55505	0.25016	0.26193
0.3	0.67023	0.67058	0.29972	0.32162

0.35	0.76625	0.76620	0.35020	0.37486
0.4	0.84230	0.84137	0.40021	0.42153
0.45	0.89713	0.89758	0.44985	0.46176
0.49999	0.93723	0.93749	0.49991	0.49994

Table 1 presents both the computed and theoretical values of the probability of incorrect decoding, as well as bit error probability before encoding and after encoding as a function of incremental values of p. The observed alignment between computed and theoretical values indicates minimal discrepancy. Moreover, an increase in the value of p correlates with a rise in the probability of incorrect decoding. This trend arises because higher values of p enhance the likelihood of multiple bit errors occurring simultaneously resulting incorrect decoding.

The performance of the hamming code in **error correction is highly dependent on the level of noise in the communication channel**. When the p value is low, specifically below a threshold **between 0.2 and 0.25**, the Hamming code is very effective. In this low-noise regime, the code can correct single-bit errors efficiently, leading to a significantly reduced bit error probability after decoding compared to the probability before decoding. This is because the likelihood of encountering only single-bit errors is high, and such errors are perfectly corrected by the Hamming code. However, as the noise level increases and p exceeds the threshold, the code's effectiveness diminishes.

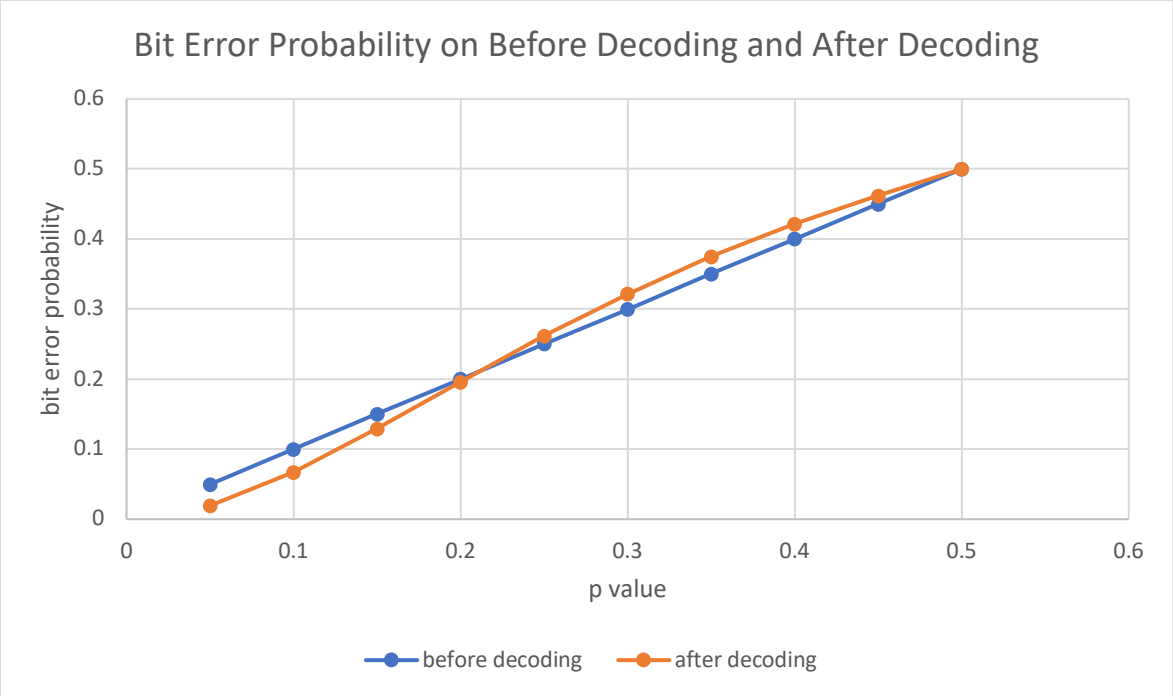

Figure 1. Bit Error Probability on Before Decoding and After Decoding

This phenomenon is illustrated in Figure 1, where an intersection occurs between the p values of 0.2 and 0.25, resulting in a higher bit error probability after decoding compared to before decoding. The probability of encountering multiple bit errors, which the Hamming code cannot correct reliably, rises. This results in a higher bit error

probability after decoding. Consequently, while the **hamming code is beneficial for low to moderate noise levels**, its performance deteriorates in highly noisy environments, indicating the need for more robust error correction methods for such scenarios.

2. Markov-type errors (PLM3 map)
In this experiment, the additional parameters below will be used:

```
List of p = {0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.49999}
List of another p (p2) = {0.16, 0.34}
```

The experiment will produce results for both computed and theoretical values of the probability of incorrect decoding, also bit error probability before decoding and after decoding. Those results are influenced by the specified parameters and incremental adjustments to the value of p. Additionally, the experiment will introduce a secondary parameter, $p2$, to satisfy the conditions of the Markov map. From p and $p2$, $p1$ will be derived using the specified formula.

$$p_1 = \frac{p}{1-p} p_2 \quad \begin{cases} p_2 : \text{another parameter} \\ p_2 \neq t = 1 - p \end{cases}$$

The theoretical values on probability of incorrect decoding will be calculated based on the provided mathematical formula.

$$P_I = 1 - \left(\frac{p_1}{p_1+p_2} \cdot p_2 \cdot (1-p_1)^5 + 5 \cdot \frac{p_2}{p_1+p_2} \cdot p_1 \cdot p_2 \cdot (1-p_1)^4 + \frac{p_2}{p_1+p_2} \cdot (1-p_1)^5 \cdot p_1 + \frac{p_2}{p_1+p_2} \cdot (1-p_1)^6\right)$$

Table 2. Probability of Incorrect Decoding and Bit Error Probability on Markov-type Map

p	$p1$	$p2$	probability of incorrect decoding		bit error probability	
			computed	theoretical	before decoding	after decoding
0.05	0.008	0.16	0.07569	0.07548	0.04998	0.04945
0.05	0.018	0.34	0.08957	0.08959	0.04997	0.04761
0.1	0.018	0.16	0.15079	0.15066	0.10000	0.09911
0.1	0.038	0.34	0.18074	0.18004	0.10043	0.09658
0.15	0.028	0.16	0.22512	0.22549	0.14959	0.14846
0.15	0.060	0.34	0.27156	0.27106	0.15028	0.14581
0.2	0.040	0.16	0.30040	0.29987	0.20033	0.19901
0.2	0.085	0.34	0.36120	0.36227	0.19953	0.19520
0.25	0.053	0.16	0.37293	0.37366	0.24948	0.24806
0.25	0.113	0.34	0.45288	0.45309	0.25004	0.24676
0.3	0.069	0.16	0.44616	0.44671	0.29937	0.29806
0.3	0.146	0.34	0.54240	0.54273	0.29963	0.29776

0.35	0.086	0.16	0.51911	0.51880	0.35014	0.34909
0.35	0.183	0.34	0.62929	0.63011	0.34959	0.34913
0.4	0.107	0.16	0.58942	0.58961	0.39992	0.39908
0.4	0.227	0.34	0.71363	0.71374	0.40006	0.40096
0.45	0.131	0.16	0.65922	0.65874	0.44990	0.44972
0.45	0.278	0.34	0.79187	0.79164	0.45049	0.45175
0.49999	0.160	0.16	0.72567	0.72556	0.49982	0.49985
0.49999	0.340	0.34	0.86108	0.86124	0.50008	0.50005

Table 2 displays both the computed and theoretical values of the probability of incorrect decoding, as well as bit error probability on before decoding and after decoding, analyzing how these values change with incremental increases in the parameter p and the combined effects of $p1$ and $p2$. Similar to the observations in Table 1, the discrepancy between the computed and theoretical values is not substantial. As the parameter p increases, the probability of undetected error also rises. The same behavior also happened when the **increments incorporate a larger $p2$**, there is **a significant increase in the probability of incorrect decoding** in both computed and theoretical assessments.

Comparing with Table 1, the observed difference in bit error probabilities between skew Bernoulli and Markov-type error models can be attributed to the distinct nature of error distributions they generate. In the simulation, using the skew Bernoulli map with a single parameter p results in independent and randomly distributed bit errors. At lower p values (below 0.2 to 0.25), the hamming code effectively corrects single-bit errors, leading to a reduced bit error probability after decoding. However, as p increases beyond this threshold, the likelihood of multiple simultaneous bit errors rises, diminishing the hamming code's effectiveness and causing the bit error probability after decoding to exceed that before decoding.

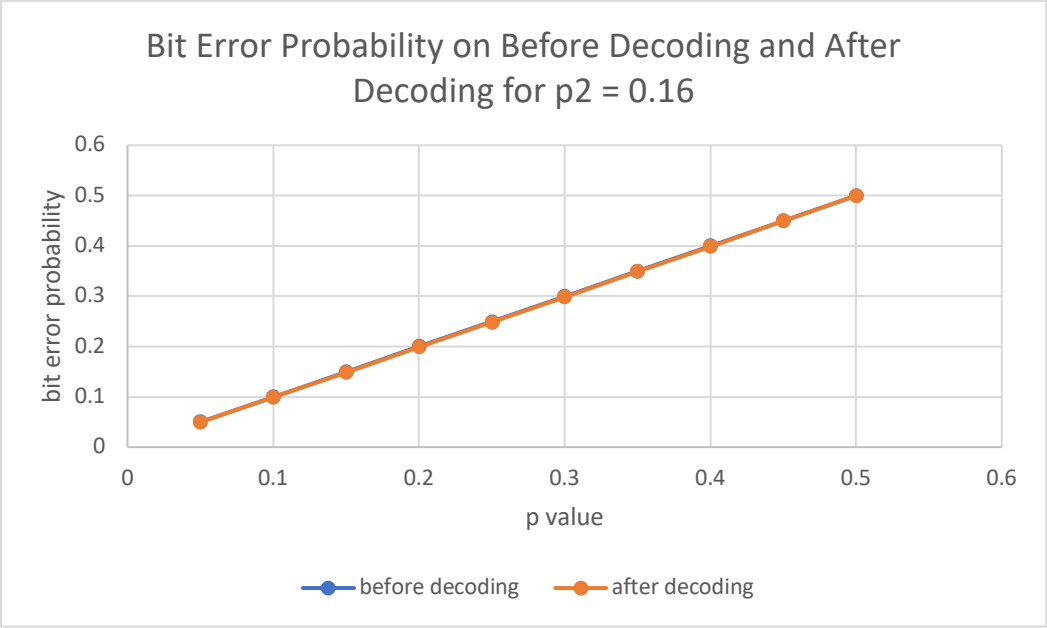

Figure 2. Bit Error Probability on Before Decoding and After Decoding for $p2 = 0.16$

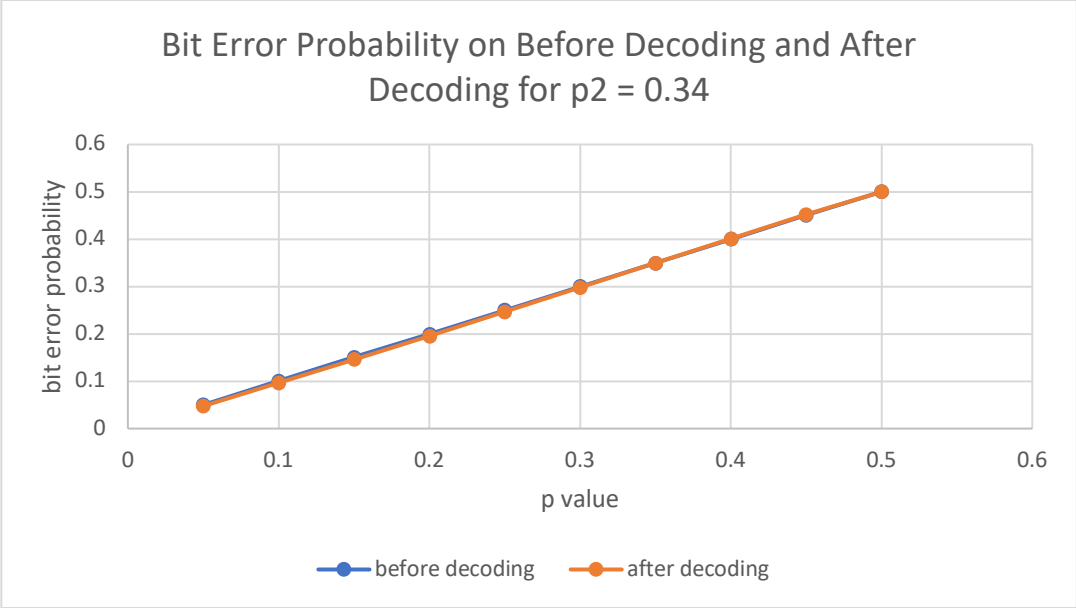

Figure 3. Bit Error Probability on Before Decoding and After Decoding for $p2 = 0.34$

In contrast, the Markov-type error model introduces correlated errors, where the occurrence of an error depends on the previous bit's state. This correlation tends to create error patterns that are more manageable for the hamming code, as it can consistently correct single-bit errors even within bursts of errors. Consequently, the bit error probability after decoding remains lower than before decoding across various p values, as shown on Figure 2 and Figure 3, which have no intersection on the values before decoding and the values after decoding. These findings underscore the **importance of selecting appropriate error models for simulation**, as they significantly impact the performance evaluation of error correction codes. The Markov-type error model's alignment with the hamming code's strengths highlights the necessity of considering realistic error patterns to ensure accurate assessments of communication system reliability.

Appendix

1. Threshold function

$$\theta_t(x) = \begin{cases} 0 & (x < t) \\ 1 & (x \geq t) \end{cases}$$

2. Bernoulli map function

$$\tau(x, c) = \begin{cases} \dfrac{x}{c} & 0 \leq x < c \\ \dfrac{x - c}{1 - c} & c \leq x \leq 1 \end{cases}$$

3. Markov source parameters construction and function

Chaotic Map for Markov Source (Summary)

➢ Choose p_1, p_2 $(p_1 + p_2 \neq 1)$

$$t = \frac{p_2}{p_1 + p_2}, \quad a = \frac{1}{1 - (p_1 + p_2)} \begin{cases} > 0 & (p_1 + p_2 < 1) \\ < 0 & (p_1 + p_2 > 1) \end{cases}$$

	$a > 0$	$a < 0$
c_1	$t(1 - a^{-1})$	$t + (1 - t)a^{-1}$
c_2	$t + (1 - t)\, a^{-1}$	$t(1 - a^{-1})$
a_1	$\dfrac{1}{c_1}$	$\dfrac{1}{c_1}$
a_2	$\dfrac{1}{1 - c_2}$	$\dfrac{1}{1 - c_2}$
$\tau(x)$	$\begin{cases} a_1 x & (0 \leq x < c_1) \\ a(x - c_1) & (c_1 \leq x < c_2) \\ a_2(x - c_2) & (c_2 \leq x \leq 1) \end{cases}$	$\begin{cases} a_1 x & (0 \leq x < c_1) \\ a(x - c_2) & (c_1 \leq x < c_2) \\ a_2(x - c_2) & (c_2 \leq x \leq 1) \end{cases}$

-13-

4. Source code

```
import numpy as np
import matplotlib.pyplot as plt
import os

def threshold_function(x, t):  # threshold function for making 0 and 1 value
    return 0 if x < t else 1
```

```
def skew_bernouli_map(x, c): # Bernoulli mapping function
    if x < c:
        return (x / c)
    else:
        return (x - c) / (1 - c)

def plm3(x, p_1, p_2, t): # Markov plm3 mapping function
    def create_parameters(p_1, p_2):
        a = 1 / (1 - (p_1 + p_2))
        if a > 0:
            a_positive = True
            c1 = t * (1 - (1/a))
            c2 = t + ((1 - t)/a)
            a1 = 1 / c1
            a2 = 1 / (1 - c2)
        else:
            a_positive = False
            c1 = t + ((1 - t)/a)
            c2 = t * (1 - (1/a))
            a1 = 1 / c1
            a2 = 1 / (1 - c2)

        return a, a_positive, c1, c2, a1, a2

    a, a_positive, c1, c2, a1, a2 = create_parameters(p_1, p_2)

    if x < c1:
        return a1 * x
    elif c1 <= x < c2:
        if a_positive:
            return a * (x - c1)
        else:
            return a * (x - c2)
    elif c2 <= x <= 1:
        return a2 * (x - c2)
    else:
        return None  # Handle case when x is outside [0, 1]

def memoryless_bernoulli():
    l = 1000000 # length (N)
    c = t = 0.49999 # t = c = 0.5 (~ 4.9999)
    def cte(p): # for sequence of errors with memoryless source (c=t=1-p)
        return 1 - p
    p_list = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.49999]

    for p in p_list:
        x0 = 0.1782612 # initial value for sequence
        z0 = 0.5673244 # initial value for error sequence
```

```
        ok = 0 # counter for correct decoding
        berr0 = 0 # counter for error probability (before decoding)
        blerr = 0 # counter for incorrect decoding
        berr = 0 # counter for error probability (after decoding)

        for i in range (l):
            # information source and coding : Bernoulli map (c = t)
            x1 = skew_bernouli_map(x0, c); x2  = skew_bernouli_map(x1, c); x3 =
skew_bernouli_map(x2, c)
            b0 = threshold_function(x0, t); b1 = threshold_function(x1, t); b2 =
threshold_function(x2, t); b3 = threshold_function(x3, c)
            b4 = b0 ^ b1 ^ b2; b5 = b0 ^ b1 ^ b3; b6 = b0 ^ b2 ^ b3
            x0 = skew_bernouli_map(x3, c) # prepare x0 for next loop

            # generating a sequence of errors with memoryless source (c=t=1-p) and information
xor error
            z1 = skew_bernouli_map(z0, cte(p)); z2 = skew_bernouli_map(z1, cte(p)); z3 =
skew_bernouli_map(z2, cte(p)); z4 = skew_bernouli_map(z3, cte(p)); z5 = skew_bernouli_map(z4,
cte(p)); z6 = skew_bernouli_map(z5, cte(p))
            e0 = threshold_function(z0, cte(p)); e1 = threshold_function(z1, cte(p)); e2 =
threshold_function(z2, cte(p)); e3 = threshold_function(z3, cte(p)); e4 =
threshold_function(z4, cte(p)); e5 = threshold_function(z5, cte(p)); e6 =
threshold_function(z6, cte(p)) # error sequence
            r0 = b0 ^ e0; r1 = b1 ^ e1; r2 = b2 ^ e2; r3 = b3 ^ e3; r4 = b4 ^ e4; r5 = b5 ^ e5;
r6 = b6 ^ e6 # received sequence
            z0 = skew_bernouli_map(z6, cte(p)) # prepare z0 for next loop

            # ecount the number of error bits (before decoding)
            berr0 = berr0 + e0 + e1 + e2 + e3 + e4 + e5 + e6

            # calculation of the syndrome
            s0 = r0 ^ r1 ^ r2 ^ r4; s1 = r0 ^ r1 ^ r3 ^ r5; s2 = r0 ^ r2 ^ r3 ^ r6

            # error-correcting based on the syndrome
            if ((s0 == 1) & (s1 == 1) & (s2 == 1)):
                r0 = r0 ^ 1
            elif ((s0 == 1) & (s1 == 1) & (s2 == 0)):
                r1 = r1 ^ 1
            elif ((s0 == 1) & (s1 == 0) & (s2 == 1)):
                r2 = r2 ^ 1
            elif ((s0 == 0) & (s1 == 1) & (s2 == 1)):
                r3 = r3 ^ 1
            elif ((s0 == 1) & (s1 == 0) & (s2 == 0)):
                r4 = r4 ^ 1
            elif ((s0 == 0) & (s1 == 1) & (s2 == 0)):
                r5 = r5 ^ 1
            elif ((s0 == 0) & (s1 == 0) & (s2 == 1)):
                r6 = r6 ^ 1
```

```
            # count the number of incorrect decoding
            if ((r0 == b0) & (r1 == b1) & (r2 == b2) & (r3 == b3) & (r4 == b4) & (r5 == b5) &
(r6 == b6)):
                ok += 1
            else:
                blerr += 1

            # count the number of error bits (after decoding)
            berr = berr + (r0 ^ b0) + (r1 ^ b1) + (r2 ^ b2) + (r3 ^ b3) + (r4 ^ b4) + (r5 ^ b5)
+ (r6 ^ b6)

        # probability of incorect decoding
        incorrect_computed_value = blerr / l
        correct_theoretical_value = 7 * p * (1 - p)**6 + (1 - p)**7
        incorrect_theoretical_value = 1 - correct_theoretical_value

        # probability of bit error (before and after decoding)
        bit_error_before = berr0 / (7 * l)
        bit_error_after = berr / (7 * l)

        # print the result
        print(f'For p: {p}\nINCORRECT DECODING; computed value: {incorrect_computed_value:.5f}
and theoretical value: {incorrect_theoretical_value:.5f}\nPROBABILITY BIT ERROR; before:
{bit_error_before:.5f} and after: {bit_error_after:.5f}\n\n')

def markov():
    l = 1000000 # length (N)
    c = t = 0.49999 # t = c = 0.5 (~ 4.9999)
    def cte(p): # for sequence of errors with memoryless source (c=t=1-p)
        return 1 - p
    p_list = [0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.49999]
    p2_list = [0.16, 0.34] # another parameter p2

    for p in p_list:
        for p2 in p2_list:
            p1 = p / (1 - p) * p2
            x0 = 0.1782612 # initial value for sequence
            z0 = 0.5673244 # initial value for error sequence
            ok = 0 # counter for correct decoding
            berr0 = 0 # counter for error probability (before decoding)
            blerr = 0 # counter for incorrect decoding
            berr = 0 # counter for error probability (after decoding)

            for i in range (l):
                # information source and coding : Bernoulli map (c = t)
                x1 = skew_bernouli_map(x0, c); x2  = skew_bernouli_map(x1, c); x3 =
skew_bernouli_map(x2, c)
```

```
                b0 = threshold_function(x0, t); b1 = threshold_function(x1, t); b2 =
threshold_function(x2, t); b3 = threshold_function(x3, c)
                b4 = b0 ^ b1 ^ b2; b5 = b0 ^ b1 ^ b3; b6 = b0 ^ b2 ^ b3
                x0 = skew_bernouli_map(x3, c) # prepare x0 for next loop

                # generating a sequence of errors with markov-type source (c=t=1-p) and
information xor error
                z1 = plm3(z0, p1, p2, cte(p)); z2 = plm3(z1, p1, p2, cte(p)); z3 = plm3(z2, p1,
p2, cte(p)); z4 = plm3(z3, p1, p2, cte(p)); z5 = plm3(z4, p1, p2, cte(p)); z6 = plm3(z5, p1,
p2, cte(p))
                e0 = threshold_function(z0, cte(p)); e1 = threshold_function(z1, cte(p)); e2 =
threshold_function(z2, cte(p)); e3 = threshold_function(z3, cte(p)); e4 =
threshold_function(z4, cte(p)); e5 = threshold_function(z5, cte(p)); e6 =
threshold_function(z6, cte(p)) # error sequence
                r0 = b0 ^ e0; r1 = b1 ^ e1; r2 = b2 ^ e2; r3 = b3 ^ e3; r4 = b4 ^ e4; r5 = b5 ^
e5; r6 = b6 ^ e6 # received sequence
                z0 = plm3(z6, p1, p2, cte(p)) # prepare z0 for next loop

                # ecount the number of error bits (before decoding)
                berr0 = berr0 + e0 + e1 + e2 + e3 + e4 + e5 + e6

                # calculation of the syndrome
                s0 = r0 ^ r1 ^ r2 ^ r4; s1 = r0 ^ r1 ^ r3 ^ r5; s2 = r0 ^ r2 ^ r3 ^ r6

                # error-correcting based on the syndrome
                if ((s0 == 1) & (s1 == 1) & (s2 == 1)):
                    r0 = r0 ^ 1
                elif ((s0 == 1) & (s1 == 1) & (s2 == 0)):
                    r1 = r1 ^ 1
                elif ((s0 == 1) & (s1 == 0) & (s2 == 1)):
                    r2 = r2 ^ 1
                elif ((s0 == 0) & (s1 == 1) & (s2 == 1)):
                    r3 = r3 ^ 1
                elif ((s0 == 1) & (s1 == 0) & (s2 == 0)):
                    r4 = r4 ^ 1
                elif ((s0 == 0) & (s1 == 1) & (s2 == 0)):
                    r5 = r5 ^ 1
                elif ((s0 == 0) & (s1 == 0) & (s2 == 1)):
                    r6 = r6 ^ 1
                else:
                    None

                # count the number of incorrect decoding
                if ((r0 == b0) & (r1 == b1) & (r2 == b2) & (r3 == b3) & (r4 == b4) & (r5 == b5)
& (r6 == b6)):
                    ok += 1
                else:
                    blerr += 1
```

```
                # count the number of error bits (after decoding)
                berr = berr + (r0 ^ b0) + (r1 ^ b1) + (r2 ^ b2) + (r3 ^ b3) + (r4 ^ b4) + (r5 ^
b5) + (r6 ^ b6)

            # probability of incorect decoding
            incorrect_computed_value = blerr / l
            correct_theoretical_value = p1 / (p1 + p2) * p2 * (1 - p1)**5 + 5 * p2 / (p1 + p2)
* p1 * p2 * (1 - p1)**4 + p2 / (p1 + p2) * (1 - p1)**5 * p1 + p2 / (p1 + p2) * (1 - p1)**6
            incorrect_theoretical_value = 1 - correct_theoretical_value

            # probability of bit error (before and after decoding)
            bit_error_before = berr0 / (7 * l)
            bit_error_after = berr / (7 * l)

            # print the result
            print(f'For p: {p}, p1: {p1:.3f}, p2: {p2}\nINCORRECT DECODING; computed value:
{incorrect_computed_value:.5f} and theoretical value:
{incorrect_theoretical_value:.5f}\nPROBABILITY BIT ERROR; before: {bit_error_before:.5f} and
after: {bit_error_after:.5f}\n\n')

if __name__ == '__main__':
    print('\nNo. 1. Memoryless with bernoulli map\n')
    memoryless_bernoulli()
    print('\nNo. 2. Markov\n')
    markov()
```