

جامعة الإسكندرية
ALEXANDRIA
UNIVERSITY
كلية الهندسة



كلية الهندسة

Faculty of Engineering

Name

Amr Mohamed Mostafa Mostafa Awad

ID

6304

Grp

4

Sec

1



DIGITAL COMMUNICATIONS LAB

Experiment 2

Basics of BER calculations and channel models

Introduction

This is the first experiment in the Digital Communications Lab. In this experiment, we study the most basic method for assessing the performance of digital communication systems, the Bit *Error Rate*. We also study simple models for communication channels and their effect on the communication system from the perspective of bit error rate.

About the lab

The digital communications lab consists of a set of experiments which introduce essential digital communication concepts to students. The lab consists mainly of MATLAB-based experiments in which the students complete the required tasks of the experiment by writing MATLAB codes. Some experiments of the lab may consist of hardware-based experiments which are performed in the digital communication laboratory on campus.

Location of the digital communication laboratory on campus:

2nd floor, Electrical Engineering Building.

Instructions for completing the experiment



The following describes the procedure for completing MATLAB-based experiments in the digital communication lab. Other experiments that are not MATLAB-based may have different instructions which will be described in the DOCX documents of those experiments.

- An experiment consists of 1) a DOCX document explaining the experiment, 2) a link to a video explaining the experiment, and 3) the MATLAB files required to complete the experiment
- Each experiment typically consists of several parts.
- The DOCX document is your starting point. It explains in detail the requirement of each experiment part.
- The accompanying video explains the experiment. The explanation is mainly based on the experiment DOCX.
- The MATLAB files are used by the student to complete the experiment. The main MATLAB file for the experiment is usually named LabX_script.m with X being the experiment number. For example, the main file for Experiment 1 would be called Lab1_script.m. This is the first file that you should open and start working on. Inside that file, there can be referrals to other MATLAB files that you need to complete.
- Follow the instructions in the DOCX file and the explanation video to fill the provided MATLAB files. The experiment is completed once you generate the outcomes required in all parts of the experiment.

Lab submission and discussion

- This experiment is due on the weeks **from 26st of March to 7th of April.**
- Please bring a **printed copy of your codes** and **the lab DOCX with your answers in it** (your answers can be handwritten, but it is preferable to type them).

Lab policy regarding grading and cheating

Grading

- Each experiment is graded out of 20 marks. If you sum the total grades of the lab, it would be larger than 20. This means that you have a chance to miss some parts of the lab and still get a full mark.
 - o The grading is very lenient. Typically, it would be **very hard to grade that is less than 16.**
 - o Any student who does the whole lab by themselves are expected to get a grade of 18 or more, without even getting everything right.



Cheating

- In exchange for this grading policy, cheating will not be tolerated.
- If a student cheats in **any part in an experiment**, he/she will get **a zero for that entire lab**.
- If a student cheats in **two or more experiments**, he/she will get **a zero in the entire lab**.
- The reason behind this policy towards cheating is the following: detecting cheating cases requires an enormous amount of work by the instructors. Given how lenient the grading of the lab is expected to be, there is no justification for a student to cheat (this is in addition to the fact that there is no justification for cheating in general).

Note: it is okay to talk with your friends about the experiment if you feel that you are stuck somewhere. But please be aware that doing any of the following **is considered cheating, and all involved students in these cases are treated as cheaters**:

- A group of students write MATLAB codes together.
- A group of students discuss the experiment together, one student of the group writes the code, and all students use the same code.

Lab inventory

File name	Description
Lab video	Link: https://youtu.be/zBtf8aqwrJE
Lab1_BER.docx	Contains the details of the experiment.
Lab1_script.m	The main m-file containing the MATLAB script for the experiment.
GenerateBits.m	Additional MATLAB function files used by the experiment script file.
BSC.m	
ComputeBER.m	
DecodeBitsFromSamples.m	
GenerateSamples.m	

Background

The most basic target of the study of digital communications is to understand digital communication systems and how digital information can be conveyed from a source or transmitter to a destination or receiver over a channel. Depending on the communication systems, channels can be wired circuits, wireless channels, satellite channels and so on. The study of digital communications begins by transforming the digital



communication system into an equivalent mathematical model, and then attempts to design transmitters and receivers which achieves the target of information transmission over the channel in an efficient manner.

Figure 1 shows an example of a digital communication system. The goal of the transmitter and receiver is to deliver the digital data from the source to the sink in the best way possible. There are several ways to define what *best* mean: one of the most common and most important methods to assess the performance of a communication system is the *Bit Error Rate (BER)*.

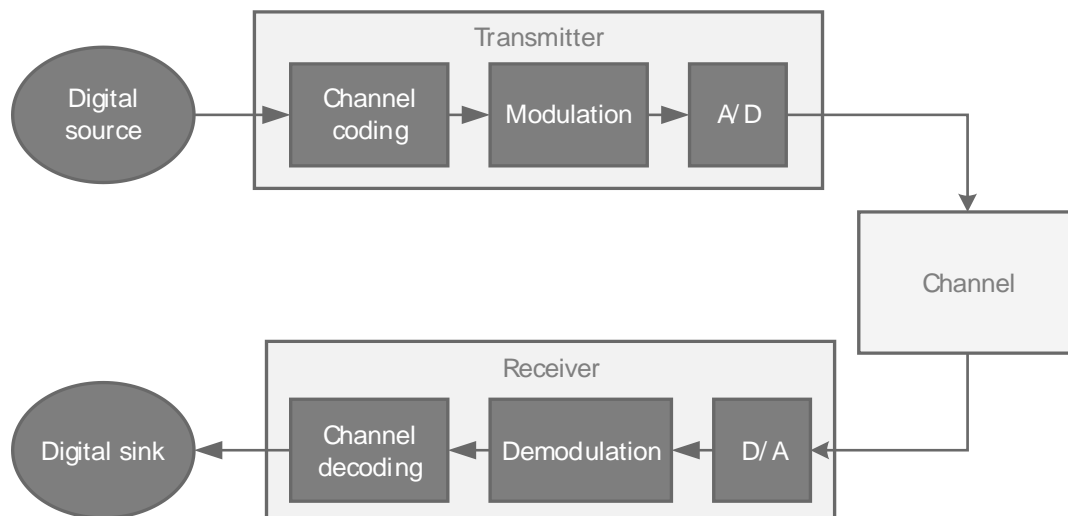


Figure 1 An example of a digital communication system

Bit Error Rate (BER): the rate of error occurrences among an output sequence of bits corresponding to an input sequence of bits.

An empirical method of computing the BER in a communication system is as follows:

1. Generate a sequence of N bits at the input side
2. Pass the sequence of input bits through the system to receive a corresponding output sequence
3. Count the number of errors in the output sequence by comparing it to the input sequence; call that number of errors E
4. The BER is given by $BER = \frac{E}{N}$

In this experiment, we will compute the BER of different digital communication systems. These systems differ in their respective channel models and therefore their corresponding transmitter and receiver designs.



Experiment

Part 1 (3 Marks)

In this part, we consider a very simple digital communication system, in which the channel takes as input binary digits $b = \{0,1\}$, and produces the corresponding output according to the following equation.

$$y = \{b \text{ with probability } 1 - p \text{ } \underline{b} \text{ with probability } p$$

The channel described above simply flips the input bit with probability p or passes the input bit unchanged with probability $1 - p$; this channel is referred to as the *Binary Symmetric Channel (BSC)*. The system is shown in Figure 2. In this system, we assume that the transmitter takes the input bits coming from the source and passes them unchanged to the channel (i.e., the transmitter does nothing). However, we would like to investigate how the receiver can be designed to produce a good BER.

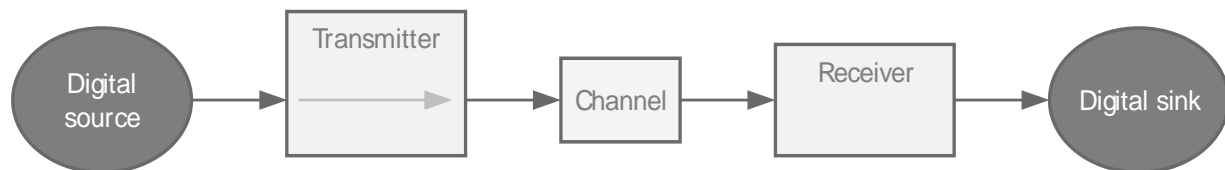


Figure 2 A digital communication system with a Binary Symmetric Channel

Your goal in this task is to design the receiver. You know that the channel takes the data, flips it randomly (with probability p) and gives you the output. What would the receiver do with that output?

Think about the following two receivers and say what is the expected performance of these receivers. As a hint to start, these two receivers are not very good.

Example 1: the receiver gives a 0 bit as output. This output does not depend at all on what the channel is giving out.

Questions	
What is the corresponding BER for that receiver? You do not need to implement it in the m-file to answer.	BER will depend on the input and p factor so if input is 0's then BER = 0, if the input is 1's then BER = 1 otherwise it will depend on p BER will nearly be 0.5 on average
What is the reason behind the performance of this receiver?	Because this receiver doesn't depend on the channel so the message will be destroyed



Example 2: the receiver gives random output, i.e., 0s and 1s with a probability of 0.5. Again, this output is not based on what the channel is giving out.

Questions	
What is the corresponding BER for that receiver? You do not need to implement it in the m-file to answer.	BER will range from 0 to 1 depending on the input sequence BER will nearly be 0.5 on average
What is the reason behind the performance of this receiver?	Because this receiver doesn't depend on the channel and message will be destroyed

The above two receivers are examples of receivers which clearly would not be considered as good receivers from a BER perspective (why?). In the following part of the experiment, you would design the best receiver and assess its performance by computing the corresponding BER.

EXP. Complete PART 1 in the experiment M-file Lab1_script.m and the missing implementation of all included functions. Then answer the following questions:

Questions	
What is the corresponding BER for receivers 1 and 2 above? You do not need to implement the two receivers to answer.	BER for receivers 1 and 2 will be nearly equal to 0.5 on average
What is the reason behind the performance of these two receivers?	Because this receiver doesn't depend on the channel and message will be destroyed
What is the BER of the best receiver?	If all bits were received without error, then best BER will be equal to zero. while at $p=0.2$ BER = 0.2

Part 1-a (2 Marks) In this part, we study the impact of the BSC channel parameter p on the BER of the digital communication system. Namely, we vary the value of p from 0 to 1, and for each value of p we compute the corresponding BER, we save these values in an array, then, later on in Part 3-a, plot the values of BER versus their corresponding parameter value p .

EXP. Complete PART 1-a in the experiment M-file Lab1_script.m. The final figure containing the required plot will be generated at the end of Part 3-a of the experiment.



Part 2 (3 Marks)

In this part, we again consider the system proposed in Figure 2 but we try to improve the transmitter a bit. Namely, the transmitter works as follows: for each input bit b , the transmitter generates a set of 5 copies of the bit b which are then passed sequentially through the channel. Note that this behavior leads to the increase in the number of bits being passed through the channel (is that good or bad?). The system is shown in Figure 3. For this transmitter, the receiver expects to receive a sequence of 5 channel outputs, all corresponding to the same input bit. Therefore, we expect that the receiver can use these outputs for a better decoding performance. In this part, we investigate how to design the best receiver and the corresponding BER performance.

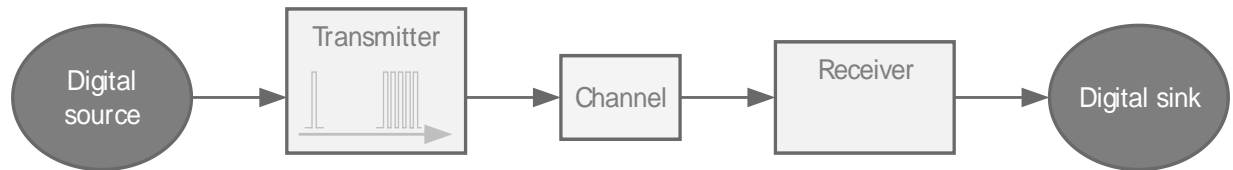


Figure 3 A digital communication system with a Binary Symmetric Channel and a modified transmitter

EXP. Complete PART 2 in the experiment M-file Lab1_script.m and the missing implementation of all included functions. Then answer the following questions:

Questions	
What is the BER of the best receiver?	BER will be nearly equal to 0.05 at $p = 0.2$, $f_s=5$
What is the expected (theoretical) BER if the number of repetitions is increase to 10?	BER will decrease when increasing number of repetitions BER is nearly equal to 0.02
What is the cost/downside of using the transmitter in Part 2?	Wasted resources on duplicate bits which lead to increase in cost

Part 2-a (2 Marks)

Similar to Part 1-a, in this part, we study the impact of the BSC channel parameter p on the BER of the digital communication system in Part 2.

EXP. Complete PART 2-a in the experiment M-file Lab1_script.m. The final figure containing the required plot will be generated at the end of Part 3-a of the experiment.

Part 3 (3 Marks)

In part 3, we consider the same system in Part 2. However, the channel in Part 3 generates correlated outputs among the 5 transmitter outputs that correspond to the same input bit. For example, for a 0 input bit to the transmitter and a corresponding five copies of the bit 0, the channel output either generates a set of



five 0's with probability $1 - p$ or a set of five 1's with probability p . In this case, we investigate the design of the best receiver and the corresponding BER.

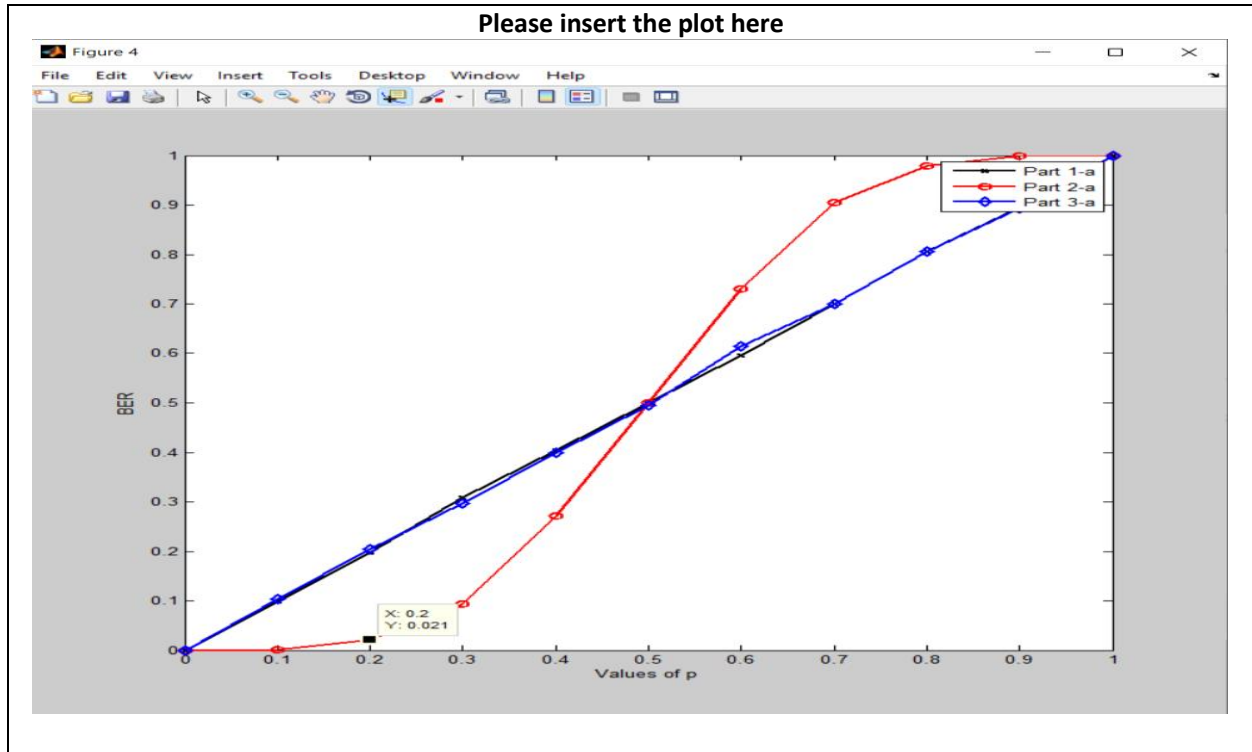
EXP. Complete PART 3 in the experiment M-file Lab1_script.m and the missing implementation of all included functions. Then answer the following questions:

Questions	
What is the BER of the best receiver?	BER will be nearly equal to 0.2 at $p=0.2$
What is the reason behind such a performance?	The reason is that it depends on the first bit only

Part 3-a (2 Marks)

Finally, we study the impact of the BSC channel parameter p on the BER of the digital communication system in Part 3.

EXP. Complete PART 3-a in the experiment M-file Lab1_script.m. The final figure containing the plots from all three parts can now be generated. Please add the generated plot in the box below.





Questions	
Which of the three systems have the best performance in terms of BER?	System 2
If the receiver you designed in any of the previous parts attain a BER more than 0.5, how can it be changed to attain a maximum of 0.5 BER?	By changing the value of p or f_s (Decrease p or increase f_s)

Part 4 (8 Marks)

In this part, we go back to the system considered in Part 2, namely the system with a transmitter which generated a set of 5 repetitions to the input bit. Now, we would like to investigate the effect of changing the number of repetitions on the decoding performance. You need to generate a figure where the x-axis shows the number of repetitions, and the y-axis shows the corresponding BER. In this part, you can consider $p = 2$.

EXP. Write your own code in PART 4 in the experiment M-file Lab1_script.m. Your code should generate a figure as described in the discussion above.

Lab1_script.m:

```
%%  
% Alexandria University  
% Faculty of Engineering  
% Electrical and Electronic Engineering Department  
%  
% Course: Digital Communications Lab  
%  
% Lab No. 1: Basics of BER calculation and channel modeling  
  
%% Simulation parameters  
  
N_bits = 10000; % Total number of bits  
p = 0.2; % Channel parameter (probability of bit flipping)  
  
%% Part 1: BER for simple BSC channel  
  
% Generate a bit sequence  
bit_seq = GenerateBits(N_bits); % IMPLEMENT THIS: Generate a sequence of bits equal to the total  
number of bits  
  
% Pass the bit sequence through the channel  
rec_sample_seq = BSC(bit_seq,1,p); % Generate the received samples after passing through the bit  
flipping channel  
  
% Decode bits from received bit sequence  
rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,'part_1'); % IMPLEMENT THIS: Decode the  
received bits
```



```
% Compute the BER
BER_case_1 = ComputeBER(bit_seq,rec_bit_seq); % IMPLEMENT THIS: Calculate the bit error rate

%% Part 1-a: Effect of bit flipping probability on BER
% GOAL: Make a plot for the BER versus different values of the channel
% parameter p

p_vect      = 0:0.1:1;          % Use this vector to extract different values of p in
your code
BER_case_1_vec = zeros(size(p_vect)); % Use this vector to store the resultant BER

%%% WRITE YOUR CODE HERE
for p_ind = 1:length(p_vect)
    rec_sample_seq = BSC(bit_seq,1,p_vect(p_ind));
    rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,'part_1');
    BER_case_1_vec(p_ind) = ComputeBER(bit_seq,rec_bit_seq);
end
figure;
plot(p_vect,BER_case_1_vec);
title('BER 1 vs. P ')
ylabel('BER 1')
xlabel('P')
%%%

%% Part 2: BER for simple bit-flipping channel with multiple samples

% System parameters
fs = 10; % Number of samples per symbol (bit)

% Generate a bit sequence
bit_seq = GenerateBits(N_bits); % Generate a sequence of bits equal to the total number of bits

% Generate samples from bits
sample_seq = GenerateSamples(bit_seq,fs); % IMPLEMENT THIS: Generate a sequence of samples for
each bit

% Pass the sample sequence through the channel
rec_sample_seq = BSC(sample_seq,fs,p); % Generate the received samples after passing through
the bit flipping channel

% Decode bits from received bit sequence
rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,'part_2',fs); % IMPLEMENT THIS: Decode the
received bits

% Compute the BER
BER_case_2 = ComputeBER(bit_seq,rec_bit_seq); % Calculate the bit error rate

%% Part 2-a: Effect of bit flipping probability on BER
% GOAL: Make a plot for the BER versus different values of the channel
% parameter p

p_vect      = 0:0.1:1;          % Use this vector to extract different values of p in
your code
BER_case_2_vec = zeros(size(p_vect)); % Use this vector to store the resultant BER

%%% WRITE YOUR CODE HERE
for p_ind = 1:length(p_vect)
    rec_sample_seq = BSC(sample_seq,fs,p_vect(p_ind));
    rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,'part_2',fs);
    BER_case_2_vec(p_ind) = ComputeBER(bit_seq,rec_bit_seq);
end
figure;
```



```
plot(p_vect,BER_case_2_vec);
title('BER 2 vs. P ')
ylabel('BER 2')
xlabel('P')
%%%

%% Part 3: BER for simple bit-flipping channel with multiple samples and correlated channel

% Generate a bit sequence
bit_seq = GenerateBits(N_bits); % Generate a sequence of bits equal to the total number of bits

% Generate samples from bits
sample_seq = GenerateSamples(bit_seq,fs); % Generate a sequence of samples for each bit

% Pass the sample sequence through the channel
rec_sample_seq = BSC(sample_seq,fs,p,'correlated'); % Generate the received samples after
passing through the bit flipping channel

% Decode bits from received bit sequence
rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,'part_3',fs); % IMPLEMENT THIS: Decode the
received bits

% Compute the BER
BER_case_3 = ComputeBER(bit_seq,rec_bit_seq); % Calculate the bit error rate

%% Part 3-a: Effect of bit flipping probability on BER
% GOAL: Make a plot for the BER versus different values of the channel
% parameter p

p_vect = 0:0.1:1; % Use this vector to extract different values of p in
your code
BER_case_3_vec = zeros(size(p_vect)); % Use this vector to store the resultant BER

%%% WRITE YOUR CODE HERE
for p_ind = 1:length(p_vect)
    rec_sample_seq = BSC(sample_seq,fs,p_vect(p_ind),'correlated');
    rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,'part_3',fs);
    BER_case_3_vec(p_ind) = ComputeBER(bit_seq,rec_bit_seq);
end
%%%
figure;
plot(p_vect,BER_case_3_vec);
title('BER 3 vs. P ')
ylabel('BER 3')
xlabel('P')
% Plotting results

figure
plot(p_vect,BER_case_1_vec,'x-k','linewidth',2); hold on;
plot(p_vect,BER_case_2_vec,'o-r','linewidth',2); hold on;
plot(p_vect,BER_case_3_vec,'d-b','linewidth',2); hold on;

xlabel('Values of p','fontsize',10)
ylabel('BER','fontsize',10)
legend('Part 1-a','Part 2-a','Part 3-a','fontsize',10)
%% Part 4: Effect of number of repetitions on BER
% GOAL: Make a plot for the BER versus the number of repetitions used in
% the transmitter of part 2
% There is no template code for this part. Please write your own complete
% code here. You can re-use any of the codes in the previous parts

%%% WRITE YOUR CODE HERE
```



```
fs_vect      = 1:10;                                % Use this vector to extract different values of fs in your
code
BER_case_4   = zeros(size(fs_vect)); % Use this vector to store the resultant BER

for ind = 1:length(fs_vect)
    % Generate a bit sequence
    bit_seq = GenerateBits(N_bits); % Generate a sequence of bits equal to the total number of
bits

    % Generate samples from bits
    sample_seq = GenerateSamples(bit_seq,fs_vect(ind)); % IMPLEMENT THIS: Generate a sequence of
samples for each bit

    % Pass the sample sequence through the channel
    rec_sample_seq = BSC(sample_seq,fs_vect(ind),p); % Generate the received samples after
passing through the bit flipping channel

    % Decode bits from received bit sequence
    rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,'part_2',fs_vect(ind)); % IMPLEMENT
THIS: Decode the received bits

    % Compute the BER
    BER_case_4(ind) = ComputeBER(bit_seq,rec_bit_seq); % Calculate the bit error rate
end
figure
plot(fs_vect,BER_case_4)
title('BER 4 vs. Fs')
ylabel('BER 4')
xlabel('Fs')
```

ComputeBER.m

```
function BER = ComputeBER(bit_seq,rec_bit_seq)
%
% Inputs:
%   bit_seq:      The input bit sequence
%   rec_bit_seq:  The output bit sequence
% Outputs:
%   BER:          Computed BER
%
% This function takes the input and output bit sequences and computes the
% BER

%%% WRITE YOUR CODE HERE
[~,BER] = biterr(bit_seq,rec_bit_seq); %comparing with original bits
%%%
```



BSC.m

```
function rec_sample_seq = BSC(sample_seq,fs,p,channel_type)
%
% Inputs:
% sample_seq: The input sample sequence to the channel
% fs: The sampling frequency used to generate the sample
sequence
% p: The bit flipping probability
% channel_type: The type of channel, 'independent' or 'correlated'
% Outputs:
% rec_sample_seq: The sequence of sample sequence after passing through the
channel
%
% This function takes the sample sequence passing through the channel, and
% generates the output sample sequence based on the specified channel type
% and parameters

sample_seq = ~~sample_seq;
rec_sample_seq = zeros(size(sample_seq));
rec_sample_seq = ~~rec_sample_seq;

if (nargin <= 3)
    channel_type = 'independent';
end

switch channel_type

    case 'independent'
        channel_effect = rand(size(rec_sample_seq))<=p;
    case 'correlated'
        channel_effect = rand(1,length(rec_sample_seq)/fs)<=p;
        channel_effect = repmat(channel_effect,fs,1);
        channel_effect = channel_effect(:)';
    end

rec_sample_seq = xor(sample_seq,channel_effect);
rec_sample_seq = rec_sample_seq + 0;
```



DecodeBitsFromSamples.m

```
function rec_bit_seq = DecodeBitsFromSamples(rec_sample_seq,case_type,fs)
%
% Inputs:
%   rec_sample_seq: The input sample sequence to the channel
%   case_type:      The sampling frequency used to generate the sample
sequence
%   fs:             The bit flipping probability
% Outputs:
%   rec_sample_seq: The sequence of sample sequence after passing through the
channel
%
% This function takes the sample sequence after passing through the
% channel, and decodes from it the sequence of bits based on the considered
% case and the sampling frequency

if (nargin <= 2)
    fs = 1;
end

switch case_type

    case 'part_1'
        %%% WRITE YOUR CODE FOR PART 1 HERE
        %   if (fs<0.5)
        %       rec_bit_seq = rec_sample_seq;
        %   else
        %       rec_bit_seq=~rec_sample_seq;
        %       %for i=1:length(rec_sample_seq)
        %           %   rec_bit_seq(i) = ~rec_sample_seq(i);
        %       %end
        %   end
        %rec_bit_seq = zeros(1,10000);
        %rec_bit_seq = randi([0, 1], 1,10000);
        %%%
    case 'part_2'
        %%% WRITE YOUR CODE FOR PART 2 HERE
        counter=0;
        rec_bit_seq=[];
        for i=1:length(rec_sample_seq)
            if (rec_sample_seq(i)==0)
                counter=counter+1;
            end

            if (mod(i,fs)==0)
                if(counter>fix(fs/2))
                    rec_bit_seq(end+1)=0;
                else
```



```
        rec_bit_seq(end+1)=1;
    end
    counter=0;
end
end
%%%
case 'part_3'
    %%% WRITE YOUR CODE FOR PART 3 HERE
    rec_bit_seq=[];
    for i=1:fs:length(rec_sample_seq)
        rec_bit_seq(end+1)=(rec_sample_seq(i));
    end
    %%%
end
```

GenerateBits.m

```
function bit_seq = GenerateBits(N_bits)
%
% Inputs:
%   N_bits:      Number of bits in the sequence
% Outputs:
%   bit_seq:     The sequence of generated bits
%
% This function generates a sequence of bits with length equal to N_bits

bit_seq = zeros(1,N_bits);
%%% WRITE YOUR CODE HERE
%bit_seq = randi([0, 1], 1,N_bits);
for i=1:N_bits
    bit_seq(i) = randi([0, 1]);
end
%%%
```




GenerateSamples.m

```
function sample_seq = GenerateSamples(bit_seq,fs)
%
% Inputs:
%   bit_seq:    Input bit sequence
%   fs:         Number of samples per bit
% Outputs:
%   sample_seq: The resultant sequence of samples
%
% This function takes a sequence of bits and generates a sequence of
% samples as per the input number of samples per bit

sample_seq = zeros(size(bit_seq*fs));

%%% WRITE YOUR CODE FOR PART 2 HERE
x=1;
for i=1:length(bit_seq)
    tempVar=bit_seq(i);
    for j=x:x+fs-1
        sample_seq(j)=tempVar;
    end
    x=x+fs;
end
%%%
```



Outputs:

