# ECE F344: Info Theory and Coding

SECOND SEMESTER 2021-22


## Assignment - III

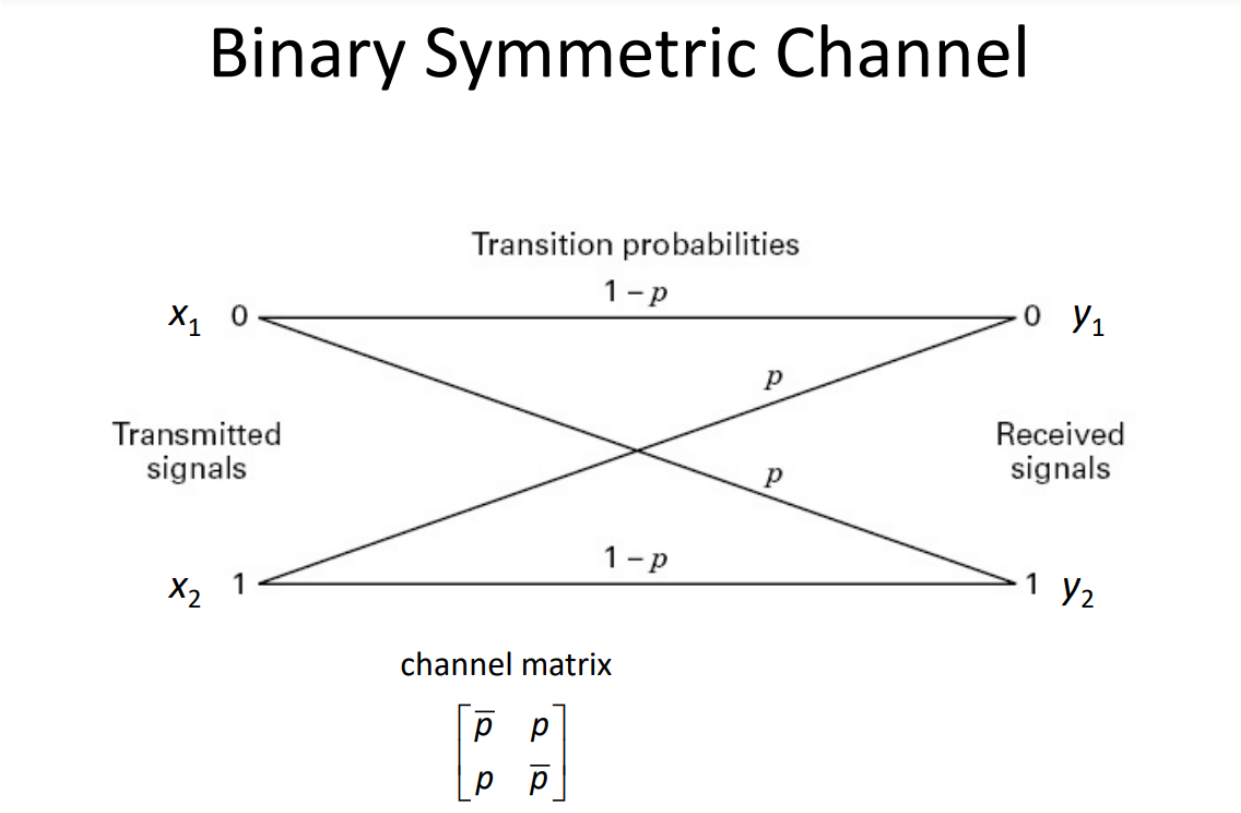
DATE: 26/04/2022

SUBMITTED BY
Parth Aggarwal
2019AAPS0218H

**For the given text file, implement a source and channel encoder-decoder pair (of your choice) and transmit the text in the file through a binary symmetric channel with an error probability p = 0.25. Describe the performance of your implemented system.**

**Theory definition (along with equations if required):**

**Binary Symmetric Channel:** A binary symmetric channel (or $BSC_p$) is a common communications channel model used in coding theory and information theory. In this model, a transmitter wishes to send a bit (a zero or a one), and the receiver will receive a bit. The bit will be "flipped" with a "crossover probability" of *p*, and otherwise is received correctly. This model can be applied to varied communication channels such as telephone lines or disk drive storage. Here the error probability is given as 0.7.




**Huffman coding** is a lossless data compression algorithm. The idea is to assign variable-length codes to input characters, lengths of the assigned codes are based on the frequencies of corresponding characters. The most frequent character gets the smallest code, and the least frequent character gets the largest code.

**Command Used:**

**main_code (the driver program) -**
disp()
fopen()
fread()
bsc()
fclose()
sum()
abs()
num2str()

**source_statistics -**
unique()
histc()
length()
sort()

**huffman_encoding -**
length()
cell()
fliplr()
zeros()
min()
strcat()

**stream_generator -**
char()
length()
strfind()
double()

**convulational_coding -**
conv2()
rem()
size()
reshape()

**viterbi_decoder -**
size()

de2bi()
zeros()
bsxfun()
sum()
rem()
length()
reshape()
ismember()
find()
abs()
min()

**huffman_decoding -**
cellfun()
min()
length()
char()
strcmp()
find()
isempty()

## Approach/Algorithm

The major approach followed in our code was in the below order.
code-> Huffman encoding -> channel encoding -> BSC -> channel decoding -> Huffman decoding.

1. Firstly the probabilities of each alphabet were calculated from the given text.
2. Then each of the alphabets was encoded by Huffman encoding codes according to the probabilities calculated in step 1.
3. The code was then channel encoded through convolution coding.
4. The channel encoded data was then passed through BSC with the error probability of 0.25.
5. The output of BSC was received at the receiver's end and was channel decoded through Viterbi decoding.
6. The output of the channel decoder was then decoded by Huffman decoding and the required data was regenerated.

# Code and corresponding output

## Main Function

```
26 -    input = bit_stream;
27
28      %      Channel Coding
29 -    disp('Channel coding: ');
30 -    channel_coded = convolutional_coding(bit_stream, Generator)
31
32      %      BSC Channel
33 -    disp('BSC: ');
34 -    ndata = bsc(channel_coded,0.25);
35
36      %     Channel Decoding
37 -    disp('Channel decoding: ');
38 -    bit_stream = viterbi_decoder(ndata, Generator, shift);
39
40 -    output = bit_stream;
41
42      %      Huffman Decoding
43 -    disp('Huffman decoding: ');
44 -    decoded_msg = huffman_decoding(unique_symbol, code_word, bit_stream);
45
```

## Huffman Encoding

```
10 -    if n == 1
11 -        code_word{1} = '1';
12 -    end
13 -    x = zeros(n, n);
14 -    x(:, 1) = (1:n)';
15
16 -    for i = 1:n-1
17 -        temp = prob;
18 -        [~, min1] = min(temp);
19 -        temp(min1) = 1;
20 -        [~, min2] = min(temp);
21 -        prob(min1) = prob(min1) + prob(min2);
22 -        prob(min2) = 1;
23 -        x(:, i+1) = x(:, i);
24 -        for j = 1:n
25 -            if x(j, i+1) == min1
26 -                code_word(j) = strcat('0', code_word(j));
27 -            elseif x(j, i+1) == min2
28 -                x(j, i+1) = min1;
29 -                code_word(j) = strcat('1', code_word(j));
```

```
Columns 1 through 14

  {'001'}    {'1101'}    {'1100'}    {'1010'}    {'1001'}    {'0111'}    {'0110'}    {'0100'}    {'0001'}    {'11111'}    {'11110'}    {'11101'}    {'11100'}    {'10110'}

Columns 15 through 26

  {'10001'}    {'01011'}    {'01010'}    {'00001'}    {'101111'}    {'100001'}    {'100000'}    {'000001'}    {'000000'}    {'1011100'}    {'10111011'}    {'101110101'}
```

## Convolutional encoding

```
function y = convolutional_coding(bit_stream, G)

y = conv2(bit_stream, G);
y = rem(y, 2);
[row, col] = size(y);
y = reshape (y, 1, row * col);
end
```

Command Window

```
    1     1     0     1     0     1     1     1     0     0     1     1     0     1     0     1     0     0

Columns 12,241 through 12,260

    1     0     0     1     0     0     1     0     1     1     0     0     0     0     1     1     0     1
```

## Viterbi decoding

```
function decoded_stream = viterbi_decoder(y, G, k)

[row, col] = size(G);
l = col;    % l = constraint length
n = row;    % n = output size

states = de2bi(0:2^(l-k)-1, (l-k), 'left-msb');

% output for zero and one contains possible output for each state for input
% zero and one
output_for_zero = zeros(2^n, n);
output_for_one = zeros(2^n, n);

for i = 1:size(states, 1)
    output = sum(bsxfun(@times, G, [0 states(i, :)]), 2);
    output_for_zero(i, :) = output';
    output = sum(bsxfun(@times, G, [1 states(i, :)]), 2);
    output_for_one(i, :) = output';
end
```

```
BSC:
Channel decoding:

output =

  Columns 1 through 29

    1     1     0     1     1     1     1     1     0     1     0     0     0     1     0     0     0     0     1     1     0     0     0     0     0     0     0     1     1

  Columns 30 through 58

    0     1     1     0     0     0     0     0     0     1     1     0     0     0     0     0     1     1     1     1     0     1     0     1     1     1     1     1     1

  Columns 59 through 87
```

## Huffman decoding

```
function decoded_msg = huffman_decoding(unique_symbol, code_word, bit_stream)


decoded_msg = [];

% minimum code word length
i_min = min(cellfun('length', code_word));

% bit stream pointer
ptr = 1;
for i = i_min:length(bit_stream)
    if isempty(find(strcmp(code_word, char(bit_stream(ptr:i) + '0')), 1)) ~= 1
        ind = find(strcmp(code_word, char(bit_stream(ptr:i) + '0')), 1);
        decoded_msg = [decoded_msg char(unique_symbol(ind))];
        ptr = i + 1;
        i = i + i_min;
    end
end
end
```

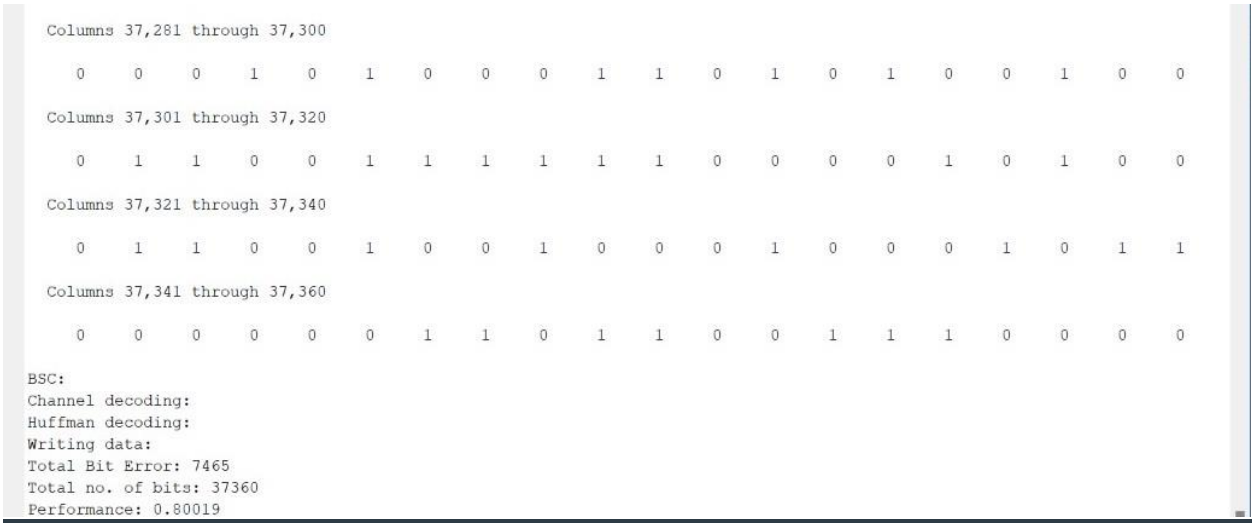
```
Huffman decoding:

decoded_msg =

    'sdflm oimdvsoiiwe kmdsflkjew nnijpojewnmk msdfkjweti kjoidfug ewruhweorhn kveqakwj kjweroijvdsn kjnjo ewhjndsf eowirndfs ejwroiwjefng ewrwet khdsfue nsdpowen mvwi slgj oewi
```

# Performance

```
%      Error Calculation
Error = sum(abs(input - output));
disp(['Total Bit Error: ' num2str(Error)]);
disp(['Total no. of bits: ' num2str(d)]);
p= Error/d;
disp('Performance:');
disp(1-p);
```




Total no. of error bits = 7465
Total no. of bits = 37360
Performance = 80.019%

The received data after the transmission is given in the zip file.

## Conclusion:

This assignment helped us to culminate every step of data transmission and data reception and understand the complete process of communication from source coding to detecting errors and decoding the data.