Analysis and simulation of a digital transmission system

Alessandro Trigolo

2023/2024

Index

Todo list

Introduction

Finish introduction

This document analyses and simulates the behavior of a digital transmission system. The document is split into two main parts:

i **Analysis.** `Raw results` section

ii **Simulation.**

You can find the full project at `imAlessas/transmission-simulation.git`.

General schematic

The full schematic - containing every step - of a transmission system is presented in figure 1. Before exploring the mathematical background hidden between the steps, it is crucial to understand what every phase of the system means.

◇ *Source.* The source device is whichever device is sending a signal; it could be a television, a computer, a smartphone, or anything else.

◇ *Formatting Device.* The formatting device's task is to translate the information from analogic to digital which translates into sampling the continuous analogic signal and creating a discrete digital signal that can be transmitted through digital devices.

◇ *Source Coding.* The source coding goal is lossless data compression. Sure enough, through the Shannon-Fano source coding, the symbols transmitted are encoded to reduce the average codeword length.

◇ *Channel Coding.*

◇ *Interleaving.* The interleaver is needed to transform package errors into independent errors. This is achieved by changing the ordering of the symbols that will be transmitted.

◇ *Scrambling.* The scrambling procedure helps with the synchronization between the two devices and improves the security of the transmission. This is achieved by adding a *pseudo-random sequence* to the symbols before the transmission.

◇ *Modulation.* The modulation process' goal is to match the spectrum of the transmitted signal with the transmission channel bandwidth making the signal more noise-immune and increasing the data-transfer rate; these operations are performed by the modulator. There are different types of modulation, the one utilized in this project is the *Binary Phase Shift Keying*, which is one of the most effective modulations against noise.

◇ *Noise.* The noise is a crucial obstacle to overcome to have a successful transmission; the noise is the main reason for a wrongly transmitted symbol. There are different types of noise, some of them are generated by other transmissions, others are due to the physical medium and others are caused by the intermediate devices between the transmission. Nevertheless, in every transmission, there will be the *Gaussiam White Noise* which is a thermal noise caused by the Big Bang.

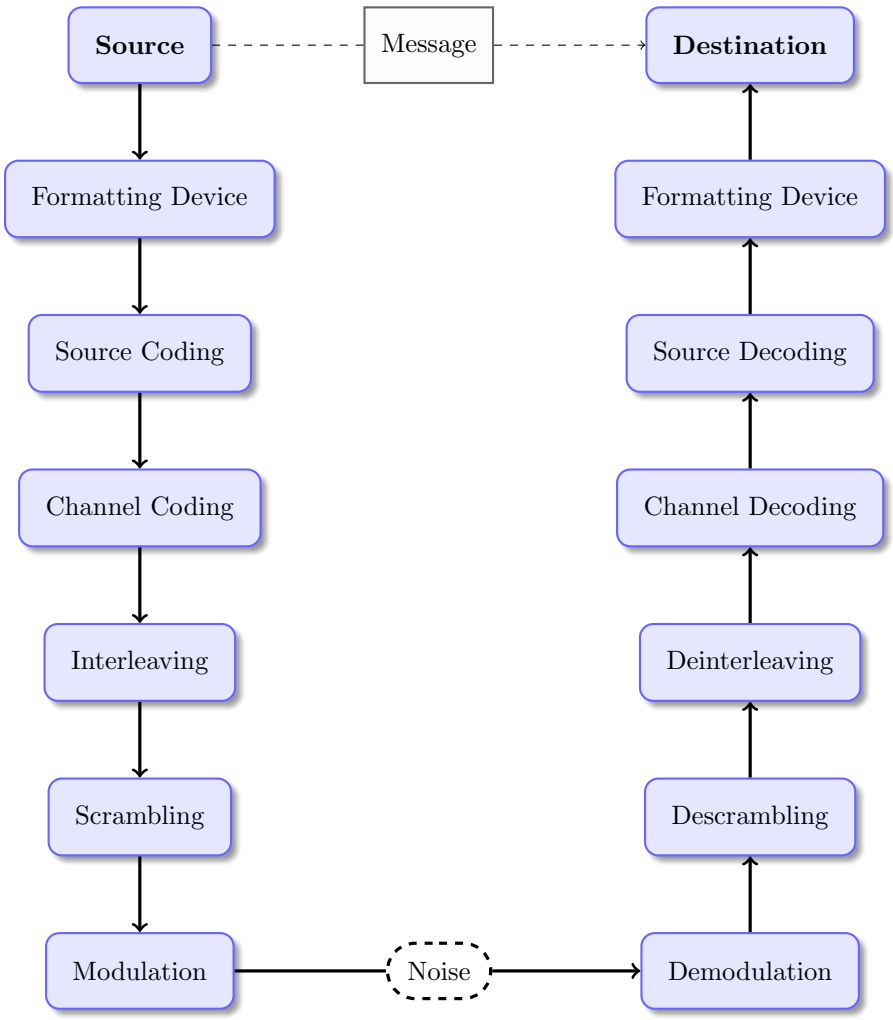

Figure 1: The diagram of the digital information transmission system.

- ◇ *Demodulation.* In this phase the demodulator device, after receiving the disturbed signal, will try to detect the signal to regenerate the original one. Sometimes the noise energy will be stronger than the signal energy generating errors that will be corrected in the next steps.
- ◇ *Descrambling.* The descrambling procedure is the opposite of the scrambling. The added *pseudo-random sequence*, after the reception is subtracted by the descrambler.
- ◇ *Deinterleaving.* The deinterleaver reorders the transmitted symbols in the opposite way that the interleaver did. In such a way the *burst* errors that occurred during the transmission will become single errors that can be easily recovered.
- ◇ *Channel Decoding.*

◇ *Source Decoding.* The source decoding procedure decompresses the received data into the original symbols. This is achieved by one of the source coding properties: symbols are easily detected because there are no shorter codes at the beginning of longer codes.

◇ *Formatting Device.* During the transmission this device converts the signal from analogic to digital, during the reception of the signal the formatting device translates the discrete digital signal into a continuous analogic signal.

◇ *Destination.* The destination device is whichever device will receive the signal. Likewise the source one, the destination device could be a satellite, a smartphone, a server, or anything else.

Initial parameters

The parameters used in this project have been assigned in a datasheet and are reported in the following list:

- *Symbol duration*: 60 ns, also called τ;
- *SNR*: 8.1 dB;
- *Source code*: Shannon-Fano coding;
- *Error correction code*: cyclic coding with codeword length $m = 31$ and generator polynomial $z^5 \oplus z^2 \oplus 1$;
- *Carrier frequency*: 2.5 GHz;
- *Modulation*: Binary Amplitude Shift Keying (BPSK) with the phase shift of π.

In addition, the source data (alphabet) and the symbols' respective probabilities are summarized in the following table.

Source 7	
a_1	0.11
a_2	0.07
a_3	0.09
a_4	0.01
a_5	0.06
a_6	0.06
a_7	0.13
a_8	0.14
a_9	0.13
a_{10}	0.05
a_{11}	0.11
a_{12}	0.04

Afterwards the parameters have been transcripted in the MATLAB program, as shown in the following snippet.

```
% source number 7
ALPHABET = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12];
PROBABILITY_VECTOR = [11, 7, 9, 1, 6, 6, 13, 14, 13, 5, 11, 4]/100;

```

```
TAU = 60e-9; % symbol duration time, [s]
SNR = 8.1; % Signal-to-Noise-Ration, [dB]
% Source Code: Shannon-Fano
% Error correction code: Cyclic
CODEWORD_LENGTH = 31; % m
%
F_0 = 2.5e+9; % carrier frequency [Hz]
% Modulation: BPSK
PHASE_SHIFT = pi; % [rad]
U = 1; % amplitude BPSK signal [V]

transmitted_symbol_number = 20;
```

Part I
Analysis

1 Source data

The source data analysis provides a general overview of how the data are generated and how this will impact the encoding scheme. Specifically, the source data analysis is achieved by calculating two important values: the source entropy and the redundancy coefficient.

Source entropy

The source entropy H and the maximum source entropy H_{max}. The entropy of a sequence of symbols is a number that summarizes the randomness of the selection of the symbols in the source sequence. The more uncertain the symbols are, the higher the entropy is and the higher the information the symbols carry. The ideal entropy is when the source symbols are `1 0 1 0 1 0 ...` while the worst entropy is when all the symbols are `1` or `0`. Given a sequence S of N symbols, where each of them has its probability P_i to occur, the entropy of the sequence is:

$$H(S) = -\sum_{i=1}^{N} P_i \log_2 P_i$$

The entropy calculation can be simply achieved with the following MATLAB code. The only thing to note is that P is the probability vector that assigns to every symbol of the alphabet its probability.

```
% sum(V .* log2(V))
H = - dot(PROBABILITY_VECTOR, log2(PROBABILITY_VECTOR));
```

Secondly, in order to calculate the maximum entropy H_{max}, two conditions have to be met: all of the symbols have the same probability $P_i = \frac{1}{N}$ and, of course, they do not correlate one another. Consequently:

$$H_{max}(S) = -\sum_{i=1}^{N} \frac{1}{N} \log_2 \frac{1}{N} = \frac{1}{N} \sum_{i=1}^{N} \log_2 N = \log_2 N$$

Also in this case the MATLAB script to calculate the maximum entropy is trivial.

```
% Number of symbols in the alphabet
N  = length(PROBABILITY_VECTOR);

% Maximum source entropy
H_max = log2(N);
```

By running the scripts, the value obtained are $H = 3.3995$ while $H_{max} = 3.5850$. Reasonably $H < H_{max}$ because the given probabilities in the datasheet weren't equal to each other.

Redundancy coefficient

The redundancy coefficient ρ summarizes in a number how much additional information is present inside the sequence. Essentially, the lower the redundancy coefficient is, the better, because it means that the source entropy is very high. Mathematically, the coefficient ρ can be obtained as follows:

$$\rho = 1 - \frac{H}{H_{max}}(S)$$

which translates into the following code snippet:

```
% Calculate the redundancy coefficient 'rho'
source_redoundancy = 1 - H/H_max;
```

Expectedly, the redundancy coefficient is not zero because $H < H_{max}$: by running the script, $\rho = 0.0517$.

2 Source encoding

The source coding analysis provides the necessary tools to evaluate the source coding algorithms for efficient data representation and compression. In this case, the analysis calculates and uses different values to provide a better understanding of the efficiency of the Shannon-Fano source coding. Particularly the values that will be analyzed are the average codeword length $\overline{m}$, the probability of `1` and `0` (P_1 and P_0), the binary entropy H_{bin}, the source data generation rate R and the compression ratio K.

Shannon-Fano algorithm

Before calculating the values it is important to encode the symbols of the alphabet through the Shannon-Fano algorithm. A brief recursive description of it is reported below.

1. Sort the symbol of the alphabet by descending probability;
2. Divide the sets of symbols into two continuous subsets with the same probability (or the lowest difference between the two);
3. Assign to one subset the symbol 1 and the other 0;
4. Repeat until every subset consists of one symbol;
5. Read the codeword from left to right.

By applying the Shannon-Fano algorithm to the given source, the result should be the following.

S	P	Shannon-Fano					Code	m	m_0	m_1
a_8	0.14	1	1			1	111	3	0	3
a_7	0.13	1	0				110	3	1	2
a_9	0.13		1				101	3	1	2
a_1	0.11		0				100	3	2	1
a_{11}	0.11	0	1				011	3	1	2
a_3	0.09		1	0	1		0101	4	2	2
a_2	0.07				0		0100	4	3	1
a_5	0.06		0	1	1		0011	4	2	2
a_6	0.06				0		0010	4	3	1
a_{10}	0.05			0	1		0001	4	3	1
a_{12}	0.04			0	0	1	00001	5	4	1
a_4	0.01					0	00000	5	5	0

After computing the Shannon-Fano algorithm to the given source, the results should be inserted into the MATLAB program, as follows.

```
% Probability vector sorted from highest to lowest
P = sort(PROBABILITY_VECTOR, 'descend');


% Values obtained with Shannon-Fano code algorithm

% Symbols codeword length
m = [3, 3, 3, 3, 3, 4, 4, 4, 4, 4, 5, 5];

% Number of 0s inside the symbols codeword
m_0 = [0, 1, 1, 2, 1, 2, 3, 2, 3, 3, 4, 5];

% Number of 1s inside the symbols codeword
m_1 = [3, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 0];
```

Binary entropy

At this point, there is all the needed information to calculate the required data for the analysis. First of all, to calculate the average codeword length $\overline{m}$ of N symbols, the following formula should be computed:

$$\overline{m} = \sum_{i=1}^{N} m_i \cdot P_i$$

Additionally, to calculate P_0 and P_1, it is necessary to calculate also the average number of 0 and 1 symbols. The formula is the same as for the average codeword length:

$$\overline{m_0} = \sum_{i=1}^{N} m_{0_i} \cdot P_i \qquad \overline{m_1} = \sum_{i=1}^{N} m_{1_i} \cdot P_i$$

After inserting these formulas in the MATLAB script, the values are $\overline{m} = 3.4300$, $\overline{m_0} = 1.6400$ and $\overline{m_1} = 1.7900$.

```
% Average codeword length
m_average = dot(P, m);

% Average number of 0 symbols
m_0_average = dot(P, m_0);

% Average number of 1 symbols
m_1_average = dot(P, m_1);
```

Moreover, by dividing the number of 0 or 1 symbols by the average length of the codeword the two probabilities, P_0 and P_1, can be computed:

$$P_0 = \frac{\overline{m_0}}{\overline{m}} \qquad P_1 = \frac{\overline{m_1}}{\overline{m}}$$

The two probabilities values are $P_0 = 0.4781$ and $P_1 = 0.5219$. Ideally, the probabilities should be $P_0 = P_1 = 0.5$; nonetheless, the two values are still very close to each other. Finally, with the two probability values the binary entropy H_{bin} can be obtained using the following calculation.

$$H_{bin}(S) = -P_0 \log_2 P_0 - P_1 \log_2 P_1$$

By running the following MATLAB script, the value of the binary entropy is $H_{bin} = 0.9986$ which is very close to 1. The higher the entropy is, the more uncertainty is associated with every symbol: this means that encoding the initial data with the Shannon-Fano algorithm provides a great value, information-wise.

```
% Probability of 0 symbol
P_0 = m_0_average / m_average;

% Probability of 1 symbol
P_1 = m_1_average / m_average;

% Binary source entropy after coding
H_bin = -P_0 * log2(P_0) - P_1 * log2(P_1);
```

Data rate and compression ratio

After encoding the source data with the Shannon-Fano algorithm, it is important to evaluate the source data generation rate R, which can be calculated as follows:

$$R = \frac{H(S)}{\overline{m}\tau}, \text{ where } \tau \text{ is the symbol duration}$$

The data compression ratio K is important as well: it helps evaluate how much the initial data has been compressed after the source coding. The following formula will help to obtain this value.

$$K = \frac{\overline{m}}{H(S)}$$

After running the MATLAB script displayed below, the data rate $R = 16.519 Mbit/s$ which should be lower than the channel capacity C_{chan} with noise. Moreover, the compression ratio $K = 1.0090$ which is very close to 1, means that the overall compression is low: this is still not a bad result because the overall entropy is increased significantly after the source coding.

```
% Calculate Data Rate
R = H * (m_average * TAU) ^ (-1);

% Calculate Compression Ratio
K = m_average / H;
```

3 Shannon's theorem condition

Shannon's theorem asserts that for reliable communication two important conditions should be verified: using a strong error correction code for a specified SNR value and $R \leq C_{chan} - \epsilon$, $\epsilon \to 0$ meaning that the source data rate R should be less (or, at most equal) that the channel capacity C_{chan}.

It is already possible to compare the data rate R with the noiseless channel capacity C_{bin} by computing this formula:

$$C = \frac{1}{\tau}$$

Expectedly, the result is $C_{bin} = 16.667$ Mbit/s and reasonably meet the Shannon's theorem contidion : $R = 16.5$ Mbps ≤ 16.7 Mbps $= C_{bin}$.

Bit Error Rate

Before calculating the channel capacity noise, the error probability P_{err}, also called BER (Bit Error Rate), shall be calculated. To do so, by reversing the SNR formula, the energy per bit to noise power spectral density ratio $\frac{E_b}{N_0}$ needs to be calculated:

$$\text{SNR} = 10 \log_{10}(\frac{E_b}{N_0}) \quad \Longrightarrow \quad \frac{E_b}{N_0} = 10^{\frac{\text{SNR}}{10}}$$

Which translates in the following MATLAB line:

```
% Energy per bit to noise power spectral density ratio
Eb_N0 = 10^(SNR / 10);
```

To calculate the error probability of the BPSK modulation time, the following formula should be used:

$$P_{err} = 1 - \Phi\left(\sqrt{2\frac{E_b}{N_0}}\right)$$

Additionally the Φ function can be created using the `erf` function in MATLAB as follows:

$$\Phi(x) = \frac{1}{2}\left[1 + \texttt{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

With this information the MATLAB script can be produced and the BER value can be calculated: $P_{err} = 1.6315 \cdot 10^{-4}$.

```
% define the phi function
phi = @(x) 1/2 * ( 1 + erf(x / sqrt(2)) );

% error probability
P_err = 1 - phi( sqrt( 2 * Eb_N0) );

% no error probability
P_err_comp = 1 - P_err;
```

Channel capacity with noise

All the information needed for the calculation of C_{chan} is now ready for use. To calculate the channel capacity with noise the following formula should be utilized:

$$C_{chan} = \frac{1}{\tau}\left[1 + P_{err}\log_2\left(P_{err}\right) + \left(1 - P_{err}\right)\log_2\left(1 - P_{err}\right)\right]$$

The formula translates in the following MATLAB code line:

```
% Channel capacity with noise
C_chan = ( 1 + P_err * log2(P_err) + P_err_comp * log2(P_err_comp)
    ) * C;
```

By running the script the result is $C_{chan} = 16.629\text{Mbps} \geq 16.519\text{Mbps} = R$ meaning that the Shannon's Theorem condition is fulfilled. Consequently, it is possible to find a coding approach that will recover the errors that occurred during the transmission. If the SNR value was, hypothetically, lower, there was a chance that $R > C_{chan}$ would've translated into the unpossibility of finding an error-correcting code for the transmission.

Raw results

Make a little introduction in the "Raw result" section

Symbol	Description	Value
Task 2		

Conitnue on next page ...

... continued from previous page

Symbol	Description	Value
$H(S)$	Source entropy	3.3995
$H(S)_{max}$	Maximum source entropy	3.5850
ρ	Source redundancy	0.0517
	Task 3	
$\overline{m}$	Average codeword length	3.4300
P_0	Probability of `"0"`	0.4781
P_1	Probability of `"1"`	0.5219
$H_{bin}(S)$	Binary entropy	0.9986
R	Source data rate	16.519 Mbps
K	Compression ratio	1.0090
	Task 4	
C_{bin}	Noiseless channel capacity	16.667 Mbps
P_{err}	Error probability (BER)	$1.6315 \cdot 10^{-4}$
C_{chan}	Channel capacity with noise	16.629 Mbps
	Task 5	
	Task 6	
	Task 7	
	Task 8	
$P_{>2\,err}$	Probability of > 2 errors occurring	$1.2338 \cdot 10^{-5}$

Part II
Simulation

1 Data generation algorithm

To simulate the transmission using the given initial parameters it is crucial to generate the symbols following the probabilities specified in the `Source 7` data sheet. To achieve such generation specifics a generation algorithm shall be implemented in MATLAB. The following MATLAB functions will generate a sequence of `n` symbols in the alphabet following their probability distribution.

Comulative distribution probabilities calculator

The first function - `distribution_probability_matrix` - will take as input the `symbol_matrix` whose first row contains the symbols in the alphabet and the second row contains their respective probabilities. The function will return a matrix whose first row is the same but the second row contains cumulative distribution meaning that the second probability value is added to the first, the third is added to the second and so on. Consequently, the last probability value will be one.

```
function result = distribution_probability_matrix(symbol_matrix)
    % Extract probability vector from the symbol matrix
    probability_vector = symbol_matrix(2, :);

    % Get the number of possible symbols
    alphabet_length = length(probability_vector);

    % Calculate the cumulative probability matrix
    cumulative_probability = 0;
    sum_probability_vector = zeros(1, alphabet_length);
    for i = 1:alphabet_length
        % Calculate cumulative probability
        cumulative_probability = cumulative_probability +
            probability_vector(i);

        % Store cumulative probability in the vector
        sum_probability_vector(i) = cumulative_probability;
    end

    % Combine symbols and cumulative probabilities into the result matrix
    result = [symbol_matrix(1, :); sum_probability_vector];
end
```

This helper function will be useful when a random number r between 0 and 1 is generated: the symbol associated with r will be the i-th symbol where $P_{i-1} < r \leq P_i$. In such a way the symbols will have the same probability to be associated with the number r as specified in the datasheet.

Sequence generator

The `distribution_probability_matrix` function will be used in the actual generation function called `symbol_sequence_generator` which generates `n` symbols conforming with their specified probabilities. The below-displayed function will generate a random number r between 0 and 1 and associate it with the i-th symbol whose probability is $P_{i-1} < r \leq P_i$. This is achieved by subtracting the random number from the cumulative probability vector and choosing the symbol with the lowest positive probability value. This procedure is computed `n` time: every temporal association is added to the final result which will be the generated symbol sequence.

```
function result = symbol_sequence_generator(symbol_matrix, n)
    % Initialize an empty vector for the symbol sequence with length n
    result = zeros(1, n);

    % Calculate the cumulative probability matrix using the provided
        function
    sum_probability_matrix =
        distribution_probability_matrix(symbol_matrix);

    % Generate the symbol sequence
    for i = 1:n
        % Generate a random number between 0.00 and 1.00
        random_number = round(rand(), 2);

        % Calculate the distance of each cumulative probability from the
            random number
        distance_from_random_number = sum_probability_matrix(2, :)
            - random_number;
        distance_from_random_number(distance_from_random_number <
            0) = +Inf;

        % Get the index of the symbol with the minimum distance
        [~, symbol] = min(distance_from_random_number);

        % Assign the selected symbol to the result vector
        result(i) = symbol;
    end
end
```

2 Source coding and decoding algorithms

The second step is to implement an algorithm that will encode and decode the newly generated symbols using the Shannon-Fano algorithm. The two algorithms are based on the results obtained and displayed in the `Code` column of the table in the "Source

encoding" section.

Shannon-Fano encoding

The Shannon-Fano encoding can be achieved by using a very simple switch. Sure enough, the helper function `encode_symbol` displayed below associates with every symbol in the alphabet and its respective codeword.

```
function result = encode_symbol(symbol)
    % Use a switch statement to assign the encoded representation based
        on the input symbol
    switch symbol
        case 1
            result = [ 1 0 0 ];
        case 2
            result = [ 0 1 0 0 ];
        case 3
            result = [ 0 1 0 1 ];
        case 4
            result = [ 0 0 0 0 0 ];
        case 5
            result = [ 0 0 1 1 ];
        case 6
            result = [ 0 0 1 0 ];
        case 7
            result = [ 1 1 0 ];
        case 8
            result = [ 1 1 1 ];
        case 9
            result = [ 1 0 1 ];
        case 10
            result = [ 0 0 0 1 ];
        case 11
            result = [ 0 1 1 ];
        case 12
            result = [ 0 0 0 0 1 ];
    end
end
```

The `shannon_fano_encoding` function takes the `symbol_sequence` as input and encodes it symbol-by-symbol using the aforementioned `encode_symbol` function.

```
function encoded_sequence = shannon_fano_encoding(symbol_sequence)
    encoded_sequence = [];

    % Iterate through the symbol sequence and encode each symbol
    for i = 1:length(symbol_sequence)
        encoded_sequence = [encoded_sequence,
            encode_symbol(symbol_sequence(i))];
    end
end
```

Shannon-Fano decoding

The decoding algorithm performs the exact reverse operation of the encoding algorithm. Sure enough, there is the `decode_symbol` function which translates the binary sequence into its respective symbol.

```
function symbol = decode_symbol(code)
    % Use a switch statement to check each possible code and return the
        corresponding symbol
    switch code
        case '[1 0 0]'
            symbol = 1;
        case '[0 1 0 0]'
            symbol = 2;
        case '[0 1 0 1]'
            symbol = 3;
        case '[0 0 0 0 0]'
            symbol = 4;
        case '[0 0 1 1]'
            symbol = 5;
        case '[0 0 1 0]'
            symbol = 6;
        case '[1 1 0]'
            symbol = 7;
        case '[1 1 1]'
            symbol = 8;
        case '[1 0 1]'
            symbol = 9;
        case '[0 0 0 1]'
            symbol = 10;
        case '[0 1 1]'
            symbol = 11;
        case '[0 0 0 0 1]'
            symbol = 12;
        otherwise
            symbol = []; % Return empty if the code does not match any
                known code
    end
end
```

This function is used in the `shannon_fano_decoding` function wich performs the decoding of the input `encoded_sequence`. The body of the function is a little more complicated than the encoding function because the length of the encoded symbol is not fixed - it can be 3, 4 or 5 - and, as such, every time a new symbol is read from the decoded data, a check should be done to understand if the symbol can be decoded or not.

```
function decoded_sequence = shannon_fano_decoding(encoded_sequence)
    decoded_sequence = [];

    % Iterate through the encoded sequence and decode each symbol
```

```
    current_code = [];
    for i = 1:length(encoded_sequence)
        current_code = [current_code, encoded_sequence(i)];

        % Check if the current code matches any known code
        symbol = decode_symbol(string(mat2str(current_code)));

        % If a symbol is found, add it to the decoded sequence and reset
            the current code
        if ~isempty(symbol)
            decoded_sequence = [decoded_sequence, symbol];
            current_code = [];
        end
    end
end
```

Start the channel coding