FACULTY OF ENGINEERING
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
ARISTOTLE UNIVERSITY OF THESSALONIKI

Error Correction Codes

ASSIGNMENT REPORT

Students:
Liouliakis Nikolaos - 10058
Syskakis Panagiotis - 10045
Papadopoulou Aikaterini - 10009

Thessaloniki
2023-2024

Contents

Contents 2
Introduction 3
Part 1 4
Part 1a 4
Theoretical Background 4
Simulations 5
Part 1b 21
Increasing throughput using multiple channels 21
Analyzing the Impact of Retransmission on Network Performance 30
Decoding for Channels with Heterogeneous Error Rates Across Bits 34
Part 2 38
LDPC codes as polynomials 38
Generating Regular LDPCs 40
Generating the parity check matrix H 41
Performing Erasure Correction 42
Simulation 44
Conclusions 52
References 53

Introduction

In digital communications, the integrity and reliability of transmitted data is crucial. Error Correction Codes (ECC) are used to detect and correct errors that may occur due to factors such as noise, interference or other disruptions in the communication system. This assignment examines several different methods that are used for error correction.

Part 1 focuses on the Binary Symmetric Channel (BSC), where 2 different cases are studied. **Part 1a** revolves around Linear Block Bodes. Both encoding and modulation are considered. The goal is to draw conclusions about how the modulation order, codeword length and Symbol to Noise Ratio (SNR) influence the Bit Error Rate (BER) and the transmission rate.

Part 1b only takes encoding into consideration (modulation is not considered). The scenario revolves around the use of more than one channel with different error probabilities. The goal is to draw conclusions about the optimal use of all available channels to achieve fast and reliable data transmission.

Part 2 is about the Binary Erasure Channel (BEC), where Low Density Parity Check (LDPC) codes are used. The goal is to compare regular and irregular LDPC codes, study their performance and evaluate the potential improvements offered by the irregular codes.

Through this assignment, we anticipate gaining a deeper understanding of error correction mechanisms and their critical role in ensuring robust and efficient digital communication.

Part 1

Part 1a

Theoretical Background

Linear block codes are represented in standard form, using generator matrix $\mathbf{G} = [\mathbf{I_k}|\mathbf{P}]$ and parity matrix $\mathbf{H} = [-\mathbf{P^T} \mid \mathbf{I_n} - \mathbf{k}]$ where *P* is a $\mathbf{k} \cdot (\mathbf{n} - \mathbf{k})$ matrix. To generalize the results and draw better conclusions, multiple linear block codes are tested. To generate codes for any desired codeword length *n* and message length *k*, *P* is dynamically generated in a random manner, using the following heuristic method.
For the desired *k* and *n*, many matrices *P* are generated at random. For each candidate *P*, all codewords of the corresponding code are created. The optimal *P* is regarded to be the one whose codewords have the largest Hamming distance d_{min}.

In favor of simplicity and efficiency, Quadrature Amplitude Modulation (QAM) is used. The order of the constellation can be adjusted through the *bits per symbol (bps)* parameter and the energy of the constellation is configured to be normalized by default. Moreover, QAM is gray-coded by default so that bit errors are minimized, but binary-coded QAM can also be selected.

After modulation, Additive White Gaussian Noise (AWGN) is added based on the selected Symbol to Noise Ratio (SNR), to simulate a noisy channel.

For the demodulation, the Maximum Likelihood Detector (MLD) is used. This detector is based on the assumption that all symbols are equally likely to appear[1].

Bit Error Rate (BER) is calculated twice, once right after the demodulation before error correction has taken place, as well as after the decoding process after the error correction has been applied.

Also, the useful throughput (actual information bits per second) $\mathbf{R_{b\_code}}$ of the code is calculated. To do this, the bit duration is calculated as $\mathbf{T_b} = \mathbf{T_s} / \mathbf{bps}$, where T_s is the symbol duration and bps is bits per symbol. Then, the code rate is calculated as $\mathbf{R} = \mathbf{k}/\mathbf{n}$. The finally, throughput is calculated as $\mathbf{R_b} = \mathbf{1}/\mathbf{T_b}$ and finally the useful throughput is calculated as $\mathbf{R_{b\_code}} = \mathbf{R} \cdot \mathbf{R_b}$.

[1] This assumption is almost never true when using linear block codes. This will be explained in greater detail later.

One last thing that must be noted is that it is not possible to have a message of an arbitrary number of bits. Those two equations must hold:

$$\mathbf{D} \bmod \mathbf{k} = \mathbf{0}$$

$$\frac{\mathbf{D} \cdot \mathbf{n}}{\mathbf{k}} \bmod \mathbf{bps} = \mathbf{0}$$

The number of message bits *D* must be a multiple of the message length *k*, thus the first equation is derived. Then, *D* divided by the code rate results in the number of bits that are produced by the encoding process. This number must be divisible by the *bps* for the modulation; thus, the second equation is derived.

To satisfy the above problem, *D* must be a multiple of *k*, $\mathbf{D} = \mathbf{k} \cdot \mathbf{m}$ where *m* is an integer. Then, *m* substituted in the second equation: $(\mathbf{k} \cdot \mathbf{m} \cdot \mathbf{n}/\mathbf{k}) \mathbf{mod} (\mathbf{bps}) = \mathbf{0}$ which simplifies to $(\mathbf{m} \cdot \mathbf{n}) \mathbf{mod} (\mathbf{bps}) = \mathbf{0}$. This is a modular arithmetic problem. Given the values of *n* and *bps* and using the modular inverse, *m* can be calculated as $\mathbf{m} = \mathbf{a} \cdot \mathbf{bps}/\mathbf{gcd}(\mathbf{n}, \mathbf{bps})$, where *a* is an integer. Substituting to get *D*: $\mathbf{D} = \mathbf{a} \cdot \mathbf{k} \cdot \mathbf{bps}/\mathbf{gcd}(\mathbf{n}, \mathbf{bps})$. The previous is rewritten as $\mathbf{D} = \mathbf{a} \cdot \mathbf{Dmin}$, where $\mathbf{Dmin} = \mathbf{k} \cdot \mathbf{bps}/\mathbf{gcd}(\mathbf{n}, \mathbf{bps})$.

If the given *D* is not a multiple of D_{min}, it is rounded up to the next multiple using this formula:

$$\mathbf{D'} = \mathbf{ceil}(\mathbf{D}/\mathbf{Dmin}) \cdot \mathbf{Dmin}$$

Simulations

The MATLAB code is largely parametrized. The user can choose for which parameter values the simulations will run, which plots will be generated, if they will be saved and in what format. Also, further customization can be made, i.e. whether the constellation will be gray-coded or binary-coded, if the energy will be normalized or not, and more.

For the simulations presented below, the constellations are chosen to be gray coded. The symbol duration is $\boldsymbol{Ts} = \mathbf{2} \cdot \mathbf{10^{-6}}$, the message length is $\boldsymbol{k} = \mathbf{5}$ and the number of message bits is $\boldsymbol{D} = \mathbf{10^7}$. The simulations run for $\boldsymbol{n} = \mathbf{6}, \mathbf{7}, \mathbf{8}, \mathbf{9}, \mathbf{10}$, $\boldsymbol{SNR} = \mathbf{5}, \mathbf{6}, \dots, \mathbf{20}$ and $\boldsymbol{bps} = \mathbf{1}, \mathbf{2}, \dots, \mathbf{8}$.

Some examples of gray coded QAM constellations can be shown in the following diagrams.

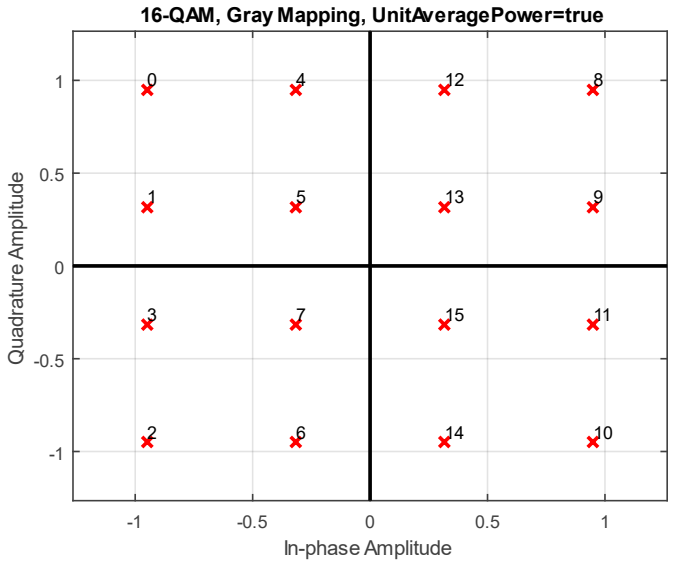

Figure 1 16 QAM constellation with gray coding

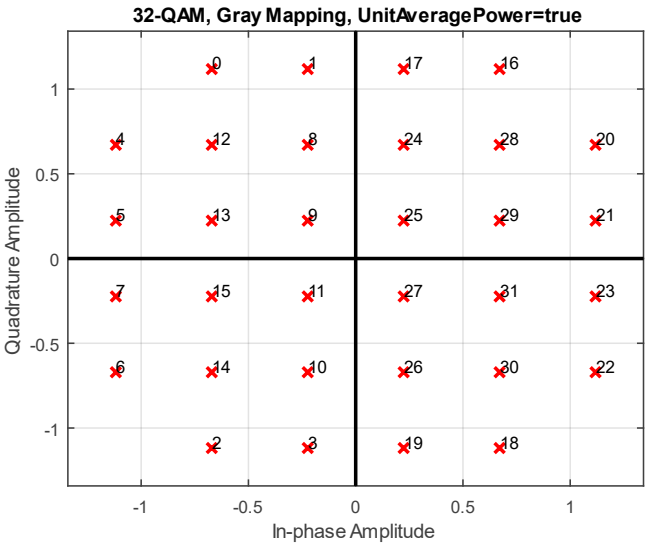

Figure 2 32 QAM constellation with gray coding

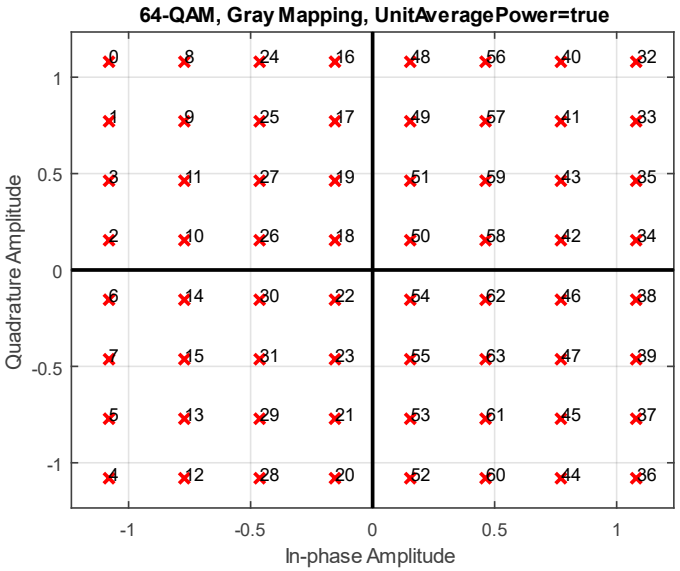

Figure 3 64 QAM constellation with gray coding

From the simulations that were run, the following diagrams were created.

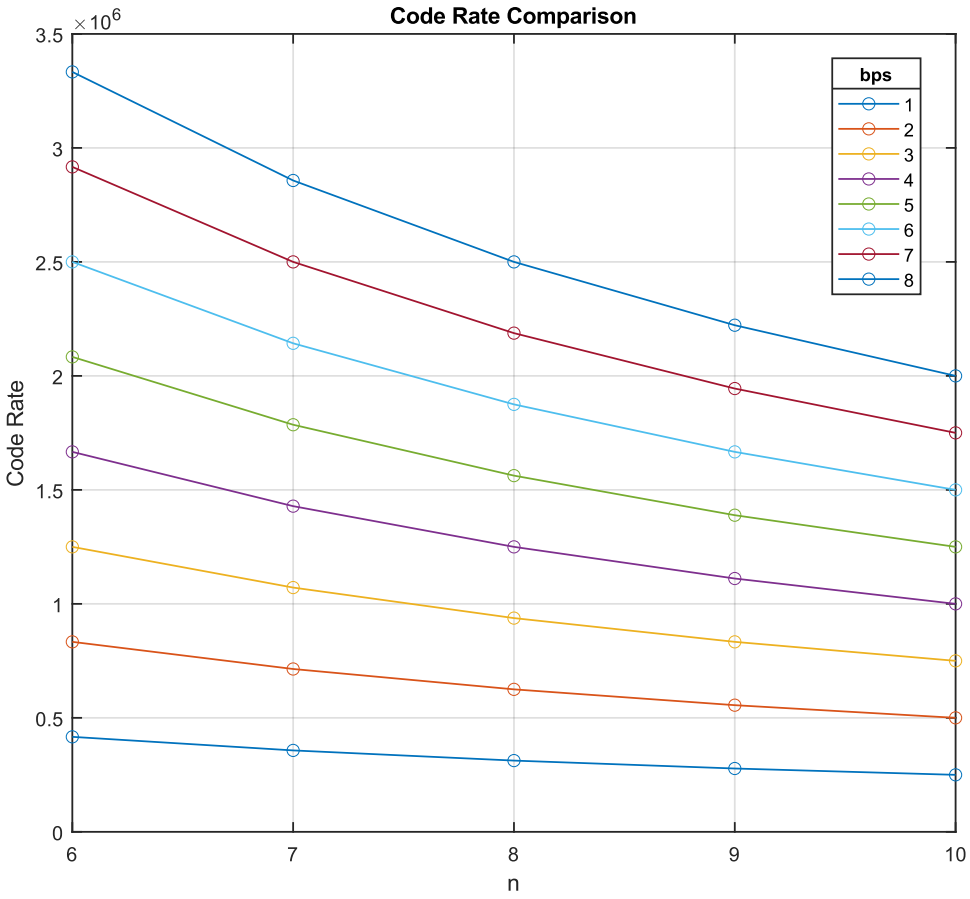

Figure 4 Comparison of code rate and codeword length for different modulation orders

Figure 4 shows a graphical representation between the codeword length and the achieved code rate for different modulation orders (bps).

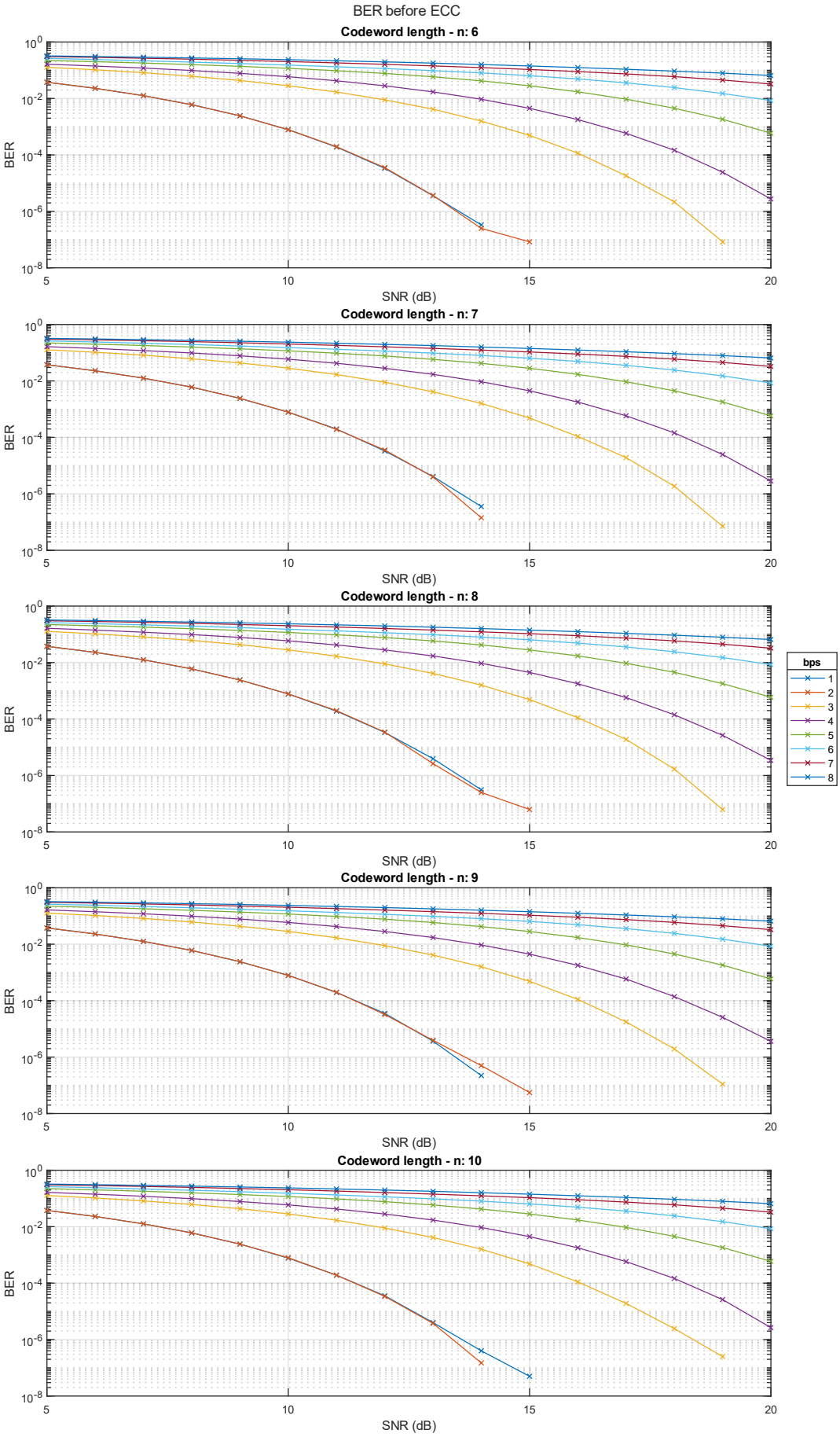

Figure 5 Comparison of achieved BER before Error Correction

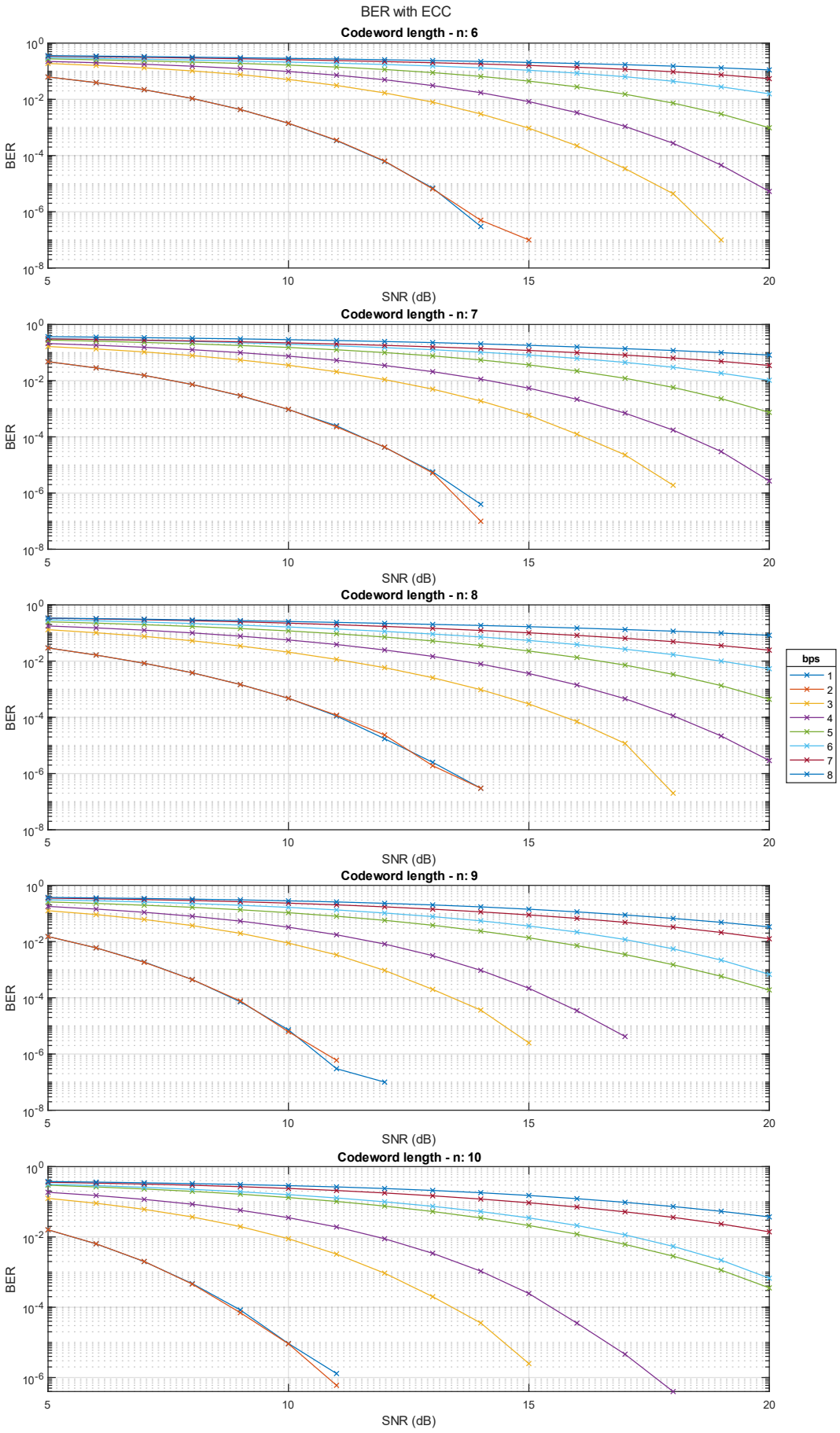

Figure 6 Comparison of achieved BER with Error Correction

Figure 5 and **Figure 6** show the achieved Bit Error Rate (BER) before and after the implemented error correction algorithm, for different codeword lengths (n), Signal to Noise Ratio (SNR) and modulation order (bps). For easier comparison, the following figure shows the ratio of BER before and after the error correction (in logarithmic scale).

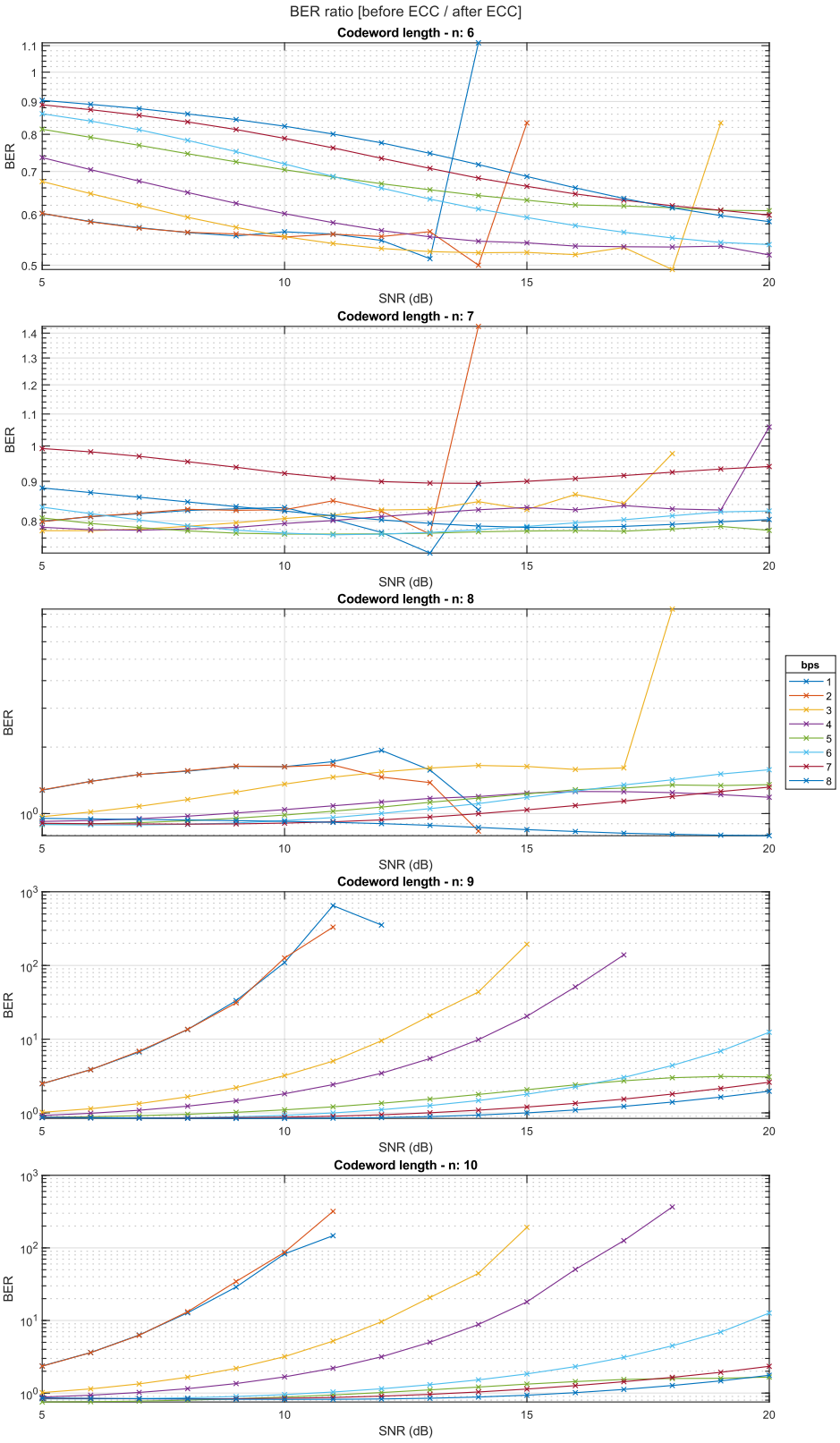

Figure 7 BER ratio (before ECC / after ECC)

The above process was repeated for bigger codeword lengths.

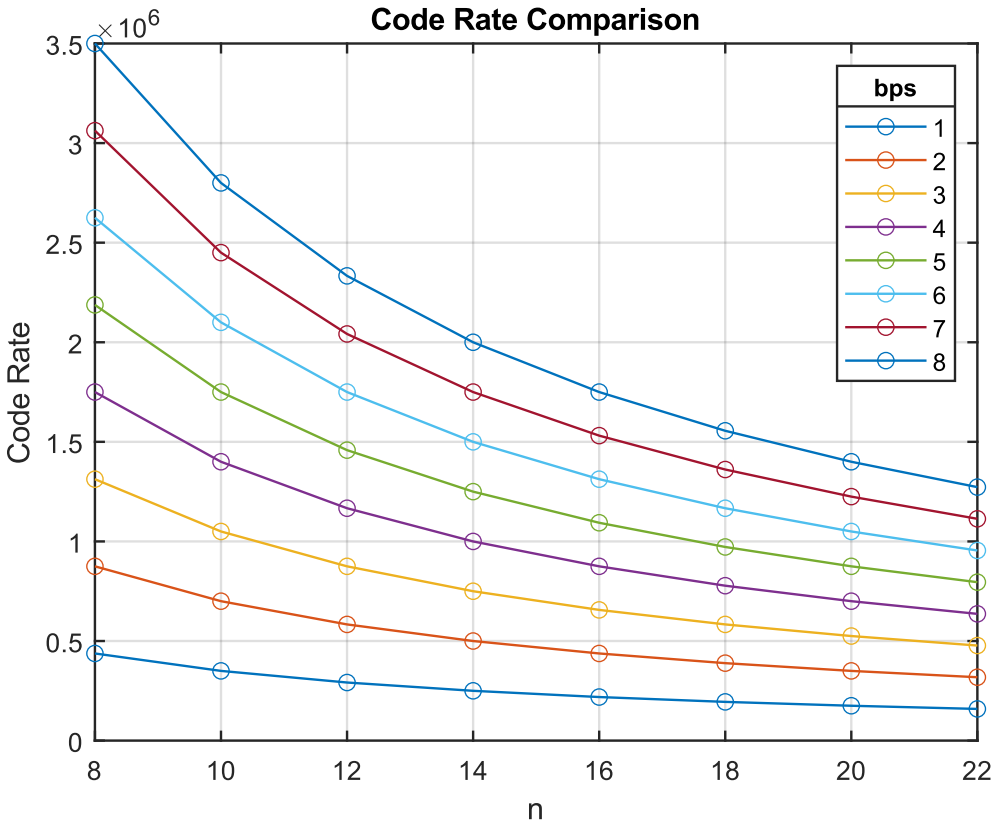

Figure 8 Comparison of code rate and codeword length for different modulation orders

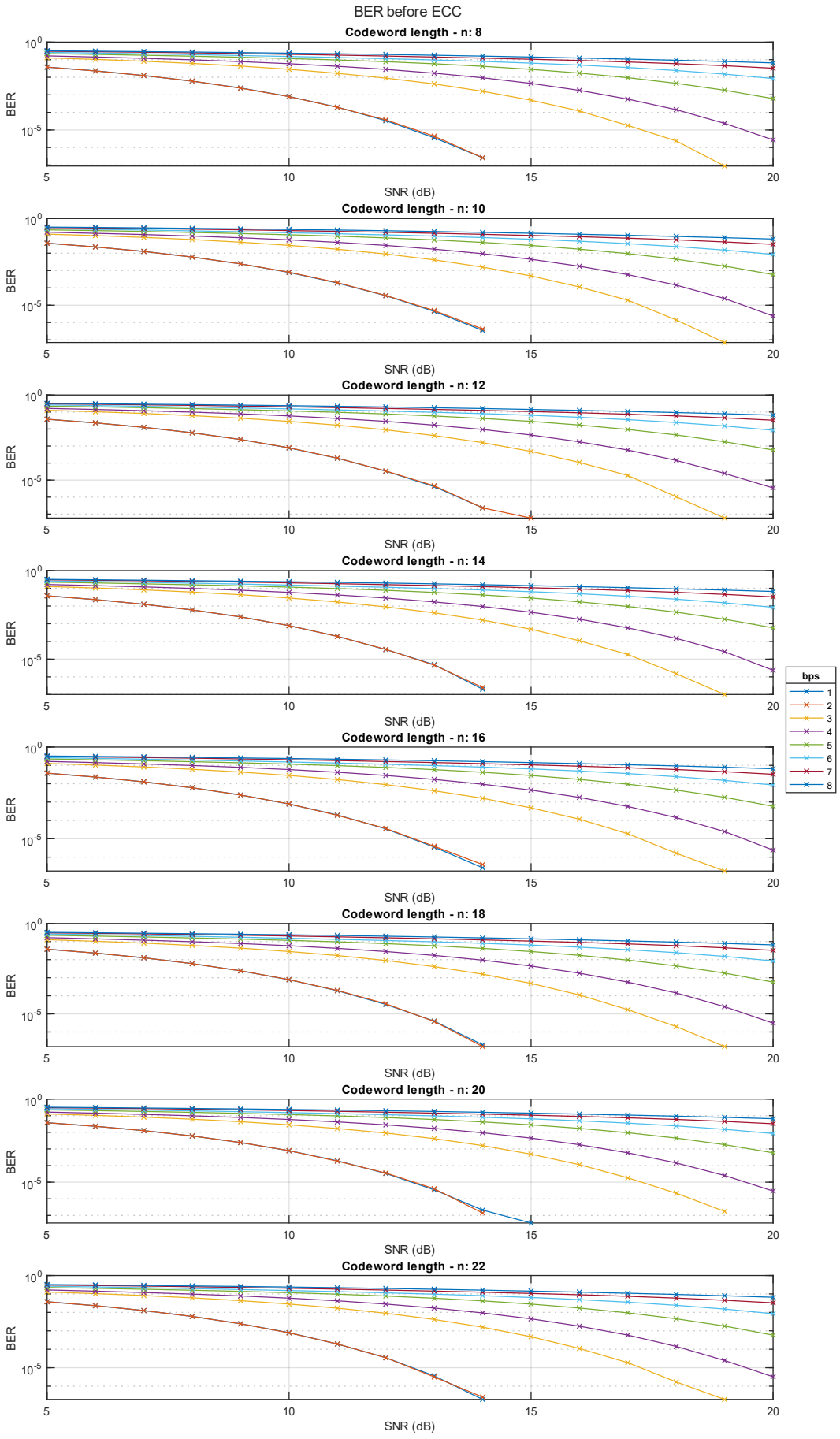

Figure 9 Comparison of achieved BER before Error Correction

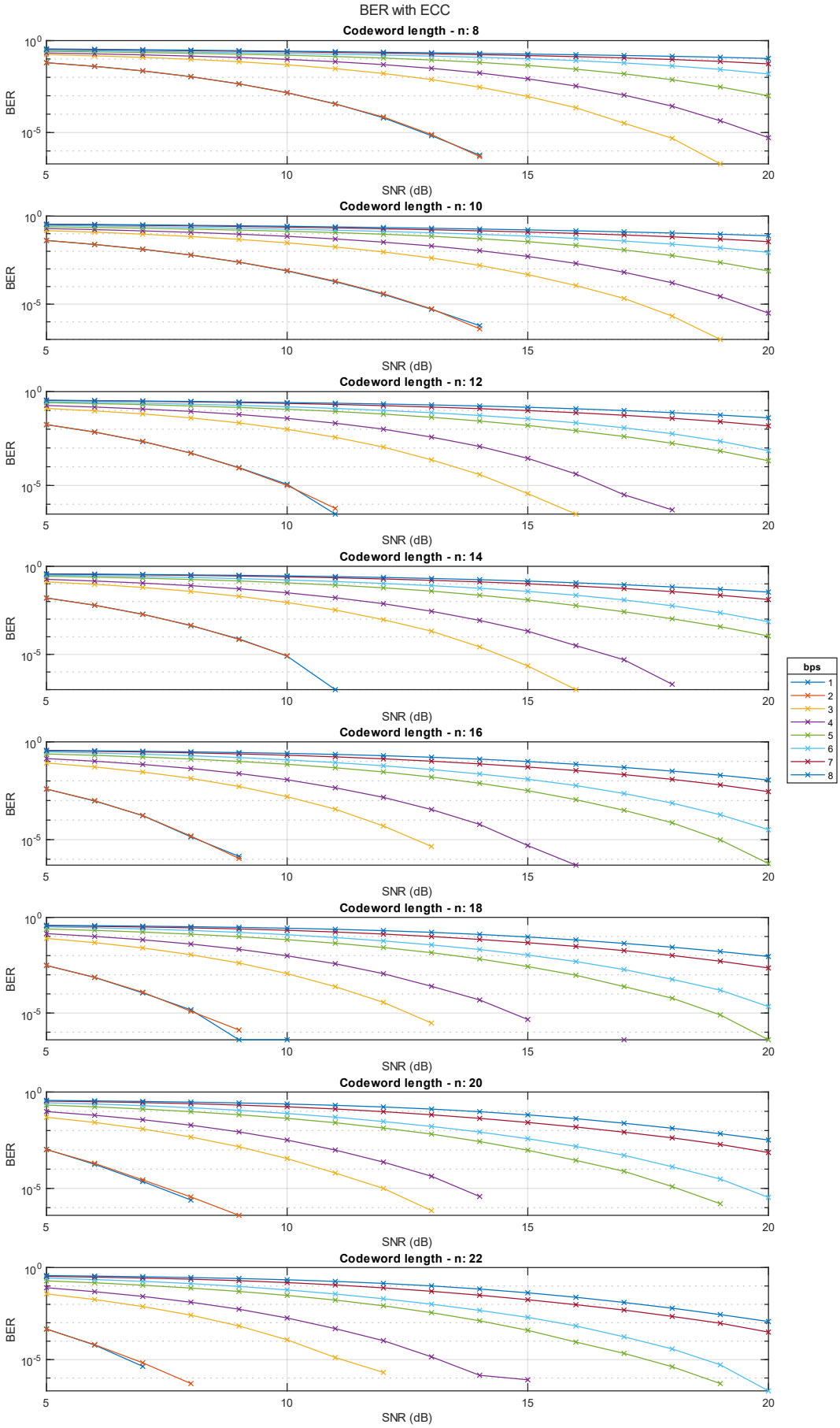

Figure 10 Comparison of achieved BER with Error Correction

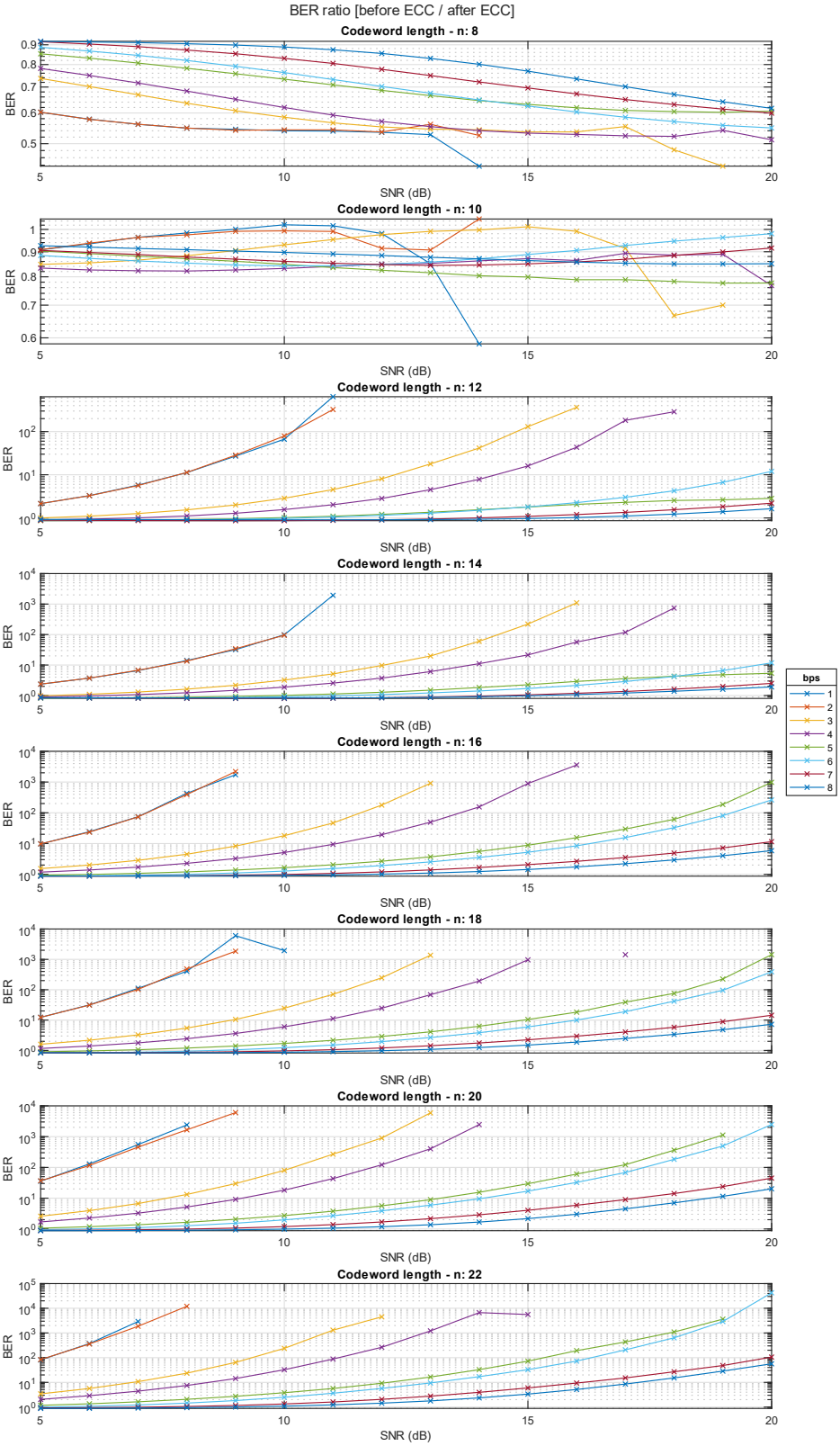

Figure 11 BER ratio (before ECC / after ECC)

A similar representation is shown in the following figures. Each surface shows the achieved BER for different codeword lengths and modulation order (bps) for a specific SNR.

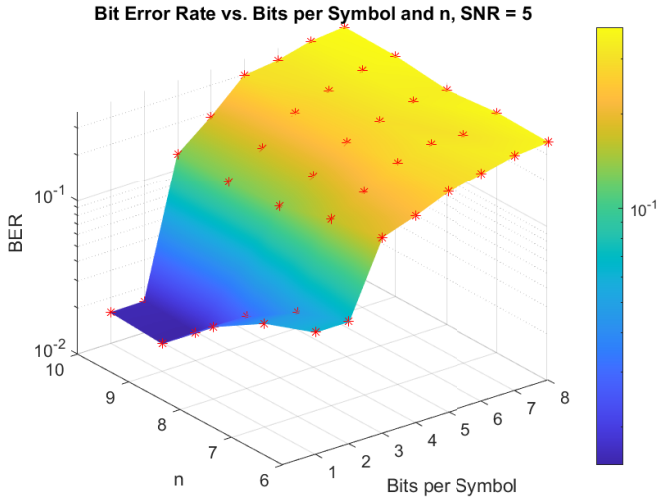

Figure 12 SNR = 5

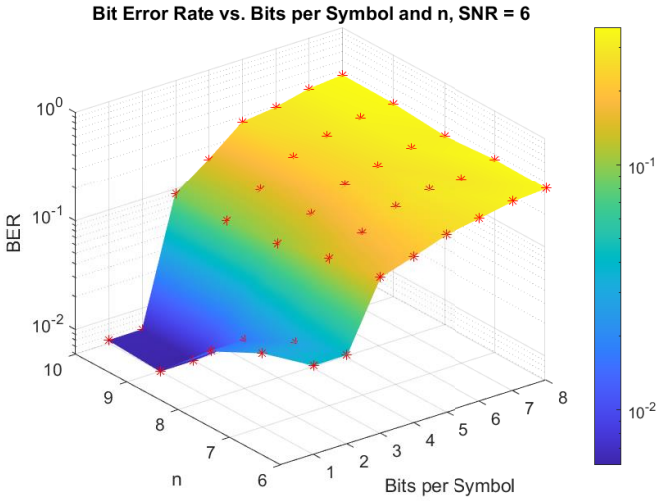

Figure 13 SNR = 6

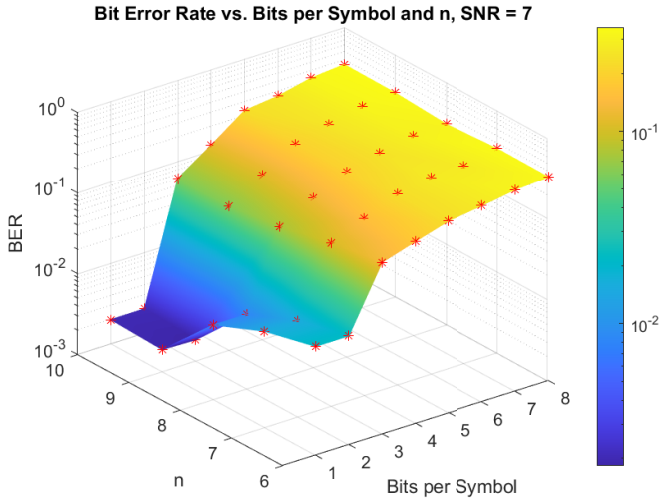

Figure 14 SNR = 7

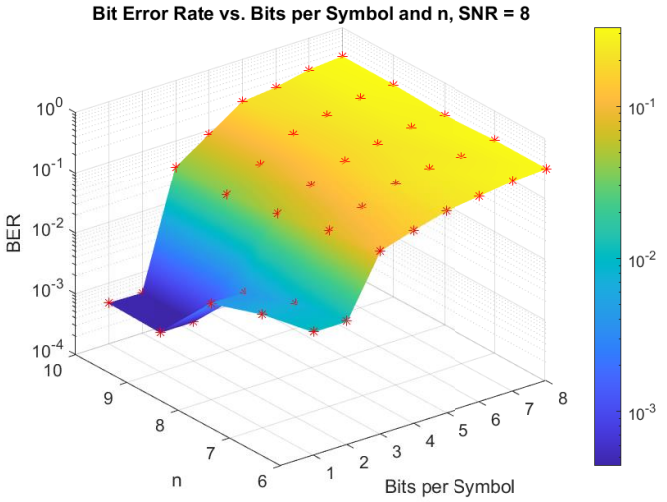

Figure 15 SNR = 8

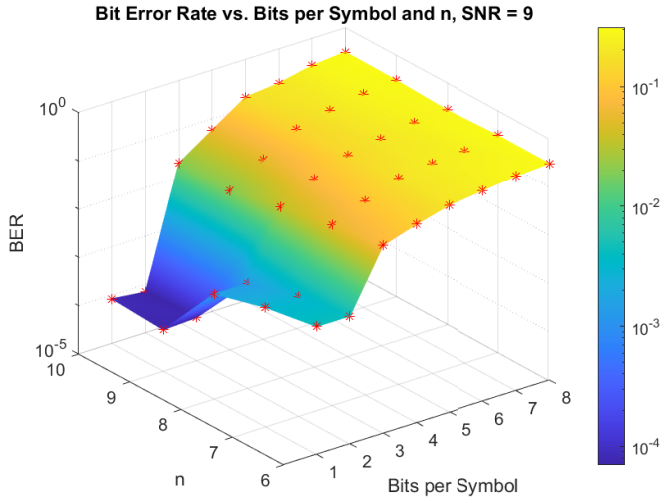

Figure 16 SNR = 9

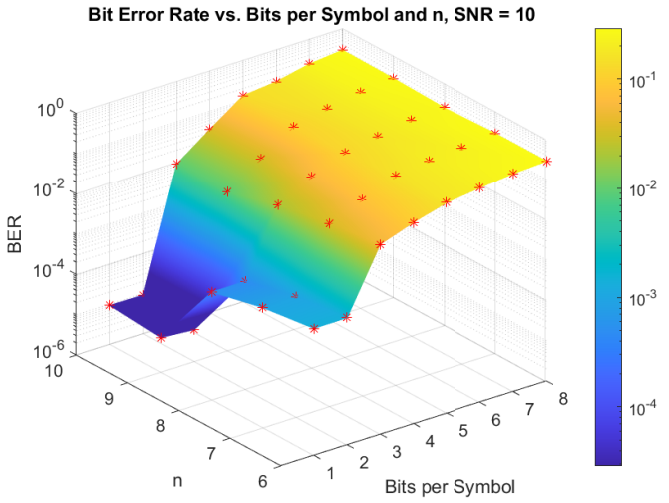

Figure 17 SNR = 10

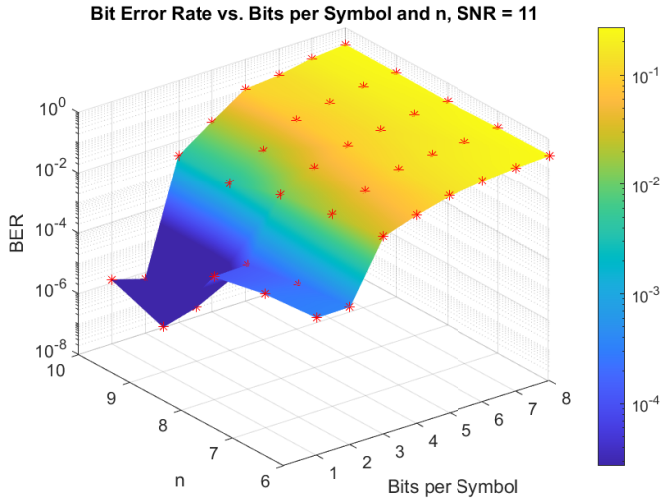

Figure 18 SNR = 11

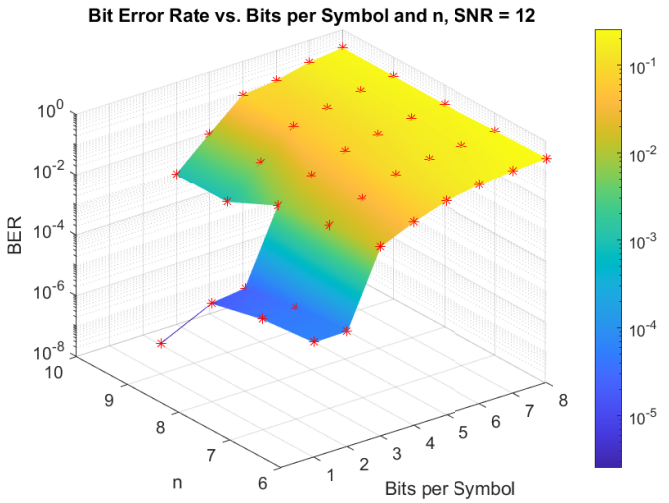

Figure 19 SNR = 12

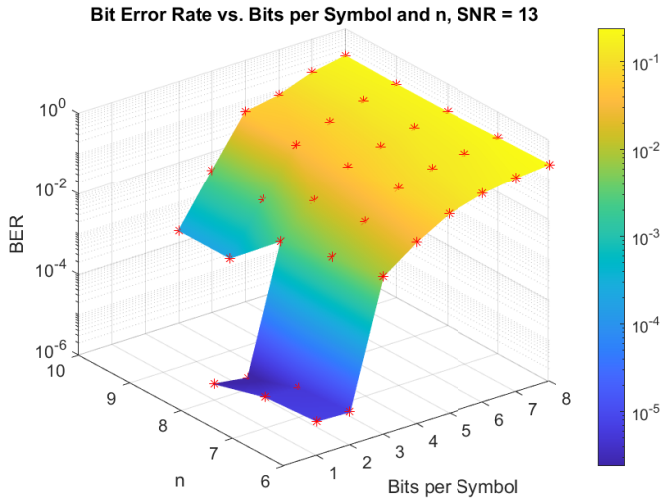

Figure 20 SNR = 13

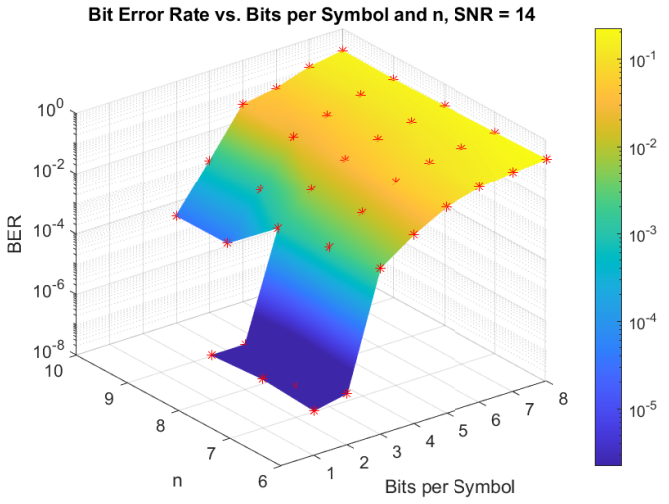

Figure 21 SNR = 14

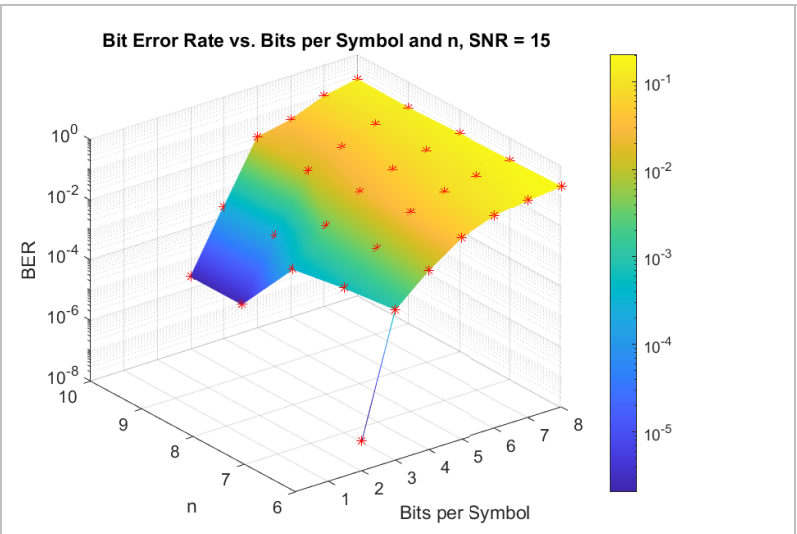

Figure 22 SNR = 15

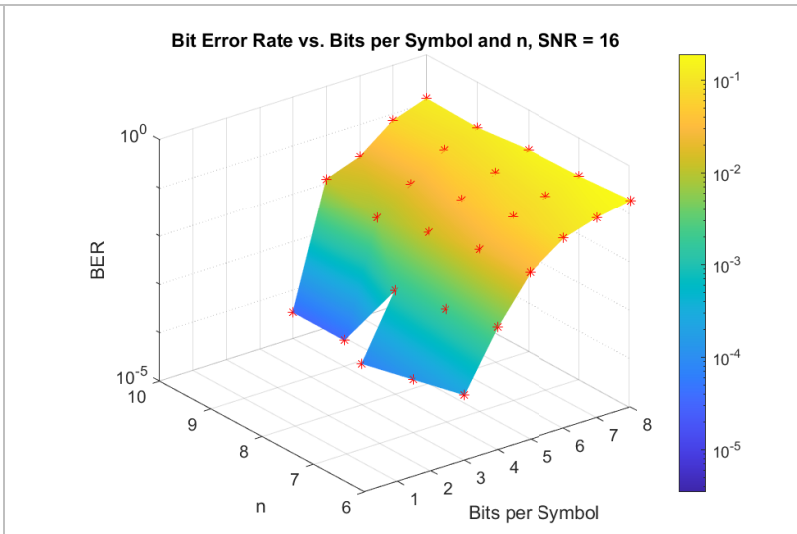

Figure 23 SNR = 16

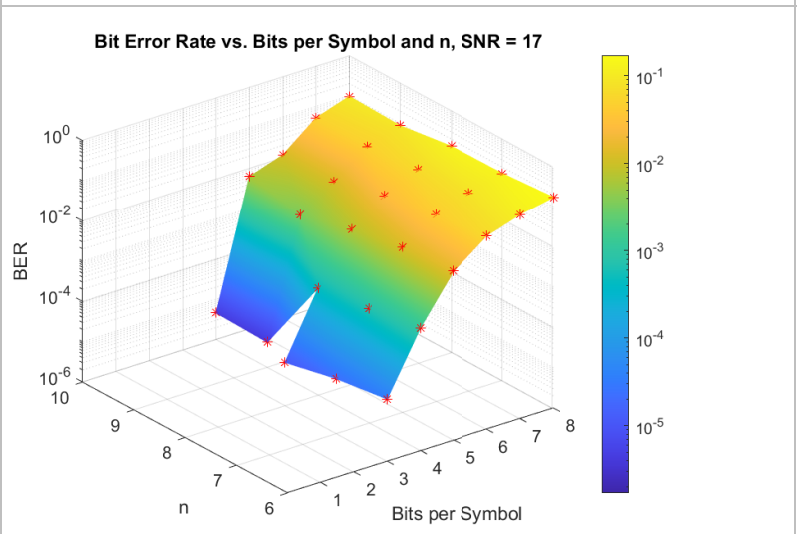

Figure 24 SNR = 17

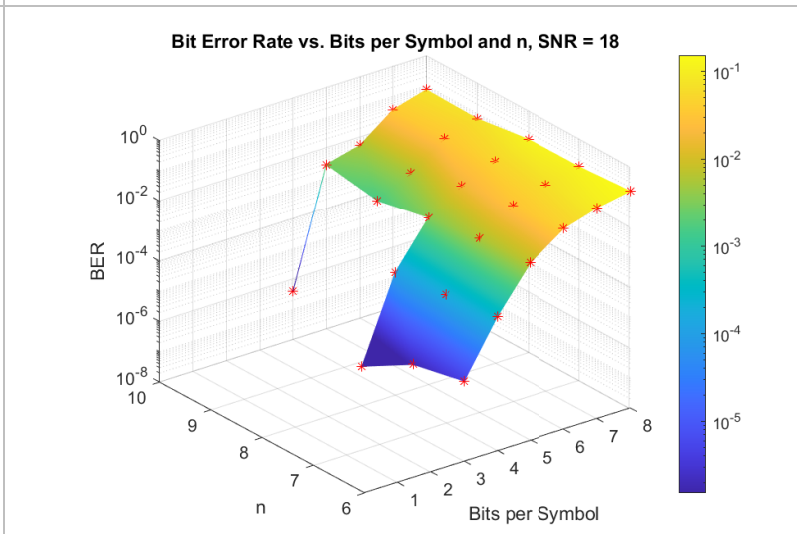

Figure 25 SNR = 18

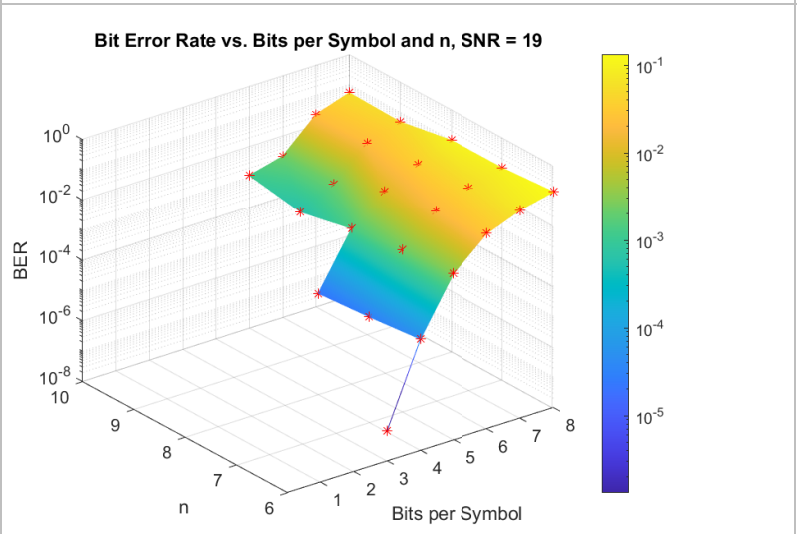

Figure 26 SNR = 19

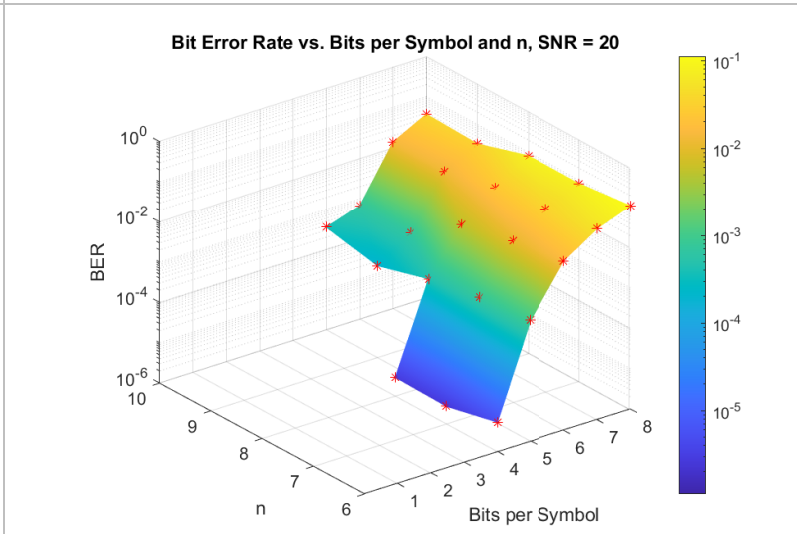

Figure 27 SNR = 20

As can be seen in the corresponding figures, not all points exist in figures **Figure 5 – Figure 27**. This happens because whenever BER is equal to zero (no error occurred in the finite simulation), the logarithmic value cannot be defined.

From **Figure 7** it is evident that for higher SNRs the encoding process improves the performance with a greater than exponential growth. For smaller constellations, this improvement is noticeable from even smaller SNR values.

Another conclusion that can be made is that the bigger the codeword length (n), the better the performance. However, increasing the codeword length results in smaller code rates, as can be seen in **Figure 4**.

An important note is that codeword lengths 6, 7 and 8 don't seem to make a huge improvement in the system's performance. That happens because the message length k was 5 for all simulations, so 1, 2 or 3 redundant bits cannot offer significant error correction, because the hamming distance between codewords is not sufficiently increased.

Figures **Figure 12** – **Figure 27** can be used to determine the most effective modulation order and codeword length if the channel's SNR is known, to achieve the best BER in this range of values. If some n and bps pairs have no corresponding BER value, then, as explained above, these combinations can achieve better BER than the others.

Another observation is that when increasing *k* and *n* proportionally, the error correction ability of the code is improved, even though the code rate remains constant. The drawback is that more bits are required to perform a single decoding operation (since the codewords are larger) and decoding becomes more resource intensive.

Another issue is that not all symbols of a constellation are used with the same frequency. Instead, some symbols are used more often, whereas others are less likely to appear. This phenomenon occurs due to how codewords are mapped to symbols. In modulation, the bits to be transmitted are grouped (the size of these groups depend on the order of the constellation). Some block codes produce code words such that not all groups of bits occur with the same probability.

In some cases, some symbols may not be used at all, because there is no way the codewords can be split and/or combined to create these symbols.

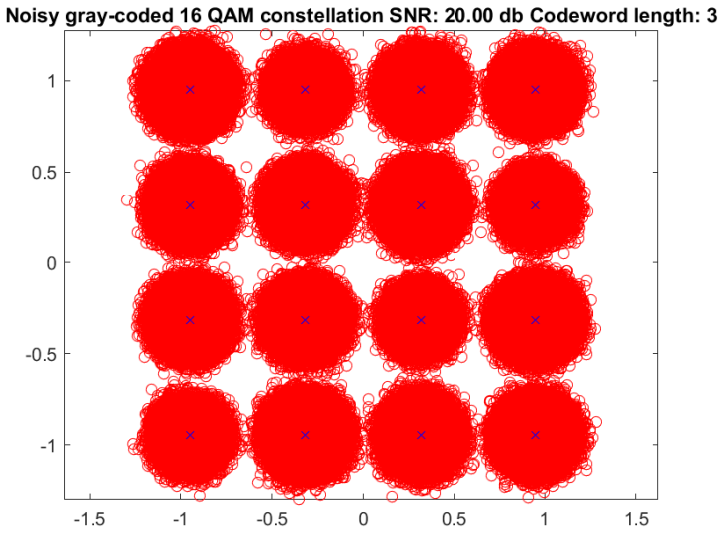

Figure 28

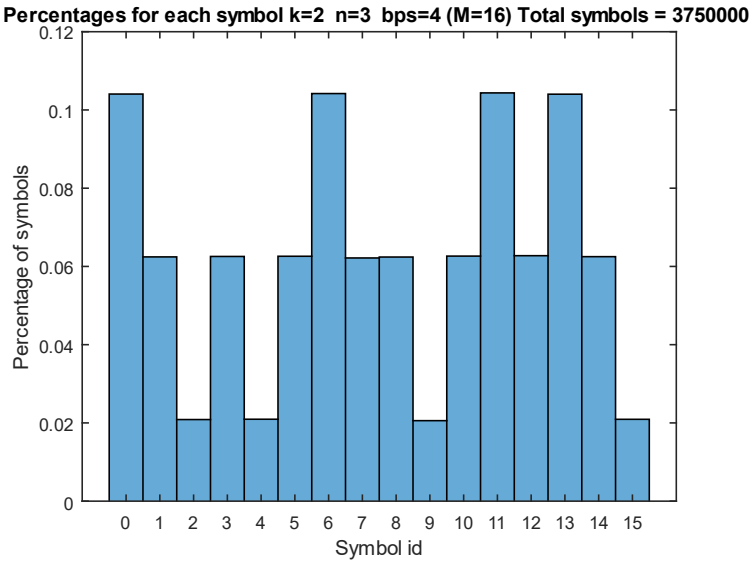

Figure 29

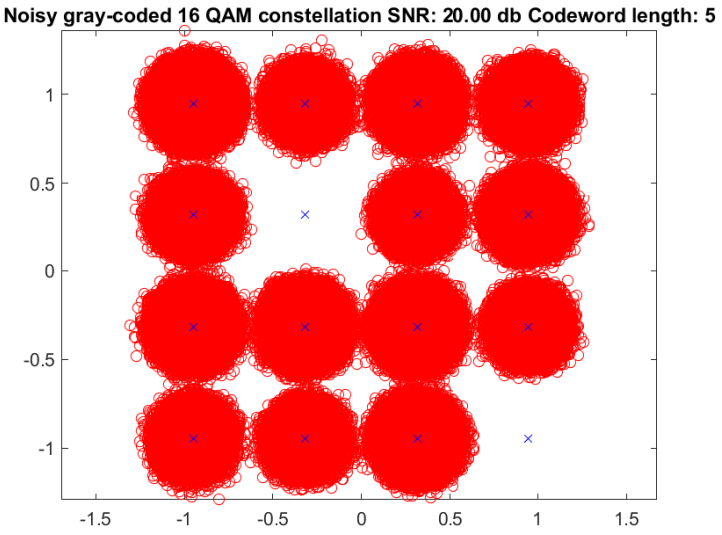

Figure 30

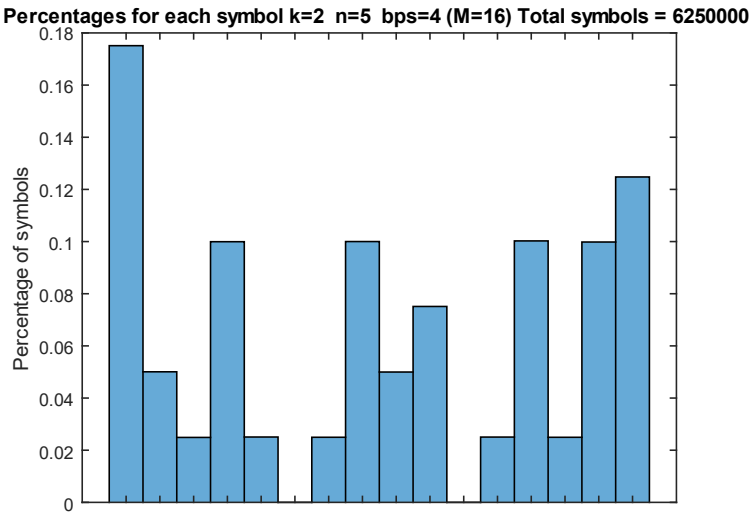

Figure 31

If only a few symbols are not being used, this phenomenon is not of great impact. But there are cases where most constellation symbols are not being used. In these cases, it would be better to use a lower constellation order. By doing so, given the constellations' energies are normalized, the distances between the symbols would be bigger, thus increasing the potential for correct detection.

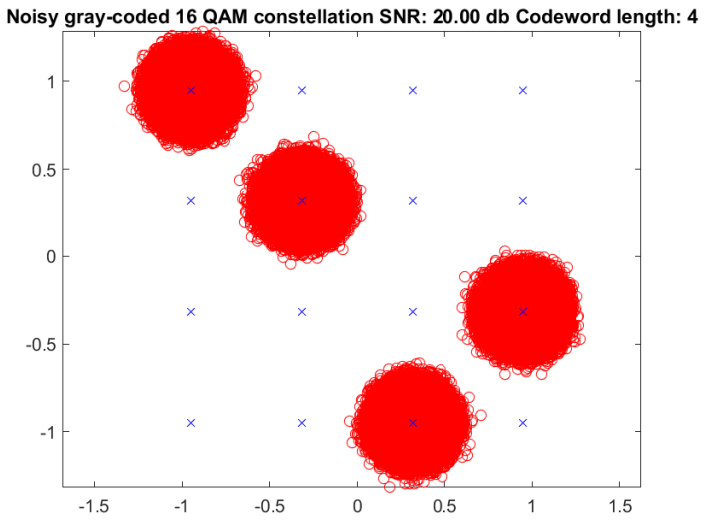

Figure 32

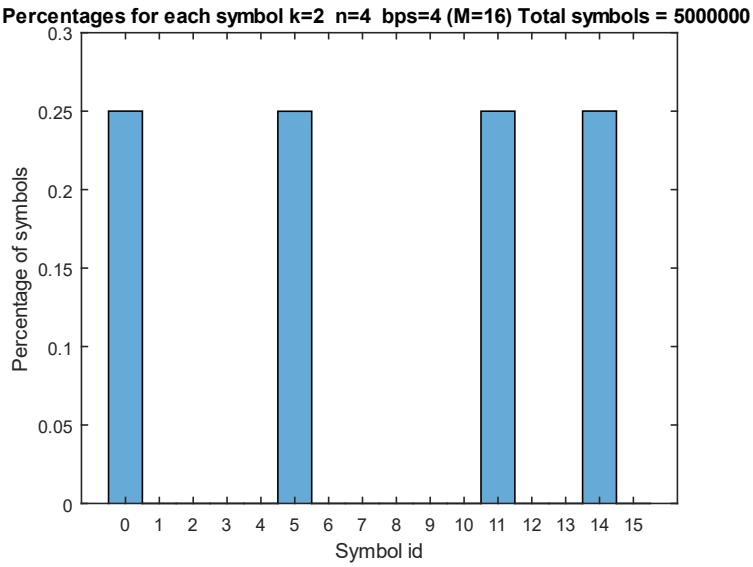

Figure 33

It is possible to know in advance which symbols will be used by considering all the possible ways codewords could be mapped to symbols. This could help the designer choose a suitable modulation order.

Part 1b

For this part, various ideas were investigated on how to utilize multiple channels to transmit information.

Increasing throughput using multiple channels

In this scenario, many channels are available. Each channel can have a different error probability and different capacity (bits per second).

Every time, the desired maximum BER is chosen. Then, the Block Error Rate (BLER) is calculated. BLER is defined as the probability that there will be an error in the message after the decoding process. In other words, BLER exists when there are erroneous bits that could not be corrected by the implemented error correction process.

This process results in the appropriate (usually different) codeword length for each channel in order to achieve the selected BER. Also, the data portion is calculated. Data portion represents the percentage of message data that should be sent through each channel, given the optimal codeword length for each channel, so that the transmission will start and end at the same time for all channels.

The results are displayed in the following tables and diagrams. As expected, when a lower BER is desired, the codeword length must increase. Also, when the channel has a larger error probability, a code of larger codeword length must be used. Finally, it can be seen that in worse channels (where larger codewords are used) the rate of useful information is lower, thus a smaller portion of the data ends up being sent using those channels.

Throughout all tests, all channels' capacity is 10^6 bits/s .

- **Run 1: 2 Channels, desired BER = 0.01, k = 2**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	3	0,002	2	0,67	0,667	6,67E+05	3,10E-03
2	2	6	0,050	4	0,33	0,333	3,33E+05	1,00E-02

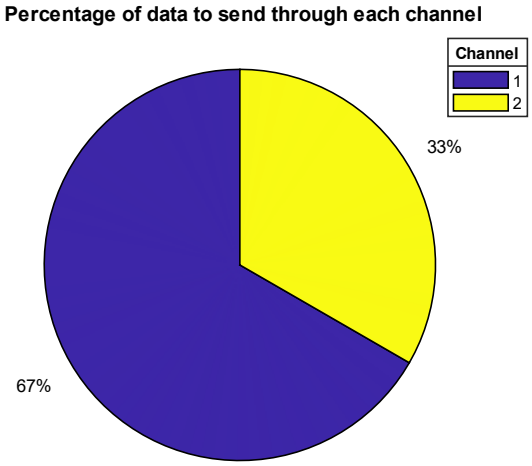

Figure 34 Run 1

- **Run 2: 2 Channels, desired BER = 10^{-4}, k = 2**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	5	0,002	3	0,40	0,737	4,00E+05	1,90E-05
2	2	14	0,050	9	0,14	0,263	1,43E+05	5,70E-05

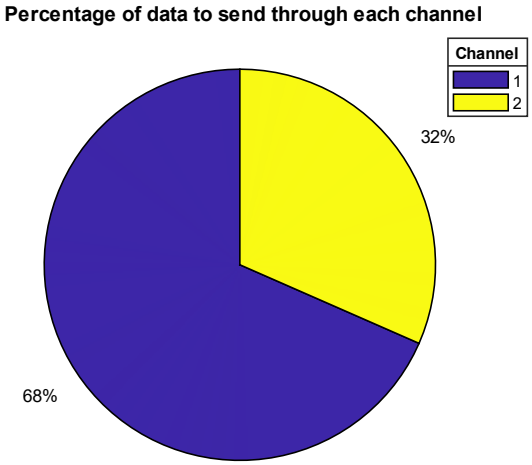

Figure 35 Run 2

- **Run 3: 2 Channels, desired BER = 0.01, k=5**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	5	6	0,002	2	0,83	0,684	8,33E+05	3,80E-03
2	5	13	0,050	5	0,38	0,316	3,85E+05	5,10E-03

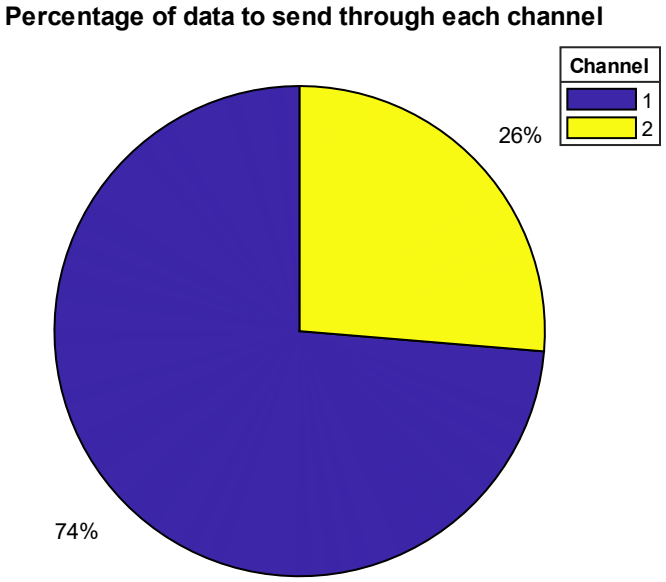

Figure 36 Run 3

- **Run 4: 10 Channels, desired BER = 0.01, k = 2, logarithmically spaced p**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	3	0,002	2	0,67	0,158	6,67E+05	2,40E-03
2	2	4	0,007	2	0,50	0,119	5,00E+05	3,20E-03
3	2	5	0,013	3	0,40	0,095	4,00E+05	1,30E-03
4	2	5	0,018	3	0,40	0,095	4,00E+05	2,10E-03
5	2	5	0,023	3	0,40	0,095	4,00E+05	2,40E-03
6	2	5	0,029	3	0,40	0,095	4,00E+05	4,10E-03
7	2	5	0,034	3	0,40	0,095	4,00E+05	6,50E-03
8	2	5	0,039	3	0,40	0,095	4,00E+05	8,10E-03
9	2	5	0,045	3	0,40	0,095	4,00E+05	9,50E-03
10	2	8	0,050	5	0,25	0,059	2,50E+05	2,40E-03

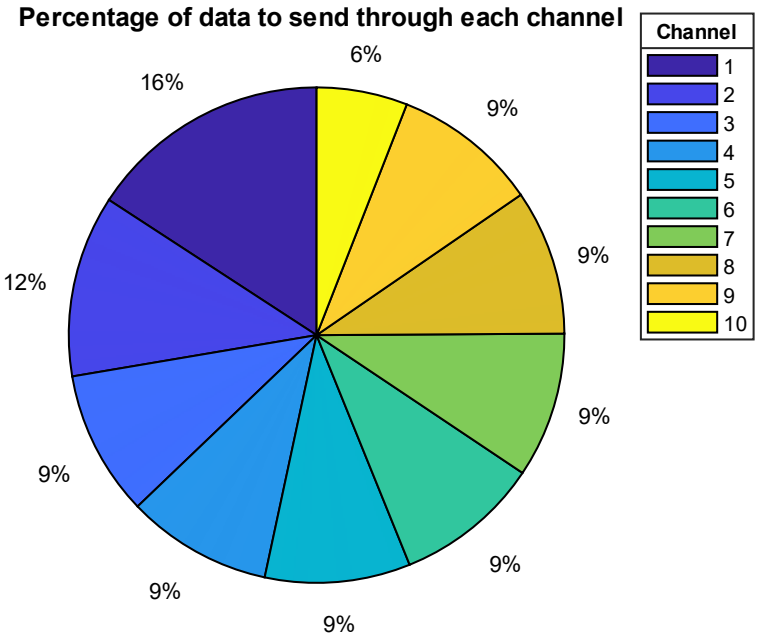

Figure 37 Run 4

- **Run 5: 10 Channels, desired BER = 10^{-3}, k = 2, logarithmically spaced p**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	5	0,002	3	0,40	0,139	4,00E+05	4,00E-05
2	2	5	0,007	3	0,40	0,139	4,00E+05	3,00E-04
3	2	5	0,013	3	0,40	0,139	4,00E+05	8,10E-04
4	2	7	0,018	4	0,29	0,100	2,86E+05	4,50E-04
5	2	8	0,023	5	0,25	0,087	2,50E+05	2,60E-04
6	2	8	0,029	5	0,25	0,087	2,50E+05	3,90E-04
7	2	8	0,034	5	0,25	0,087	2,50E+05	6,70E-04
8	2	8	0,039	5	0,25	0,087	2,50E+05	9,10E-04
9	2	10	0,045	6	0,20	0,070	2,00E+05	5,60E-04
10	2	11	0,050	7	0,18	0,063	1,82E+05	4,30E-04

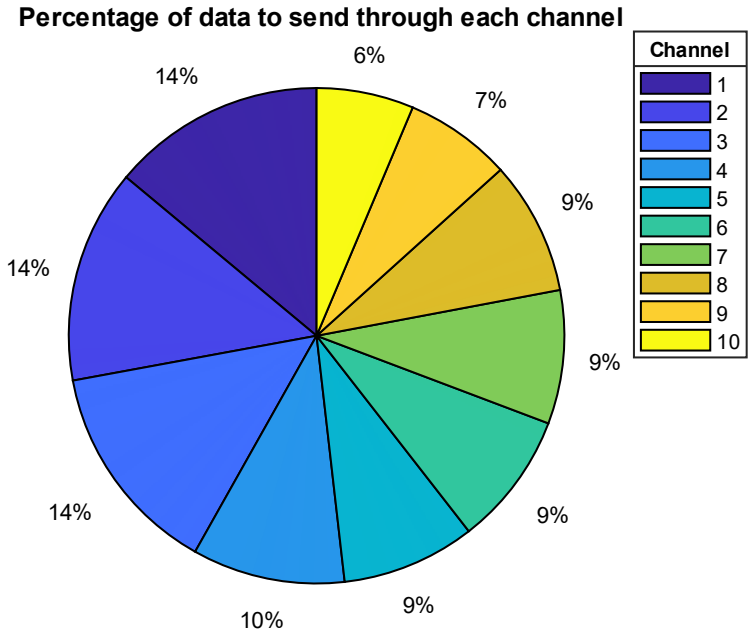

Figure 38 Run 5

- **Run 6: 10 Channels, desired BER = 10^{-4}, k = 2, logarithmically spaced p**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	5	0,002	3	0,40	0,187	4,00E+05	2,40E-05
2	2	7	0,007	4	0,29	0,134	2,86E+05	7,80E-05
3	2	8	0,013	5	0,25	0,117	2,50E+05	3,40E-05
4	2	10	0,018	6	0,20	0,094	2,00E+05	3,30E-05
5	2	10	0,023	6	0,20	0,094	2,00E+05	8,80E-05
6	2	11	0,029	7	0,18	0,085	1,82E+05	4,00E-05
7	2	11	0,034	7	0,18	0,085	1,82E+05	6,50E-05
8	2	13	0,039	8	0,15	0,072	1,54E+05	4,30E-05
9	2	14	0,045	9	0,14	0,067	1,43E+05	4,90E-05
10	2	14	0,050	9	0,14	0,067	1,43E+05	7,20E-05

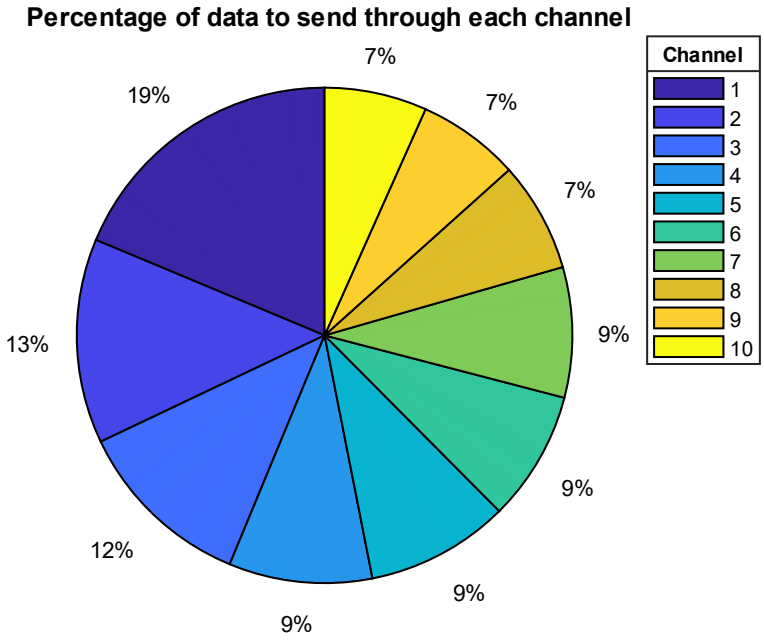

Figure 39 Run 6

- **Run 7: 10 Channels, desired BER = 0.01, linearly spaced k, logarithmically spaced p**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	3	0,002	2	0,67	0,130	6,67E+05	2,80E-03
2	2	4	0,007	2	0,50	0,098	5,00E+05	4,10E-03
3	3	6	0,013	3	0,50	0,098	5,00E+05	7,00E-04
4	3	6	0,018	3	0,50	0,098	5,00E+05	2,30E-03
5	4	7	0,023	3	0,57	0,112	5,71E+05	5,00E-03
6	4	7	0,029	3	0,57	0,112	5,71E+05	6,70E-03
7	5	9	0,034	3	0,56	0,109	5,56E+05	9,80E-03
8	5	11	0,039	4	0,45	0,089	4,55E+05	8,70E-03
9	6	15	0,045	5	0,40	0,078	4,00E+05	5,10E-03
10	6	15	0,050	5	0,40	0,078	4,00E+05	4,80E-03

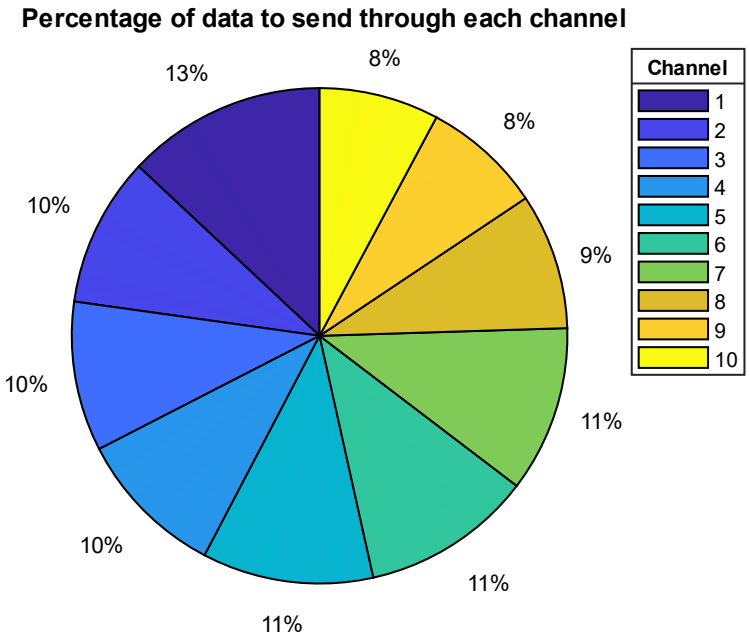

Figure 40 Run 7

- **Run 8: 10 Channels, desired BER = 0.01, linearly spaced k, constant p**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	4	0,010	2	0,50	0,088	5,00E+05	4,70E-03
2	2	4	0,010	2	0,50	0,088	5,00E+05	7,90E-03
3	3	6	0,010	3	0,50	0,088	5,00E+05	4,00E-04
4	3	6	0,010	3	0,50	0,088	5,00E+05	2,00E-04
5	4	7	0,010	3	0,57	0,101	5,71E+05	1,20E-03
6	4	7	0,010	3	0,57	0,101	5,71E+05	1,80E-03
7	5	7	0,010	2	0,71	0,126	7,14E+05	9,30E-03
8	5	8	0,010	2	0,63	0,110	6,25E+05	6,70E-03
9	6	10	0,010	3	0,60	0,106	6,00E+05	4,00E-04
10	6	10	0,010	3	0,60	0,106	6,00E+05	2,10E-03

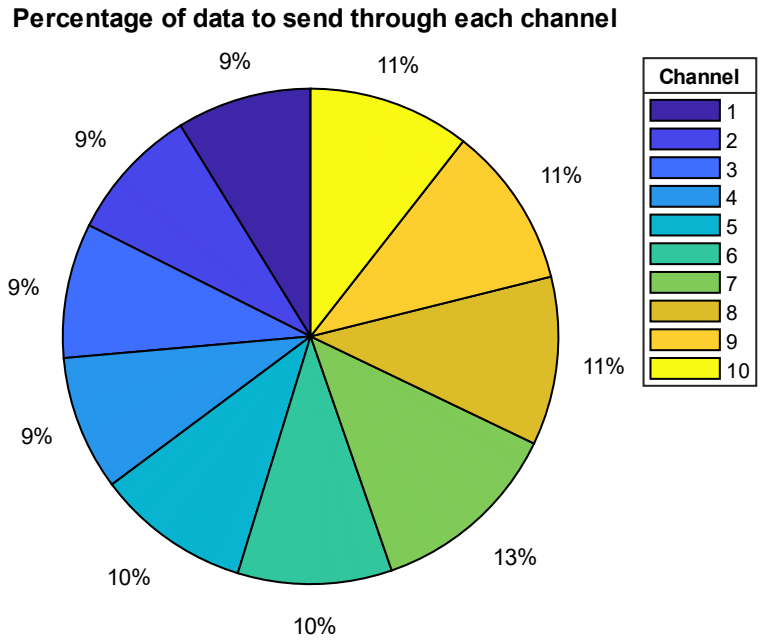

Figure 41 Run 8

- **Run 9: 10 Channels, desired BER = 0.01, linearly spaced k, constant p**

Channel	k	n	p	d_{min}	R	Data Portion	Bitrate	BER with ECC
1	2	3	0,002	2	0,67	0,085	6,67E+05	3,60E-03
2	2	3	0,002	2	0,67	0,085	6,67E+05	3,50E-03
3	3	4	0,002	2	0,75	0,096	7,50E+05	2,20E-03
4	3	4	0,002	2	0,75	0,096	7,50E+05	3,30E-03
5	4	5	0,002	2	0,80	0,102	8,00E+05	3,70E-03
6	4	5	0,002	2	0,80	0,102	8,00E+05	1,90E-03
7	5	6	0,002	2	0,83	0,107	8,33E+05	6,30E-03
8	5	6	0,002	2	0,83	0,107	8,33E+05	4,20E-03
9	6	7	0,002	2	0,86	0,110	8,57E+05	3,90E-03
10	6	7	0,002	2	0,86	0,110	8,57E+05	4,50E-03

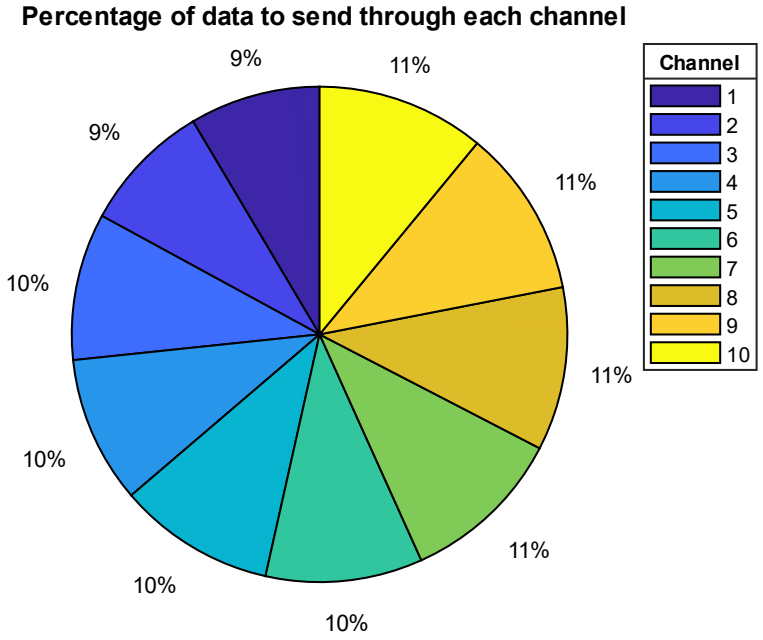

Figure 42 Run 9

Analyzing the Impact of Retransmission on Network Performance

Scenario

For this scenario, the additional time penalty for retransmission on error detection (T_{ack}) is taken into consideration. If the codeword doesn't have many redundant bits, there is a higher probability for an erroneous bit that the encoding algorithm cannot correct. Although the transmission is faster, it is likely that the codeword will have to be re-transmitted. On the other hand, if there are many redundant bits, the probability for an erroneous bit is significantly lower, but transmission takes much more time.

Analysis

Let X be the number of times a block needs to be transmitted until first successful transmission (no errors after error correction). It is apparent that X follows a geometric distribution with probability of success $\mathbf{1} - \mathbf{BLER}$. For a given channel and code, BLER is estimated using simulation. Thus, the expected number of times a block needs to be transmitted is:

$$\mathbf{E(X|n)} = \frac{\mathbf{1}}{\mathbf{1} - \mathbf{BLER}}$$

Given the *rate* of the communication channel (seconds/bit), the time it takes to transmit a block once is:

$$\mathbf{T_{block}}(\boldsymbol{n}) = \mathbf{rate} \cdot \mathbf{n}$$

Combining the above, it is evident that transmission time accounting for potential retransmissions can be modeled as:

$$\mathbf{T(n)} = \mathbf{T_{block}(n)} \cdot \mathbf{E(X|n)}$$

To model a delay (penalty) between each retransmission, another term can be added to compensate as follows:

$$\mathbf{T'(n)} = \mathbf{T_{block}(n)} \cdot \mathbf{E(X|n)} + \mathbf{T_{ack}} \cdot (\mathbf{E(X|n)} - \mathbf{1})$$

Simulation

Different T_{ack} were tested for constant message length k=2 and channel throughput (rate) is 10^{-6} seconds/bit. The codeword length n varies from 3 to 10. The length of the transmitted data is 10^7. The results are shown below.

It is observed that when the retransmission penalty is insignificant or when the probability of error is low, it is more efficient to do a retransmission instead of increasing the code word length. However, when the penalty is significant or when errors are more likely, then increasing the code

word length becomes beneficial for minimizing the average transmission time, as the costly retransmissions are avoided.

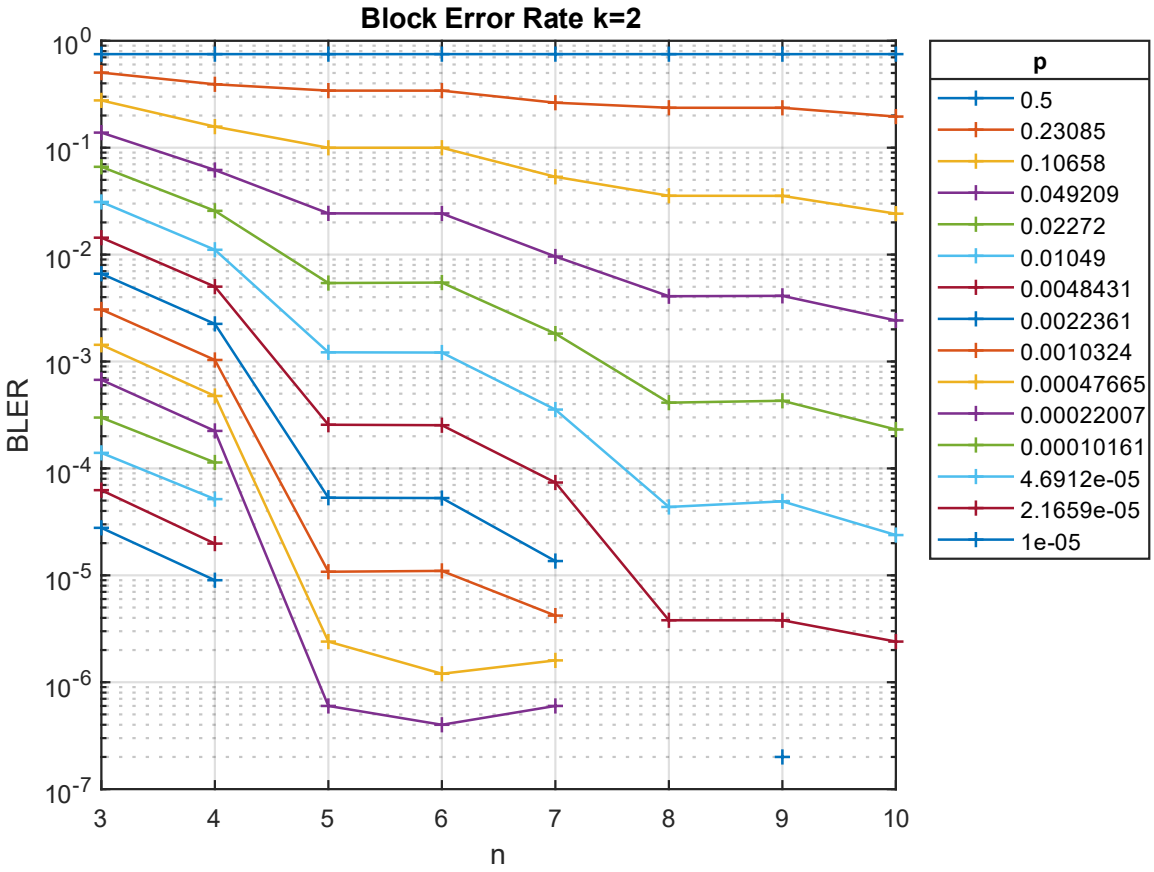

Figure 43 BLER

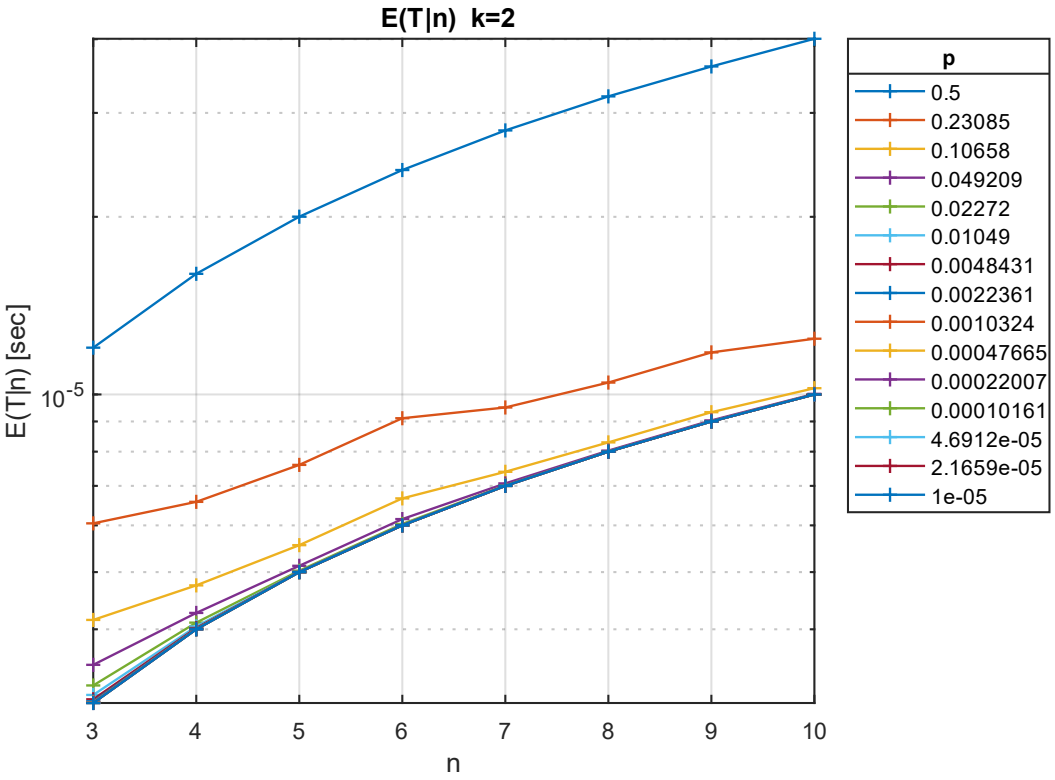

Figure 44 Expected Block Retransmissions without T_{ack}

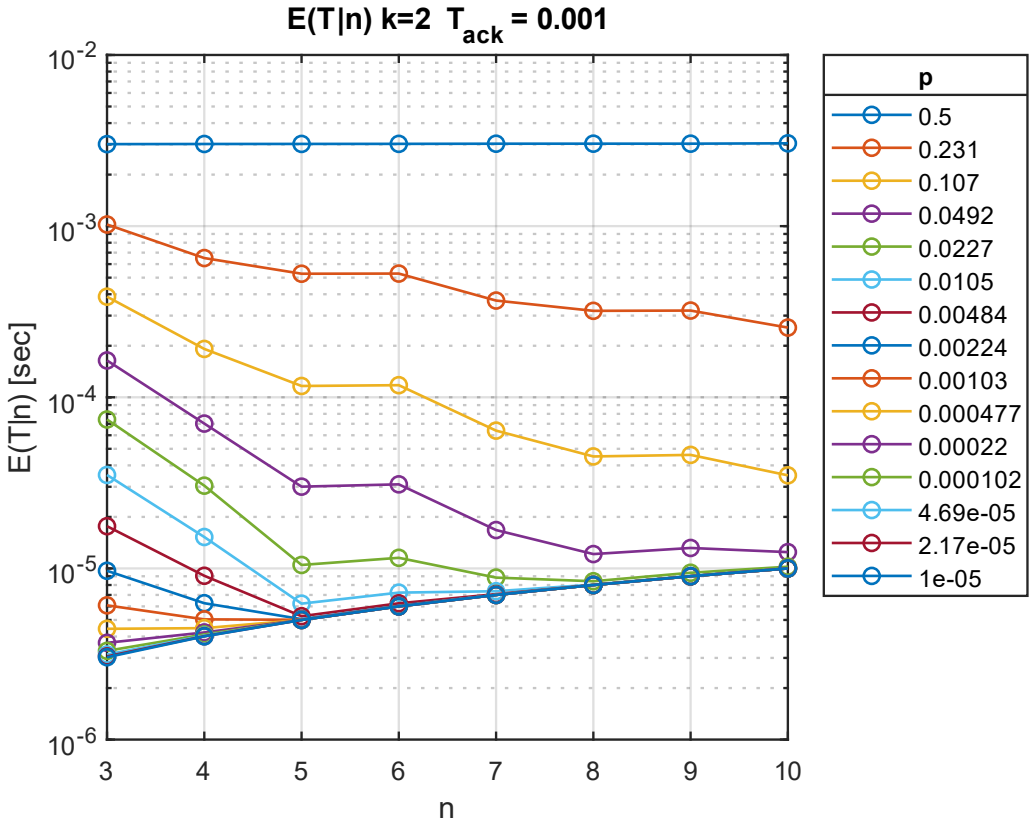

Figure 45 Expected Block Retransmissions with T_{ack}=0.001

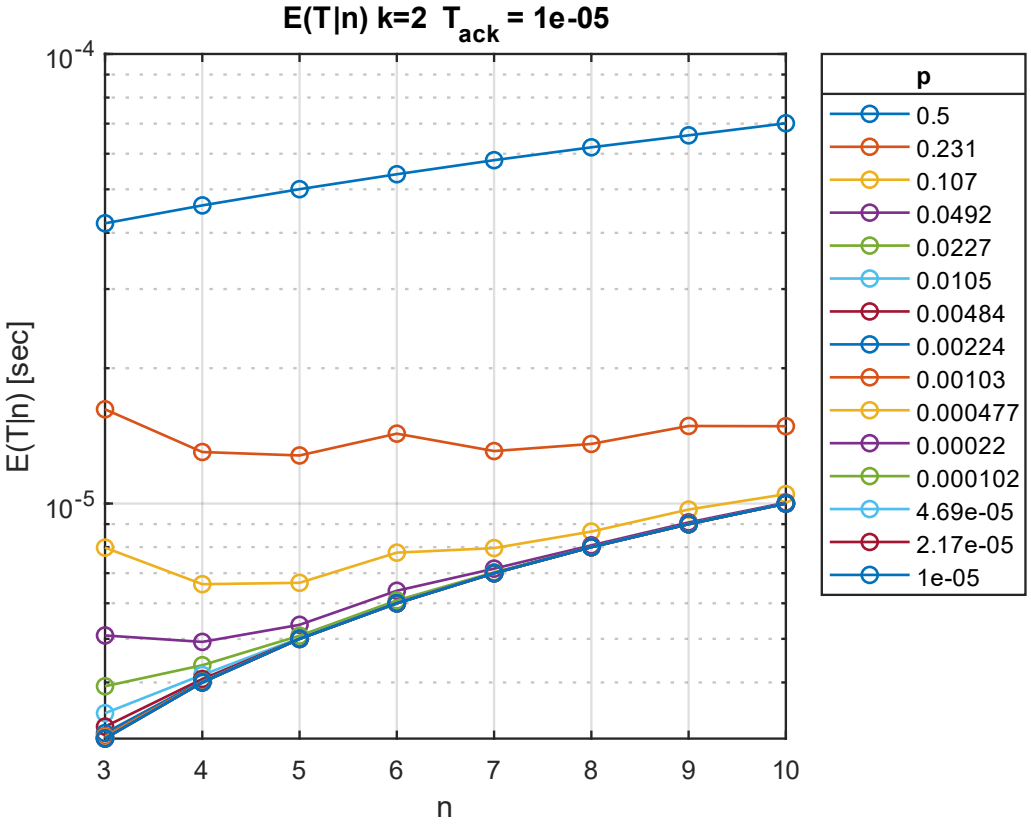

Figure 46 Expected Block Retransmissions with $T_{ack}=10^{-5}$

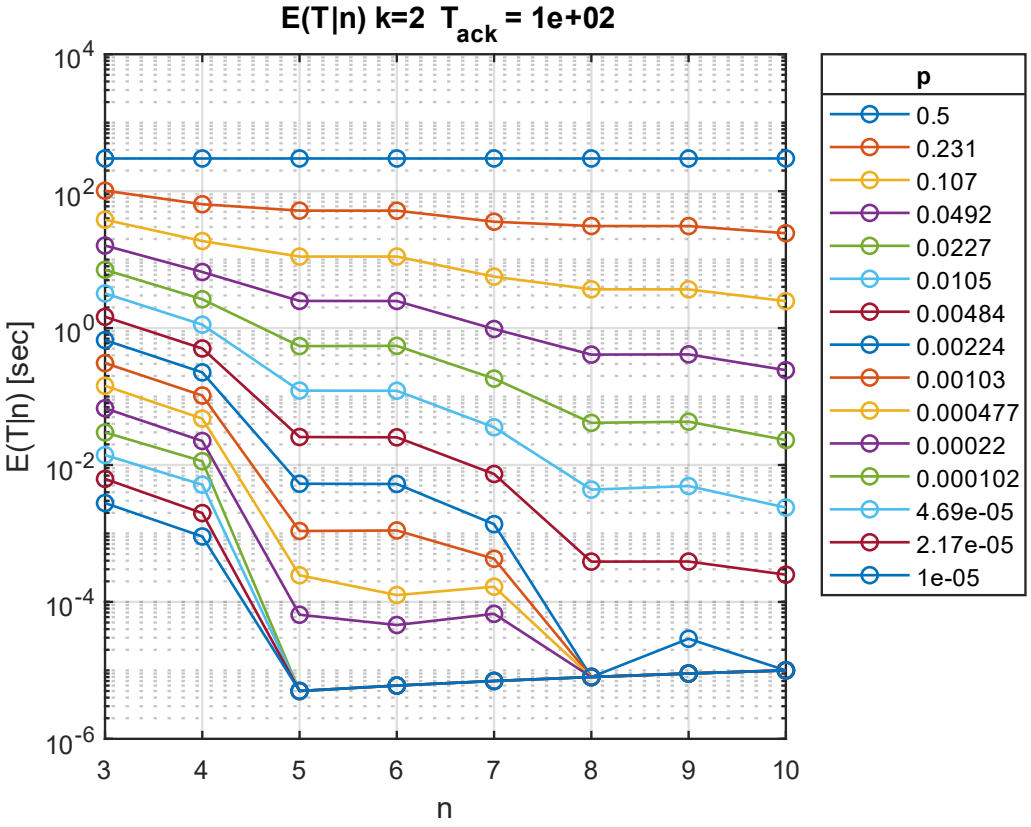

Figure 47 Expected Block Retransmissions with $T_{ack}=10^{2}$

Decoding for Channels with Heterogeneous Error Rates Across Bits

When combining multiple channels of different characteristics to send information, there may occur a case where different bits in the received bit-stream are expected to have a different probability of error. In this section, probability analysis is used to create an optimal decoder for such a scenario.

Scenario

Suppose in a block coding scenario, 2 bits of information (k=2) are encoded into a 4-bit codeword (n=4):

Message		**Codeword**
00	→	0000
01	→	0111
10	→	1010
11	→	1101

Additionally, suppose that the codeword is sent using a channel that transmits information in 4-bit groups, with each bit having a different error probability $\boldsymbol{p} = [\mathbf{0.4},\ \mathbf{0.01},\ \mathbf{0.01},\ \mathbf{0.01}]$.
If codeword 1010 is transmitted and the receiver gets 0010 due to an error, a decoder based on the minimum Hamming distance would find 0000 and 1010 equally likely, since both are one Hamming distance away from the received word. However, assuming all messages have equal probability of being transmitted, the first bit's higher error likelihood suggests 1010 was the original codeword.
From this observation and given the channel's distinct bit error probabilities, probability analysis can be used to create an optimal mapping from all possible 2^n received words to the code's 2^k codewords, which directly map to the original information message:

Received		**Codeword**		**Message**
0000	→	0000	→	00
0001	→	0000	→	00
0010	→	0000	→	00
…		…		…
1110	→	1010	→	10
1111	→	0111	→	01

Analysis

Assuming $\mathbf{p_l}$, $\mathbf{l} \in \{\mathbf{1}, \dots, \mathbf{n}\}$ the probability of the $\mathbf{l^{th}}$ bit to change and $\mathbf{t_i}$, $\mathbf{i} \in \{\mathbf{0}, \mathbf{1}, \dots, \mathbf{2^k - 1}\}$ the message to be transmitted, it has a cardinality of $\mathbf{2^k}$ equal to the number of messages. Also $\mathbf{r_j}$, $\mathbf{j} \in \{\mathbf{0}, \mathbf{1}, \dots, \mathbf{2^n - 1}\}$ the received message which has a cardinality of $\mathbf{2^n}$ equal to the number of codewords.

Assuming that each message is equally likely to be transmitted, the probability of some message $\boldsymbol{t_i}$ being transmitted can expressed as:

$$\mathbf{P(t_i) = \frac{1}{2^k}, \forall i \in \{0, 1, \dots, 2^k - 1\}}$$

For further analysis, the probability of receiving codeword $\boldsymbol{r_j}$ is expressed as follows (law of total probability):

$$\mathbf{P(r_j) = \sum_{i=0}^{i=2^k-1} P(r_j \mid t_i) \cdot P(t_i), \quad j \in \{0, 1, \dots, 2^n - 1\}} \Rightarrow$$

$$\mathbf{P(r_j) = \sum_{i=0}^{i=2^k-1} P(r_j \mid t_i) \frac{1}{2^k}}$$

In the above equation, $\boldsymbol{P(r_j \mid t_i)}$ expresses the probability that, given that message $\boldsymbol{t_i}$ has been transmitted, the received message is $\boldsymbol{r_j}$. This probability can be calculated by considering, for each possible transmitted codeword, the probability that the specific error pattern that produces $\boldsymbol{r_j}$ has occurred.
All possible error patterns can be expressed as:

$$\mathbf{mask_{i,j,l}} = \begin{cases} \mathbf{0}, & \textbf{if the } \mathbf{l^{th}} \textbf{ bit of } \mathbf{r_j} \textbf{ is the same as} \\ & \textbf{the corresponding bit of } \mathbf{t_i \cdot G} \\ \mathbf{1}, & \textbf{otherwise} \end{cases}$$

As $\boldsymbol{p_l}$ is the error probability for the $\mathbf{l^{th}}$ bit, the probability of interest is calculated as:

$$\mathbf{P(r_j \mid t_i) = \prod_{l=1}^{n} \left(p_l \cdot (mask_{i,j,l}) + (1 - p_l) \cdot \left(1 - mask_{i,j,l}\right)\right)}$$

When $\boldsymbol{r_j}$ is received the best way to decode it is:

$$\mathbf{\arg\max_i} P(t_i \mid r_j)$$ [2]

Using Bayes' theorem and the above analysis, it is rewritten as:

$$\mathbf{P(t_i \mid r_j) = \frac{P(r_j \mid t_i) \cdot P(t_i)}{P(r_j)}} \Rightarrow$$

[2] This can result in a single element or a set.

$$\mathbf{P}(\mathbf{t_i}\,|\mathbf{r_i}) \;=\; \frac{\mathbf{P}(\mathbf{r_j}\,|\mathbf{t_i})\cdot\mathbf{P}(\mathbf{t_i})}{\sum_{\mathbf{i=0}}^{\mathbf{i=2^k-1}}\mathbf{P}(\mathbf{r_j}\,|\mathbf{t_i})\cdot\mathbf{P}(\mathbf{t_i})} \Rightarrow$$

$$\mathbf{P}(\mathbf{t_i}\,|\mathbf{r_i}) \;=\; \frac{\mathbf{P}(\mathbf{r_j}\,|\mathbf{t_i})}{\sum_{\mathbf{i=0}}^{\mathbf{i=2^k-1}}\mathbf{P}(\mathbf{r_j}\,|\mathbf{t_i})}$$

Thus, the message that was most likely transmitted for a given received word can be calculated as:

$$\arg\max_{i} P(t_i\,|r_j) = \arg\max_{i}\left(\frac{P(r_j\,|t_i)}{\sum_{i=0}^{i=2^k-1}P(r_j\,|t_i)}\right) = \arg\max_{i} P(r_j\,|t_i)$$

This is used to implement an algorithm that constructs a probability-optimal decoding table for a given $\mathbf{p_l}$ (channel) and $\mathbf{G}$ (code).

Below is an example of this algorithm used on a channel with bit error probabilities:

$$\mathbf{p} = [\mathbf{0.0081}, \mathbf{0.1083}, \mathbf{0.0699}, \mathbf{0.2889}]$$

For code

Message		Codeword
00	→	0000
01	→	0111
10	→	1010
11	→	1101

The algorithm produces the following decoding table:

Received		Codeword		Message
0000	→	0000	→	00
0001	→	0000	→	00
0010	→	0000	→	00
0011	→	0111	→	01
0100	→	0000	→	00
0101	→	0111	→	01
0110	→	0111	→	01
0111	→	0111	→	01
1000	→	1010	→	10
1001	→	1101	→	11
1010	→	1010	→	10
1011	→	1010	→	10
1100	→	1101	→	11
1101	→	1101	→	11
1110	→	1010	→	10
1111	→	1101	→	11

As a result, the following diagram is produced

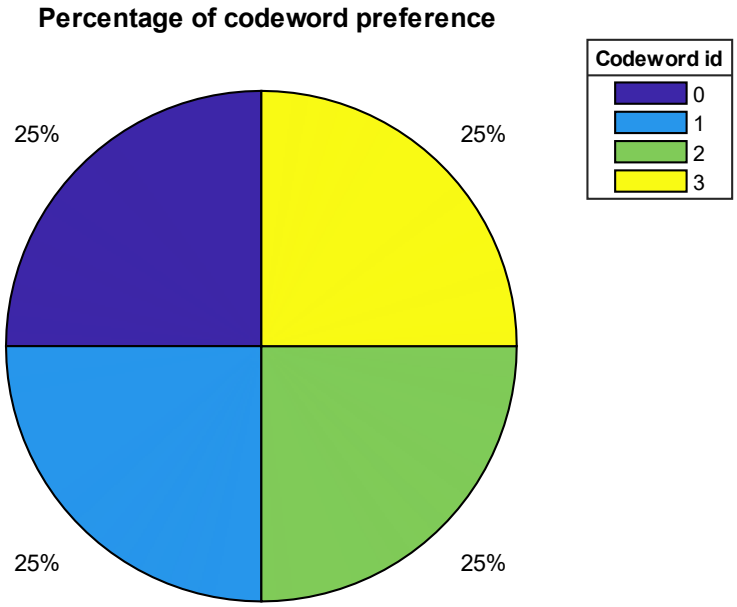

Figure 48

All codewords are used with the same probability.

Part 2

This part of the assignment focuses on Low-Density Parity-Check (LDPC) codes used on the binary erasure channel. The task is to select design parameters for an irregular LDPC code and compare it to a regular LDPC code of the same code rate and codeword size. The potential improvements offered by irregular LDPC codes are investigated.

LDPC codes as polynomials

A single LDPC code is described by a sparse parity-check matrix H.
In addition, the distribution of constraints (parity checks) in an LDPC code can be generally modeled using polynomials:

$$\boldsymbol{\Lambda(x) = \Lambda_1 x + \Lambda_2 x^2 + \ldots + \Lambda_{l_{max}} x^{l_{max}}}$$
$$\boldsymbol{P(x) = P_1 x + P_2 x^2 + \ldots + P_{r_{max}} x^{r_{max}}}$$

where $\boldsymbol{\Lambda_k}$ the number of rows in $\boldsymbol{H}$ with $\boldsymbol{k}$ constraints, and $\boldsymbol{P_j}$ the number of columns in $\boldsymbol{H}$ with $\boldsymbol{j}$ constraints.
These two polynomials do not define a single LDPC code, rather they describe a family of possible codes, which are expected to achieve similar performance. In other words, for distributions defined by given $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$**,** there can be many parity-check matrices H that satisfy those distributions.

Generating Irregular LDPCs

Irregular LDPC codes are generated by solving a series of optimization problems to achieve erasure correction ability over certain channel characteristics.
The first optimization problem is, in essence, finding a distribution of the constraints such that the code rate is maximized, while ensuring that erasure correction can take place. This is done by calculating polynomials $\boldsymbol{\lambda(x)}$ and $\boldsymbol{\rho(x)}$, a normalized and codelength-independent version of polynomials $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$.
It is claimed[3] that the $\boldsymbol{\rho(x)}$ that achieves optimal performance is check-concentrated, meaning that the degree of column constraints is best kept around a selected $\boldsymbol{r_{avg}}$, as suggested by the following formula:

[3] Richardson & Urbanke, Modern Coding Theory, p. 115

$$\boldsymbol{\rho(x) = \frac{r(r+1-r_{avg})}{r_{avg}} x^{r-1} + \frac{r_{avg}-r(r+1-r_{avg})}{r_{avg}} x^{r}, \quad r = \lfloor r_{avg} \rfloor}$$

Probabilistic analysis also yields the following linear optimization problem for finding *λ(x)*:

$$\begin{aligned} \max_{\lambda_2,\ldots,\lambda_{l_{\max}}} \quad & \textstyle\sum_{i\geq 2} \frac{\lambda_i}{i} \\ \text{s.t.} \quad & \mathrm{C}_1 : \textstyle\sum_{i\geq 2} \lambda_i = 1, \\ & \mathrm{C}_2 : \varepsilon \textstyle\sum_{i\geq 2} \lambda_i (1-\rho(1-x))^{i-1} - x \leq 0,\ \forall x \in (0,1) \\ & \mathrm{C}_3 : \lambda_i \geq 0, \forall i \geq 2, \end{aligned}$$

Here $\mathbf{r_{avg}}$, $\mathbf{l_{max}}$ and $\boldsymbol{\varepsilon}$ are design parameters.
The above optimization problem is solved in Matlab using the *linprog* function, giving us optimal polynomials $\boldsymbol{\rho(x)}$ and $\boldsymbol{\lambda(x)}$ for the selected parameters.

The next problem is finding polynomials $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$ that most closely follow the code rate of $\boldsymbol{\lambda(x)}$, $\boldsymbol{\rho(x)}$. Here, an exact method can be used for adhering exactly to the desired code rate, however this generates unreasonably large code word length ($\boldsymbol{n}$).
A second optimization problem is used instead, which results to an approximate solution for $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$ that closely follow the code rate of $\boldsymbol{\lambda(x)}$, $\boldsymbol{\rho(x)}$ for a desired codeword length $\boldsymbol{n}$.
For $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$ to describe valid LDPC distributions, their coefficients must be non-negative integers. Thus, they are firstly approximated as follows:

$$\boldsymbol{\hat{\Lambda}_i = \left\lfloor \frac{\lambda_i / i}{\sum_{j=2}^{l_{max}} \lambda_j / j} n \right\rfloor, \quad \hat{P}_i = \left\lfloor \frac{\rho_i / i}{\sum_{j=2}^{l_{max}} \lambda_j / j} n \right\rfloor}$$

Due to the floor function in this approximation, some decimal portion has been truncated. For $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$ to be valid LDPC polynomials, the following conditions must hold:

$$\boldsymbol{\sum_i \Lambda_i = n \ , \quad \sum_i i\Lambda_i = \sum_i iP_i}$$

Thus, some correction must be introduced to derive valid $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$. This correction is found by solving the following set of integer linear optimization problems, which optimize for minimal code rate deviation from the code rate of $\boldsymbol{\lambda(x)}$, $\boldsymbol{\rho(x)}$*:*

$$\sum_{i=2}^{r_{\max}} \hat{x}_i^P - A \geq 0$$

$$\begin{aligned} \min_{\hat{x}_2^\Lambda,\ldots,\hat{x}_{l_{\max}}^\Lambda,\hat{x}_2^P,\ldots,\hat{x}_{r_{\max}}^P} \quad & \textstyle\sum_{i=2}^{r_{\max}} \hat{x}_i^P \\ \text{s.t.} \quad & \mathrm{C}_1 : \textstyle\sum_{i=2}^{l_{\max}} \hat{x}_i^\Lambda = n - \sum_{i=2}^{l_{\max}} \hat{\Lambda}_i, \\ & \mathrm{C}_2 : \textstyle\sum_{i=2}^{l_{\max}} i\hat{x}_i^\Lambda - \sum_{i=2}^{r_{\max}} i\hat{x}_i^P = \sum_{i=2}^{r_{\max}} i\hat{P}_i - \sum_{i=2}^{l_{\max}} i\hat{\Lambda}_i, \\ & \mathrm{C}_3 : \hat{x}_i^\Lambda \in \{0,1\}, \hat{x}_i^P \in \{0,1\}, \\ & \mathrm{C}_4 : \textstyle\sum_{i=2}^{r_{\max}} \hat{x}_i^P \geq \lceil A \rceil. \end{aligned}$$

$$\max_{\hat{x}_2^{\Lambda},\ldots,\hat{x}_{l_{\max}}^{\Lambda},\hat{x}_2^{P},\ldots,\hat{x}_{r_{\max}}^{P}} \sum_{i=2}^{r_{\max}} \hat{x}_i^P$$

$$\text{s.t.} \quad \sum_{i=2}^{r_{\max}} \hat{x}_i^P - A \leq 0$$

$$\begin{aligned}
&C_1 : \textstyle\sum_{i=2}^{l_{\max}} \hat{x}_i^{\Lambda} = n - \sum_{i=2}^{l_{\max}} \hat{\Lambda}_i, \\
&C_2 : \textstyle\sum_{i=2}^{l_{\max}} i\hat{x}_i^{\Lambda} - \sum_{i=2}^{r_{\max}} i\hat{x}_i^{P} = \sum_{i=2}^{r_{\max}} i\hat{P}_i - \sum_{i=2}^{l_{\max}} i\hat{\Lambda}_i, \\
&C_3 : \hat{x}_i^{\Lambda} \in \{0,1\}, \hat{x}_i^{P} \in \{0,1\}, \\
&C_4 : \textstyle\sum_{i=2}^{r_{\max}} \hat{x}_i^{P} \leq \lfloor A \rfloor.
\end{aligned}$$

where:

$$A = \sum_{i=2}^{r_{max}} \left(P_i - \widehat{P}_i\right)$$

The above problems are solved using the Matlab function *intlinprog*.

Generating Regular LDPCs

Regular LDPC codes are LDPC codes where there is an equal number of constraints among the rows of H, and an equal number of constraints among the columns of H. Thus, the polynomials that model the distributions of regular LDPC codes are of the form:

$$\begin{aligned}
\mathbf{\Lambda(x)} &= \mathbf{\Lambda x^l} \\
\mathbf{P(x)} &= \mathbf{P x^r}
\end{aligned}$$

where codeword length is $\mathbf{n} = \mathbf{\Lambda}$ and $\mathbf{\Lambda}$, $\mathbf{P}$, $\mathbf{l}$, $\mathbf{r}$ are positive integers selected such that:

$$\mathbf{l \cdot \Lambda = r \cdot P}$$

The code rate is:

$$\mathbf{r = \frac{k}{n} = \frac{\Lambda - P}{\Lambda}}$$

For the purposes of comparison, the problem of interest is to generate regular LDPC codes of the same codeword length and code rate as a given irregular LDPC code. To do this, the following equations must hold:

$$\mathbf{n = n_{irregular}}$$

$$\mathbf{r = r_{irregular}}$$

while

$$\mathbf{r \leq r_{max}} \text{ and } \mathbf{l \leq l_{max}}$$

Adhering to all these restrictions is usually not possible, because the arbitrary code rate that is imposed by the irregular code leads to $\mathbf{\Lambda}$ and $\mathbf{P}$ that have a very large common multiple, and thus need very large $\mathbf{l}$, $\mathbf{r}$ to satisfy $\mathbf{l} \cdot \mathbf{\Lambda} = \mathbf{r} \cdot \mathbf{P}$.

Thus, the strict restriction of $\mathbf{r} = \mathbf{r_{irregular}}$ is dropped, and the following algorithm is used instead to get potential regular LDPC codes with $\mathbf{r} \approx \mathbf{r}_{irregular}$:

1. $\mathbf{\Lambda}$ is set equal to $\mathbf{n_{irregular}}$
2. For each $\mathbf{l}$ between 2 and $\mathbf{l}_{max}$:
    a. The number of constraints (tanner graph edges) is calculated as $\mathbf{n\_edges} = \mathbf{\Lambda} \cdot \mathbf{l}$. Knowing that P must be a divisor of $\mathbf{n\_edges}$, all possible values of P are found.
    b. From possible values of $\mathbf{P}$, calculate $\mathbf{r} = \mathbf{l} \cdot \mathbf{\Lambda}/\mathbf{P}$ and discard those that yield $\mathbf{r} > \mathbf{rmax}$
    c. From remaining $\mathbf{P}$, select that which results in code rate closest to $\mathbf{r_{irregular}}$
    d. Store $\mathbf{\Lambda}$, $\mathbf{P}$, $\mathbf{l}$, $\mathbf{r}$ as a candidate regular LDPC.
3. Return all stored regular LDPCs for different values of $\mathbf{l}$

This algorithm returns multiple regular LDPC codes, each generated based on a different value of $\mathbf{l}$. Since it is not clear which one of these will perform better, all of them are tested against the irregular LDPC.

As an example, for given Irregular LDPC code:

Λ(X)	P(X)	RATE
$72x^2 + 43x^3 + 5x^4$	$x^6 + 41x^7$	0.65

The algorithm generates the following regular LDPC codes, which have similar rates to the irregular LDPC code:

Λ(X)	P(X)	RATE
$120x^2$	$40x^6$	0.667
$120x^3$	$40x^9$	0.667
$120x^4$	$40x^{12}$	0.667

Generating the parity check matrix H

The next problem of interest is generating $\mathbf{H}$ for given Λ(x) and P(x). This is achieved by the following algorithm:

1. An empty matrix H of the desired dimensions is created, and its rows and columns are labeled using a unique identifier (id).
2. A pool of row identifiers is created, $\mathbf{row\_id\_pool}$. Row ids are added to the pool according to **Λ(x)**, such that there are $\mathbf{\Lambda_i}$ rows with i occurrences of their id in $\mathbf{row\_id\_pool.}$
3. A pool of column identifiers is created, $\mathbf{col\_id\_pool}$. Similarly, column ids are added to the pool according to $\mathbf{P(x)}$, such that there are $\mathbf{P_i}$ columns with i occurrences of their id in $\mathbf{col\_id\_pool}$.
4. Rows and columns from the two pools are randomly matched. Duplicate row-column pairs are reshuffled to reduce duplicates.

As an example, let's examine how parity matrix H is generated for an LDPC described by:

$$\mathbf{\Lambda(x)} = \mathbf{7x^2 + 2x^3}$$
$$\mathbf{P(x)} = \mathbf{4x^5}$$

Row and columns id pools are created according to the polynomials:

$$\boldsymbol{row\_id\_pool} = [\mathbf{1,1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,8,9,9,9}]$$
$$\boldsymbol{column\_id\_pool} = [\mathbf{1,1,1,1,1,2,2,2,2,2,3,3,3,3,3,4,4,4,4,4}]$$

The pools are then shuffled:

$$\boldsymbol{row\_id\_pool} = [\mathbf{8,6,3,9,5,2,7,8,9,7,1,6,4,9,1,3,4,2,5,8}]$$
$$\boldsymbol{column\_id\_pool} = [\mathbf{3,2,3,2,4,3,2,1,4,2,1,4,1,1,3,4,4,1,2,3}]$$

Pairs $(\mathbf{x,y}) \in \{ (\mathbf{8,3}), (\mathbf{6,2}), (\mathbf{3,3}), (\mathbf{9,2}), \dots \}$ are the coordinates where $\mathbf{H(x,y) = 1}$.

For duplicate pairs $(\mathbf{8,3})$ and $(\mathbf{7,2})$ (which appear twice), the column coordinate is swapped, creating pairs $(\mathbf{8,2})$, $(\mathbf{7,3})$. In this case, this eliminates duplicate entries in $\mathbf{H}$.

Performing Erasure Correction

For performing erasure correction, a belief-propagation method was used. Using a Matlab GUI, a step-by-step visualization of a Tanner Graph was created, which is a common graphical representation of LDPC codes.

Below is an example of erasure correction on an irregular LDPC code with $\mathbf{n = 9}$ and $\mathbf{k = 5}$.

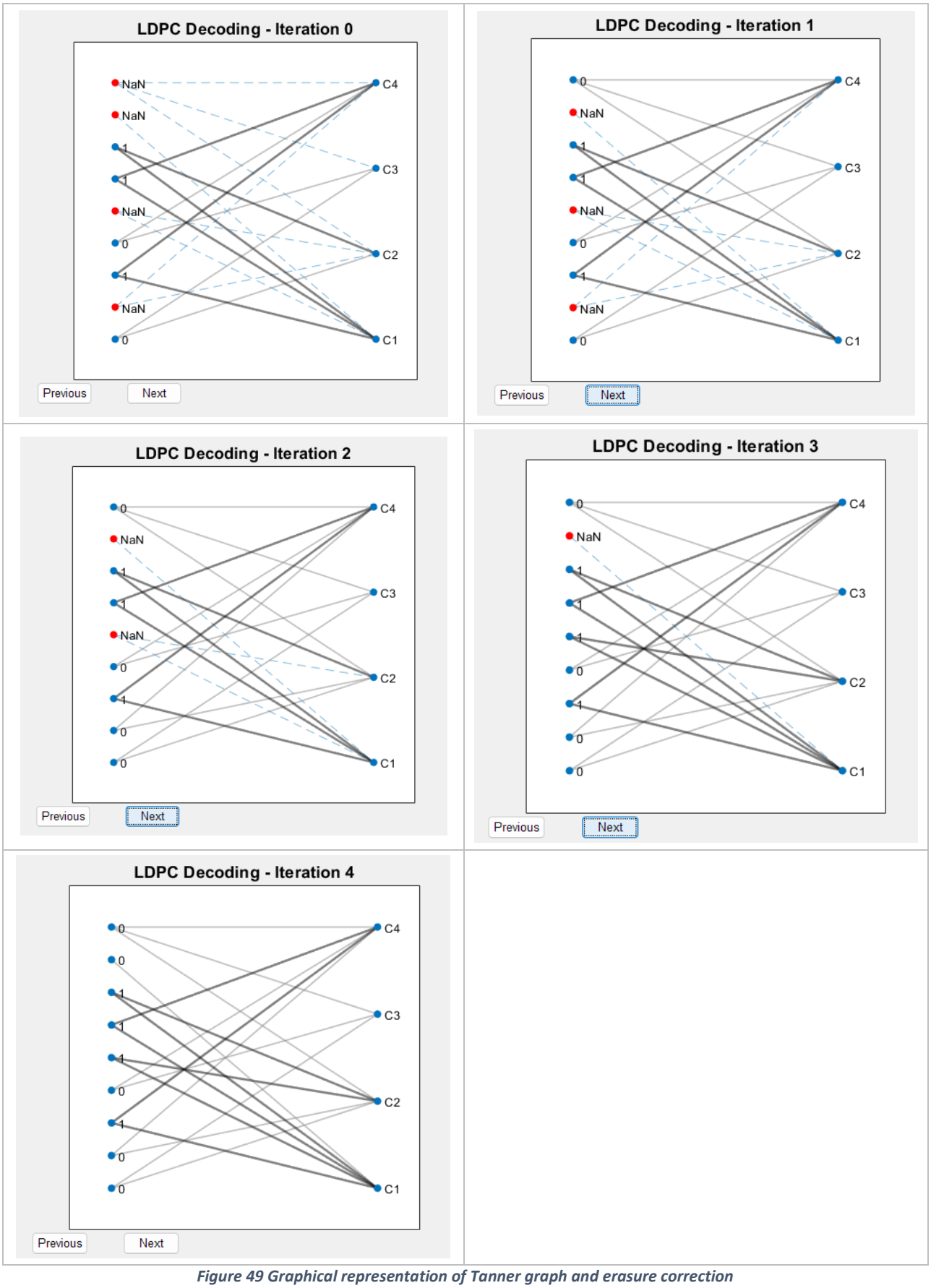

Figure 49 Graphical representation of Tanner graph and erasure correction

Simulation

All parts described above are composed to achieve the final simulation. Firstly, the irregular LDPC code polynomials are constructed, using the optimization process that has been mentioned. Then the regular LDPC code construction algorithm constructs several pairs of regular LDPC polynomials with the same codeword size and a similar code rate to the irregular LDPC. Finally, each LDPC code is tested on the binary erasure channel. Given the erasure probability of the channel, each LDPC code is evaluated on the erasure rate after applying correction.

There are two main non-deterministic processes that must be repeated for multiple iterations for each code, to gain a fair insight on its performance. Those affect the computational workload and accuracy of the simulation:
Firstly, polynomials $\boldsymbol{\Lambda(x)}$ and $\boldsymbol{P(x)}$ define the distribution of parity checks of the LDPC code. This means that for each pair of these polynomials, multiple instances of the code must be created, where each instance is a different parity-check matrix $\boldsymbol{H}$ that follows that distribution.
Secondly, for each $\boldsymbol{H}$ multiple messages must be sent through the erasure channel, each message with random erasures applied to it. It must be noted that only the erasure pattern on the received message (and not the binary information itself) is important for testing erasure correction.

First, the behavior of irregular LDPC codes designed by the linear programming process is examined for different design parameters ($\boldsymbol{n = 100}$, $\boldsymbol{l_{max} = 10}$):

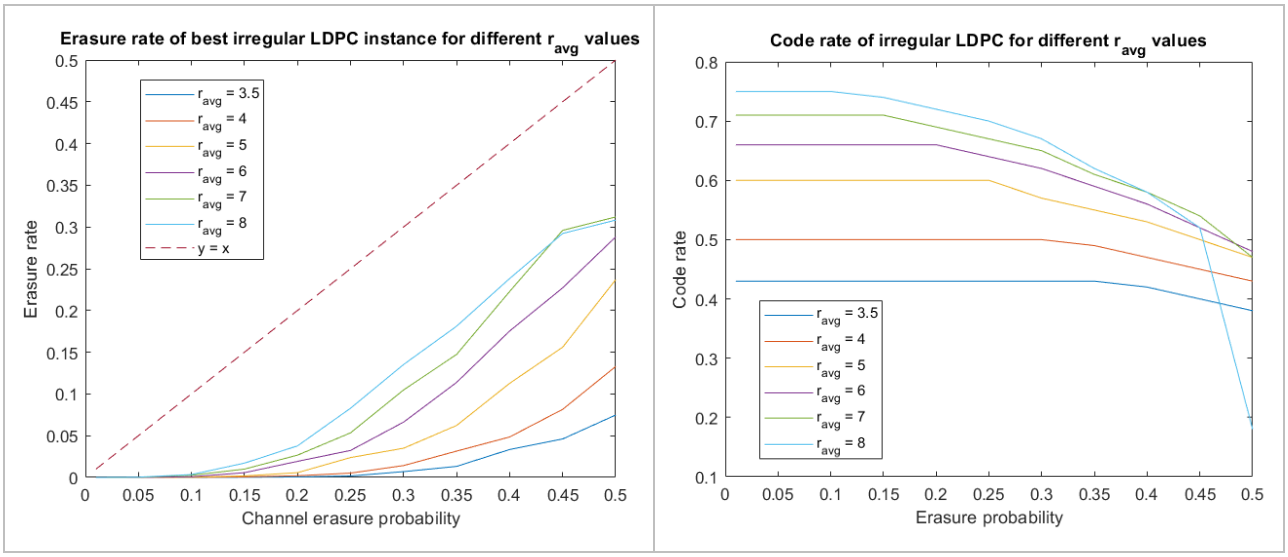

Figure 50 Erasure rate and Code rate of irregular LDPC codes for different r_{avg} values

It is observed that a higher value of $\boldsymbol{r_{avg}}$ (average number of edges per check node) leads to a higher code rate, in the expense of worse performance in terms of erasure correction.

An important relation that has to be investigated is the one between Bir Error Rate and erasure rate. Bellow is the relevant diagram:

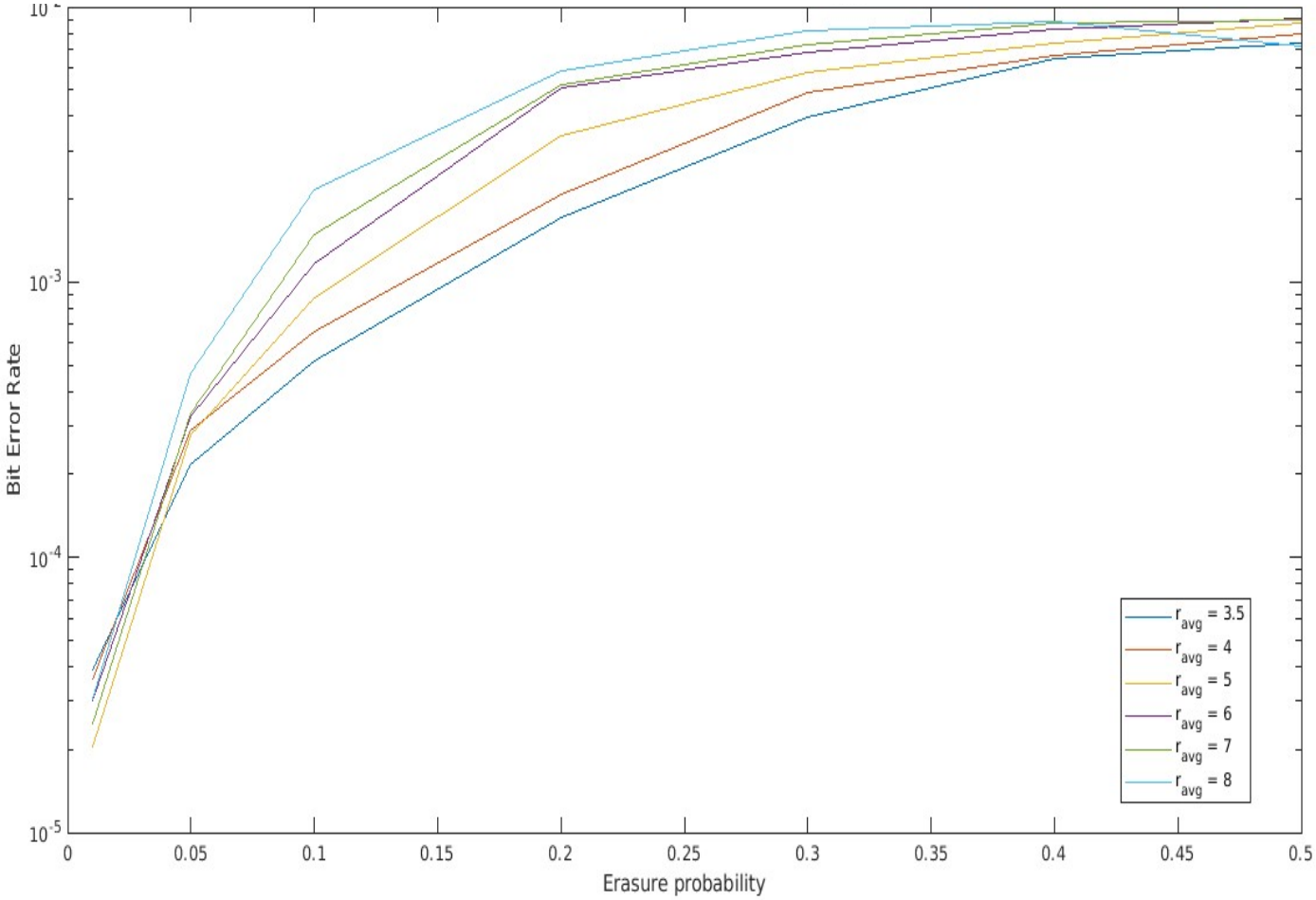

Figure 51 VER vs Erasure Rate

As expected, higher erasure probability results to higher BER. r_{avg} seems to play an important role too, as for higher values smaller BER is achieved.

Each LDPC code is characterized by 2 polynomials. However, each pair of polynomials corresponds to a family of LDPC codes rather than to a single specific LDPC code. This is because the polynomials describe the overall degree distributions of the nodes in the bipartite graph, but they do not specify the exact connections between the variable nodes and check nodes.

So, for each pair of polynomials, 200 different LDPC codes (instances) are generated in a random manner. Each instance is simulated on 1000 messages on the binary erasure channel.

In the following graphs (swarm charts), each dot represents a different instance of the same LDPC code.

Now, a comparison will be made between irregular and regular LDPC code performance. Below is the visualization that is used for the comparison:

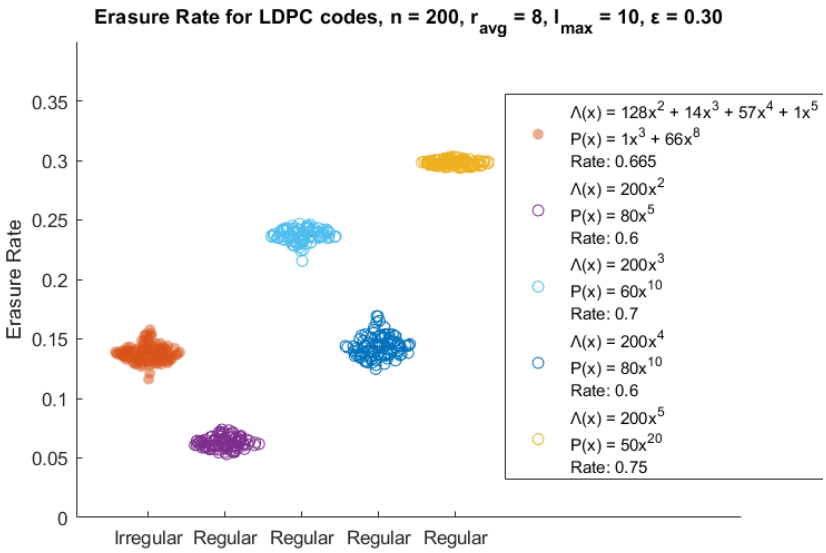

Figure 52 Erasure Rate for LDPC codes, n=200, r_{avg}=8, l_{max}=10

This is an example from the simulations that were performed, comparing 5 different LDPC codes (1 irregular and 4 regular ones) of similar rate. Each dot on the graph represents the performance of an instance (matrix $\mathbf{H}$) of an LDPC code simulated on multiple (1000 in this case) messages on the binary erasure channel. There are 200 instances (dots) of each LDPC code.
From this example, it can be observed that the performance of different instances of an LDPC code is concentrated around a mean (although the best-performing instance is of particular interest).
It is also observed that the best performing LDPC code in this case is a regular one, which can drop the erasure rate down from 0.3 to <0.1.

Now results for different parameter values will be presented:

Firstly, the effect of different values of ε (erasure probability) will be examined, for $\mathbf{r_{avg}} = \mathbf{4}$:

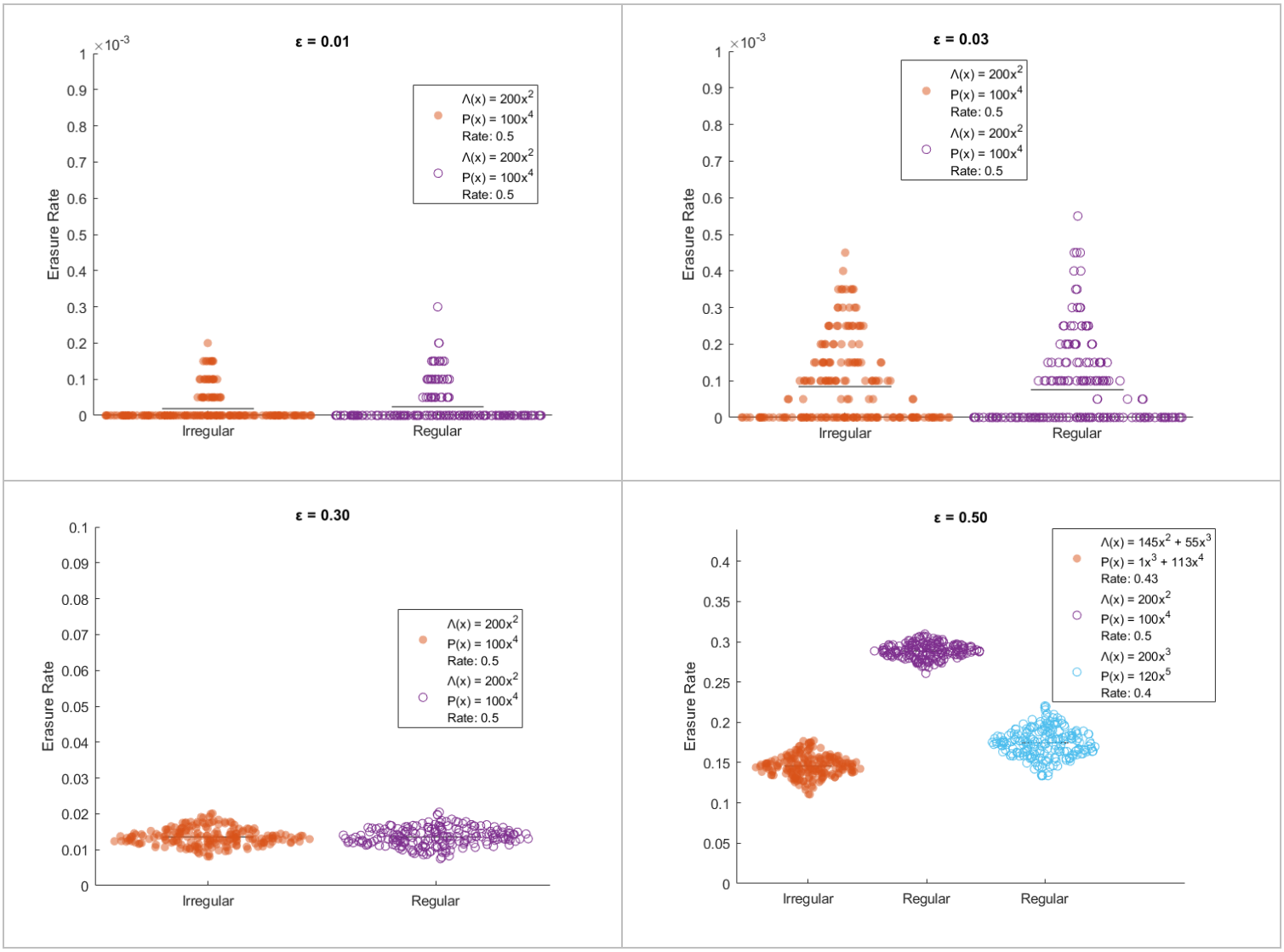

Figure 53 Erasure Rate for LDPC codes, n=200, r_{avg}=4, l_{max}=10

It is observed that for lower channel erasure rates, the "irregular" LDPC code that is generated is, in fact, a regular LDPC code. For a high erasure rate ($\boldsymbol{\varepsilon = 0.50}$), an irregular LDPC code is generated, which performs better than the regular ones.

Next, the effect of different values of ε (erasure probability) will be examined, for $\mathbf{r_{avg}} = \mathbf{8}$:

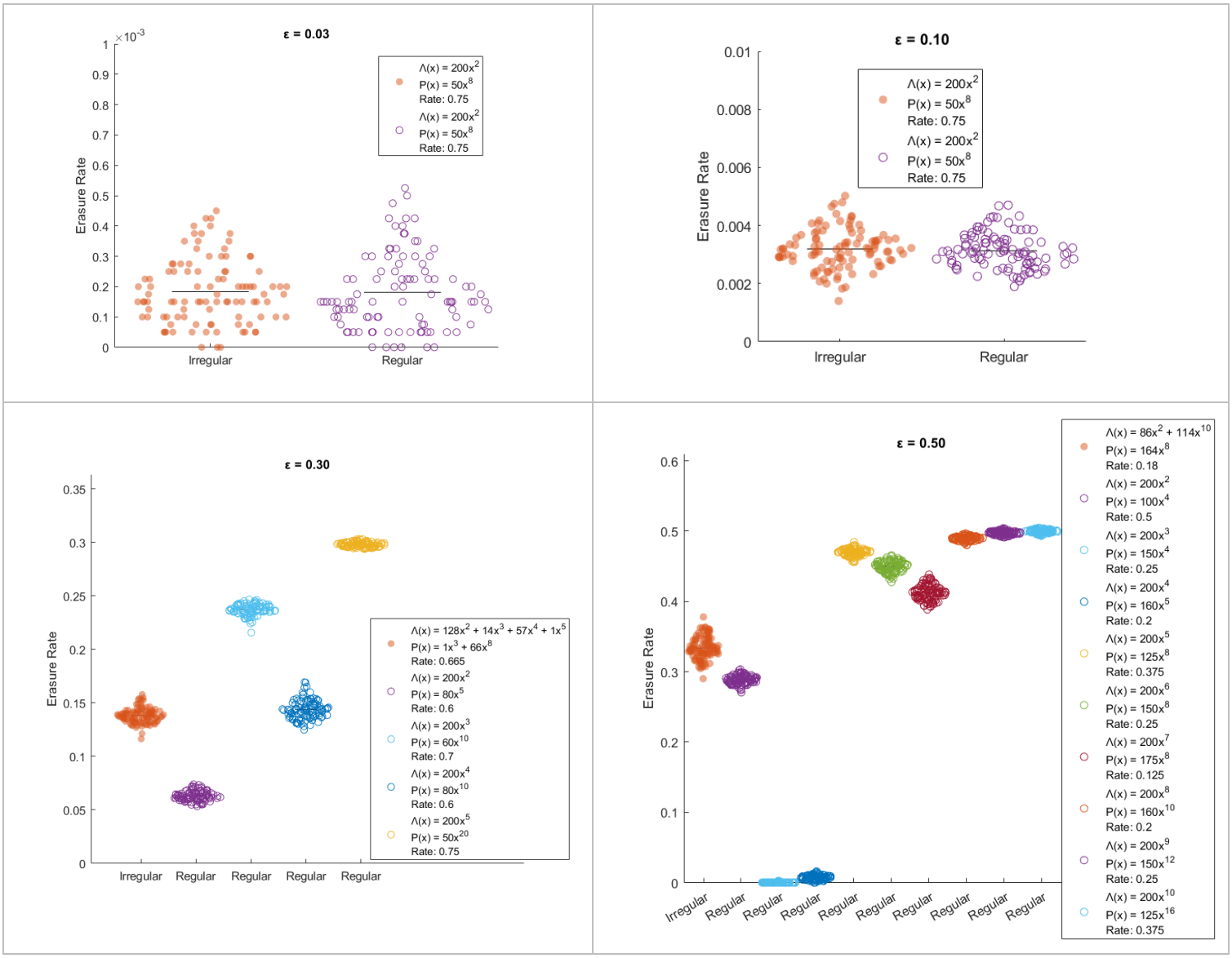

Figure 54 Erasure Rate for LDPC codes, n=200, ravg=8, lmax=10

Here it is observed that some regular LDPC codes may perform much better than others, and better than the irregular ones that were generated.

Finally, lets examine the effect of different $\mathbf{r_{avg}}$ for $\boldsymbol{\varepsilon = 0.4}$

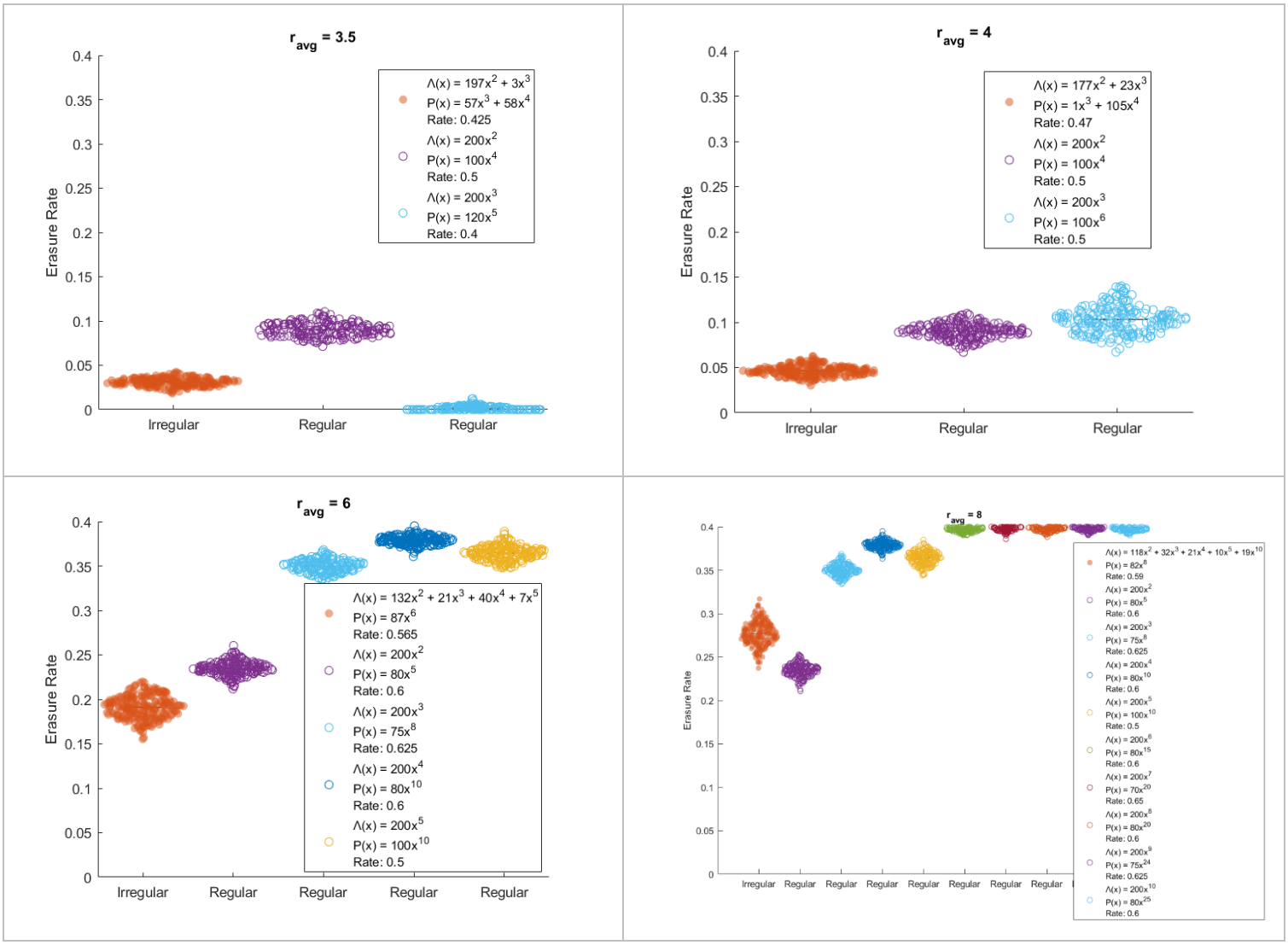

Figure 55 Erasure Rate for LDPC codes, n=200, ε=0.4, lmax=10

Again, it is observed that there are certain regular LDPC codes may perform better than the irregular LDPC code.

Similar simulations were performed for larger codewords length, specifically for **n = 1500** .
We tested 3 different scenarios, for $\boldsymbol{\varepsilon = 0.1}$, $\boldsymbol{\varepsilon = 0.3}$ and $\boldsymbol{\varepsilon = 0.4}$.

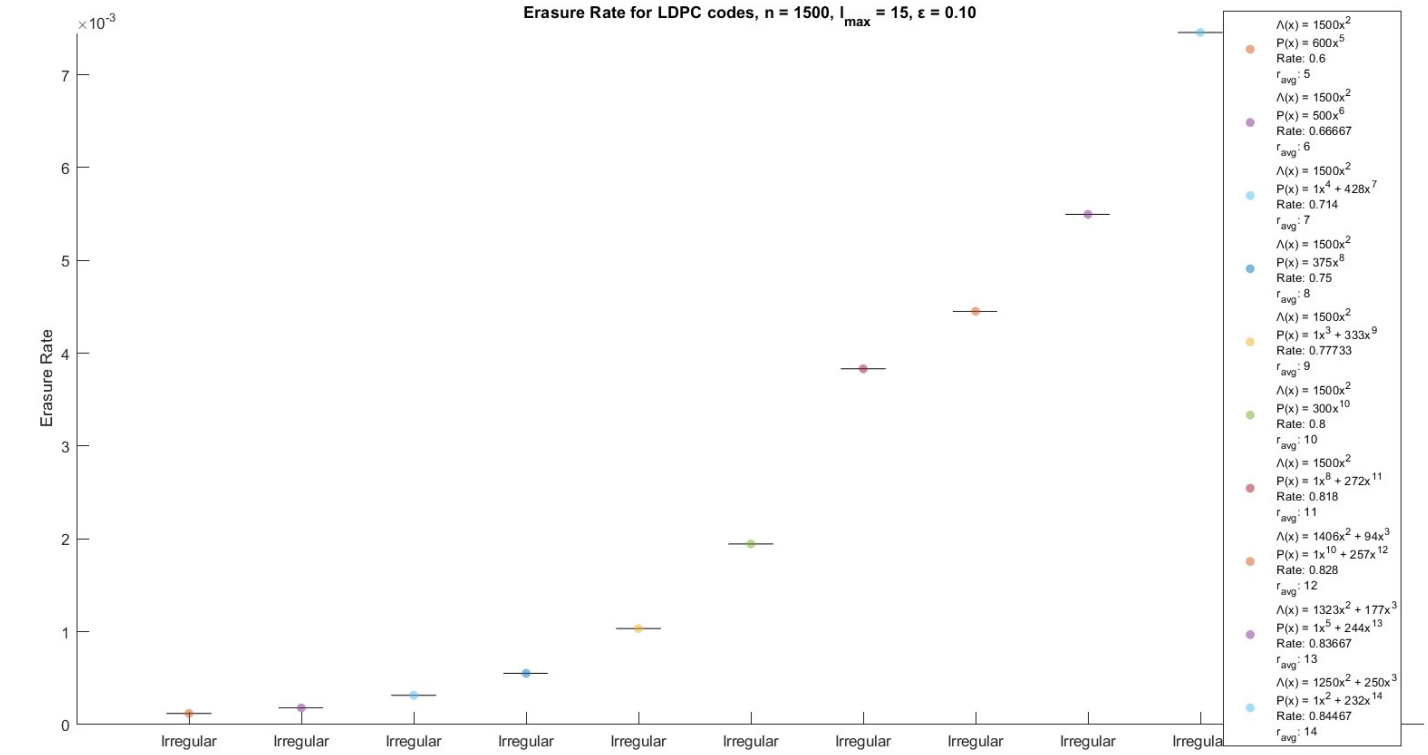

Figure 576 Erasure Rate for LDPC codes, n=1500, ravg=15, epsilon = 0.1

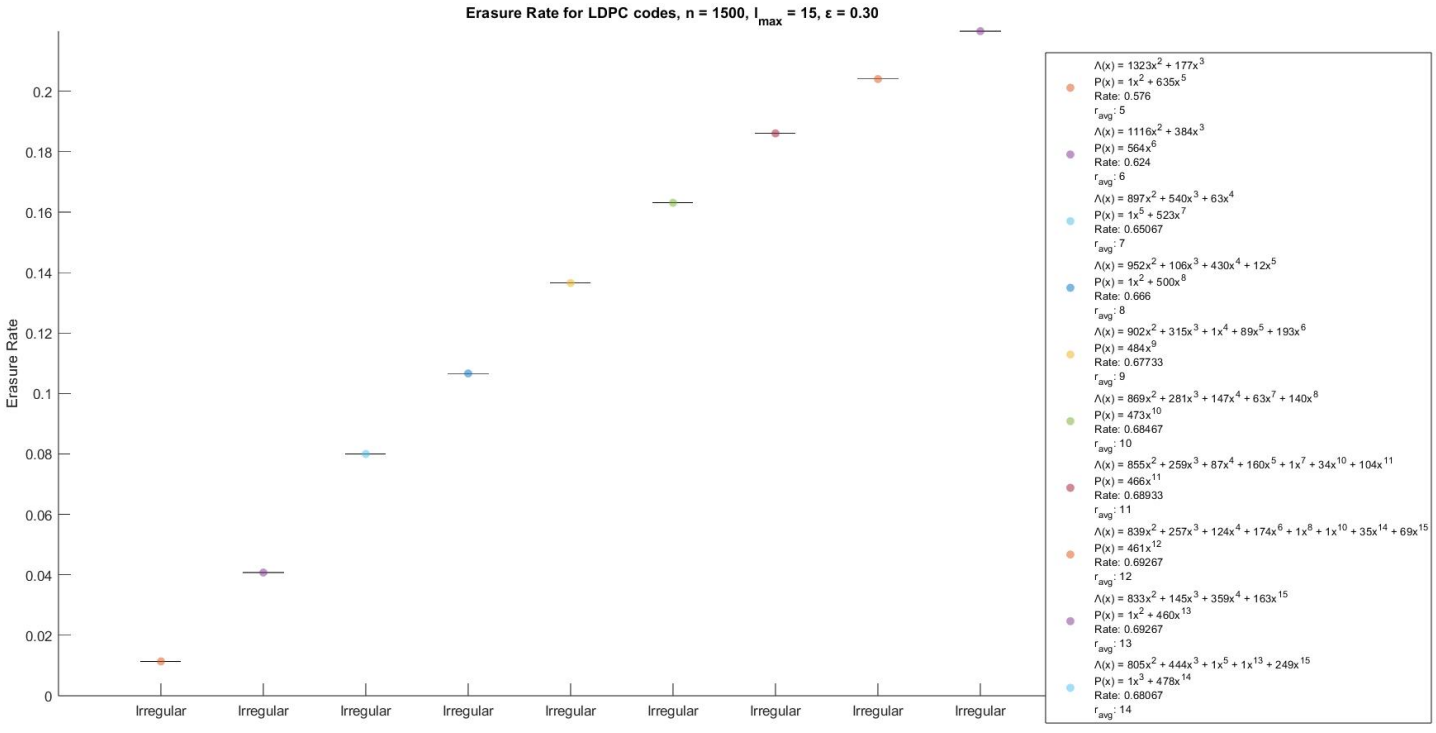

Figure 567 Erasure Rate for LDPC codes, n=1500, ravg=15, epsilon = 0.3

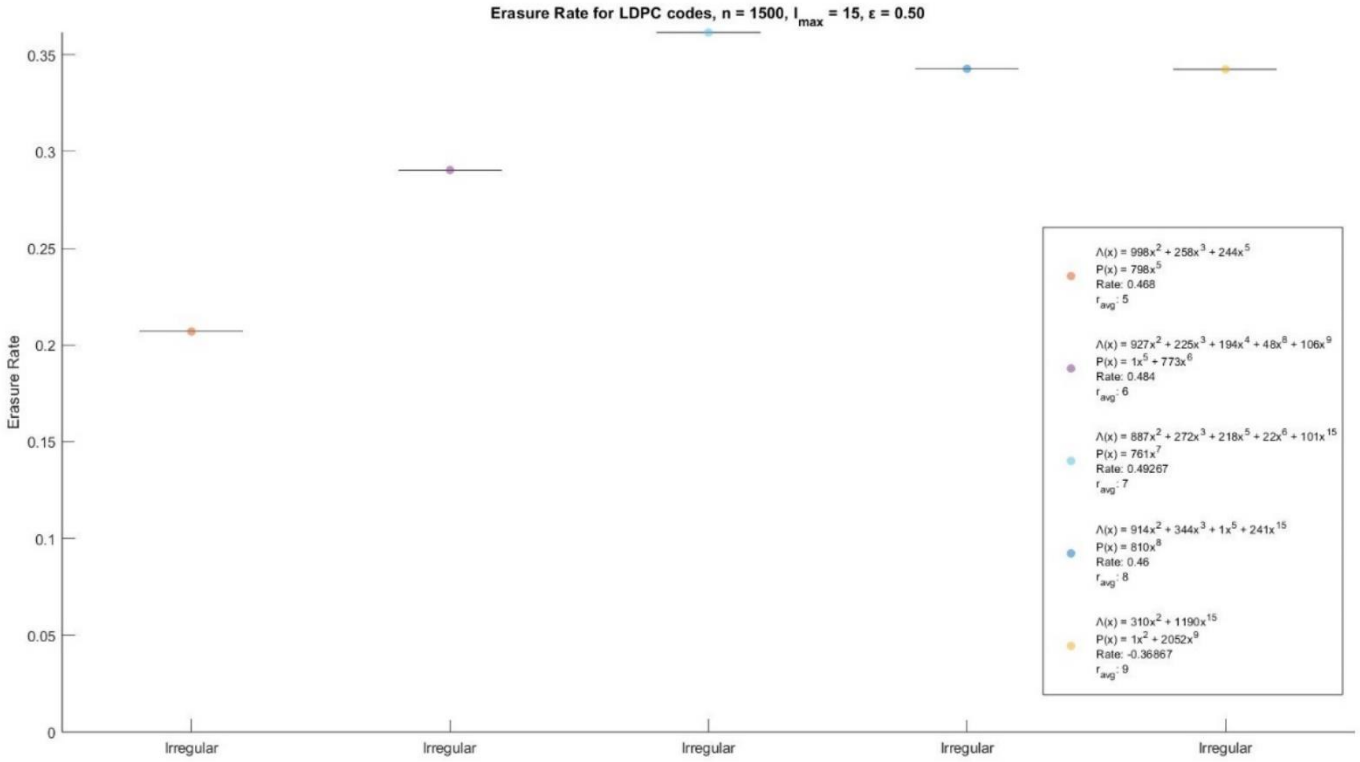

Figure 58 Erasure Rate for LDPC codes, n=1500, ravg=15, epsilon = 0.5

In all cases $r_{avg,max}$ was 15. If there is no irregular LDPC for an r_{avg}value, the corresponding optimization problem has no solution.

A final test we conducted was to find the optimal LDPC code for a specific channel erasure rate and compare different LDPC codes for all possible channel erasures rates. Specifically, we found the optimal LDPC codes for $\boldsymbol{\varepsilon = 0.2}$, $\boldsymbol{\varepsilon = 0.3}$ and $\boldsymbol{\varepsilon = 0.4}$ and then we compared them.

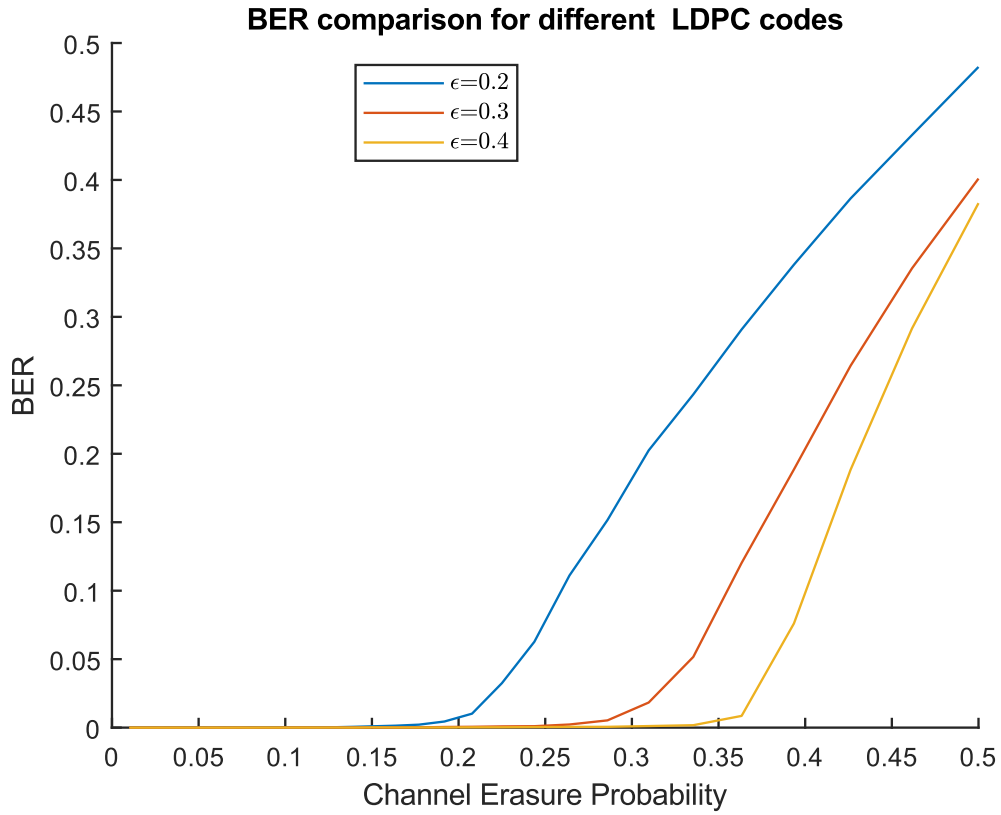

Figure 59 Comparison of different LDPC codes

As expected, each LDPC performs well up to the erasure rate for which it had the best performance. After this point, BER increases exponentially.

Conclusions

Regular LDPC codes are limited in their variety, as the values of the parameters (code size, code rate etc.) must adhere to more strict numerical constraints compared to irregular LDPC codes. For small codeword lengths, irregular codes end up being regular. For bigger lengths, irregular codes were produced. These codes had significantly improved performance.

References

[1] J. G. Proakis and M. Salehi, *Communication Systems Engineering*, Second. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

[2] Todd K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*, USA, Wiley-Interscience, 2005.

[3] D. J. C. MacKay, *Information theory, inference, and learning algorithms*. Cambridge: Cambridge University Press, 2003.

[4] T. J. Richardson and R. L. Urbanke, *Modern Coding Theory.* Cambridge University Press, 2008, p. I–XVI, 1-572.

[5] V. Guruswami and A. Rudra and M. Sudan, *Essential Coding Theory*, USA, 2019

[6] https://en.wikipedia.org/wiki/Singleton_bound