Error Control Coding Simulation Project

Professor: Brian Frost

Fall 2020

Introduction

In class, we developed a framework for error control coding through vector and matrix arithmetic over Galois field 2 ($\mathbb{F}_2$). Theoretically, we explored linear block codes and error detection in the previous homework for very specific cases, and did most of our work by hand. In reality, these sorts of operations are being done on massive scales by low-level hardware rapidly each and every moment.

Somewhere in between your hand calculations and a specialized integrated circuit is a regular computer with MATLAB installed on it. In this project, you will use MATLAB to explore the properties of linear block codes, and most importantly, how they improve performance in noisy environments. To facilitate this exploration, you will first develop a few tools for manipulating $\mathbb{F}_2$ objects in MATLAB.

MATLAB Problems

1. We need to be able to perform matrix and vector multiplications over $\mathbb{F}_2$, and preferably don't want to download any MATLAB toolboxes and learn what on Earth they do. Instead, write a function which takes an $M \times N$ matrix and an $N \times K$ matrix assumed to contain only the matlab integer values 0 and 1, and returns the product over $\mathbb{F}_2$ as an $M \times K$ matrix containing only MATLAB integer values 0 and 1. Test this on a few matrices to show it works as desired. **Hint**: Do not use a for loop to do this – You can actually start by just taking the regular matrix product.

2. Similarly, write a function which performs matrix addition over $\mathbb{F}_2$. This should use a similar method – no loops.

3. Write a binary symmetric channel function which takes a string of bits and a bit error probability p, and outputs a corrupted string of bits. No loops.

   For the rest of the assignment, I am going to ask you to consider two systematic codes with different values of n and $k = 4$. They are described by the following generator matrices:

$$G_1 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}, \; G_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

4. It will be a small hassle, but enter these matrices into MATLAB, and make a matrix containing all 4-bit data words as rows. You can use functions to do this but it will probably be quicker to

do it by hand. Come up with the two codebooks, each of which will have 16 rows ("elements"), either 8 or 12 bits long. Using the relationship between minimum Hamming weight and distance, what is the maximum number of errors that can be corrected with either code?

5. Using MATLAB, create the parity check matrices for either code. Ensure that $GH = 0$ in either case.

6. For these two codes, we know the maximum number of errors we can correct for. Thus, we know all possible error sequences we can correct for as well. In class, we learned to make a massive array detailing every possible error – here, we will constrain ourselves to the case in which the error is correctable. For each code, create an array which contains all of the correctable error words (for example, if $n = 15$ and we can correct for 3 errors, every possible binary sequence with 12 0s and 3 1s should appear in the array). Create the truncated syndrome array corresponding to the correctable errors. How much smaller is this than the real syndrome array?

7. Write a function that takes the error word set, the truncated syndrome array and a received word as inputs. The function should return the corrected word in the case that the error is correctable, or return a string of NaNs if the error is not correctable (easily determined through the syndrome). Test your error correction scheme before moving to the next step.

8. Randomly generate 10^5 words, and encode them using both schemes. Simulate the transmission of the uncoded bits and the two sets of coded words through binary symmetric channels with error probabilities ranging from 0.001 to 0.1. Find the bit error rate of the uncoded signal and the bit error rates of the two codes, with the interpretation that an "uncorrectable error" should count as a full word of errors. Plot these against p.

Report Guidelines

I would like you to present your findings in a short report, containing at least four sections:

1. **Introduction**: What did you set out to accomplish? What are the coding methods being explored and how are they defined?

2. **Methods**: Describe how you implemented Galois field 2 arithmetic. Describe your binary symmetric channel function. Describe how you determined the maximum number of correctable errors. What does it mean that larger errors are "not correctable"? Are they detectable? How did you generate the truncated syndrome array, and what does it mean? How do you correct errors?

3. **Results**: There's only really one plat you need to show here, describe and show it.

4. **Discussion**: How do the error correction schemes compare in bit rate and error rate? What is lost through the method of creating a truncated syndrome array? In practice, why might we only correct smaller errors in this way? What would we do, practically, about the tossed information?

I would suggest using a LaTeX editor if you can to write this report – it is useful to start learning this skill early – but I will not require it. There is no upper or lower page limit on the report. You must send me the MATLAB files used to generate the results.

Grading Breakdown

1. **Report Content (70%)**: The majority of your grade will be from the *content* of your report, including your responses to the questions and your ability to produce correct graphs (whether or not they are pretty).
2. **Report Style (15%)**: It is important that you present your results with labeled graphs and grammatically correct sentences. Your report should be organized well enough that someone (for example, yourself in the future) might be able to look at it as a reference on the topic it is covering.
3. **MATLAB Style (25%)**: Just as in your homeworks – don't write bad MATLAB code!