Information Theory
Project 3

Salvador Castagnino

Introduction

The following project consists of the task of working and studying Reed-Muller codes. In particular we will take a look at the generation matrix used to encode messages, the space spanned by it, its dual space and the way in which the dual can be used for error correction. Calculations are done using python code which can be found in the appendix.

Generator

We start by calculating the generator matrix G(1, 3) using the definition presented in the assignment, we get

$$\begin{matrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{matrix}$$

By multiplying this matrix by every single 4 component binary vector we generate the space of codewords. The obtained codewords are the following row vectors,

[0. 0. 0. 0. 0. 0. 0. 0.], [0. 0. 0. 0. 1. 1. 1. 1.]
[0. 0. 1. 1. 0. 0. 1. 1.], [0. 0. 1. 1. 1. 1. 0. 0.]
[0. 1. 0. 1. 0. 1. 0. 1.], [0. 1. 0. 1. 1. 0. 1. 0.]
[0. 1. 1. 0. 0. 1. 1. 0.], [0. 1. 1. 0. 1. 0. 0. 1.]
[1. 0. 1. 0. 1. 0. 1. 0.], [1. 0. 1. 0. 0. 1. 0. 1.]
[1. 0. 0. 1. 1. 0. 0. 1.], [1. 0. 0. 1. 0. 1. 1. 0.]
[1. 1. 1. 1. 1. 1. 1. 1.], [1. 1. 1. 1. 0. 0. 0. 0.]
[1. 1. 0. 0. 1. 1. 0. 0.], [1. 1. 0. 0. 0. 0. 1. 1.]

With the codewords it is easy to figure out the minimum distance between vectors of the space, we just need to count the minimum number of ones in the vectors. In this case we have d_min = 4. By using the formulas presented in Lecture 9 it is easy to derive which errors can

be detected and which corrected. Only those errors with Hamming weight smaller or equal to 3 could be detected and those errors with Hamming weight equal to 1 could be corrected.

Dual and Error Correction

It is easy to validate that G(1, 3) is its own dual, it suffices to multiply the matrix by its transpose and check that the resulting matrix is zero. To get the syndromes of every one bit error (observe that every compound error can be generated by adding one bit errors) we proceed by multiplying the 8x8 identity matrix by the transpose of the dual matrix which in this case is G(1, 3) itself. The syndromes are the following row vectors,

[1. 0. 0. 0.], [0. 1. 0. 0.]
[1. 0. 1. 0.], [0. 1. 1. 0.]
[1. 0. 0. 1.], [0. 1. 0. 1.]
[1. 0. 1. 1.], [0. 1. 1. 1.]

Finally we take the second row of G(1, 3) as our codeword and the first row of the 8x8 identity matrix as our error (observe that the chosen error can have at most one nonnegative value in order to be corrected). By adding the error to our codeword and multiplying the result by the dual matrix we get the [1. 0. 0. 0.] syndrome, which as expected is the first one on the list of syndromes (as the error was the first row of the identity matrix). We then take the associated error and add it to the value we evaluated the dual on, this will correct the error and return our original codeword. This process is carried out step by step in the python code.

Appendix

```
import numpy as np
import itertools

#Function used to calculate values of the type G(1, m) where m is any
value equal or greater than 1
#It can be proven that G(1, m) has dimensions (m+1)x(2^m), we use this
to calculate the dimension of the 0 block
def G(r, m):
    if r > 1 or m < r:
        raise ValueError("r cannot be greater than 1 neither m can be
smaller than r")

    if r == 0:
        return np.ones((2 ** m,))

    if r == m:
        return np.identity(2 ** m)
```

```
   top_matrix = G(r, m-1)
   return np.block([[top_matrix, top_matrix], [np.zeros((1, 2**(m-1))),
G(r-1, m-1)]])


m = 3
generator = G(1, m)

bin = {0,1}
bin_power = itertools.product(bin, repeat=2**(m-1))
codewords = [np.dot(np.array(u), generator)%2 for u in bin_power]
for codeword in codewords:
   print(codeword)

errors = np.identity(2 ** m)
syndromes = np.dot(errors, generator.T)%2
codeword = generator[1]

print(generator)
print(np.dot(generator, generator.T) % 2)
print(syndromes)
#We get the syndrome which we can use to find the error and add it to
the codeword
print(np.dot((codeword + errors[0]), generator.T)%2)
#Add the error to intermediate value
print(((codeword + errors[0]) + errors[0])%2)
```