EE5160: Error Control Coding (Project-1) BCH-encoder-decoder

Chinni Chaitanya EE13B072
Prafulla Chandra EE16D402

March 30, 2017

1 Introduction

The problem statement is to design a narrow-sense binary BCH (Bose–Chaudhuri–Hocquenghem) encoder and decoder with design distance, $\delta = 15$ and code length, $n = 127$.
The Galois field $\mathbb{F}_{128}$ is constructed using the primitive polynomial $x^7 + x^3 + 1$.

Note: The implementation is done in MATLAB and all the functions described below are to be considered MATLAB functions unless otherwise stated.

2 Construction of code

2.1 Generator polynomial

The generator polynomial, $g(x)$ is constructed by taking the LCM of the minimal polynomials of elements, $\{\alpha, \alpha^2, \ldots \alpha^{14}\}$ which is effectively the product of the minimal polynomials of the elements $\{\alpha, \alpha^3, \alpha^5, \alpha^7, \alpha^9, \alpha^{11}, \alpha^{13}\}$.

The Galois field is generated using the function `gftuple()` and the minimal polynomials are generated using the function `gfminpol()`. The polynomial multiplication in $\mathbb{F}_{128}$ is implemented using the function `gfconv()`.

2.2 Calculation of k

The degree of the generator polynomial determined above is 49. Therefore,

$$\begin{aligned} & n - k = 49 \\ \implies & k = 127 - 49 \\ \implies & k = 78 \end{aligned}$$

3 Encoding

The encoding is done in *systematic form.* The messages are read from `msg.txt` file and each message is encoded and saved to file `codeword.txt`. Systematic encoding can be described in the following steps,

(i) Multiply the message polynomial, $m(x)$ with x^{n-k} i.e. multiply with x^{49} in our case

(ii) Calculate the remainder when $x^{n-k}m(x)$ is divided with the generator polynomial, $g(x)$

(iii) Subtract the remainder from the polynomial $x^{n-k}m(x)$

The multiplication with x^{n-k} is essentially shifting the message bit array to right[1] by $(n-k)$ zeros. This is implemented using `padarray()` function. For calculating remainder in Galois field $\mathbb{F}_{128}$, we have used `gfdeconv()` function. The `gfsub()` function is used for subtracting the remainder from $x^{n-k}m(x)$.

4 Decoding

The standard error correcting procedure for BCH codes is of three steps:

(i) Compute the syndrome $S = (S_1, S_2, ..., S_{2t})$ from the received polynomial $r(x)$.

(ii) Determine the error location polynomial $\sigma(x)$ from the syndrome components, $S_1, S_2, ..., S_{2t}$.

(iii) Determine the locations of error by finding the roots of $\sigma(x)$ and correct the errors in $r(x)$.

The input to the decoder i.e. the received vector is read from the file `rx.txt`. The first step in the decoding procedure is achieved by using the function `polyval()` with the received polynomial $r(x)$ and vector containing $\left\{\alpha, \alpha^2, \ldots \alpha^{14}\right\}$ as input arguments. The Galois field polynomial of $r(x)$ is constructed[2] using the function `gf()`.

The error location polynomial $\sigma(x)$ is found using the simplified Berlekamp-Massey iterative algorithm for binary BCH codes. If $\sigma(x)$ has a degree greater than $t = \frac{\delta-1}{2}$, then the received codeword cannot be corrected. Else, proceed to step (iii).

In the step (iii), each bit of the received polynomial/codeword is corrected by checking whether it's location has an error or not, i.e. to correct the i^{th} bit in the received vector, we check whether α^{n-i} is a root of the error locator polynomial, $\sigma(x)$ or not(using the function `polyval()`). If it is, we correct the bit by adding a 1 to it, otherwise, we leave it unaltered. This is done for each bit of received polynomial/codeword from the highest order position to lowest order position. The resulting polynomial corresponds to the correct codeword or the transmitted codeword.

If the channel is a binary symmetric erasure channel, the received codeword may contain both errors and erasures. In this case, the decoding is accomplished in two steps:

(i) All the erased positions are replaced with 0 and the resulting vector is decoded using the above standard BCH decoding algorithm.

[1] We take the last bit of the message bit array to be m_k and the first bit to be m_0

[2] Again, the last bit of the received bit array is considered r_{n-1} and first bit is considered r_0

(ii) All the erased positions are replaced with 1 and the resulting vector is decoded in the same way. $\hat{C}$

5 Results

We have tested our encoder and decoder algorithm for a few test message bits. The number of erasures and errors are also diversified to account for all possible cases listed below.

- `M(x)` denotes the message vector
- `C(x)` denotes the codeword
- `R(x)` denotes the received vector
- `C'(x)` denotes the estimated codeword
- `M'(x)` denotes the estimated message vector
- The errors and erasures are marked red in both codeword, received vector and the estimated codeword
- δ is the design distance ($= 15$)

Example-1 (erasures + 2×errors $< \delta$)

```
M(x):  000000000000000111001000010001101000110101101100001100
       010111000000100110111010
C(x):  000110001011011010110010010110101100100110000101100000
       000000000011100100001000110100011010110110000110001011
       1000000100110111010
R(x):  000120021011011010110010010110101100100110000101100000
       200000000001100100001000110100011010110110000110001011
       1001000100110111010
C'(x): 000110001011011010110010010110101100100110000101100000
       000000000011100100001000110100011010110110000110001011
       1000000100110111010
M'(x): 000000000000000111001000010001101000110101101100001100
       010111000000100110111010
```

Example-2 (erasures + 2×errors $< \delta$)

```
M(x):  000000000000000001111000100111000010111111101111111110
       011101001111010100100100
C(x):  010010101100101101011110010110001010011110110101100000
       000000000000111100010011100001011111110111111111001110
       1001111010100100100
R(x):  010010101100101101011110010110001010011110110000000000
       000000000000000000010011100001011111110111111111001110
       1001111010100100100
C'(x): 010010101100101101011110010110001010011110110101100000
       000000000000111100010011100001011111110111111111001110
       1001111010100100100
M'(x): 000000000000000001111000100111000010111111101111111110
       011101001111010100100100
```

Example-3 (erasures + 2×errors > δ)

```
M(x): 000000000000000111111011000110010110010011010100001001
      010111101000010011010100
C(x): 000101011011101110001011101000110100111111001101100000
      000000000011111101100011001011001001101010000100101011
      1101000010011010100
R(x): 000101011011101110001011101000110100111111001101100000
      011110000121200100101011001011001001101010000100101011
      1101000010011010100

Output: Cannot decode since the number of errors > t (=7)
```

Example-4 (erasures + 2×errors > δ)

```
M(x): 000000000000000010010011000111101001000111010001111100
      010010100110110100100011
C(x): 100001001101111100111000111010101111110011110111100000
      000000000001001001100011110100100011101000111110001001
      0100110110100100011
R(x): 200001001101111100111000111010101111110011110121100100
      110021000002001001100010111100100011101000111110001001
      0100110110100100011
C'(x):100001001101111100111000111010101111110011110111100000
      000000000001001001100011110100100011101000111110001001
      0100110110100100011
M'(x):000000000000000010010011000111101001000111010001111100
      010010100110110100100011
```

This shows that the BCH code can decode beyond minimum distance but not in all cases. We have shown two cases where the number of errors and erasures don't satisfy minimum distance condition and in one of them it fails (Example-3) and in one of them it decodes correctly(Example-4).

The following table gives the intermediate table values for the Example-4 case when the erasures are replaced by 1 (leads to less errors after replace when compared to replacing with 0).

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$2\mu - l_\mu$
$-\frac{1}{2}$	0	0	0	1
0	0	28	0	0
1	0 28	66	1	1
2	0 28 38	69	2	2
3	0 28 121 31	108	3	3
4	0 28 44 80 77	68	4	4
5	0 28 120 71 74 118	22	5	5
6	0 28 11 29 41 67 31	84	6	6
7	0 28 105 4 47 108 77 53	-	7	7

References

[1] "Error Control Coding: Fundamentals and Applications" by Shu Lin, Daniel J. Costello Jr.