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

Chinni Chaitanya EE13B072

Prafulla Chandra EE16D402

March 31, 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,

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

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 $x^{n-k}m(x)$.

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

4 Decoding

The standard (systematic) 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 $r(x)$.

(iv) Determine the estimated message as the last[2] k-bits of the estimated codewords (since systematic decoding)

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 $\{\alpha, \alpha^2, \dots \alpha^{14}\}$ as input arguments. The Galois field polynomial of $r(x)$ is constructed[3] using the function `gf()`.

The error locator polynomial $\sigma(x)$ is found using the simplified Berlekamp-Massey iterative algorithm for binary BCH codes where we iterate for t-steps. The BM-algorithm can fail in many cases listed below,

- Some roots of the error locator polynomial might not be in the current Galois field, $\mathbb{F}_{128}$ and it occurs when the number of errors exceed t, or $2\times$ *number of errors* $> \delta$
- The error locator polynomial might have more than t roots but within the Galois field, $\mathbb{F}_{128}$
- The BM-algorithm might return an error locator polynomial which has repeated roots which results in decoder failure

In all the above cases, we report *Decoder failure: <relevant error message>*. 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 (by using the function `polyval()`). If it is, we correct the bit by adding 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. We then compute the syndrome of the resulting polynomial and check if it is all 0. If it is, we consider it to be an estimate of the transmitted codeword and we call it the **estimated codeword**. Else, we report the decoder failure which is one of the three categories listed above.

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.

(ii) All the erased positions are replaced with 1 and the resulting vector is decoded in the same way.

We will then have two estimated codewords and we need to choose one of them. This is done in the following steps,

(i) If both the estimated codewords have non-zero syndrome, we report decoder failure

(ii) If one of them have non-zero syndrome, we discard it and proceed to further tests listed below for the other estimated codeword

  - The number of roots of the error locator polynomial must be $< t$ and all of them must be distinct and must belong to the Galois field, $\mathbb{F}_{128}$ for it to be the estimated codeword
  - In any other case, we report decoder failure with appropriate error message

[2] Since the last bit of estimated codeword will be $\hat{c}_{n-1}$ and the first bit will be $\hat{c}_0$

[3] Again, we consider the last bit of the received bit array as r_{n-1} and first bit as r_0

5 Results

Below are the results of our encoder-decoder algorithm implemented for a few test cases. The errors and erasures are marked red in both codeword, received vector and the estimated codeword for ease of checking.

- `M(x)` and `M'(x)` denote the message vector and the estimated message vector respectively
- `C(x)` and `C'(x)` denote the codeword and the estimated codeword respectively
- `R(x)` denotes the received vector

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

```
M(x):  0000000000000001110010000100011010001101011011000011000101110000001001
       10111010
C(x):  0001100010110110101100100101101011001001100001011000000000000000111001
       000010001101000110101101100001100010111000000100110111010
R(x):  0001200210110110101100100101101011001001100001011000002000000000011001
       000010001101000110101101100001100010111001000100110111010
C'(x): 0001100010110110101100100101101011001001100001011000000000000000111001
       000010001101000110101101100001100010111000000100110111010
M'(x): 0000000000000001110010000100011010001101011011000011000101110000001001
       10111010
```

Example-2 (errors $= t$, erasures $= 0$)

```
M(x):  0000000000000000011110001001110000101111111011111111100111010011110101
       00100100
C(x):  0100101011001011010111100101100010100111101101011000000000000000001111
       000100111000010111111101111111110011101001111010100100100
R(x):  0100101011001011010111100101100010100111101100000000000000000000000000
       000100111000010111111101111111110011101001111010100100100
C'(x): 0100101011001011010111100101100010100111101101011000000000000000001111
       000100111000010111111101111111110011101001111010100100100
M'(x): 0000000000000000011110001001110000101111111011111111100111010011110101
       00100100
```

Example-3 (erasures + 2×errors $> \delta$)

```
M(x):  0000000000000001111110110001100101100100110101000010010101111010000100
       11010100
C(x):  0001010110111011100010111010001101001111110011011000000000000000111111
       011000110010110010011010100001001010111101000010011010100
R(x):  0001010110111011100010111010001101001111110011011000000111100001212001
       001010110010110010011010100001001010111101000010011010100

Output: 'Decoder failure: Some roots of ELP do not belong to GF[128] because
   ↪ number of errors > t (=7)'
```

Example-4 (erasures + 2×errors $> \delta$)

```
M(x):  0000000000000000100100110001111010010001110100011111000100101001101101
       00100011
C(x):  1000010011011111001110001110101011111100111101111000000000000000010010
       011000111101001000111010001111100010010100110110100100011
R(x):  2000010011011111001110001110101011111100111101211001001100210000020010
       011000101111001000111010001111100010010100110110100100011
C'(x): 1000010011011111001110001110101011111100111101111000000000000000010010
       011000111101001000111010001111100010010100110110100100011
M'(x): 0000000000000000100100110001111010010001110100011111000100101001101101
       00100011
```

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

6 Instructions to run the code

6.1 Encoding

(i) Add the message bit vectors to the file `msg.txt` with one message vector in each line

(ii) Execute `BCH_encoder.m` and it will output the encoded message vectors to the file `codeword.txt` in the same order as the messages in `msg.txt`

6.2 Decoding

(i) Add the received codewords to the file `rx.txt` with erasures represented with **2**

(ii) Execute `BCH_decoder.m` and it will output the decoded/estimated codewords to the file `decoderOut_coderowd.txt` and the decoded/estimated message bit vectors to the file `decoderOut_msg.txt`

(iii) It also prints the intermediate table values of the Berlekamp-Massey algorithm, for all the cases to `logfile.log`

(iv) Note that if the decoder fails, the corresponding error message will be printed instead of the estimated codeword and the estimated message vector

References

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