

---

# Locator decoding for BCH codes

---

JUAN SILVERIO DOMINGUEZ Y SAINZA  
SUPERVISOR: DR. JAAP TOP

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                                      | <b>2</b>  |
| 1.1      | Introduction . . . . .                                   | 2         |
| 1.2      | The binary $[7, 4]$ -Hamming Code . . . . .              | 3         |
| 1.3      | Notations and definitions . . . . .                      | 4         |
| <b>2</b> | <b>Syndromes and Cyclic Codes</b>                        | <b>6</b>  |
| 2.1      | Syndromes . . . . .                                      | 6         |
| 2.2      | Dual code . . . . .                                      | 7         |
| 2.3      | Example: The $(7, 4)$ -Hamming Code . . . . .            | 8         |
| 2.4      | Cyclic codes . . . . .                                   | 10        |
| 2.4.1    | BCH and Reed-Solomon codes . . . . .                     | 12        |
| 2.4.2    | Fourier transform . . . . .                              | 14        |
| <b>3</b> | <b>Majority Decoding</b>                                 | <b>16</b> |
| 3.1      | Reed-Muller Codes . . . . .                              | 16        |
| 3.1.1    | Reed algorithm . . . . .                                 | 19        |
| <b>4</b> | <b>Locator decoding</b>                                  | <b>22</b> |
| 4.1      | Locator polynomials and the Peterson algorithm . . . . . | 22        |
| 4.1.1    | Example of the Peterson algorithm . . . . .              | 25        |
| 4.1.2    | Linear complexity . . . . .                              | 27        |
| 4.2      | The algorithms . . . . .                                 | 29        |
| 4.2.1    | Sugiyama algorithm . . . . .                             | 31        |
| 4.2.2    | The Sugiyama algorithm in Maple . . . . .                | 33        |
| 4.2.3    | Berlekamp-Massey algorithm . . . . .                     | 35        |
| 4.2.4    | The Berlekamp-Massey algorithm in Maple . . . . .        | 38        |
| 4.3      | Forney . . . . .   | 40        |
| <b>5</b> | <b>Example</b>   | <b>42</b> |
| 5.1      | Example using the Sugiyama algorithm . . . . .           | 42        |
| 5.2      | Conclusion . . . . .                                     | 45        |

# Chapter 1

## Introduction

### 1.1 Introduction

Nowadays digital communication is present in every aspect of our lives: satellite data transmissions, network transmissions, computer file transfers, radio communications, cellular communications, etc. These transmissions transfer data information through a channel that is prone to error. An error-correcting code allows the receiver to identify the intended transmission. The main idea of error-correcting coding and decoding is to modify the data before transmission so that after the possibly errors crept into the message, the receiver can deduce what the intended data was without having to request a retransmission.

There are many different approaches to modify the data before transmission, each with various methods to recover the intended message from the message with errors.

The idea of error-correcting codes came with the development of the computer technology industry. In the late 1930s Bell Telephone Laboratories built one of the first mechanical relay computers. This computer is unlike anything currently in use. However, the mechanical relay computer while executing a program was prone to errors like today's computers. In 1947 Richard W. Hamming conducted calculations on the mechanical relay computer at Bell Telephone Laboratories and found himself constantly re-running his programs due to computer halts. In an interview Hamming says:

*"Two weekends in a row I came in and found that all my stuff had been dumped and nothing was done. And so I said to myself: "Damn it, if the machine can detect an error, why can't it locate the position of the error and correct it?"*

The relay computers operated with self-checking codes only. The machine would

detect the error then halt the current job and move to the next. The coding technique used is similar to a repetition code.

With this coding technique the relay computer halted two or three times a day. The need for a better code was indispensable once the computer became 1000 times faster, so the computer would stop two or three hundred times a day. Hamming knew that if a repetition code was to be used, the relay computer would grow in size as the computer became faster. So Hamming invented a single error-correcting code, the  $[7, 4]$ -Hamming Code. In the next section and further we will look at this code.

This Master's Thesis first gives an example of the construction of the binary  $[7, 4]$ -Hamming Code, then I shall treat two classes of decoders for block codes: *majority decoding* and *locator decoding*. The main goal is to study locator decoding, the most interesting and most important class of decoders. As examples of this, we study the *Sugiyama* and the *Berlekamp-Massey* algorithm.

## 1.2 The binary $[7, 4]$ -Hamming Code

Binary codes are concerned with channels where the standard unit of information is a bit, it has the value 0 or 1.

Suppose somebody wants to send four bits across a channel where the errors that can occur are changing 0's into 1's and 1's into 0's. To encode the 4-bit word with the  $[7, 4]$ -Hamming code, place the value of bit 1 into area 1 in the following diagram, bit 2 in area 2, and so on.

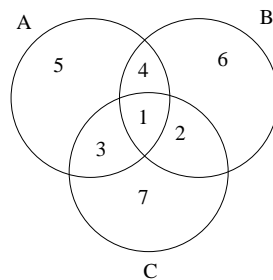


Figure 1.1: The Venn diagram for the  $[7, 4]$ -Hamming code.

Into the areas 5, 6, and 7 place a 0 or a 1 in such a way that each circle A, B and C contains an even number of 1's. Read the seven values back out of the diagram into a 7-bit word. The first 4 values are the word to be sent and the places 5, 6 and 7 are determined by the bits in their corresponding areas in the diagram.

This 7-bit word is called the *codeword* and that is the word that will be sent across

the channel. To decode this, the receiver should place bits 1 through 7 into their positions in the diagram, determine which circles have an odd number of 1's, turn the bit from a 0 to a 1 or from a 1 to a 0 of the area which influences exactly those circles, then read the values of areas 1 through 4. If only one error occurred, then this coding will always correct the received word.

**Example 1** *Suppose the sender wants to send 1011. Then only circle A has an odd number of 1's, so only position 5 should be 1. The codeword for 1011 is 1011100. Now suppose the receiver receives 1001100 instead of 1011100. There is one error made so we can correct it. The receiver places these bits back into their positions in the diagram and counts the 1's in each of the circles. Circle A and circle C have an odd number of 1's, so he assumes area 3 was in error and correctly deduces that the word sent was 1011100, thus our message is 1011. Notice that if an error occurs in bit 5, 6 or 7, the receiver will be able to identify the error, but it doesn't affect the message word.  $\square$*

## 1.3 Notations and definitions

More abstractly, consider every possible message the sender might wish to send as a string of elements of a field  $\mathbb{F}$ .

Then we can define a code of length  $n$  as follows

**Definition 1 (code of length  $n$ )** *A code of length  $n$  is a (nonempty) subset  $C$  of  $\mathbb{F}^n$ . An element  $c \in C$  is a codeword.*

In this paper I will only consider finite fields, denoted as  $\mathbb{F}_q$ , the field of  $q$  elements. A code  $C$  is then a subset of  $\mathbb{F}_q^n$ .

**Definition 2 (alphabet)** *The alphabet is the set of symbols from which codewords can be composed.*

**Definition 3 (distance)** *The distance (Hamming distance) of two words  $x$  and  $y$  of equal length is the number of positions in which  $x$  and  $y$  differ. This is denoted  $d(x, y)$ .*

*The minimum distance of a code is the smallest  $d(x, y) > 0$  that occurs.*

One can verify that distance applied to the set of words of length  $n$  forms a metric space, which means that:

- $d(x, y) \geq 0$ ,  $d(x, y) = 0 \Leftrightarrow x = y$ ;
- $d(x, y) = d(y, x)$ ;

- $d(x, z) \leq d(x, y) + d(y, z)$ .

**Definition 4 (weight)** The weight  $wt(a)$  of  $a$  is the number of nonzero components of the sequence  $a$ .

Note that  $d(x, y) = wt(x - y)$ .

**Theorem 1** A code with minimum distance  $d_{min}$  can correct  $t = \frac{d_{min}-1}{2}$  errors.

**PROOF:** We define a sphere of radius  $r$  about a vector  $\mathbf{c}_1$  denoted by  $B(\mathbf{c}_1, r)$  as follows:

$$B(\mathbf{c}_1, r) = \{\mathbf{c}_2 \in C \mid d(\mathbf{c}_1, \mathbf{c}_2) \leq r\}$$

We prove that spheres of radius  $t = \frac{d_{min}-1}{2}$  about codewords are disjoint.

Suppose not. Let  $\mathbf{c}_1$  and  $\mathbf{c}_2$  be distinct vectors in  $C$  and assume that  $B(\mathbf{c}_1, t) \cap B(\mathbf{c}_2, t)$  is nonempty. Suppose  $\mathbf{c}_3 \in B(\mathbf{c}_1, t) \cap B(\mathbf{c}_2, t)$ . Then  $d(\mathbf{c}_1, \mathbf{c}_2) \leq d(\mathbf{c}_1, \mathbf{c}_3) + d(\mathbf{c}_2, \mathbf{c}_3)$  by the triangle inequality and this is  $\leq 2t$  since the points are in the spheres. Now  $2t \leq d_{min} - 1$  so that  $d(\mathbf{c}_1, \mathbf{c}_2) \leq d_{min} - 1$  what is impossible. This contradiction shows that the spheres of radius  $t$  about codewords are disjoint. This means that if  $t$  or fewer errors occur, the received vector  $\mathbf{c}_3$  is in a sphere of radius  $t$  about a unique, closest codeword  $\mathbf{c}_1$ . We decode  $\mathbf{c}_3$  to  $\mathbf{c}_1$ .  $\square$

The largest integer  $t \leq (d_{min} - 1)/2$  is called the packing radius  $T$ .

The data word  $\mathbf{d}$  is the message we want to send. The codeword  $\mathbf{c}$  is transmitted through a channel, and the vector  $\mathbf{v}$  called the sense word is received at the output of the channel.

**Definition 5 (linear code)** A linear code  $C$  is a linear subspace of  $\mathbb{F}_q^n$

**Theorem 2** For a linear code  $C$  the minimum distance is equal to the minimum weight.

**PROOF:**  $d(x, y) = d(x - y, 0) = wt(x - y)$ .  $\square$

We shall denote a linear code  $C$  as a  $[n, k, d]$ -code if:

- $n =$  length of the code
- $k = \dim_{\mathbb{F}_q}(C)$
- $d = d_{min}$

# Chapter 2

## Syndromes and Cyclic Codes

### 2.1 Syndromes

In this section  $C$  is a linear  $[n, k, d]$ -code over  $\mathbb{F}_q$ . Our codeword  $\mathbf{c}$  is transmitted through a channel, and the sense word  $\mathbf{v}$  is received at the output of the channel. If not more than  $T$  components are in error the decoder must recover the codeword (from which the data word can be derived) from the sense word.

I shall only consider words whose alphabet is a finite field.

**Definition 6 (error)** *The error in the  $i$ -th component of the codeword is  $e_i = v_i - c_i$ .*

So the sense word can be seen as the codeword with an error  $\mathbf{e}$ ,  $\mathbf{v} = \mathbf{c} + \mathbf{e}$ , and the error vector  $\mathbf{e}$  is nonzero in at most  $T$  components. A linear code over a finite field  $\mathbb{F}_q$  is associated with a *check matrix*  $\mathbf{H}$ .

**Definition 7 (parity check matrix)**  $\mathbf{H}$  is a (parity) check matrix for  $C$  if and only if

- its rows are independent,
- $C = \{\mathbf{c} \in \mathbb{F}_q^n \mid \mathbf{c}\mathbf{H}^T = 0\}$  (That is,  $C$  is the null space of the linear map:  $x \mapsto x \cdot \mathbf{H}^T$ ).

Therefore,  $\mathbf{H}$  is a  $(n - k) \times n$  matrix and

$$\mathbf{v}\mathbf{H}^T = (\mathbf{c} + \mathbf{e})\mathbf{H}^T = \mathbf{e}\mathbf{H}^T$$

**REMARK:** Note that  $\mathbf{H}^T$  means the transpose of  $\mathbf{H}$ . The exponent  $T$  here has nothing to do with the packing radius  $T$ .

The parity check matrix helps us decoding a codeword, checking if a word is a codeword. We have also a matrix that generates codewords, a *generator matrix*.

**Definition 8 (generator matrix)** A generator matrix  $G$  for a linear code  $C$  is a  $k$  by  $n$  matrix for which the rows are a basis of  $C$ .

We shall say that  $G$  is in standard form (often called *reduced echelon form*) if  $G = [I_k \mid A]$ .

**Definition 9 (syndrome)** For a linear code with check matrix  $H$ , and a sense word  $\mathbf{v}$ , the syndrome vector  $\mathbf{s}$  is  $\mathbf{s} = \mathbf{vH}^T = \mathbf{eH}^T$

Note that by definition of  $H$ ,  $\mathbf{s} = \mathbf{0}$  if and only if  $\mathbf{v} \in C$ .

The task of decoding consists of two parts:

- Computing the syndrome vector  $\mathbf{s} = \mathbf{vH}^T$ , which is a linear operation taking an  $n$  vector to an  $n - k$  vector.
- Solving the equation  $\mathbf{s} = \mathbf{eH}^T$ .

The task of decoding is now the task of solving these  $n - k$  equations for the  $n$ -vector  $\mathbf{e}$  of minimum weight ( $\leq T$ ).

## 2.2 Dual code

Since we have the definition of codewords as vectors in  $\mathbb{F}_q^n$ , it is natural to define the 'inner product' of two codewords  $\mathbf{x} = (x_1, \dots, x_n)$  and  $\mathbf{y} = (y_1, \dots, y_n)$  in the natural way:  $\mathbf{x} \cdot \mathbf{y} = x_1y_1 + \dots + x_ny_n$ .

Further, two vectors  $\mathbf{x}$  and  $\mathbf{y}$  are orthogonal if and only if  $\mathbf{x} \cdot \mathbf{y} = 0$ . With these definitions, we can look at the set of all vectors in  $\mathbb{F}_q^n$  which are orthogonal to all vectors in  $C$ , called the dual code of  $C$  and denoted  $C^\perp$ .

**Theorem 3** If  $C$  is an arbitrary linear code, then  $C^\perp$  is linear.

**PROOF:** Suppose  $\mathbf{x}$  and  $\mathbf{y}$  are two vectors in  $C^\perp$  and  $a$  and  $b$  are elements of  $\mathbb{F}_q$ . Let  $\mathbf{z}$  be a codeword in  $C$ , then

$$\begin{aligned} (a\mathbf{x} + b\mathbf{y}) \cdot \mathbf{z} &= a(\mathbf{x} \cdot \mathbf{z}) + b(\mathbf{y} \cdot \mathbf{z}) \\ &= a(0) + b(0) \\ &= 0 \end{aligned}$$

and so  $a\mathbf{x} + b\mathbf{y}$  is also in  $C^\perp$ . □.

**Theorem 4** If  $C$  is a linear  $[n, k]$ -code, then the dual code  $C^\perp$  of  $C$  is a linear  $[n, n - k]$ -code.

**PROOF:** If the code  $C$  over  $\mathbb{F}_q$  has basis  $\{c_1, \dots, c_k\}$  then  $C^\perp$  is precisely

$$\{(a_0, \dots, a_{n-1}) \in \mathbb{F}_q^n \mid \begin{pmatrix} c_1 \\ \vdots \\ c_k \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix} = 0\}$$

The matrix with the  $c_i$ 's as rows has rank  $k$  and thus the dimension of the null space is  $n - k$ . Hence,  $C^\perp$  has dimension  $n - k$ .  $\square$

## 2.3 Example: The (7, 4)-Hamming Code

Let us now see how we can construct the check matrix for our Hamming code. According to the Venn diagram we must have an even number of 1's in each circle. In other words,  $C$  is fully specified as those  $\mathbf{x}$ 's that satisfy the following equations

$$\begin{aligned} x_1 + x_3 + x_4 + x_5 &= 0 \\ x_1 + x_2 + x_4 + x_6 &= 0 \\ x_1 + x_2 + x_3 + x_7 &= 0. \end{aligned}$$

In matrix form this is written as  $\mathbf{xH}^T = \mathbf{0}$  where

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \text{ is the parity check matrix,}$$

and  $\mathbf{x} = (x_1, \dots, x_7)$  with  $x_1, \dots, x_4$  the information symbols and  $x_5, \dots, x_7$  the parity checks.

This matrix can *check* if a word is really a codeword. For example, suppose we received the following word:

$$(1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0).$$

We want to see if errors have been occurred during transmission. So we multiply by the transpose of our check matrix:

$$(1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0) \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}^T = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}.$$

We see immediately that the error occurred in position 5.

Now we have described a method of decoding a word using a parity check matrix.

As we know from the definition of a generator matrix given before, the rows of it form a basis of  $C$ .

A generator matrix for the  $[7, 4]$ -Hamming code is (in standard form)

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

To encode for example (1011), the multiplication is carried out as follows

$$(1 \ 0 \ 1 \ 1) \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix} = (1 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0)$$

which happily is the same result as when the Venn diagram is used.

Now that we know something more about the  $[7, 4]$ -Hamming code I want to prove that it is a linear code.

**Theorem 5** *The  $[7, 4]$ -Hamming code is a linear code.*

**PROOF:** Suppose  $\mathbf{H}$  is the parity check matrix for the  $[7, 4]$ -Hamming code  $C$ , and  $\mathbf{x}, \mathbf{y} \in C$  that is,  $\mathbf{x}\mathbf{H}^T = \mathbf{0}$  and  $\mathbf{y}\mathbf{H}^T = \mathbf{0}$ . Let  $a, b \in \mathbb{F}_q$ . Then

$$(a\mathbf{x} + b\mathbf{y})\mathbf{H}^T = a\mathbf{x}\mathbf{H}^T + b\mathbf{y}\mathbf{H}^T = a \cdot \mathbf{0} + b \cdot \mathbf{0} = \mathbf{0}$$

so  $a\mathbf{x} + b\mathbf{y} \in C$  and  $C$  is linear.  $\square$

**Theorem 6 (Hamming distance)** *The  $(7, 4)$ -Hamming code  $C$  with parity check matrix  $H$  has Hamming distance 3, and so is 1-error-correcting.*

**PROOF:** We will prove this by showing that all nonzero codewords in  $C$  have weight at least 3. Suppose  $\mathbf{x} = (x_1, \dots, x_7) \in C$  has weight 1, that is,  $x_i = 1$  for some  $i$ , with  $x_j = 0$  for all  $j \neq i$ . This contradicts that  $\mathbf{x}\mathbf{H}^T = \mathbf{0}$  since no  $i^{\text{th}}$  column of  $\mathbf{H}$  consists of all zeros, as it would need to be.

Next suppose  $\mathbf{x} \in C$  has weight 2, and let  $x_i = x_j = 1$  with  $x_k = 0$  for all  $k$  other than  $j$  and  $i$ . Denoting the  $s$ -th row of  $\mathbf{H}$  by  $h_{s1}, h_{s2}, \dots, h_{s7}$ , we have, since  $\mathbf{x}$  is orthogonal to each row in  $\mathbf{H}$  that for all  $1 \leq s \leq n - k$ ,  $h_{si} + h_{sj} = 0$  which under modulo 2 means  $h_{si} = h_{sj}$ . This would mean that some two columns of  $\mathbf{H}$  were identical, which no two are, and we have reached a contradiction.

Finally, consider the codeword  $(0, 1, 0, 0, 0, 1, 1)$ , which satisfies the parity check matrix. It has weight 3, and so we have proven  $d(C) = 3$ .  $\square$

## 2.4 Cyclic codes

Cyclic codes are much studied and very useful error correcting codes. BCH codes (developed by Bose, Chaudhuri, and Hocquenghem) are a very important type of cyclic codes. Reed-Solomon codes are a special type of BCH codes that are commonly used in compact disc players. The cyclic codes we will explore in this paper are also linear codes.

**Definition 10 (cyclic codes)** *A code is called cyclic if  $(c_n, c_0, c_1, \dots, c_{n-1})$  is a codeword whenever  $(c_0, c_1, \dots, c_{n-1}, c_n)$  is also a codeword.*

The most important tool in the description of cyclic codes is the isomorphism between  $\mathbb{F}_q^n$  and  $\mathbb{F}_q[x]/(x^n - 1)$ . We can make the following identification:

$$\mathbb{F}_q^n \ni (c_0, c_1, \dots, c_{n-1}) \leftrightarrow c_0 + c_1x + \dots + c_{n-1}x^{n-1} \in \mathbb{F}_q[x]/(x^n - 1)$$

We shall often speak of a codeword  $\mathbf{c}$  as the codeword  $c(x)$ . Extending this, we interpret a linear code as a subset of  $\mathbb{F}_q[x]/(x^n - 1)$ .

**Theorem 7** *A linear code  $C$  in  $\mathbb{F}_q^n$  is cyclic if and only if  $C$  is an ideal in  $\mathbb{F}_q[x]/(x^n - 1)$ .*

**PROOF:**

- If  $C$  is an ideal in  $\mathbb{F}_q[x]/(x^n - 1)$  and  $c(x) = c_0 + c_1x + \dots + c_{n-1}x^{n-1}$  is any codeword, then  $xc(x)$  is also a codeword i.e.  $(c_{n-1}, c_0, c_1, \dots, c_{n-2}) \in C$ .
- If  $C$  is cyclic, then for every codeword  $c(x)$  the word  $xc(x)$  is also in  $C$ . Therefore  $x^i c(x)$  is in  $C$  for every  $i$ , and since  $C$  is linear  $a(x)c(x)$  is also in  $C$  for every polynomial  $a(x)$ . Hence  $C$  is an ideal.  $\square$

From now on we only consider cyclic codes of length  $n$  over  $\mathbb{F}_q$  with  $\text{GCD}(n, q) = 1$ .

The ring  $\mathbb{F}_q[x]/(x^n - 1)$  is a principal ideal ring so each ideal in this ring is a principal ideal. This means that every element in an ideal  $I$  is a multiple of a fixed monic polynomial of lowest degree  $g(x)$ :  $g(x)$  generates  $I$  (notice that  $g(x)$  is not necessarily the only polynomial that generates  $I$ ). In other words:  $g(x)$  generates a cyclic code of length  $n$ . This polynomial  $g(x)$  is called the generator polynomial of the cyclic code.

The generator polynomial is a divisor of  $x^n - 1$  (since otherwise the greatest common divisor of  $x^n - 1$  and  $g(x)$  would be a polynomial in  $C$  of degree lower

than the degree of  $g(x)$ ). Let  $x^n - 1 = f_1(x)f_2(x)\dots f_t(x)$  be the decomposition of  $x^n - 1$  into irreducible factors. We can now find all cyclic codes of length  $n$  by picking (in every possible way) one of the  $2^t$  factors of  $x^n - 1$  as generator polynomial  $g(x)$  and defining the corresponding code to be the set of multiples of  $g(x) \bmod (x^n - 1)$ .

If we want to encode messages of length  $k$  we must find a generator polynomial of degree  $n - \dim(C) = n - k$ .

**Theorem 8** *Let  $C$  be a cyclic code over  $\mathbb{F}_q^n$  with generator polynomial  $g(x)$ . Then the degree of  $g(x)$  is equal to  $n - k$ .*

**PROOF:** Since each code word must be divisible by  $g(x)$  and has degree at most  $n$ , it can be written as  $d(x)g(x)$  with degree of  $d(x)$  less than  $n - \deg(g(x))$ . Since  $C$  has dimension  $k$ , we can conclude that  $k = n - \deg(g(x))$ , thus  $\deg(g(x)) = n - k$ .  $\square$

**Example 2** *We try to find a generating polynomial for an arbitrary code of length  $n = 15$  which will encode messages of length  $k = 7$ .*

*If we are to find a generator for a code of length 15 to encode messages of length 7 we need to find a divisor of  $x^{15} + 1$  of degree  $15 - 7 = 8$ .*

*The polynomial  $x^{15} + 1$  can be written as*

$$x^{15} + 1 = (1 + x)(1 + x + x^2)(1 + x + x^2 + x^3 + x^4)(1 + x + x^4)(1 + x^3 + x^4)$$

*so we can choose*

$$g(x) = (1 + x + x^2 + x^3 + x^4)(1 + x + x^4) = 1 + x^4 + x^6 + x^7 + x^8$$

*Using this generator polynomial we can create a code with minimum distance 5 and thus correct 2 errors. Now that we have a generator polynomial we can use it to encode for example (0110110). The polynomial corresponding to this vector is  $x + x^2 + x^4 + x^5$ . To encode it, we multiply this by  $g(x)$ :*

$$\begin{aligned} (x + x^2 + x^4 + x^5)(1 + x^4 + x^6 + x^7 + x^8) = \\ x + x^2 + x^4 + x^6 + x^7 + x^8 + x^9 + x^{13} \end{aligned}$$

*Thus the word (0110110) is encoded to the codeword (011010111100010).  $\square$*

If  $g(x)$  has degree  $n - k$  then the codewords  $g(x), xg(x), \dots, x^{k-1}g(x)$  clearly form a basis for  $C$  ( $C$  is an  $[n, k]$ -code). Hence, if  $g(x) = g_0 + g_1x + \dots + g_{n-k}x^{n-k}$  then

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & \cdots & g_{n-k-1} & g_{n-k} & 0 & \cdots & 0 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & \cdots & & g_0 & g_1 & \cdots & g_{n-k} \end{pmatrix}$$

is a generator matrix for  $C$ . This means that we encode a data message  $\mathbf{d} = (d_0, d_1, \dots, d_{k-1})$  as  $\mathbf{d}G$  which is the polynomial

$$(d_0 + d_1x + \dots + d_{k-1}x^{k-1})g(x)$$

Since  $g(x)$  is a divisor of  $x^n - 1$  there is a polynomial  $h(x) = h_0 + h_1x + \dots + h_kx^k$  such that  $g(x)h(x) = x^n - 1$  (in  $\mathbb{F}_q[x]$ ). In the ring  $\mathbb{F}_q[x]/(x^n - 1)$  we have  $g(x)h(x) = 0$ . It follows that

$$H = \begin{pmatrix} 0 & 0 & \cdots & 0 & h_k & \cdots & h_1 & h_0 \\ 0 & 0 & \cdots & h_k & \cdots & h_1 & h_0 & 0 \\ \vdots & & & & & & & \vdots \\ h_k & \cdots & h_1 & h_0 & 0 & \cdots & 0 \end{pmatrix}$$

is a parity check matrix for the code  $C$ . We call  $h(x)$  the *check polynomial* of  $C$ . The code  $C$  consists of all  $c(x)$  such that  $c(x)h(x) = 0$ .

### 2.4.1 BCH and Reed-Solomon codes

Hocquenghem (1959) and Bose and Ray-Chaudhuri (1960) independently discovered an important class of linear cyclic codes which enable us to correct several errors. These are polynomial codes<sup>1</sup> and are now called Bose-Chaudhuri-Hocquenghem codes (BCH codes). Recall that a polynomial code is determined as soon as the generator polynomial is determined.

**Definition 11 (BCH code)** A cyclic code of length  $n$  over  $\mathbb{F}_q$  is called a BCH code of minimum distance  $d$  if its generator polynomial  $g(x)$  is the least common multiple of the minimal polynomials of  $\alpha^1, \dots, \alpha^{d-1}$  where  $\alpha$  is a primitive  $n$ -th root of unity. If  $n = q^m - 1$ , (so if  $\alpha$  is a primitive element of  $\mathbb{F}_{q^m}$ ), then the BCH code is called primitive.

Note that the minimum distance  $d$  of the above definition is a essentially different than the  $d$  given in definition 3.

**Theorem 9** The BCH code of minimum distance  $d$  has minimum distance at least  $d$ .

<sup>1</sup>from now on we call the code words in  $\mathbb{F}_q[x]/(x^n - 1)$ , polynomial codes

**PROOF:** Let  $h(x)$  be any polynomial over  $\mathbb{F}_q$  which has  $\alpha, \alpha^2, \dots, \alpha^{d-1}$  among its roots. Let  $m_i(x)$  be the minimal polynomial of  $\alpha^i$  then

$$m_i(x) \mid h(x), \forall 1 \leq i \leq d-1$$

and hence  $g(x) \mid h(x)$ . Thus  $g(x)$  is the polynomial of least possible degree with roots  $\alpha, \alpha^2, \dots, \alpha^{d-1}$ .

If  $c(x)$  is a code polynomial then we have  $c(x) = a(x)g(x)$  for some  $a(x) \in \mathbb{F}_q[x]$  and thus  $\alpha, \alpha^2, \dots, \alpha^{d-1}$  are zeros of  $c(x)$ . The code generated by  $g(x)$  has minimum distance at least  $d$  if there is no code word  $(c_0, c_1, \dots, c_{n-1})$  with less than  $d$  non-zero entries.

Suppose, on the contrary, that a code word has less than  $d$  non-zero entries. Then the corresponding polynomial code is of the form

$$c(x) = b_1x^{n_1} + b_2x^{n_2} + \dots + b_{d-1}x^{n_{d-1}}$$

where  $b_1, b_2, \dots, b_{d-1} \in \mathbb{F}_q$  and also, we may assume that  $n_1 > n_2 > \dots > n_{d-1} \geq 0$ .

Since the code is of length  $n$ , every code polynomial is of degree at most  $n-1$  and therefore  $n_1 \leq n-1$ . We have that  $\alpha, \alpha^2, \dots, \alpha^{d-1}$  are roots of  $c(x)$ , which implies

$$\begin{aligned} b_1\alpha^{n_1} + \dots + b_{d-1}\alpha^{n_{d-1}} &= 0 \\ b_1\alpha^{2n_1} + \dots + b_{d-1}\alpha^{2n_{d-1}} &= 0 \\ \vdots &\vdots \\ b_1\alpha^{(d-1)n_1} + \dots + b_{d-1}\alpha^{(d-1)n_{d-1}} &= 0. \end{aligned}$$

In matrix form:

$$A \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{d-1} \end{pmatrix} = 0 \text{ where } A = \begin{pmatrix} \alpha^{n_1} & \alpha^{n_2} & \dots & \alpha^{n_{d-1}} \\ \alpha^{2n_1} & \alpha^{2n_2} & \dots & \alpha^{2n_{d-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha^{(d-1)n_1} & \alpha^{(d-1)n_2} & \dots & \alpha^{(d-1)n_{d-1}} \end{pmatrix} \quad (2.1)$$

We know from linear algebra that the determinant of  $A$  is equal to  $\det A = \prod_{i>j} (\alpha^{n_i} - \alpha^{n_j})$ . Now (2.1) is a system of  $d-1$  homogeneous linear equations in  $d-1$  variables  $b_1, \dots, b_{d-1}$  and  $\det(A) \neq 0$ . Therefore the system of equations admits only the zero solution and  $c(x) = 0$ . Hence there is no non-zero code word with less than  $d$  non-zero entries and the code has minimum distance at least  $d$ .  $\square$

**Example 3** As an example we are going to construct a binary BCH code of length 7 and minimum distance 3. We need to construct an extension of  $\mathbb{F}_2$  of degree  $p$  where  $2^p - 1$  is a multiple of 7. Thus we take  $p = 3$ . We know that  $\mathbb{F}_2[x]/(x^3 + x + 1)$  is a field of order 8 and that  $\alpha = x + (x^3 + x + 1)$  is a primitive element of  $\mathbb{F}_2[x]/(x^3 + x + 1)$ . Then  $\alpha$  satisfies  $\alpha^3 + \alpha + 1 = 0$  and  $\alpha^7 = 1$  and  $x^3 + x + 1$  is the minimal polynomial of  $\alpha$ . We take  $x^3 + x + 1$  as a generator polynomial for our code.

The data polynomials are of degree at most 3. If  $d(x) = d_0 + d_1x + d_2x^2 + d_3x^3$  is an arbitrary data polynomial, the corresponding code polynomial is  $d(x)(x^3 + x + 1) = (d_0, d_1 + d_0, d_2 + d_1, d_3 + d_2 + d_0, d_3 + d_1, d_2, d_3)$ .

It is easy to see that the minimum possible weight for this vector is 3, therefore the code has minimum distance 3.

If we had started with the primitive polynomial  $x^3 + x^2 + 1$ , the corresponding BCH code with code word length 7 and minimum distance at least 3 is the polynomial code with generator polynomial  $x^3 + x^2 + 1$ .

Notice that this is the  $[7, 4]$ -Hamming code as explained before.  $\square$

One of the simplest examples of BCH codes, namely the case  $n = q - 1$ , turns out, as we shall see later, to have many important applications.

**Definition 12 (Reed-Solomon)** A Reed-Solomon code is a primitive BCH code of length  $n = q - 1$  over  $\mathbb{F}_q$ . The generator polynomial of such a code has the form  $g(x) = \prod_{i=1}^{d-1} (x - \alpha^i)$  where  $\alpha$  is a primitive element.

Because decoding algorithms for cyclic codes are often using properties of the Fourier transform, we shall present its relevant properties here.

### 2.4.2 Fourier transform

The (discrete) Fourier transform can be defined as follows:

**Definition 13 (Fourier Transform)** Let  $\mathbf{v}$  be a vector of length  $n$  over the field  $\mathbb{F}$ . Let  $\omega$  be an element of  $\mathbb{F}$  of order  $n$ . The Fourier Transform of  $\mathbf{v}$  is given by

$$\mathbf{V} = (V_0, \dots, V_n), \text{ with } V_j = \sum_{i=0}^{n-1} \omega^{ij} v_i, \quad j = 0, \dots, n-1$$

The vector  $\mathbf{V}$  is called the *Fourier spectrum* and the components of  $\mathbf{V}$  the *spectral components*. If  $\mathbb{F}$  has no elements of order  $n$ , then a Fourier transform does not exist.

If  $V$  is the Fourier transform of  $v$ , then  $v$  can be recovered from  $V$  by *the inverse Fourier transform*, which is given by

$$v_i = \frac{1}{n} \sum_{k=0}^{n-1} \omega^{-ik} V_k, \quad i = 0, \dots, n-1.$$

**PROOF:**

$$\sum_{k=0}^{n-1} \omega^{-ik} V_k = \sum_{k=0}^{n-1} \omega^{-ik} \sum_{l=0}^{n-1} \omega^{lk} v_l = \sum_{l=0}^{n-1} v_l \left[ \sum_{k=0}^{n-1} \omega^{-k(i-l)} \right]$$

The sum over  $k$  is clearly equal to  $n$  if  $l = i$ . But if  $l$  is not equal to  $i$ , then the summation becomes

$$\sum_{k=0}^{n-1} (\omega^{-(l-i)})^k = \frac{1 - \omega^{-(l-i)n}}{1 - \omega^{-(l-i)}}.$$

Notice that  $1 - \omega^{-(l-i)} \neq 0$ , because  $-n < l - 1 < n$  and  $l - i \neq 0$ . This is equal to zero because  $\omega^{-n} = 1$ . Hence

$$\sum_{k=0}^{n-1} \omega^{-ik} V_k = \sum_{l=0}^{n-1} v_l (n\delta_{il}) = nv_i$$

where  $\delta_{il} = 1$  if  $i = l$  and otherwise  $\delta_{il} = 0$ . □

# Chapter 3

## Majority Decoding

Majority decoding is a method of decoding which finds the errors by a majority vote. This type of decoding is only suitable for small codes that must be decoded quickly and simply.

As an example we consider the Reed-Muller codes. For decoding these, a majority algorithm called the Reed algorithm is very suitable.

### 3.1 Reed-Muller Codes

Reed-Muller codes are a class of linear codes over  $\mathbb{F}_2$  that can be decoded by a simple voting technique. Although this class of codes can be generalized to other fields, we shall only discuss the binary case.

Let  $v = (v_1, \dots, v_m)$  denote a vector in  $\mathbb{F}_2^m$ . We shall choose the 'lexicographical' ordering of the  $2^m = n$  points of  $\mathbb{F}_2^m$ , where the first coordinate is the most significant. The successive points of  $\mathbb{F}_2^m$  are named  $\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_n$ . For example, when  $m = 3$ , one gets in this way

$$\tilde{v}_1 = (0, 0, 0)$$

$$\tilde{v}_2 = (0, 0, 1)$$

$$\tilde{v}_3 = (0, 1, 0)$$

$$\tilde{v}_4 = (0, 1, 1)$$

$$\tilde{v}_5 = (1, 0, 0)$$

$$\tilde{v}_6 = (1, 0, 1)$$

$$\tilde{v}_7 = (1, 1, 0)$$

$$\tilde{v}_8 = (1, 1, 1)$$

**Definition 14 (Reed-Muller code)** Let  $0 \leq r \leq m$ . Consider the linear space  $L_m(r)$  of all polynomials over  $\mathbb{F}_2$  of degree at most  $r$  in  $m$  variables. Put  $n = 2^m$  and consider the evaluation map:

$$\begin{aligned} ev : L_m(r) &\longrightarrow \mathbb{F}_2^n \\ f &\mapsto (f(\tilde{v}_1), f(\tilde{v}_2), \dots, f(\tilde{v}_n)) \end{aligned}$$

Then the  $r$ -th order binary Reed-Muller code  $RM(r, m)$  of length  $2^m = n$  is  $RM(r, m) = ev(L_m(r))$ .

In other words, evaluate all  $f$  with  $\deg(f) \leq r$  at  $\tilde{v}_1, \tilde{v}_2, \dots, \tilde{v}_n$ .

**Example 4 (RM(1,3))** The first order Reed-Muller code of length 8 consist of the 16 codewords

$$a_0 \mathbf{1} + a_1 \mathbf{v}_1 + a_2 \mathbf{v}_2 + a_3 \mathbf{v}_3, \quad a_i = 0 \text{ or } 1$$

which are shown below

|   |            |
|---|------------|
| $\mathbf{0}$  | (00000000) |
| $\mathbf{v}_1$  | (00001111) |
| $\mathbf{v}_2$  | (00110011) |
| $\mathbf{v}_3$  | (01010101) |
| $\mathbf{v}_1 + \mathbf{v}_2$                             | (00111100) |
| $\mathbf{v}_1 + \mathbf{v}_3$                             | (01011010) |
| $\mathbf{v}_2 + \mathbf{v}_3$                             | (01100110) |
| $\mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3$              | (01101001) |
| $\mathbf{1}$  | (11111111) |
| $\mathbf{1} + \mathbf{v}_1$                               | (11110000) |
| $\mathbf{1} + \mathbf{v}_2$                               | (11001100) |
| $\mathbf{1} + \mathbf{v}_3$                               | (10101010) |
| $\mathbf{1} + \mathbf{v}_1 + \mathbf{v}_2$                | (11000011) |
| $\mathbf{1} + \mathbf{v}_1 + \mathbf{v}_3$                | (10100101) |
| $\mathbf{1} + \mathbf{v}_2 + \mathbf{v}_3$                | (10011001) |
| $\mathbf{1} + \mathbf{v}_1 + \mathbf{v}_2 + \mathbf{v}_3$ | (10010110) |

□

There are exactly  $\binom{m}{r}$  distinct monomials of degree  $r$  in  $m$  variables in which no variable occurs to a power  $\geq 2$ , the power is 0 or 1. The total number of distinct monomials of degree at most  $r$  is:

$$1 + \binom{m}{1} + \cdots + \binom{m}{r} = k$$

In this section we want to prove that the dimension of this code is  $k$  and that it has minimal distance equal to  $2^{m-r}$ .

**Theorem 10**  $RM(r, m)$  has dimension equal to  $k$ .

In order to prove that the dimension of the Reed-Muller code is  $k$ , we define  $L'_m(r)$  to be the subspace of  $L_m(r)$  spanned by this type of monomials. Note that  $\dim(L'_m(r)) = k$ .

**Proposition 1** The evaluation map

$$\begin{aligned} ev : L'_m(r) &\longrightarrow \mathbb{F}_2^n \\ f &\mapsto (f(\tilde{v}_1), f(\tilde{v}_2), \dots, f(\tilde{v}_n)) \end{aligned}$$

is injective.

**PROOF:** It is clear that if the evaluation map is injective for  $r = m$  it is also for smaller  $r$ . Consider therefore  $r = m$ , then we have  $1 + \binom{m}{1} + \cdots + \binom{m}{m} = 2^m = \dim(L'_m(m))$ . Notice that  $\dim(\mathbb{F}_2^n) = n = 2^m = \dim(L'_m(m))$ , we know now from linear algebra that if the evaluation map is surjective it is also injective because both spaces have the same dimension. Consider the polynomial  $F = \prod_{i=1}^m (x_i + a_i + 1) \in L_m(r)$ . This polynomial attains the value 1 for  $\mathbf{x} = \mathbf{a}$  and is zero at all other points of  $\mathbb{F}_2^m$ . Each vector with weight one in  $\mathbb{F}_2^n$  is an element of  $RM(m, m)$ , hence  $\mathbb{F}_2^n \leq RM(m, m) \leq \mathbb{F}_2^n$ , so  $RM(m, m) = \mathbb{F}_2^n$ . Therefore the evaluation map is surjective and also injective.  $\square$

**Proposition 2**  $ev(L'_m(r)) = ev(L_m(r))$

**PROOF:**  $\mathbf{x}^2 = \mathbf{x}$  for  $\mathbf{x} \in \mathbb{F}_2$  implies that  $f(x_1, \dots, x_i^k, \dots, x_m) = f(x_1, \dots, x_m)$  for every polynomial  $f(x)$ , from which the result follows.  $\square$

We can conclude now that the vectors corresponding to the monomials in  $L'_m(r)$  form a basis for the code and the code has dimension  $k$ .  $\square$

For example when  $m = 4$  the 16 possible basis vectors for Reed-Muller codes of length 16 are shown below

$$\begin{aligned}
\mathbf{v}_0 = \mathbf{1} &= (1111111111111111) \\
\mathbf{v}_4 &= (0000000011111111) \\
\mathbf{v}_3 &= (0000111100001111) \\
\mathbf{v}_2 &= (0011001100110011) \\
\mathbf{v}_1 &= (0101010101010101) \\
\mathbf{v}_3\mathbf{v}_4 &= (0000000000001111) \\
\mathbf{v}_2\mathbf{v}_4 &= (0000000000110011) \\
\mathbf{v}_1\mathbf{v}_4 &= (0000000001010101) \\
\mathbf{v}_2\mathbf{v}_3 &= (0000001100000011) \\
\mathbf{v}_1\mathbf{v}_3 &= (0000010100000101) \\
\mathbf{v}_1\mathbf{v}_2 &= (0001000100010001) \\
\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4 &= (0000000000000011) \\
\mathbf{v}_1\mathbf{v}_3\mathbf{v}_4 &= (0000000000000101) \\
\mathbf{v}_1\mathbf{v}_2\mathbf{v}_4 &= (0000000000001001) \\
\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3 &= (0000000100000001) \\
\mathbf{v}_1\mathbf{v}_2\mathbf{v}_3\mathbf{v}_4 &= (0000000000000001)
\end{aligned}$$

### 3.1.1 Reed algorithm

The Reed algorithm for decoding a Reed-Muller code can be explained by an example. Consider the second order code for  $m = 4$ , the  $[16, 11, 4]$ -Reed-Muller code.

The Reed algorithm is unusual in that it does not compute syndromes or the error pattern. It computes the data symbols directly from the sense word.

**Example 5** *As we have seen the generator matrix of this code is:*

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

The 11 data symbols  $d_0, \dots, d_{10}$  are coded into the vector:

$$\begin{aligned} & d_0 \mathbf{v}_0 + d_1 \mathbf{v}_4 + d_2 \mathbf{v}_3 + d_3 \mathbf{v}_2 + d_4 \mathbf{v}_1 \\ & + d_5 \mathbf{v}_3 \mathbf{v}_4 + d_6 \mathbf{v}_2 \mathbf{v}_4 + d_7 \mathbf{v}_1 \mathbf{v}_4 + d_8 \mathbf{v}_2 \mathbf{v}_3 + d_9 \mathbf{v}_1 \mathbf{v}_3 \\ & + d_{10} \mathbf{v}_1 \mathbf{v}_2 = (c_0, \dots, c_{15}) \end{aligned}$$

The problem is to determine the  $d$ 's from the received word even if errors have occurred.

The sum of the first four components (as elements of  $\mathbb{F}_2$ ) is zero for every basis vector except  $\mathbf{v}_1 \mathbf{v}_2$ . Thus if no error occurs:

$$c_0 + c_1 + c_2 + c_3 = d_{10}$$

the same is true for the next four components and further, so we have:

$$\begin{aligned} c_4 + c_5 + c_6 + c_7 &= d_{10} \\ c_8 + c_9 + c_{10} + c_{11} &= d_{10} \\ c_{12} + c_{13} + c_{14} + c_{15} &= d_{10} \end{aligned}$$

So there are four independent determinations of  $d_{10}$ , in general there would be  $2^{m-r}$  independent determinations. If there is an error in it, it only can affect one determination and so  $d_{10}$  is equal to the value occurring most frequently. If the errors made are at most  $2^{m-r-1} - 1 = 1$ ,  $d_{10}$  will still be decoded correctly.

If we take the first component, the fifth, the ninth and the thirteenth and so further we get equations for  $d_5$ . Similar we can determine  $d_6, d_7, d_8$  and  $d_9$ .

After these have been determined,

$$\begin{aligned} & d_5 \mathbf{v}_3 \mathbf{v}_4 + d_6 \mathbf{v}_2 \mathbf{v}_4 + d_7 \mathbf{v}_1 \mathbf{v}_4 + d_8 \mathbf{v}_2 \mathbf{v}_3 \\ & + d_9 \mathbf{v}_1 \mathbf{v}_3 + d_{10} \mathbf{v}_1 \mathbf{v}_2 \end{aligned}$$

can be subtracted from the received word. This would leave in absence of errors:

$$\begin{aligned}\mathbf{r}' &= d_0\mathbf{v}_0 + d_1\mathbf{v}_4 + d_2\mathbf{v}_3 + d_3\mathbf{v}_2 + d_4\mathbf{v}_1 \\ &= (c'_0, c'_1, c'_2, c'_3, c'_4)\end{aligned}$$

And  $d_0, \dots, d_4$  can be determined as above:

$$\begin{aligned}d_1 &= c'_0 + c'_1 = c'_2 + c'_3 = c'_4 + c'_5 = c'_6 + c'_7 = c'_8 + c'_9 \\ &= c'_{10} + c'_{11} = c'_{12} + c'_{13} = c'_{14} + c'_{15}\end{aligned}$$

Similar equations hold for  $d_2, \dots, d_4$ . Finally, in the absence of errors, we have

$$\mathbf{r}' - d_4\mathbf{v}_1 - d_3\mathbf{v}_2 - d_2\mathbf{v}_3 - d_1\mathbf{v}_4 = d_0\mathbf{v}_0$$

This should be all 0's if  $d_0 = 0$ , all 1's if  $d_0 = 1$  and  $d_0$  can be taken to which occurs most.  $\square$

Since this decoding algorithm can correct all combinations of  $2^{m-r-1} - 1$  or fewer errors, the minimal distance must be at least  $2(2^{m-r-1} - 1) + 1 = 2^{m-r} - 1$ , and since the code vectors all have even weight (because the image of a basis vector from  $L'_m(r)$  has even weight), it must be at least  $2^{m-r}$ . But the last basis vector has weight  $2^{m-r}$ , so this is exactly the minimum distance.

If  $r = m$  we have already concluded that the code is the whole space  $\mathbb{F}_2^m$ , and therefore has minimum distance 1.

# Chapter 4

## Locator decoding

There are a lot of decoding algorithms based on *locator decoding*. Each of these algorithms is based on the use of a certain polynomial, the *(error)-locator polynomial*. We shall denote this locator polynomial as  $\Lambda(x)$ . Locator decoding uses much of the algebraic properties of the (finite) field. These algorithms are very suitable for large codes.

### 4.1 Locator polynomials and the Peterson algorithm

BCH codes are cyclic codes and hence can be decoded by any technique for decoding cyclic codes. The algorithm studied in this section was first developed by Peterson for binary cyclic codes.

Suppose we want to decode a BCH code. The error polynomial is:

$$e(x) = e_0 + e_1x + \cdots + e_{n-1}x^{n-1}$$

where at most  $t$  coefficients are nonzero. Suppose that  $\nu$  errors actually occur,  $0 \leq \nu \leq t$  and that they occur in unknown locations  $i_1, i_2, \dots, i_\nu$ . The error polynomial can be written

$$e(x) = e_{i_1}x^{i_1} + e_{i_2}x^{i_2} + \cdots + e_{i_\nu}x^{i_\nu}$$

where  $e_{i_\ell}$  is the magnitude of the  $\ell$ -th error ( $e_{i_\ell} = 1$  for binary codes).

We do not know  $i_1, \dots, i_\nu$  neither do we know  $e_{i_1}, \dots, e_{i_\nu}$ . In fact, we do not even know the value of  $\nu$ .

These must be computed to correct the errors. Evaluate the received polynomial

at  $\alpha$  to obtain the syndrome<sup>1</sup>  $S_1$ :

$$\begin{aligned} S_1 &= v(\alpha) = c(\alpha) + e(\alpha) = e(\alpha) \\ &= e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \cdots + e_{i_\nu}\alpha^{i_\nu} \end{aligned}$$

Similarly, we can evaluate the received polynomial at each of the powers of  $\alpha$ :

$$S_j = e(\alpha^j) \text{ for } j = 1, \dots, 2t.$$

Note that doing this is the same as using the Fourier transform to compute the syndromes.

Our task is now to find the unknowns given the syndromes. This is a problem of solving a system of equations

$$\begin{aligned} S_1 &= e_{i_1}\alpha^{i_1} + e_{i_2}\alpha^{i_2} + \cdots + e_{i_\nu}\alpha^{i_\nu} \\ &\vdots \\ S_{2t} &= e_{i_1}\alpha^{2ti_1} + e_{i_2}\alpha^{2ti_2} + \cdots + e_{i_\nu}\alpha^{2ti_\nu} \end{aligned}$$

where  $e_{i_\ell}$  is the magnitude of the  $\ell - th$  error and  $\nu$  the number of errors that have occurred.

The set of equations is too difficult to solve directly. Instead, we define some variables that can be computed from the syndromes and from which the error locations can then be computed.

**Definition 15 (error locator polynomial)** *The (error) locator polynomial  $\Lambda(x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1}x^{\nu-1} + \cdots + \Lambda_1x + 1$  is defined to be the polynomial with zeros at all  $\alpha^{-i_\ell}$  for  $\ell = 1, \dots, \nu$ . That is*

$$\Lambda(x) = \prod_{\ell=1}^{\nu} (1 - x\alpha^{i_\ell})$$

If we knew the coefficients of  $\Lambda(x)$ , we could find the zeros of  $\Lambda(x)$  to obtain the error locations. Therefore we want to compute  $\Lambda_1, \dots, \Lambda_\nu$  from the syndromes.

**Theorem 11 (Convolution theorem)** *Suppose that*

$$e_i = f_i g_i, \quad i = 0, \dots, n-1$$

*Then*

$$E_j = \frac{1}{n} \sum_{k=0}^{n-1} F_{(j-k)} \text{ mod } n G_k \text{ for } j = 0, \dots, n-1,$$

*with  $E, F$  and  $G$  the Fourier transforms of  $e, f$  and  $g$  respectively.*

---

<sup>1</sup>actually it is the syndrome spectrum

This theorem can easily be proved just by taking the Fourier transform of  $e_i = f_i g_i$ .

Clearly,  $\Lambda(\alpha^{-i})$  equals zero if and only if  $i$  is an error location. Therefore  $e_i \Lambda(\alpha^{-i}) = 0$  for all  $i$  and thus by the convolution theorem:

$$\sum_{j=0}^{n-1} \Lambda_j E_{k-j} = 0, \quad k = 0, \dots, n-1,$$

where we can conclude that  $\Lambda(x)E(x) = 0 \pmod{x^n - 1}$ .

Because  $\Lambda(x)$  is a polynomial of degree at most  $t$ ,  $\Lambda_j = 0$  for  $j > t$ . Then

$$\sum_{j=0}^t \Lambda_j E_{k-j} = 0, \quad k = 0, \dots, n-1.$$

Because  $\Lambda_0$  equals one, this can be rewritten in the form

$$E_k = - \sum_{j=1}^t \Lambda_j E_{k-j} = 0, \quad k = 0, \dots, n-1.$$

This is a set of linear equations relating the error spectrum (and so the syndromes) to the coefficients of  $\Lambda(x)$ . We can write this in matrix form:

$$\begin{bmatrix} E_{t-1} & E_{t-2} & \cdots & E_0 \\ E_t & E_{t-1} & \cdots & E_1 \\ \vdots & & \ddots & \vdots \\ E_{2t-2} & E_{2t-3} & \cdots & E_{t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix} = - \begin{bmatrix} E_t \\ E_{t+1} \\ \vdots \\ E_{2t-1} \end{bmatrix}$$

Note that the degree of  $\Lambda(x)$  is at most  $t$  thus the matrix can be smaller if less than  $t$  errors occur. This system can be solved by inverting the matrix if it is nonsingular. I want now to give a theorem (without proof!) which gives conditions implying that the matrix above is nonsingular:

**Theorem 12** *The matrix is nonsingular if it is a  $\nu$  by  $\nu$  matrix.*

One can prove this theorem by using that the Vandermonde matrix is nonsingular, and a little linear algebra. For the exact proof see [?], chapter 7.

Now we can find the correct value of  $\nu$ : we assume  $\nu = t$  and compute the determinant of the matrix. If it is nonzero, this is the correct value of  $\nu$ . Otherwise, if the determinant is zero, reduce the assumed value of  $\nu$  by one and repeat. Continue in this way until a nonzero determinant is obtained. The actual number of

errors that occurred is then known. Next, invert the matrix and compute  $\Lambda(x)$ . Find the zeros of  $\Lambda(x)$  to find the error locations. Usually, because there are only a finite number of elements to check, the simplest way to find the zeros of  $\Lambda(x)$  is by trial and error. One simply computes in turn  $\Lambda(\alpha^j)$  for each  $j$  and checks for zero.

### 4.1.1 Example of the Peterson algorithm

As an example we are going to decode a  $[15, 9]$ -Reed-Solomon code using the Peterson algorithm. This  $[15, 9]$ -Reed-Solomon is a code over  $\mathbb{F}_{16}$ . Because  $n = 15$  is of the form  $q^m - 1$  we know that  $\omega = \alpha$  where  $\alpha$  is a primitive element of  $\mathbb{F}_{16}$  satisfying  $\alpha^4 + \alpha + 1 = 0$ .

From the definition of the Reed-Solomon code (definition 12) the generator polynomial for this code has degree  $d - 1$ . From theorem 8 and 9 we have that the minimum distance for this code is at least  $n - k + 1$ . The dimension of this code is at most  $n - d + 1$  and thus  $d \leq n - k + 1$ . We can conclude that  $d = n - k + 1$ . We shall study in this example the  $[15, 9, 7]$  code.

Suppose that the data word, codeword and the sense word are respectively:

$$\begin{aligned} \mathbf{d} &= 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ \mathbf{c} &= 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 \\ \mathbf{v} &= 0, 0, \alpha^{11}, 0, 0, \alpha^5, 0, \alpha, 0, 0, 0, 0, 0, 0, 0 \end{aligned}$$

We first want to calculate the six ( $2t, t = 3$ ) components of the Fourier transform. I will write each of these components in the form  $\alpha^i, i = 0, \dots, 14$  not because it is easier to work with but because it is easier to write down. So we get:

$$\begin{aligned} V &= -, \alpha^{12}, 1, \alpha^{14}, \alpha^{13}, 1, \alpha^{11}, -, -, -, -, -, -, -, - \\ V &= -, S_1, S_2, S_3, S_4, S_5, S_6, -, -, -, \dots \end{aligned}$$

in which  $\alpha^{12} = \alpha^{13} + \alpha^{10} + \alpha^8, 1 = 3\alpha^{15}$ , etc.

We choose these six syndromes to work with and try to calculate the error-spectrum. These syndromes are equal to the corresponding components of  $\mathbf{E}$ . Thus we want to solve:

$$\begin{bmatrix} S_3 & S_2 & S_1 \\ S_4 & S_3 & S_2 \\ S_5 & S_4 & S_3 \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \Lambda_3 \end{bmatrix} = - \begin{bmatrix} S_4 \\ S_5 \\ S_6 \end{bmatrix}$$

Thus,

$$\begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \Lambda_3 \end{bmatrix} = \begin{bmatrix} \alpha^{14} & 1 & \alpha^{12} \\ \alpha^{13} & \alpha^{14} & 1 \\ 1 & \alpha^{13} & \alpha^{14} \end{bmatrix}^{-1} \begin{bmatrix} \alpha^{13} \\ 1 \\ \alpha^{11} \end{bmatrix} = \begin{bmatrix} \alpha^{14} \\ \alpha^{11} \\ \alpha^{14} \end{bmatrix}$$

And so we have our locator polynomial:

$$\begin{aligned} \Lambda(x) &= 1 + \alpha^{14}x + \alpha^{11}x^2 + \alpha^{14}x^3 \\ &= 1 + (1 + \alpha^3)x + (\alpha^2 + \alpha + \alpha^3)x^2 + (1 + \alpha^3)x^3 \end{aligned}$$

Then we use the recursion  $E_j = -\sum_{k=1}^t \Lambda_k E_{j-k}$  to generate the complete error spectrum:

$$\begin{aligned} E_7 &= \Lambda_1 E_6 + \Lambda_2 E_5 + \Lambda_3 = \alpha^{14} \cdot \alpha^{11} + \alpha^{11} \cdot 1 + \alpha^{14} \cdot \alpha^{13} = \alpha^5 \\ E_8 &= \alpha^{14} \cdot \alpha^5 + \alpha^{11} \cdot \alpha^{11} + \alpha^{14} \cdot 1 = 1 \\ &\vdots \end{aligned}$$

Finally we have:

$$\mathbf{E} = (\alpha^9, \alpha^{12}, 1, \alpha^{14}, \alpha^{13}, 1, \alpha^{11}, \alpha^5, 1, \alpha^6, \alpha^7, 1, \alpha^{10}, \alpha^3, 1)$$

And then we can easily compute the error vector  $\mathbf{e}$  using the inverse Fourier transform.

As an alternative of computing the error spectrum  $\mathbf{E}$  from the recursion we can also factor  $\Lambda(x)$  into:

$$\Lambda(x) = (1 + \alpha^2 x)(1 + \alpha^5 x)(1 + \alpha^7 x)$$

to find that the errors occurred at positions 2, 5 and 7 (by definition).

Now we can write down (knowing that  $E_j = \sum_{i=0}^{n-1} e_i \omega^{ij}$ ) the following matrix equation:

$$\begin{bmatrix} \alpha^2 & \alpha^5 & \alpha^7 \\ \alpha^4 & \alpha^{10} & \alpha^{14} \\ \alpha^6 & 1 & \alpha^6 \end{bmatrix} \begin{bmatrix} e_2 \\ e_5 \\ e_7 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix}$$

This matrix equation can simply be inverted to get the errors  $e_2$ ,  $e_5$  and  $e_7$ . This alternative method of the Peterson algorithm is called the *Peterson-Gorenstein-Zierler algorithm*.

### 4.1.2 Linear complexity

**Definition 16 (linear complexity)** *The linear complexity of the (finite or infinite) sequence  $V = V_0, V_1, \dots, V_{r-1}$  is the smallest value  $L$  for which a recursion*

$$V_k = - \sum_{j=1}^L \Lambda_j V_{k-j}, \quad k = L_1, \dots, r-1$$

*exists that will produce the rest of the sequence  $V$  from its first  $L$  components.*

The linear complexity of  $V$  will be denoted as  $L(V)$ .

For a finite sequence of length  $r$ ,  $L(V)$  is always well defined and is at most  $r$ .

**Definition 17** *The recursion*

$$V_k = - \sum_{j=1}^L \Lambda_j V_{k-j}$$

*of smallest  $L$  that will produce a sequence  $(V_0, V_1, \dots, V_{r-1})$  is denoted by  $(\Lambda(x), L)$ .*

It is easy to see that we can identify the following:

$$\begin{bmatrix} E_{t-1} & E_{t-2} & \cdots & E_0 \\ E_t & E_{t-1} & \cdots & E_1 \\ \vdots & & \ddots & \vdots \\ E_{2t-2} & E_{2t-3} & \cdots & E_{t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix} = - \begin{bmatrix} E_t \\ E_{t+1} \\ \vdots \\ E_{2t-1} \end{bmatrix} \longleftrightarrow$$

$$(\Lambda(x), L) \longleftrightarrow$$

$$\Lambda(x)E(x) \bmod x^{2t} = p(x) \text{ with } p(x) \text{ a polynomial with } \deg(p(x)) \leq t-1$$

From the matrix equation we can calculate  $E_t$  in terms of  $E_0, \dots, E_{t-1}, E_{t+1}$  in terms of  $E_1, \dots, E_t$ , etc. From the linearity of the Fourier transform and ?? we can conclude that  $S_j = V_j = E_j$  and thus we can identify the matrix equation with  $(\Lambda(x), L)$ .

From the convolution theorem (theorem 11) we know that  $\Lambda(x)E(x) \bmod (x^n - 1) = 0$  and thus  $\Lambda(x)E(x) \bmod x^{2t} = p(x)$  and we have identified  $(\Lambda(x), L)$  with  $\Lambda(x)E(x) \bmod x^{2t} = p(x)$ .

**Theorem 13 (Agreement theorem)** *If  $(\Lambda(x), L)$  and  $(\Lambda'(x), L')$  both produce  $V_0, V_1, \dots, V_{r-1}$  and if  $r \geq L + L'$  then*

$$-\sum_{k=1}^L \Lambda_k V_{r-k} = -\sum_{k=1}^{L'} \Lambda'_k V_{r-k}$$

**PROOF:** We have

$$V_i = -\sum_{j=1}^L \Lambda_j V_{i-j}, i = L, \dots, r-1$$

$$V_i = -\sum_{j=1}^{L'} \Lambda'_j V_{i-j}, i = L', \dots, r-1$$

Because  $r \geq L + L'$  we can write

$$V_{r-k} = -\sum_{j=1}^{L'} \Lambda'_j V_{r-k-j}, k = 1, \dots, L$$

and

$$V_{r-k} = -\sum_{i=1}^L \Lambda_i V_{r-k-i}, k = 1, \dots, L'$$

Now we have

$$-\sum_{k=1}^L \Lambda_k V_{r-k} = \sum_{k=1}^L \Lambda_k \sum_{j=1}^{L'} \Lambda'_j V_{r-k-j} = \sum_{j=1}^{L'} \Lambda'_j \sum_{k=1}^L \Lambda_k V_{r-k-j} = -\sum_{j=1}^{L'} \Lambda'_j V_{r-j}.$$

□

**Theorem 14 (Massey)** *If  $(\Lambda(x), L)$  is a linear recursion that produces  $(V_0, V_1, \dots, V_{r-2})$  but  $(\Lambda(x), L)$  does not produce  $(V_0, V_1, \dots, V_{r-2}, V_{r-1})$ , then  $L(V) \geq r - L$ .*

**PROOF:** Suppose that there exists a linear recursion  $(\Lambda'(x), L')$  that produces  $\mathbf{V}$  with  $L' < r - L$ . Then  $(\Lambda(x), L)$  and  $(\Lambda'(x), L')$  both produce  $V_0, V_1, \dots, V_{r-2}$  and  $L' + L \leq r - 1$ . By theorem 13, both must produce the same value at iteration  $r - 1$ , contrary to the assumption of the theorem. □

If the first 8 components of the Fibonacci sequence are periodically repeated then we have:

$$1, 1, 2, 3, 5, 8, 13, 21, 1, 1, 2, 3, 5, 8, 13, 21, 1, 1, \dots$$

The Fibonacci sequence is produced by the recursion  $\Lambda(x), L = (1 - x - x^2, 2)$ . The linear complexity of this sequence is at least  $9 - 2 = 7$ .

## 4.2 The algorithms

The Peterson algorithm, treated in section 4.1 and 4.1.1, is based on the solution of the matrix equation

$$\begin{bmatrix} E_{\nu-1} & E_{\nu-2} & \cdots & E_0 \\ E_{\nu} & E_{\nu-1} & \cdots & E_1 \\ \vdots & & \ddots & \vdots \\ E_{2\nu-2} & E_{2\nu-3} & \cdots & E_{\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_{\nu} \end{bmatrix} = - \begin{bmatrix} E_{\nu} \\ E_{\nu+1} \\ \vdots \\ E_{2\nu-1} \end{bmatrix}$$

For small  $\nu$  a matrix inversion is easy to do. The number of computations necessary to invert a  $\nu$  by  $\nu$  matrix is proportional to  $\nu^3$ . But for codes that correct a large number of errors one needs a more efficient method, the matrix inversion requires too many computations. E. R. Berlekamp found such a method. This method relies on the structure of the matrix.

First we want to consider another decoding algorithm, *the Sugiyama algorithm*. The Sugiyama algorithm is, as we shall see, based on the Euclidean algorithm.

First we introduce a new polynomial that the Sugiyama algorithm, treated in the next section, will need.

**Definition 18 (error-evaluator polynomial)** *The error-evaluator polynomial  $\Gamma(x)$  is the unique polynomial of degree  $< \nu$  which satisfies*

$$\Gamma(x) = \Lambda(x)E(x) \pmod{x^{\nu}}$$

The Sugiyama algorithm computes both the error-locator polynomial and the error-evaluator polynomial. This algorithm can be improved by eliminating the need to compute  $\Gamma(x)$ . The Berlekamp-Massey algorithm, which will be presented in section 4.2.3, does not need to calculate  $\Gamma(x)$ .

**Lemma 1** *The error-evaluator polynomial  $\Gamma(x)$  satisfies*

$$-\Gamma(x) = \frac{\Lambda(x)E(x)}{1 - x^n}$$

**PROOF:** Suppose

$$\frac{\Lambda(x)E(x)}{1 - x^n} = g(x)$$

Then we have  $\Lambda(x)E(x) = g(x)(x^n - 1) = g(x)x^n - g(x)$ . We have also from the definition of  $\Gamma(x)$ :

$$\Lambda(x)E(x) = \Gamma(x) \pmod{x^{\nu}}$$

This becomes

$$g(x)x^n - g(x) = \Gamma(x) \bmod x^\nu.$$

$$\text{Since } g(x)x^n = 0 \bmod x^\nu \text{ thus } -g(x) = \Gamma(x) \bmod x^\nu$$

The degree of  $\Gamma(x)$  is at most  $\nu - 1$ . If the degree of  $g(x)$  is also at most  $\nu - 1$  then  $-g(x) = \Gamma(x)$ .

$$\begin{aligned} n + \deg(g(x)) &= \deg(\Lambda(x)E(x)) \\ \deg(g(x)) &= \deg(\Lambda(x)) + \deg(E(x)) - n \\ &\leq \nu + n - 1 - n = \nu - 1 \end{aligned}$$

□

**Lemma 2** *The error-evaluator polynomial satisfies*

$$\Gamma(x) = \Lambda(x)E(x) \bmod x^r \text{ for any } r \text{ between } \nu \text{ and } n.$$

**PROOF:** We know that  $\Lambda(x)E(x) = 0 \pmod{x^n - 1}$ . This can be written as  $\Lambda(x)E(x) = -\Gamma(x)(x^n - 1) = \Gamma(x) - x^n\Gamma(x)$ , hence modulo  $x^r$  for any  $\nu \leq r \leq n$ , we have

$$\Lambda(x)E(x) = \Gamma(x) \bmod x^r$$

□

In many books about locator decoding (for example in [?] and [?]) and in many articles about this subject one finds another definition of the error-evaluator polynomial:  $\Gamma(x) = \sum_{j=1}^{\nu} e_{j_i} \alpha^{i_j} \prod_{\ell \neq j} (1 - \alpha^{i_\ell})$ . We want to prove that this definition is the same as the one given above.

**Theorem 15** *The error-evaluator polynomial can be written*

$$\Gamma(x) = \sum_{j=1}^{\nu} e_{j_i} \alpha^{i_j} \prod_{\ell \neq j} (1 - \alpha^{i_\ell})$$

with  $e_{i_j}$  and  $\alpha^{i_j}$  as in section 4.1.

**PROOF:** We have

$$\Gamma(x) = E(x)\Lambda(x) \bmod x^{2t}$$

with

$$E(x) = \sum_{k=1}^{2t} E_k x^{k-1} \text{ and } \Lambda(x) = \prod_{\ell=1}^{\nu} (1 - \alpha^{i_\ell} x)$$

We know that  $E_k = \sum_{j=0}^{n-1} \alpha^{ij} e_{i_j}$ , thus we can write

$$E(x) = \sum_{k=1}^{2t} \sum_{j=1}^{\nu} e_{i_j} \alpha^{ij} x^{k-1}.$$

$$\begin{aligned} \Gamma(x) &= E(x)\Lambda(x) \bmod x^{2t} \\ &= \left[ \sum_{k=1}^{2t} \sum_{j=1}^{\nu} e_{i_j} \alpha^{ij} x^{k-1} \right] \prod_{\ell=1}^{\nu} (1 - \alpha^{i_\ell} x) \bmod x^{2t} \\ &= \sum_{j=1}^{\nu} e_{i_j} \alpha^{ij} \left[ (1 - \alpha^{ij}) \sum_{k=1}^{2t} (\alpha^{ij} x)^{k-1} \right] \prod_{\ell \neq j} (1 - \alpha^{i_\ell} x) \bmod x^{2t} \end{aligned}$$

Note that

$$(1 - \alpha^{ij}) \sum_{k=1}^{2t} (\alpha^{ij} x)^{k-1} = (1 - \alpha^{2ti_j} x^{2t})$$

thus we have

$$\begin{aligned} \Gamma(x) &= \sum_{j=1}^{\nu} e_{i_j} \alpha^{ij} (1 - \alpha^{2ti_j} x^{2t}) \prod_{\ell \neq j} (1 - \alpha^{i_\ell} x) \bmod x^{2t} \\ &= \sum_{j=1}^{\nu} e_{i_j} \alpha^{ij} \prod_{\ell \neq j} (1 - \alpha^{i_\ell} x) \end{aligned}$$

□

### 4.2.1 Sugiyama algorithm

The Sugiyama algorithm inverts any system of equations in the field  $\mathbb{F}$  of the form

$$\begin{bmatrix} E_{t-1} & E_{t-2} & \cdots & E_0 \\ E_t & E_{t-1} & \cdots & E_1 \\ \vdots & & \ddots & \vdots \\ E_{2t-2} & E_{2t-3} & \cdots & E_{t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix} = - \begin{bmatrix} E_t \\ E_{t+1} \\ \vdots \\ E_{2t-1} \end{bmatrix}$$

This is the central problem of locator decoding. As we have seen before we have

$$E_j = - \sum_{i=1}^t \Lambda_i E_{j-i}, \quad j = t, \dots, 2t-1.$$

Recall the notations

$$\Lambda_0 = 1, \quad \Lambda(x) = \sum_{j=0}^t \Lambda_j x^j \quad \text{and} \quad E(x) = \sum_{j=0}^{2t-1} E_j x^j$$

We have now that the coefficients of the polynomial product  $\Lambda(x)E(x)$  are equal to zero for  $j = t, \dots, 2t - 1$ .

Solving the original matrix equation is the same as solving the polynomial equation

$$\Lambda(x)E(x) = \Gamma(x) \pmod{x^{2t}}$$

for  $\Lambda(x)$  of degree at most  $t$  and  $\Gamma(x)$  of degree at most  $t - 1$ . With  $E(x)$  as input, the Sugiyama algorithm solves this polynomial equation for  $\Lambda(x)$  and  $\Gamma(x)$ .

Notice that the substeps of the Sugiyama algorithm are the same as the substeps of the Euclidean algorithm for polynomials. We begin initializing  $a^{(0)}(x) = x^{2t}$  and  $b^{(0)} = E(x)$ . At iteration  $r$  let

$$a^{(r-1)}(x) = Q^{(r)}(x)b^{(r-1)}(x) + b^{(r)}(x), \quad \text{with } \deg(b^{(r)}(x)) < \deg(b^{(r-1)}(x))$$

Define  $a^{(r)}(x) = b^{(r-1)}(x)$ . In matrix form

$$\begin{bmatrix} a^{(r)}(x) \\ b^{(r)}(x) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} \begin{bmatrix} a^{(r-1)}(x) \\ b^{(r-1)}(x) \end{bmatrix}$$

Also define the matrix  $A^{(r)}(x)$  inductively by:

$$A^{(r)}(x) = \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix} A^{(r-1)}(x)$$

and

$$A^{(0)}(x) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

For every  $r \geq 0$  we have

$$\begin{bmatrix} a^{(r)}(x) \\ b^{(r)}(x) \end{bmatrix} = \begin{bmatrix} A_{11}^{(r)}(x) & A_{12}^{(r)}(x) \\ A_{21}^{(r)}(x) & A_{22}^{(r)}(x) \end{bmatrix} \begin{bmatrix} x^{2t} \\ E(x) \end{bmatrix}$$

The equation

$$b^{(r)}(x) = A_{22}^{(r)}(x)E(x) \pmod{x^{2t}}$$

which is of the desired form. To solve the problem we need to find an  $r$  for which  $\deg(A_{22}^{(r)}(x)) \leq t$  and  $\deg(b^{(r)}(x)) \leq t - 1$ . We will satisfy the requirements by choosing  $r'$  as the value of  $r$  satisfying  $\deg(b^{(r'-1)}(x)) \geq t$  and  $\deg(b^{(r')}(x)) \leq t - 1$ .

We have a unique value  $r'$  because  $\deg(b^{(0)}(x)) = 2t - 1$ , and the degree of  $b^{(r)}(x)$  is strictly decreasing as  $r$  becomes larger.

It is easy to see that as  $r$  becomes larger, the degree of  $A_{22}^{(r)}(x)$  becomes larger: For every  $r$  we have

$$A_{22}^{(r)}(x) = A_{12}^{(r-1)}(x) + Q^{(r)}(x)A_{22}^{(r-1)}(x) \text{ by definition.}$$

Because  $A_{12}^{(r-1)}(x) = A_{22}^{(r-2)}(x)$  and  $\deg(A_{12}^{(r-1)}(x)) < \deg(A_{22}^{(r-1)}(x))$  we have

$$\deg(A_{22}^{(r-2)}(x)) < \deg(A_{22}^{(r-1)}(x))$$

We still only need to show that

$$\deg(A_{22}^{(r')}(x)) \leq t$$

We know that

$$A^{(r')}(x) = \prod_{r=1}^{r'} \begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix}$$

Since every matrix  $\begin{bmatrix} 0 & 1 \\ 1 & -Q^{(r)}(x) \end{bmatrix}$  has determinant equal to  $-1$ , the determinant of  $A^{(r')}(x)$  is  $(-1)^{r'}$ . Therefore, the earlier matrix equation can be inverted to give

$$\begin{bmatrix} x^{2t} \\ E(x) \end{bmatrix} = (-1)^r \begin{bmatrix} A_{22}^{(r)}(x) & -A_{12}^{(r)}(x) \\ -A_{21}^{(r)}(x) & A_{11}^{(r)}(x) \end{bmatrix} \begin{bmatrix} a^{(r)}(x) \\ b^{(r)}(x) \end{bmatrix}$$

It is clear that  $\deg(A_{22}^{(r)}(x)) > \deg(A_{12}^{(r)}(x))$  and  $\deg(a^{(r)}(x)) \geq \deg(b^{(r)}(x))$  and thus

$$\deg(x^{2t}) = \deg(A_{22}^{(r)}(x)) + \deg(a^{(r)}(x))$$

this becomes

$$\deg(A_{22}^{(r)}(x)) = \deg(x^{2t}) - \deg(a^{(r)}(x)) \leq 2t - t = t$$

because  $\deg(a^{(r)}(x)) = \deg(b^{(r-1)}(x)) \geq t$ .

And this proves the algorithm.

### 4.2.2 The Sugiyama algorithm in Maple

We are going to make a procedure in Maple that computes the locator polynomial using the Sugiyama algorithm. The procedure is written strictly following the algorithm. For a binary cyclic code we have the Maple code:

```

sugiyama:=proc (E)
local a, b, x, A, t, new_a, Q;
global loc;
  x := indets(E)[]; #x need not to be the variable
  t := 1/2*degree(E,x)+1/2;
      #maximal errors one can correct
  a := x^(2*t);
  b := E;
  A := [[1, 0], [0, 1]];
  while t <= degree(b,x) do
    Q := `mod`(Quo(a,b,x),2); #binary code
    new_a := b;
    b := `mod`(simplify(a-Q*b),2);
    a := new_a;
    A := evalm(`&*`([[0, 1], [1, -Q]],A))
  od;
  L:=collect(`mod`(simplify(A[2,2]),2),x);
      #the locator polynomial
  loc:=collect(`mod`(simplify(L/subs(x=0,L)),2),x);
      #the locator polynomial with L[1]=0
end:

```

To check this algorithm and as an example we are going to look again at the [15, 9, 7]-Reed-Solomon code, with the same vectors as in the Peterson algorithm. The word we have received is again

$$\mathbf{v} = 0, 0, \alpha^{11}, 0, 0, \alpha^5, 0, \alpha, 0, 0, 0, 0, 0, 0, 0$$

As we have seen the locator polynomial for this word is

$$\Lambda(x) = 1 + (1 + \alpha^3)x + (\alpha + \alpha^2 + \alpha^3)x^2 + (1 + \alpha^3)x^3$$

Now we want to see if we can get this polynomial using the procedure *sugiyama* written in Maple. The first six components of the syndrome spectrum are again

$$\alpha^9, \alpha^{12}, 1, \alpha^{14}, \alpha^{13}, 1$$

If we identify this with the polynomial<sup>2</sup>  $\alpha^9 + \alpha^{12}t + t^2 + \alpha^{14}t^3 + \alpha^{13}t^4 + t^5$  we can use the sugiyama procedure

```

> alias(al=RootOf(x^4+x+1));
> sugiyama(al^9+al^12*t+t^2+al^14*t^3+al^13*t^4+t^5);

```

<sup>2</sup>note that  $x$  need not be the variable

And we get as output

$$(1 + al^3)t^3 + (al^2 + al + al^3)t^2 + (1 + al^3)t + 1$$

what is indeed the locator polynomial as calculated before.

### 4.2.3 Berlekamp-Massey algorithm

Just like the Sugiyama algorithm, the Berlekamp-Massey algorithm inverts a system of equations in the field  $\mathbb{F}$  of the form

$$\begin{bmatrix} E_{t-1} & E_{t-2} & \cdots & E_0 \\ E_t & E_{t-1} & \cdots & E_1 \\ \vdots & & \ddots & \vdots \\ E_{2t-2} & E_{2t-3} & \cdots & E_{t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix} = - \begin{bmatrix} E_t \\ E_{t+1} \\ \vdots \\ E_{2t-1} \end{bmatrix}$$

More precisely, given  $E_0, E_1, \dots, E_{2t-1}$  the algorithm will find the recursion

$$E_j = - \sum_{k=1}^t \Lambda_k E_{j-k} \quad j = t, \dots, 2t-1$$

for which  $t$  is smallest.

Suppose  $\Lambda$  is known, and we find it has degree  $\nu$ . Then the first row of the above matrix, with  $t = \nu$  defines  $E_\nu$  in terms of  $E_0, \dots, E_{\nu-1}$ . The second row defines  $E_{\nu+1}$  in terms of  $E_1, \dots, E_\nu$  and so forth.

We wish to find  $\Lambda(x)$  of smallest degree that produces the above sequence. The smallest possible degree of  $\Lambda(x)$  will be  $\nu$ , and there is only one locator polynomial of degree  $\nu$  because only then the  $\nu$  by  $\nu$  matrix is invertible as we have seen before.

The Berlekamp-Massey algorithm is an iterative procedure for finding the shortest recursion for producing successive terms of the sequence  $E$ .

At the  $r$ -th step, the algorithm will find the shortest recursion  $(\Lambda^{(r)}(x), L_r)$ , that produces the first  $r$  terms of  $E$ . Thus,

$$E_j = - \sum_{k=1}^{L_r} \Lambda_k^{(r)} E_{j-k} \quad j = L_r, \dots, r-1.$$

Given  $(\Lambda^{(r)}(x), L_r)$ , the shortest recursion that produces  $(E_0, \dots, E_{r-1})$ , let  $\Lambda_0^{(r)} = 1$  and define

$$\Delta_r = E_r - \left( - \sum_{k=1}^{L_r} \Lambda_k^{(r)} E_{r-k} \right) = \sum_{k=0}^{L_r} \Lambda_k^{(r)} E_{r-k}.$$

It is easy to see that if  $\Delta_r$  is zero then

$$(\Lambda^{(r+1)}(x), L_{r+1}) = (\Lambda^{(r)}(x), L_r)$$

is the shortest recursion that produces  $(E_0, \dots, E_r)$ . From now on we will denote the sequence  $(E_0, \dots, E_{r-1})$  as  $\mathbf{E}^r$ .

Now we have enough information to present the Berlekamp-Massey algorithm:

Let  $S_1, \dots, S_{2t}$  be known. We begin initializing  $\Lambda^{(0)}(x) = 1$ ,  $B^{(0)}(x) = 1$  and  $L_0 = 0$ . The following set of recursive equations is used to compute  $\Lambda(x)$ :

$$\begin{aligned} \Delta_r &= \sum_{j=0}^d \Lambda_j^{(r-1)} S_{r-j}, \quad d = \deg(\Lambda^{(r-1)}(x)) \\ L_r &= \delta_r(r - L_{r-1}) + (1 - \delta_r)L_{r-1} \\ \begin{bmatrix} \Lambda^{(r)}(x) \\ B^{(r)}(x) \end{bmatrix} &= \begin{bmatrix} 1 & -\Delta_r x \\ \Delta_r^{-1} \delta_r & (1 - \delta_r)x \end{bmatrix} \begin{bmatrix} \Lambda^{(r-1)}(x) \\ B^{(r-1)}(x) \end{bmatrix} \end{aligned}$$

$r = 1, \dots, 2t$  where  $\delta_r = 1$  if both  $\Delta_r \neq 0$  and  $2L_{r-1} \leq r - 1$  and otherwise  $\delta_r = 0$ .

Then  $\Lambda^{(2t)}$  is the polynomial of smallest degree with the properties

$$\begin{aligned} \Lambda_0^{(2t)} &= 1 \\ S_r + \sum_{j=1}^{n-1} \Lambda_j^{(2t)} S_{r-j} &= 0, \quad r = L_{2t}, \dots, 2t - 1. \end{aligned}$$

Note that many terms in the sum  $\Delta_r$  are equal to zero if we write the sum from 1 to  $n - 1$ . That is the reason why the sum goes only to the degree of  $\Lambda^{(r-1)}(x)$ .

We are going to prove the algorithm using the following two lemmas.

**Lemma 3** *If the recursion  $(\Lambda^{(r)}, L_r)$  produces  $\mathbf{E}^{r-1}$  but not  $\mathbf{E}^r$ , and the recursion  $(\Lambda^{(m)}, L_m)$  with  $m < r$  produces  $\mathbf{E}^{m-1}$  but not  $\mathbf{E}^m$ , then the recursion*

$$(\Lambda(x), L) = (\Lambda^{(r)} - \Delta_m^{-1} \Delta_r x^{r-m} \Lambda^{(m)}(x), \max[L_r, L_m + r - m])$$

*produces  $\mathbf{E}^r$ .*

**PROOF** Let  $E^{(r)}(x)$  and  $E^{(m)}(x)$  be the polynomials formed in the obvious way from  $\mathbf{E}^r$  and  $\mathbf{E}^m$ .

We can write

$$\begin{aligned} \Lambda^{(r)}(x) E^{(r)}(x) &= \sum_{k=0}^{L_r} \Lambda_k x^k \sum_{i=0}^{r-1} E_i x^i \\ &= \sum_{j=k}^{k+r-1} \sum_{k=0}^{L_r} \Lambda_k E_{j-k} x^j, \quad \text{with } j = i + k \end{aligned}$$

and this can be written in terms of monomials as follows

$$\Lambda^{(r)}(x)E^{(r)}(x) = p^{(r)}(x) + \Delta_r x^r + x^{r+1}g^{(r+1)}(x)$$

with  $\deg(p^{(r)}(x)) < L_r$ ,  $\Delta_r$  is nonzero and the monomial  $g^{(r+1)}(x)$  is of no interest.

In the same way we can write

$$\Lambda^{(m)}(x)E^{(m)}(x) = p^{(m)}(x) + \Lambda_m x^m + x^{m+1}g^{(m+1)}(x)$$

Extending  $E^{(m)}(x)$  to  $E^{(r)}(x)$  in the product  $\Lambda^{(m)}(x)E^{(m)}(x)$  simply introduces new monomials of degree larger than  $m$ , so (note that  $g^{(m+1)}$  is a different one than above)

$$\Lambda^{(m)}(x)E^{(r)}(x) = p^{(m)}(x) + \Lambda_m x^m + x^{m+1}g^{(m+1)}(x), \deg(p^{(m)}) < L_m.$$

If we subtract  $\Delta_r x^r \Lambda^{(m)} E^{(r)}$  from  $\Delta_m x^m \Lambda^{(r)} E^{(r)}$  we get

$$\begin{aligned} [\Lambda^{(r)}(x) - \frac{\Delta_r}{\Delta_m} x^{r-m} \Lambda^{(m)}(x)]E^{(r)}(x) = \\ [p^{(r)}(x) - \frac{\Delta_r}{\Delta_m} x^{r-m} p^{(m)}(x)] + \\ [\Delta_r - \frac{\Delta_r}{\Delta_m} \Delta_m]x^r + x^{r+1}[\dots] \end{aligned}$$

And this has the form

$$\begin{aligned} \Lambda(x)E^{(r)}(x) &= p(x) + x^{r+1}g(x) \\ \deg(\Lambda(x)) &\leq \max[L_r, L_m + n - m] \\ \deg(p(x)) &< \max[L_r, L_m + n - m] \end{aligned}$$

as was to be proved. □

In general, there will be more than one value of  $m < r$  for which  $\Delta \neq 0$ . The preceding lemma does not tell us which to choose. The following lemma tells us that we have to choose that  $m$  as the most recent iteration for which  $\Delta_m$  is nonzero. Then the recursion is of shortest length.

**Lemma 4** *Suppose that  $L(\mathbf{E}^{n-1}) = L$ . If  $(\Lambda(x), L)$  produces  $\mathbf{E}^{n-1}$  but not  $\mathbf{E}^n$ , then  $L(\mathbf{E}^n) = \max[L, n - L]$ .*

**PROOF** we know from Massey's theorem that

$$L(\mathbf{E}^n) \geq \max[L, n - L].$$

Thus it suffices to prove that

$$L(\mathbf{E}^n) \leq \max[L, n - L].$$

Let  $\mathbf{0}^{n-1}$  denote a sequence of  $n-1$  zeros, then we have two possibilities:  $\mathbf{E}^{n-1} = \mathbf{0}^{n-1}$  or  $\mathbf{E}^{n-1} \neq \mathbf{0}^{n-1}$ .

- $\mathbf{E}^{n-1} = \mathbf{0}^{n-1}$  and  $\mathbf{E}^n = \mathbf{0}^{n-1}, \Delta$  where  $\Delta \neq 0$ . Then  $L(\mathbf{E}^n) \leq \max[L, n - L] = n$ .
- $\mathbf{E}^{n-1} \neq \mathbf{0}^{n-1}$ . The proof is by induction. Let  $m$  be such that

$$L(\mathbf{E}^{m-1}) < L(\mathbf{E}^m) = L(\mathbf{E}^{n-1}).$$

The induction hypothesis is that

$$L(\mathbf{E}^{n-1}) = L(\mathbf{E}^m) = m - L(\mathbf{E}^{m-1}) = m - L_{m-1}.$$

By Lemma 1 we have

$$\begin{aligned} L(\mathbf{E}^n) &\leq \max[L, L_{m-1} + n - m] \\ &= \max[L(\mathbf{E}^{n-1}), n - L] \\ &= \max[n - L, L] \end{aligned}$$

□

Notice that the matrix calculation requires at most  $2t$  multiplications per iteration, and that the calculation of  $\Delta_r$  requires no more than  $t$  multiplications per iteration. There are  $2t$  iterations, and thus at most  $6t^2$  multiplications. Thus using this algorithm will usually be much better than using a matrix inversion, which requires on the order of  $t^3$  operations.

#### 4.2.4 The Berlekamp-Massey algorithm in Maple

We want to make a procedure in Maple for the Berlekamp-Massey algorithm too. The source code of the procedure can be:

```

bm := proc (A)
local B, L, l, t, r, i, x, Dt, dt;
global S, loc;
  x := indets(A)[1]; #x need not to be the variable
  t := 1/2*degree(A,x); #maximal errors one can correct
  B[0] := 1;
  L[0] := 1;
  l[0] := 0;
  r := 1;
  for i to degree(A,x) do
    S[i] := coeff(A,x^i) #the value of S from the polynomial
  od;
  while r <> 2*t+1 do #the loop
    Dt[r] := `mod`(simplify(add(
      coeff(x*L[r-1],x^(j+1))*S[r-j],j = 0 .. l[r-1])),2);
    if Dt[r] <> 0 and 2*l[r-1] <= r-1 then dt[r] := 1
    else dt[r] := 0
    fi;
    l[r] := dt[r]*(r-l[r-1])+(1-dt[r])*l[r-1];
    L[r] := `mod`(simplify(L[r-1]-Dt[r]*x*B[r-1]),2);
    if Dt[r] <> 0 then B[r] := `mod`(simplify(
      dt[r]*L[r-1]/Dt[r]+(1-dt[r])*x*B[r-1]),2)
    fi;
    r := r+1
  od;
  loc := collect(`mod`(simplify(L[2*t]),2),x) #the locator polynomial
end:

```

With  $L$  is  $\Lambda(x)$ ,  $l$  is  $L$ ,  $Dt$  is  $\Delta$ ,  $dt$  is  $\delta$  and  $loc$  *the* locator polynomial. And the rest of the remaining variables are obvious.

To check this algorithm and as an example again we are going to look at the same code and at the same received word as in the Sugiyama algorithm.

We have already seen a few times that the locator polynomial for this received word

$$\Lambda(x) = 1 + (1 + \alpha^3)x + (\alpha^2 + \alpha + \alpha^3)x^2 + (1 + \alpha^3)x^3$$

is. We want to check this with Maple using the above procedure. We already know that

$$\begin{aligned} S_1 &= \alpha^{12} \\ S_2 &= 1 \\ S_3 &= \alpha^{14} \\ S_4 &= \alpha^{13} \\ S_5 &= 1 \\ S_6 &= \alpha^{11} \end{aligned}$$

If we identify this with the polynomial  $\alpha^{12}x + x^2 + \alpha^{14}x^3 + \alpha^{13}x^4 + x^5 + \alpha^{11}x^6$  we can use our procedure

```
> bm(a1^12*x+x^2+a1^14*x^3+a1^13*x^4+x^5+a1^11*x^6);
```

Where we get as output, as expected, the locator polynomial

$$\Lambda(x) = (1 + \alpha^3)x^3 + (\alpha^2 + \alpha + \alpha^3)x^2 + (1 + \alpha^3)x + 1.$$

## 4.3 Forney

The Forney algorithm computes the error vector  $\mathbf{e}$  from the error-evaluator polynomial and the error-locator polynomial.

**Theorem 16 (Forney)** *Suppose that  $\Lambda(x)$  has degree  $\nu$ . The components of the error vector  $\mathbf{e}$  can be computed as follows*

$$e_i = \begin{cases} 0 & \text{if } \Lambda(\omega^{-i}) \neq 0 \\ -\frac{\Gamma(\omega^{-i})}{\omega^{-i}\Lambda'(\omega^{-i})} & \text{if } \Lambda(\omega^{-i}) = 0 \end{cases}$$

where  $\Lambda'(x) = \sum_{j=1}^{\nu} j\Lambda_jx^{j-1}$  and  $\Gamma(x) = \Lambda(x)E(x) \bmod x^{\nu}$ .

**PROOF:** Recall that

$$\Lambda(x)E(x) = -\Gamma(x)(x^n - 1)$$

for some polynomial  $\Gamma(x)$ . The derivative of this equation is

$$\Lambda'(x)E(x) + \Lambda(x)E'(x) = \Gamma'(x) - nx^{n-1}\Gamma(x) - x^n\Gamma'(x)$$

Take  $x = \omega^{-i}$ , noting that  $\omega^{-n} = 1$ , this gives

$$\Lambda'(\omega^{-i})E(\omega^{-i}) + \Lambda(x)E'(\omega^{-i}) = -n\omega^i\Gamma(\omega^{-i}) \quad (4.1)$$

The error component  $e_i$  is nonzero only if  $\Lambda(\omega^{-i}) = 0$  and in that case (using the inverse Fourier transform)

$$e_i = \frac{1}{n} \sum_{j=0}^{n-1} E_j \omega^{-ij} = \frac{1}{n} E(\omega^{-i}) \quad (4.2)$$

Using that equation 4.1 can be rewritten as

$$E(\omega^{-i}) = \frac{-n\omega^{-i}\Gamma(\omega^{-i}) - \Lambda(\omega^{-i})E'(\omega^{-i})}{\Lambda'(\omega^{-i})}$$

equation 4.2 can be written as

$$e_i = \frac{-n\omega^{-i}\Gamma(\omega^{-i}) - \Lambda(\omega^{-i})E'(\omega^{-i})}{n\Lambda'(\omega^{-i})}$$

And the theorem is proved because  $\Lambda(\omega^{-i}) = 0$ . □

**Remark:** Since the Berlekamp-Massey algorithm and the Sugiyama algorithm only compute the locator polynomial, we need the theorem above to calculate the magnitude of the errors made. In the next chapter I will give an example of decoding a code. I will use the Sugiyama algorithm to compute the locator polynomial. If we have the locator polynomial we can easily find on which component an error have been made. Then I will use the theorem above to calculate the magnitude of the error.

# Chapter 5

## Example

In this chapter I want to present an example that contains all the theory of locator decoding we have seen so far. We take an arbitrary word, the data word, that we wish to send through a noisy channel. First we encode the data word to get the code word. We put errors into this code word, so that we can treat it as to be the received word and can use the algorithms given before.

### 5.1 Example using the Sugiyama algorithm

We are going to study again the  $[15, 9, 7]$ -Reed-Solomon code over  $\mathbb{F}_{16}$ . From definition 12 of the Reed-Solomon code we know that the generator polynomial for this code has the form

$$\begin{aligned} g(x) &= \prod_{i=1}^{7-1} (x - \alpha^i) \\ &= (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6) \\ &= x^6 + (1 + \alpha + \alpha^2)x^5 + (1 + \alpha^3)x^4 + (1 + \alpha)x^3 \\ &\quad + (\alpha^2 + \alpha^3)x^2 + (\alpha + \alpha^3)x + \alpha^2 + \alpha^3 \end{aligned}$$

with  $\alpha$  a generator of  $\mathbb{F}_{16}$ .

The maple code to calculate  $g(x)$  is the following:

```
> product(x-alpha^i, i=1..6);
```

Suppose we want to send  $(\alpha, \alpha^2, \alpha, \alpha^2, \alpha^2, \alpha^3, \alpha, \alpha^6, \alpha)$  across a noisy channel. The polynomial corresponding to this vector is

$$d(x) = \alpha + \alpha^2x + \alpha x^2 + \alpha^2x^3 + \alpha^2x^4 + \alpha^3x^5 + \alpha x^6 + \alpha^6x^7 + \alpha x^8.$$

To encode it, we multiply this by the generator polynomial  $g(x)$  to find our code word

$$c = (1 + \alpha + \alpha^3, \alpha, \alpha + \alpha^2 + \alpha^3, 1, 1 + \alpha^2 + \alpha^3, 1 + \alpha + \alpha^3, 1, \alpha + \alpha^2, \\ 1 + \alpha^2, 1, \alpha^2, \alpha + \alpha^2, 1, \alpha, \alpha)$$

In maple we calculate  $c(x)$

```
> c:=collect(simplify(d*g) mod 2, x);
```

We put three arbitrary errors in the code word, for example on the first, third and twelfth place, we get:

$$v = (1, \alpha, 1, 1, 1 + \alpha^2 + \alpha^3, 1 + \alpha + \alpha^3, 1, \alpha + \alpha^2, 1 + \alpha^2, 1, \alpha^2, 1, 1, \alpha, \alpha)$$

Suppose that this word with the errors is the received word, the sense word. We are going to look if we can find the errors and their magnitude.

Therefore we first compute the syndromes  $S_0, \dots, S_5$  with the following procedure written in Maple:

```
syndroom:=proc(v)
  local k, m, n, v2:
  global S, syn;
  v2:=v*x; for k from 0 to 5 do
    S[k]:=simplify(add(coeff(v2,x^m)*a1^(k*(m-1)),m=1..16))
    mod 2;print(S[k]);
  od:
  syn:=collect(simplify(add(S[n]*x^n, n=0..5)) mod 2,x);
end:
```

We get

$$\begin{aligned} S_0 &= \alpha \\ S_1 &= 1 + \alpha + \alpha^2 + \alpha^3 \\ S_2 &= \alpha^2 + \alpha^3 \\ S_3 &= 1 + \alpha + \alpha^2 + \alpha^3 \\ S_4 &= \alpha + \alpha^2 \\ S_5 &= 1 + \alpha + \alpha^2 \end{aligned}$$

We identify this with the polynomial

$$\alpha + (1 + \alpha + \alpha^2 + \alpha^3)x + (\alpha^2 + \alpha^3)x^2 + (1 + \alpha + \alpha^2 + \alpha^3)x^3 + \\ (\alpha + \alpha^2)x^4 + (1 + \alpha + \alpha^2)x^5 \quad (5.1)$$

to be able to use the procedure sugiyama given in section 4.2.2. This procedure with input 5.1 finds the locator polynomial

$$\Lambda(x) = 1 + (1 + \alpha + \alpha^3)x + (1 + \alpha + \alpha^2)x^2 + (1 + \alpha^2 + \alpha^3)x^3 \quad (5.2)$$

To find on which components errors have been made we use a simple procedure written in Maple that computes for which  $i$ ,  $\Lambda(\alpha^{-i})$  for  $i = 0, \dots, 14$  is zero:

```
errors:=proc(loc, n)
local i;
global al;
  for i from 0 to n-2 do
    if (simplify(subs(x=al^(-i),loc)) mod 2 = 0)
      then print(i) fi;
    od;
end;
```

With `loc` the locator polynomial( 5.2) and  $n = 16$  as input we find that the components for  $i = 0, 2, 11$  are in error.

We can now use the Forney algorithm explained in section 4.3, to find the error magnitude:

```
forney:=- (subs(x=al^(-i), gam)/(al^(-i)*subs(x=al^(-i), dloc)));
```

For  $i = 0, i = 2$  and  $i = 11$  we have

$$e_0 = -\frac{\Gamma(\alpha^{-0})}{\alpha^{-0}\Lambda'(\alpha^{-0})} = \alpha + \alpha^3$$

$$e_2 = -\frac{\Gamma(\alpha^{-2})}{\alpha^{-2}\Lambda'(\alpha^{-2})} = 1 + \alpha + \alpha^2 + \alpha^3$$

$$e_{11} = -\frac{\Gamma(\alpha^{11})}{\alpha^{-11}\Lambda'(\alpha^{-11})} = 1 + \alpha + \alpha^2$$

with  $\Lambda'(x) = 1 + \alpha + \alpha^3 + (1 + \alpha^2 + \alpha^3)x^2$ , the derivative of  $\Lambda(x)$ .

If we subtract  $e_i$  from  $v_i$  (for those  $i$  that are in error)

$$v_0 - e_0 = 1 - \alpha + \alpha^3 = 1 + \alpha + \alpha^3$$

$$v_2 - e_2 = 1 - 1 + \alpha + \alpha^2 + \alpha^3 = \alpha + \alpha^2 + \alpha^3$$

$$v_{11} - e_{11} = 1 - 1 + \alpha + \alpha^2 = \alpha + \alpha^2$$

we see that these are indeed the components of our code word.

The last thing to do to get the original word (the data word) is to divide the corrected sense word by the generator polynomial.

In maple

```
> collect(simplify(Quo(c,g)) mod 2, x);
```

## 5.2 Conclusion

We have seen that BCH codes can easily be decoded using locator decoding. It is a very simple technique that turns out to be very important. Besides the algorithms given in this Master's Thesis there are many more algorithms based on locator decoding. All of these algorithms have one common feature: their construction can be traced back to finite fields.

Without these techniques there would be many things in daily life impossible. Take for example a CD player: without locator decoding the requirements of the quality of a CD would be so high that production would be impossible. The error-correcting code<sup>1</sup> in a CD player is based on two Reed-Solomon codes: a  $[32, 28, 5]$ -code over  $\mathbb{F}_{2^8}$  and a  $[28, 24, 5]$ -code also over  $\mathbb{F}_{2^8}$ . They can both, with a version of the Berlekamp-Massey algorithm, correct two errors. If you have a CD full of tiny scratches you will see that it will reproduce the music on it without errors; the code has corrected all the errors. You can imagine how good this error-correcting code has to be and you have seen how relatively simple Reed-Solomon codes are.

---

<sup>1</sup>This code is called a CIRC-code, a Cross Interleaved Reed-Solomon Code