Improved Decoding Algorithms for Reed-Solomon Codes

Tesi di Laurea Magistrale
3 dicembre 2012

Candidato:
Irene Giacomelli

Relatori:
Prof.ssa Patrizia Gianni  Prof. Barry Trager

Controrelatore:
Prof. Massimo Caboara

Anno Accademico 2011/2012
Contents

Preface v

1 Coding Theory 1
  1.1 Error-Correcting Codes 1
  1.2 Linear Codes 4
  1.3 Cyclic Codes 5
  1.4 Reed-Solomon Codes 7
  1.5 Decoding Reed-Solomon Codes 9

2 Peterson-Gorenstein-Zierler Decoder 19
  2.1 PGZ Decoding Algorithm 19
  2.2 fPGZ Decoding Algorithm 26
  2.3 Avoiding Malfunctions in fPGZ 36

3 Berlekamp-Massey Decoder 43
  3.1 BM Decoding Algorithm 43
  3.2 Avoiding Malfunctions in BM 59
  3.3 Comparing fPGZ.2 and BM.2 63

4 Error Value Formulas 69
  4.1 Horiguchi’s formula 76
  4.2 Application to the fPGZ Decoder 76
  4.3 Comparison with the BM Decoder 78

5 Parallel Implementation 83
  5.1 Parallel Implementation of PGZ Decoder 84
  5.2 Parallel Implementation of the BM decoder 90

Conclusions 101

Bibliography 102
Preface

In coding theory, Reed-Solomon codes are one of the most well-known and widely used classes of error-correcting codes. These codes have excellent mathematical features and important applications, from deep-space communication to consumer electronics, such as CDs and DVDs. The extensive practical use of Reed-Solomon codes is made possible by the existence of efficient decoding procedures and, at the same time, makes it necessary to devise ever-faster decoding algorithms. Indeed, while the structure of Reed-Solomon codes is well understood, the problem of designing optimal decoding algorithms still remains an active area of research.

In this thesis we study and compare two major strategies known for decoding Reed-Solomon codes: the Peterson-Gorenstein-Zierler (PGZ) and the Berlekamp-Massey (BM) decoder. Our aim is to improve existing decoding algorithms, by decreasing their computational time complexity, and propose faster new ones, based on a parallel implementation in integrated circuits.

The PGZ decoder, introduced in 1961, was the first practical decoder for Reed-Solomon codes. It is based on simple tools of linear algebra and finds the number, the positions and the values of errors, which occurred during transmission, by computing determinants or solving linear systems. Despite its simplicity, the PGZ decoder has often been considered quite inefficient because of its $O(t^4)$ computational time complexity, where $t$ is the error correction capability of the code. This opinion is not justified, indeed there exists an improved version of the PGZ decoder with quadratic computational time complexity, which we will call the fast Peterson-Gorenstein-Zierler (fPGZ) decoding algorithm. This improvement, which was presented by M. Schimidt and G.P. Fettweis in 1996, is obtained by exploiting the Hankel structure of the matrices involved in the decoding and fast inversion techniques for Hankel matrices.

Because of this, the PGZ decoder can finally be considered an effective alternative to the BM decoder, which was introduced in 1969 as a first example of a quadratic time decoding algorithm for Reed-Solomon codes. In addition we show that the modified version of the PGZ decoder is not only
an alternative to the BM decoder, but in a sense it is a particular case of the latter. In fact, we prove that the intermediate outcomes obtained in the implementation of f PGZ are a subset of those of the BM decoding algorithm. Thus we show that the relationship between these two decoding strategies for Reed-Solomon codes is much closer than previously thought.

The BM decoder is an iterative algorithm which computes a recursive sequence of polynomials converging to the error-locator polynomial $\sigma(x)$, whose degree is the number of errors and whose roots identify the error positions. The error values are usually computed by evaluating $\omega(x)$ in the roots of $\sigma(x)$, where $\omega(x)$ is the error-evaluator polynomial. We prove that it is possible to improve the error value computation in the BM decoder avoiding the additional operations needed to compute $\omega(x)$ from $\sigma(x)$. We achieve this result by using a polynomial which is a byproduct of the computation of $\sigma(x)$ in the place of $\omega(x)$. This alternative method for the evaluation of errors in the BM decoder had already been observed by T. Horiguchi, but in this thesis we obtain a more direct proof, recovering the new formula as a corollary of the correctness of the BM decoding algorithm.

Nevertheless we study the techniques of linear algebra used by T. Horiguchi in order to generalize the applications of his new error evaluation method. Thanks to this generalization, we also improve the error value computation in the PGZ decoder. In fact we prove that the error values can be calculated by evaluating polynomials that are intermediate outcomes of the f PGZ decoding algorithm, instead of solving a separate linear system as usual.

Moreover thanks to the study done on the structure of the syndrome matrix and its leading principal minors, we can state a new iterative formulation of the PGZ decoder well suited to a parallel implementation on integrated microchips. Indeed we prove that the number, the positions and the values of the errors can be computed directly by using the leading principal minors of the syndrome matrix, whose computation can be accomplished iteratively via a parallel implementation of the Laplace expansion for determinants. We show that this parallel version of the PGZ decoder is a decoding algorithm for Reed-Solomon codes with an $O(e)$ computational time complexity, where $e$ is the number of errors which occurred, although a fairly large number (about $t\left(\frac{e}{1/2}\right)$) of elementary circuit elements is needed.

Finally a parallel implementation for the BM decoder is given, which is less expensive and simpler from the point of view of the hardware required. In this case we restructure the BM decoding algorithm in order to avoid some instructions hindering the parallel implementation. The result is a decoding algorithm with an $O(t)$ computational time complexity.

Our conclusions are that the PGZ and BM decoder are both valid alternatives for decoding Reed-Solomon codes. As seen, the choice of a decoding
algorithm and an architecture for its hardware implementation has to be influenced by the particular application of the code and its resulting conditions. We analyze several different cases throughout our thesis.

The organization of the thesis is as follows. In chapter 1, Reed-Solomon codes and the general outline of their decoding algorithms are described. In chapter 2, the Peterson-Gorenstein-Zierler decoder and its quadratic improved version are discussed. We also describe the necessary and sufficient conditions to avoid decoder malfunctions in the $fPGZ$ decoding algorithm. In chapter 3, we study in detail the Berlekamp-Massey decoder and its implementation as a $t$-bounded distance decoding algorithm and we compare the latter with the $fPGZ$ decoding algorithm. Chapter 4 deals with Horiguchi’s formula to compute the error values and its applications to the $fPGZ$ and the BM decoding algorithms and continues the comparison between these two decoding strategies. Finally, in chapter 5 we present and compare two parallel decoding algorithms implementable as integrated circuits, one based on the PGZ decoder and linear algebra and the other on the BM decoding algorithm.
Chapter 1

Coding Theory

Coding theory deals with mathematical models and algorithms to transmit data across noisy channels. In sections 1.1, 1.2 and 1.3 we will give a brief introduction to error-correcting codes and how they work. For the proofs of the propositions and theorems stated in these sections, we refer to [HLL+91]. In section 1.4 we will introduce Reed-Solomon codes, with the notations and the properties that will be used in the next chapters to investigate their decoding procedure. The general outline of the latter is described in section 1.5.

1.1 Error-Correcting Codes

Error-correcting codes are used to detect and correct errors that occur when data are transmitted across some noisy channel or stored on some medium. In many cases, the data are transmitted or stored as a sequence of words of fixed length $n$ and each word is made up of symbols taken from a finite alphabet. We describe this situation considering a finite field $K$ and a fixed natural number $n$. A block code (or simply code) of length $n$ is a subset of $K^n$, denoted by $C$. In this model, the elements of the field $K$ represent the symbols of the finite alphabet and the vectors in $K^n$ represent all possible words of length $n$. For this reason, throughout this thesis, words (of length $n$) are synonymous with vectors of $K^n$. The vectors of $C$ represent all words that can form the transmitted sequence of data and they will be called codewords.

The noise of the channel used for the transmission may cause some errors, namely some components of a transmitted codeword, during the transmission, may be modified (but cannot be eliminated, indeed we suppose that a codeword of length $n$ is always received as a word length $n$). In this case the received word is of the same length but different from the transmitted one.
CHAPTER 1. CODING THEORY

In order to measure this difference, we introduce the following tools:

**Definition 1.1.1.** The Hamming weight of a word \( v = (v_0, v_1, \ldots, v_{n-1}) \in \mathbb{K}^n \) is the number of the nonzero components of \( v \), that is

\[
\text{wt}(v) = \# \{ i \mid v_i \neq 0 \}
\]

The Hamming distance between two words \( u, v \in \mathbb{K}^n \) is the Hamming weight of the vector \( u - v \), that is

\[
\text{dist}(u, v) = \text{wt}(u - v) = \# \{ i \mid u_i \neq v_i \}
\]

It can be easily shown that the Hamming distance defines a metric on the vector space \( \mathbb{K}^n \).

Note that if \( c \in C \) is sent over a channel and \( r \in \mathbb{K}^n \) is received, then the Hamming distance between \( r \) and \( c \) is the number of errors which occurred, that is the number of components of \( c \) modified during the transmission. The vector \( e = r - c \) is called the error vector and we have that \( \text{wt}(e) = \text{dist}(r, c) \).

We assume that the errors are distributed randomly, that is a component of the transmitted word can be wrong independently of the other components and the probability that a component is affected in transmission is the same for all the components. If we indicate with \( p \) this probability (\( 0 \leq p \leq 1 \)) and with \( r \in \mathbb{K}^n \) a received word, then the probability that the word \( v \in \mathbb{K}^n \) is the sent word is

\[
p^d (1 - p)^{n - d}
\]

where \( d = \text{dist}(r, v) \). We can reasonably suppose that \( 0 < p < \frac{1}{2} \) and thus we have that

\[
p^{d_1} (1 - p)^{n - d_1} \leq p^{d_2} (1 - p)^{n - d_2} \iff d_1 \geq d_2
\]

namely the closest codeword (in the sense of the Hamming distance) to \( r \) is the most likely to have been sent. For this reason, we correct a received word \( r \) with the codeword that is the closest to \( r \). If there are several words in \( C \) at the same distance from \( r \), we may arbitrarily choose one of these or we may ask for a retransmission. Any algorithm doing this will be denoted by a decoding algorithm.

Clearly with this decoding strategy mistakes may happen since the closest codeword to \( r \) may not be the one sent. In order to understand and avoid decoding mistakes we study and classify the error vectors in the following way:
1.1. ERROR-CORRECTING CODES

Definition 1.1.2. A vector \( e \in \mathbb{K}^n \setminus \{0\} \) is a detectable error if for any \( c \in C \) we have that \( c + e \notin C \), while it is an correctable error if for any \( c \in C \) we have that \( \text{dist}(c + e, c) < \text{dist}(c + e, v) \) for any \( v \in C \setminus \{c\} \).

In other words, \( e \) is detectable if for any transmitted codeword \( c \), we can recognize that \( c + e \) is not a codeword, whereas it is correctable if \( c + e \) is closer to \( c \) than any other codeword. In order to distinguish between detectable and correctable errors, the following definition is useful:

Definition 1.1.3. If \( C \) is a code with at least two different words, then the integer \( \text{dist}(C) \) defined by

\[
\text{dist}(C) = \min\{\text{dist}(u, v) \mid u, v \in C \text{ and } u \neq v\}
\]

is called the distance of the code \( C \).

The following two propositions show the relation between the distance of a code and the identification of detectable or correctable error vectors.

Proposition 1.1.4. Let \( C \subseteq \mathbb{K}^n \) be a code of distance \( d \). All error vectors \( e \in \mathbb{K}^n \setminus \{0\} \) such that \( \text{wt}(e) \leq d - 1 \) are detectable and there is at least one error vector in \( \mathbb{K}^n \) of weight \( d \) that is not detectable.

Proposition 1.1.5. Let \( C \subseteq \mathbb{K}^n \) be a code of distance \( d \). All error vectors \( e \in \mathbb{K}^n \) such that \( \text{wt}(e) \leq \left\lfloor \frac{d - 1}{2} \right\rfloor \) are correctable and there is at least one error vector in \( \mathbb{K}^n \) of weight \( \left\lfloor \frac{d - 1}{2} \right\rfloor + 1 \) that is not correctable.

For this reason, in a code of distance \( d \) the integer

\[
\mathbb{C} = \left\lfloor \frac{d - 1}{2} \right\rfloor
\]

is called the error correction capability of the code. Indeed as seen if the number of errors happened is less or equal to \( t \), then a decoding algorithm corrects the received word with the sent codeword and no decoding mistakes can happen.

This codes property leads to define an useful kind of decoding algorithm: we call \( t \)-bounded distance decoding algorithm a decoding algorithm that either decodes a received word \( r \) into the unique codeword \( c \) at distance not more than \( t \) from \( r \) (if such codeword exists) or indicates that no such codeword exists, declaring a decoder failure. To understand the behavior of a \( t \)-bounded distance decoding algorithm, we introduce the set formed by the disjoint union of all the closed balls of radius \( t \) centered at a codeword:

\[
\mathcal{B} = \bigcup_{c \in C} B_t(c)
\]
There are three possible cases that may occur when the codeword $c \in C$ is transmitted and $r = c + e$ is received:

1. If $\text{wt}(e) \leq t$, then $r \in B$. A $t$-bounded distance decoding algorithm decodes correctly $r$, giving as output $c$.

2. If $\text{wt}(e) > t$ and $r \in B$, then there exists $\bar{c} \in C$ such that $\bar{c} \neq c$ and $r \in B_t(\bar{c})$. In this case, the output of a $t$-bounded distance decoding algorithm will be the “wrong” codeword $\bar{c}$. Thus we talk about decoder error. Note that evidently decoder errors cannot be detected in any way and they are unavoidable for any decoding algorithm.

3. If $r \notin B$, then a $t$-bounded distance decoding algorithm must detect the error vector $e$ and declare the decoder failure, even though it is not capable of correcting it.

As we will see later on, when $r \notin B$ some decoding algorithms may not indicate decoder failure as expected and may instead produce an output vector that is not a codeword at all. This can happen even if they behave as $t$-bounded distance decoding algorithm in the case that $r \in B$. We refer to this event as a decoder malfunction. We will show that decoder malfunctions can be detected and avoided by adding conditions which declare a decoder failure.

### 1.2 Linear Codes

Linear codes are a large family of error-correcting codes, for which tools and techniques of linear algebra are used in encoding and decoding algorithms. In this section we briefly recall the main properties of these codes.

**Definition 1.2.1.** Let $C$ be a code of length $n$. If $C$ is also a linear subspace of $\mathbb{K}^n$, then it is called a linear code. In this case, the dimension of $C$ as a linear subspace of $\mathbb{K}^n$ is called the dimension of the code $C$.

We indicate with $C(n, k, d)$ a linear code of length $n$, dimension $k$ and distance $d$. The structure of vector space allows a simpler description of the code. For example, if $C$ is a linear code, then its distance can be calculated simply as

$$\text{dist}(C) = \min\{\text{wt}(v) \mid v \in C \text{ and } v \neq 0\}$$

and moreover it is possible to define the following useful matrices, which characterize a linear code:
1.3. CYCLIC CODES

Definition 1.2.2. Let \( C \) be a linear code of length \( n \) and dimension \( k \). Any \( k \times n \) matrix whose rows form a basis for \( C \), as a linear subspace of \( \mathbb{K}^n \), is called a generator matrix. While any \( n \times (n-k) \) matrix whose columns form a basis for the orthogonal complement of \( C \) is called a parity-check matrix.

Obviously, if \( G \) is a generator matrix for the code \( C(n,k,d) \), then we have \( \text{rk}(G) = k \) and
\[
\mathbf{v} \in C \iff \exists \mathbf{u} \in \mathbb{K}^k \text{ such that } \mathbf{v} = \mathbf{u}G
\]
while if \( P \) is a parity-check matrix for \( C(n,k,d) \), then \( \text{rk}(P) = n - k \) and
\[
\mathbf{v} \in C \iff \mathbf{v}P = 0
\]

The following proposition shows how to calculate the distance of a linear code knowing a parity-check matrix.

Proposition 1.2.3. Consider a linear code \( C \) of length \( n \) and dimension \( k \). Let \( P \) be a parity-check matrix for \( C \). Then \( C \) has distance \( d \) if and only if any \( d-1 \) rows of \( P \) are linearly independent, and at least \( d \) rows of \( P \) are linearly dependent.

As consequence, we obtain the following theorem that shows the existing relation between the three main characteristics of a linear code: length, dimension and distance.

Theorem 1.2.4 (Singleton Bound). For any linear code \( C(n,k,d) \),
\[
d \geq n - k + 1.
\]

A linear code \( C(n,k,d) \) that realizes the equality \( d = n - k + 1 \) is said to be a maximum distance separable (or MDS) code. If we recall that the integer \( t = \lfloor \frac{d-1}{2} \rfloor \) represents the error correction capability of the code, we can understand that MDS codes are optimal codes, in the sense that they have the greatest error correction capability for the same length and dimension.

1.3 Cyclic Codes

In order to have very efficient decoding algorithms, linear codes may be no sufficient. So we introduce codes with more algebraic structure:

Definition 1.3.1. A code \( C \) of length \( n \) is cyclic if it is closed under the operation cyclic right shift of a codeword. In other words \( C \subseteq \mathbb{K}^n \) is a cyclic code if
\[
\mathbf{v} = (v_0, v_1, \ldots, v_{n-1}) \in C \implies \pi(\mathbf{v}) \overset{def}{=} (v_{n-1}, v_0, \ldots, v_{n-2}) \in C
\]
CHAPTER 1. CODING THEORY

To appreciate the algebraic structure of a cyclic code we introduce the representation of the elements in \( \mathbb{K}^n \) as polynomials. There is an one-to-one correspondence between vectors in \( \mathbb{K}^n \) and polynomials in \( \mathbb{K}[x] \) of degree less than \( n-1 \), defined by

\[
\mathbf{v} = (v_0, v_1, \ldots, v_{n-1}) \leftrightarrow \mathbf{v}(x) = \sum_{i=0}^{n-1} v_i x^i
\]

Thus a code of length \( n \) can be represented as a set of polynomials of degree at most \( n-1 \). Moreover if the vector \( \mathbf{v} \) corresponds to the polynomial \( \mathbf{v}(x) \), as showed above, the vector \( \pi(\mathbf{v}) \) corresponds to the polynomial \( x \mathbf{v}(x) \pmod{x^n - 1} \). So a cyclic code of length \( n \) can be seen as a subspace of the ring \( \mathbb{K}[x]/(x^n - 1) \) closed under the multiplication by \( x \). That is, a cyclic code of length \( n \) corresponds to an ideal of the ring \( \mathbb{K}[x]/(x^n - 1) \). Moreover any ideal of \( \mathbb{K}[x]/(x^n - 1) \) corresponds to a cyclic code and so we have the one-to-one correspondence given by:

\[
\mathcal{C} \subseteq \mathbb{K}^n \text{ cyclic code} \leftrightarrow \mathcal{C} \subseteq \mathbb{K}[x]/(x^n - 1) \text{ ideal}
\]

Henceforth, when studying cyclic codes, we will refer to the elements in \( \mathbb{K}^n \) both as vectors and as polynomials.

Since \( \mathbb{K}[x]/(x^n - 1) \) is a principal ideal ring, every cyclic code, as an ideal, can be generated by just one element. More precisely, we have:

**Proposition 1.3.2.** Let \( \mathcal{C} \) be a cyclic code, then there exists a unique nonzero polynomial \( g(x) \in \mathcal{C} \) such that

\[
\mathbf{v}(x) \in \mathcal{C} \iff g(x) | \mathbf{v}(x)
\]

We define the *generator polynomial* of the cyclic code \( \mathcal{C} \) the polynomial \( g(x) \) described above and we call the cyclic code of length \( n \) generated by the polynomial \( g(x) \), the code formed by all vectors \( \mathbf{v} \in \mathbb{K}^n \) such that \( g(x) | \mathbf{v}(x) \). Remark that not all polynomials in \( \mathbb{K}[x] \) can be generator polynomials of a cyclic code. Indeed the set \( \{ \mathbf{v} \in \mathbb{K}^n : g(x) | \mathbf{v}(x) \} \) is a cyclic code if and only if the polynomial \( g(x) | x^n - 1 \).

**Proposition 1.3.3.** If \( \mathcal{C} \) is a cyclic code of length \( n \) and \( g(x) \) is its generator polynomial, then

\[
\dim(\mathcal{C}) = n - \deg(g(x))
\]
1.4. REED-SOLOMON CODES

An important class of cyclic codes, still used a lot in practice, was discovered by R. C. Bose and D. K. Ray-Chaudhuri (1960) and independently by A. Hocquenghem (1959). The codes are known as BCH codes.

Definition 1.3.4 (BCH code). Let $\mathbb{F}_q$ be the finite field consisting of $q$ elements and let $n$ and $\delta$ be two positive integer. Given the smallest $a \in \mathbb{N}$ such that $\mathbb{F}_{q^a}$ has a primitive $n$th root of unity $\beta$, then a cyclic code of length $n$ over $\mathbb{F}_q$ is called a BCH code of parameters $n$ and $\delta$ generated by $\beta$ if its generator polynomial is the least common multiple of the minimal polynomials over $\mathbb{F}_q$ of $\beta^l, \beta^{l+1}, \ldots, \beta^{l+\delta-2}$, for some $l \in \mathbb{N}$. This code is indicated by $BCH(n, \delta, \beta)$.

We observe that the the least common multiple of the minimal polynomials of $\beta^l, \beta^{l+1}, \ldots, \beta^{l+\delta-2}$ divides $x^n - 1$ because $\beta$ is a primitive $n$th root of unity. So the above definition is correct.

Example 1.3.5. Let $l = 1$ and $q = 2$. Let $\beta \in \mathbb{F}_{16}$ be a primitive fourth root of unity satisfying $\beta^4 + \beta + 1 = 0$, then its minimal polynomials over $\mathbb{F}_2$ is the irreducible polynomial $f_1(x) = x^4 + x + 1$. Since $f_1(\beta^2) = f_1(\beta)^2$ and $f_1(\beta^4) = f_1(\beta)^4$, $f_1(x)$ is also the minimal polynomials over $\mathbb{F}_2$ of $\beta^2$ and of $\beta^4$. Moreover it can be easily proved that the minimal polynomials over $\mathbb{F}_2$ of $\beta^3$ is $f_2(x) = x^4 + x^3 + 1$. Thus the code $BCH(15, 5, \beta)$ over $\mathbb{F}_2$ is generated by the polynomial

$$g(x) = f_1(x) \cdot f_2(x) = x^8 + x^7 + x^5 + x^4 + x^3 + x + 1$$

1.4 Reed-Solomon Codes

Reed-Solomon codes, named after their inventors [RS60], are an extensively studied family of error-correcting codes, heavily used in theoretical and practical settings. Furthermore, the optimality of the Reed-Solomon codes, in terms of error correction capability, and their algebraic properties are the main reasons for the great fame of these codes and their widespread use in coding theory and computer science.

From here on, let $\mathbb{F}_q$ be the finite field consisting of $q$ elements ($\mathbb{F}_q = \mathbb{F}_{p^m}$ with $m, p \in \mathbb{N}$ and $p$ prime). We indicate with $n$ the cardinality of the multiplicative group of $\mathbb{F}_q$, that is $n = q - 1$, and with $\alpha$ a primitive element of $\mathbb{F}_q$, i.e. an element with multiplicative order equal to $n$. Reed-Solomon codes are special case of cyclic codes of length $n$ over the field $\mathbb{F}_q$. More precisely:
CHAPTER 1. CODING THEORY

Definition 1.4.1 (Reed-Solomon code). Let $n$ and $\alpha$ be as above and let $d$ be an integer less than or equal to $n$. The Reed-Solomon code of parameters $n$ and $d$ generated by $\alpha$ (denoted by $RS(n, d, \alpha)$) is the cyclic code of length $n$ over $\mathbb{F}_q$ generated by the polynomial

$$g(x) = (x - \alpha^l)(x - \alpha^{l+1}) \cdots (x - \alpha^{l+d-2})$$

for some $l \in \mathbb{N}$.

We observe that Reed-Solomon codes are examples (the simplest ones) of BCH codes, namely in the case $n = q - 1$. For simplicity, we will always take $l = 1$, but the general treatment is equivalent. The following proposition summarizes the properties of a Reed-Solomon code.

Proposition 1.4.2. Let $RS(n, d, \alpha)$ be as in definition 1.4.1. Then the following holds:

1. $\dim (RS(n, d, \alpha)) = n - d + 1$;
2. $v \in RS(n, d, \alpha) \iff v(\alpha^i) = 0 \forall i = 1, 2, \ldots, d - 1$ and a parity-check matrix for the code $RS(n, d, \alpha)$ is
   
   $$P = \begin{pmatrix}
   1 & 1 & \cdots & 1 \\
   \alpha & \alpha^2 & \cdots & \alpha^{d-1} \\
   \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^{d-1})^2 \\
   \vdots & \vdots & \ddots & \vdots \\
   \alpha^{n-1} & (\alpha^{n-1})^2 & \cdots & (\alpha^{d-1})^{n-1}
   \end{pmatrix};$$
3. $\text{dist} (RS(n, d, \alpha)) = d$;
4. the code $RS(n, d, \alpha)$ is an MDS code.

Proof. 1. From definition 1.4.1 it follows that $\deg(g(x)) = d - 1$, thus (1) is an immediate consequence of proposition 1.3.3.

2. Since the code $RS(n, d, \alpha)$ is a cyclic code generated by the polynomial $g(x)$, the equivalence stated in (2) follows directly from proposition 1.3.2. To verify that $P$ is a parity-check matrix for the code $RS(n, d, \alpha)$ is sufficient to observe that, if $v \in (\mathbb{F}_q)^n$, then we may evaluate the polynomial $v(x)$ in $\alpha^j$ with an inner product, in fact:

$$v(\alpha^j) = (v_0, v_1, \ldots, v_{n-1}) \begin{pmatrix}
1 \\
\alpha^j \\
\vdots \\
\alpha^{(n-1)j}
\end{pmatrix} = \sum_{i=0}^{n-1} v_i \alpha^{ij}$$

Therefore we have that $v(\alpha^i) = 0 \forall i = 1, 2, \ldots, d - 1 \iff vP = 0$. 

3. From the Singleton bound, using (1), it follows that
\[ \text{dist}\,(RS(n, d, \alpha)) \leq d \]

To conclude the proof it is sufficient to show that any \( d - 1 \) rows of \( P \) are linearly independent. In fact, by proposition 1.2.3, this leads to \( \text{dist}\,(RS(n, d, \alpha)) \geq d \). Consider \( d - 1 \) rows in \( P \). In other words, for any \( i = 1, 2, \ldots, d - 1 \), we choose \( m_i \in \{0, 1, \ldots, n - 1\} \) (with the condition \( m_i \neq m_j \) if \( i \neq j \)) and we define \( a_i = \alpha^{m_i} \). So we obtain the square submatrix
\[
M = \begin{pmatrix}
a_1 & a_1^2 & \cdots & a_1^{d-1} \\
a_2 & a_2^2 & \cdots & a_2^{d-1} \\
\vdots & \vdots & \ddots & \vdots \\
a_{d-1} & a_{d-1}^2 & \cdots & a_{d-1}^{d-1}
\end{pmatrix}
\]

and it is easy to show that \( M \) is non-singular because it can be written as a product of a non-singular Vandermonde matrix and a non-singular diagonal matrix. That is
\[
M = \begin{pmatrix}
1 & a_1 & a_1^2 & \cdots & a_1^{d-2} \\
1 & a_2 & a_2^2 & \cdots & a_2^{d-2} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & a_{d-1} & a_{d-1}^2 & \cdots & a_{d-1}^{d-2}
\end{pmatrix}
\begin{pmatrix}
a_1 & 0 & \cdots & 0 \\
0 & a_2 & \cdots & \vdots \\
\vdots & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & a_{d-1}
\end{pmatrix}
\]

4. Trivial using (1) and (3).

\( \square \)

1.5 Decoding Reed-Solomon Codes

Consider a Reed-Solomon code of length \( n \) and distance \( d \) over \( \mathbb{F}_q \) and let \( \alpha \) be the primitive \( n \)th root of unity that defines the code.

Suppose that \( c = (c_0, c_1, \ldots, c_{n-1}) \in RS(n, d, \alpha) \) is the codeword sent and \( e = (e_0, e_1, \ldots, e_{n-1}) \in (\mathbb{F}_q)^n \) is the error vector, thus the received word is \( r = c + e = (r_0, r_1, \ldots, r_{n-1}) \in (\mathbb{F}_q)^n \). We represent this situation with the polynomials
\[
c(x) = \sum_{i=0}^{n-1} c_i x^i \in \mathbb{F}_q[x]
\]
\[
e(x) = \sum_{i=0}^{n-1} e_i x^i \in \mathbb{F}_q[x]
\]
\[ \mathbf{r}(x) = \sum_{i=0}^{n-1} r_i x^i \in \mathbb{F}_q[x] \]

and, clearly, we have \( \mathbf{r}(x) = \mathbf{c}(x) + \mathbf{e}(x) \).

From now on, we call \( e \) the Hamming weight of the error vector \( \mathbf{e} \) and we define
\[
\mathcal{I} = \{ i \mid e_i \neq 0 \} = \{ p_1 < p_2 < \cdots < p_e \}
\]
the set of the positions where an error occurs. Thus for any \( i \in \{1, 2, \ldots, e\} \),
\( E_i = e_{p_i} \in \mathbb{F}_q \) represents the value of the error that occurs in the position \( p_i \).
In order to correct the received word \( \mathbf{r} \), we need to find the set \( \mathcal{I} \) and the values \( E_i \). For this purpose we introduce the following tools:

**Definition 1.5.1.** For any \( i \in \{1, 2, \ldots, e\} \), we define
\[
X_i = \alpha^{p_i} \in \mathbb{F}_q
\]
Moreover we call
\[
\sigma(x) = \prod_{i=1}^{e} (1 - X_i x) \quad \text{the error-locator polynomial}
\]
\[
\omega(x) = \sum_{i=1}^{e} E_i X_i \prod_{j \neq i} (1 - X_j x) \quad \text{the error-evaluator polynomial}.
\]
Evidently the elements \( X_1^{-1}, X_2^{-1}, \ldots, X_e^{-1} \) are precisely the roots of \( \sigma(x) \) in the field \( \mathbb{F}_q \) and they identify the errors positions thanks to the relation
\[
X_i^{-1} = \alpha^{n-p_i} \quad \text{for any } i \in \{1, 2, \ldots, e\}
\]
which follows by (1.1). Thus if we are able to calculate the error-locator polynomial, then we can know all the error positions \( \{p_1, p_2, \ldots, p_e\} \) from its roots. While the error-locator polynomial permits calculating the error positions, the error-evaluator polynomial is linked to the computation of the error values \( E_1, E_2, \ldots, E_e \). Indeed it is easy to prove the following relation, stated for the first time in [For65] and known as Forney’s formula:
\[
E_i = -\frac{\omega(X_i^{-1})}{\sigma'(X_i^{-1})} \quad \forall i = 1, 2, \ldots, e
\]
where \( \sigma'(x) \) is the formal derivative of \( \sigma(x) \). Thus the knowledge of the polynomials \( \sigma(x) \) and \( \omega(x) \) permits a complete decoding of the received word \( \mathbf{r} \), since it allows the computation of the error positions and of error values. For this reason from here on our aim is the determination of the these polynomials.

Other simple properties of the error-locator and the error-evaluator follow immediately by their definition. We list them below:
Remark 1.5.2.

1. deg(σ(x)) = e = wt(e) and σ(0) = 1;
2. σ(x) splits completely in \( \mathbb{F}_q[x] \) and does not have multiple roots;
3. deg(ω(x)) ≤ e − 1;
4. \( \gcd(\sigma(x), \omega(x)) = 1 \).

In proposition 1.4.2 we saw that \( v \in RS(n, d, \alpha) \iff v(\alpha^i) = 0 \ \forall i = 1, 2, \ldots, d - 1 \) and this justifies the introduction of the following definition:

**Definition 1.5.3.** For any \( i \in \{1, 2, \ldots, d - 1\} \) we define

\[ S_i = r(\alpha^i) \quad \text{the } i\text{th syndrome} \]

Moreover \( S(x) = \sum_{i=1}^{d-1} S_i x^{i-1} \) is the *syndrome polynomial*.

It is clear that if \( S_1 = S_2 = \cdots = S_{d-1} = 0 \), then \( r \in RS(n, d, \alpha) \) and thus either \( e = 0 \) or \( e \) is a not detectable error. If we suppose that \( e = \text{wt}(e) \) is less or equal to \( t \), the error correction capability of the code \( RS(n, d, \alpha) \), then the second case is not possible by proposition 1.1.4 and hence the condition \( S_1 = S_2 = \cdots = S_{d-1} = 0 \) implies that no error has occurred.

Since \( c \in RS(n, d, \alpha) \), \( c(\alpha^i) = 0 \) for every \( i \in \{1, 2, \ldots, d - 1\} \), thus we can observe that

\[
S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) = \sum_{j=1}^{e} E_j X_j^i \quad (1.3)
\]

for any \( i \in \{1, 2, \ldots, d - 1\} \). This characterization of the syndromes is of fundamental importance because it allows to write a relation, more precisely a congruence, stated in the following proposition and satisfied by the error-locator and error-evaluator polynomials.

**Proposition 1.5.4** (key equation). If \( \sigma(x) \), \( \omega(x) \) and \( S(x) \) are the polynomials respectively defined in definition 1.5.1 and 1.5.3, then

\[
\sigma(x) S(x) \equiv \omega(x) \pmod{x^{d-1}} \quad (1.4)
\]
CHAPTER 1. CODING THEORY

Proof. We observe that

\[
\frac{\omega(x)}{\sigma(x)} = \frac{\sum_{i=1}^{e} E_iX_i \prod_{j \neq i}(1 - X_jx)}{\prod_{k=1}^{d-1}(1 - X_kx)} = \sum_{i=1}^{e} E_iX_i \frac{1}{1 - X_i x}
\]

and using the identity \(\sum_{i \geq 0} y^i = \frac{1}{1 - y}\) we obtain

\[
\frac{\omega(x)}{\sigma(x)} = \sum_{i=1}^{e} E_iX_i \sum_{j=0}^{+\infty} (X_i x)^j = \sum_{j=0}^{+\infty} \left( \sum_{i=1}^{e} E_i X_i^{j+1} \right) x^j
\]

By (1.3), we have

\[
\frac{\omega(x)}{\sigma(x)} = \sum_{j=0}^{d-2} S_{j+1} x^j + \sum_{j \geq d-1} \left( \sum_{i=1}^{e} E_i X_i^{j+1} \right) x^j
\]

Considering the last equality modulo \(x^{d-1}\) the proof is completed. \(\square\)

Given the syndrome polynomial, the pair \((\sigma(x), \omega(x))\) is not the unique possible solution of the key equation (1.4). For this reason we consider the set:

\[
M = \{ (a(x), b(x)) \in \mathbb{F}_q[x]^2 \mid a(x)S(x) \equiv b(x) \pmod{x^{d-1}} \}
\]

Evidently, \(M\) is a submodule of \(\mathbb{F}_q[x]^2\) and \((\sigma(x), \omega(x)) \in M\). Considering that \(\deg(\omega(x)) < \deg(\sigma(x))\), we are interested in the pairs \((a(x), b(x)) \in M\) such that \(\deg(b(x)) < \deg(a(x))\) and thus we introduce the followings definitions:

**Definition 1.5.5.** We call a pair \((a(x), b(x)) \in \mathbb{F}_q[x]^2\) a **valid solution** of the key equation (1.3) if \((a(x), b(x)) \in M\) and \(\deg(b(x)) < \deg(a(x))\). Moreover we call a pair \((a(x), b(x)) \in M\) a **valid solution of minimal degree** if it is a valid solution and for any other valid solution \((a_1(x), b_1(x))\), we have that \(\deg(a(x)) \leq \deg(a_1(x))\).

Recalling that \(e = \text{wt}(e)\) and \(t\) is the error correction capability of the code \(RS(n, d, \alpha)\), we have:
Theorem 1.5.6. If $e \leq t$, then the pair $(\sigma(x), \omega(x))$ is the unique, up to a multiplicative constant, valid solution of minimal degree of the key equation (1.4).

Proof. We have already observed that $(\sigma(x), \omega(x))$ is a valid solution of the key equation (1.4). Now we show that $(\sigma(x), \omega(x))$ is a valid solution of minimal degree. If there were a valid solution $(a(x), b(x))$ such that $\deg(a(x)) < \deg(\sigma(x))$ then we would have

$$a(x)\omega(x) \equiv a(x)\sigma(x)S(x) \equiv b(x)\sigma(x) \pmod{x^{d-1}}$$

and, since $\deg(\sigma(x)) = e$,

$$\deg(a(x)\omega(x)) = \deg(a(x)) + \deg(\omega(x)) < 2e < d - 1$$
$$\deg(b(x)\sigma(x)) = \deg(b(x)) + \deg(\sigma(x)) < 2e < d - 1$$

From the degree inequalities, it follows that the congruence (1.5) is an equality, i.e. :

$$a(x)\omega(x) = b(x)\sigma(x).$$

Therefore $\sigma(x)|a(x)\omega(x)$. Since $\gcd(\sigma(x), \omega(x)) = 1$, it follows that $\sigma(x)|a(x)$ and this is absurd. Hence $(\sigma(x), \omega(x))$ is a valid solution of minimal degree. Finally we show that $(\sigma(x), \omega(x))$ is the unique (up to a multiplicative constant) valid solution of minimal degree. Let $(\tilde{\sigma}(x), \tilde{\omega}(x)) \in \mathcal{M}$ be a valid solution of minimal degree of the key equation (1.4). Then $\deg(\tilde{\sigma}(x)) = \deg(\sigma(x)) = e$ and

$$\tilde{\sigma}(x)\omega(x) \equiv \tilde{\sigma}(x)\sigma(x)S(x) \equiv \tilde{\omega}(x)\sigma(x) \pmod{x^{d-1}}$$

with $\deg(\tilde{\sigma}(x)\omega(x)) < 2e < d - 1$ and $\deg(\tilde{\omega}(x)\sigma(x)) < 2e < d - 1$. Thus we have that

$$\tilde{\sigma}(x)\omega(x) = \tilde{\omega}(x)\sigma(x)$$

Since $\gcd(\sigma(x), \omega(x)) = 1$ and $\deg(\tilde{\sigma}(x)) = \deg(\sigma(x))$, we can conclude that there exists $k \in \mathbb{F}_q$ such that

$$\tilde{\sigma}(x) = k\sigma(x) \text{ and } \tilde{\omega}(x) = k\omega(x)$$

The proof is concluded. 

Thanks to theorem 1.5.6 developing procedures that, given the syndrome polynomial $S(x)$, solve the key equation (1.4) finding its minimal valid solution, will allow us to be able to find the polynomials $\sigma(x)$ and $\omega(x)$, that is
to decode $r$.

Any procedure that solves the key equation (1.4) and more in general any procedure that performs the decoding of Reed-Solomon codes, requires computations using finite field arithmetic. In this thesis we do not discuss in detail the implementations and the circuits that perform addition, multiplication and division over $\mathbb{F}_q$ (for which we refer to [LC83] and [PW72]). We only recall that the elements of $\mathbb{F}_q = \mathbb{F}_{p^m}$ can be represented as vectors of $(\mathbb{F}_p)^m$ with respect to a basis (for example $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ with $\alpha$ primitive element of $\mathbb{F}_q$) and that, with the vector representation, additions and multiplications by constant field element are simple, but unconstant multiplication and division are not. In particular:

- To add two field elements, we simply add their vector representations over the base field $\mathbb{F}_p$. This operation is the simplest among $\mathbb{F}_q$ arithmetic.

- Multiplying a field element by a fixed element from the same field is simple enough because it can be seen as a $\mathbb{F}_p$ linear map of $\mathbb{F}_q$. Thus it involves only few additions and multiplications over the base field $\mathbb{F}_p$ (see example 1.5.7), whereas multiplying two arbitrary field elements is more expensive in terms of number of operations over $\mathbb{F}_p$ involved and in terms of implementation.

- Division can be handled by first computing the inverse and then multiplying by it. It is the most expensive operation in $\mathbb{F}_q$ because it adds to the multiplication complexity the time of an inverse calculation. The inversion in $\mathbb{F}_q$ is a quite complicated operation: a direct approach is to use a table of $mp^m$ positions in which the inverses of the field elements are stored, but there are also alternative methods that can be more advantageous for decoder where codes are defined over $\mathbb{F}_q$ for different values of $q$. For example there are several methods for computing an inverse based on the Extended Euclidean Algorithm. See [BHLN94] for details.

**Example 1.5.7.** Suppose we want to multiply a field element $\beta$ in $\mathbb{F}_{2^4}$ by the primitive element $\alpha$ whose minimal polynomial is $f(x) = x^4 + x + 1$. The element $\beta$ can be expressed as $\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$, with $b_i \in \mathbb{F}_2$. Thus

$$\beta \cdot \alpha = b_0\alpha + b_1\alpha^2 + b_2\alpha^3 + b_3 \alpha^4 = b_3 + (b_0 + b_3)\alpha + b_1\alpha^2 + b_2\alpha^3$$
This multiplication can be carried out by one shift of the feedback register shown in figure 1.1 with the cost of only one addition in the base field \( \mathbb{F}_2 \). The circle represents an adder over \( \mathbb{F}_2 \), while the rectangular elements are storage devices.

First the vector representation \((b_0, b_1, b_2, b_3)\) of \( \beta \) is loaded into the register. Then the register is pulsed: all the stored elements shift right and the element \( b_3 \) goes from the fourth storage device into the first one. At the same time, the element \( b_0 \) is moved into the adder where it is added with \( b_3 \). The sum is stored in the second device. The final configuration contains the vector representation of \( \beta \alpha \) (see figure 1.2).

At this point, we can outline the general procedure of the decoding algorithms for Reed-Solomon codes, which consists of four major steps:

**Step 1. Computation of the syndromes.** Since for any \( i \in \{1, 2, \ldots, d-1\} \)
\( S_i = r(\alpha^i) \), this first task may be accomplished using Horner’s method for evaluating polynomials. This technique is based on the polynomial decomposition given by
\[
 r_{n-1}x^{n-1} + \cdots + r_1x + r_0 = \left( \cdots (r_{n-1}x + r_{n-2})x + \cdots \right)x + r_0
\]
that allows to compute \( S_j \) gradually as \( r_i \)’s are received, as it is shown by the following calculations:
Input: $r = (r_0, r_1, \ldots, r_{n-1})$, $\alpha^j$;
Output: $S_j$;

Begin
$S_j := r_{n-1}$;
for $i = n - 2, n - 3, \ldots, 0$ do
  $S_j := \alpha^j \cdot S_j + r_i$;
endfor;
End

We note that all syndromes can be computed simultaneously and that $r_{n-1}$ is the first received symbol. Thus after $r_0$ is received, all $d - 1$ syndrome computations are completed at the same time.

Figure 1.3 represents a circuit that after $n$ shifts contains $S_i = r(\alpha^i)$. The vector $r = (r_0, r_1, \ldots, r_{n-1})$ is shifted into the circuit one component at a time. After the first shift, the storage device contains $r_{n-1}$, after the second it contains $r_{n-1} \alpha^i + r_{n-2}$ and so on.

![Figure 1.3: circuit for computing $r(\alpha^i)$](image)

**Step 2. Determination of the error-locator polynomial and of the number of errors that occurred.** This is done exploiting the characterization of $\sigma(x)$ due to the key equation (1.4). In chapters 2 and 3 we will study two different strategies to accomplish this point.

**Step 3. Finding the error positions.** Recalling that the elements $X_i = \alpha^{p_i}$ are the inverses of the roots of $\sigma(x)$, the error positions $p_1, p_2, \ldots, p_e$ are computed using Chien’s search. This is an exhaustive search over all the elements in $\mathbb{F}_q$ that finds the roots of

$$
\sigma(x) = \sigma_e x^e + \sigma_{e-1} x^{e-1} + \cdots + \sigma_1 x + 1
$$
based on the fact that
\[ \alpha^i \text{ is a root of } \sigma(x) \iff 1 + \sum_{j=1}^{e} \sigma_j \alpha^{ij} = 0 \]

Thus if \( i \) goes from 1 to \( n - 1 \), at each step it is necessary only to multiply the \( j \)th addend \( \sigma_j \alpha^{ij} \) by \( \alpha^i \) in order to obtain the next set of addends. Chien’s search procedure can be summarized as follow:

**Algorithm 1.5.8** (Chien’s search for error positions).

**Input:** the length of the code \( n \), the error number \( e \) and the coefficients of \( \sigma(x) = \sigma_e x^e + \sigma_{e-1} x^{e-1} + \cdots + \sigma_1 x + 1 \).

**Output:** the error positions \( p_1 < p_2 < \cdots < p_e \).

**Begin**

\( k := e \);

for \( i = 1, 2, \ldots, n \) do

\[ \sigma_l := \sigma_l \cdot \alpha^l \text{ for any } l = 1, 2, \ldots, e; \]

\( S := 1 + \sum_{j=1}^{e} \sigma_j; \)

if \( S = 0 \) then

\( p_k := n - i; \)

\( k := k - 1; \)

endif

endfor

If \( k \neq 0 \) then declare a decoder malfunction;

**End**

Chien’s search can be implemented in a single circuit (see figure 1.4) with \( t \) multipliers for multiplying by \( \alpha, \alpha^2, \ldots, \alpha^t \) respectively and few adders, with a computational complexity upper bounded by \( n \) multiplications and \( en \) additions. More details are given in \[Chi64\].

**Step 4. Finding the error values.**

This task can be accomplished using Forney’s formula \[1.2\], but throughout this thesis we will also see other alternative methods to compute error values. We observe that, depending on the decoding algorithm used, this last step may be implemented not only after the step 3, but also during the step 3: each time that Chien’s search finds an error position \( p_i \), the decoding algorithm corrects the \( p_i \)th components of \( r \) computing \( E_i \) immediately.
It is important to note that steps 1 and 3 of the general procedure involve only additions and multiplications by fixed elements (the first $d - 1$ powers of $\alpha$) and their implementation is simple, so these steps requires a negligible computational costs compared with steps 2 and 4. Moreover steps 1 and 3 are essentially the same in all the decoding algorithms that we will study in this thesis. For these reasons they will be not discussed over and they will be not counted in estimating of the computational cost of the decoding algorithms that we will analyze later on.
Chapter 2

Peterson-Gorenstein-Zierler Decoder

W. Wesley Peterson in 1960 developed the first practical decoder for Reed-Solomon codes based on syndrome computation only using tools of linear algebra. In particular, Peterson developed an algorithm for binary BCH codes that finds the location of the errors solving a linear system and Daniel E. Gorenstein and Neal Zierler, a year later, extended this algorithm for nonbinary codes.

In this chapter we will analyze the Peterson-Gorenstein-Zierler decoding algorithm and we will present an important improvement \[\text{SF97}\] obtained using a fast inversion method of an Hankel matrix. Finally we will describe the necessary and sufficient conditions to avoid decoder malfunctions in the fast implementation of the Peterson-Gorenstein-Zierler decoder.

As discussed in section \[1.5\] while studying the Peterson-Gorenstein-Zierler decoding algorithm we will focus our attention only in how it carries out step 2 (computing the error-locator polynomial) and step 4 (computing the error values) of the general outline.

2.1 PGZ Decoding Algorithm

Consider a code \(RS(n, d, \alpha)\) over the field \(\mathbb{F}_q\) and let \(c, e\) and \(r = c + e\) be the codeword sent, the error vector and the word received in a data transmission, like in section \[1.5\]. We recall that \(t\) represents the error correction capability of the code \(RS(n, d, \alpha)\) (i.e. \(t = \left\lfloor \frac{d-1}{2} \right\rfloor\)) and from now on we assume that \(e = \text{wt}(e) \leq t\). If \(e > t\) we do not expect to be able to correct the errors. By remark \[1.5.2\] we can express the error-locator polynomial \(\sigma(x)\) and the
The validity of the key equation (1.4) for the polynomials \( \sigma(x) \) and \( \omega(x) \) implies that the coefficients of \( x^e, x^{e+1}, \ldots, x^{d-2} \) in the polynomial \( \sigma(x)S(x) \) are equal to zero. Thus we have the following linear system of \( d - 1 - e \) equations for the unknowns \( \sigma_1, \sigma_2, \ldots, \sigma_e \):

\[
\begin{align*}
\sum_{i=1}^{e} S_{e+1-i} \sigma_i + S_{e+1} &= 0 \\
\sum_{i=1}^{e} S_{e+2-i} \sigma_i + S_{e+2} &= 0 \\
& \quad \vdots \\
\sum_{i=1}^{e} S_{d-1-i} \sigma_i + S_{d-1} &= 0
\end{align*}
\]

(2.1)

whose solution (if it exists and is unique) determines the coefficients of the error-locator polynomial \( \sigma(x) \). The linear system (2.1) may be expressed in matrix form as

\[
\begin{pmatrix}
S_1 & S_2 & \cdots & S_e \\
S_2 & S_3 & \cdots & S_{e+1} \\
\vdots & \vdots & \ddots & \vdots \\
S_{d-1-e} & S_{d-e} & \cdots & S_{d-2}
\end{pmatrix}
\begin{pmatrix}
\sigma_e \\
\sigma_{e-1} \\
\vdots \\
\sigma_1
\end{pmatrix}
= -
\begin{pmatrix}
S_{e+1} \\
S_{e+2} \\
\vdots \\
S_{d-1}
\end{pmatrix}
\]

(2.2)

and in order to study its solutions, we define the following useful matrices:

**Definition 2.1.1.** Let \( t \) be the error correction capability of the code \( RS(n, d, \alpha) \), we call

\[
A = \begin{pmatrix}
S_1 & S_2 & \cdots & S_{t+1} \\
S_2 & S_3 & \cdots & S_{t+2} \\
\vdots & \vdots & \ddots & \vdots \\
S_{d-1-t} & S_{d-t} & \cdots & S_{d-1}
\end{pmatrix}
\]

the **syndrome matrix** and we indicate with \( A_i \) its \( i \times i \) leading principal minor. That is

\[
A_i = \begin{pmatrix}
S_1 & S_2 & \cdots & S_i \\
S_2 & S_3 & \cdots & S_{i+1} \\
\vdots & \vdots & \ddots & \vdots \\
S_i & S_{i+1} & \cdots & S_{2i-1}
\end{pmatrix}
\]
2.1. PGZ DECODING ALGORITHM

In the following proposition we study the properties of the syndrome matrix and of its minors $A_i$:

**Proposition 2.1.2.** Let $t$ be the error correction capability of $RS(n,d,\alpha)$ and $e = \text{wt}(e)$. If $e \leq t$, then

(1) $A_e$ is a non-singular matrix;

(2) the rank of the syndrome matrix $A$ is equal to $e$;

(3) if $\rho = \max\{i \leq t \mid \det(A_i) \neq 0\}$, then the rank of $A$ is equal to $\rho$. So we have $e = \rho$.

**Proof.**

1. We consider the matrices

$$V = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ X_1 & X_2 & \cdots & X_e \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{e-1} & X_2^{e-1} & \cdots & X_e^{e-1} \end{pmatrix}, \quad D = \begin{pmatrix} 0 & E_1X_1 & \cdots & 0 \\ 0 & E_2X_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & E_eX_e \end{pmatrix}$$

Since the values $X_i$'s are all distinct and different from zero and the value $E_i$'s are all different from zero, the Vandermonde matrix $V$ and the diagonal matrix $D$ are both non-singular. The proof can be easily concluded verifying that $A_e = VDV^T$ by (1.3).

2. Since the matrix $A_e$ is a submatrix of $A$, it follows from (1) that $\text{rk}(A) \geq e$. We consider the matrices

$$V_1 = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ X_1 & X_2 & \cdots & X_e \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{d-2-t} & X_2^{d-2-t} & \cdots & X_e^{d-2-t} \end{pmatrix}, \quad V_2 = \begin{pmatrix} 1 & X_1 & \cdots & X_1^t \\ 1 & X_2 & \cdots & X_2^t \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_e & \cdots & X_e^t \end{pmatrix}$$

It is easy to verify that $A = V_1DV_2$. Recalling that the rank of a product of matrices is less or equal to the rank of each factor matrix and observing that $\text{rk}(D) = e$, we obtain that $\text{rk}(A) \leq e$. So the proof is complete.

3. Since, by definition, $\det(A_\rho) \neq 0$, we have that $\text{rk}(A) \geq \rho$. On the other hand, we saw in (1) that $\det(A_e) \neq 0$, therefore we have $e \leq \rho$. Hence, from (2) it follows that $\text{rk}(A) \leq \rho$ and this concludes the proof. \qed
An immediate consequence is that, given the syndrome matrix $A$, we can calculate the number of errors which occurred $e$ as

$$e = \max\{i \leq t \mid \det(A_i) \neq 0\}$$

Furthermore by proposition 2.1.2 we know that there exists a unique solution of the linear system (2.1) and the first $e$ equations are sufficient to calculate it. In other words, the coefficients $\sigma_1, \sigma_2, \ldots, \sigma_e$ of the error-locator polynomial are completely determined by solving

$$A_e \begin{pmatrix} \sigma_e \\ \sigma_{e-1} \\ \vdots \\ \sigma_1 \end{pmatrix} = - \begin{pmatrix} S_{e+1} \\ S_{e+2} \\ \vdots \\ S_{2e} \end{pmatrix} \quad (2.3)$$

Once that the error-locator polynomial is known, we will calculate its roots and the error positions $p_1, p_2, \ldots, p_e$ using (1.1) and Chien’s search as seen in section 1.5.

At this stage it remains to calculate the error values, knowing their positions. For this aim we consider the relations (1.3) about the syndromes $S_1, S_2, \ldots, S_{d-1}$. As the elements $X_1, X_2, \ldots, X_e$ are known, we obtain a set of $d - 1$ linear equations in the unknowns $E_1, E_2, \ldots, E_e$:

$$\begin{cases} 
S_1 = \sum_{j=1}^{e} E_j X_j \\
S_2 = \sum_{j=1}^{e} E_j X_j^2 \\
\vdots \\
S_{d-1} = \sum_{j=1}^{e} E_j X_j^{d-1} 
\end{cases}$$

The first $e$ equations can be solved for the error values $E_i$’s considering the following linear system:

$$\begin{pmatrix} X_1 & X_2 & \cdots & X_e \\
X_1^2 & X_2^2 & \cdots & X_e^2 \\
\vdots & \vdots & \ddots & \vdots \\
X_1^e & X_2^e & \cdots & X_e^e \end{pmatrix} \begin{pmatrix} E_1 \\ E_2 \\ \vdots \\ E_e \end{pmatrix} = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_e \end{pmatrix} \quad (2.4)$$
2.1. *PGZ DECODING ALGORITHM*

The coefficient matrix of linear system (2.4) has a special structure, indeed it is the product of a Vandermonde matrix and a diagonal matrix:

\[
\begin{pmatrix}
X_1 & X_2 & \cdots & X_e \\
X_1^2 & X_2^2 & \cdots & X_e^2 \\
\vdots & \vdots & \ddots & \vdots \\
X_1^e & X_2^e & \cdots & X_e^e
\end{pmatrix}
= \begin{pmatrix}
1 & 1 & \cdots & 1 \\
X_1 & X_2 & \cdots & X_e \\
\vdots & \vdots & \ddots & \vdots \\
X_1^{e-1} & X_2^{e-1} & \cdots & X_e^{e-1}
\end{pmatrix}
\begin{pmatrix}
1 & 0 & \cdots & 0 \\
0 & X_2 & \ddots & \vdots \\
\vdots & \ddots & \ddots & \vdots \\
0 & \cdots & 0 & X_e
\end{pmatrix}
\]

Since the $X_i$’s are distinct and nonzero by definition, both the Vandermonde matrix $V$ and the diagonal matrix $X$ are invertible. Thus,

\[
\begin{pmatrix}
E_1 \\
E_2 \\
\vdots \\
E_e
\end{pmatrix}
= X^{-1}V^{-1}
\begin{pmatrix}
S_1 \\
S_2 \\
\vdots \\
S_e
\end{pmatrix}
\]

Since $V$ is a Vandermonde matrix, the step $V^{-1}s$ can be efficiently carried out using the *Björck-Pereyra algorithm* for the solution of Vandermonde systems, while the step $X^{-1}(V^{-1}s)$ consists simply in $e$ divisions. The Björck-Pereyra algorithm transforms the right side vector of a Vandermonde system by a sequence of simple transformations into the solution vector. The components are modified in a suitable order so that no extra storage is needed. For more details and a complete exposition of the Björck-Pereyra algorithm, we refer to [BP70]. In the following we present an algorithm that first compute the vector $y = (y_0, y_1, \ldots, y_{e-1})^T \in (\mathbb{F}_q)^e$ such that $y = V^{-1}s$ (implementing the Björck-Pereyra algorithm for the matrix $V$ and the right side vector $s$) and after it solves the diagonal system.

**Algorithm 2.1.3 (BP algorithm for computing error values).**

Input: the number of error $e$, the elements $X_1, X_2, \ldots, X_e$ and the syndrome vector $s = (S_1, S_2, \ldots, S_e)^T$;

Output: the error values $E_1, E_2, \ldots, E_e$;

Begin

**BP.1 (finding $y = V^{-1}s$)**

\[
y := s;
\]

for $k = 1, 2, \ldots, e - 1$ do

for $i = e - 1, e - 2, \ldots, k$ do
\[ y_i := y_i - X_k y_{i-1}; \]
endfor
endfor

for \( k = e - 1, e - 2, \ldots, 1 \) do
  for \( i = k, k + 1, \ldots, e - 1 \) do
    \[ y_i := \frac{y_i}{X_{i+1} - X_{i+1-k}}; \]
  endfor
  for \( i = k - 1, k, \ldots, e - 2 \) do
    \[ y_i := y_i - y_{i+1}; \]
  endfor
endfor

\textit{BP.2 (solving the diagonal system)}

for \( i = 1, 2, \ldots, e \) do
  \[ E_i := y_{i-1} X_i^{-1}; \]
endfor

End

By direct counting, it is easy to verify that algorithm 2.1.3 has a computational time complexity of \( O(e^2) \) arithmetic operations.

With this observations, the Peterson-Gorenstein-Zierler (PGZ) decoding algorithm consists of the followings steps:

\textbf{Algorithm 2.1.4 (PGZ decoding algorithm) for } RS(n, d, \alpha).

\textbf{Input:} the received word \( r(x) \);

\textbf{Output:} the codeword \( c(x) \);

\textbf{Begin}

\textbf{PGZ.1 (syndrome computation)}
for \( i = 1, 2, \ldots, d - 1 \) do
  \[ S_i := r(\alpha^i); \]
endfor

\textbf{PGZ.2 (error-locator polynomial computation)}
\[ i := \left\lfloor \frac{d}{2} \right\rfloor; \]
\[ \text{det} := \det(A_i); \]
while $(\det = 0)$ repeat
    $i := i - 1$;
    $\det := \det(A_i)$;
endwhile;

$e := i$;
solve the linear system (2.3) to find $\sigma_1, \sigma_2, \ldots, \sigma_e$;

$\sigma(x) := \sigma_e x^e + \sigma_{e-1} x^{e-1} + \cdots + \sigma_1 x + 1$;

**PGZ.3** (finding error positions)
calculate the error positions $p_1, p_2, \ldots, p_e$ and the elements $X_1^{-1}, X_2^{-1}, \ldots, X_e^{-1}$ using Chien’s search (alg. 1.5.8);

**PGZ.4** (finding error values)
solve linear system (2.4) with algorithm 2.1.3

$$
\begin{pmatrix}
E_1 \\
E_2 \\
\vdots \\
E_e
\end{pmatrix}
= X^{-1} V^{-1}
\begin{pmatrix}
S_1 \\
S_2 \\
\vdots \\
S_e
\end{pmatrix}
$$

Return $c(x) := r(x) - \sum_{i=1}^{e} E_i x^{p_i}$;
End

The correctness of the PGZ decoding algorithm follows immediately from proposition 2.1.2. We are now interested in estimating its computational cost. The calculation of $e$ in PGZ is done by testing the non-singularity of $A_i$, beginning from $i = t$ and decrementing $i$. Since the computational complexity of $\det(A_i)$ is $O(i^3)$ and

$$
\sum_{i=1}^{t} i^3 = \left( \frac{t(t+1)}{2} \right)^2 - \left( \frac{(e-1)e}{2} \right)^2
$$

then calculating $e$ has a computational cost of order $O(t^4)$. The computational cost of finding the coefficients $\sigma_1, \sigma_2, \ldots, \sigma_e$ is the cost of solving an $e \times e$ linear system, that is $O(e^3)$, whereas the linear system solved in PGZ has a computational complexity of $O(e^3)$ operations due to algorithm 2.1.3. However the total computational time complexity of PGZ decoding algorithm is of order $O(t^4)$ and this makes it not applicable for large $t$.

We also observe that in PGZ decoder the steps 3 and 4 of the general outline are sequential. Indeed step PGZ can be implemented only after
the end of PGZ, because algorithm 2.1.3 used in PGZ to calculate the error values needs all the elements $X_i$'s together. In chapter 4 we will see a modified version of PGZ decoder with a different strategy for the error value computation.

2.2 fPGZ Decoding Algorithm

Although the PGZ decoding algorithm 2.1.4 involves simple linear algebra, its implementation is computationally expensive in general. For this reason in this section we will study and formalize an alternative implementation of the PGZ decoder, shown in SF97, in which some special proprieties of the syndrome matrix are used to decrease the computational cost of finding the number of error $e$ and solving the linear system (2.3) (step PGZ 2).

We observe that the syndrome matrix $A$ is a Hankel matrix, that is a matrix with constant positive sloping diagonals. Techniques for fast inversion of Hankel matrices are well known (see HR85) and this section we use them to calculate the rank of the syndrome matrix $A$ and the solutions of the linear system (2.3). For a more convenient notation, we will indicate with $0^{(j)}$ the zero vector of $j$ components, for any $j \in \mathbb{N}$.

The algorithm that we present is iterative and is based on the following idea: if for an index $i \leq t$ we have $\det(A_i) \neq 0$ and we already know the vectors $w^{(i)}$ and $y^{(i)}$ in $(\mathbb{F}_q)^i$ such that

$$A_i w^{(i)} = \begin{pmatrix} S_{i+1} \\ S_{i+2} \\ \vdots \\ S_{2t} \end{pmatrix}$$
($2.5$)

$$A_i y^{(i)} = 0^{(i-1)}$$
($2.6$)

then, using the Hankel form of the syndrome matrix, we can calculate the first index $j > i$ such that $\det(A_j) \neq 0$ and the vectors $w^{(j)}$ and $y^{(j)}$ in $(\mathbb{F}_q)^j$ such that the systems (2.5) and (2.6) also hold for $i = j$. Clearly, if we are able to do this step, repeating it, we can implement an iterative algorithm which calculates the maximum index $\rho \leq t$ such that $\det(A_\rho) \neq 0$ and the vectors $w^{(\rho)}$ and $y^{(\rho)}$ that solve (2.5) and (2.6) for $i = \rho$. As we saw in proposition 2.1.2 supposing $e \leq t$, we have that the index $\rho$ is equal to $e$. So the vector $w^{(\rho)}$ given as output is the unique vector that satisfies the linear
system (2.3). In other words

\[ \mathbf{w}(\rho) = \begin{pmatrix} \sigma_e \\ \sigma_{e-1} \\ \vdots \\ \sigma_1 \end{pmatrix} \]

We will call the **fast Peterson-Gorenstein-Zierler** (fPGZ) **decoding algorithm** the decoding algorithm obtained by solving the linear system (2.3) with this iterative procedure.

In order to study the iterative step on which is based the fPGZ decoding algorithm, we need the following definitions and a property of Hankel matrices.

**Definition 2.2.1.** Let \( i \leq t \) such that \( \det(A_i) \neq 0 \). If

\[ \mathbf{w}^{(i)} = \begin{pmatrix} w_0^{(i)} \\ w_1^{(i)} \\ \vdots \\ w_{i-1}^{(i)} \end{pmatrix} \in (\mathbb{F}_q)^i \]

is the solution of system (2.3), then for any \( j \in \{1, 2, \ldots, t - i\} \) we define

\[ \varepsilon_j = S_{i+j}w_0^{(i)} + S_{i+j+1}w_1^{(i)} + \cdots + S_{2i+j-1}w_{i-1}^{(i)} + S_{2i+j} \in \mathbb{F}_q \]

Furthermore if the elements \( \varepsilon_j \)'s are not all equal to zero, then we call

\[ r = \min\{j \in \{1, 2, \ldots, t - i\} | \varepsilon_j \neq 0\} \in \mathbb{N}^+ \]

the **singularity gap** of the step \( i \).

The following lemma is stated and proved in section 1 chapter 1 of [HR85] for Hankel matrices with entries in \( \mathbb{C} \), but it is easy to verify that the same proof also holds for Hankel matrices with entries in the field \( \mathbb{F}_q \).

**Lemma 2.2.2.** Let

\[ M = \begin{pmatrix} m_1 & m_2 & \cdots & m_i \\ m_2 & m_3 & \cdots & m_{i+1} \\ \vdots & \vdots & \ddots & \vdots \\ m_i & m_{i+1} & \cdots & m_{2t-1} \end{pmatrix} \in (\mathbb{F}_q)^{t \times t} \]
a general $l \times l$ Hankel matrix. If there exist $m \in \mathbb{F}_q$, $w \in (\mathbb{F}_q)^l$ and $y \in (\mathbb{F}_q)^l$ such that

\[
Mw = \begin{pmatrix} m_{l+1} \\ m_{l+2} \\ \vdots \\ m_{2l-1} \\ m \\
\end{pmatrix} \quad \text{and} \quad My = \begin{pmatrix} 0^{(i-1)} \\ 1 \\
\end{pmatrix}
\]

then $M$ is non-singular.

At this stage we can enunciate the theorem that shows formally how the iterative step of fPGZ decoding algorithm works. The proof of this theorem is a constructive proof and gives the formulas necessary to implement the algorithm.

**Theorem 2.2.3.** Let $i$ be an index less or equal to $t$ such that $\det(A_i) \neq 0$. Let $w^{(i)}$ and $y^{(i)}$ be the vectors in $(\mathbb{F}_q)^l$ that solve respectively (2.5) and (2.6). Recalling the notation of definition 2.2.1, we have that:

1. If $\varepsilon_j = 0$ for all $j \in \{1, 2, \ldots, t-i\}$, then

   \[
i = \max\{j \leq t \mid \det(A_j) \neq 0\}
\]

2. If there exists $j \in \{1, 2, \ldots, t-i\}$ such that $\varepsilon_j \neq 0$ and $r$ is the singularity gap, then we have that

   \[
r = \min\{j \in \{1, 2, \ldots, t-i\} \mid \det(A_{i+j}) \neq 0\}
\]

**Proof.** By the linear system (2.5) and by definition 2.2.1 it follows that for any $j \in \{1, 2, \ldots, t-i\}$:

\[
A_{i+j} \begin{pmatrix} w^{(i)} \\ 1 \\ 0^{(j-1)} \\
\end{pmatrix} = \begin{pmatrix} 0^{(i)} \\ \varepsilon_1 \\ \vdots \\ \varepsilon_j \\
\end{pmatrix}
\]

If the $\varepsilon_j$'s are all equal to zero, then $\ker(A_{i+j}) \neq \{0\}$ for any $j \in \{1, 2, \ldots, t-i\}$. In particular $\det(A_{i+j}) = 0$ for any $j \in \{1, 2, \ldots, t-i\}$ and hence (1) is proved. If not all the $\varepsilon_j$'s are zero, then by definition of $r$ we have that $\varepsilon_1 = \varepsilon_2 = \cdots = \varepsilon_{r-1} = 0$. Hence $\det(A_{i+j}) = 0$ for any $j \in \{1, 2, \ldots, r-1\}$ and to complete the proof of (2) it is sufficient to show that $\det(A_{i+r}) \neq 0$. 


By lemma [2.2.2], it is sufficient to show the existence of vectors \( w^{(i+r)} \) and \( y^{(i+r)} \) in \((\mathbb{F}_q)^{i+r}\) such that

\[
A_{i+r}w^{(i+r)} = -\begin{pmatrix} S_{i+r+1} \\ S_{i+r+2} \\ \vdots \\ S_{2i+2r} \end{pmatrix}
\]  
(2.7)

\[
A_{i+r}y^{(i+r)} = \begin{pmatrix} 0^{(i+r-1)} \\ 1 \end{pmatrix}
\]  
(2.8)

Since \( A_{i+r} \begin{pmatrix} w^{(i)} \\ 1 \\ 0^{(r-1)} \end{pmatrix} = \begin{pmatrix} 0^{(i+r-1)} \\ \varepsilon_r \end{pmatrix} \) and \( \varepsilon_r \neq 0 \), it follows that the vector

\[
y^{(i+r)} = \frac{1}{\varepsilon_r} \begin{pmatrix} w^{(i)} \\ 1 \\ 0^{(r-1)} \end{pmatrix} \in (\mathbb{F}_q)^{i+r}
\]  
(2.9)

is well defined and satisfies the system (2.8). Moreover we define the vectors \( a^{(0)}, a^{(1)}, \ldots, a^{(r-1)} \in (\mathbb{F}_q)^{i+r} \) as

\[
\begin{cases}
a^{(0)} = y^{(i+r)} \\
a^{(j)} = \begin{pmatrix} 0 \\ a_0^{(j-1)} \\ a_1^{(j-1)} \\ \vdots \\ a_{i+j-1}^{(j-1)} \\ 0^{(r-j-1)} \end{pmatrix} - \alpha_j a^{(0)} & \forall j = 1, 2, \ldots, r - 1
\end{cases}
\]  
(2.10)

where

\[
\alpha_j = S_{i+r+1}a_0^{(j-1)} + S_{i+r+2}a_1^{(j-1)} + \cdots + S_{2i+r+j}a_{i+j-1}^{(j-1)}
\]  
(2.11)

It can be easily proved by induction that, for any \( j \in \{0, 1, \ldots, r - 1\} \), the product \( A_{i+r}a^{(j)} \) is equal to the vector with all the components zero except the component \( i + r - j \), which is equal to 1. Indeed the base case is implied by (2.8) and if

\[
A_{i+r}a^{(j)} = \begin{pmatrix} 0^{(i+r-j-1)} \\ 1 \\ 0^{(j)} \end{pmatrix}
\]
then

\[
A_{i+r}a^{(j+1)} = A_{i+r} \begin{pmatrix}
0 \\
a_0^{(j)} \\
a_1^{(j)} \\
\vdots \\
a_{i+j}^{(j)} \\
0^{(r-j-2)}
\end{pmatrix} - \alpha_{j+1} A_{i+r} a^{(0)} = \\
\begin{pmatrix}
S_2 & S_3 & \cdots & S_{i+j+2} \\
S_3 & S_4 & \cdots & S_{i+j+3} \\
\vdots & \vdots & \ddots & \vdots \\
S_{i+r} & S_{i+r+1} & \cdots & S_{2i+j+r}
\end{pmatrix} \begin{pmatrix}
a_0^{(j)} \\
a_1^{(j)} \\
\vdots \\
a_{i+j}^{(j)}
\end{pmatrix} - \alpha_{j+1} A_{i+r} y^{(i+r)} \overset{\text{inductive hypothesis}}{=} \\
\begin{pmatrix}
0^{(i+r-j-2)} \\
1 \\
\alpha_{j+1}
\end{pmatrix} - \alpha_{j+1} \begin{pmatrix}
0^{(i+r-1)} \\
1
\end{pmatrix} = \begin{pmatrix}
0^{(i+r-j-2)} \\
1 \\
0^{(j+1)}
\end{pmatrix}
\]

In a similar way, we recursively define the vectors \( b^{(0)}, b^{(1)}, \ldots, b^{(r)} \) in \( (\mathbb{F}_q)^i \) as

\[
\begin{aligned}
b^{(0)} &= w^{(i)} \\
b^{(j)} &= \begin{pmatrix}
0 \\
b_0^{(j-1)} \\
b_1^{(j-1)} \\
\vdots \\
b_{i-2}^{(j-1)}
\end{pmatrix} - b_{i-1}^{(j-1)} w^{(i)} - \beta_j y^{(i)} & \forall j = 1, 2, \ldots, r 
\end{aligned}
\tag{2.12}
\]

where

\[
\beta_j = S_{i+1} b_0^{(j-1)} + S_{i+2} b_1^{(j-1)} + \cdots + S_{2i} b_{i-1}^{(j-1)} + S_{2i+j} \tag{2.13}
\]

It can be proved, again by induction, that for any \( j \in \{0, 1, \ldots, r\} \) we have

\[
A_i b^{(j)} = \begin{pmatrix}
S_{i+j+1} \\
S_{i+j+2} \\
\vdots \\
S_{2i+j}
\end{pmatrix}
\]
Finally, for any $j \in \{1, 2, \ldots, r\}$ we define

$$\gamma_j = S_{i+j}b_0^{(r)} + S_{i+j+1}b_1^{(r)} + \cdots + S_{2i+j-1}b_{i-1}^{(r)}$$  \hspace{1cm} (2.14)$$

and we observe that

$$A_{i+r} \begin{pmatrix} b^{(r)} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} -S_{i+r+1} \\ -S_{i+r+2} \\ \vdots \\ -S_{2i+r} \\ \gamma_1 \\ \vdots \\ \gamma_r \end{pmatrix}$$

Thus we conclude the proof defining the vector $w^{(i+r)}$ as

$$w^{(i+r)} = \begin{pmatrix} b^{(r)} \\ 0 \end{pmatrix} - \sum_{l=0}^{r-1} [\gamma_{r-l} + S_{2i+2r-l}] a^{(l)}$$  \hspace{1cm} (2.15)$$

In order to work, the $fPGZ$ decoding algorithm needs a base step. For this reason we observe that:

**Remark 2.2.4.** If $e = \text{wt}(e) \leq t$, then either $S_1 = S_2 = \cdots = S_{d-1} = 0$ or there exists $i \leq e$ such that $S_i \neq 0$. In fact, if not all the syndromes were zero and $S_1 = S_2 = \cdots = S_e = 0$ held, then we would have at the same time that $\det(A_e) \neq 0$ (by proposition 2.1.2) and that all the entries of the first row of $A_e$ are equal to zero. This obviously is absurd.

Thus we can define

$$i_0 = \min \{ j \in \mathbb{N}^+ \mid S_j \neq 0 \}$$

and we have that

$$A_{i_0} = \begin{pmatrix} 0 & \cdots & 0 & 0 & S_{i_0} \\ 0 & \cdots & 0 & S_{i_0} & S_{i_0+1} \\ \vdots & \ddots & \vdots & \vdots & \vdots \\ S_{i_0} & \cdots & S_{2i_0-3} & S_{2i_0-2} & S_{2i_0-1} \end{pmatrix}$$

is a non-singular lower triangular matrix. So

$$y^{(i_0)} = \begin{pmatrix} S_{i_0-1}^{-1} \\ 0^i_{i-1} \end{pmatrix}$$
satisfies (2.6) for \( i = i_0 \) and the vector \( w^{(i_0)} \), which satisfies (2.5) for \( i = i_0 \), can be calculated with linear algebra algorithms for triangular matrix of complexity \( O((i_0)^2) \).

We are now ready to summarize the \( fPGZ \) decoding algorithm as follows. Since when \( r = 1 \) the formula (2.15) to calculate \( w^{(i+r)} \) is simpler than in the general case, in the following decoding algorithm we prefer to consider separately the case \( \varepsilon_1 \neq 0 \).

**Algorithm 2.2.5 (\( fPGZ \) decoding algorithm for \( RS(n, d, \alpha) \)).**

**Input:** the received word \( r(x) \);

**Output:** the codeword \( c(x) \);

**Begin**

\( fPGZ.1 \) (syndrome computation)

for \( i = 1, 2, \ldots, d - 1 \) do

\[
S_i := r(\alpha^i);
\]

endfor

\( fPGZ.2 \) (error-locator polynomial computation)

(a) (base step)

\[
t := \left\lfloor \frac{d-1}{2} \right\rfloor;
\]

\[
i_0 := 1;
\]

while \((S_{i_0} = 0 \text{ and } i_0 \leq d - 1)\) repeat

\[
i_0 := i_0 + 1;
\]

endwhile

if \( i_0 = d \) then return \( r \);

if \( i_0 > t \) then declare a failure;

calculate \( w^{(i_0)} \) and \( y^{(i_0)} \);

(b) (iterative procedure)

\[
i := i_0;
\]

while \((i < t)\) repeat

\[
\varepsilon_1 := S_{i+1}w_0^{(i)} + S_{i+2}w_1^{(i)} + \cdots + S_{2i}w_i^{(i)} + S_{2i+1};
\]

if \((\varepsilon_1 \neq 0)\) then

\[
y^{(i+1)} := \frac{1}{\varepsilon_1} \begin{pmatrix} w_0^{(i)} \\ 1 \end{pmatrix};
\]

endwhile

**End**
2.2. FPGZ DECODING ALGORITHM

\[ \eta := S_{i+1}y_0^{(i)} + S_{i+2}y_1^{(i)} + \cdots + S_{2i}y_{i-1}^{(i)}; \]
\[ \varepsilon_2 := S_{i+2}w_0^{(i)} + S_{i+3}w_1^{(i)} + \cdots + S_{2i+1}w_{i-1}^{(i)} + S_{2i+2}; \]
\[ w^{(i+1)} := \begin{pmatrix} 0 \\ y^{(i)} \end{pmatrix} - \varepsilon_1 \begin{pmatrix} \varepsilon_1^{(i)} \\ 0 \end{pmatrix} + (\varepsilon_1\eta - \varepsilon_2)y^{(i+1)}; \]
\[ i := i + 1; \]

else
\[ r := 1; \]
while (\( \varepsilon_r = 0 \)) repeat
\[ r := r + 1; \]
if \( r > t - i \) then
  go to step c;
else
\[ \varepsilon_r := S_{i+r}w_0^{(i)} + S_{i+r+1}w_1^{(i)} + \cdots + S_{2i+r-1}w_{i-1}^{(i)} + S_{2i+r}; \]
endwhile
\[ y^{(i+r)} := \frac{1}{\varepsilon_r} \begin{pmatrix} w^{(i)} \\ 1 \\ 0^{(r-1)} \end{pmatrix}; \]
calculate vectors \( a^{(j)} \)'s using (2.10);
calculate vectors \( b^{(j)} \)'s using (2.12);
l := 1;
for \( i = 1, 2, \ldots, r \)
\[ \gamma_l := S_{i+l}b_0^{(r)} + S_{i+l+1}b_1^{(r)} + \cdots + S_{2i+l-1}b_{i-1}; \]
endfor
\[ w^{(i+r)} := \begin{pmatrix} b^{(r)}_0 \\ 0^{(r)} \end{pmatrix} - \sum_{l=0}^{r-1} [\gamma_r - l + S_{2l+2r-l}]a^{(l)}; \]
\[ i := i + r; \]
endwhile

(c) \( e := i; \)
\[ \sigma(x) := w_0^{(i)}x^1 + w_1^{(i)}x^{i-1} + \cdots + w_{i-1}^{(i)}x + 1; \]

\textbf{fPGZ.3} (finding error positions)
calculate the error positions \( p_1, p_2, \ldots, p_e \) and the elements \( X_1^{-1}, X_2^{-1}, \ldots, X_e^{-1} \) using Chien’s search (alg. 1.5.8);

\textbf{fPGZ.4} (finding error values)
solve linear system (2.4) with algorithm 2.1.3
\[
\begin{pmatrix}
E_1 \\
E_2 \\
\vdots \\
E_e
\end{pmatrix}
= X^{-1}V^{-1}
\begin{pmatrix}
S_1 \\
S_2 \\
\vdots \\
S_e
\end{pmatrix}
\]

Return \( c(x) := r(x) - \sum_{i=1}^{e} E_{i}x^{p_{i}}; \)

End

From theorem 2.2.3 and from what we have seen until now, it follows that if \( e \leq t \), then the fPGZ decoding algorithm 2.2.5 correctly gives as output the transmitted codeword \( c \). Now we will estimate its computational complexity. We consider step fPGZ 2 by remark 2.2.4 it follows that the computational cost of (a) is of order \( O(e^2) \), thus we can focus our attention on the iterative step from \( i \) to \( i + r \) in (b):

- the computation of the gap \( r \) and of the elements \( \varepsilon_{j} \)'s needs at most \( ir \) multiplications and \( ir \) additions
- from (2.9) we deduce that the vector \( y^{(i+r)} \) can be calculated from \( w^{(i)} \) with one inversion and \( i \) multiplications;
- from (2.11) we find that the computational cost of the elements \( \alpha_{j} \)'s is upper bounded by \( ir + r^2 \) multiplications and \( ir + r^2 \) additions, while from (2.13) and (2.14) we find that the elements \( \beta_{j} \)'s and \( \gamma_{j} \)'s are computed with at most \( 2ir \) multiplications and \( 2ir \) additions;
- from (2.10) we deduce that the cost of the vectors \( a^{(j)} \)'s is at most \( ir + r^2 \) multiplications and \( ir + r^2 \) additions, while (2.12) implies that the vectors \( b^{(j)} \)'s can be calculated with \( ir \) multiplications and \( ir \) additions;
- finally (2.15) implies that the number of operations necessary to calculate the vector \( w^{(i+r)} \) from the vectors \( b^{(r)} \) and \( a^{(j)} \)'s is upper bounded by \( ir + r^2 \) multiplications and \( ir + r^2 + r + i \) additions.

Considering that \( i, r \leq t \), the iterative step from \( i \) to \( i + r \) requires at most one inversion, \( 9tr + t + r \) multiplications and \( 10tr + t + r \) additions. Evidently the total computational cost of (b) in fPGZ 2 is given by the sum of the costs of the iterative steps and since the sum of the singularity gaps is clearly equal to \( e - i_{0} \) (i.e. less or equal to \( e \)), we can conclude that in (b) the total number of operations is

- \( e \) inversions
- \( 10et + e \) multiplications
- \( 11et + e \) additions
Step \(f_{PGZ}\) is the same of \(PGZ\) and has a computational cost is of \(O(e^2)\) operations. Hence the computational complexity of the \(f_{PGZ}\) decoding algorithm \(2.2.5\) is \(O(et)\), much better than in the case of the \(PGZ\) decoding algorithm \(2.1.4\).

To conclude we give an example of decoding of a Reed-Solomon code using the \(f_{PGZ}\) decoding algorithm \(2.2.5\).

**Example 2.2.6.** Let \(\alpha\) be a primitive element of \(F_{2^4}\) satisfying \(\alpha^4 + \alpha + 1 = 0\). Consider over \(F_{2^4}\) the code, \(RS(15, 9, \alpha)\) generated by

\[g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^8)\]

The code has distance 9, so \(t = 4\). Suppose that the codeword sent is \(c = 0\) and the error vector is

\[e(x) = \alpha^2x^2 + \alpha x^8 + \alpha^7x^{13}\]

Clearly \(r(x) = e(x)\) and step \(f_{PGZ}\) reads out the syndromes

\[
S_1 = r(\alpha) = \alpha^{12} \quad S_2 = r(\alpha^2) = 0 \quad S_3 = r(\alpha^3) = 0 \quad S_4 = r(\alpha^4) = \alpha^5 \quad S_5 = r(\alpha^5) = \alpha^{11} \quad S_6 = r(\alpha^6) = \alpha^{13} \quad S_7 = r(\alpha^7) = \alpha^3 \quad S_8 = r(\alpha^8) = \alpha
\]

Step \(f_{PGZ}\) in (a) sets \(i_0 = 1\), \(w^{(1)} = 0\), \(y^{(1)} = \alpha^3\), while in (b) calculates:

\[
(i = 1) \quad \varepsilon_1 = S_2w_0^{(1)} + S_3 = 0 \\
\varepsilon_2 = S_3w_0^{(1)} + S_4 = 0 + \alpha^5 = \alpha^5
\]

\(\varepsilon_1 = 0\) and \(\varepsilon_2 \neq 0 \Rightarrow\) the singularity gap is \(r = 2\), hence

\[
i = i + 2 = 3;
\]

\[
y^{(3)} = \frac{1}{\varepsilon_2} \begin{pmatrix} w_0^{(1)} \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \alpha^{10} \\ 0 \end{pmatrix};
\]

\[
w^{(3)} = \begin{pmatrix} \alpha^8 \\ \alpha^9 \\ \alpha^6 \end{pmatrix};
\]

\[
(i = 3) \quad \varepsilon_1 = S_4w_0^{(3)} + S_5w_1^{(3)} + S_6w_2^{(3)} + S_7 = \alpha^{13} + \alpha^5 + \alpha^4 + \alpha^3 = 0
\]

\(\varepsilon_1 = 0 \Rightarrow\) the singularity gap is \(r > t - i = 1\), hence

\[
e = i = 3;
\]

\[
\sigma(x) = \alpha^8x^3 + \alpha^9x^2 + \alpha^6x + 1;
\]
Since \( \sigma(x) = \alpha^8x^3 + \alpha^9x^2 + \alpha^6x + 1 = (1 + \alpha^2x)(1 + \alpha^8x)(1 + \alpha^{13}x) \), step 3 calculates that:

\[
X_1 = \alpha^2, \quad X_2 = \alpha^8, \quad X_3 = \alpha^{13}
\]

Therefore the error positions are:

\[
p_1 = 2, \quad p_2 = 8, \quad p_3 = 13
\]

Finally in fPGZ 4 the linear system

\[
\begin{pmatrix}
\alpha^2 & \alpha^8 & \alpha^{13} \\
(\alpha^2)^2 & (\alpha^8)^2 & (\alpha^{13})^2 \\
(\alpha^2)^3 & (\alpha^8)^3 & (\alpha^{13})^3
\end{pmatrix}
\begin{pmatrix}
E_1 \\
E_2 \\
E_3
\end{pmatrix}
= 
\begin{pmatrix}
\alpha^{12} \\
0 \\
0
\end{pmatrix}
\]

is solved to find that the error values are:

\[
E_1 = \alpha^2, \quad E_2 = \alpha, \quad E_3 = \alpha^7
\]

Thus \( e(x) = E_1x^{p_1} + E_2x^{p_2} + E_3x^{p_3} = \alpha^2x^2 + \alpha x^8 + \alpha^7x^{13} \) and the received word \( r \) is correctly decoded in \( c = r - e = 0 \).

### 2.3 Avoiding Malfunctions in fPGZ

In section 1.1 we studied the definition and the behavior of a \( t \)-bounded distance decoding algorithm. In this section we want to understand whether the fPGZ decoding algorithm 2.2.5 for Reed-Solomon codes is \( t \)-bounded distance or not. In particular we will describe the necessary and sufficient conditions to avoid malfunctions of this decoding algorithm.

Consider the code \( RS(n, d, \alpha) \) and let \( t \) be its error correction capability. Let \( c, e \) and \( r = c + e \) respectively denote the transmitted codeword, the vector error and the received word. Without any conditions about \( e = wt(e) \), we suppose that the received word \( r \) is the input given to the fPGZ decoding algorithm 2.2.5. In section 1.1 we defined the set \( B \) as:

\[
B = \bigsqcup_{c \in RS(n, d, \alpha)} \overline{B}_t(c)
\]

By what we have already proved, if \( wt(e) \leq t \), then \( r \in B \) and the fPGZ decoding algorithm 2.2.5 decodes correctly \( r \), giving as output \( c \). Whereas if
2.3. AVOIDING MALFUNCTIONS IN FPGZ

\( \text{wt}(e) > t \) and \( r \in \mathcal{B} \), then an unavoidable decoder error has happened. So to know whether the FPGZ decoding algorithm is \( t \)-bounded distance or not, we have to study what happens when \( r \notin \mathcal{B} \), recalling that when \( r \notin \mathcal{B} \) then a \( t \)-bounded distance decoding algorithm does not correct \( r \) and declare the decoder failure.

For this aim, we introduce the following notation: denote by \( \tilde{e} \) the assumed number of occurred errors computed at the end of FPGZ and recall that

\[ \tilde{e} = \max\{i \in \{1, 2, \ldots, t \} \mid \det(A_i) \neq 0\} \leq t \]

Similarity we assume that:

- \( \tilde{\sigma}(x) = \tilde{\sigma}_e x^e + \cdots + \tilde{\sigma}_1 x + 1 \) is the polynomial calculated in FPGZ,
- \( \tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_e \) are the inverses of the roots of \( \tilde{\sigma}(x) \) calculated in FPGZ,
- \( \tilde{E}_i \) is the \( i \)th assumed error value,
- \( \tilde{e} \) is the assumed error vector. Note that \( \tilde{e} = \text{wt}(\tilde{e}) \).

and we call \( \tilde{c} = r - \tilde{e} \) the output vector. We observe that, since \( \tilde{e} \leq t \), it holds that

\[ r \in \mathcal{B} \iff \tilde{c} \in RS(n, d, \alpha) \quad (2.16) \]

and we show in the following example a malfunction of the FPGZ decoder.

Example 2.3.1. Let \( \alpha \) denote a primitive element of \( \mathbb{F}_{2^4} \) satisfying \( \alpha^4 + \alpha + 1 = 0 \). Consider over \( \mathbb{F}_{2^4} \) the code, \( RS(15, 9, \alpha) \) generated by

\[ g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^8) \]

The code has distance 9, so \( t = 4 \). Suppose that the sent codeword is \( 0 \) and the error vector is

\[ e(x) = \alpha^3 x + \alpha^3 x^2 + \alpha^{14} x^{10} + \alpha^5 x^{12} + \alpha^8 x^{13} \]

Clearly \( e = 5 \) and \( r(x) = e(x) \). The syndrome values are:

\[
\begin{align*}
S_1 &= r(\alpha) = \alpha^{10} & S_2 &= r(\alpha^2) = \alpha^2 & S_3 &= r(\alpha^3) = \alpha^8 & S_4 &= r(\alpha^4) = \alpha^7 \\
S_5 &= r(\alpha^5) = 0 & S_6 &= r(\alpha^6) = \alpha^3 & S_7 &= r(\alpha^7) = \alpha^9 & S_8 &= r(\alpha^8) = \alpha^8
\end{align*}
\]

So FPGZ starts setting \( i_0 = 1 \), \( w^{(1)} = \alpha^7 \) and \( y^{(1)} = \alpha^5 \) and continues with the iterations:
(i = 1) \[ \varepsilon_1 = S_2w_0^{(1)} + S_3 = \alpha^2\alpha^7 + \alpha^8 = \alpha^{12}; \]
\[ \varepsilon_1 \neq 0 \Rightarrow y^{(2)} = \frac{1}{\varepsilon_1} \left( \frac{w_0^{(1)}}{1} \right) = \frac{1}{\alpha^{12}} \left( \frac{\alpha^7}{1} \right) = (\alpha^{10}/\alpha^3); \]
\[ \varepsilon_2 = S_3w_0^{(1)} + S_4 = \alpha^8\alpha^7 + \alpha^7 = \alpha^9; \]
\[ \eta = S_2y_0^{(1)} + S_4 = \alpha^2\alpha^5 = \alpha^7; \]
\[ w^{(2)} = \begin{pmatrix} 0 \\ \alpha^7 \end{pmatrix} - \alpha^{12} \begin{pmatrix} \alpha^5 \\ 0 \end{pmatrix} + [\varepsilon_1\eta - \varepsilon_2] \begin{pmatrix} \alpha^{10} \\ \alpha^3 \end{pmatrix} = \begin{pmatrix} \alpha^{11} \\ \alpha^{12} \end{pmatrix}; \]

(i = 2) \[ \varepsilon_1 = S_3w_0^{(2)} + S_4w_1^{(2)} + S_5 = \alpha^8\alpha^{11} + \alpha^7\alpha^{12} + 0 = 0; \]
\[ \varepsilon_2 = S_4w_0^{(2)} + S_5w_1^{(2)} + S_6 = \alpha^7\alpha^{11} + 0 + \alpha^3 = 0; \]
\[ \varepsilon_1 = \varepsilon_2 = 0 \Rightarrow \tilde{\varepsilon} = 2; \]
\[ \tilde{\sigma}(x) = \alpha^{11}x^2 + \alpha^{12}x + 1; \]

Since \( \tilde{\sigma}(x) = (1 + x)(1 + \alpha^{11}x) \), in \textit{fPGZ} we have \( \tilde{X}_1 = 1 = \alpha^0 \) and \( \tilde{X}_2 = \alpha^{11} \). So the assumed error positions are \( i_1 = 0 \) and \( i_2 = 11 \). Finally in \textit{fPGZ} calculates the assumed error values solving
\[ \begin{pmatrix} 1 & \alpha^{11} \\ 1 & \alpha^7 \end{pmatrix} \begin{pmatrix} \tilde{E}_1 \\ \tilde{E}_2 \end{pmatrix} = \begin{pmatrix} \alpha^{10} \\ \alpha^2 \end{pmatrix} \]

Thus we have that \( \tilde{E}_1 = \alpha^6 \) and \( \tilde{E}_2 = \alpha^{11} \) and the output vector is
\[ \tilde{c}(x) = r(x) - (\alpha^{11}x^{11} + \alpha^6) = \alpha^6 + \alpha^3x + \alpha^3x^2 + \alpha^{14}x^{10} + \alpha^{11}x^{11} + \alpha^5x^{12} + \alpha^8x^{13} \neq c(x) \]

We note that
\[ \tilde{c}(\alpha^7) = \alpha^7 \neq 0 \Rightarrow \tilde{c} \notin \text{RS}(15,9,\alpha) \Rightarrow r \notin \mathcal{B} \]

Thus a decoder malfunction has occurred.
To avoid malfunctions of this kind in the $fPGZ$ decoding algorithm we need to add in its implementation the conditions necessary and sufficient to assure that the received word $r$ belongs to the set $B$. If one of these conditions will be not satisfied, then the $fPGZ$ decoder must declare a failure without carry out the decoding. By (2.16), one way to prevent malfunction from occurring is simply to add a step at the end of the algorithm which checks if $	ilde{c} \in RS(n, d, \alpha)$. This involves checking $d - 1$ equations. We will see later in this section that the same goal may be achieved with a lower number of equations to check. First a lemma about Hankel matrices is necessary:

**Lemma 2.3.2.** Let $M$ be a $k \times k$ or $k \times (k + 1)$ Hankel matrix and let $m$ be its rank. If the $m \times m$ leading principal minor in $M$ is non-singular, then all the entries of $M$ are completely determined by the first $2m$ entries.

We leave the proof of lemma 2.3.2 to the reader since it uses only the symmetric structure of an Hankel matrix and some easy linear algebra.

We state now the proposition that fixes the conditions to add in the $fPGZ$ decoding algorithm in order to make it $t$-bound distance:

**Proposition 2.3.3.** With previous notations, we have that $	ilde{c} \in RS(n, d, \alpha)$ if and only if

1. there exists $i_0 \leq t$ as defined in remark 2.2.4;
2. $\tilde{\sigma}(x)$ has exactly $\tilde{e}$ distinct roots, all belonging to the field $\mathbb{F}_q$ and all different from zero;
3. $\text{rk}(A) = \tilde{e}$;
4. for any $i \in \{1, 2, \ldots, \tilde{e}\}$, $\tilde{E}_i \neq 0$.

Condition (1) has been already included in the $fPGZ$ decoding algorithm but it is not sufficient as seen in example 2.3.1.

**Proof.** If $\tilde{c} \in RS(n, d, \alpha)$, then $r = \tilde{c} + \tilde{e}$ with $\text{wt}(\tilde{e}) \leq t$. Thus we can repeat the construction of section 2.1 with $\tilde{c}$ and $\tilde{e}$ in place of $c$ and $e$, concluding that conditions (1), (2), (3) and (4) are satisfied. On the other hand, supposing that conditions (1), (2), (3) and (4) are satisfied, we have to prove that $\tilde{c} \in RS(n, d, \alpha)$, which is equivalent to prove that $\tilde{c}(\alpha^i) = 0$ for any $i = 1, 2, \ldots, d - 1$. We observe that

$$\tilde{c}(\alpha^i) = r(\alpha^i) - \tilde{e}(\alpha^i) = S_i - \tilde{e}(\alpha^i)$$

and we define $\tilde{S}_i = \tilde{e}(\alpha^i)$. Condition (1) ensures that step $fPGZ$ can start correctly, while condition (2) ensures that in step $fPGZ$ there exist
\(p_1, p_2, \ldots, p_n\) in \(\{1, 2, \ldots, n\}\) such that \(\bar{X}_i = \alpha^{p_i}\) and \(j_a \neq j_b\) if \(a \neq b\). Thus \(\bar{e}(x) = \sum_{i=1}^{\bar{e}} \tilde{E}_i x^{p_i}\). Since the assumed error values \(\tilde{E}_i\)'s are computed in \(fPGZ\) solving the linear system

\[
\begin{pmatrix}
\tilde{X}_1 & \tilde{X}_2 & \cdots & \tilde{X}_{\bar{e}} \\
\tilde{X}_1^2 & \tilde{X}_2^2 & \cdots & \tilde{X}_{\bar{e}}^2 \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{X}_1^\bar{e} & \tilde{X}_2^\bar{e} & \cdots & \tilde{X}_{\bar{e}}^\bar{e}
\end{pmatrix}
\begin{pmatrix}
\tilde{E}_1 \\
\tilde{E}_2 \\
\vdots \\
\tilde{E}_{\bar{e}}
\end{pmatrix}
= \begin{pmatrix}
S_1 \\
S_2 \\
\vdots \\
S_{\bar{e}}
\end{pmatrix}
\]

we know that

\[
S_j = \sum_{i=1}^{\bar{e}} \tilde{E}_i \tilde{X}_i^j = \sum_{i=1}^{\bar{e}} \tilde{E}_i (\alpha^{p_i})^j = \bar{e}(\alpha^j) \quad \forall j = 1, 2, \ldots, \bar{e}
\]

So we have already proved that \(S_i = \bar{S}_i\) for any \(i = 1, 2, \ldots, \bar{e}\). We will prove now that the same equality also holds for \(i = \bar{e} + 1, \bar{e} + 2, \ldots, d - 1\). From

\[
A_{\bar{e}} \begin{pmatrix}
\bar{\sigma}_{\bar{e}} \\
\vdots \\
\bar{\sigma}_1
\end{pmatrix}
= - \begin{pmatrix}
S_{\bar{e} + 1} \\
\vdots \\
S_{2\bar{e}}
\end{pmatrix}
\]

it follows that the first \(2\bar{e}\) syndromes are completely determined by the first \(\bar{e}\) syndromes, in fact:

\[
-S_{\bar{e} + l} = \sum_{i=1}^{\bar{e}} S_{\bar{e} + l - i} \bar{\sigma}_i \quad \forall l = 1, 2, \ldots, \bar{e} \quad (2.17)
\]

From \(\bar{\sigma}(\bar{X}_i^{-1}) = 0\) for any \(i = 1, 2, \ldots, \bar{e}\) it holds that

\[
\bar{X}_i + \bar{\sigma}_1 \bar{X}_i^{-1} + \cdots + \bar{\sigma}_{\bar{e}} = 0
\]

Let \(j \neq 0\) be a natural number, we multiply the former equation by \(\tilde{E}_i \tilde{X}_i^j\) (different from zero by conditions (22) and (31)) and after we sum over \(i = 1, 2, \ldots, \bar{e}\). Thus we obtain

\[
\left(\sum_{i=1}^{\bar{e}} \tilde{E}_i \tilde{X}_i^{\bar{e} + j}\right) + \left(\sum_{i=1}^{\bar{e}} \tilde{E}_i \tilde{X}_i^{\bar{e} + j - 1}\right) \bar{\sigma}_1 + \cdots + \left(\sum_{i=1}^{\bar{e}} \tilde{E}_i \tilde{X}_i^j\right) \bar{\sigma}_{\bar{e}} = 0 \quad \forall j \geq 1
\]

that is

\[
-S_{\bar{e} + j} = \sum_{i=1}^{\bar{e}} S_{\bar{e} + j - i} \bar{\sigma}_i \quad \forall j \geq 1 \quad (2.18)
\]
2.3. AVOIDING MALFUNCTIONS IN FPGZ

Hence from (2.17) and (2.18) we also find that \( S_i = \tilde{S}_i \) for \( i = 1, 2, \ldots, 2\tilde{c} \).

Finally we consider the matrix \( \tilde{A} \) defined by

\[
\tilde{A} = \begin{pmatrix}
\tilde{S}_1 & \tilde{S}_2 & \cdots & \tilde{S}_{t+1} \\
\tilde{S}_2 & \tilde{S}_3 & \cdots & \tilde{S}_{t+2} \\
\vdots & \vdots & \ddots & \vdots \\
\tilde{S}_{d-1-t} & \tilde{S}_{d-t} & \cdots & \tilde{S}_{d-1}
\end{pmatrix}
\]

Since \( \tilde{A} \) is the syndrome matrix for the error vector \( \tilde{e} \), which has weight less or equal to \( t \), we know that \( \text{rk}(\tilde{A}) = \text{wt}(\tilde{e}) = \tilde{e} \). So we can conclude the proof using condition (3) and lemma 2.3.2, which implies that \( A \) and \( \tilde{A} \) are equal.

Note that in the example 2.3.1 the syndrome matrix is

\[
A = \begin{pmatrix}
\alpha^{10} & \alpha^2 & \alpha^8 & \alpha^7 & 0 \\
\alpha^2 & \alpha^8 & \alpha^7 & 0 & \alpha^3 \\
\alpha^8 & \alpha^7 & 0 & \alpha^3 & \alpha^9 \\
\alpha^7 & 0 & \alpha^3 & \alpha^9 & \alpha^7
\end{pmatrix}
\]

and it holds that \( \text{rk}(A) \geq 3 \) since

\[
\det \begin{pmatrix}
\alpha^{10} & \alpha^2 & 0 \\
\alpha^2 & \alpha^8 & \alpha^3 \\
\alpha^8 & \alpha^7 & \alpha^9
\end{pmatrix} = \alpha^{14} \neq 0
\]

but \( \tilde{e} = 2 \), thus the condition \( \text{rk}(A) = \tilde{e} \) cannot hold.

While conditions (1), (2) and (4) in proposition 2.3.3 may be easily implemented without any significant extra cost, condition (3) is not good from an implementation point of view. For this reason we observe that the following equivalent condition holds:

**Proposition 2.3.4.** Supposing that conditions (1), (2) and (4) of proposition 2.3.3 hold, we have that

\[
\text{rk}(A) = \tilde{e} \iff A \begin{pmatrix}
0^i \\
\sigma_{\tilde{e}} \\
\vdots \\
\sigma_1 \\
1
\end{pmatrix} = 0 \quad \forall i = 0, 1, \ldots, t - \tilde{e}
\]
Proof. If \( \text{rk}(A) = \tilde{e} \), then from what we saw in the proof of proposition 2.3.3 it follows that

\[
-S_{\tilde{e}+j} = \sum_{i=1}^{\tilde{e}} S_{\tilde{e}+j-i} \tilde{\sigma}_i \quad \forall j = 1, 2, \ldots, d - 1 - \tilde{e}
\]

and we obtain straightly the right side condition. On the other hand, since \( \tilde{e} \) is calculated as \( \max \{i \in \{1, 2, \ldots, t\} \mid \det(A_i) \neq 0\} \), it is always true that \( \text{rk}(A) \geq \tilde{e} \) and it remains to prove only that \( \text{rk}(A) \leq \tilde{e} \). If the right side condition is accomplished, then there are \( t + 1 - \tilde{e} \) linearly independent vectors in \( \ker(A) \). It follows that \( \dim(\ker(A)) \geq t + 1 - \tilde{e} \). By definition of the syndrome matrix, we have that \( t + 1 = \dim(\ker(A)) + \text{rk}(A) \), hence \( \text{rk}(A) \leq \tilde{e} \) and the proof is complete.

Thanks to the Hankel structure of the syndrome matrix, checking the right side condition in proposition 2.3.4 is equivalent to checking the following \( d - 1 - 2\tilde{e} \) equations:

\[
\begin{align*}
S_{2\tilde{e}+1} + \sum_{i=1}^{\tilde{e}} S_{2\tilde{e}+1-i} \tilde{\sigma}_i &= 0 \\
S_{2\tilde{e}+2} + \sum_{i=1}^{\tilde{e}} S_{2\tilde{e}+2-i} \tilde{\sigma}_i &= 0 \\
&\vdots \\
S_{d-1} + \sum_{i=1}^{\tilde{e}} S_{d-1-i} \tilde{\sigma}_i &= 0
\end{align*}
\]

Note that the first \( t - \tilde{e} \) equations hold by the condition

\[
\varepsilon_1 = \varepsilon_2 = \cdots = \varepsilon_{t-\tilde{e}} = 0
\]

that happens at the end of PGZ decoder. Checking the remaining equations has a computational complexity of \( O(\tilde{e}t) \) operations, so we may add this check to step PGZ without increasing its total complexity.

Finally we observe that the PGZ decoder can malfunction because it may not use all the syndrome values in the decoding, indeed step PGZ calculates \( \tilde{e} \) and \( \tilde{\sigma}(x) \) using only the first \( \tilde{e} + t \) syndromes. Thus it is understandable that the controls necessary to avoid malfunctions involve the remaining syndromes.
Chapter 3

Berlekamp-Massey Decoder

Elwyn R. Berlekamp published his algorithm to solve the key equation (1.4) in 1968 and in 1969 James Massey gave a simplified version of it. The described algorithm found the polynomials $\sigma(x)$ and $\omega(x)$ with an iterative procedure based on polynomial families defined by recursion. In section 3.1 we will study in detail this procedure, while in section 3.2 we will describe the necessary and sufficient conditions to avoid decoder malfunctions in the implementation given of the Berlekamp-Massey decoder. Finally in section 3.3 we will compare the latter with the $fPGZ$ decoding algorithm.

3.1 BM Decoding Algorithm

The Berlekamp-Massey (BM) decoding algorithm is iterative and solves the key equation (1.4) in successively higher degrees. In other words, for $i = 0, 1, \ldots, d - 1$ we attempt to find polynomials of “small” degree

$$
\sigma^{(i)}(x) = \sum_{j=0}^{i} \sigma_{j}^{(i)} x^j \in \mathbb{F}_q[x]
$$

$$
\omega^{(i)}(x) = \sum_{j=0}^{i-1} \omega_{j}^{(i)} x^j \in \mathbb{F}_q[x]
$$

which satisfy

$$
\sigma^{(i)}(x) S(x) \equiv \omega^{(i)}(x) \pmod{x^i} \quad \text{(3.1)}
$$

We look for solutions of small degree because we have already seen in the theorem 1.5.6 that the pair $(\sigma(x), \omega(x))$ is the valid solution of minimal degree for the key equation (1.4). The iterative nature of the algorithm follows from
the observation that if we know a solution to (3.1) that satisfies the conditions
\[ \deg(\sigma^{(i)}(x)) \leq i \text{ and } \deg(\omega^{(i)}(x)) \leq i - 1, \]
then we may write
\[ \sigma^{(i)}(x)S(x) \equiv \omega^{(i)}(x) + \Delta_ix^i \pmod{x^{i+1}} \]
where \( \Delta_i \) is the coefficient of \( x^i \) in the polynomial \( \sigma^{(i)}(x)S(x) \), that is
\[ \Delta_i \overset{def}{=} \sum_{j=0}^{i} S_{i+1-j} \sigma^{(i)}_{j} \quad (3.2) \]
and it is called \( i \)th discrepancy. If \( \Delta_i = 0 \), then we may evidently proceed taking \( \sigma^{(i+1)}(x) = \sigma^{(i)}(x) \) and \( \omega^{(i+1)}(x) = \omega^{(i)}(x) \). In order to define \( \sigma^{(i+1)}(x) \) and \( \omega^{(i+1)}(x) \) also in the case when \( \Delta_i \neq 0 \), we introduce the auxiliary polynomials \( \tau^{(i)}(x) \) and \( \gamma^{(i)}(x) \), which will be chosen so that they are a solution of the auxiliary equation
\[ \tau^{(i)}(x)S(x) \equiv \gamma^{(i)}(x) + x^{i-1} \pmod{x^{i}} \quad (3.3) \]
Supposing the existence of the auxiliary polynomials \( \tau^{(i)}(x) \) and \( \gamma^{(i)}(x) \) for any \( i \in \{0, 1, \ldots, d - 2\} \), then

**Definition 3.1.1.** We define
\[
\begin{align*}
\sigma^{(0)}(x) & = 1 \\
\sigma^{(i+1)}(x) & = \sigma^{(i)}(x) - \Delta_ix\tau^{(i)}(x) \\
\omega^{(0)}(x) & = 0 \\
\omega^{(i+1)}(x) & = \omega^{(i)}(x) - \Delta_ix\gamma^{(i)}(x)
\end{align*}
\]
for any \( i \in \{0, 1, \ldots, d - 2\} \).

We will show later on in proposition 3.1.5 that the polynomials \( \sigma^{(i)}(x) \)'s and \( \omega^{(i)}(x) \)'s just defined satisfy (3.1) for every \( i \in \{0, 1, \ldots, d - 1\} \) and moreover we will prove that \( \sigma^{(d-1)}(x) = \sigma(x) \) and \( \omega^{(d-1)}(x) = \omega(x) \).

The auxiliary polynomials are recursively defined during the algorithm. Indeed if \( \tau^{(i)}(x) \) and \( \gamma^{(i)}(x) \) satisfy the congruence (3.3), then we have two obvious ways to define \( \tau^{(i+1)}(x) \) and \( \gamma^{(i+1)}(x) \):
\[
\begin{align*}
\tau^{(i+1)}(x) & = \frac{\sigma^{(i)}(x)}{\Delta_i} \\
\gamma^{(i+1)}(x) & = \frac{\omega^{(i)}(x)}{\Delta_i}
\end{align*}
\]
If \( \Delta_i = 0 \), then our choice is forced, while, if \( \Delta_i \neq 0 \), then our choice must be based upon the aim to minimize the degrees of \( \sigma^{(i+1)}(x) \) and \( \omega^{(i+1)}(x) \). For this reason we introduce the function \( D : \mathbb{N} \to \mathbb{N} \), defined by
Definition 3.1.2. Let $D(0) = 0$ and

$$D(i + 1) = \begin{cases} 
D(i) & \text{if } \Delta_i = 0 \text{ or } 2D(i) \geq i + 1 \\
i + 1 - D(i) & \text{if } \Delta_i \neq 0 \text{ and } 2D(i) < i + 1
\end{cases}$$

for any $i \geq 0$.

The function $D$ is evidently nondecreasing and nonnegative and $D(i) \leq i$ for any $i \in \mathbb{N}$. Moreover, as we will prove later on in this section, it represents an upper bound for $\deg (\sigma^{(i)}(x))$ and permits the right definition of the auxiliary polynomials as follows:

Definition 3.1.3. Let $\tau^{(0)}(x) = 1$ and $\gamma^{(0)}(x) = -x^{-1}$. Then

$$\tau^{(i+1)}(x) = \begin{cases} 
x \tau^{(i)}(x) & \text{if } \Delta_i = 0 \text{ or } 2D(i) \geq i + 1 \\
\frac{\sigma^{(i)}(x)}{\Delta_i} & \text{if } \Delta_i \neq 0 \text{ and } 2D(i) < i + 1
\end{cases}$$

$$\gamma^{(i+1)}(x) = \begin{cases} 
x \gamma^{(i)}(x) & \text{if } \Delta_i = 0 \text{ or } 2D(i) \geq i + 1 \\
\frac{\omega^{(i)}(x)}{\Delta_i} & \text{if } \Delta_i \neq 0 \text{ and } 2D(i) < i + 1
\end{cases}$$

for any $i \in \{0, 1, \ldots, d - 2\}$.

With these definitions it is immediate to show that:

Proposition 3.1.4. If $\sigma^{(i)}(x)$, $\omega^{(i)}(x)$, $\tau^{(i)}(x)$ and $\gamma^{(i)}(x)$ are the polynomials defined respectively in definitions 3.1.2 and 3.1.3, then

$$\sigma^{(i)}(x)S(x) \equiv \omega^{(i)}(x) \pmod{x^i}$$

$$\tau^{(i)}(x)S(x) \equiv \gamma^{(i)}(x) + x^{i-1} \pmod{x^i}$$

for any $i \in \{0, 1, \ldots, d - 1\}$. Moreover

$$\omega^{(i)}(x)\tau^{(i)}(x) - \sigma^{(i)}(x)\gamma^{(i)}(x) = x^{i-1} \quad (3.4)$$

and thus we have that $\gcd (\sigma^{(i)}(x), \omega^{(i)}(x)) = 1$.

Proof. We will prove the proposition by induction. If $i = 0$ the congruences are both trivial. Suppose the congruences true for $i$, it follows that

$$\begin{cases} 
\sigma^{(i)}(x)S(x) \equiv \omega^{(i)}(x) + \Delta_i x^i \pmod{x^{i+1}} \\
x^i \equiv x \tau^{(i)}(x)S(x) - x \gamma^{(i)}(x) \pmod{x^{i+1}}
\end{cases}$$
where $\Delta_i$ is defined by (3.2). Substituting the expression for $x^i$ in the first congruence we obtain that

$$\sigma^{(i)}(x)S(x) \equiv \omega^{(i)}(x) + \Delta_i \left( x\tau^{(i)}(x)S(x) - x\gamma^{(i)}(x) \right) \pmod{x^{i+1}}$$

and using that

$$\sigma^{(i+1)}(x) = \sigma^{(i)}(x) - \Delta_i x\tau^{(i)}(x)$$
$$\omega^{(i+1)}(x) = \omega^{(i)}(x) - \Delta_i x\gamma^{(i)}(x)$$

we can conclude that

$$\sigma^{(i+1)}(x)S(x) \equiv \omega^{(i+1)}(x) \pmod{x^{i+1}}$$

Now we consider the relations that can define $\tau^{(i+1)}(x)$ (definition 3.1.3). In the first case

$$\tau^{(i+1)}(x)S(x) = x\tau^{(i)}(x)S(x) \equiv x\gamma^{(i)}(x) + x^i \pmod{x^{i+1}}$$

In the other case

$$\tau^{(i+1)}(x)S(x) = \frac{\sigma^{(i)}(x)}{\Delta_i}S(x) \equiv \frac{\omega^{(i)}(x) + \Delta_i x^i}{\Delta_i} \pmod{x^{i+1}}$$

$$\equiv \gamma^{(i+1)}(x) + x^i \pmod{x^{i+1}}$$

and this concludes the proof of the inductive step. Thus the congruences are proved. In order to prove (3.4) we note that base case is again trivial, so we focus our attention on the inductive step:

$$\omega^{(i+1)}\tau^{(i+1)} - \sigma^{(i+1)}\gamma^{(i+1)} = \left( \omega^{(i)} - \Delta_i x\gamma^{(i)} \right) \tau^{(i+1)} - \left( \sigma^{(i)} - \Delta_i x\tau^{(i)} \right) \gamma^{(i+1)}$$

Depending on the values of $\Delta_i$ and $D(i)$ we will substitute in the former expression the right values for $\tau^{(i+1)}(x)$ and $\gamma^{(i+1)}(x)$. It is an easy computation to verify that, in both cases, using the inductive hypothesis we obtain that

$$\omega^{(i+1)}(x)\tau^{(i+1)}(x) - \sigma^{(i+1)}(x)\gamma^{(i+1)}(x) = x^i$$

From the relation just proved, it follows that $\gcd\left(\sigma^{(i)}(x), \omega^{(i)}(x)\right) \mid x^{(i-1)}$, but it is immediate to prove that $\sigma^{(i)}(0) = 1$ for every $i$, that is $x \nmid \sigma^{(i)}(x)$. Thus we conclude that $\gcd\left(\sigma^{(i)}(x), \omega^{(i)}(x)\right) = 1$. \hfill $\square$

In order to prove that the polynomials $\sigma^{(d-1)}(x)$ and $\omega^{(d-1)}(x)$ are respectively the error-locator polynomial and the error-evaluator polynomial, we need the followings propositions and theorems:
Proposition 3.1.5. If $\sigma^{(i)}(x)$, $\omega^{(i)}(x)$, $\tau^{(i)}(x)$ and $\gamma^{(i)}(x)$ are the polynomials defined respectively in definitions 3.1.1 and 3.1.3, then $\forall i \in \{0, 1, \ldots, d - 1\}$ we have that

\[
\begin{align*}
(1) & \quad \begin{cases}
\deg(\sigma^{(i)}(x)) \leq D(i) \\
\deg(\tau^{(i)}(x)) \leq i - D(i)
\end{cases} \\
(2) & \quad \begin{cases}
\deg(\omega^{(i)}(x)) \leq D(i) - 1 \\
\deg(\gamma^{(i)}(x)) \leq i - D(i) - 1
\end{cases}
\end{align*}
\]

where $D$ is the integer function defined in definition 3.1.2.

Proof. We will prove both claims by induction. The base case of (1) is trivial. Suppose that the inequalities are true for $i$ and consider the polynomial $\tau^{(i+1)}(x)$. Depending on the values of $\Delta_i$ and $D(i)$ we have:

\[
\deg(\tau^{(i+1)}) = \begin{cases}
1 + \deg(\tau^{(i)}) \leq 1 + i - D(i) = 1 + i - D(i + 1) \\
\deg(\sigma^{(i)}) \leq D(i) = i + 1 - D(i + 1)
\end{cases}
\]

So, in both cases, $\deg(\tau^{(i+1)}(x)) \leq i + 1 - D(i + 1)$. Moreover, using that $\sigma^{(i+1)}(x) = \sigma^{(i)}(x) - \Delta_i x \tau^{(i)}(x)$ we have that

- if $\Delta_i = 0 \Rightarrow \deg(\sigma^{(i+1)}) = \deg(\sigma^{(i)}) \leq D(i) = D(i + 1)$
- if $\Delta_i \neq 0 \Rightarrow \deg(\sigma^{(i+1)}) \leq \max\{D(i), i + 1 - D(i)\} = D(i + 1)$

and (1) is proved. The proof of (2) is almost identical to that of (1), using the relation that defines $\omega^{(i+1)}(x)$ (definition 3.1.1).

Consider now the set of all possible solutions of the key equation (3.1), that is

\[\mathcal{M}_i = \{(a(x), b(x)) \in \mathbb{F}_q[x]^2 \mid a(x)S(x) \equiv b(x) \pmod{x^i}\}\]

Clearly $\mathcal{M}_i$ is a free submodule of rank 2 of $\mathbb{F}_q[x]^2$, since $\{(1, S(x)), (0, x^i)\}$ is obviously a basis of $\mathcal{M}_i$. In the following theorem we will show that even the set $\{(\sigma^{(i)}(x), \omega^{(i)}(x)), (x\tau^{(i)}(x), x\gamma^{(i)}(x))\}$ is a base for $\mathcal{M}_i$. 

\[\square\]
Theorem 3.1.6. Let $i$ be a fixed index in $\{1, \ldots, d-1\}$. If $(a(x), b(x)) \in \mathcal{M}_i$ then there exist polynomials $u(x), v(x) \in \mathbb{F}_q[x]$ such that $u(0) = a(0)$ and

\[
\begin{cases}
    a(x) = u(x)\sigma^{(i)}(x) + v(x)\tau^{(i)}(x) \\
    b(x) = u(x)\omega^{(i)}(x) + v(x)\gamma^{(i)}(x)
\end{cases}
\]

Moreover, if there exists $\delta \in \mathbb{N}^+$ such that $\deg(a(x)) \leq \delta$ and $\deg(b(x)) \leq \delta - 1$ then we have $\deg(u(x)) \leq \delta - D(i)$ and $\deg(v(x)) \leq \delta + D(i) - i - 1$.

Proof. By hypothesis we have that

\[
a(x)\omega^{(i)}(x) \equiv a(x)\sigma^{(i)}(x)S(x) \equiv b(x)\sigma^{(i)}(x) \pmod{x^i}
\]

\[
a(x)\gamma^{(i)}(x) \equiv a(x)\left[\tau^{(i)}(x)S(x) - x^{i-1}\right] \equiv b(x)\tau^{(i)}(x) - a(0)x^{i-1} \pmod{x^i}
\]

so there exist polynomials $\tilde{u}(x)$ and $v(x)$ such that

\[
a(x)\omega^{(i)}(x) - b(x)\sigma^{(i)}(x) = x^iv(x)
\]

\[
a(x)\gamma^{(i)}(x) - b(x)\tau^{(i)}(x) = x^{i-1}[x\tilde{u}(x) - a(0)]
\]

We define $u(x) = a(0) - x\tilde{u}(x)$, hence we have that $u(0) = a(0)$ and

\[
a(x)\omega^{(i)}(x) - b(x)\sigma^{(i)}(x) = x^iv(x) \tag{3.5}
\]

\[-a(x)\gamma^{(i)}(x) + b(x)\tau^{(i)}(x) = x^{i-1}u(x) \tag{3.6}
\]

from which, using (3.4), it follows that:

\[
x^{i-1}a(x) = \left[\omega^{(i)}(x)\tau^{(i)}(x) - \sigma^{(i)}(x)\gamma^{(i)}(x)\right] a(x) = \\
= \tau^{(i)}(x) \left[a(x)\omega^{(i)}(x) - b(x)\sigma^{(i)}(x)\right] - \sigma^{(i)}(x) \left[a(x)\gamma^{(i)}(x) - b(x)\tau^{(i)}(x)\right] = \\
= x^{i-1} \left[u(x)\sigma^{(i)}(x) + v(x)\tau^{(i)}(x)\right]
\]

and

\[
x^{i-1}b(x) = \left[\omega^{(i)}(x)\tau^{(i)}(x) - \sigma^{(i)}(x)\gamma^{(i)}(x)\right] b(x) = \\
= \gamma^{(i)}(x) \left[a(x)\omega^{(i)}(x) - b(x)\sigma^{(i)}(x)\right] - \omega^{(i)}(x) \left[a(x)\gamma^{(i)}(x) - b(x)\tau^{(i)}(x)\right] = \\
= x^{i-1} \left[u(x)\omega^{(i)}(x) + v(x)\gamma^{(i)}(x)\right]
\]

So the first part of the theorem is proved. From (3.5) and (3.6) it also follows that

\[i-1+\deg(u(x)) \leq \max \{\deg (\gamma^{(i)}(x)) + \deg (a(x)), \deg (\tau^{(i)}(x)) + \deg(b(x))\}\]
that is
\[ D \]

Using proposition 3.1.5, we see that
\[ u \]

follows that
\[ u \]

Since
\[ u \]

max \{deg (\(i\)) + deg(a(x)), \(i\) \} ≤ \(i\) − \(D(i) − 1\)

\[
\begin{align*}
\text{max} \{\text{deg} (\omega^{(i)}(x)) + \text{deg}(a(x)), \text{deg} (\tau^{(i)}(x)) + \text{deg}(b(x))\} & \leq \delta + i - D(i) - 1 \\
\text{max} \{\text{deg} (\omega^{(i)}(x)) + \text{deg}(a(x)), \text{deg} (\sigma^{(i)}(x)) + \text{deg}(b(x))\} & \leq D(i) + \delta - 1
\end{align*}
\]

so we have deg(u(x)) ≤ \(\delta - D(i)\) and deg(v(x)) ≤ \(\delta + D(i) - i - 1\) and this concludes the proof.

\[
\Box
\]

Corollary 3.1.7. Let \(i\) be a fixed index in \(\{1, \ldots, d - 1\}\) and let \(\delta \in \mathbb{N}^+\) be less or equal to \(\left\lfloor \frac{i}{2} \right\rfloor\). If \((a(x), b(x)) \in M_i, a(0) \neq 0\), deg(a(x)) ≤ \(\delta\) and deg(b(x)) ≤ \(\delta - 1\), then we have \(D(i) \leq \delta\) and

\[
\begin{align*}
\begin{cases}
\text{deg} (a(x)) = u(x)\sigma^{(i)}(x) \\
\text{deg} (b(x)) = u(x)\omega^{(i)}(x)
\end{cases}
\end{align*}
\]

where \(u(x) = \gcd(a(x), b(x)) \in \mathbb{F}_q[x]\).

Proof. Since \((a(x), b(x)) \in M_i,\) from theorem 3.1.6 it follows that there are \(u(x)\) and \(v(x)\) in \(\mathbb{F}_q[x]\) such that \(u(0) = a(0)\) and

\[
\begin{align*}
\begin{cases}
\text{deg} (a(x)) = u(x)\sigma^{(i)}(x) + v(x)\tau^{(i)}(x) \\
\text{deg} (b(x)) = u(x)\omega^{(i)}(x) + v(x)\gamma^{(i)}(x)
\end{cases}
\end{align*}
\]

Since \(u(0) = a(0) \neq 0\), we have that \(\deg(u(x)) \geq 0\) hence, from (3.6) it follows that

\[
\deg (-a(x)\gamma^{(i)}(x) + b(x)\tau^{(i)}(x)) \geq i - 1
\]

that is

\[
\deg (a(x)\gamma^{(i)}(x)) \geq i - 1 \quad \text{or} \quad \deg (b(x)\tau^{(i)}(x)) \geq i - 1
\]

Using proposition 3.1.5 we see that

\[
\deg (a(x)) \geq D(i) \quad \text{or} \quad \deg (b(x)) \geq D(i) - 1
\]

In both cases, using the hypothesis on the degree of \(a(x)\) or \(b(x)\), we conclude that \(D(i) \leq \delta\). Since \(\delta \leq \left\lfloor \frac{i}{2} \right\rfloor\) and \(D(i) \leq \delta\), using theorem 3.1.6 we have that

\[
\deg (v(x)) \leq \delta + D(i) - i - 1 \leq 2\delta - i - 1 \leq 2 \left\lfloor \frac{i}{2} \right\rfloor - i - 1 < 0
\]

that is \(v(x) = 0\). Moreover, since \(\gcd (\sigma^{(i)}(x), \omega^{(i)}(x)) = 1\), it is clear that \(u(x) = \gcd(a(x), b(x))\) and the proof is concluded.

\[
\Box
\]
Corollary 3.1.8. We recall the definitions given on page 44 of the polynomials \( \sigma^{(i)}(x) \)'s and \( \omega^{(i)}(x) \)'s for \( i \in \{0, 1, \ldots, d - 2\} \):

\[
\begin{align*}
\sigma^{(0)}(x) &= 1 \\
\sigma^{(i+1)}(x) &= \sigma^{(i)}(x) - \Delta_i x \tau^{(i)}(x) \\
\omega^{(0)}(x) &= 0 \\
\omega^{(i+1)}(x) &= \omega^{(i)}(x) - \Delta_i x \gamma^{(i)}(x)
\end{align*}
\]

If \( e \leq t \), then we have

\[
\begin{align*}
\sigma^{(d-1)}(x) &= \sigma(x) \\
\omega^{(d-1)}(x) &= \omega(x)
\end{align*}
\]

Moreover \( D(d - 1) = e \).

Proof. By the key equation (1.4), we already know that \((\sigma(x), \omega(x)) \in \mathcal{M}_{d-1}\). Hence, using remark 1.5.2 and corollary 3.1.7, we immediately conclude that \( \sigma^{(d-1)}(x) = \sigma(x) \), \( \omega^{(d-1)}(x) = \omega(x) \) and \( D(d - 1) \leq e \). Since

\[
e = \deg(\sigma(x)) = \deg(\sigma^{(d-1)}(x)) \leq D(d - 1)
\]

the proof is completed. \( \square \)

Before entering into the details of the BM decoding algorithm and stating it explicitly, we observe that if we know the polynomial \( \sigma(x) \), then the polynomial \( \omega(x) \) is completely determined, since the key equation (1.4) also implies that the coefficients of the error-evaluator polynomial are equal to the coefficients of the product \( \sigma(x)S(x) \) for all terms \( 1, x, \ldots, x^{e-1} \). In other words the coefficients of \( \omega(x) \) are determined by:

\[
\begin{align*}
\omega_0 &= S_1 \\
\omega_i &= S_{i+1} + \sum_{j=1}^{i} S_{i+1-j} \sigma_j \quad \text{for } i = 1, 2, \ldots, e - 1
\end{align*}
\]

(3.7)

Thus the computation of the polynomials \( \omega^{(i)}(x) \)'s and \( \gamma^{(i)}(x) \)'s is not necessary and, in order to save arithmetic operations and memory space, we prefer to calculate the error-evaluator polynomial \( \omega(x) \) by (3.7) after we have computed the error-locator polynomial \( \sigma(x) \).

Algorithm 3.1.9 (BM decoding algorithm for RS\((n, d, \alpha)\)).

Input: the received word \( r(x) \);
Output: the codeword \( c(x) \);
Begin
3.1. BM DECODING ALGORITHM

BM.1 (syndrome computation)

for \( i = 1, 2, \ldots, d - 1 \) do

\[ S_i := r(\alpha^i); \]

endfor

BM.2 (error-locator polynomial computation)

\[ \sigma^{(0)} := 1; \]
\[ \tau^{(0)} := 1; \]
\[ D(0) := 0; \]

for \( i = 0, 1, \ldots, d - 2 \) do

\[ \Delta_i := \sum_{j=0}^{D(i)} S_{i+1-j} \sigma_j^{(i)}; \]
\[ \sigma^{(i+1)}(x) := \sigma^{(i)}(x) - \Delta_i x \tau^{(i)}(x); \]

if \((\Delta_i = 0 \text{ or } 2D(i) \geq i + 1)\) then

\[ D(i + 1) := D(i); \]
\[ \tau^{(i+1)}(x) := x \tau^{(i)}(x); \]

else

\[ D(i + 1) := i + 1 - D(i); \]
\[ \tau^{(i+1)}(x) := \frac{\sigma^{(i)}(x)}{\Delta_i}; \]

endfor

\[ e := D(d - 1); \]
\[ \sigma(x) := \sigma^{(d-1)}(x); \]

BM.3 (finding error positions)

calculate the error positions \( p_1, p_2, \ldots, p_e \) and the elements \( X_1^{-1}, X_2^{-1}, \ldots, X_e^{-1} \) using Chien’s search (alg. 1.5.8);

BM.4 (finding error values)

\[ \omega(x) := S_1 + \sum_{i=1}^{e-1} \left( S_{i+1} + \sum_{j=1}^{i} S_{i+1-j} \sigma_j \right) x^i; \]

for \( i = 1, 2, \ldots, e \) do

\[ E_i := -\frac{\omega(X_i^{-1})}{\sigma'(X_i^{-1})}; \]

endfor
Return \( c(x) := r(x) - \sum_{i=1}^{e} E_i x^p; \)

End

When \( e \leq t \), corollary 3.1.8 ensures the correctness of the BM decoding algorithm. Its computational complexity (ignoring BM 1 and BM 3) is of order \( O(t^2) \). In fact, the cost of the \( i \)th iterative step in BM 2 is due to:

- the computation of the \( i \)th discrepancy (\( i \) multiplications and \( i \) additions);
- the computation of the coefficients of \( \sigma^{(i+1)}(x) \) as
  \[
  \begin{cases}
    \sigma_0^{(i+1)} := 1 \\
    \sigma_j^{(i+1)} := \sigma_j^{(i)} - \Delta_i \tau_j^{(i-1)}
  \end{cases}
  \]
  for any \( j = 1, 2, \ldots, i+1 \) (\( i+1 \) multiplications and \( i+1 \) additions);
- the computation of the coefficients of \( \tau^{(i+1)}(x) \) as
  \[
  \begin{cases}
    \tau_0^{(i+1)} := 0 \\
    \tau_j^{(i+1)} := \tau_{j-1}^{(i)}\quad \text{or} \quad \delta := \Delta_i^{-1} \\
    \tau_l^{(i+1)} := \delta \cdot \sigma_l^{(i)}
  \end{cases}
  \]
  for \( j = 1, 2, \ldots, i+1 \) and \( l = 0, 1, \ldots, i \) (at most one inversion and \( i+1 \) multiplications);

Hence the \( i \)th iterative step in BM 2 has a complexity upper bounded by \( 3i + 2 \) multiplications, \( 2i + 1 \) additions and one inversion. Thus summing for \( i = 0 \) to \( i = d - 2 \) and recalling that \( t = \left\lfloor \frac{d-1}{2} \right\rfloor \) (that is \( d - 1 = 2t + 1 \) or \( d - 1 = 2t \)), we obtain that the total complexity of BM 2 is upper bounded by:

\[
\begin{align*}
2t + 1 & \quad \text{inversions} \\
6t^2 + 7t + 4 & \quad \text{multiplications} \\
4t^2 + 4t + 1 & \quad \text{additions}
\end{align*}
\]

Step BM 4 does not increase the order of this upper bound because:

- the coefficient \( \omega_i \) is computed by \( i \) multiplications and \( i \) additions, thus the computation of \( \omega(x) \) requires a number of multiplications and additions less than \( c^2 \);
• the computation of the error $E_i$ requires two polynomial evaluations and one division; using the same procedure seen for Chien’s search (see algorithm 1.5.8) we can evaluate polynomials $\omega(x)$ and $\sigma'(x)$ in all the field elements with a negligible cost compared to the cost of computing the coefficients $\omega_i$’s and the following $e$ divisions $\frac{\omega(X^{-1})}{\sigma'(X^{-1})}$.

Remark 3.1.10. In section 1.5 we studied Horner’s method to evaluate polynomials for computing the syndromes $S_i = r(\alpha^i)$. Despite that we prefer use Chien’s search to evaluate the polynomials $\omega(x)$ and $\sigma'(x)$ because these polynomials have lower degree than $r(x)$ ($n > d - 1 > t \geq e$) and are evaluated over a wider range of elements than $r(x)$. These two differences make it inefficient to implement the evaluation of $\omega(x)$ and $\sigma'(x)$ in a manner similar to used for syndrome computations, since it would takes $n - 1$ circuits like figure 1.3. While only two circuits like figure 1.4 are sufficient, if we use Chien’s search.

Remark 3.1.11. We note the BM decoding algorithm 3.1.9 is an example of decoder for Reed-Solomon codes in which the steps 3 and 4 of the general outline are not necessarily consecutive (see section 1.5). Indeed we note that if the computation of $\omega(x)$ is brought forward at the end of BM 2 then the evaluations of the polynomials $\omega(x)$ and $\sigma'(x)$ in $\alpha^i$ can be simultaneously implemented with the same evaluation of $\sigma(x)$ done in step BM.3. Moreover, each time that the evaluation of $\sigma(x)$ gives a value equal to zero, the division necessary to calculate $E_i$ in BM.4 can be implement in a separate divider. For example if $\sigma(\alpha) = 0$, then the divider calculates $\frac{\omega(\alpha)}{\sigma'(\alpha)}$ in BM.4 while the circuits that implement Chien’s search continue to evaluate $\sigma(x)$, $\omega(x)$, and $\sigma'(x)$ in $\alpha^2$, $\alpha^3$, … Thanks to this pipelined strategy, after the $n$ loops in which Chien’s search evaluates the polynomials $\sigma(x)$, $\omega(x)$, and $\sigma'(x)$ in all the field elements, some of the divisions needed to compute the error values may have been already executed. Thus in practice the number of divisions that remain to perform in BM.4, after the end of step BM.3, is less than $e$.

Remark 3.1.12. One last observation on the polynomial evaluation: if the field $\mathbb{F}_q$ has characteristic equal to 2, then

$$\sigma'(x) = \sum_{\substack{j=1
\text{j odd}}}^e \sigma_j x^j$$
Thus the value $\sigma'(\alpha^i)$ can be obtained as a byproduct of the computation of $\sigma(\alpha^i)$ summing the odd terms of the latter.

In addition, we show with the next proposition that it is possible to obtain a new formula to calculate error values in which the error-evaluator polynomial $\omega(x)$ doesn’t appear.

**Proposition 3.1.13.** Let $\tau^{(d-1)}(x)$ be the last polynomial of the polynomial sequence $\{\tau^{(j)}(x)\}_{j=0}^{d-1}$ (definition 3.1.3) and let $\sigma'(x)$ be the formal derivative of the error-locator polynomial $\sigma(x)$. Then for any $i \in \{1, 2, \ldots, e\}$, we have that

$$ E_i = -\frac{(X_i^{-1})^{d-2}}{\sigma'(X_i^{-1})\tau^{(d-1)}(X_i^{-1})} \quad (3.8) $$

**Proof.** We write the relation (3.4) seen in proposition 3.1.4 setting $i = d - 1$ and we obtain that

$$ \omega(x)\tau^{(d-1)}(x) - \sigma(x)\gamma^{(d-1)}(x) = x^{d-2} $$

Evaluating the former equality in $X_i^{-1}$, we have

$$ \omega(X_i^{-1}) = \frac{(X_i^{-1})^{d-2}}{\tau^{(d-1)}(X_i^{-1})} $$

At last substituting this expression in Forney’s formula we can conclude the proof.
3.1. BM DECODING ALGORITHM

For this reason, it is possible to implement the BM decoder without computing the error-evaluator polynomial \( \omega(x) \) saving memory space and operations. In this case we can substitute step BM.4 with

\[
\text{BM.4b for } i = 1, 2, \ldots, e \quad \text{do}
\]

\[
E_i := -\frac{(X_i^{-1})^{d-2}}{\sigma'(X_i^{-1})\tau^{(d-4)}(X_i^{-1})};
\]

endfor

Step BM.4b can be implemented via the same pipelined strategy described in remark 3.1.11 for BM.4. In this way the polynomial evaluation needed by (3.8) are simultaneously implemented with step BM.3 and afterwards step BM.4b requires at most \( e \) divisions and \( e \) multiplications.

We give an example of decoding of a Reed-Solomon code using the BM decoding algorithm 3.1.9:

**Example 3.1.14.** Let \( \alpha \) be a primitive element of \( \mathbb{F}_{2^4} \) satisfying \( \alpha^4 + \alpha + 1 = 0 \).

Consider over \( \mathbb{F}_{2^4} \) the code, \( RS(15, 9, \alpha) \) generated by

\[
g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^8)
\]

The code has distance 9, so \( t = 4 \). Suppose that the sent codeword is \( c = 0 \) and the error vector is

\[
e(x) = \alpha^2 x^2 + \alpha x^8 + \alpha^7 x^{13}
\]

Clearly \( r(x) = e(x) \) and the syndrome values are:

\[
S_1 = r(\alpha) = \alpha^{12}; \quad S_2 = r(\alpha^2) = 0; \quad S_3 = r(\alpha^3) = 0; \quad S_4 = r(\alpha^4) = \alpha^5; \quad S_5 = r(\alpha^5) = \alpha^{11}; \quad S_6 = r(\alpha^6) = \alpha^{13}; \quad S_7 = r(\alpha^7) = \alpha^3; \quad S_8 = r(\alpha^8) = \alpha
\]

Hence the iterations in BM.2 are:

\[
(i = 0) \rightarrow \Delta_0 = S_1 = \alpha^{12}; \\
\sigma^{(1)}(x) = \sigma^{(0)}(x) - \Delta_0 x \tau^{(0)} = 1 + \alpha^{12} x; \\
\Delta_0 \neq 0 \text{ and } 2D(0) < 1 \implies D(1) = 1; \\
\tau^{(1)}(x) = \frac{1}{\Delta_0} = \alpha^3;
\]

\[
(i = 1) \rightarrow \Delta_1 = S_2 + S_1 \sigma^{(1)} = \alpha^9; \\
\sigma^{(2)}(x) = \sigma^{(1)}(x) - \Delta_1 x \tau^{(1)} = 1; \\
2D(1) \geq 2 \implies D(2) = 1; \\
\tau^{(2)}(x) = \alpha^3 x;
\]
At the end $BM$ sets $e = D(8) = 3$ and

$$\sigma(x) = \alpha^8x^3 + \alpha^9x^2 + \alpha^6x + 1 = (1 + \alpha^2x)(1 + \alpha^8x)(1 + \alpha^{13}x)$$

After that $BM$ calculates that

$$X_1 = \alpha^2, \quad X_2 = \alpha^8, \quad X_3 = \alpha^{13}$$
Hence the error positions are
\[ p_1 = 2, \quad p_2 = 8, \quad p_3 = 13 \]
Finally in BM\[4\] the error-evaluator polynomial is computed
\[ \omega(x) = \alpha^6 x^2 + \alpha^3 x + \alpha^{12} \]
and the error values are found by Forney’s formula:
\[ E_1 = \alpha^2, \quad E_2 = \alpha, \quad E_3 = \alpha^7 \]
Thus
\[ e(x) = E_1 x^{p_1} + E_2 x^{p_2} + E_3 x^{p_3} = \alpha^2 x^2 + \alpha x^8 + \alpha^7 x^{13} \]
and the received word \( r \) is correctly decoded in \( c = r - e = 0 \).

**Inversionless BM decoder**

As remembered in section 1.5, the inversion is one of the most complicated operations among those of finite field arithmetic. For this reason, an implementation in which inversions can be replaced by some multiplications is preferable.

For the BM decoder the discrepancy inversions executed the step BM.2 can be avoided introducing the following tools:

**Definition 3.1.15.** We recursively define the function \( \beta : \mathbb{N} \to \mathbb{F}_q^* \) and the polynomial families \( \{ \hat{\sigma}(i)(x) \}_{i=0}^{d-1} \) and \( \{ \hat{\tau}(i)(x) \}_{i=0}^{d-1} \) as \( \beta(0) = 1, \hat{\sigma}(0)(x) = 1, \hat{\tau}(0)(x) = 1 \) and
\[
\hat{\sigma}(i+1)(x) = \beta(i)\hat{\sigma}(i)(x) - \hat{\Delta}_i x \hat{\tau}(i)(x) \\
\hat{\tau}(i+1)(x) = \begin{cases} 
  x \hat{\tau}(i)(x) & \text{if } \hat{\Delta}_i = 0 \text{ or } 2D(i) \geq i + 1 \\
  \hat{\sigma}(i)(x) & \text{if } \hat{\Delta}_i \neq 0 \text{ and } 2D(i) < i + 1 
\end{cases} \\
\beta(i + 1) = \begin{cases} 
  \beta(i) & \text{if } \hat{\Delta}_i = 0 \text{ or } 2D(i) \geq i + 1 \\
  \hat{\Delta}_i & \text{if } \hat{\Delta}_i \neq 0 \text{ and } 2D(i) < i + 1 
\end{cases}
\]
where \( \hat{\Delta}_i \) is the coefficient of the term \( x^i \) in the polynomial \( \hat{\sigma}(i)(x) S(x) \).

It is immediate to prove by induction that:
Proposition 3.1.16. If \( i \in \{0, 1, \ldots, d - 1\} \), then

\[
\hat{\tau}^{(i)}(x) = \beta(i) \tau^{(i)}(x) \quad \text{and} \quad \hat{\sigma}^{(i)}(x) = \left( \prod_{j=0}^{i-1} \beta(j) \right) \sigma^{(i)}(x)
\]

Proof. The base case is trivial. Suppose that the equalities are true for \( i \), then

\[
\hat{\Delta}_i = \left( \prod_{j=0}^{i-1} \beta(j) \right) \Delta_i
\]

and thus

\[
\hat{\sigma}^{(i+1)}(x) = \beta(i) \hat{\sigma}^{(i)}(x) - \hat{\Delta}_i x \hat{\tau}^{(i)}(x) = \\
= \beta(i) \left( \prod_{j=0}^{i-1} \beta(j) \right) \sigma^{(i)}(x) - \left( \prod_{j=0}^{i-1} \beta(j) \right) \Delta_i x \beta(i) \tau^{(i)}(x) = \\
= \left( \prod_{j=0}^{i} \beta(j) \right) \left( \sigma^{(i)}(x) - \Delta_i x \tau^{(i)}(x) \right) = \left( \prod_{j=0}^{i} \beta(j) \right) \sigma^{(i+1)}
\]

Moreover we have that either

\[
\hat{\tau}^{(i+1)}(x) = x \hat{\tau}^{(i)}(x) = x \beta(i) \tau^{(i)}(x) = \beta(i + 1) \tau^{(i+1)}(x)
\]

or

\[
\hat{\tau}^{(i+1)}(x) = \hat{\sigma}^{(i)}(x) = \left( \prod_{j=0}^{i-1} \beta(j) \right) \sigma^{(i)}(x) = \frac{\hat{\Delta}_i}{\Delta_i} \sigma^{(i)}(x) = \beta(i + 1) \tau^{(i+1)}(x)
\]

This concludes the proof.

The \( BM \) decoding algorithm 3.1.9 implemented with the polynomials \( \hat{\sigma}^{(i)}(x) \) and \( \hat{\tau}^{(i)}(x) \) in the place of the polynomials \( \sigma^{(i)}(x) \) and \( \tau^{(i)}(x) \) actually finds scalar multiples \( b \cdot \sigma(x) \) and \( b \cdot \omega(x) \) instead of the polynomials \( \sigma(x) \) and \( \omega(x) \), where

\[
b = \prod_{j=0}^{d-2} \beta(j)
\]

However, it is obvious that Chien’s search will find the same error positions and it follows from Forney’s formula (1.2) that the same error values are obtained. While formula (3.8) has to be modified in

\[
E_i = -\frac{b \beta(d-1) (X_i^{-1})^{d-2}}{\sigma'(X_i^{-1}) \hat{\tau}'(d-1)(X_i^{-1})}
\]
3.2 Avoiding Malfunctions in \textit{BM}

In this section we will study the necessary and sufficient conditions to make the \textit{BM} decoding algorithm 3.1.9 for Reed-Solomon codes a $t$-bounded distance decoder.

Consider again the code $RS(n, d, \alpha)$ with error correction capability equal to $t$. Let $c$, $e$ and $r = c + e$ respectively be the transmitted codeword, the vector error and the received word. The input given to the \textit{BM} decoding algorithm 3.1.9 is $r$ and no assumption is made on the weight of $e$. Let $B$ be the set defined in section 1.1, we have already shown that if $r \in B$ the \textit{BM} decoding algorithm 3.1.9 either decodes correctly $r$ (when $wt(e) \leq t$) or runs into an inevitable decoder error occurs (if $wt(e) > t$). It remains to study what happens when $r \notin B$. For this aim we denote:

- $\tilde{c}$ the degree of $\sigma^{(d-1)}(x)$ and
  
  \begin{equation*}
  \sigma^{(d-1)}(x) = \sigma^{(d-1)}_0 x^\tilde{c} + \sigma^{(d-1)}_{-1} x^{\tilde{c}-1} + \cdots + \sigma^{(d-1)}_1 x^1 + 1,
  \end{equation*}

- $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_{\tilde{c}}$ the inverses of the roots of $\sigma^{(d-1)}(x)$ calculated in \textit{BM} 3,

- $\tilde{E}_i$ the $i$th assumed error value,

- $\tilde{e}$ the assumed error vector.

 Moreover we define $\tilde{S}_i = \tilde{e}(\alpha^i)$ and $\tilde{S}(x) = \tilde{S}_1 + \tilde{S}_2 x + \cdots + \tilde{S}_{d-1} x^{d-1}$. Finally, let $\bar{e} = r - \tilde{e}$ be again the output vector.

First of all we give two example of possible malfunctions for the \textit{BM} decoder.

\textbf{Example 3.2.1.} Let $\alpha$ be a primitive element of $F_{2^4}$ satisfying $\alpha^4 + \alpha + 1 = 0$. Consider over $F_{2^4}$ the code, $RS(15, 5, \alpha)$ generated by

\begin{equation*}
  g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)
\end{equation*}

The code has distance 5, so $t = 2$. Suppose that the sent codeword is $c = 0$ and the error vector is

\begin{equation*}
  e(x) = \alpha^6 + \alpha^3 x + \alpha^4 x^2 + x^7
\end{equation*}

Clearly $r(x) = e(x)$, $e = 4$ and the syndrome values are:

\begin{align*}
  S_1 &= r(\alpha) = \alpha^3 & S_2 &= r(\alpha^2) = \alpha^5 \\
  S_3 &= r(\alpha^3) = \alpha^7 & S_4 &= r(\alpha^4) = \alpha^8
\end{align*}

The iterations \textit{BM} 2 are:
(i = 0) \to \Delta_0 = S_1 = \alpha^3;
\sigma^{(1)}(x) = \sigma^{(0)}(x) - \Delta_0 \tau^{(0)} = 1 + \alpha^3 x;
\Delta_0 \neq 0 \text{ and } 2D(0) < 1 \implies D(1) = 1;
\tau^{(1)}(x) = \frac{1}{\Delta_0} = \alpha^{12};

(i = 1) \to \Delta_1 = S_2 + S_1 \sigma^{(1)}_1 = \alpha^5 + \alpha^6 = \alpha^8;
\sigma^{(2)}(x) = \sigma^{(1)}(x) - \Delta_1 \tau^{(1)} = 1 + \alpha^2 x;
2D(1) \geq 2 \implies D(2) = 1;
\tau^{(2)}(x) = \alpha^{12} x;

(i = 2) \to \Delta_2 = S_3 + S_2 \sigma^{(2)}_1 = \alpha^7 + \alpha^7 = 0;
\sigma^{(3)}(x) = \sigma^{(2)}(x) = 1 + \alpha^2 x;
\Delta_2 = 0 \implies D(3) = 1;
\tau^{(3)}(x) = \alpha^{12} x^2;

(i = 3) \to \Delta_3 = S_4 + S_3 \sigma^{(3)}_1 = \alpha^8 + \alpha^9 = \alpha^{12};
\sigma^{(4)}(x) = \sigma^{(3)}(x) - \Delta_3 \tau^{(3)} = 1 + \alpha^2 x + \alpha^9 x^3;
\Delta_3 \neq 0 \text{ and } 2D(3) < 4 \implies D(4) = 3;
\tau^{(4)}(x) = \frac{\sigma^{(3)}(x)}{\Delta_3} = \alpha^3 + \alpha^5 x;

Hence the BM decoding algorithm 3.1.9 sets \tilde{e} = 3 and since
\sigma^{(4)}(x) = 1 + \alpha^2 x + \alpha^9 x^3 = \alpha^9 (x + \alpha^5) (x^2 + \alpha^5 x + \alpha)

BM cannot proceed because the polynomial \(x^2 + \alpha^5 x + \alpha\) is irreducible over \(F_q[x]\). In this case the BM decoder have to declare a decoder failure. Note that \(\tilde{e} = D(4)\) but \(\tilde{e} > t\).

Example 3.2.2. Consider the same code of example 3.2.1, supposing that \(c = 0\) and
\(e(x) = \alpha^3 x + \alpha x^2 + x^{10}\)
then \(r(x) = e(x), e = 4\) and the syndrome values are:
\[S_1 = r(\alpha) = \alpha^6, \quad S_2 = r(\alpha^2) = \alpha^5, \quad S_3 = r(\alpha^3) = \alpha^5, \quad S_4 = r(\alpha^4) = \alpha^5\]
The iterations BM in this case are:
3.2. AVOIDING MALFUNCTIONS IN BM

\( (i = 0) \rightarrow \Delta_0 = S_1 = \alpha^6; \)
\( \sigma^{(1)}(x) = \sigma^{(0)}(x) - \Delta_0 x \tau^{(0)} = 1 + \alpha^6 x; \)
\( \Delta_0 \neq 0 \text{ and } 2D(0) < 1 \implies D(1) = 1; \)
\( \tau^{(1)}(x) = \frac{1}{\Delta_0} = \alpha^9; \)

\( (i = 1) \rightarrow \Delta_1 = S_2 + S_1 \sigma^{(1)}_1 = \alpha^5 + \alpha^2 = \alpha^8; \)
\( \sigma^{(2)}(x) = \sigma^{(1)}(x) - \Delta_1 x \tau^{(1)} = 1 + \alpha^{14} x; \)
\( 2D(1) \geq 2 \implies D(2) = 1; \)
\( \tau^{(2)}(x) = \alpha^9 x; \)

\( (i = 2) \rightarrow \Delta_2 = S_3 + S_2 \sigma^{(2)}_1 = \alpha^5 + \alpha^2 = \alpha^8; \)
\( \sigma^{(3)}(x) = \sigma^{(2)}(x) - \Delta_2 x \tau^{(2)} = 1 + \alpha^{14} x + \alpha^2 x^2; \)
\( \Delta_2 \neq 0 \text{ and } 2D(2) < 3 \implies D(3) = 2; \)
\( \tau^{(3)}(x) = \frac{\sigma^{(2)}(x)}{\Delta_2} = \alpha^7 + \alpha^6 x; \)

Thus \( \tilde{e} = 1 \) and, since \( \sigma^4(x) = 1 + x \), BM calculates that \( \tilde{X}_1 = 1 = \alpha^0 \). The assumed error position is \( j_1 = 0 \). Finally in BM the algorithm calculate the error-evaluator polynomial \( \omega(x) = \alpha^6 \) and the assumed error value \( \tilde{E}_1 = \alpha^6 \). Thus \( \tilde{e} = \tilde{E}_1 x^{j_1} = \alpha^6 \) and the received word \( r \) is wrongly decoded in

\[ \tilde{c} = r - \tilde{e} = \alpha^6 + \alpha^3 x + \alpha x^2 + x^{10} \]

Evidently a decoder malfunction has happened. Note that \( D(4) = 2 > \tilde{e} = 1. \)

In both the examples it occurs that \( r \notin B \) but the BM decoder does not declare the failure and carries on the decoding, bringing about a decoder malfunction. In order to avoid such behavior of the BM decoder we have to add to its implementations the conditions necessary and sufficient to assure that \( r \in B \). For this reason we give the following proposition.

**Proposition 3.2.3.** With previous notations, we have that \( r \in B \) if and only if

\[ \sigma^{(1)}(x) = \sigma^{(0)}(x) - \Delta_0 x \tau^{(0)} = 1 + \alpha^6 x; \]
\( \Delta_0 \neq 0 \text{ and } 2D(0) < 1 \implies D(1) = 1; \)
\( \tau^{(1)}(x) = \frac{1}{\Delta_0} = \alpha^9; \)
Thus we have

(1) $\sigma^{(d-1)}(x)$ has exactly $\tilde{e}$ distinct roots, all belonging to the field $\mathbb{F}_q$ and all different from zero;

(2) $D(d-1) = \tilde{e}$ and $\tilde{e} \leq t$;

Proof. If $\mathbf{r} \in \mathcal{B}$, then there exists a unique $\mathbf{c}_1 \in RS(n, d, \alpha)$ such that $\mathbf{r} \in \overline{B}_i(\mathbf{c}_1)$. We repeat the construction saw in section 3.1 with $\mathbf{c}_1$ in place of $\mathbf{c}$. Since $wt(\mathbf{r} - \mathbf{c}_1) \leq t$, then $\mathbf{c}_1$ is equal to the output vector $\tilde{\mathbf{c}}$ and the right side conditions are accomplished. On the other hand, we suppose that $\tilde{\mathbf{r}} \in \mathcal{B}$. Since $\mathbf{r} = \tilde{\mathbf{r}} + \tilde{\mathbf{c}}$ and $\tilde{\mathbf{c}} \leq t$, we need only to prove that $\tilde{\mathbf{c}} \in RS(n, d, \alpha)$. For condition (1) we may assume that $\sigma^{(d-1)}(x) = \prod_{i=1}^{\tilde{e}} (1 - \tilde{X}_i x)$ and that $\tilde{S}_i = \tilde{\mathbf{c}}(a^i) = \sum_{j=1}^{\tilde{e}} \tilde{E}_j \tilde{X}_j^i$ for any $i = 1, 2, \ldots, \tilde{e}$. Moreover we know that $\tilde{E}_i = -\frac{\omega^{(d-1)}(\tilde{X}_j^{-1})}{(\sigma^{(d-1)})'(\tilde{X}_j^{-1})}$ for any $i = 1, 2, \ldots, \tilde{e}$. Then

$$
\sigma^{(d-1)}(x) \tilde{S}(x) = \prod_{i=1}^{\tilde{e}} \left(1 - \tilde{X}_i x\right) \left(\sum_{l=1}^{d-1} x^{l-1} \sum_{j=1}^{\tilde{e}} \tilde{E}_j \tilde{X}_j^l\right) = \\
= \prod_{i=1}^{\tilde{e}} \left(1 - \tilde{X}_i x\right) \left(\sum_{j=1}^{\tilde{e}} \tilde{E}_j \sum_{l=1}^{d-1} x^{l-1} \tilde{X}_j^l\right) = \\
= \sum_{j=1}^{\tilde{e}} \left(\tilde{E}_j \prod_{i=1}^{\tilde{e}} \left(1 - \tilde{X}_i x\right) \sum_{l=1}^{d-1} x^{l-1} \tilde{X}_j^l\right) = \\
= \sum_{j=1}^{\tilde{e}} \left(-\frac{\omega^{(d-1)}(\tilde{X}_j^{-1})}{(\sigma^{(d-1)})'(\tilde{X}_j^{-1})} \prod_{i=1}^{\tilde{e}} \left(1 - \tilde{X}_i x\right) \sum_{l=1}^{d-1} x^{l-1} \tilde{X}_j^l\right)
$$

Now we observe that the sum on $l$ is equal to $\tilde{X}_j \frac{1 - (\tilde{X}_j x)^{d-1}}{1 - X_j x}$ and that

$$
(\sigma^{(d-1)})'(\tilde{X}_j^{-1}) = -\tilde{X}_j \prod_{i=1, i \neq j}^{\tilde{e}} \left(1 - \tilde{X}_i \tilde{X}_j^{-1}\right).
$$

Thus we have

$$
\sigma^{(d-1)}(x) \tilde{S}(x) = \sum_{j=1}^{\tilde{e}} \left(\omega^{(d-1)}(\tilde{X}_j^{-1}) \prod_{i \neq j} \frac{1 - \tilde{X}_i x}{1 - X_i X_j} \left(1 - (\tilde{X}_j x)^{d-1}\right)\right) = 
$$
The first sum on the right side is the Lagrange interpolation formula for the unique polynomial that has degree less than $\tilde{e}$ and takes value $\omega^{(d-1)}(\tilde{X}_j)$ at $\tilde{X}_j$ for $i = 1, 2, \ldots, \tilde{e}$. Since $\deg \omega^{(d-1)}(x) \leq D(d-1) - 1 = \tilde{e} - 1$, this unique polynomial must be $\omega^{(d-1)}(x)$ itself. So we obtain that

$$
\sigma^{(d-1)}(x) \tilde{S}(x) \equiv \omega^{(d-1)}(x) \pmod{x^{d-1}}
$$

But for proposition 3.1.3 we know that

$$
\sigma^{(d-1)}(x) S(x) \equiv \omega^{(d-1)}(x) \pmod{x^{d-1}}
$$

Hence it follows that $S(x) = \tilde{S}(x)$, that is

$$
S_i - \tilde{S}_i = r(\alpha^i) - \tilde{e}^i = \tilde{c}(\alpha^i) = 0
$$

for any $i = 1, 2, \ldots, d - 1$. This concludes the proof.

Thus we can conclude that adding the conditions (1) and (2) of proposition 3.2.3 at the end of step $BM$ 2 the $BM$ decoding algorithm 3.1.9 becomes a $t$-bounded distance decoding algorithm.

Observe that since the elements $\tilde{X}_1, \tilde{X}_2, \ldots, \tilde{X}_{\tilde{e}}$ are the inverses of the roots of $\sigma^{(d-1)}(x)$, by (3.4) it follows that

$$
\tilde{E}_i = -\frac{\omega^{(d-1)}(\tilde{X}_i^{-1})}{(\sigma^{(d-1)})'(\tilde{X}_i^{-1})} = -\frac{(\tilde{X}_i^{-1})^{d-2}}{(\sigma^{(d-1)})'(\tilde{X}_i^{-1}) \cdot \tau^{(d-1)}(\tilde{X}_i^{-1})}
$$

for any $i = 1, 2, \ldots, \tilde{e}$. Hence the proposition 3.2.3 also holds for the implementation of the $BM$ decoder with the formula (3.8) in the place of Forney’s formula (1.2) (see proposition 3.1.13).

### 3.3 Comparing $fPGZ.2$ and $BM.2$

From a computational point of view, when the average number of errors is much less than the error correction capability $t$, then the $fPGZ$ decoding algorithm 2.2.5 (on page 32) has some significant advantages in comparison with the $BM$ decoding algorithm 3.1.9 (on page 50). In fact, whatever the number of errors is, the complexity of $BM$ decoding algorithm 3.1.9 is of
order $O(t^2)$, whereas the complexity of $fPGZ$ decoding algorithm 2.2.5 is of order $O(et)$ and so it depends on $e$. Nevertheless the program structure in the $fPGZ$ decoding algorithm 2.2.5, especially when the singularity gap is strictly greater than 1, is more complicated and requires more program code than the $BM$ decoding algorithm 3.1.9.

Now we want to examine and compare the structure of the two iterative algorithms in order to find some common features and computations. For this purpose we state the following two remarks and theorem 3.3.3.

**Remark 3.3.1.** We recall the one-to-one correspondence between vectors in $(\mathbb{F}_q)^i$ and polynomials in $\mathbb{F}_q[x]$ of degree less than $i$ studied in section 1.3 and given by

$$v = (v_0, v_1, \ldots, v_{i-1}) \iff v(x) = \sum_{j=0}^{i-1} v_j x^j = v_0 + v_1 x + \cdots + v_{i-1} x^{i-1}$$

and we note that there exists another possible correspondence. Indeed if $v = (v_0, v_1, \ldots, v_{i-1}) \in (\mathbb{F}_q)^i$ then we will call $\overline{v}(x)$ the unique polynomial of $\mathbb{F}_q[x]$ of degree less than $i$ given by:

$$\overline{v}(x) = \sum_{j=0}^{i-1} v_j x^{i-1-j} = v_{i-1} + v_{i-2} x + \cdots + v_0 x^{i-1}$$

Usually $\overline{v}(x)$ is called the reciprocal polynomial of $v(x)$. Observe that

$$\overline{v}(x) = x^{i-1} \cdot v(x^{-1})$$

**Remark 3.3.2.** Let $i$ be an index less or equal to $t$ and such that $\det(A_i) \neq 0$. Recall that in section 2.2 we have defined $w^{(i)} = \left( w^{(i)}_0, w^{(i)}_1, \ldots, w^{(i)}_{i-1} \right)^T$ as the vector in $(\mathbb{F}_q)^i$ that satisfies

$$A_i w^{(i)} = - \begin{pmatrix} S_{i+1} \\ S_{i+2} \\ \vdots \\ S_{2i} \end{pmatrix} \quad (3.9)$$

Now we introduce the polynomial

$$P_{w^{(i)}}(x) \overset{def}{=} x \cdot \overline{w^{(i)}}(x) + 1 = w^{(i)}_0 x^i + w^{(i)}_1 x^{i-1} + \cdots + w^{(i)}_{i-1} x + 1$$
and we observe that for any \( j \in \{1, 2, \ldots, t - i\} \), the element \( \varepsilon_j \), defined as the sum
\[
\varepsilon_j = S_{i+j} w_0^{(i)} + S_{i+j+1} w_1^{(i)} + \cdots + S_{2i+j-1} w_{i-1}^{(i)} + S_{2i+j} \in \mathbb{F}_q
\]
(see definition 2.2.1) is equal to the coefficient of \( x^{2i+j-1} \) in \( P_{w^{(i)}}(x)S(x) \).

We further recall that the vectors \( w^{(i)} \)'s are computed in step \( fPGZ \) of the \( fPGZ \) decoding algorithm 2.2.5. While the \( BM \) decoding algorithm 3.1.9 in step \( BM \) 2 calculates the polynomials \( \sigma^{(i)}(x) \)'s as in definition 3.1.1. Finally we remark that for any \( j \in \{0, 1, \ldots, d - 1\} \), \( \Delta_j \) is defined as the coefficient of \( x^{j} \) in the polynomial \( \sigma^{(j)}(x)S(x) \).

**Theorem 3.3.3.** Let \( i \) an index less or equal to \( t \) and such that \( \det(A_i) \neq 0 \) and let \( w^{(i)} \) be the vector in \( (\mathbb{F}_q)^i \) that satisfies 3.3. If \( r \) is the singularity gap as in definition 2.2.1, i.e.
\[
r = \min \{ j \in \{1, 2, \ldots, t - i\} \mid \varepsilon_j \neq 0 \} \in \mathbb{N}^+
\]
then we have that:
\[
P_{w^{(i)}}(x) = \sigma^{(2i)}(x) = \sigma^{(2i+1)}(x) = \cdots = \sigma^{(2i+r-1)}(x)
\]
Moreover it follows from this that \( \Delta_{2i+r-1} = \varepsilon_r \neq 0 \) and that if \( r \geq 2 \), then \( \Delta_2 = \Delta_{2+1} = \cdots = \Delta_2+r-2 = 0 \).

**Proof.** From 3.3.9 and \( \varepsilon_1 = \varepsilon_2 = \cdots = \varepsilon_{r-1} = 0 \) it follows that
\[
\begin{pmatrix}
S_1 & \cdots & S_1 \\
\vdots & & \vdots \\
S_i & \cdots & S_{2i-1} \\
S_{i+1} & \cdots & S_{2i} \\
\vdots & & \vdots \\
S_{i+r-1} & \cdots & S_{2i-r-2}
\end{pmatrix}
\begin{pmatrix}
w^{(i)}
\end{pmatrix}
= -
\begin{pmatrix}
S_{i+1} \\
\vdots \\
S_{2i} \\
S_{2i+1} \\
\vdots \\
S_{2i+r-1}
\end{pmatrix}
\]
that is
\[
\sum_{l=0}^{i-1} S_{j+l} w_l^{(i)} + S_{i+j} = 0 \quad \forall \ j = 1, 2, \ldots, i + r - 1
\]
Since \( S(x) = \sum_{j=1}^{d-1} S_j x^{j-1} \) and \( P_{w^{(i)}}(x) = w_0^{(i)} x^i + w_1^{(i)} x^{i-1} + \cdots + w_{i-1}^{(i)} x + 1 \), the sums on the right side in the former equalities are the coefficients of
CHAPTER 3. BERLEKAMP-MASSEY DECODER

$x^i, x^{i+1}, \ldots, x^{2i+r-2}$ in the polynomial $P_{w(i)}(x)S(x)$. These coefficients are all zero, hence there exists $f^{(i)}(x) \in \mathbb{F}_q[x]$ such that $\deg(f^{(i)}) \leq i - 1$ and

$$P_{w(i)}(x)S(x) \equiv f^{(i)}(x) \pmod{x^{2i+j}} \quad \forall \ j = 0, 1, \ldots, r - 1$$

So we have obtained that $(P_{w(i)}(x), f^{(i)}(x)) \in \mathcal{M}_{2i+j}, \ \forall \ j = 0, 1, \ldots, r - 1$. Now we may use the corollary 3.1.7 to conclude that, for any $j \neq 0, 1, \ldots, r - 1$

$$\begin{align*}
\begin{cases}
P_{w(i)}(x) = u(x)\sigma^{(2i+j)}(x) \\
f^{(i)}(x) = u(x)\omega^{(2i+j)}(x)
\end{cases}
\end{align*}$$

(3.10)

where $u(x) = \gcd(P_{w(i)}(x), f^{(i)}(x))$.

It immediately follows from the first equations of systems (3.10) that

$$\sigma^{(2i)}(x) = \sigma^{(2i+1)}(x) = \ldots = \sigma^{(2i+r-1)}(x)$$

and therefore, using that $\sigma^{(2i+1)}(x) - \sigma^{(2i)}(x) = \Delta_2 x^{2i} \tau^{(i)}(x)$ we find that if $r \geq 2$, then $\Delta_2 = \Delta_2 + 1 = \ldots = \Delta_2 + r - 2 = 0$. It also follows from (3.10) with $j = 0$ that

$$\deg(\sigma^{(2i)}(x)) \leq i \quad \text{and} \quad \deg(\omega^{(2i)}(x)) \leq i - 1.$$

In particular, due to the relation $\sigma^{(2i)}(x)S(x) \equiv \omega^{(2i)}(x) \pmod{x^{2i}}$, the upper bound of $\deg(\omega^{(2i)})$ implies that the coefficients of $x^i, x^{i+1}, \ldots, x^{2i-1}$ in $\sigma^{(2i)}(x)S(x)$ are all zero. Hence, we may write

$$\sigma^{(2i)}(x) = \sigma^{(2i)}_i x^i + \sigma^{(2i)}_{i-1} x^{i-1} + \ldots + \sigma^{(2i)}_1 x + 1$$

and

$$\begin{align*}
\begin{cases}
S_{i+1} + \sigma^{(2i)}_i S_i + \ldots + \sigma^{(2i)}_1 S_1 &= 0 \\
S_{i+2} + \sigma^{(2i)}_i S_{i+1} + \ldots + \sigma^{(2i)}_1 S_2 &= 0 \\
\vdots \\
S_{2i} + \sigma^{(2i)}_i S_{2i-1} + \ldots + \sigma^{(2i)}_1 S_i &= 0
\end{cases}
\end{align*}$$

that is

$$A_i \begin{pmatrix} \sigma^{(2i)}_i \\ \sigma^{(2i)}_{i-1} \\ \vdots \\ \sigma^{(2i)}_1 \end{pmatrix} = - \begin{pmatrix} S_{i+1} \\ S_{i+2} \\ \vdots \\ S_{2i} \end{pmatrix}$$

Since $\det(A_i) \neq 0$, from the definition of the vector $w^{(i)}$ we deduce that

$$w^{(i)} = \begin{pmatrix} \sigma^{(2i)}_i \\ \sigma^{(2i)}_{i-1} \\ \vdots \\ \sigma^{(2i)}_1 \end{pmatrix}$$
Thus $P_{w(i)}(x) = \sigma^{(2i)}(x)$.

Finally we will show that $\Delta_{2i+r-1} = \varepsilon_r$. As we saw in remark 3.3.2, $\varepsilon_r$ is the coefficient of $x^{2i+r-1}$ in the polynomial $P_{w(i)}(x)S(x)$, while (3.2) on page 44 defines $\Delta_{2i+r-1}$ as the coefficient of $x^{2i+r-1}$ in the polynomial $\sigma^{(2i+r-1)}(x)S(x)$. Since $P_{w(i)}(x) = \sigma^{(2i+r-1)}(x)$, it follows that $\Delta_{2i+r-1} = \varepsilon_r$, which is different from zero by definition.

Theorem 3.3.3 shows that every vector $w^{(i)}$ computed during the step $f \text{PGZ}$.2 corresponds always to the polynomial $\sigma^{(2i)}(x)$, in the sense that the components of $w^{(i)}$ are the coefficients of $\sigma^{(2i)}(x)$. In other words the intermediate outcomes of the $f \text{PGZ}$ decoding algorithm are a subset of the intermediate outcomes of the $BM$ decoder. The vice versa is not true, indeed, as we can see comparing example 2.2.6 and example 3.1.14, some of polynomials $\sigma^{(j)}(x)$’s may not appear in the set of the polynomials $P_{w(j)}(x)$’s. In example 2.2.6 we decode the received word $r(x) = \alpha_2 x^2 + \alpha x^5 + \alpha_7 x^{13}$ by the $f \text{PGZ}$ decoder and we have as intermediate outcomes the polynomials

$$P_{w(1)}(x) = 0x + 1 = 1 \quad P_{w(3)}(x) = \alpha^8 x^3 + \alpha^9 x^2 + \alpha^6 x + 1$$

While, in example 3.1.14, decoding the same received word with the $BM$ decoder, we obtain the polynomials:

$$\sigma^{(0)}(x) = 1 \quad \sigma^{(1)}(x) = 1 + \alpha^{12} \quad \sigma^{(2)}(x) = \sigma^{(3)}(x) = 1$$

$$\sigma^{(4)}(x) = 1 + \alpha^8 x^3 \quad \sigma^{(5)}(x) = 1 + \alpha^6 x + \alpha^8 x^3$$

$$\sigma^{(6)}(x) = \sigma^{(7)}(x) = \sigma^{(8)}(x) = 1 + \alpha^6 x + \alpha^9 x^2 + \alpha^8 x^3$$

It easy verify that

$$P_{w(1)}(x) = \sigma^{(2)}(x) = \sigma^{(3)}(x)$$

$$P_{w(3)}(x) = \sigma^{(6)}(x) = \sigma^{(7)}(x) = \sigma^{(8)}(x)$$

and the polynomial $\sigma^{(1)}(x)$, $\sigma^{(4)}(x)$, $\sigma^{(5)}(x)$ do not corresponds to any polynomial of the form $P_{w(i)}(x)$.

Moreover theorem 3.3.3 states that when the singularity gap $r$ of the $i$th iterative step of $f \text{PGZ}$.2 is greater than 1 (that is the $f \text{PGZ}$ decoder “jumps” from $i$ to $i+r$) then even the iterative steps $BM$.2 “jump” from $2i$ to $2i+r$, since $\sigma^{(2i)}(x) = \sigma^{(2i+1)}(x) = \cdots = \sigma^{(2i+r-1)}(x)$.
Chapter 4

Error Value Formulas

In the former chapters we saw that both in the \textit{fPGZ} decoding algorithm \cite{2.2.5} and in the \textit{BM} decoding algorithm \cite{3.1.9} the error locations are determined by the roots of the error-locator polynomial \( \sigma(x) \) through an exhaustive search among the elements of \( \mathbb{F}_q \). Instead, the error values are calculated by different methods in each different decoding algorithm. In the \textit{fPGZ} \cite{4} the error values \( E_1, E_2, \ldots, E_e \) are calculated by the \textit{BP} algorithm \cite{2.1.3} to solve the following linear system

\[
\begin{pmatrix}
X_1 & X_2 & \cdots & X_e \\
X_1^2 & X_2^2 & \cdots & X_e^2 \\
\vdots & \vdots & \ddots & \vdots \\
X_1^e & X_2^e & \cdots & X_e^e
\end{pmatrix}
\begin{pmatrix}
E_1 \\
E_2 \\
\vdots \\
E_e
\end{pmatrix}
=
\begin{pmatrix}
S_1 \\
S_2 \\
\vdots \\
S_e
\end{pmatrix}.
\]

where \( X_i \) is the inverse of \( i \)th root of \( \sigma(x) \) and \( S_i \) is the \( i \)th syndrome (see definitions \cite{1.5.1} and \cite{1.5.3}). In the \textit{BM} decoding algorithm \cite{3.1.9} the error values are usually determined simultaneously to the error positions using the error-evaluator polynomial \( \omega(x) \) and Forney’s formula \cite{1.2} (see remark \cite{3.1.11}). But we have also shown in proposition \cite{3.1.13} that the computation of the error-evaluator polynomial in the \textit{BM} decoding algorithm can be avoided. Indeed, we proved that for any \( i \in \{1, 2, \ldots, e\} \):

\[
E_i = -\frac{(X_i^{-1})^{d-2}}{\sigma'((X_i^{-1})^{d-1})}\tau^{(d-1)}(X_i^{-1})
\]

where the polynomial \( \tau^{(d-1)}(x) \) is one of the auxiliary polynomials calculate in the \textit{BM} decoding algorithm.

The aim of this chapter is to complete the comparison between the \textit{fPGZ} and the \textit{BM} decoders showing how to calculate the error values in both the
decoding algorithms by using the linear algebra of the syndrome matrix and the byproducts of the computations for the error-locator polynomial.

\section{Horiguchi’s formula}

In \cite{Hor88} T. Horiguchi presents a new error-evaluation method for computing error values in decoding Reed-Solomon codes through the BM decoding algorithm using simply tools of linear algebra. In this section we presented this method in order to apply it both to the \textit{fPGZ} and the BM decoding algorithms. With this aim, we introduce the following tools:

\begin{definition}
For any $i \in \{1, 2, \ldots, t\}$, let $B_i$ be the $i \times (i+1)$ matrix defined by
\[
B_i = \begin{pmatrix}
S_1 & S_2 & \cdots & S_{i+1} \\
S_2 & S_3 & \cdots & S_{i+2} \\
\vdots & \vdots & \ddots & \vdots \\
S_i & S_{i+1} & \cdots & S_{2i}
\end{pmatrix}
\]
Moreover for any $j = 0, 1, \ldots, i$ we call $B_i^{(j)}$ the square $i \times i$ matrix obtained from $B_i$ removing the $(j+1)$th column and we define
\[
k_j^{(i)} = (-1)^{i+j} \det(B_i^{(j)})
\]
Finally we call $k^{(i)}$ the vector in $(\mathbb{F}_q)^{i+1}$ given by
\[
k^{(i)} = \begin{pmatrix}
k_0^{(i)} \\
k_1^{(i)} \\
\vdots \\
k_i^{(i)}
\end{pmatrix}
\]
\end{definition}

\begin{remark}
Note that for any $i \in \{2, 3, \ldots, t\}$ we have that
\[
k_i^{(i)} = (-1)^{2i} \det(B_i^{(i)}) = \det(A_i) = \sum_{j=0}^{i-1} S_{i+j} k_j^{(i-j)}
\]
where $A_i$ is the $i \times i$ leading principal minor of the syndrome matrix (see definition 2.1.1). Moreover for any $i \in \{1, 2, \ldots, t\}$ and $j \in \{1, 2, \ldots, i\}$, expanding the determinant of the following singular matrix along the last
4.1. HORIGUCHI’S FORMULA

We obtain that \( \sum_{l=0}^{i} S_{j+l}k_l^{(i)} = 0 \). Hence \( B_i k_l^{(i)} = 0 \) for any \( i \in \{1, 2, \ldots, t\} \).

We recall that \( c \in RS(n, d, \alpha) \), \( e \) and \( r = c + e \) respectively denote the transmitted codeword, the vector error and the received word. Whereas \( t \) is the error correction capability of the code \( RS(n, d, \alpha) \). From now on, throughout this chapter, we suppose that \( e = wt(e) > 1 \).

**Proposition 4.1.3.** If \( e = wt(e) \), then

\[
\dim(\ker(B_e)) = \dim(\ker(B_{e-1})) = 1
\]

In particular \( \ker(B_e) = \text{span}\{k^{(e)} \} \) and \( \ker(B_{e-1}) = \text{span}\{k^{(e-1)} \} \).

**Proof.** We saw in proposition 2.1.2 that the square matrix \( A_e \) is non-singular, so the matrix \( B_e \) has rank equal to \( e \) and it immediately follows that the dimension of \( \ker(B_e) \) is equal to 1. Moreover as \( k_e^{(e)} = \det(A_e) \neq 0 \) and \( k_e^{(e)} \in \ker(B_e) \), it is clear that \( \ker(B_e) = \text{span}\{k^{(e)} \} \). Since

\[
k_e^{(e)} = \sum_{j=0}^{e-1} S_{e+j} k_j^{(e-1)}
\]

from \( k_e^{(e)} \neq 0 \) it follows that there is an index \( l \in \{1, 2, \ldots, e\} \) such that \( k_l^{(e-1)} \neq 0 \). Therefore, \( k_l^{(e-1)} \neq 0 \) and there is a \( (e-1) \times (e-1) \) non-singular minor in the matrix \( B_{e-1} \). Hence we can conclude that \( \text{rk}(B_{e-1}) = e - 1 \), i.e. \( \ker(B_{e-1}) \) has dimension equal to 1, and that \( \ker(B_{e-1}) = \text{span}\{k_e^{(e-1)} \} \).

\[\Box\]

We recall the notation used for polynomials of \( \mathbb{F}_q[x] \) (see remark 3.3.1):

**Remark 4.1.4.** If \( v = (v_0, v_1, \ldots, v_{i-1}) \in (\mathbb{F}_q)^i \) then we will call:

\[
v(x) \overset{\text{def}}{=} \sum_{j=0}^{i-1} v_j x^j = v_0 + v_1 x + \cdots + v_{i-1} x^{i-1}
\]
\[ \overline{\sigma}(x) \overset{\text{def}}{=} \sum_{j=0}^{i-1} v_j x^{i-1-j} = v_{i-1} + v_{i-2} x + \cdots + v_0 x^{i-1} \]

\[ P_v(x) \overset{\text{def}}{=} 1 + v_{i-1} x + \cdots + v_0 x^1 \]

**Corollary 4.1.5.** Let \( \overline{k}^{(e)}(x) \) be the polynomial obtained from vector \( k^{(e)} \) as in remark 4.1.4, that is \( \overline{k}^{(e)}(x) = k_0^{(e)} x^e + \cdots + k_{e-1}^{(e)} x + k_e^{(e)} \). If \( e \leq t \), then we have that

\[ \sigma(x) = \frac{1}{k_e^{(e)}} \overline{k}^{(e)}(x) \]

**Proof.** If \( \sigma(x) = \sigma_e x^e + \cdots + \sigma_1 x + 1 \), then we call \( \sigma = (\sigma_e, \ldots, \sigma_1, 1)^T \in (\mathbb{F}_q)^{e+1} \). Since \( e \leq t \), from the linear system (2.2) on page 20 it follows that \( \sigma \in \ker(B_e) \). Thus, by proposition 4.1.3 there exists \( \lambda \in \mathbb{F}_q \) such that \( \sigma = \lambda k^{(e)} \). Using the last component of the former equality we find that \( \lambda = \frac{1}{k_e^{(e)}} \). Thus

\[ \sigma = \frac{1}{k_e^{(e)}} k^{(e)} \]

and the proof is completed. \( \square \)

We recall that if \( e = \text{wt}(e) \) is the number of the error that have occurred in the positions \( p_1, p_2, \ldots, p_e \), then the elements \( X_1, X_2, \ldots, X_e \) are defined as \( X_i = \alpha^{p_i} \) and are the inverses of the roots of the error-locator polynomial \( \sigma(x) \) (see definition 1.5.1).

**Theorem 4.1.6** (Horiguchi’s formula). Consider the vectors \( k^{(e-1)} \) and \( k^{(e)} \) as in definition 4.1.1 and let \( \overline{k}^{(e-1)}(x) \) be the polynomial obtained as in remark 4.1.4. For any \( i \in \{1, 2, \ldots, e\} \), the following equation holds:

\[ E_i = -\frac{k_e^{(e)} \cdot \left(X_i^{-1}\right)^{2(e-1)}}{\sigma'(X_i^{-1}) \overline{k}^{(e-1)}(X_i^{-1})} \]

where \( \sigma'(x) \) is the formal derivative of the error-locator polynomial \( \sigma(x) \).

**Proof.** In order to prove the theorem, we introduce the following elements: for any \( i, j \in \{1, 2, \ldots, e\} \) let

\[ \widehat{E}_j^{(i)} = E_j X_j (1 - X_j X_i^{-1}) \in \mathbb{F}_q, \]

\[ \widehat{X}_j^{(i)} = X_j X_i^{-1} \in \mathbb{F}_q. \]
Moreover for any $l \geq 1$ we consider the sum given by
\[
\hat{S}_l^{(i)} = \sum_{j=1 \atop j \neq i}^{e} \hat{E}_j^{(i)} \left( \hat{X}_j^{(i)} \right)^{l-1}.
\]

Finally let $\hat{A}_{e-1}^{(i)}$ be the $(e-1) \times (e-1)$ matrix defined by
\[
\hat{A}_{e-1}^{(i)} = \begin{pmatrix}
\hat{S}_1^{(i)} & \hat{S}_2^{(i)} & \cdots & \hat{S}_{e-1}^{(i)} \\
\hat{S}_2^{(i)} & \hat{S}_3^{(i)} & \cdots & \hat{S}_e^{(i)} \\
\vdots & \vdots & \ddots & \vdots \\
\hat{S}_{e-1}^{(i)} & \hat{S}_e^{(i)} & \cdots & \hat{S}_{2e-3}^{(i)}
\end{pmatrix}
\]

It is easy to verify that $\hat{A}_{e-1}^{(i)} = (\hat{V}^{(i)})^T \hat{D}^{(i)} \hat{V}^{(i)}$, where $\hat{V}^{(i)}$ and $\hat{D}^{(i)}$ are the square matrices defined by:
\[
\hat{V}^{(i)} = \begin{pmatrix}
1 & \hat{X}_1^{(i)} & \cdots & \left( \hat{X}_1^{(i)} \right)^{e-2} \\
\vdots & \vdots & \ddots & \vdots \\
1 & \hat{X}_{i-1}^{(i)} & \cdots & \left( \hat{X}_{i-1}^{(i)} \right)^{e-2} \\
1 & \hat{X}_{i+1}^{(i)} & \cdots & \left( \hat{X}_{i+1}^{(i)} \right)^{e-2} \\
\vdots & \vdots & \ddots & \vdots \\
1 & \hat{X}_e^{(i)} & \cdots & \left( \hat{X}_e^{(i)} \right)^{e-2}
\end{pmatrix}
\]
\[
\hat{D}^{(i)} = \begin{pmatrix}
\hat{E}_1^{(i)} & \hat{X}_1^{(i)} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
0 & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \hat{E}_{i-1}^{(i)} & \hat{X}_{i-1}^{(i)} & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \hat{E}_{i-1}^{(i)} & \hat{X}_{i-1}^{(i)} & \vdots & \vdots & \vdots \\
0 & \cdots & \cdots & \hat{E}_{i-1}^{(i)} & \hat{X}_{i-1}^{(i)} & \vdots & \vdots & \vdots \\
0 & \cdots & \cdots & \cdots & \hat{E}_{i-1}^{(i)} & \hat{X}_{i-1}^{(i)} & \cdots & \hat{E}_e^{(i)} & \hat{X}_e^{(i)}
\end{pmatrix}
\]

Hence, by Binet’s formula, we have that
\[
\det(\hat{A}_{e-1}^{(i)}) = \prod_{j=1 \atop j \neq i}^{e} \hat{E}_j^{(i)} \hat{X}_j^{(i)} \prod_{l>j \atop l,j \neq i} (\hat{X}_1^{(i)} - \hat{X}_j^{(i)})^2 =
\]
\[
= (X_i^{-1})^{(e-1)(e-2)} \prod_{j=1 \atop j \neq i}^{e} E_j X_j (1 - X_j X_i^{-1}) \prod_{l>j \atop l,j \neq i} (X_l - X_j)^2
\]
On the other hand, we observe that

\[ S_i^{(i)} = \sum_{j=1}^{e} \hat{E}_j^{(i)} \left( \hat{X}_j^{(i)} \right)^{l-1} = \sum_{j=1}^{e} \hat{E}_j^{(i)} \left( \hat{X}_j^{(i)} \right)^{l-1} = \]

\[ = \sum_{j=1}^{e} E_j X_j (1 - X_j X_i^{-1}) (X_j X_i^{-1})^{l-1} = \]

\[ = \sum_{j=1}^{e} E_j X_j^l (X_i^{-1})^{l-1} - \sum_{j=1}^{e} E_j X_j^{l+1} (X_i^{-1})^l = \]

\[ = S_l (X_i^{-1})^{l-1} - S_{l+1} (X_i^{-1})^l \]

Thus \( \hat{A}_{i-1}^{(i)} \) is equal to

\[
\begin{pmatrix}
S_1 - S_2 X_i^{-1} & \cdots & S_{e-1} (X_i^{-1})^{e-2} - S_e (X_i^{-1})^{e-1} \\
S_2 X_i^{-1} - S_3 (X_i^{-1})^2 & \cdots & S_e (X_i^{-1})^{e-1} - S_{e+1} (X_i^{-1})^e \\
\vdots & \ddots & \vdots \\
S_{e-1} (X_i^{-1})^{e-2} - S_e (X_i^{-1})^{e-1} & \cdots & S_{2e-3} (X_i^{-1})^{2e-4} - S_{2e-2} (X_i^{-1})^{2e-3}
\end{pmatrix}
\]

and, by linearity, we have that

\[
\det(\hat{A}_{i-1}^{(i)}) = \det \begin{pmatrix}
S_1 & S_2 X_i^{-1} & \cdots & S_e (X_i^{-1})^{e-1} \\
S_2 X_i^{-1} & S_3 (X_i^{-1})^2 & \cdots & S_{e+1} (X_i^{-1})^e \\
\vdots & \vdots & \ddots & \vdots \\
S_{e-1} (X_i^{-1})^{e-2} & S_e (X_i^{-1})^{e-1} & \cdots & S_{2e-2} (X_i^{-1})^{2e-3}
\end{pmatrix} =
\]

\[
= (X_i^{-1})^{\frac{(e-2)(e-1)}{2}} \det \begin{pmatrix}
S_1 & S_2 X_i^{-1} & \cdots & S_e (X_i^{-1})^{e-1} \\
S_2 & S_3 X_i^{-1} & \cdots & S_{e+1} (X_i^{-1})^e \\
\vdots & \vdots & \ddots & \vdots \\
S_{e-1} & S_e X_i^{-1} & \cdots & S_{2e-2} (X_i^{-1})^{e-1}
\end{pmatrix} =
\]

\[
= (X_i^{-1})^{\frac{(e-2)(e-1)}{2}} \sum_{j=0}^{e-1} (X_i^{-1})^{\frac{(e-1)\epsilon}{2}} j k_{j}^{(e-1)}
\]
4.1. HORIGUCHI’S FORMULA

\[
(X_i^{-1})^{(e-2)(e-1) \cdot \frac{e+1}{2}} \sum_{j=0}^{e-1} (X_i^{-1})^{e-1-j} k_j^{(e-1)} = \\
= (X_i^{-1})^{(e-2)(e-1) \cdot \frac{e+1}{2}} \cdot \frac{e+1}{2} + (e-1) e 2.
\]

Now, comparing the two formulas just obtained for \( \det(A_i) \), we deduce that

\[
\mathcal{E}_i^{-1}(X_i^{-1}) = \prod_{j=1 \atop j \neq i}^{e} E_j X_j (1 - X_j X_i^{-1}) \prod_{l > j \atop l, j \neq i} (X_l - X_j)^2
\]

Finally, using the last equation we can calculate that

\[
\frac{k_e^{(e)}}{\mathcal{E}_i^{-1}(X_i^{-1})} = \frac{\det(A_e)}{\mathcal{E}_i^{-1}(X_i^{-1})} = \prod_{j=1 \atop j \neq i}^{e} E_j X_j \prod_{l > j \atop l, j \neq i} (X_l - X_j)^2 = \\
= \prod_{j=1 \atop j \neq i}^{e} (1 - X_j X_i^{-1}) = E_i X_i X_i^{2(e-1)} \prod_{j=1 \atop j \neq i}^{e} (1 - X_j X_i^{-1}) = \\
= -E_i X_i^{2(e-1)} \sigma'(X_i^{-1})
\]

This concludes the proof.

\[\square\]

**Corollary 4.1.7** (generalized Horiguchi’s formula).

If \( \mathbf{u} = (u_0, u_1, \ldots, u_{e-1}) \in \ker(B_{e-1}) \setminus \{0\} \), then for any \( i \in \{1, 2, \ldots, e\} \) it holds that

\[
E_i = -\frac{(X_i^{-1})^{2(e-1)} \sum_{j=0}^{e-1} S_{e+j} u_j}{\sigma'(X_i^{-1}) \overline{u}(X_i^{-1})}
\]

where \( \overline{u}(x) = u_0 x^{e-1} + u_1 x^{e-2} + \cdots + u_{e-1} \) as in remark 4.1.4.
Proof. By proposition 4.1.3, we know that there exists $\lambda \in \mathbb{F}_q^*$ such that

$$u = \lambda k^{(e-1)}$$

Hence we have that

$$\left( X_i^{-1} \right)^{2(e-1)} \sum_{j=0}^{e-1} S_{e+j} u_j \sigma'(X_i^{-1}) \mathfrak{m}(X_i^{-1}) = \left( X_i^{-1} \right)^{2(e-1)} \sum_{j=0}^{e-1} S_{e+j} \lambda v_j^{(e-1)} \sigma'(X_i^{-1}) \mathfrak{m}(X_i^{-1})$$

$$= \left( X_i^{-1} \right)^{2(e-1)} \sum_{j=0}^{e-1} S_{e+j} v_j^{(e-1)} \sigma'(X_i^{-1}) \mathfrak{m}(X_i^{-1})$$

$$= \left( X_i^{-1} \right)^{2(e-1)} k_e^{(e-1)} \lambda^{(e-1)} (X_i^{-1})$$

$$= \left( X_i^{-1} \right)^{2(e-1)} k_e^{(e-1)} (X_i^{-1})$$

Thus the proof is completed. $\square$

Throughout the next two sections, the formula stated in corollary 4.1.7 will be the key to find new error-evaluation formulas for the $fP$GZ and the BM decoding algorithms.

### 4.2 Application to the fP$GZ$ Decoder

In this section we will show how use corollary 4.1.7 of Horiguchi’s formula in order to find a new method of computing error values suited to the $fP$GZ decoder. We recall the $fP$GZ decoding algorithm 2.2.5 (summarized on page 32): in step $fP$GZ 2 the vectors $w^{(i)} \in (\mathbb{F}_q)^i$ satisfying

$$A_i w^{(i)} = - \begin{pmatrix} S_{i+1} \\ S_{i+2} \\ \vdots \\ S_{2i} \end{pmatrix}$$

are computed for any index $i \leq t$ such that $\det(A_i) \neq 0$. In particular we will assume to have computed

$$\theta = \max \{ j < e \mid \det(A_j) \neq 0 \}$$

and we will show how use the vector $w^{(\theta)}$, which is a byproduct of the computation of $\sigma(x)$, and Horiguchi’s formula (theorem 4.1.6) in order to calculate the error values.
4.2. APPLICATION TO THE FPGZ DECODER

Proposition 4.2.1. Let \( w^{(\theta)} = \left( w_0^{(\theta)}, w_1^{(\theta)}, \ldots, w_{\theta-1}^{(\theta)} \right)^T \in (\mathbb{F}_q)^\theta \) be the vector satisfying

\[
A_\theta w^{(\theta)} = - \begin{pmatrix} S_{\theta+1} \\ S_{\theta+2} \\ \vdots \\ S_{2\theta} \end{pmatrix}
\] (4.2)

and

\[
P_w^{(\theta)}(x) = w_0^{(\theta)} x^\theta + w_1^{(\theta)} x^{\theta-1} + \cdots + w_{\theta-1}^{(\theta)} x + 1 \in \mathbb{F}_q[x]
\]

\[
\varepsilon_{e-\theta} = S_{e} w_0^{(\theta)} + S_{e+1} w_1^{(\theta)} + \cdots + S_{e+\theta-1} w_{\theta-1}^{(\theta)} + S_{e+\theta} \in \mathbb{F}_q
\]

(see remark 4.1.4 and definition 2.2.1). Then for any \( i \in \{1, 2, \ldots, e\} \) we have that

\[
E_i = -\frac{\varepsilon_{e-\theta} \cdot (X_i^{-1})^{e-\theta-1}}{\sigma'(X_i^{-1}) P_w^{(\theta)}(X_i^{-1})}
\] (4.3)

Proof. Recalling definition 4.1.1, system (4.2) implies that

\[
B_\theta \begin{pmatrix} w_0^{(\theta)} \\ S_{\theta+1} \\ \vdots \\ S_{2\theta} \end{pmatrix} = 0
\]

Further the definition of \( \theta \) implies that the coefficients

\[
\varepsilon_j = S_{\theta+j} w_0^{(\theta)} + \cdots + S_{2\theta+j-1} w_{\theta-1}^{(\theta)} + S_{2\theta+j}
\]

are equal to zero for any \( j \in \{1, 2, \ldots, e - \theta - 1\} \). It follows that

\[
\begin{pmatrix} B_\theta \\ S_{\theta+1} \\ \vdots \\ S_{2\theta+1} \\ \vdots \\ S_{e-1} \\ S_{e+\theta-1} \end{pmatrix} \begin{pmatrix} w_0^{(\theta)} \\ \vdots \\ w_{\theta-1}^{(\theta)} \\ 1 \end{pmatrix} = 0
\]

Hence we have that \( \begin{pmatrix} w_0^{(\theta)} \\ 1 \\ 0^{e-\theta-1} \end{pmatrix} \in \ker(B_{e-1}) \setminus \{0\} \) and by corollary 4.1.7 we can conclude that:

\[
E_i = -\frac{S_{e+\theta} + \sum_{j=0}^{\theta-1} S_{e+j} w_j^{(\theta)} (X_i^{-1})^{e-\theta-1}}{\sigma'(X_i^{-1}) (X_i^{-1})^{e-\theta-1} P_w^{(\theta)}(X_i^{-1})}
\]
\[ = -\frac{\varepsilon_{e-\theta} \cdot \left( X_i^{e-1} \right)^{e+\theta-1}}{\sigma'(X_i^{-1}) P_{w(\theta)}(X_i^{-1})}. \]

Proposition 4.2.1 gives an alternative way to compute the error values \( E_i \)'s in the \( fPGZ \) decoder without the need of separately solving the linear system (2.4) in step \( fPGZ.4 \). Indeed (4.3) relates each \( E_i \), independently of the others error values, directly with the syndromes, the roots of \( \sigma(x) \) and the vector \( w(\theta) \). Note that the latter and the coefficient \( \varepsilon_{e-\theta} \) are byproducts of the computation of \( \sigma(x) \), thus no extra computations are needed to know them. Step \( fPGZ.4 \) can be replaced with

\[
\text{FPGZ.4b for } i = 1, 2, \ldots, e \text{ do}
\]

\[
E_i := -\frac{\varepsilon_{e-\theta} \cdot \left( X_i^{-1} \right)^{e+\theta-1}}{\sigma'(X_i^{-1}) P_{w(\theta)}(X_i^{-1})};
\]

\text{endfor}

As already seen in remark 3.1.11 for the \( BM \) decoder, formula (4.3) has the advantages of allowing to the \( fPGZ \) decoder the calculation of error values together with those of positions. Indeed the polynomials \( \sigma'(x) \) and \( P_{w(\theta)}(x) \) can be evaluated in \( \alpha_i \) simultaneously to the error-locator polynomial during Chien’s search of \( fPGZ.3 \) and, when \( \sigma(\alpha_i) = 0 \), the multiplication and the division necessary to calculate the occurring error value can be executed by a multiplier and a divider while the polynomial evaluations go on in the element \( \alpha_{i+1}, \alpha_{i+2}, \ldots \).

With this procedure step \( fPGZ.4b \), after the polynomial evaluations, requires \( e \) division and \( e \) multiplications to compute the error values \( E_i \)'s.

### 4.3 Comparison with the BM Decoder

Now consider the \( BM \) decoding algorithm 3.1.9 as described on page 50. In particular we focus our attention on step 2 of algorithm 3.1.9. In that step the algorithm finds the polynomials \( \sigma(i)(x) \) and \( \omega(i)(x) \) such that

\[ \sigma(i)(x) S(x) \equiv \omega(i)(x) \pmod{x^i} \]

for any \( i = 1, 2, \ldots, d - 1 \) (see definition 1.5.1 and proposition 3.1.4).
4.3. **COMPARISON WITH THE BM DECODER**

In this section, using again Horiguchi’s formula, we will show how the polynomials $σ_i(x)$’s can be used to find the error values in the place of the error-evaluator polynomial. For this purpose we will give the following definition, for which we recall that

$$\Delta_i = \sum_{j=0}^{i} S_{i+1-j} σ_j^{(i)}$$

is the coefficient of $x^i$ in the polynomial $σ_i(x)S(x)$ and that the function $D : \mathbb{N} \rightarrow \mathbb{N}$ is defined in definition 3.1.2 on page 45.

**Definition 4.3.1.** Let $C : \mathbb{N} \rightarrow \mathbb{Z}$ the integer function defined by:

$$C(0) = -1;$$

$$C(i + 1) = \begin{cases} C(i) & \text{if } \Delta_i = 0 \text{ or } 2D(i) \geq i + 1; \\ i & \text{otherwise}; \end{cases}$$

We observe that $C(i)$ represents the iterative step where the most recent change of the function $D(i)$ occurred prior to step $i$. In other words,

$$C(i) = j \iff D(i) = D(i - 1) = \cdots = D(j + 1) > D(j)$$

**Lemma 4.3.2.** Let $c = C(d - 1)$. If $e \leq t$, then we have that

$$c - D(c) = e - 1$$

and moreover $1 + D(c) \leq e$.

**Proof.** By definition 3.1.1 since $C(d - 1) = c$, we have that

$$D(d - 1) = D(d - 2) = \cdots = D(c + 1) = c + 1 - D(c) > D(c)$$

By corollary 3.1.7 we know that $D(d - 1) = e$. Hence $e > D(c)$ and $e = c + 1 - D(c)$.

$\square$
Proposition 4.3.3. If \( e = C(d - 1) \) and \( e \leq t \), then for any \( i \in \{1, 2, \ldots, e\} \) the following equation holds:

\[
E_i = -\frac{(X_i^{-1})^e \Delta_e}{\sigma'(X_i^{-1}) \sigma^{(e)}(X_i^{-1})}
\]  

(4.4)

where \( \Delta_e \) is the coefficient of \( x^e \) in the polynomial \( \sigma^{(e)}(x)S(x) \) as defined in section 3.1.

Proof. The BM decoding algorithm 3.1.9 calculates the polynomials

\[
\sigma^{(e)}(x) = \sigma^{(e)}_{D(c)} x^{D(c)} + \cdots + \sigma^{(e)}_1 x + 1
\]

\[
\omega^{(e)}(x) = \omega^{(e)}_{D(c)-1} x^{D(c)-1} + \cdots + \omega^{(e)}_1 x + 1
\]

such that

\[
\sigma^{(e)}(x)S(x) \equiv \omega^{(e)}(x) \pmod{x^c}
\]

It follows that the coefficients of \( x^{D(c)}, \ldots, x^{c-1} \) in the polynomial \( \sigma^{(e)}(x)S(x) \) are equal to zero. Hence if we call \( \sigma^{(e)} = \left( \sigma^{(e)}_{D(c)}, \ldots, \sigma^{(e)}_1, 1 \right)^T \in (\mathbb{F}_q)^{D(c)+1} \), then we have

\[
\begin{pmatrix}
S_1 & \cdots & S_{D(c)+1} \\
S_2 & \cdots & S_{D(c)+2} \\
\vdots & \ddots & \vdots \\
S_{c-D(c)} & \cdots & S_c
\end{pmatrix} \sigma^{(e)} = 0^{(e-D(c))}
\]

By lemma 4.3.2 we obtain that

\[
\begin{pmatrix}
S_1 & \cdots & S_c \\
S_2 & \cdots & S_{e+1} \\
\vdots & \ddots & \vdots \\
S_{e-1} & \cdots & S_{2e-2}
\end{pmatrix} \begin{pmatrix}
\sigma^{(e)} \\
0^{(e-D(c)-1)}
\end{pmatrix} = B_{e-1} \begin{pmatrix}
\sigma^{(e)} \\
0^{(e-D(c)-1)}
\end{pmatrix} = 0^{(e-1)}
\]

Thus

\[
\begin{pmatrix}
\sigma^{(e)} \\
0^{(e-D(c)-1)}
\end{pmatrix} \in \ker(B_{e-1}) \setminus \{0^{(e)}\}
\]

Now we use corollary 4.1.7 and we have that

\[
E_i = -\frac{\left( X_i^{-1} \right)^{2(e-1)} \left[ S_{e+D(c)} + \sum_{j=1}^{D(c)} S_{e+D(c)-j} \sigma_j^{(c)} \right]}{\sigma'(X_i^{-1}) \left( X_i^{-1} \right)^{e-1-D(c)} \sigma^{(c)}(X_i^{-1})} = \uparrow_{e+D(c)=c+1}
\]
4.3. COMPARISON WITH THE BM DECODER

\[ - \frac{(X_i^{-1})^c \Delta_c}{\sigma'(X_i^{-1})\sigma^{(c)}(X_i^{-1})} \]

This concludes the proof. \(\square\)

Since the polynomial \(\sigma^{(c)}(x)\) and the coefficient \(\Delta_c\) are obtained as byproducts of the computation for \(\sigma(x)\), the formula stated in \(\text{4.3.3}\) permits a more efficient decoding algorithm than Forney’s formula \((\text{1.2})\) saving the cost of computing the error-evaluator polynomial \(\omega(x)\).

Remark 4.3.4. We have already seen in proposition \(\text{3.1.13}\) that the computation of the error-evaluator polynomial \(\omega(x)\) in step \(\text{BM 4}\) can be avoided. Indeed, we proved that for any \(i \in \{1, 2, \ldots, e\}\)

\[ E_i = -\frac{(X_i^{-1})^{d-2}}{\sigma'(X_i^{-1})\tau^{(d-1)}(X_i^{-1})} \]

where the polynomial \(\tau^{(d-1)}(x)\) is the last of the auxiliary polynomials calculate in step \(\text{BM 2}\). It is easy to show that the latter and \((\text{4.4})\) are the same formula. Indeed

\[ C(d - 1) = c \Rightarrow D(d - 1) = D(d - 2) = \cdots = D(c + 1) > D(c) \]

\[ \Rightarrow \tau^{(d-1)}(x) = x\tau^{(d-2)}(x) = \cdots = x^{d-c-2}\tau^{(c+1)}(x) = x^{d-c-2}\sigma^{(c)}(x) \]

Hence

\[ E_i = -\frac{(X_i^{-1})^{d-2}}{\sigma'(X_i^{-1})\tau^{(d-1)}(X_i^{-1})} = -\frac{(X_i^{-1})^c \Delta_c}{\sigma'(X_i^{-1})\sigma^{(c)}(X_i^{-1})} \]

As seen in section \(\text{3.1}\) the polynomials in \((\text{4.4})\) and in \((\text{3.8})\) are evaluated in the field element using Chien’s search circuit (figure \(\text{1.4}\)).

We observe that

\[ e \leq t \leq \frac{d - 1}{2} \quad \Rightarrow \quad e \leq d - 1 - e \]

and moreover

\[ \deg (\sigma^{(c)}(x)) \leq D(c) \leq e - 1 \]

\[ \deg (\tau^{(d-1)}(x)) \leq d - 1 - D(d - 1) = d - 1 - e \]

Thus \(\deg (\sigma^{(c)}(x)) < \deg (\tau^{(d-1)}(x))\) and so \((\text{4.4})\) can be a better choice than \((\text{3.8})\) because the evaluation of \(\tau^{(d-1)}(x)\) in the field elements needs more circuit components than the evaluation of \(\sigma^{(c)}(x)\).
Chapter 5
Parallel Implementation

The continuing improvements of microelectronics technology leads to the availability of integrated circuits (microchips) with high-speed parallel architectures. Indeed, as at the state of the art one microchip can contain a large number of circuits and it is possible that independent tasks are accomplished simultaneously by different circuits of the same microchip. A very simple example is a microchip with \( m \) adders: with the computational time complexity of only one addition, this microchip can add two vectors of \( m \) components because the \( i \)th adder of the microchip adds the \( i \)th components of the two vectors simultaneously for \( i = 1, 2, \ldots, m \). The same idea is used to implement the multiplication of a vector of \( m \) components by a scalar element with \( m \) multipliers and with a time complexity equal to only one multiplication. Another example regards the sum of \( m \) elements: observing that

\[
\sum_{i=1}^{m} a_i = \left( \sum_{i=1}^{[m/2]} a_i \right) + \left( \sum_{i=[m/2]+1}^{m} a_i \right)
\]

and repeating this splitting, the sum of \( m \) elements can be computed with \( \lceil \log_2 m \rceil \) additions using at most \( m \) adders forming a tree of depth \( \lceil \log_2 m \rceil \) (see figure 5.1).

We can note immediately that the complexity of parallel algorithms is estimated in terms of the time (as the classical sequential algorithms) and of the space (number of circuit elements) that they take.

We also observe that the improvement brought by the parallel implementation also depends on the inner parallelism of the instructions that have to be implemented: adding two vectors (or two polynomials) and multiplying a vector (or a polynomial) by a scalar element are operations that have a high level of inner parallelism. Instead the same is not true for the sum of
$m$ elements. Indeed with $m$ adders the complexity of this operation decrease only from $m$ to $\lceil \log_2 m \rceil$ additions.

![Adder Tree](image)

**Figure 5.1:** Adder tree for summing four elements

In more complicated cases, it is necessary to have a fundamental understanding of the algorithm structure to realize the full potential of parallel computing. Indeed often it may happen that the most efficient algorithms for classic sequential implementation are not necessarily the most efficient for parallel architectures. For this, the task of this chapter is revising the main steps of the \textit{PGZ} and the \textit{BM} decoding algorithms studied in chapters 2 and 3 in order to understand when a parallel implementation of these steps can bring some significant advantages decreasing the computational cost of the decoding. Moreover, when it is necessary, we will modify some steps of the decoding algorithms mentioned above in order to achieve the maximum benefits from their parallelization.

### 5.1 Parallel Implementation of \textit{PGZ} Decoder

As seen in section 2.1 step \textit{PGZ} 2 of the \textit{PGZ} decoding algorithm 2.1.4 calculates first $e$ the number of errors that occur, computing for $i = t, t-1, \ldots$
5.1. PARALLEL IMPLEMENTATION OF PGZ DECODER

the determinants of the matrices

\[ A_i = \begin{pmatrix} S_1 & S_2 & \cdots & S_i \\ S_2 & S_3 & \cdots & S_{i+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_i & S_{i+1} & \cdots & S_{2t-1} \end{pmatrix} \]

and setting \( e := \max\{i \leq t \mid \det(A_i) \neq 0\} \), and after calculates the error-locator polynomial \( \sigma(x) \) solving the linear system

\[ A_e \begin{pmatrix} \sigma_e \\ \sigma_{e-1} \\ \vdots \\ \sigma_1 \end{pmatrix} = - \begin{pmatrix} S_{e+1} \\ S_{e+2} \\ \vdots \\ S_{2e} \end{pmatrix} \]

In this section we will give a parallel implementation of this procedure, exploiting the Laplace expansion for determinants for the calculation of the determinants of the matrices \( A_i \)'s, which are nothing but the leading principal minors of the syndrome matrix

\[ A = \begin{pmatrix} S_1 & S_2 & \cdots & S_{t+1} \\ S_2 & S_3 & \cdots & S_{t+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_{d-1-t} & S_{d-t} & \cdots & S_{d-1} \end{pmatrix} \]

and exploiting Horiguchi’s formula (theorem 4.1.6) for finding the error-locator polynomial using the minors already computed during the calculation of \( e \). More generally, we will use the properties of the minors of the syndrome matrix, which have been proved in section 4.1, to also express the error values in terms of minors already computed, modifying so the implementation of the step PGZ [4]

For this, we introduce the following tools:

**Definition 5.1.1.** Let \( i \) be an index less or equal to \( t \). We consider the set \( \mathcal{N}_i \) defined by

\[ \mathcal{N}_i = \{ j = (j_1, j_2, \ldots, j_i) \in \mathbb{N}^i \mid 1 \leq j_1 < j_2 < \cdots < j_i \leq t \} \]

For any \( j = (j_1, j_2, \ldots, j_i) \in \mathcal{N}_i \) we define \( D_j^{(i)} \) as the determinant of the \( i \times i \) minor of \( A \) given by the first \( i \) rows and the columns \( j_1, j_2, \ldots, j_i \). That is

\[ D_j^{(i)} = \det\begin{pmatrix} S_{j_1} & S_{j_2} & \cdots & S_{j_i} \\ S_{j_1+1} & S_{j_2+1} & \cdots & S_{j_{i+1}} \\ \vdots & \vdots & \ddots & \vdots \\ S_{j_{1+i-1}} & S_{j_{2+i-1}} & \cdots & S_{j_{i+i-1}} \end{pmatrix} \]
We now observe that using the Laplace expansion for determinants we have that:

\[ \det^{(i)}_{(j_1, j_2, \ldots, j_i)} = \sum_{l=1}^{i} S_{j_{l+i-1}}^{i-l} \det^{(i-1)}_{(j_1, j_2, \ldots, j_{l+1}, \ldots, j_i)} \quad (5.1) \]

for any \((j_1, j_2, \ldots, j_i) \in \mathcal{N}_i\) and for any \(i \in \{1, 2, \ldots, t\}\).

**Remark 5.1.2.** We recall that the elements \(k^{(i)}_j\)'s are defined in section 4.1 as

\[ k^{(i)}_j = (-1)^{i+j} \begin{pmatrix} S_1 & S_2 & \cdots & S_{i+1} \\ S_2 & S_3 & \cdots & S_{i+2} \\ \vdots & \vdots & \ddots & \vdots \\ S_i & S_{i+1} & \cdots & S_{2i} \end{pmatrix} \]

for any \(i \geq 1, j \in \{0, 1, \ldots, i\}\) and we note that they are a subset of the elements \(D_j^{(i)}\)'s. More precisely for any \(i \geq 1, j \in \{0, 1, \ldots, i\}\) we have that:

\[ k^{(i)}_j = (-1)^{i+j} D_j^{(i)} \quad (5.2) \]

The next proposition shows that we can express the number of errors that occur, the error-locator polynomial and the error values in terms of the minors \(D_j^{(i)}\)'s.

**Proposition 5.1.3.** If \(e \leq t\), then

1. \(e = \min \left\{ i \in \{1, 2, \ldots, t\} \mid D_j^{(i+1)} = 0 \quad \forall j \in \mathcal{N}_{i+1} \right\} \);

2. \(\sigma(x) = \frac{1}{D^{(e)}_{(1,2,\ldots,e)}} \left( \sum_{j=0}^{e} (-1)^{e+j} D^{(e)}_{(1,\ldots,j,j+2,\ldots,e+1)} x^{e-j} \right) \);

3. For any \(i \in \{1, 2, \ldots, e\}\) we have that

\[ E_i = -\frac{D^{(e)}_{(1,2,\ldots,e)} \left( X_i^{-1} \right)^{2(e-1)}}{\sigma'(X_i^{-1}) \tilde{\omega}(X_i^{-1})} \]

where \(\tilde{\omega}(x) = \sum_{j=0}^{e-1} (-1)^{e-1+j} D^{(e-1)}_{(1,\ldots,j,j+2,\ldots,e)} x^{e-1-j} \).

**Proof.** To prove (1) we define

\[ m = \min \left\{ i \in \{1, 2, \ldots, t\} \mid D_j^{(i+1)} = 0 \quad \forall j \in \mathcal{N}_{i+1} \right\} \]
and we will show that \( e = m \). Recalling proposition 2.1.2 we have that
\[
e \leq t \Rightarrow e = \text{rk}(A) \Rightarrow (D_j^{e+1}) = 0 \quad \forall j \in N_{e+1} \Rightarrow m \leq e
\]

Further we note that by (5.1), if \( m < e \), then \( D_j^{e}(1, 2, \ldots, e) = \det(A_e) = 0 \) and this is absurd (see proposition 2.1.2). Thus \( e = m \) and (1) is proved. Using (5.2), (2) follows immediately by corollary 4.1.5 and (3) follows from Horiguchi’s formula (theorem 4.1.6).

Remark 5.1.4. In order to save divisions, we may consider the polynomial
\[
\hat{\sigma}(x) \overset{\text{def}}{=} D_{(1, 2, \ldots, e)}^{(e)} \cdot \sigma(x)
\]
It is clear that \( \hat{\sigma}(x) \) has the same roots as the error-locator polynomial \( \sigma(x) \) and that it holds
\[
E_i = -\frac{(D_{(1, 2, \ldots, e)}^{(e)})^2 (X_i^{-1})^{2(e-1)}}{\hat{\sigma}'(X_i^{-1})\hat{\omega}(X_i^{-1})}
\]
for any \( i \in \{1, 2, \ldots, e\} \)

Finally we deduce from (5.1) that if we already know the minor \( D_j^{(i-1)} \) for every \( j \in N_{i-1} \), then any minor \( D_j^{(i)} \) can be calculated with only \( i \) independent multiplications and one sum of \( i \) elements. Furthermore we observe that the computation of one \( D_j^{(i)} \) is independent of the computation of the others minors \( i \times i \). Hence we may think of an iterative parallel algorithm in which the \( i \)th step computes at the same time all the determinants \( D_j^{(i)} \) using (5.1). We write in details this algorithm, which we will call parallel Peterson-Gorenstein-Zierler (pPGZ) decoding algorithm:

Algorithm 5.1.5 (pPGZ decoding algorithm for RS\((n, d, \alpha)\)).

Input: the received word \( r(x) \);

Output: the codeword \( c(x) \);

Begin

\[pPGZ.1\] (syndrome computation)
for \( i = 1, 2, \ldots, d - 1 \) do
\[
S_i := r(\alpha^i);
\]
endfor

\[pPGZ.2\] (error-locator polynomial computation)
a) (initialization)
\[
\begin{align*}
t &:= \left\lfloor \frac{d-1}{2} \right\rfloor; \\
i &:= 1; \\
\text{while } (i \leq t) & \text{ repeat} \\
D^{(1)}_i &:= S_i; \\
i &:= i + 1;
\end{align*}
\]
endwhile

b) (iterative parallel procedure)
\[
\begin{align*}
i &:= 2; \\
\text{while } (i \leq t) & \text{ do in parallel} \\
D^{(i)}_{(j_1, \ldots, j_i)} &:= \sum_{l=1}^{i} S_{j_l+i-1}(-1)^{i+l}D^{(i-1)}_{(j_1, \ldots, j_l, j_{l+1}, \ldots, j_i)}; \\
\text{if } (\text{all } D^{(i)}_{(j_1, \ldots, j_i)}'s \text{ are zero}) & \text{ then} \\
\text{if } (D^{(i-1)}_{(j_1, \ldots, j_{i-1})} \neq 0) & \text{ then} \\
\quad e &:= i - 1; \\
\quad \text{go to step c;} \\
\quad \text{else declare a malfunction;} \\
\text{else } i &:= i + 1;
\end{align*}
\]
endwhile
\[
e := t;
\]

c) \(\hat{\sigma}(x) := \sum_{j=0}^{e} (-1)^{e+j}D^{(e)}_{(1, \ldots, j, j+2, \ldots, e+1)} x^{e-j};\)
\[
\hat{\omega}(x) := \sum_{j=0}^{e-1} (-1)^{e-1+j}D^{(e-1)}_{(1, \ldots, j, j+2, \ldots, e)} x^{e-1-j};
\]

**pPGZ.3** (finding errors positions)

calculate the error positions \(p_1, p_2, \ldots, p_e\) and the elements \(X_1^{-1}, X_2^{-1}, \ldots, X_e^{-1}\) using Chien’s search;

**pPGZ.4** (finding error values)

for \(i = 1, 2, \ldots, e\) do in parallel
5.1. PARALLEL IMPLEMENTATION OF PGZ DECODER

\[ E_i := - \frac{\left( D_{(1,2,...,e)}^{(e)} \right)^2 (X_i^{-1})^{2(e-1)}}{\sigma'(X_i^{-1}) \tilde{\omega}(X_i^{-1})} \]

endfor

Return \( c(x) := r(x) - \sum_{i=1}^{e} E_i x^p_i; \)
End

The correctness of the \( pPGZ \) algorithm is assured by proposition 5.1.3. As regards the computational cost, we observe that since

\[ \#\{D^{(i)}_j \mid j \in \mathcal{N}_i \} = \#\mathcal{N}_i = \binom{t}{i} \]

in order that the \( i \)th step of \( pPGZ.2 \) can calculate the minors \( D^{(i)}_j \)'s simultaneously for every \( j \in \mathcal{N}_i \), at most \( \binom{t}{i} \) circuit (with shared memory) are necessary. Each of these is formed from \( i \) multipliers and \( i \) adders and computes one of the minor \( D^{(i)}_j \) with:

- 1 multiplication (the multipliers compute simultaneously the products of the type \( S_{j_{i+1}} (-1)^{i+i} \cdot D^{(i-1)}_{j_1,...,j_{i+1},...,j_i} \);

- \( \lceil \log_2 i \rceil \) additions (the products computed are added in an adder tree of maximum depth \( \lceil \log_2 t \rceil \));

As \( i \) goes from 1 to \( e \), the computational cost of \( pPGZ.2 \) is upper bounded by \( e \) multiplications and \( e \lceil \log_2 e \rceil \) additions using at most

\[ \max \left\{ \binom{t}{i} \mid 1 \leq i \leq e \right\} = \binom{t}{\lfloor t/2 \rfloor} \]

circuits. That is

\( t \binom{t}{\lfloor t/2 \rfloor} \) multipliers \( t \binom{t}{\lfloor t/2 \rfloor} \) adders

In order that this number is not too big, we have to suppose to be correcting Reed-Solomon codes with a small correction capability. For example if \( t \leq 8 \) we need only some hundreds of multipliers and adders that nowadays can be held in only one microchip (see table 5.1).
As already seen for step \( f \) (see section 4.2), for the implementation of \( pPGZ.3 \) and \( pPGZ.4 \) we suppose that there are 4 circuits for Chien’s search that compute all the polynomial evaluations needed to the error positions and error values computation with a negligible computational time complexity compared to the cost of other operations involved. After the polynomial evaluations, first the multiplications \( \hat{\sigma}'(X^{-1}_i) \cdot \hat{\omega}(X^{-1}_i) \) are executed at the same time by \( e \) multipliers and then the divisions needed to calculate \( E_i \)'s are simultaneously handled by \( e \) dividers. The complexity of the main step of the \( pPGZ \) decoding algorithm are summarized in the following table:

| \( t \) | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|--------|-----|-----|-----|-----|-----|-----|-----|
| \( t( \frac{t}{\lfloor t/2 \rfloor} ) \) | 9   | 24  | 50  | 120 | 245 | 560 | 1134 |

Table 5.1: number of circuit elements required by \( pPGZ.2 \)

| \( pPGZ.2 \) | \( e \) multiplications | \( e \lfloor \log_2 e \rfloor \) additions | \( t\left( \frac{t}{\lfloor t/2 \rfloor} \right) \) multipliers | \( t\left( \frac{t}{\lfloor t/2 \rfloor} \right) \) adders |
| \( pPGZ.4 \) | 1 division | 1 multiplication | \( t \) dividers | 2t multipliers |

Table 5.2: parallel complexity of \( pPGZ.2 \) and \( pPGZ.4 \)

5.2 Parallel Implementation of the \( BM \) decoder

We recall the implementation given in section 3.1 of the \( BM \) decoding algorithm [3.1.9] in which the error-locator polynomial is calculated computing
5.2. PARALLEL IMPLEMENTATION OF THE BM DECODER

recursively the polynomials $\sigma^{(1)}(x), \sigma^{(2)}(x), \ldots, \sigma^{(d-1)}(x) = \sigma(x)$ (see definition 3.1.1 and corollary 3.1.8) by the instructions of the step $BM_2$ recalled below.

$BM_2$ (error-locator computation)

\[
\begin{align*}
\sigma^{(0)} & := 1; \\
\tau^{(0)} & := 1; \\
D(0) & := 0; \\
\text{for } i = 0, 1, \ldots, d - 2 & \text{ do} \\
\Delta_i & := \sum_{j=0}^{D(i)} S_{i+1-j} \sigma_j^{(i)}; \\
\sigma^{(i+1)}(x) & := \sigma^{(i)}(x) - \Delta_i x \tau^{(i)}(x); \\
\text{if } (\Delta_i = 0 \text{ or } 2D(i) \geq i + 1) & \text{ then} \\
D(i + 1) & := D(i); \\
\tau^{(i+1)}(x) & := x \tau^{(i)}(x); \\
\text{else} & \\
D(i + 1) & := i + 1 - D(i); \\
\tau^{(i+1)}(x) & := \frac{\sigma^{(i)}(x)}{\Delta_i}; \\
\text{endfor} \\
e & := D(d - 1); \\
\sigma(x) & := \sigma^{(d-1)}(x);
\end{align*}
\]

We can note that the polynomial updates (*) and (**) do not constitute obstacles to parallel computation. Indeed as seen we can add two polynomials or multiply a polynomial by a field element with the cost of one multiplication or one addition (with a linear number of multipliers and adders). Thus the instructions (*) and (**) can be implemented at same time, after the discrepancy $\Delta_i$ has been computed, with the overall cost of one inversion, one multiplication an one addition. It is the computation of discrepancy the $\Delta_i$ that constitutes the parallel computing bottleneck. This is because $\Delta_i$ is computed via a sum of $i$ elements (which requires one multiplication but $\lceil \log_2 i \rceil$ additions implemented in a circuit with $i$ multipliers and $i$ adders) and moreover because it must be computed before the polynomial update (*).

In order to eliminate this bottleneck and reach a greater parallelism, the $BM$ decoding algorithm 3.1.9 is modified following an idea of Dilip V. Sarwate and Naresh R. Shanbhag in [SS01].

For this we introduce the following polynomials:
Definition 5.2.1. Let $\sigma^{(i)}(x)$ and $\tau^{(i)}$ the polynomials given in definitions 3.1.1 and 3.1.3. If $S(x) = S_1 + S_2 x + \cdots + s_{d-1} x^{d-2}$ is the syndrome polynomial (definition 1.5.3), then for any $i \in \{0, 1, \ldots, d-1\}$ we define the polynomials

$$\Delta^{(i)}(x) = \sigma^{(i)}(x)S(x)$$
$$\Theta^{(i)}(x) = \tau^{(i)}(x)S(x)$$

Moreover $\Delta^{(i)}_j$ and $\Theta^{(i)}_j$ will denote respectively the coefficient of the term $x^j$ in $\Delta^{(i)}(x)$ and in $\Theta^{(i)}(x)$.

We recall that the $i$th discrepancy $\Delta_i$ is defined as the coefficient of the term $x^i$ in the polynomial $\sigma^{(i)}(x)S(x)$, hence it is obvious that

$$\Delta^{(i)}_i = \Delta_i$$

for every $i = 0, 1, \ldots, d - 2$.

The following proposition states the recursive properties of the polynomial families $\{\Delta^{(i)}(x)\}_{i=0}^{d-1}$ and $\{\Theta^{(i)}(x)\}_{i=0}^{d-1}$;

**Proposition 5.2.2.** With the notations of definition 5.2.1 it holds that:

1. $\Delta^{(0)}(x) = S(x)$ and $\Theta^{(0)}(x) = S(x)$;

2. For any $i \in \{0, 1, \ldots, d - 2\}$

$$\Delta^{(i+1)}(x) = \Delta^{(i)}(x) - \Delta^{(i)}_i x \Theta^{(i)}(x) \quad (5.3)$$

3. If $D : \mathbb{N} \to \mathbb{N}$ is the function stated in definition 3.1.2 as $D(0) = 0$ and

$$D(i + 1) = \begin{cases} D(i) & \text{if } \Delta_i^{(i)} = 0 \text{ or } 2D(i) \geq i + 1 \\ i + 1 - D(i) & \text{if } \Delta_i^{(i)} \neq 0 \text{ and } 2D(i) < i + 1 \end{cases}$$

then

$$\Theta^{(i+1)}(x) = \begin{cases} x \Theta^{(i)}(x) & \text{if } \Delta_i^{(i)} = 0 \text{ or } 2D(i) \geq i + 1 \\ \frac{\Delta_i^{(i)}(x)}{\Delta_i^{(0)}} & \text{if } \Delta_i^{(i)} \neq 0 \text{ and } 2D(i) < i + 1 \end{cases} \quad (5.4)$$

for any $i \in \{0, 1, \ldots, d - 2\}$

**Proof.** 1. Trivial using that $\sigma^{(0)}(x) = \tau^{(0)}(x) = 1$ by definition;
2. From the definition 3.1.1 it follows that
\[
\Delta^{(i+1)}(x) = \sigma^{(i+1)}(x)S(x) = \left(\sigma^{(i)}(x) - \Delta^{(i)} xT^{(i)}(x)\right) S(x) = \\
= \sigma^{(i)}(x)S(x) - \Delta^{(i)} xT^{(i)}(x)S(x) = \\
= \Delta^{(i)}(x) - \Delta^{(i)} x\Theta^{(i)}(x)
\]

3. From definition 3.1.3 it follows that if \(\Delta^{(i)} = 0\) or \(2D(i) \geq i + 1\) then
\[
\Theta^{(i+1)}(x) = \tau^{(i+1)}(x)S(x) = x\tau^{(i)}(x)S(x) = x\Theta^{(i)}(x)
\]
Whereas if \(\Delta^{(i)} \neq 0\) and \(2D(i) < i + 1\) then
\[
\Theta^{(i+1)}(x) = \tau^{(i+1)}(x)S(x) = \frac{\sigma^{(i)}(x)}{\Delta^{(i)}} S(x) = \frac{\Delta^{(i)}(x)}{\Delta^{(i)}}
\]

Proposition 5.2.2 allows us to calculate the discrepancies \(\Delta_1, \Delta_2, \ldots, \Delta_{d-2}\) avoiding the sum of several elements as in (*). Indeed it is sufficient to implement the update of the polynomial \(\Delta^{(i)}(x)\) as described in the previous proposition in order to compute the \((i + 1)\)th discrepancy at the same time that the polynomial \(\sigma^{(i)}(x)\) is been computed.

Moreover we note that for any \(j < i\), the coefficients \(\Delta^{(i)}_j\) and \(\Theta^{(i)}_j\) cannot affect the value of any later discrepancies \(\Delta_{i+k}\). Consequently, we need not store the coefficients \(\Delta^{(i)}_j\) and \(\Theta^{(i)}_j\) for \(j < i\) and we can save memory space and circuit elements, defining
\[
\hat{\Delta}^{(i)}(x) \overset{\text{def}}{=} \sum_{j=0}^{d-2} \Delta^{(i)}_{i+j} x^j 
\]
\[
\hat{\Theta}^{(i)}(x) \overset{\text{def}}{=} \sum_{j=0}^{d-2} \Theta^{(i)}_{i+j} x^j 
\]
From this it holds that:

**Proposition 5.2.3.** If \(\hat{\Delta}^{(i)}_j\) is the coefficient of \(x^j\) in the polynomial \(\hat{\Delta}^{(i)}(x)\), then

1) \(\hat{\Delta}^{(0)} = S(x)\) and \(\hat{\Theta}^{(0)}(x) = S(x)\);
2) For any \(i \in \{0, 1, \ldots, d - 2\}\),
\[
\hat{\Delta}^{(i+1)}(x) = \left(\sum_{j=1}^{d-2} \hat{\Delta}^{(i)}_j x^{j-1}\right) - \hat{\Delta}^{(i)}_0 \hat{\Theta}^{(i)}(x)
\]
(3) For any \( i \in \{0, 1, \ldots, d-2\} \),

\[
\hat{\Theta}^{(i+1)}(x) = \begin{cases} 
\hat{\Theta}^{(i)}(x) & \text{if } \Delta_0^{(i)} = 0 \text{ or } 2D(i) \geq i + 1 \\
\frac{1}{\Delta_0^{(i)}} \left( \sum_{j=1}^{d-2} \hat{\Delta}^{(i)}_j x^{j-1} \right) & \text{if } \Delta_0^{(i)} \neq 0 \text{ and } 2D(i) < i + 1
\end{cases}
\]

Proof. 1. By (5.5) and (5.6) it follows that \( \hat{\Delta}^{(0)}(x) = \Delta^{(0)}(x) = S(x) \) and \( \hat{\Theta}^{(0)}(x) = \hat{\Theta}^{(0)}(x) = S(x) \).

2. By (5.5) and (5.3) it follows that

\[
\hat{\Delta}^{(i+1)}(x) = \sum_{j=0}^{d-2} \Delta^{(i+1)}_{i+1+j} x^j = \\
= \sum_{j=0}^{d-2} \left( \Delta^{(i)}_{i+1+j} - \Delta^{(i)}_i \Theta^{(i)}_{i+j} \right) x^j \\
= \sum_{j=1}^{d-2} \Delta^{(i)}_{i+j} x^{j-1} - \Delta^{(i)} \sum_{j=0}^{d-2} \Theta^{(i)}_{i+j} x^j = \\
= \left( \sum_{j=1}^{d-2} \hat{\Delta}^{(i)}_j x^{j-1} \right) - \Delta_0^{(i)} \hat{\Theta}^{(i)}(x)
\]

where we used that \( \Delta^{(i)}_{i+d-1} = 0 \) because

\[
\deg(\Delta^{(i)}(x)) = \deg(\sigma^{(i)}(x)) + \deg(S(x)) \leq i + d - 2
\]

3. (3) follows by (5.6) and (5.4) similarly to what already done for (2).

Before showing in detail the parallel implementation of the BM decoding algorithm due to the reformulated discrepancy computation as seen in proposition 5.2.3, we present in the following proposition the relation between the error values and the coefficients of the polynomial \( \hat{\Delta}^{(d-1)}(x) \). We will use it to reformulate the step BM [1].

**Proposition 5.2.4.** Let \( \hat{\Delta}^{(d-1)}_j \)'s be the coefficients of the polynomial \( \hat{\Delta}^{(d-1)}(x) \) and let \( e = \text{wt}(e) \) the number of errors which occurred. If \( e \leq t \) then for any \( i \in \{1, 2, \ldots, e\} \) it holds that

\[
E_i = \frac{\hat{\Delta}^{(d-1)}(X_i^{-1}) \cdot (X_i^{-1})^{d-1}}{\sigma'(X_i^{-1})} \tag{5.7}
\]
5.2. PARALLEL IMPLEMENTATION OF THE BM DECODER

Proof. By corollary 3.1.8 we know that if \( e \leq t \) then \( \sigma^{(d-1)}(x) = \sigma(x) \). Thus we have that:

\[
\Delta^{(d-1)}(x) = \sigma^{(d-1)}(x)S(x) = \sigma(x)S(x)
\]

and by the key equation (1.4) it follows that

\[
\Delta^{(d-1)}(x) \equiv \omega(x) \pmod{x^{d-1}}
\]

Now we observe that \( \deg(\Delta^{(d-1)}(x)) = \deg(\sigma(x)) + \deg(S(x)) = e + d - 2 \), while \( \deg(\omega(x)) \leq e - 1 \). Thus we conclude that \( \Delta^{(d-1)}_{e+d-2+j} = 0 \) for any \( j \leq 1 \) and that the previous congruence implies:

\[
\Delta^{(d-1)}(x) = \omega(x) + \left( \Delta^{(d-1)}_{d-1} + \Delta^{(d-1)}_{d} x + \cdots + \Delta^{(d-1)}_{e+d-2} x^{e-1} \right) x^{d-1} =
\]

\[
= \omega(x) + \hat{\Delta}^{(d-1)}(x) x^{d-1}
\]

Hence for any \( i \in \{1, 2, \ldots, e\} \) we have that

\[
0 = \Delta^{(d-1)}(X_i^{-1}) = \omega(X_i^{-1}) + \hat{\Delta}^{(d-1)}(X_i^{-1}) \cdot (X_i^{-1})^{d-1}
\]

and substituting \( \omega(X_i^{-1}) = -\hat{\Delta}^{(d-1)}(X_i^{-1}) \cdot (X_i^{-1})^{d-1} \) in Forney’s formula (1.2) we can conclude the proof.

We state now the parallel Berlekamp-Massey (pBM) decoding algorithm:

Algorithm 5.2.5 (pBM decoding algorithm for RS(\( n, d, \alpha \))).

Input: the received word \( r(x) \);

Output: the codeword \( c(x) \);

Begin

\[\text{pBM.1 (syndrome computation)}\]

\[
\text{for } i = 1, 2, \ldots, d - 1 \text{ do in parallel}
\]

\[
S_i := r(\alpha^i);
\]

endfor

\[\text{pBM.2 (error-locator polynomial computation)}\]

\[
\sigma^{(0)} := 1;
\]

\[
\tau^{(0)} := 1;
\]

\[
\hat{\Delta}^{(0)} := S(x);
\]

\[
\hat{\Theta}^{(0)} := S(x);
\]

\[
D(0) := 0;
\]
for \( i = 0, 1, \ldots, d - 2 \) do in parallel
\[
\sigma^{(i)}(x) := \sigma^{(i)}(x) - \hat{\Delta}^{(i)}_0 x T^{(i)}(x); \\
\hat{\Delta}^{(i+1)}(x) := \left( \sum_{j=1}^{d-2} \hat{\Delta}^{(i)}_j x^{j-1} \right) - \hat{\Delta}^{(i)}_0 \hat{\Theta}^{(i)}(x); \\
\text{if } (\hat{\Delta}^{(i)}_0 = 0 \text{ or } 2D(i) \geq i + 1) \text{ then} \\
D(i + 1) := D(i); \\
\tau^{(i+1)}(x) := x T^{(i)}(x); \\
\hat{\Theta}^{(i+1)}(x) := \hat{\Theta}^{(i)}(x); \\
\text{else} \\
D(i + 1) := i + 1 - D(i); \\
\tau^{(i+1)}(x) := \frac{\sigma^{(i)}(x)}{\Delta^{(i)}_0}; \\
\hat{\Theta}^{(i+1)}(x) := \frac{1}{\Delta^{(i)}_0} \left( \sum_{j=1}^{d-2} \hat{\Delta}^{(i)}_j x^{j-1} \right); \\
\text{endif}
\]
\( e := D(d - 1); \)
\( \sigma(x) := \sigma^{(d-1)}(x); \)

**pBM.3** *(finding error positions)*
calculate the error positions \( p_1, p_2, \ldots, p_e \) and the elements \( X_1^{-1}, X_2^{-1}, \ldots, X_e^{-1} \) using Chien’s search;

**pBM.4** *(finding the error values)*
for \( i = 1, 2, \ldots, e \) do in parallel
\[
E_i := \frac{\hat{\Delta}^{(d-1)}(X_i^{-1}) \cdot (X_i^{-1})^{d-1}}{\sigma'(X_i^{-1})};
\]
endfor

Return \( c(x) := r(x) - \sum_{i=1}^{e} E_i x^{p_i}; \)

End

The correctness of the \( pBM \) decoding algorithm follows by propositions 5.2.3 and 5.2.4. The instructions of step \( pBM.2 \) can be implemented in a
5.2. PARALLEL IMPLEMENTATION OF THE BM DECODER

systolic architecture composed of an array of \(d + t\) circuit of the same type, which store and update the polynomial coefficients, and of a control unit, which computes the function \(D\) and determines whether the discrepancy is zero computing its inverse if necessary (see figure 5.2).

![Figure 5.2: architectures for pBM](image)

More precisely, each circuit is composed of two storage devices, two multipliers and an adder. After \(i\) steps the circuit \(c_j\) for \(j = 0, 1, \ldots, d - 2\) contains the coefficients \(\hat{\Delta}^{(i)}_j\) and \(\hat{\Theta}^{(i)}_j\), while for \(j = d - 1, d, \ldots, d + t - 1\) stores the coefficients \(\sigma^{(i)}_{d+t-1-j}\) and \(\tau^{(i)}_{d+t-1-j}\). During the \((i + 1)\)th step, the coefficients contained in each circuit shift left and the circuit \(c_j\) carries out the following instructions to update the polynomial coefficient which it contains:

\[
\hat{\Delta}^{(i+1)}_j = \hat{\Delta}^{(i)}_{j+1} - \hat{\Delta}^{(i)}_0 \hat{\Theta}^{(i)}_j
\]

\[
\hat{\Theta}^{(i+1)}_j = \begin{cases} 
\hat{\Theta}^{(i)}_j & \text{if } \hat{\Delta}^{(i)}_0 = 0 \text{ or } 2D(i) \geq i + 1 \\
\frac{\hat{\Delta}^{(i)}_{j+1}}{\hat{\Delta}^{(i)}_0} & \text{if } \hat{\Delta}^{(i)}_0 \neq 0 \text{ and } 2D(i) < i + 1
\end{cases}
\]

for \(j = 0, 1, \ldots, d - 2\)

or

\[
\sigma^{(i+1)}_j = \sigma^{(i)}_j - \hat{\Delta}^{(i)}_0 \tau^{(i)}_{j-1}
\]

\[
\tau^{(i+1)}_j = \begin{cases} 
\tau^{(i)}_{j-1} & \text{if } \hat{\Delta}^{(i)}_0 = 0 \text{ or } 2D(i) \geq i + 1 \\
\sigma^{(i)}_j & \text{if } \hat{\Delta}^{(i)}_0 \neq 0 \text{ and } 2D(i) < i + 1
\end{cases}
\]
for \( j = d - 1, d, \ldots, d + t - 1 \)

Step \( pBM.4 \) requires 2 circuits for Chien’s search in order to evaluate the polynomials \( x^{d-1} \hat{\Delta}^{(d-1)}(x) \) and \( \sigma'(x) \) in the field elements. As usual these polynomial evaluations are executed at the same time of the evaluation of \( \sigma(x) \) done in step \( pBM.3 \) with a negligible cost with respect to the others instructions. Following the evaluations, the error values \( E_i \)'s are computed executing simultaneously the divisions needed with \( e \) dividers. Thus, recalling that \( t = \left\lfloor \frac{d-1}{2} \right\rfloor \), the complexity of step \( pBM.2 \) and \( pBM.4 \) can be summarized as in table 5.3. Note that we consider the upper bound of the quantities involved, which is reached when \( d \) is even and \( d - 1 = 2t + 1 \). When \( d \) is odd and \( d - 1 = 2t \), the number of arithmetic operations needed decreases to \( 2t \), while \( 6t + 2 \) multipliers and \( 3t + 1 \) adders are enough.

|        | time complexity          | space complexity       |
|--------|--------------------------|------------------------|
| \( pBM.2 \) | 2t + 1 inversions        | 1 inversion circuit    |
|        | 2t + 1 multiplications   | 6t + 4 multipliers     |
|        | 2t + 1 additions         | 3t + 2 adders          |
| \( pBM.4 \) | 1 division               | \( t \) dividers       |

Table 5.3: parallel complexity of \( pBM.2 \) and \( pBM.4 \)

**Remark 5.2.6.** As seen at the end of section 3.1, the \( BM \) decoding algorithm 3.1.9 can be modified to avoid the inversions computed in step \( BM.2 \). If in this section we consider the polynomials \( \hat{\sigma}^{(i)}(x) \) and \( \hat{\tau}^{(i)}(x) \) (see definition 3.1.15) instead of the polynomials \( \sigma^{(i)}(x) \) and \( \tau^{(i)}(x) \), then in step \( pBM.2 \) we will have the following instructions

\[
\begin{align*}
\hat{\sigma}^{(0)} & := 1; \\
\hat{\tau}^{(0)} & := 1; \\
\hat{\Delta}^{(0)} & := S(x); \\
\hat{\Theta}^{(0)} & := S(x); \\
D(0) & := 0; \\
\beta(0) & := 1;
\end{align*}
\]

for \( i = 0, 1, \ldots, d - 2 \) do in parallel
5.2. PARALLEL IMPLEMENTATION OF THE BM DECODER

\[ \hat{\sigma}^{(i+1)}(x) := \beta(i)\hat{\sigma}^{(i)}(x) - \hat{\Delta}_0^{(i)}x\hat{\tau}^{(i)}(x); \]

\[ \hat{\Delta}^{(i+1)}(x) := \beta(i) \left( \sum_{j=1}^{d-2} \hat{\Delta}_j^{(i)}x^{j-1} \right) - \hat{\Delta}_0^{(i)}\hat{\Theta}^{(i)}(x); \]

if \((\hat{\Delta}_0^{(i)} = 0 \text{ or } 2D(i) \geq i + 1)\) then

\[ D(i + 1) := D(i); \]
\[ \beta(i + 1) := \beta(i); \]
\[ \hat{\tau}^{(i+1)}(x) := x\hat{\tau}^{(i)}(x); \]
\[ \hat{\Theta}^{(i+1)}(x) := \hat{\Theta}^{(i)}(x); \]

else

\[ D(i + 1) := i + 1 - D(i); \]
\[ \beta(i + 1) := \hat{\Delta}_0^{(i)} \]
\[ \hat{\tau}^{(i+1)}(x) := \hat{\tau}^{(i)}(x); \]
\[ \hat{\Theta}^{(i+1)}(x) := \sum_{j=1}^{d-2} \hat{\Delta}_j^{(i)}x^{j-1}; \]

endfor

\[ e := D(d - 1); \]
\[ \hat{\sigma}(x) := \hat{\sigma}^{(d-1)}(x); \]

In this way each discrepancy inversion is replaced by multiplications (of the coefficients of the polynomials involved) executable at the same time the other multiplications already present (see [SS01] for more details). So the polynomials \(\hat{\sigma}^{(d-1)}(x) = b \cdot \sigma^{(d-1)}\) and \(\hat{\Delta}^{(d-1)}(x) = b \cdot \hat{\Delta}^{(d-1)}(x)\), where

\[ b = \prod_{j=0}^{d-2} \beta(j) \]

can be computed by the circuit \(c\) with a complexity upper bounded by:

|               | time complexity | space complexity |
|---------------|----------------|-----------------|
| \(pBM\_2b\)  | 2t + 1 multiplications | 6t + 4 multipliers |
|               | 2t + 1 additions     | 3t + 2 adders    |

Table 5.4: parallel complexity of inversionless \(pBM\_2\)
Finally we note that the error value formula (5.7) used in pBM.4 does not change, indeed it holds that

$$E_i = \frac{\Delta^{(d-1)}(X_i^{-1}) \cdot (X_i^{-1})^{d-1}}{\hat{\sigma}(X_i^{-1})}$$
Conclusions

In this thesis we have studied and proposed several decoding algorithms for Reed-Solomon codes, dwelling on their computational time complexity. In particular, we have proved that, when the decoding is expressed in terms of linear systems and tools of linear algebra, a detailed study of the matrices involved (the syndrome matrix $A$, the matrices $A_i$ and $B_j$) leads to efficient procedures for computing both the error-locator polynomial and the error values. This permits to see the linear algebra approach in a new light, since it is now competitive with other decoding strategies. Finally the linear algebra techniques allow to reach the goal of a parallel implementation of the PGZ decoding strategy. We note that the computational time cost is lower for the $pPGZ$ decoding algorithm 5.1.5 than for the $pBM$ decoding algorithm 5.2.5 (proposed in [SS01]), but the second one allows a parallel implementation that employs a smaller number of circuit elements arranged in a simpler systolic architecture (see table 5.5 on page 103). Thus the $pPGZ$ decoding algorithm 5.1.5 can represent the better choice for Reed-Solomon codes in some special cases, as when the error correction capability is small and the transmission channel is quite good, while the $pBM$ decoding algorithm 5.2.5 allows a linear time decoding with a linear number of hardware elements also in more general cases.

As intermediate results we have proved the following:

- from a theoretical point of view, we have used the $fPGZ$ decoding algorithm 2.2.5 to uncover the existing relationship between the leading principal minors $A_i$ of the syndrome matrix and the discrepancies $\Delta_j$ computed by the $BM$ decoding algorithm 3.1.9. Indeed in theorem 3.3.3 we have proved that if $\det(A_i) \neq 0$, then

$$r = \min\{j \mid \det(A_{i+j}) = 0\} \Rightarrow \Delta_{2i} = \cdots = \Delta_{2i+r-2} = 0 \text{ and } \Delta_{2i+r-1} \neq 0$$

We have achieved this result by comparing the intermediate outcomes of the $fPGZ$ decoding algorithm 2.2.5 with the ones of the $BM$ decoding algorithm 3.1.9.
• we have found the necessary and sufficient conditions that must be added to the fPGZ and the BM decoding algorithms in order to make them \( t \)-bounded distance decoding algorithms;

• for what concerns the error value computation, we have proved a new formula for the BM decoder (used in BM.4b), which needs fewer arithmetic operations than Forney’s formula. Moreover we have noted that step BM.4 (or BM.4b) allows an advantageous pipelined implementation with step BM.3. In step fPGZ.4 this is not possible because the error values are calculated by solving a linear system in which the coefficient matrix is formed by some powers of the \( \sigma(x) \) roots. We have eliminated this disadvantage of the fPGZ decoding algorithm by proving in proposition 4.2.11 that

\[
E_i = -\frac{\varepsilon_{e-\theta} \cdot (X_i^{-1})^{e+\theta-1}}{\sigma'(X_i^{-1}) P_{w(\theta)}(X_i^{-1})}
\]

This formula allows to compute each \( E_i \) using the vector \( w^{(\theta)} \) and coefficient \( \varepsilon_{e-\theta} \), which are byproducts of the computation of \( \sigma(x) \) in fPGZ.2. Moreover it can be executed by a pipelined implementation with step fPGZ.3;

• as regards the parallel implementation, we have proved that the proposed pPGZ decoding algorithm [5.1.3] has an \( O(e) \) multiplicative time complexity with \( O(t \cdot (\lceil t/2 \rceil)) \) circuit elements. Moreover we have studied the pBM decoding algorithm [5.2.4] which has an \( O(t) \) multiplicative time complexity with \( O(t) \) circuit elements. For the second one our main contribution concerns the formalization of the proof of the algorithm correctness.
|                  | time complexity                   | space complexity                   |
|------------------|-----------------------------------|------------------------------------|
| **pPGZ**         | 1 divisions                       | $t$ dividers                       |
|                  | $e + 1$ multiplications            | $t \left( \frac{t}{\lfloor t/2 \rfloor} \right) + 2t$ multipliers |
|                  | $e \left\lceil \log_2 e \right\rceil$ additions | $t \left( \frac{t}{\lfloor t/2 \rfloor} \right)$ adders |
| **pBM** (inversionless) | 1 divisions                       | $t$ dividers                       |
|                  | $2t + 1$ multiplications           | $6t + 4$ multipliers               |
|                  | $2t + 1$ additions                 | $3t + 2$ adders                    |

Table 5.5: complexity comparison
Bibliography

[Ber68] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Series in Systems Science, M. Dekker, 1968.

[BHLN94] V.K. Bhargava, M.A. Hasan, and T. Le-Ngoc, *Algorithms and Architectures for the Design of a VLSI Reed-Solomon Codec*, Reed-Solomon Codes and Their Applications, IEEE Press, 1994.

[Bla03] R.E. Blahut, *Algebraic Codes for Data Transmission*, Cambridge University Press, 2003.

[BP70] A. Björck and V. Pereyra, *Solution of Vandermonde Systems of Equations*, Mathematical Computation 24 (1970), 893–903.

[Chi64] R. T. Chien, *Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes*, IEEE Trans. Inf. Theory 10 (1964), 357–363.

[For65] G. D. Forney, *On Decoding BCH Codes*, IEEE Trans. Inf. Theory IT-11 (1965), 547–557.

[HLL+91] G. Hoffman, D.A. Leonard, C.C. Lidner, K.T. Phelps, C.A. Rodger, and J.R. Wall, *Coding Theory: The Essential*, New York, McGraw-Hill, 1991.

[Hor88] T. Horiguchi, *High-Speed Decoding of BCH Codes Using a New Error-Evaluation Algorithm*, Electronics and Communications in Japan 71-A (1988), no. 12, 2183–2191.

[HR85] G. Heinig and K. Rost, *Algebraic Methods for Toeplitz-like Matrices and Operators*, Operator Theory: Advances and Applications, vol. 13, Birkhäuser, 1985.

[HV95] J. Hong and M. Vetterli, *Simple Algorithms for BCH Decoding*, IEEE Transactions on Communications 43 (1995), 2324–2333.
[LC83] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Computer Applications in electrical engineering, Prentice-Hall, 1983.

[Moo05] T.K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*, Wiley-Interscience, 2005.

[PW72] W.W. Peterson and E.J. Weldon, *Error-correcting Codes*, 2 ed., MIT Press (MA), 1972.

[RS60] I. S. Reed and G. Solomon, *Polynomial Codes over Certain Finite Fields*, Journal of society for Industrial ans Applied Mathematics Rendus de l’Académie des Sciences 8 (1960), 300–304.

[SF97] M. Schmidt and G.P. Fettweis, *Levinson-Durbin Algorithm Used for Fast BCH decoding*, Digital Signal Processing for Communication Systems (T. Wysocki, H. Razavi, and B. Honary, eds.), The Springer International Series in Engineering and Computer Science, vol. 403, 1997.

[SM90] D. V. Sarwate and R. D. Morrison, *Decoder Malfunction in BCH Decoders*, IEEE Trans. Inf. Theory 36 (1990).

[SS94] M. Srinivasan and D. V. Sarwate, *Malfunction in the Peterson-Gorenstein-Zierler Decoder*, IEEE Trans. Inf. Theory 40 (1994).

[SS01] D. V. Sarwate and N. R. Shanbhag, *High-Speed Architectures for Reed-Solomon Decoders*, IEEE Transactions on VLSI Systems 9 (2001), no. 5, 641–655.

[VL99] J.H Van Lint, *Introduction to Coding Theory*, third ed., Graduate Texts in Mathematics, Springer Verlag, 1999.