Johannes-Kepler-Universität Linz, Altenbergerstraße 69, A-4040 Linz

Research Institute for Symbolic Computation

# Diplomarbeit

# Multivariate Polynomial Codes

Diplomarbeit zur Erlangung des akademischen Grades eines
Diplomingenieurs
der Studienrichtung Technische Mathematik
(Studienzweig Mathematische Computerwissenschaften)

Verfasser: Michaela Hackl
Betreuer: Univ.Prof. DI. Dr. Franz Winkler

Wilhering, den 7.2.2000

## Eidesstattliche Erklärung

Ich, Michaela Hackl, erkläre hier an Eides statt, daß ich die vorliegende Arbeit zur Gänze selbst verfaßt und keine Quellen als die angegebenen verwendet habe.

Wilhering, Februar 2000

# Contents

# Preface

1-dimensional cyclic codes or codes derived from cyclic codes play an important role in the application of coding theory. To find even better codes, there are basically two known ways of extending them to higher dimension.
A very natural way leads to the construction of multi-dimensional cyclic codes.
A more tricky way found by V. D. Goppa leads to the construction of algebraic geometric codes, which are also called Goppa codes.
In this diploma thesis we present these two types of codes and show how they both can be decoded using the Berlekamp Massey Sakata algorithm.

In the first chapter we summarize the necessary basic theorems from coding theory.
In Chapter 2 we describe 1-dimensional cyclic codes, show how encoding can be done and present the Berlekamp-Massey algorithm for decoding.
In Chapter 3 we give the basic definitions and theorems of the theory of Gröbner bases, which we will need for multi-dimensional codes.
In Chapter 4 we present multi-dimensional cyclic codes as a natural generalization of 1-dimensional cyclic codes. We show how to encode and decode them and, in particular, how to use the Berlekamp Massey Sakata algorithm for decoding.
In Chapter 5 we give a short introduction to the more successful algebraic geometric codes and show how the Berlekamp Massey Sakata algorithm can be used for decoding algebraic geometric codes.
Finally we present a tutorial implementation, which allows the user to construct 2-dimensional cyclic codes and algebraic geometric codes and to experiment with the decoding algorithms.


# Acknowlegdments

I would like to thank my supervisor Franz Winkler for his support during the work on this thesis. I also would like to thank Harald Deischinger for his valuable help on algebraic geometric codes and UNIX and Shojiro Sakata for his valuable advise to leave the field of multi-dimensional cyclic codes and to concentrate on 1-point algebraic geometric codes.

Wilhering, February 2000 Michaela Hackl

# Chapter 1

# Basics from coding theory

In this chapter we summarize some basic facts and definitions from coding theory, which we shall use in later chapters. Details and proofs can be found in [vL92, LP98].
The basic problem of coding theory is to transmit a message over a noisy channel. To do that, control symbols are added to the message $m$ (the information symbols) in order to be able to detect and correct errors which might occur during the transmission.

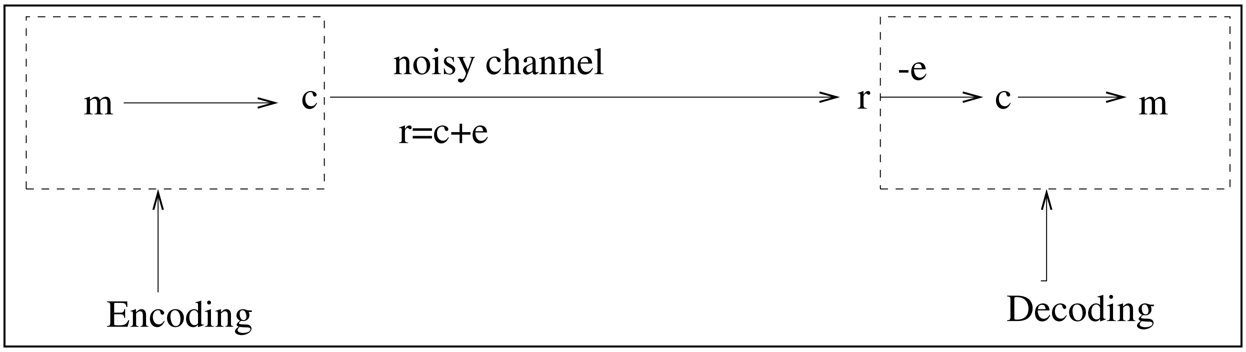



Figure 1.1: Encoding and decoding. The message $m \in Q^k$ has to be sent over a noisy channel. To do this control bits are added to $m$ and the vector $c \in Q^n$ is sent. Decoding consists of finding the error vector and the original message $m$.

**Definition 1.1** *Let $Q$ be a finite non-empty set. A* (block) code *$C$ over the alphabet $Q$ of length $n$ is a non-empty subset of $Q^n$. Elements of $C$ are called* code words*, elements of $Q^n$ are called* words.

**Definition 1.2** *Let $w$ and $w'$ be elements of $Q^n$ and let $C$ be a code.*
*The* Hamming distance *$d(w, w')$ is the number of positions at which $w$ and $w'$ differ,that is*

$$d(w, w') = |\{1 < i \leq n | w_i \neq w'_i\}|.$$

*The* Hamming distance d(w,C) *between a word $w$ and $C$ is defined as*

$$d(w, C) = \min\{d(w, c) | c \in C\}.$$

**Definition 1.3** *Let $C$ be a code over $Q$. The* minimum distance d *of the code $C$ is the minimum Hamming distance of two distinct code words, i.e.*

$$d(C) = \min\{d(c, c')|c \in C, c' \in C, c \neq c'\}.$$

Whenever a word $r$ is received through the channel, we perform a so called 'nearest neighbor decoding' for determining the most likely sent code word $c$. This means that for every received word $r$ the code word $c$ is determined, for which the distance $d(c, r)$ is minimal. Using nearest neighbor decoding, we can correct every occurrence of $t$ errors, if we can find a uniquely determined nearest neighbor for every word $r$ with $d(r, C) \leq t$. This leads to the following definition.

**Definition 1.4** *Let $C$ be a code over $Q$. The code $C$ is called a* t-error correcting code*, iff for every word $w \in Q^n$ with $d(w, C) \leq t$ there exists exactly one $c \in C$ with $d(w, c) \leq t$.*

This definition means that, if only up to t errors are made during the transmission of a code word we can find the errors and correct them.

**Theorem 1.1** *Let $C$ be a code over $Q$ with minimum distance $d$.*
*The code $C$ can correct up to $t = \lfloor \frac{d-1}{2} \rfloor$ errors. $d - 1$ errors can be detected.*

## 1.1 Linear codes

*Notation:* Let $q$ be a prime power. Throughout the following chapters we denote by $\mathbb{F}_q$ the finite field with $q$ elements.
The typical alphabet $Q$ of a linear code is a finite field $\mathbb{F}_q$. So we define a linear code as a code over the alphabet $\mathbb{F}_q$.

**Definition 1.5** *A* linear code C *of length n over the finite field $\mathbb{F}_q$ is a linear subspace of $\mathbb{F}_q^n$.*
*The* dimension *of the code is the vector space dimension of $C$.*
*If a linear code $C$ of length $n$ over $\mathbb{F}_q$ has dimension $k$ and minimum distance $d$, we call this code a* linear $[n, k, d]$-code *over $\mathbb{F}_q$.*

One advantage of linear codes is, that it is easier to determine their minimal distance.

**Theorem 1.2** *Let $C$ be a linear code. Then*

$$d(C) = \min\{d(c, 0)|c \in C\}.$$

Another advantage of a linear code is, that since it is a linear subspace of $\mathbb{F}_q^n$ it can be represented by a basis or a linear function from $\mathbb{F}_q^k$ to $\mathbb{F}_q^n$ which can be represented by a $k \times n$-matrix over $\mathbb{F}_q$.

**Definition 1.6** *Let $C$ be a linear $[n,k,d]$-code over $\mathbb{F}_q$. A $k \times n$ matrix $G$ over $\mathbb{F}_q$ is called a* generator matrix *for $C$ iff $C$ is the image of $G$,i.e.*

$$C = \{a \cdot G | a \in \mathbb{F}_q^k\}.$$

*A $(n-k) \times n$ matrix $H$ over $\mathbb{F}_q$ is called a* parity check matrix *for $C$ iff*

$$C = \{w \in \mathbb{F}_q^n | H \cdot w^t = 0\}.$$

*Remark:* The rows of a generator matrix $G$ of a code $C$ form a basis for $C$. If we have a linear code, we don't have to store all the code words, but only a corresponding generator matrix.
If we know the parity check matrix $H$, we can calculate the minimum distance in the following way:

**Theorem 1.3** *Let $H$ be the parity check matrix of a linear code $C$.*

$$\begin{aligned} d(C) = d' \iff\ & d' \textit{ is the biggest natural number such that any} \\ & d'-1 \textit{ columns of } H \textit{ are linearly independent.} \end{aligned}$$

To be honest, this theorem is not very convenient for calculating $d(C)$, but it is often used for proving a lower bound for the minimal distance of a certain family of codes.
Let $G$ be the generator matrix of a linear $[n,k,d]$-code. From the definition of $G$, we get an easy method for encoding, that is multiplying the message $m$ by $G$. If the multiplication by $G$ doesn't change the first $k$ symbols, the encoding procedure is called *systematic*. This is the case if and only if $G$ has the following form.

**Definition 1.7** *A generator matrix $G$ is called in* canonical form *iff*

$$G = (I_k | A_k^{n-k}).$$

*where $I_k$ is the $k \times k$ identity matrix and $A_k^{n-k}$ can be any $k \times (n-k)$ matrix.*

As already mentioned, decoding a received word $r$ is equivalent to finding its nearest neighbor $c \in C$.
One very easy but inefficient way of decoding would be searching through all the code words. There is a more intelligent way of decoding using syndroms and coset leaders, but nevertheless this method is still too clumsy and requires a lot of storing and time for comparing code words. So far there is no efficient way of decoding all linear codes, but there are efficient ways of decoding for certain families of linear codes, which we will present in Chapter(2.2).

## 1.2 Bounds on linear codes

Let C be a linear $[n,k,d]$-code over $\mathbb{F}_q$ and let $t = \lfloor \frac{d-1}{2} \rfloor$ denote the maximum number of errors the code $C$ can correct. Obviously one would like to construct codes with as little redundancy as possible ($\frac{k}{n}$ should be as high as possible) which can correct many errors ($\frac{d}{n}$ should be as high as possible). These two aims are contradicting, since we can't choose $n$, $k$ and $d$ independently. We now present some bounds we will refer to in later chapters.

**Theorem 1.4** *The parameters $q$, $n$, $t$ and $k$ satisfy the inequality*

$$q^k(1 + (q-1)\binom{n}{1} + \cdots + (q-1)^t\binom{n}{t}) \leq q^n.$$

*This bound is called the* Hamming bound.

**Definition 1.8** *A $t$-error correcting linear $[n, k, d]$-code $C$ is called* perfect *iff $C$ achieves the Hamming bound, i.e.*

$$q^k(1 + (q-1)\binom{n}{1} + \cdots + (q-1)^t\binom{n}{t}) = q^n.$$

**Theorem 1.5** *The parameters $n$, $d$ and $k$ satisfy the inequality*

$$d \leq n - k + 1.$$

*This bound is called the* Singleton bound.

*Remark:* The last theorem is a consequence of Theorem 1.3 and Definition 1.6.

# Chapter 2

# 1-dimensional cyclic codes

In this chapter we summarize basic facts about 1-dimensional cyclic codes, because most of these facts have a direct counterpart in the multi-dimensional case. Proofs of the theorems presented here can be found in [vL92, LP98]. At the end of the chapter, we present two important algorithms for decoding BCH-codes, which have also generalizations which can be used for decoding multi-dimensional cyclic codes.

**Definition 2.1** *A linear code $C$ of length $n$ over $\mathbb{F}_q$ is called a* (1-dimensional) cyclic code*, iff for every code word $c \in C$ every cyclic shift of $c$ is also a code word, i.e.*

$$c = (c_1, c_2, ..., c_n) \in C \Longrightarrow (c_2, c_3, ..., c_n, c_1) \in C.$$

*If a cyclic code of length $n$ has dimension $k$ and minimum distance $d$, we call it a* cyclic $[n, k, d]$-code.

## 2.1 The structure of 1-dimensional cyclic codes

*Notation:* Throughout this chapter let $W_n$ denote $\mathbb{F}_q[x]_{/<x^n-1>}$ . Let $q$ be equal to $p^k$ where $p$ denotes a prime number and $k$ an integer number.

*Convention:* We are certainly aware of the fact that the elements in $W_n$ are equivalence classes. For convenience we use representatives (the polynomials over $\mathbb{F}_q$ of degree less than $n$) for these equivalence classes and skip the equivalence class brackets.

Cyclic codes have a special algebraic structure, which we can use for constructing, encoding and decoding.
We use the following isomorphism $f$ to identify elements of $\mathbb{F}_q^n$ with elements in $W_n$.

$$f : \mathbb{F}_q^n \to W_n$$

$$w = (w_0, w_2, \cdots, w_{n-1}) \to w_0 + w_1 x + \cdots + w_{n-1} x^{n-1}.$$

Here the multiplication of two elements $w$ and $w'$ in $\mathbb{F}_q^n$ is defined by

$$w * w' := f^{-1}(f(w) \cdot f(w')).$$

*Notation:* Since a code $C$ can be interpreted as a subset of $W_n$ and as a subset of $\mathbb{F}_q^n$, we will use both representations of a code $C$. Further on $c$ will denote an element in the vector representation of $C$ and $c(x) := f(c)$ the corresponding element in $W_n$. Here $C$ can either denote the cyclic code $C$ as defined in Definition 2.1 or its isomorphic representation as a subset of $W_n$.
Using the isomorphism $f$ and the fact that the multiplication by $x$ in $W_n$ corresponds to a cyclic shift in $\mathbb{F}_q^n$, we get the following theorem.

**Theorem 2.1** *A linear code $C \subseteq W_n$ is cyclic if and only if $C$ is an ideal in $W_n$.*

**Theorem 2.2** *Let $C$ be an ideal $\neq \{0\}$ of $W_n$. Then there exists a unique $g(x) \in W_n$ with the following properties:*

- *$g(x)|x^n - 1$ in $\mathbb{F}_q(x)$*
- *$C =< g(x) >$*
- *$g(x)$ is monic*

*Proof.* Let $g(x)$ be a monic polynomial of minimal degree in $C \setminus \{0\}$. $g(x)|x^n - 1$, since otherwise there would be a polynomial $r(x)$ with $\deg(r(x)) < \deg(g(x))$ and $r(x) = x^n - 1 - q(x)g(x)$. Since $C$ is an ideal in $W_n$, $r(x)$ would also be a code word, contradicting the assumption on $g(x)$.
Since $g(x)$ is a code word and $C$ is an ideal in $W_n$, $< g(x) >\subseteq C$. On the other hand let $h(x)$ be a code word. Suppose $g(x) \nmid h(x)$. Let $r(x) = h(x) - q(x)g(x)$ with $\deg(r(x)) < \deg(g(x))$. Then $r(x)$ would be a code word, contradicting the assumption on $g(x)$. So $< g(x) >= C$.
Finally let $h(x)$ be a code word which fulfills the three conditions. Then we get that $g(x)|h(x)$ and $h(x)|g(x)$. Since both $g(x)$ and $h(x)$ have to be monic, $g(x) = h(x)$ and $g(x)$ is unique. $\square$

Hence $W_n$ is a principal domain and every 1-dimensional cyclic code can be represented by a monic polynomial which divides $x^n - 1$. Therefore all cyclic codes of length $n$ are determined by the factors of $x^n - 1$.

**Definition 2.2** *Let $C$ be a cyclic code over $\mathbb{F}_q$. The polynomial $g(x)$ in Theorem 2.2 is called the* generator polynomial *of the code $C$. $h(x) := \frac{x^n-1}{g(x)}$ is called the* parity check polynomial *of $C$.*

Therefore the generator polynomial has the same meaning for cyclic codes as the generator matrix for linear codes. It determines the code and can be used for encoding.

**Theorem 2.3** *Let $g(x)$ be the generator polynomial of a cyclic $[n, k, d]$-code $C$ over $\mathbb{F}_q$. Then the degree of $g(x)$ is equal to $n - k$ and*

$$c(x) \in C \iff c(x) = m(x)g(x).$$

*Here $m(x)$ is a polynomial of degree less or equal to $k - 1$.*

So encoding can be done by multiplying the message polynomial by the generator polynomial.
The parity check polynomial has a similar meaning for cyclic codes as the parity check matrix for linear codes.

**Theorem 2.4** *Let $C$ be a cyclic code and $h(x)$ its parity check polynomial. Then*

$$c(x) \in C \Longleftrightarrow h(x)c(x) = 0.$$

One way of defining a cyclic code is to define it by its generator polynomial $g(x)$. An equivalent way is to define it by the roots of its generator polynomial $g(x)$. Since $g(x)$ generates $C$, the roots of $g(x)$ are the common roots of all polynomials in $C$.
From the roots of $g(x)$, we get some information about the minimal distance of the generated code.

**Theorem 2.5** *Let $g(x)$ be the generator polynomial of a cyclic $[n, k, d]$-code. Let $\alpha$ be a primitive $n$-th root of unity.*
*If $g(x)$ vanishes on $r$ subsequent powers of $\alpha$ then the minimum distance $d(C)$ is at least $r + 1$.*

Now we show in an example, how to use Theorem 2.2 and the last theorem to construct a cyclic code.

**Example 2.1** *Let $n$ be 15 and $\mathbb{F}_q$ be $\mathbb{F}_2$. We want to construct a binary cyclic code of length 15, which can correct at least 2 errors. Since we want to construct a 2-error correcting code, we have to find a binary polynomial which divides $x^{15} - 1$ and vanishes on 4 subsequent powers of $\alpha$, where we choose $\alpha$ as a root of $x^4 + x + 1$ in an extension field of $\mathbb{F}_2$.*
*To be able to construct the binary polynomial, we need a simple fact: Let $q = p^r$ and $0 \neq f(x) \in \mathbb{F}_q(x)$.*

$$f(x) \in \mathbb{F}_p(x) \Longleftrightarrow \text{for all } \alpha \in \mathbb{F}_q\text{: } (f(\alpha) = 0 \Longrightarrow f(\alpha^p) = 0).$$

*So we know that a binary polynomial which vanishes on $\alpha$, has to vanish also on $\alpha^2$, $\alpha^4$ and $\alpha^8$. The non-zero binary polynomial of minimal degree which vanishes on $\alpha$ is called its minimal polynomial and in this case it is*

$$m_\alpha = (x - \alpha)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8) = x^4 + x + 1.$$

*In a similar way we get:*

$$\begin{aligned} m_{\alpha^3}(x) &= (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^9) = x^4 + x^3 + x^2 + x + 1 \\ m_{\alpha^5}(x) &= (x - \alpha^5)(x - \alpha^{10}) = x^2 + x + 1 \\ m_{\alpha^7}(x) &= (x - \alpha^7)(x - \alpha^{14})(x - \alpha^{13})(x - \alpha^{11}) = x^4 + x^3 + 1. \end{aligned}$$

*We now want to get a binary polynomial of least possible degree which vanishes on at least 4 subsequent powers of $\alpha$. We take $g(x)$=$m_\alpha(x) \cdot m_{\alpha^3}(x)$ with degree 8 vanishing on $\alpha$, $\alpha^2$, $\alpha^3$ and $\alpha^4$. By this way we constructed a cyclic $[15, 7, d]$-code where $d \geq 5$.*

In the last example we constructed a cyclic code, which belongs to a special class of cyclic codes, we will consider in the next chapter.

## 2.2 BCH and Reed-Solomon codes

**Definition 2.3** *A cyclic code of length $n$ over $\mathbb{F}_q$ is called a* BCH code *of designed distance $\delta$ if its generator polynomial $g(x)$ is the least common multiple of the minimal polynomials of $\alpha^l$,$\alpha^{l+1}$,...,$\alpha^{l+\delta-2}$ for some $l \in \mathbb{Z}$, where $\alpha$ is a primitive $n$-th root of unity. If $l = 1$ we will call the BCH-code a* narrow-sense BCH-code.

*Remark:* In Example 2.1 we constructed a BCH code of length 15, with $\delta = 5$ and $l = 1$. Since the generator polynomial of a BCH code of designed minimum distance $\delta$ obviously vanishes on $\delta - 1$ subsequent powers of $\alpha$ we get from Theorem 2.5:

**Theorem 2.6** *Let $C$ be a BCH-code of designed distance $\delta$.Then*

$$d(C) \geq \delta.$$

**Definition 2.4** *Let $q$ be a prime power. A BCH code of length $q - 1$ over $\mathbb{F}_q$ of designed distance $\delta$ is called a* Reed-Solomon code *of designed distance $\delta$.*

Let $C$ be a Reed-Solomon code of designed distance $\delta$ and let $\alpha$ be a primitive element in $\mathbb{F}_q$. Because of Theorem 2.6 we have that $d(C) \geq \delta$.
On the other hand $g(x) = (x - \alpha^l)...(x - \alpha^{l+\delta-2})$ and $\deg(g(x)) = \delta - 1$. Since for every cyclic code $k = n - \deg(g(x))$ we get that $k = q - 1 - (\delta - 1) = q - \delta$. If we plug this into the Singleton bound in Theorem 1.5 we get that $q - 1 - (q - \delta) + 1 \geq d(C)$ which means that $d(C) \leq \delta$.

**Theorem 2.7** *Let $C$ be a Reed-Solomon code of designed distance $\delta$.Then*

$$d(C) = \delta$$

*and $C$ reaches the Singleton bound.*

## 2.3 Decoding BCH-codes

Throughout this chapter, let $C$ be a narrow-sense BCH-code ($l = 1$) of designed distance $\delta$ and length $n$ over the field $\mathbb{F}_q$. Let $t$ denote $\lfloor \frac{\delta-1}{2} \rfloor$ and let the generator polynomial $g(x)$ of $C$ be the least common multiple of the minimal polynomials of $\alpha$, $\alpha^2$,...,$\alpha^{2t}$, where $\alpha$ is a primitive $n$-th root of unity. Let $\mathbb{F}_{\text{ext}}$ denote an extension field of $\mathbb{F}_q$ which contains $\alpha$. Suppose we wanted to transmit the code word $c(x) \in C$ and the channel noise causes an additive error $e(x)$. We receive the word $r(x) = c(x) + e(x)$. Let $m := d(c, r) = d(e, 0)$ denote the actual number of errors which occurred during the transmission and $i_1, ..., i_m$ the positions at which the errors occurred.
Since every code word has to vanish on $\alpha$,. . .,$\alpha^{2t}$ we get that

$$r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = \sum_{k=1}^{m} e_{i_k} \alpha^{j i_k} =: s_j. \tag{2.1}$$

More generally we define for all $j > 0$

$$s_j := \sum_{k=1}^{m} e_{i_k} \alpha^{j i_k}.$$

**Definition 2.5** *The polynomial* $s(x) := \sum_{j=1}^{2t} s_j x^j \in \mathbb{F}_{\text{ext}}[x]$ *is called the* syndrome polynomial *of the (received) word* $r(x)$*, with* $s_j$ *defined in (2.1).*

For convenience in further calculations we define the generating function $s'(x) := \sum_{j=1}^{\infty} s_j x^j$, which has periodic coefficients and for which $s'(x) \equiv s(x) \mod x^{2t+1}$. Here $s_j$ is unknown for $j > 2t$.
In (2.1) we set $X_k := \alpha^{i_k}$ and $e_{i_k} := Y_k$ and get for $j \in 1, ..., 2t$

$$\sum_{k=1}^{m} Y_k X_k^j = s_j. \tag{2.2}$$

In general these equations will have many solutions, each corresponding to a different error pattern. The decoder should find a solution with $m \leq t$.

**Definition 2.6** *The polynomial*

$$\sigma(x) := (1 - X_1 x)(1 - X_2 x) \cdots (1 - X_m x) = 1 + \sum_{i=1}^{m} \sigma_j x^j$$

*is called the* error-locator polynomial*.*

**Definition 2.7** *The polynomial*

$$\omega(x) := \sum_{i=1}^{m} x X_i Y_i \prod_{j=1, j \neq i}^{m} (1 - X_j x)$$

*is called the* error evaluator polynomial*.*

Since $\sigma(0) = 1$, the generating function of the quotient $\frac{\omega(x)}{\sigma(x)}$ is well defined. Its coefficients are given by

$$\frac{\omega(x)}{\sigma(x)} = \sum_{j=1}^{m} \frac{x X_j Y_j}{1 - X_j x} = \sum_{k=1}^{\infty} (\sum_{j=1}^{m} Y_j X_j^k) x^k = s'(x)$$

Because of these calculations we get the so called *key equation*, which plays a fundamental role in the decoding of BCH-codes.

$$s(x)\sigma(x) \equiv \omega(x) \mod x^{2t+1} \tag{2.3}$$

**Theorem 2.8** *Consider the key equation in (2.3). Suppose that* $m \leq t$*. For given* $s(x)$*, there is exactly one pair of functions* $(\sigma(x), \omega(x)) \in (\mathbb{F}_{\text{ext}}[x])^2$ *which satisfies the key equation and the following conditions:*

- $\sigma(x)$ *and* $\omega(x)$ *are relatively prime.*
- $\sigma(0) = 1$.

- $\deg(\sigma(x)) \leq t$ *and* $\deg(\omega(x)) \leq t$.

*Proof.* Obviously the error-locator polynomial $\sigma(x)$ and the error evaluator polynomial $\omega(x)$ solve the key equation and satisfy the conditions. Suppose there were two solutions $(\sigma(x), \omega(x))$ and $(\tilde{\sigma}(x), \tilde{\omega}(x))$ to the key equation, satisfying the conditions. Then we get that

$$s(x)\sigma(x) \equiv \omega(x) \mod x^{2t+1} \tag{2.4}$$
$$s(x)\tilde{\sigma}(x) \equiv \tilde{\omega}(x) \mod x^{2t+1}. \tag{2.5}$$

From these two equations we get that

$$\sigma(x)\tilde{\omega}(x) \equiv \tilde{\sigma}(x)\omega(x) \mod x^{2t+1}.$$

Because of the degree bounds for $\sigma(x)$, $\omega(x)$, $\tilde{\sigma}(x)$ and $\tilde{\omega}(x)$ the two sides of the equivalence relation are equal. Therefore

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

Since $\tilde{\sigma}(x)$ and $\tilde{\omega}(x)$ are relatively prime, this is only possible, if $\tilde{\omega}(x)$ divides $\omega(x)$. Vice versa

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

Since $\sigma(x)$ and $\omega(x)$ are relatively prime, this is only possible, if $\omega(x)$ divides $\tilde{\omega}(x)$. Therefore $\omega(x)$ must be equal to $c \cdot \tilde{\omega}(x)$, where c is a constant $\in \mathbb{F}_{\text{ext}}$. But since $\omega(0) = \tilde{\omega}(0) = 1$, c must be 1 and we get that $\omega(x) = \tilde{\omega}(x)$ and further on that $\sigma(x) = \tilde{\sigma}(x)$. □

There are many different methods for solving the key equation. The two best known fast ways of solving it are using GCD-computation and the Berlekamp-Massey algorithm.

## 2.4 Decoding BCH-codes using GCD-computation

The following algorithm was first presented by Sugiyama et al. in 1975 [Sug75]. It uses the Extended Euclidean Algorithm.

**Definition 2.8** *Let $a(x)$ and $b(x)$ be elements of $\mathbb{F}_{\text{ext}}[x]$. The* greatest common divisor *of $a(x)$ and $b(x)$ (denoted by* $\text{GCD}(a(x), b(x))$*) is the monic polynomial of maximum degree which divides both $a(x)$ and $b(x)$.*

The Euclidean Algorithm is based on the fact that $\text{GCD}(a(x), b(x)) = \text{GCD}(b(x), \text{Rem}(a(x), b(x)))$ for two polynomials $a(x)$ and $b(x)$ and that $\text{GCD}(a(x), 0) = a(x)$ for all polynomials $a(x)$. Here $\text{Rem}(a(x), b(x))$ is the remainder $r(x)$ of the division of $a(x)$ by $b(x)$:

$$a(x) = b(x)q(x) + r(x) \quad \text{where} \quad \deg(r(x)) \leq \deg(a(x)) \text{ or } r = 0.$$

The polynomial $q(x)$ is called the quotient and denoted by $\text{Quo}(a(x), b(x))$.

For the GCD$(a(x), b(x))$ we can find two polynomials $u(x)$,$v(x)$ called the co-factors which satisfy the equation

$$\mathrm{GCD}(a(x), b(x)) = u(x)a(x) + v(x)b(x).$$

If one wants to calculate the GCD$(a(x), b(x))$ and the co-factors, one often uses the following algorithm. Details and degree bounds for the co-factors can be found in [Win96].

**Algorithm 2.1 (Extended Euclidean Algorithm)** *Without restriction let* $\deg(b(x)) \leq \deg(a(x))$.

**Step1.** $a_0(x) := a(x)$; $a_1(x) := b(x)$; $u_0(x) := 1$; $v_0(x) := 0$; $u_1(x) := 0$; $v_1(x) := 1$; $i := 2$;

**Step2.** *While* $(a_{i-1}(x) \neq 0)$

$a_i(x) := \mathrm{Rem}(a_{i-2}(x), a_{i-1}(x))$
$u_i(x) := u_{i-2}(x) - u_{i-1}(x) \cdot \mathrm{Quo}(a_{i-2}(x), a_{i-1}(x))$
$v_i(x) := v_{i-2}(x) - v_{i-1}(x) \cdot \mathrm{Quo}(a_{i-2}(x), a_{i-1}(x))$

$i := i + 1$

**Step3.** *GCD*$(a(x), b(x)) := a_{i-2}(x)$ *and the co-factors are* $u(x) := u_{i-2}(x)$ *and* $v(x) := v_{i-2}(x)$

In every step the condition $a_i(x) = u_i(x)a_0(x) + v_i(x)a_1(x)$ holds. This is the main idea why this algorithm is used to solve the key equation. If we start the algorithm with $a_0(x) = x^{2t+1}$ and $a_1(x) = s(x)$ we get that $a_i = u_i x^{2t+1} + v_i s(x)$ which means that $a_i \equiv v_i s(x) \bmod x^{2t+1}$. Therefore $a_i$ and $v_i$ solve the key equation for every $i$. We now only have to take care of the degree bounds, and ensure that $a_i$ and $v_i$ are relatively prime. Then using Theorem 2.8, we can be sure that we have really found the error-locator and the error-evaluation-polynomial. The exact proof for this algorithm, which is straight forward, can be found in ([Sug75]) and ([Adá91]).

## 2.5 Berlekamp-Massey algorithm

In the description of the Berlekamp-Massey algorithm for solving the key equation we use notations from [Ber68], but mostly follow Massey's description of the Berlekamp algorithm in [Mas69], because this description of the Berlekamp-Massey algorithm was extended by Sakata in [Sak86] to the multi-dimensional case.
In order to stick to Sakata's notation (and this is in fact really only a matter of notation), we do not search for a polynomial vanishing on the inverses of $\alpha^{i_j}$, $j \in \{1, ...m\}$ (the error-locator polynomial as defined in the previous section), but for a polynomial of minimal degree vanishing on all $\alpha^{i_j}$, $j \in \{1, ..., m\}$.

**Definition 2.9** *Let* $i_i, ...i_m$ *be the error positions. The monic polynomial of minimal degree vanishing on all the error positions, i.e*

$$\tilde{\sigma}(x) := \prod_{j=1}^{m}(x - \alpha^{i_j})$$

*is called the* error-locator polynomial. *The ideal of all polynomials vanishing on* $\alpha^{i_j}$*,* $j = 1, ..., m$*, is called the* error-locator ideal.

*Remark:* We get that $\sigma(x) = x^{deg(\sigma(x))}\tilde{\sigma}(\frac{1}{x})$, which means that $\sigma(x)$ is a 'reversed version' of $\tilde{\sigma}(x)$, that is $\sigma_i = \tilde{\sigma}_{deg(\sigma(x))-i}$ for all $i = 0, \ldots, \deg(\sigma)$ and $\deg(\sigma(x)) = \deg(\tilde{\sigma}(x))$. Remember the equation, which led to the key equation. We had that $s'(x)\sigma(x) = \omega(x)$. Since the degree of $\omega(x)$ is at most $m$, where $m$ is the number of errors and the degree of $\sigma$, we get that

$$\omega_{j+1} = \sum_{i=0}^{deg(\sigma)} \sigma_i s_{j-i+1} = 0 \qquad \text{for all } j \geq \deg(\sigma(x)).$$

If we rewrite this condition for $\tilde{\sigma}(x)$, we get that

$$\sum_{i=0}^{deg(\tilde{\sigma})} \tilde{\sigma}_i s_{j-deg(\tilde{\sigma}(x))+i+1} = 0 \text{ for all } j \geq deg(\tilde{\sigma}(x)).$$

**Definition 2.10** *A polynomial* $f(x) = \sum_{i=0}^{l} f_i x^i$ *of degree* $l$ generates an infinite sequence $(c_1, c_2, ...)$ *, iff*

$$\sum_{i=0}^{l} c_{j-l+i+1} f_i = 0 \textit{ for all } j \geq l.$$

**Theorem 2.9** *The error-locator ideal is exactly the ideal of all polynomials generating the infinite sequence* $\{s_1, ...\}$*.*

*Proof.* Let $f$ be an element of the error-locator ideal.

$$f(\alpha^{i_j}) = 0 \text{ for } j \in 1, ..., m \Rightarrow \sum_{j=1}^{m} e_{i_j} \alpha^{\beta i_j} f(\alpha^{i_j}) = 0 \qquad \text{for all } \beta \in \mathbb{N}$$

But

$$\sum_{j=1}^{m} e_{i_j} \alpha^{\beta i_j} f(\alpha^{i_j}) = \sum_{k=0}^{deg(f)} f_k \sum_{j=1}^{m} e_{i_j} (\alpha^{k+\beta})^{i_j} = \sum_{k=0}^{\deg(f)} f_k s_{k+\beta} \text{ for all } \beta \geq 1$$

This implies that:

$$\sum_{k=0}^{deg(f(x))} f_k s_{k+\beta-deg(f(x))+1} = 0 \qquad \text{for all } \beta \geq \deg(f(x))$$

Let on the other hand $h(x)$ be a polynomial generating the infinite sequence $(s_1, ...)$:

$$\sum_{k=0}^{deg(h(x))} h_k s_{k+\beta-deg(h(x))+1} = 0 \qquad \text{for all } \beta \geq \deg(h(x))$$

But

$$\sum_{k=0}^{deg(h(x))} h_k s_{k+\beta-\deg(h(x))+1} = \sum_{k=0}^{\deg(h(x))} h_k \sum_{j=1}^{m} e_{i_j} (\alpha^{k+\beta-\deg(h(x))+1})^{i_j} =$$

$$\sum_{j=1}^{m} e_{i_j} \alpha^{(\beta-\deg(h(x))+1)i_j} \sum_{k=0}^{\deg(h(x))} h_k(\alpha^{i_j})^k$$

Set $e'_{i_j} := e_{i_j} \sum_{i=0}^{\deg(h(x))} h_k(\alpha^{i_j})^k$. For this we get that

$$\sum_{j=1}^{m} e'_{i_j}(\alpha^{\beta-\deg(h(x))+1})^{i_j} = 0 \text{ for all } \beta - \deg(h(x)) \geq 0.$$

This means that $e'_{i_j}$ is an error pattern, for which all the syndroms are 0, which implies that $e'_{i_j} = 0$ for all $1 \leq j \leq m$. But then (since $e_{i_j}$ doesn't vanish by definition), we get that

$$\sum_{k=0}^{\deg(h(x))} h_k(\alpha^{i_j})^k = 0 \text{ for all } j \in \{1, ..., m\}.$$

□

We can't calculate all the syndroms from the received word $r$, but only $s_i$ for $1 \leq i \leq 2t$. But we will show afterwards that if we have a polynomial of minimal degree which generates the finite sequence $(s_1, ..., s_{2t})$ and not more than t errors occurred, the same polynomial automatically generates the infinite sequence $(s_1, ...)$.

**Definition 2.11** *A polynomial $f(x) = \sum_{i=0}^{l} f_i x^i$ of degree $l$* generates a finite sequence *$(c_1, c_2, ..., c_n)$, iff*

$$\sum_{i=0}^{l} c_{j-l+i+1} f_i = 0 \textit{ for all } l \leq j \leq n-1.$$

To stress the relationship between this definition and the key equation in the previous section, consider once again the key equation in matrix form if exactly $m$ errors occurred:

$$\begin{pmatrix} s_1 & 0 & 0 & 0 & \dots & \dots & 0 \\ s_2 & s_1 & 0 & 0 & \dots & \dots & 0 \\ s_3 & s_2 & s_1 & 0 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ s_m & s_{m-1} & \dots & \dots & s_2 & s_1 & 0 \\ s_{m+1} & s_m & \dots & \dots & \dots & \dots & s_1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ s_{2t} & s_{2t-1} & \dots & \dots & \dots & \dots & s_{2t-m} \end{pmatrix} \begin{pmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \dots \\ \dots \\ \sigma_{m-1} \\ \sigma_m \end{pmatrix} = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \dots \\ \dots \\ \omega_m \\ 0 \\ 0 \\ \dots \\ \dots \\ \dots \\ 0 \end{pmatrix}$$

From this we get that we can first calculate the coefficients of $\sigma(x)$ independently from the coefficients of $\omega(x)$ using the last equations. Since $\sigma_i = \tilde{\sigma}_{\deg(\sigma)-i}$ we get that the last equations are equivalent to the following equations for the coefficients of $\tilde{\sigma}$:

$$\sum_{i=0}^{\deg(\tilde{\sigma}(x))} s_{j+1+i-\deg(\tilde{\sigma}(x))} \tilde{\sigma}_i = 0 \text{ for all } \deg(\tilde{\sigma}(x)) \leq j \leq 2t-1$$

which means that $\tilde{\sigma}(x)$ generates the finite sequence $s_1, ...s_{2t}$. On the other hand, if a polynomial generates the finite sequence $s_1, ..., s_{2t}$, its 'reversed' polynomial can be used to determine the coefficients of a corresponding $\omega(x)$ and we can find a solution of the original key equation.
According to this new notation, the problem of finding the error locator polynomial is equivalent to finding a non-zero polynomial of minimal degree, which generates the finite sequence $(s_1, ..., s_{2t})$.
The following two theorems are important for developing an algorithm for finding a non-zero polynomial of minimal degree, generating a given sequence.

**Theorem 2.10** *If some polynomial $f$ of degree $l$ generates the sequence $(c_1, ..., c_N)$ but fails to generate the sequence $(c_1, ..., c_N, c_{N+1})$ then the degree $l'$ of any non-zero polynomial generating $(c_1, ..., c_{N+1})$ must satisfy:*

$$l' \geq N + 1 - l$$

*Proof.* If $l \geq N$ the theorem is trivially true. So we may suppose that $l < N$.
Let $f$ be the polynomial of degree $l$, generating the sequence $(c_1, ..., c_N)$, which fails to generate $(c_1, ..., c_{N+1})$. For this polynomial we have:

$$\sum_{i=0}^{l} c_{j+1+i-l} f_i \quad = \quad 0 \qquad \text{for all } l \leq j \leq N-1 \tag{2.6}$$

$$\sum_{i=0}^{l} c_{N+1+i-l} f_i \quad := \quad d_{N+1} \neq 0. \tag{2.7}$$

Assume that there is a non-zero polynomial $h$ of degree $l'$ with $l' \leq N - l$ which generates $(c_1, ..., c_{N+1})$. For this polynomial we have

$$\sum_{i=0}^{l'} c_{j+1+i-l'} h_i = 0 \qquad \text{for all } l' \leq j \leq N. \tag{2.8}$$

From the last equation, we get for $0 \leq k \leq l$ if we choose $j$ as $N - l + k$ (this is possible because $\{N - l, ..., N\}$ is a subset of $\{l', ..., N\}$) by solving equation (2.8) for $c_{N+1+k-l}$

$$c_{N+k+1-l} = -\frac{1}{h_{l'}} \sum_{j=0}^{l'-1} c_{j+N-l+k-l'+1} h_j \qquad \text{for } 0 \leq k \leq l. \tag{2.9}$$

By the use of the last equation we get that

$$\sum_{i=0}^{l} c_{N+1+i-l} f_i = -\sum_{i=0}^{l} f_i \frac{1}{h_{l'}} (\sum_{j=0}^{l'-1} c_{j+N-l+i-l'+1} h_j). \tag{2.10}$$

Changing the order of summation in equation (2.10) we get that

$$\sum_{i=0}^{l} c_{N+1+i-l} f_i = -\frac{1}{h_{l'}} \sum_{j=0}^{l'-1} h_j \sum_{i=0}^{l} c_{j+N-l+i-l'+1} f_i = 0.$$

Here the last equality can be derived from equation (2.6) because $\{j+N-l'|0 \leq j \leq l'-1\}$ is a subset of $\{l, ..., N-1\}$, since by assumption $l' \leq N-l$. The last equality contradicts (2.7). □

Let $L_N(c)$ denote the minimal degree of a non-zero polynomial generating the sequence $(c_1, \cdots, c_N)$.

**Theorem 2.11** *If some polynomial of length $L_N(c)$ generates the sequence $(c_1, \cdots, c_N)$, also generates the sequence $(c_1, \cdots, c_{N+1})$, then $L_{N+1}(c) = L_N(c)$. Conversely, if some polynomial of degree $L_N(c)$ generates $(c_1, \cdots, c_N)$, but fails to generate $(c_1, \cdots, c_{N+1})$, then $L_{N+1}(c) = \max\{L_N(c), N+1-L_N(c)\}$.*

*Proof.* Since the sequence $(L_k(c))_{k\in\mathbb{N}}$ is obviously non-decreasing with respect to $k$, we immediately get from Theorem 2.10 that $L_{N+1}(c) \geq \max(\{L_N(c), N+1-L_N(c)\}$.
In order to show that even equality holds, we construct a polynomial generating the sequence $(c_1, ..., c_{N+1})$ with degree $\max\{L_N(c), N+1-L_N(c)\}$. To construct this, we assume that we know a polynomial of degree $L_m(c)$ generating $(c_1, ..., c_m)$ for all $m < N+1$. If the polynomial of degree $L_N(c)$, let's call it $f$, generates the sequence $(c_1, ..., c_{N+1})$, we can take $f$ and are done. If not we get that

$$\begin{aligned}
\sum_{i=0}^{L_N(c)} c_{i+j-L_N(c)+1} f_i &= 0 \qquad \text{for all } L_N(c) \leq j \leq N-1 && (2.11)\\
\sum_{i=0}^{L_N(c)} c_{i+N-L_N(c)+1} f_i &= d_{N+1} \neq 0. && (2.12)
\end{aligned}$$

Let $n$ be the sequence length before the last change in the sequence $(L_k(c))_{k\leq N}$,i.e

$$\begin{aligned}
L_n(c) &< L_N(c) \qquad \text{and} && (2.13)\\
L_{n+1}(c) &= L_N(c). && (2.14)
\end{aligned}$$

Let g be a polynomial of degree $L_n(c)$ generating the sequence $(c_1, ..., c_n)$. Since $L_{n+1}(c) > L_n(c)$, g can't generate $(c_1, ..., c_{n+1})$, therefore we get that

$$\begin{aligned}
\sum_{i=0}^{L_n(c)} c_{i+j-L_n(c)+1} g_i &= 0 \qquad \text{for all } L_n(c) \leq j \leq n-1 && (2.15)\\
\sum_{i=0}^{L_n(c)} c_{i+n-L_n(c)+1} g_i &= d_{n+1} \neq 0. && (2.16)
\end{aligned}$$

By induction hypotheses, (2.13) and (2.14), we know that

$$L_{n+1}(c) = L_N(c) = \max\{L_n(c), n+1-L_n(c)\} = n+1-L_n(c).$$

Therefore we get that

$$L_n(c) = n + 1 - L_N(c). \tag{2.17}$$

The polynomial

$$h(x) := x^{r-L_N(c)} f(x) - \frac{d_{N+1}}{d_{n+1}} x^{n-N-L_n(c)+r} g(x)$$

has degree at most $r$, where $r$ denotes $\max\{L_N(c), N+1-L_N(c)\}$.
Moreover the polynomial $h(x)$ generates the sequence $(c_1, ..., c_{N+1})$ because

$$\sum_{i=0}^{\deg(h(x))} c_{i+j-L_{N+1}(c)+1} h_i = \sum_{i=0}^{L_N(c)} c_{i+j+1-L_N(c)} f_i - \frac{d_{N+1}}{d_{n+1}} \sum_{i=0}^{L_n(c)} g_i c_{i+j+1+n-N-L_n(c)} =$$

$$= \begin{cases} 0 & j = \deg(h(x)), ..., N-1 \\ d_{N+1} - \frac{d_{N+1}}{d_{n+1}} d_{n+1} = 0 & j = N \end{cases}$$

The last equality follows from the equations 2.17, 2.15, 2.11 and the definition of $r$. $\square$

Theorem (2.10) tells us a bound for the degree of a polynomial generating the sequence $(c_1, \cdots, c_{N+1})$, if we know a polynomial with degree $L_N(c)$ generating $(c_1, \cdots, c_N)$. The second theorem tells us, that this bound can be reached and since the proof of this theorem is constructive, we get a way how to construct a polynomial of degree $L_{N+1}(c)$ generating $(c_1, \cdots, c_{N+1})$. So we get the following algorithm for calculating a polynomial of minimal degree generating a given sequence c.

**Algorithm 2.2 (Berlekamp-Massey algorithm)**

**Input:**

- $c = (c_1, ..., c_n) \in (\mathbb{F}_q)^n$

**Output:**

- $f \in \mathbb{F}_{\text{ext}}[x]$, *such that* $f$ *is a monic polynomial of minimal degree generating c.*

**Step1(Initialization):** $f(x) := 1$;$g(x) := 1$;$\text{dif} := 1$;$L_N := 0$;$d_{n+1} := 1$;$N := 0$;$L_n := 0$;

**Step2** *If* $(N = n)$ *then stop else* $d_{N+1} := \sum_{i=0}^{L_N} f_i c_{N+i-L_N+1}$ *end if*

**Step3** *If* $(d_{N+1} = 0)$ *then* $\text{dif} := \text{dif} + 1$;
*else if* $(d_{N+1} \neq 0$ *and* $2L_N > N)$ *then*
$r := \max\{L_N, N+1-L_N\}$;

$f(x) := x^{r-L_N} f(x) - \frac{d_{N+1}}{d_{n+1}} x^{-dif-L_n+r} g(x)$;
$\text{dif} := \text{dif} + 1$;

*else* $(d_{N+1} \neq 0$ *and* $2L_N \leq N)$
$r := \max\{L_N, N+1-L_N\};$
$h(x) := f(x);$
$f(x) := x^{r-L_N} f(x) - \frac{d_{N+1}}{d_{n+1}} x^{-dif-L_n+r} g(x);$
$g(x) := h(x);$
$d_{n+1} := d_{N+1};$
$\mathrm{dif} := 1;$
$L_n := L_N;$
$L_N := N+1-L_N;$
*end if*

**Step4** *N:=N+1;goto Step2*

*Remark:* This algorithm directly follows the construction in the proof of Theorem 2.11. In every step we have a polynomial $f(x)$ of degree $L_N(c)$ which generates the sequence up to the $N$th element and a polynomial $g(x)$ of degree $L_n(c)$, which generates the sequence up to the $n$th element, where $n$ denotes the biggest integer smaller than $N$, for which $L_{n+1}(c) > L_n(c)$. If $f(x)$ also generates $(c_1, ...c_{N+1})$ (if-case) we are done and we only have to increase dif which denotes $N - n$. If $f(x)$ doesn't generate $(c_1, ..., c_{N+1})$, there are two possibilities:

- $L_{N+1} = L_N$. This is the case, if and only if $L_N \geq N - L_N + 1$ or equivalently if $2L_N > N$. In this case we do not have a change in the sequence $(L_k)_{k\in\mathbb{N}}$, so we only have to update $f(x)$ and increase dif.
- $L_{N+1} > L_N$. This is the case if and only if $2L_N \leq N$. In this case we have a change in the sequence $(L_k)_{k\in\mathbb{N}}$ and we have to update both $g(x)$ and $f(x)$, as well as $d_{n+1}$, $L_n$ and $L_N$. The difference between $N$ and $n$ (dif) gets 1.

The correctness of this algorithm follows from Theorem 2.10 and the proof of Theorem 2.11, if we can ensure that the initialization is correct.
The algorithm finds a non-zero polynomial of minimal degree, generating the given sequence.

Suppose that $m \leq t$ errors occurred. We can use this algorithm for finding a polynomial of minimal degree generating $(s_1, ..., s_{2t})$. Suppose this polynomial, lets call it $f(x)$, doesn't generate the whole sequence $(s_1, ...)$. Let $i$ be the index at which $f$ fails to generate $(s_1, ..., s_i)$ for the first time. Then we have that $L_i(c) \geq \max\{L_{i-1}(c), i - L_{i-1}(c)\} = \max\{\deg(f(x)), i - \deg(f(x))\}$. But since $i > 2t$ and $\deg(f(x)) \leq \deg(\tilde{\sigma}(x)) = m \leq t$ we get that $L_i > t$, which can't be true, since $L_\infty(c) = \deg(\tilde{\sigma}(x)) = m \leq t$ and $(L_k)$ is non-decreasing in $k$. Moreover the polynomial $f(x)$ must vanish on all error positions $\alpha^{i_j}$, since it is an element of the error-locator ideal, but since it has minimal degree, it can't have any other roots. So we get the following theorem.

**Theorem 2.12** *If less or equal to $t$ errors occurred during transmission, the polynomial of degree $L_{2t}(c)$ which generates the finite sequence $(s_1, ...s_{2t})$ found by the Berlekamp-Massey-algorithm is a constant multiple of the error-locator-polynomial $\tilde{\sigma}(x)$. Its roots are $\alpha^{i_j}$, where $i_j$, $1 \leq j \leq m$, are the error positions.*

In the next example we use the Berlekamp-Massey algorithm for decoding the BCH-code we constructed in example (2.1).

**Example 2.2 (Continuation of Example 2.1)** *Let $C$ be the binary 2-error correcting $[15,7,d]$ BCH-code with $\delta = 5$. The generator-polynomial is $g(x) = (x^4+x+1)(x^4+x^3+x^2+x+1) = x^8+x^7+x^6+x^4+1$. Suppose we want to transmit the message $m(x) = x^6+x^4+x^2+1$. We encode it by multiplying the message with g(x) and get $c(x) := m(x)g(x) = x^{14}+x^{13}+x^{10}+x^9+x^8+x^7+x^6+x^2+1$. During the transmission two errors occur at position $x^{14}$ and $x^9$. So we receive $r(x) = x^{13}+x^{10}+x^8+x^7+x^6+x^2+1$. The first step in decoding consists of calculating the syndroms $(s_1,...,s_{2t})$:*

$$
\begin{aligned}
s_1 &= r(\alpha) && = \alpha+1 \\
s_2 &= r(\alpha^2) = (r(\alpha))^2 && = \alpha^2+1 \\
s_3 &= r(\alpha^3) && = 0 \\
s_4 &= r(\alpha^4) && = \alpha
\end{aligned}
$$

*So we start the Berlekamp-Massey-algorithm with $s = (\alpha+1, \alpha^2+1, 0, \alpha)$. The algorithm leads to:*

| $N$ | $d_{N+1}$ | $f$ | $g$ |
|---|---|---|---|
| 0 | $\alpha+1$ | $x+\alpha+1$ | $1$ |
| 1 | $0$ | $x+\alpha+1$ | $1$ |
| 2 | $\alpha^3+\alpha^2+\alpha+1$ | $x^2+(\alpha+1)x+\alpha^2+1$ | $x+\alpha+1$ |
| 3 | $0$ | $x^2+(\alpha+1)x+\alpha^2+1$ | $x+\alpha+1$ |

*So the polynomial $f(x) = x^2+(\alpha+1)x+\alpha^2+1$ generates the sequence $(s_1,s_2,s_3,s_4)$. Its roots are $\alpha^9$ and $\alpha^{14}$. So we know that two errors occurred and that $m(x) := \frac{r(x)-x^{14}-x^9}{g(x)}$ was the most probable message.*

# Chapter 3

# Basics from the theory of Gröbner bases

In this chapter we summarize the definitions and theorems of Gröbner bases we will need for encoding and decoding 2-dimensional cyclic codes. Proofs and details can be found in [Win96].

Gröbner bases are a powerful tool in ideal theory. Since multi-dimensional cyclic codes will turn out to be ideals, it is natural to apply the theory of Gröbner bases also to coding theory.

In fact the first occasion, where the theory of Gröbner bases was applied to coding theory was the problem of encoding 2-dimensional cyclic codes: In [Sak81] Sakata developed an algorithm for constructing a Gröbner basis (he doesn't call it so) for encoding 2-dimensional cyclic codes without knowledge of Buchberger's algorithm [Buch65].

Throughout this chapter let $K[X] = K[x_1, ..., x_n]$ denote the polynomial ring in $n$ indeterminates over the field $K$. For a subset $F$ of $K[X]$ we denote by $< F >$ or ideal$(F)$ the ideal generated by $F$ in $K[X]$. By $[X]$ we denote the monoid (under multiplication) of all power products $x_1^{i_1}...x_n^{i_n}$.

For Gröbner basis computation we have to introduce an ordering of the power products.

**Definition 3.1** *Let $<$ be an ordering on $[X]$ that is compatible with the monoid structure:*

1. $1 = x_1^0...x_n^0 < t$ *for all* $t \in [X] \setminus \{1\}$
2. $s < t \Longrightarrow su < tu$ *for all* $s,t,u \in [X]$.

*We call such an ordering $<$ on $[X]$ an* admissible ordering.

The most important class of admissible orderings we will need in decoding are the graduated lexicographical orderings.

Let $w$ be a weight function $w : \{1, ..., n\} \to \mathbb{R}^+$. Let s,t $\in [X]$ with $s = x_1^{s_1}...x_n^{s_n}$ and $t = x_1^{t_1}...x_n^{t_n}$

$$s <_{glex,w} t \Longleftrightarrow \sum_{k=1}^{n} w(k)s_k < \sum_{k=1}^{n} w(k)t_k \ \vee \ (\sum_{k=1}^{n} w(k)s_k = \sum_{k=1}^{n} w(k)t_k \ \wedge \ s <_{lex} t).$$

Here $<_{lex}$ means the lexicographical ordering:

$$x_1^{s_1}...x_n^{s_n} <_{lex} x_1^{t_1}...x_n^{t_n} \iff \exists k \in \{1,...n\} : (\forall l < k : s_l = t_l) \wedge s_k < t_k$$

**Definition 3.2** *For $f \neq 0 \in K[x]$, $F \subseteq K[X]$, $s \in [X]$ and $<$ an admissible ordering on $[X]$, we define :*
*By* $\text{coeff}(f,s)$ *we denote the coefficient of $s$ in $f$.*
$S(f) := \{t \in X | \text{coeff}(f,t) \neq 0\}$ *(Support of $f$)*
$\text{lpp}(f) := \max_<\{t \in S(f)\}$ *(leading power product of f)*
$\text{lc}(f) := \text{coeff}(f, \text{lpp}(f))$ *(leading coefficient of f)*
$\text{in}(f) := \text{coeff}(f, \text{lpp}(f))\text{lpp}(f)$ *(initial of f or leading term of f)*
$\text{lpp}(F) := \{\text{lpp}(f) | f \in F \setminus \{0\}\}$
$\text{lc}(F) := \{\text{lc}(f) | f \in F \setminus \{0\}\}$
$\text{in}(F) := \{\text{in}(f) | f \in F \setminus \{0\}\}$
$\Delta(F) := \{t \in X | \nexists f \in F : \text{lpp}(f) | t\}$ *(the delta set of F)*

Now we are able to define a polynomial reduction relation.

**Definition 3.3** *Let $f$,$g$,$h \in K[X]$, $F \subseteq K[X]$ and $<$ be an admissible ordering on $[X]$. We say that $g$* reduces to $h$ w.r.t $f$ *($g \rightarrow_f h$) iff there are $s$,$t \in [X]$ such that* $\text{coeff}(g,s) = c \neq 0$, $s = \text{lpp}(f)t$ *and*

$$h = g - \frac{c}{\text{lc}(f)} \cdot t \cdot f$$

*We say that $g$* reduces to $h$ w.r.t $F$ *($g \rightarrow_F h$) iff there is $f \in F$ such that $g \rightarrow_f h$.*
*For $i \in \mathbb{N}_0$ we say that $g \rightarrow_F^i h$ iff there exist $g_i$ such that $g = g_0 \rightarrow g_1 \rightarrow ... \rightarrow g_i = h$, which means that $g$* can be reduced to $h$ in $i$ steps.
*We say that $g \rightarrow_F^* h$, iff $g$ can be reduced to $h$ in a finite number of steps.*

**Definition 3.4** *Let $I$ be an ideal in $K[X]$. Let $F$ be a set of polynomials such that $F \subseteq I$. $F$ is called a* Gröbner basis *for I, iff for all $g \in K[X]$*

$$g \in I \Leftrightarrow g \rightarrow_F^* 0.$$

**Definition 3.5** *A Gröbner basis $G$ in $K[X]$ is called* minimal *iff* $\text{lpp}(g) \nmid \text{lpp}(h)$ *for all $g$,$h \in G$ with $g \neq h$.*

**Definition 3.6** *Let $F$ be a subset of $K[X]$. $p \in \Delta(F)$ is called an* inner corner *of $\Delta(F)$ iff $p$ is a maximal element of $\Delta(F)$ according to the partial ordering $|$,i.e*

$$\nexists p' \neq p \in \Delta(F) : p | p'.$$

*$s \in [X] \setminus \Delta(F)$ is called an* exterior corner *of $\Delta(F)$ iff $s$ is a minimal element of $[X] \setminus \Delta(F)$ with respect to the partial ordering $|$,i.e*

$$\nexists s' \neq s \in [X] \setminus \Delta(F) : s' | s.$$

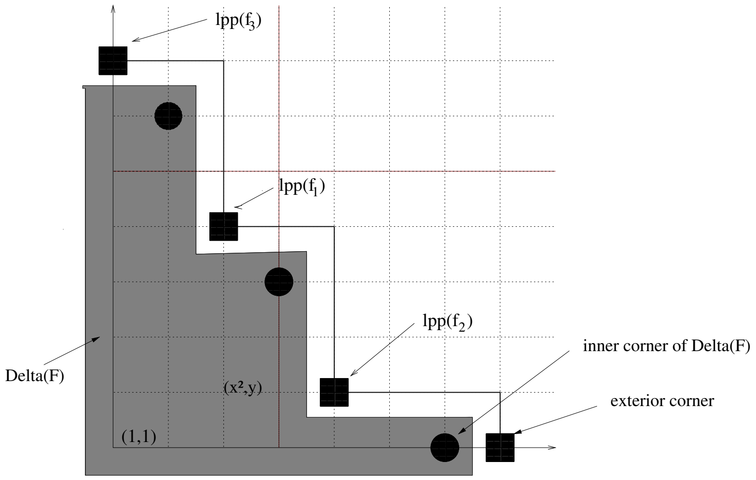



Figure 3.1: Visualization of the inner/ exterior corners of $\Delta(F)$ in Example 3.1

*The set of all exterior corners of* $\Delta(F)$ *is denoted by* $\mathrm{Ext}(\Delta(F))$. *The set of all interior corners of* $\Delta(F)$ *is denoted by* $\mathrm{Int}(\Delta(F))$.
*The set* $[X] \setminus \Delta(F)$ *is called the* exterior of $\Delta(F)$.
*If* $F$ *is a Gröbner basis,* $\Delta(F)$ *is also called the* set of the irreducible power products *and every* $x \in \Delta(F)$ *is called an* irreducible power product. *Every* $x \in [X]$ *which is not irreducible is called* reducible.

**Example 3.1** *Let* $F = \{f_1 = x^2y^4 + x^3y + xy, f_2 = x^4y + xy^2, f_3 = y^7 + x^6 + x^2y, f_4 = x^7 + x\}$. *We take as the admissible order* $\leq$

$$x^{i_1}y^{j_1} \leq x^{i_2}y^{j_2} \iff i_1 + j_1 < i_2 + j_2 \ \vee \ (i_1 + j_1 = i_2 + j_2 \ \wedge \ i_1 \leq i_2).$$

*We draw the power products according to their exponents* ($x^iy^j \simeq (i,j)$) *and visualize* $\Delta(F)$ *and the inner and the exterior corners of* $\Delta(F)$ *in Figure 3.1.*

**Definition 3.7** *Let* $I$ *be an ideal in* $K[X]$. *Let* $\overline{K}$ *denote the algebraic closure of* $K$.
*By* V(I) *we denote the* set of common zeros of I, *i.e*

$$V(I) = \{p \in \overline{K}^n | f(p) = 0 \quad \forall f \in I\}.$$

**Theorem 3.1** *Let* $G$ *be a Gröbner basis for the ideal* $I$. *Then* $|V(I)| < \infty$ *if and only if for every* $i$, $1 \leq i \leq n$ *there is a polynomial* $g_i \in G$, *such that* $\mathrm{lpp}(g_i)$ *is a pure power of* $x_i$.
*Moreover if* $|V(I)| < \infty$, $|V(I)| \leq |\Delta(G)|$ *and the numbers of common zeros of I (counted with multiplicity over* $\overline{K}$*) is equal to* $|\Delta(G)|$.

# Chapter 4

# 2-dimensional cyclic codes

**Definition 4.1** *A* 2-dimensional code C *over the field* $\mathbb{F}_q$ *of domain* $(N_1, N_2)$ *is a subset of all* $(N_1 \times N_2)$*-matrices over* $\mathbb{F}_q$*. C is called linear, iff C forms a linear subspace of the space of all* $(N_1 \times N_2)$*-matrices over* $\mathbb{F}_q$*. A linear 2-dimensional code C is called* cyclic*, iff for each code word* $c \in C$ *all words obtained by cyclic permutation of the columns or the rows of C are also code words.*

## 4.1 The structure of 2-dimensional cyclic codes

For 2-dimensional cyclic codes one identifies the $(N_1 \times N_2)$-matrices over $\mathbb{F}_q$ with the elements of $W_{N_1N_2} := \mathbb{F}_q[x, y]_{/<x^{N_1}-1, y^{N_2}-1>}$ by the isomorphism

$$\begin{aligned}
\tau\colon\ (\mathbb{F}_q)_{N_1}^{N_2} &\rightarrow W_{N_1N_2} &(4.1)\\
(c_{ij})_{0\leq i\leq N_1-1, 0\leq j\leq N_2-1} &\mapsto \sum_{i=0}^{N_1-1}\sum_{j=0}^{N_2-1} c_{ij}x^iy^j. &(4.2)
\end{aligned}$$

*Remark:* We are aware that the elements of $W_{N_1N_2}$ are equivalent classes. For convenience we calculate with representatives (the polynomials which contain only powers products $x^iy^j$ with $i < N_1$,$j < N_2$) and skip the equivalence class brackets.
Using this isomorphism a cyclic shift of the columns of a matrix corresponds to a multiplication by $y$ in $W_{N_1N_2}$ and a cyclic shift of the rows to a multiplication by $x$ in $W_{N_1N_2}$. So we get the following theorem.

**Theorem 4.1** *A 2-dimensional linear code* $C \subseteq W_{N_1N_2}$ *is cyclic if and only if C is an ideal in* $W_{N_1N_2}$*.*

Obviously every one-dimensional cyclic code can be regarded as a 2-dimensional cyclic code (just take $N_1 = 1$). On the other hand some 2-dimensional codes cyclic codes are 'equivalent' to a 1-dimensional cyclic code.

**Definition 4.2** *Two codes C and C' are called equivalent, iff there is a permutation $\pi$ of $\{0, ..., n-1\}$ such that:*

$$c = (c_0, ..., c_{n-1}) \in C \Longleftrightarrow (c_{\pi(0)}, ..., c_{\pi(n-1)}) \in C'$$

**Theorem 4.2** *Let $C$ be a 2-dimensional cyclic code of domain $(N_1, N_2)$ with $\gcd(N_1, N_2) = 1$. $C$ is equivalent to a 1-dimensional cyclic code of length $N_1N_2$. A word $c'$ of the one-dimensional cyclic code is obtain by*

$$c'_k = c_{k \bmod N_1, k \bmod N_2}, \qquad \text{for all } 0 \le k < N_1N_2.$$

**Example 4.1** *Let $C$ be a 2-dimensional cyclic code of domain $(3, 7)$. Let's take a $c \in C$ and number all the elements of the array.*

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| 15 | 16 | 17 | 18 | 19 | 20 | 21 |

*The corresponding word of the one-dimensional code word is:*

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 9 | 17 | 4 | 12 | 20 | 7 | 8 | 16 | 3 | 11 | 19 | 6 | 14 | 15 | 2 | 10 | 18 | 5 | 13 | 21 |

*If we shift this code word cyclically, we get:*

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21 | 1 | 9 | 17 | 4 | 12 | 20 | 7 | 8 | 16 | 3 | 11 | 19 | 6 | 14 | 15 | 2 | 10 | 18 | 5 | 13 |

*If we transform this back to 2 dimensions, we get:*

| | | | | | | |
|---|---|---|---|---|---|---|
| 21 | 15 | 16 | 17 | 18 | 19 | 20 |
| 7 | 1 | 2 | 3 | 4 | 5 | 6 |
| 14 | 8 | 9 | 10 | 11 | 12 | 4 |

*This new code word turns out to be the original 2-dimensional code word with rows and columns cycled once. On the other hand cycling the columns of the 2-dimensional code word once is equivalent to cycling the 1- dimensional code word 15 times and cycling the rows 1 time is equivalent to cycling the one-dimensional code 7 times.*

When we had one-dimensional cyclic codes, we were able to define them by the common roots of all code words. This was very convenient, because it gave us a bound on the minimum distance and it led to the definition of BCH-codes. In two dimensions, such a definition is possible, if $N_1$ and $N_2$ are relatively prime to the characteristic of the field $\mathbb{F}_q$ ([Imai77]).

**Theorem 4.3** *Let $C$ be a 2-dimensional cyclic code of domain $(N_1, N_2)$ over the field $\mathbb{F}_{p^r}$ with gcd$(N_1, p)$ =gcd$(N_2, p) = 1$. Let $\alpha$ be a primitive $N_1 - th$ and $\beta$ a primitive $N_2 - th$ root of unity in an extension field $\tilde{\mathbb{K}}$ of $\mathbb{F}_{p^r}$. Then $C$ can be uniquely represented by the set of its* common positions of zeros, *which is defined as*

$$Z(C) := \{(i, j) | c(\alpha^i, \beta^j) = 0, 0 \le i \le N_1 - 1, 0 \le j \le N_2 - 1 \text{ for all } c(x, y) \in C\}.$$

If we have that $(i, j)$ is an element of the set of common positions of zeros for a code C, $(ip^r \bmod N_1, jp^r \bmod N_2)$ must also be a common position of zeros of C. A cyclotomic coset for $(i, j)$ contains all elements $w$ such that $w = (ip^{kr} \bmod N_1, jp^{kr} \bmod N_2)$ for a $k \in \mathbb{N}$.

**Example 4.2** *Let $N_1 = 3$ and $N_2 = 7$ let $\mathbb{F}_q$ be $\mathbb{F}_2$. The cyclotomic cosets of the common positions of zeros are:*

$$\{(0,0)\}$$

$$\{(1,0),(2,0)\}$$

$$\{(0,1),(0,2),(0,4)\}$$

$$\{(1,1),(2,2),(1,4),(2,1),(1,2),(2,4)\}$$

$$\{(0,3),(0,6),(0,5)\}$$

$$\{(1,3),(2,6),(1,5),(2,3),(1,6),(2,5)\}$$

A one-dimensional BCH-code of designed minimum distance $\delta$ is defined as the set of all polynomials vanishing on $\delta - 1$ subsequent powers of $\alpha$. In order to make a similar definition for the two dimensional case, we have to define an ordering on the roots.

**Definition 4.3** *Let $c$, $d$ be two elements of $\mathbb{N}_0^2$.*

$$c = (c_1, c_2) <_g d = (d_1, d_2) :\Longleftrightarrow c_1 + c_2 < d_1 + d_2 \vee (c_1 + c_2 = d_1 + d_2 \wedge c_2 < v_2)$$

*Let $p_1 = x^i y^j$, $p_2 = x^{i'} y^{j'}$ be elements in $[X]$ (= monoid of all power products in 2 variables).*

$$p_1 <_g p_2 :\Longleftrightarrow (i, j) <_g (i', j')$$

Corresponding to this ordering we can define the successor of a given vector.

**Definition 4.4** *Let $c = (c_1, c_2)$ be an element of $\mathbb{N}_0^2$.*

$$\operatorname{succ}(c) := \begin{cases} (c_1 - 1, c_2 + 1) & \text{if } c_1 > 0 \\ (c_2 + 1, 0) & \text{if } c_1 = 0 \end{cases}$$

*is called the* successor of $c$.

**Definition 4.5** *A 2-dimensional code of domain $(N_1, N_2)$ over $\mathbb{F}_{p^r}$ with $gcd(N_1, p) = gcd(N_2, p) = 1$ is called a* 2-dimensional BCH-code of parameter $\delta \in \mathbb{N}_0^2$ *iff its set of common positions of zeros contains the set $\Gamma^\delta := \{v \in \mathbb{N}_0^2 | v <_g \delta\}$.*

## 4.2 Encoding of 2-dimensional cyclic codes

Since 2-dimensional cyclic codes are ideals in $W_{N_1 N_2}$, we can represent a 2-dimensional cyclic code by a finite basis $\{f_1, .., f_s\}$. $c(x,y)$ is a code word if and only if it can be represented as a linear combination (with polynomial coefficients) of the basis elements, i.e.

$$c(x,y) \equiv_{<x^{N_1}-1, y^{N_2}-1>} \sum_{i=1}^{s} f_i(x,y) a_i(x,y) \qquad \text{for some } a_i(x,y) \in \mathbb{F}_{q^r}(x,y).$$

This is equivalent to the fact that $c(x,y)$ can be represented by a linear combination of $\{f_1, ..., f_s, x^{N_1}-1, y^{N_2}-1\}$. Let I denote the ideal generated by the set $\{f_1, ..., f_s, x^{N_1} - 1, y^{N_2} - 1\}$. If we know a Gröbner basis (denoted by $G$) for $I$, we can easily determine whether a given word is a code word and get a way for encoding 2-dimensional cyclic codes systematically. This encoding procedure can be found in [Lit98].

**Theorem 4.4** *Let $C$ be a 2-dimensional cyclic code of domain $(N_1, N_2)$ over the field $\mathbb{F}_{p^r}$. Let $(f_1, ..., f_s)$ be a basis for $C$ in $W_{N_1,N_2}$. Let $G$ be a Gröbner basis for the ideal $I =< f_1, ..., f_s, x^{N_1} - 1, y^{N_2} - 1 >$.*

$$c(x,y) \in C \Longleftrightarrow c(x,y) \to_G^* \underline{0}$$

**Theorem 4.5** *The following algorithm gives a systematic encoder for $C$.*

**Input:** *m(x,y), a linear combination (with coefficients in $\mathbb{F}_{p^r}$) of reducible power products $(x^i y^j)$ with respect to $G$, with $i < N_1$ and $j < N_2$. (the message)*

**Output:** *E(m(x,y)) (the encoded message)*

*Let $r(x,y)$ such that $m(x,y) \to_G^* \underline{r}(x,y)$.*
*$E(m(x,y)) := m(x,y) - r(x,y)$.*

*Moreover we get that the parameters of the code $C$ are:*

$$\begin{aligned} n &= N_1 N_2 \\ k &= n - \#\textit{irreducible power products with respect to } G = n - |Z(C)|. \end{aligned}$$

*Proof.* Let $m(x,y)$ be a linear combination of the reducible power products $(x^i y^j)$ with respect to $G$ with $i < N_1$ and $j < N_2$. $E(m(x,y))$ is a code word since $E(m(x,y)) \to_G^* \underline{0}$. Moreover the coefficients of the reducible power products are not changed by the encoding procedure, since $r(x,y)$ is a linear combination of the irreducible power products. The encoding procedure is obviously bijective, therefore the dimension of the code $C$, is the number of reducible power products with $i < N_1$, $j < N_2$ which is $N_1 N_2 - \#$irreducible power products with respect to $G$.
Since the ideal $I$ contains the functions $x^{N_1} - 1 = \prod_{i=0}^{N_1-1}(x - \alpha^i)$ and $y^{N_2} - 1 = \prod_{i=0}^{N_2-1}(x - \beta^i)$, its set of common zeros $(V(I))$ is a subset of $\{(\alpha^i, \beta^j) | 0 \le i < N_1, 0 \le j < N_2\}$ and each common zero of $I$ can only be of multiplicity one.
So we get from Theorem (3.1) that $|\Delta(G)| = |V(I)|$.

But

$$\begin{aligned}|V(I)| &= |\{(\alpha^i,\beta^j)|0\le i<N_1, 0\le j<N_2, f_1(\alpha^i,\beta^j)=\ldots=f_s(\alpha^i,\beta^j)=0\}|\\ &= |\{(i,j)|0\le i<N_1, 0\le j<N_2, c(\alpha^i,\beta^j)=0 \text{ for all } c\in C\}| = |Z(C)|.\end{aligned}$$

□

Another algorithm for encoding cyclic codes is to determine a generator matrix.
We have that $c(x,y)\in C \Leftrightarrow c(\alpha^i,\beta^j)=0$ for all $(i,j)\in Z(C)$.
We have that

$$\begin{aligned}c\in C &\Leftrightarrow \forall(i,j)\in Z(C): c(\alpha^i,\beta^j)=0\\ &\Leftrightarrow \forall(i,j)\in Z(C) \sum_{\substack{0\le k<N_1\\0\le l<N_2}} c_{k,l}(\alpha^i)^k(\beta^j)^l=0\\ &\Leftrightarrow \forall(i,j)\in Z(C) \sum_{\substack{0\le k<N_1\\0\le l<N_2}} c_{k,l}(\alpha^k)^i(\beta^l)^j=0\\ &\Leftrightarrow \forall(i,j)\in Z(C) \sum_{\substack{0\le k<N_1\\0\le l<N_2}} c_{k,l}f_{(i,j)}(\alpha^k,\beta^l)=0 \qquad \text{where } f_{(i,j)}(x,y):=x^iy^j \end{aligned} \tag{4.3}$$

If we order the set of functions $\{f_{(i,j)}|(i,j)\in Z(C)\}$ and the the set of 'points' P:=$\{(\alpha^k,\beta^l)|0\le k<N_1, 0\le j<N_2\}$ and denote by $(f_l)$ and by $(P_l)$ the resulting sequences we can rewrite the last equation as

$$H\cdot c=0$$

where H denote the matrix $(f_i(P_j))_{1\le i\le|Z(C)|,1\le j\le N_1N_2}$ and $c$ denotes the vector $(c_{i,j})$ ordered according to the order of $P$. So we can easily construct a parity check matrix for $C$. We get a generator matrix for $C$ by finding a basis for the nullspace of $H$.
Let $(x_1,...,x_k)$ be a basis for the nullspace of $H$.

$$G=\begin{pmatrix}x_1\\x_2\\\cdots\\x_k\end{pmatrix}$$

is a generator matrix for $C$.

## 4.3 Decoding of 2-dimensional cyclic codes

In this section we try to develop a decoding algorithm for 2-dimensional BCH-codes. The following algorithm was developed by Sakata in [Sak91] and is based on his extension of the Berlekamp-Massey algorithm to 2 dimensions (called the BMS- or simply Sakata's algorithm) [Sak86].
The problem with 2-dimensional cyclic codes is, that we do not have good bounds for their minimum distance and that no 2-dimensional cyclic code with really good performance has been found until now.

Nevertheless the decoding algorithm (although inefficient for decoding two-dimensional cyclic codes) turns out to be very useful, because its main part (Sakata's algorithm) can be used efficiently for decoding certain algebraic geometric codes.

Throughout this section, let $C$ be a 2-dimensional cyclic code of domain $(N_1, N_2)$ over $\mathbb{F}_{p^r}$, such that $\gcd(N_1, p) = \gcd(N_2, p) = 1$. Let $\alpha$ be a primitive $N_1 - th$ and $\beta$ be a primitive $N_2 - th$ root of unity in an extension field $\tilde{\mathbb{K}}$ of $\mathbb{F}_{p^r}$. Let $Z(C)$ denote the set of common positions of zeros of $C$. To explain the idea of Sakata's algorithm, we assume that the code $C$ is a 2-dimensional BCH-code of parameter $\delta$.

Assume that we wanted to transmit the code word $c(x,y)$, but we received $r(x,y) := c(x,y) + e(x,y)$, where $e(x,y) := \sum_{(i,j)\in E} e_{i,j} x^i y^j$ is the error word and $E$ is the set of the error positions.
As in the one-dimensional case we define:

$$s_{i,j} := e(\alpha^i, \beta^j)$$

the syndroms of the received word $r(x)$. The syndroms can be calculated from the received word, if $(i,j)$ is a common position of zeros of the code $C$, since then $s_{i,j} = r(\alpha^i, \beta^j)$, otherwise the syndroms are called unknown at the beginning.

*Notation:* For convenience we want to be able to interpret a monomial $x^i y^j$ as the corresponding vector $(i,j)$. Of course these two representations of a monomial are isomorphic. For two vectors $u = (i,j), v = (i', j')$ we define

$$u \leq v :\Longleftrightarrow x^i y^j | x^{i'} y^{j'}.$$

For this definition we get isomorphic definitions to the definitions of Chapter 3, which turn out to be more convenient for the notation in Sakata's algorithm.

**Definition 4.6** *For $f \neq 0 \in \tilde{\mathbb{K}}[X]$, $F \subseteq \tilde{\mathbb{K}}[X]$, $s \in [X]$ and $<_T$ an admissible ordering on $[X]$ and respectively on $\mathbb{N}_0^2$, we define :*
*By* $\mathrm{coeff}(f,s)$ *we denote the coefficient of $s$ in $f$.*
$S(f) := \{(i,j) | \mathrm{coeff}(f, x^i y^j) \neq 0\}$ *(Support of f)*
$\mathrm{lpp}(f) := \max_{<_T}\{t \in S(f)\}$ *(leading power product of f)*
$\mathrm{lc}(f) := \mathrm{coeff}(f, x^{\mathrm{lpp}(f)_1} y^{\mathrm{lpp}(f)_2})$ *(leading coefficient of f)*
$\mathrm{lpp}(F) := \{\mathrm{lpp}(f) | f \in F \setminus \{0\}\}$
$\mathrm{lc}(F) := \{\mathrm{lc}(f) | f \in F \setminus \{0\}\}$
$\Delta(F) := \{(i,j) | \nexists f \in F : \mathrm{lpp}(f) \leq (i,j)\}$ *(the delta set of F)*

*Let $F$ be a subset of $K[X]$.*
*$p \in \Delta(F)$ is called an* inner corner *of $\Delta(F)$ iff $p$ is a maximal element of $\Delta(F)$ according to the partial ordering $\leq$, i.e*

$$\nexists p' \neq p \in \Delta(F) : p \leq p'.$$

*$s \in {\mathbb{N}_0}^2 \backslash \Delta(F)$ is called an* exterior corner *of $\Delta(F)$ iff $s$ is a minimal element of ${\mathbb{N}_0}^2 \backslash \Delta(F)$ with respect to the partial ordering $\leq$, i.e*

$$\nexists s' \neq s \in \Delta(F) : s' \leq s.$$

*The set of all exterior corners of* $\Delta(F)$ *is denoted by* $Ext(\Delta(F))$. *The set* ${\mathbb{N}_0}^2 \setminus \Delta(F)$ *is called the* exterior of $\Delta(F)$.

Let r,q be two elements of $\mathbb{N}_0^2$.

$$\Sigma_r^q := \{m \in \mathbb{N}_0^2 | r \leq m \wedge m <_g q\}$$

Just as in the one dimensional case we define:

**Definition 4.7** *The ideal (in* $\tilde{\mathbb{K}}[x, y]$*) of all polynomials which vanish on all* $(\alpha^i, \beta^j)$ *for* $(i, j) \in E$ *is called the* error-locator ideal.

**Definition 4.8** *A polynomial* $f(x, y) := \sum_{(i,j)\in S_f} f_{(i,j)} x^i y^j \in \tilde{\mathbb{K}}[x, y]$ generates the infinite array $(c_{i,j})_{0\leq i, 0\leq j}$, *iff*

$$\sum_{(i,j)\in S_f} f_{(i,j)} c_{l-\mathrm{lpp}(f)+(i,j)} = 0 \qquad \textit{for all } l \geq \mathrm{lpp}(f).$$

*The set of all polynomials which generate the infinite array c is called* VALPOL(c).

Just as in the one-dimensional case we get the following theorem.

**Theorem 4.6** *The error-locator ideal is exactly the ideal of all polynomials generating the infinite array* $(s_{i,j})_{0\leq i,0\leq j}$.

*Proof.* The proof of the Theorem 2.9 can be rewritten for the 2-dimensional case, without big changes. □

We don't know all the syndroms, but only $s_{i,j}$ for $(i, j) \in Z(C)$. Since we restricted ourselves to 2-dimensional BCH-codes, $\Sigma_0^\delta \subseteq Z(C)$, where $\delta$ is the parameter of the BCH-code.
In the one-dimensional case, we got that, if we have a polynomial of minimal degree, which generates the finite sequence $(s_1, ...s_d)$ where $d \geq 2t$, this polynomial automatically generates the infinite sequence $(s_1, ...)$ and we can find the error-locator polynomial.
In the two-dimensional case we will get that, if we have a 'minimal' set of polynomials, which generates the infinite array $(s_{i,j})_{0\leq i,0\leq j}$ up to a value $b$ in $\mathbb{N}_0^2$ where b is 'big enough', every polynomial of this set automatically generates the infinite array $(s_{i,j})_{0\leq i,0\leq j}$ and the set forms a Gröbner basis of the error-locator ideal. To make this more precise, we need some definitions.

**Definition 4.9** *A polynomial* $f$ generates the infinite array $(u_{i,j})_{0<i,0<j}$ up to the value $b$, *iff*

$$\sum_{(i,j)\in S_f} f_{(i,j)} u_{l-\mathrm{lpp}(f)+(i,j)} = 0 \qquad \textit{for all } b >_t l \geq \mathrm{lpp}(f)$$

*For* $k \in \mathbb{N}_0^2$ *and for* $f \in \tilde{\mathbb{K}}[x, y]$ *we define:*

$$d_{f,u,k} := \begin{cases} \sum_{(i,j)\in S_f} f_{(i,j)} u_{k-\mathrm{lpp}(f)+(i,j)} & \textit{for } k \geq \mathrm{lpp}(f) \\ 0 & \textit{otherwise} \end{cases}$$

*If* $d_{f,u,k} = 0$ *then* $f$ *is said to be* valid at $k$.
*The set of all polynomials* $f$ *generating the infinite array up to the value* $b$*, which is equivalent to the fact that* $f$ *is valid at all* $k <_g b$*, is denoted by* VALPOL(u,b).

*Remark*: If $u$ and $k$ are clear from the context, we will write $d_f$ instead of $d_{f,u,k}$.
*Remark*: VALPOL($u$,$b$) is in general no ideal in $\tilde{\mathbb{K}}[x,y]$. We can only assure that $f \in \text{VALPOL}(u,b) \Rightarrow p \cdot f \in \text{VALPOL}(u,b)$ for any power product $p$. But the sum of two elements of VALPOL$(u,b)$ need not be an element of VALPOL$(u,b)$.

**Definition 4.10** *A set of polynomials $F$* generates the infinite array $(u_{i,j})_{0\leq i,0\leq j}$ up to the value $b$ *iff every element of $F$ generates the infinite array up to the value $b$.*
*A set of polynomials $F$, which generates the infinite array $(u_{i,j})_{0\leq i,0\leq j}$ up to the value $b$ is called* minimal*, iff*

1.
$$\underset{g\,\in\,\text{VALPOL}(u,b)}{\forall} \text{lpp}(g) \notin \Delta(F)$$

2.
$$\underset{f,f'\,\in\,F,f\neq f'}{\forall} \text{lpp}(f) \not\leq \text{lpp}(f')$$

*Remark:* Please notice the similarity between the last definition and the definition (3.4). If the set $F$ also generates the infinite array $u$, condition 1) means that $F$ is a Gröbner basis for the ideal VALPOL$(u)$ which is identical to the error-locator ideal.
Proof:
Suppose there is a polynomial $g \in \text{VALPOL}(u)$ such that $g \to_F^* \underline{h} \neq 0$, where $h$ can't be reduced any more with respect to $F$. Then $h \in \text{VALPOL}(u)$, but $\text{lpp}(h) \in \Delta(F)$ because otherwise $h$ would be reducible with respect to $F$. This contradicts the definition of a minimal set, so $F$ is a Gröbner basis for VALPOL$(u)$.
Condition 2) means that $F$ is a minimal Gröbner basis (compare definition 3.5).
Sakata's algorithm calculates a minimal set of polynomials generating a given infinite array up to the given value $b$.

## 4.4 Sakata's algorithm

Sakata's algorithm [Sak86] provides a fast way for finding a minimal set of polynomials generating a given infinite array up to the given value $b$. In the previous section we defined a total ordering $<_g$ which turns out to be appropriate for 2-dimensional BCH-codes.
But Sakata's algorithm works for every total ordering (let's denote it by $<_T$), which is admissible.

In this section succ$(b)$ denotes the successor of $b \in \mathbb{N}_0^2$ with respect to $<_T$.
For convenience we introduce the following notation. Let $r \in \mathbb{N}_0^2$.

$$x^r := x^{r_1}y^{r_2}.$$

Let $u$, $v$ be two elements in $\mathbb{N}_0^2$

$$u + v := (u_1 + v_1, u_2 + v_2).$$

We can now describe the input and output of Sakata's algorithm.

**Input:**

- infinite array $(u_{i,j})_{i\geq 0, j\geq 0}$
- value $b$ in $N_0^2$
- $\leq_T$ an admissible total ordering

**Output:**

- $F$: minimal set of polynomials generating the array $u$ up to the value $b$, with respect to $\leq_T$.

The BM- algorithm calculates a polynomial of minimal degree generating the infinite sequence $(s_i)_{i\geq 1}$ by calculating step by step an $f_i$ of minimal degree which generates $(s_1, ..., s_i)$. $f_{i+1}$ is found by updating $f_i$ with the help of a polynomial g(x), which failed to generate the sequence up to j+1 but succeeded to generate s up to j and ensures, that the degree of $f_{i+1}$ must be greater than $j - \deg(g)$.
We will use the same principle in the two-dimensional case. Just as in the case of the BM-algorithm, we will need two main theorems. The first theorem (which corresponds to Theorem 2.10) will give us a way of determining power products which can't be the leading power product of any polynomial in $\mathrm{VALPOL}(u, \mathrm{succ}(b))$.

**Theorem 4.7** *If some polynomial $f(x, y)$ with $\mathrm{lpp}(f) = l$ generates the array $u$ up to the value $b$, but fails to generate the array $u$ up to the value $\mathrm{succ}(b)$ then the leading power product $l'$ of any non-zero polynomial generating the array $u$ up to the value $\mathrm{succ}(b)$ must satisfy*

$$l' \not\leq b - l.$$

*Proof.* The proof can be rewritten without big changes from the proof of Theorem 2.10.□

We define (motivated by 4.7):

**Definition 4.11** *Let $g$ be a polynomial which generates the array $u$ up to the value $b_g$, but fails to generate $u$ up to the value $\mathrm{succ}(b_g)$.*

$$\mathrm{span}(g) := b_g - \mathrm{lpp}(g)$$

*Let $G$ be set of polynomials, such that for every $g \in G$, $g$ fails to generate $u$ up to the value $\mathrm{succ}(b_g)$ but generates $u$ up to the value $b_g$. Then*

$$\mathrm{Span}(G) := \{p \in \mathbb{N}_0^2 | p \leq \mathrm{span}(g) \textit{ for any } g \in G\}.$$

*Remark:* $\mathrm{Span}(G)$ is the set of all $p \in \mathbb{N}_0^2$ for which we can be sure (because of the set $G$ and Theorem 4.7) that $p$ can't be a leading power product of any $f \in \mathrm{VALPOL}(u)$.
The second theorem tells us that the bound in Theorem 4.7 can be reached and its proof will give us an algorithm for updating a minimal polynomial set which generates the infinite array $u$ up to the value $b$, to a minimal polynomial set which generates $u$ up to the value $\mathrm{succ}(b)$.

As in the one-dimensional case, where a polynomial $g(x)$ ensures that a polynomial $f_{i+1}(x)$ generating $(s_1, ..., s_{i+1})$ can't have a 'leading power product' (=degree) less than $j - \deg(g) + 1$, we need in the 2-dimensional case a set of polynomials $G$ (called the witness set), which ensures that for any $f$ in VALPOL$(u, \mathrm{succ}(b))$, $\mathrm{lpp}(f)$ can't be smaller or equal (with respect to $<$) to $b_g - \mathrm{lpp}(g)$ for a $g \in G$.

**Definition 4.12** *Let $F$ be a set of polynomials generating the infinite array $u$ up to the value $b$. A set of polynomials $G$ is called a* witness set *for the set $\Delta(F)$ iff*

$$\mathrm{Span}(G) = \Delta(F).$$

*Remark:* If we have a witness set $G$ for $\Delta(F)$, we can be sure that $F$ fulfills condition(1) in the definition of a minimal polynomial set.
In order to have a witness set for $\Delta(F)$, we only need a witness for every inner corner of $\Delta(F)$. So it is enough that for every inner corner $p$ of $\Delta(F)$ there is a $g \in G$ such that $\mathrm{span}(g) = p$.
On the other hand $\Delta(F)$ can be uniquely determined by its exterior corners. So we only need a set of polynomials $f \in F$ such that for every exterior corner $p$ of $\Delta(F)$ there is a $f \in F$ such that $\mathrm{lpp}(f) = p$. If we take only polynomials which are necessary to fulfill this condition, the polynomial set $F$ also fulfills condition(2) of the definition of a minimal polynomial set.
The algorithm of Sakata consists in every step (updating from $b$ to $\mathrm{succ}(b)$) of the following substeps:

**Algorithm 4.1 (InductionStep(b,u,F,G))**

**Input:**

- $b \in {\mathbb{N}_0}^2$
- *$u$...infinite array*
- *Minimal set of polynomials $F$, which generates the array up to the value $b$.*
- *$G$, a witness set for $F$ (for every inner corner $p$ of $\Delta(F)$ we have a $g \in G$ such that* $\mathrm{span}(g) = p$*).*

**Output:**

- *Set of Polynomials $F'$, which generates the array $u$ up to the parameter $succ(b)$.*
- *Gnext : A witness set for $\Delta(F')$*

1. *Initialization: $N = \emptyset$,$Gnext := G$,$F' := \emptyset$;*
2. *For every $f$ in $F$ calculate $d_f := \sum_{(i,j) \in S_f} f_{(i,j)} u_{b - \mathrm{lpp}(f) + (i,j)}$.*
3. *If $d_f \neq 0$ ($f$ doesn't generate $u$ up to* $\mathrm{succ}(b)$ *and might be used as a witness)*
   $Gnext = Gnext \cup \{f\}$
   $N = N \cup \{f\}$

4. *calculate* Span*(Gnext)*

5. *for every inner corner p of Span(Gnext) keep a g in Gnext with* span*(g)=p.*

6. *F':=GenerateNewF(Span(Gnext),F,G,N,b);*

7. *return([F',Gnext])*

**Example 4.3** *Let*

$$u = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha^3 & \alpha^4 & \alpha^2 & ?_{2,4} & \alpha^6 & ?_{2,6} & ?_{2,7} \\ \alpha^6 & \alpha^5 & \alpha & ?_{3,4} & \alpha^4 & ?_{3,6} & ?_{3,7} \\ \alpha^2 & ?_{4,2} & ?_{4,3} & ?_{4,4} & ?_{4,5} & ?_{4,6} & ?_{4,7} \\ \alpha^5 & \alpha & \alpha^3 & ?_{5,4} & \alpha^2 & ?_{5,6} & ?_{5,7} \\ \alpha & ?_{6,2} & ?_{6,3} & ?_{6,4} & ?_{6,5} & ?_{6,6} & ?_{6,7} \\ \alpha^4 & ?_{7,2} & ?_{7,3} & ?_{7,4} & ?_{7,5} & ?_{7,6} & ?_{7,7} \end{bmatrix}$$

*where $\alpha$ is a root of $x^3 + x^2 + 1$ in an extension field of $\mathbb{F}_2$. As an admissible order we choose $\leq_g$. Assume that we know a minimal set of polynomials $F = \{y + 1, x + \alpha^3\}$ generating the array $u$ up to the value $(1,1)$ and a witness set $G = \{1\}$ for $F$.*
*As a first step we calculate $d_f$ for every $f \in F$. We get that*

$$d_{y+1} = \alpha \neq 0 \qquad d_{x+\alpha^3} = \alpha \neq 0$$

*So both functions can be used us a witness.*

$$\mathrm{span}(y+1) = (1,1) - \mathrm{lpp}(y+1) = (1,1) - (0,1) = (1,0)$$

$$\mathrm{span}(x+\alpha^3) = (1,1) - \mathrm{lpp}(x+\alpha^3) = (0,1)$$

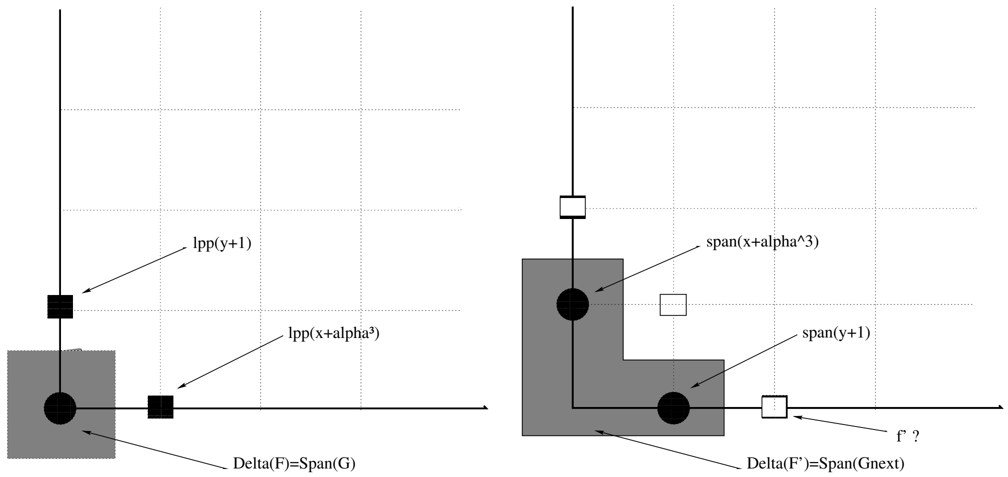
Gnext $= \{1, y+1, x+\alpha^3\}$, Span(Gnext) $= \{(0,0),(1,0),(0,1)\}$ *The interior corners of* Span(Gnext) *are* $(0,1),(1,0)$ *and we keep* $y+1$ *and* $x+\alpha^3$ *as witness for these two inner corners in the set* Gnext. *We visualize this updating procedure in Figure 4.1.*
*The exterior corners of* Span(Gnext) *are* $(2,0)$,$(0,2)$ *and* $(1,1)$. *The next task is to find polynomial* $\{f_1, f_2, f_3\}$,*with* $\mathrm{lpp}(f_1) = (2,0)$, $\mathrm{lpp}(f_2) = (1,1)$ *and* $\mathrm{lpp}(f_3) = (0,2)$ *which generate* $u$ *up to the value* $\mathrm{succ}(1,1) = (0,2)$.

If we can find a way to calculate step 6) of the algorithm, we are done, because Gnext (condition (1)) and the fact that we only take exterior corners of a delta set (condition (2)), verify that F' is minimal.

**Algorithm 4.2 (GenerateNewF(Span(Gnext),F,G,N,b))**

**Input:**

- Span(Gnext) *set of power products* $p$, *with* $\mathrm{lpp}(f) \not\leq p$ *for any* $f$ *in* $\mathrm{VALPOL}(u, \mathrm{succ}(b))$
- *Minimal set of polynomials* $F$, *which generates the array up to the value* $b$.




$\Delta(F)$,lpp$(F)$, and Span$(G)$ Span(Gnext),lpp$(F')$,$\Delta(F')$

Figure 4.1: Visualization of Example 4.3

- *$G$, a witness set for $F$ (for every inner corner $p$ of $\Delta(F)$ we have a $g \in G$ such that* $\mathrm{span}(g) = p$*).*
- *$N$ subset of $F$ with $d_{f,u,b} \neq 0$.*
- $b \in \mathbb{N}_0^2$

**Output:**

- *Set of Polynomials $F'$, which generates the array $u$ up to the parameter $succ(b)$.*

*for each* $s \in \mathrm{Ext}(\mathrm{Span}(\mathrm{Gnext}))$ *do*
  *if there exists* $f \in F \setminus N$ *with* $\mathrm{lpp}(f) = s$ *then*
    $f'_s := f$
  *elsif* $s \not\leq b$
    *Find* $f \in N$ *with* $\mathrm{lpp}(f) \leq s$
    $f'_s := x^{s-\mathrm{lpp}(f)} f$
  *else*
    *Find* $f \in N$ *with* $\mathrm{lpp}(f) \leq s$
    *Find* $g \in G$ *with* $\mathrm{span}(g) \geq b - s$
    *Let* $q = s - \mathrm{lpp}(f)$
    *Let* $p = \mathrm{span}(g) - b + s$
    $f'_s := x^q f(x) - \frac{d_f}{d_g} x^p g(x)$
  *end if*
*end do*
*return(*$F' := \{f'_s | s \in \mathrm{Ext}(\mathrm{Span}(\mathrm{Gnext}))\}$*)*

**Theorem 4.8** *Let $b \in \mathbb{N}_0^2$, let $u$ be a infinite array. The Algorithm 4.1 calculates for a given $F$ (minimal polynomial set generating $u$ up to $b$) and $G$ (witness set for $\Delta(F)$) a minimal polynomial set $F'$ which generates $u$ up to* $\mathrm{succ}(b)$ *and a witness set* Gnext *for $\Delta(F)$.*

*Proof.* It it enough to prove that the Algorithm 4.2 succeeds in finding for each $s$ a polynomial $f'_s$ with $\mathrm{lpp}(f'_s) = s$ and $f'_s$ generates $u$ up to the value $\mathrm{succ}(b)$.
The computations of $f'_s$ break down to three mutually exclusive cases:

**Case1:** $\exists f(x) \in F \setminus N : \mathrm{lpp}(f) = s$

**Case2:** $(\nexists f(x) \in F \setminus N : \mathrm{lpp}(f) = s) \wedge s \nleq b$

**Case3:** $(\nexists f(x) \in F \setminus N$: $\mathrm{lpp}(f) = s) \wedge s \leq b$

In **Case1** we find an $f(x)$ in the original $F$, which has leading power product $s$ and generates the array up to the value $\mathrm{succ}(b)$ (since $d_f = 0$), so we can take this $f$ and are done.
In both cases **Case2** and **Case3**, we must find a polynomial $f \in N$ with $\mathrm{lpp}(f) \leq s$. We show first that this is always possible. The point $s$ is an exterior corner of the delta set Span(Gnext). This delta set obviously (since $G \subseteq$ Gnext) contains $\mathrm{Span}(G) = \Delta(F)$. First we assume that $s$ is not an exterior corner of the delta set $\Delta(F)$. The point $s$ is nonetheless a point on the exterior of $\Delta(F)$, and so there exists a polynomial $f(x) \in F$, whose leading power product $\mathrm{lpp}(f)$ is an exterior corner of $\Delta(F)$, satisfying $\mathrm{lpp}(f) \leq s$. The fact that $s$ is an exterior corner of the new delta set Span(Gnext) thus implies that $\mathrm{lpp}(f)$ must be in the interior of Span(Gnext). Since $\mathrm{lpp}(f) \in$ Span(Gnext), $f$ can't generate $u$ up to $\mathrm{succ}(b)$. But then $f \in N$ because $f \in F$ and $d_f \neq 0$.
Now we assume that $s$ is an exterior corner of $\Delta(F)$. Thus there is a polynomial $f(x) \in F$ with $\mathrm{lpp}(f) = s$. Moreover $f \in N$ because otherwise Case1 would apply. Therefore it is possible in Case1 and Case2 to find a polynomial $f \in N$ with $\mathrm{lpp}(f) \leq s$.
**Case2:** Clearly $\mathrm{lpp}(f'_s) = s$. Since $s \nleq b$, $d_{f'_s}$ is by definition 0. So we only have to show that $f'_s$ generates $u$ up to $b$. But $f'_s$ is a monomial multiple of an element of $\mathrm{VALPOL}(u, b)$, therefore it is also an element in $\mathrm{VALPOL}(u, b)$. Therefore $f'_s \in \mathrm{VALPOL}(u, b)$.
**Case3:** First we have to show that it is possible to find $g \in G$ with $\mathrm{span}(g) \geq b - s$. Suppose this is not possible. This means that $b-s$ is in the exterior of $\Delta(F)$ and therefore, $b-s \geq p$ for some exterior corner of $\Delta(F)$. There exists a polynomial $f' \in F$ with leading power product $\mathrm{lpp}(f') = p$. Let $f$ be a polynomial with $f \in N$ and $\mathrm{lpp}(f) \leq s$. Then we have $\mathrm{lpp}(f') + \mathrm{lpp}(f) \leq p + s \leq b$. So $\mathrm{lpp}(f') \leq b - \mathrm{lpp}(f)$ and since $f$ doesn't generate u up to $\mathrm{succ}(b)$ ,$f'$ doesn't generate the array up to the value $\mathrm{succ}(b)$ either. Thus $f' \in N$ and hence $\mathrm{span}(f') = b - p \in$ Span(Gnext). But now $s \leq b - p$ implies that $s$ is in the interior of Span($Gnext$), which means that we have arrived at a contradiction. Thus it is always possible to find $g \in G$ with $\mathrm{span}(g) \geq b - s$.
Next we have to verify that $f'_s$ fulfills $\mathrm{lpp}(f'_s) = s$ and $f'_s \in \mathrm{VALPOL}(u, \mathrm{succ}(b))$. $x^{s-\mathrm{lpp}(f)}f$ obviously has leading power product $s$.
Note that $g$ doesn't generate the array up to the value $b$. Therefore $\mathrm{span}(g) <_T b - \mathrm{lpp}(g)$ and $\mathrm{lpp}(g) + \mathrm{span}(g) <_T b$. So

$$x^{\mathrm{span}(g)-b+s}g(x)$$

has leading power product $\mathrm{span}(g) - b + s + \mathrm{lpp}(g) <_T s$. Therefore $\mathrm{lpp}(f'_s) = s$.

To see that $f'_s$ generates $u$ up to $\mathrm{succ}(b)$, let $l$ satisfy $l \geq \mathrm{lpp}(f'_s) = s$ and $l \leq_T b$.

$$\sum_{(i,j)\in S_{f'_s}} f'_{s_{i,j}} u_{l-s+(i,j)} := T_1 - \frac{d_f}{d_g} T_2$$

where the two terms $T_1$ and $T_2$ are obtained from the expansion of the coefficients of $f'_s$:

$$\begin{aligned}
T_1 &= \sum_{p-s+\mathrm{lpp}(f)\in S(f)} f_{p-s+\mathrm{lpp}(f)} u_{p+l-s} \\
&= \sum_{t\in S(f)} f_t u_{t+l-\mathrm{lpp}(f)} \\
&= \begin{cases} 0 & \text{for } l <_T b \\ d_f & \text{for } l = b \end{cases} \\
T_2 &= \sum_{p-\mathrm{span}(g)+b-s\in S(g)} g_{p-\mathrm{span}(g)+b-s} u_{p+l-s} \\
&= \sum_{t\in S(g)} g_t u_{t+l+\mathrm{span}(g)-b} \\
&= \begin{cases} 0 & \text{for } l-b+\mathrm{span}(g)+\mathrm{lpp}(g) <_T \mathrm{span}(g)+\mathrm{lpp}(g) \\ d_g & \text{for } l-b=0 \end{cases} \\
&= \begin{cases} 0 & \text{for } l <_T b \\ d_g & \text{for } l = b \end{cases}
\end{aligned}$$

So we get that:

$$T_1 - \frac{d_f}{d_g} T_2 = \begin{cases} 0 & \text{for } l \leq b \\ d_f - \frac{d_f}{d_g} d_g = 0 & \text{for } l = b \end{cases}$$

Therefore $f'_s \in \mathrm{VALPOL}(u, \mathrm{succ}(b))$. □

So we have proved the 'induction step' in Sakata's algorithm. We only have to write a loop around 4.1 (InductionStep) to get Sakata's algorithm.

**Algorithm 4.3 (Sakata's algorithm(u,l,$<_T$))**

**Input:**

- *infinite array* $(u_{i,j})_{i\leq 0, j\leq 0}$
- *value* $l$ *in* $N_0^2$
- $<_T$ *an admissible total ordering*

**Output:**

- $F$*: minimal set of polynomials generating the array* $u$ *up to the value* $l$*, with respect to* $\leq_T$*.*

*begin*
    *Initialize b:=(0,0);* $F := \{1\}$*;* $G := \emptyset$
    *while* $b <_T l$ *do*
        $[F', \text{Gnext}] := \text{InductionStep}(u, F, G, b)$*;*
        $F := F'$*;* $G := \text{Gnext}$*;* $b := \text{succ}(b)$*;*
    *enddo*
*end*

## 4.5 Sakata's algorithm for 2-dimensional BCH-codes

In order to apply Sakata's algorithm to 2-dimensional BCH-codes, we have to choose an admissible ordering and determine the parameter $l$.
As an admissible ordering we choose the ordering $<_g$ which is defined in Definition (4.3), it is the total order we defined for BCH-codes.
We want to choose the parameter $l$, such that VALPOL$(s, l)$ is equal to the error locator ideal, under the assumption that at most $t$ errors occurred, i.e we want to be able to correct $t$ errors.

**Definition 4.13** *A* delta set *is a subset of* ${\mathbb{N}_0}^2$ *for which the following condition holds: For every element p of the delta set, every* $q \leq p$ *is also an element of the delta set. By* $\Delta_t$ *we denote the set of all delta sets containing at most t elements.*

**Theorem 4.9** *If* $l >_g \max_{<_g}\{p + \text{ext} | p \in G, \text{ext} \in \text{Ext}(G), G \in \Delta_t\}$ *and at most t errors occurred, then the error locator ideal is equal to* $VALPOL(s, l)$*.*

*Proof.* We know that the number of common zeros of the error-locator ideal is equal to $m \leq t$, where $m$ denotes the number of errors which occurred during the transmission.
Let $G$ be a minimal Gröbner basis of the error-locator ideal. Since $V(G) = \{(\alpha^i, \beta^j) | (i, j) \in E\}$ and all roots are only of multiplicity one, we get from from Theorem 3.1 that $|\Delta(G)| = m \leq t$ and therefore $\Delta(G) \in \Delta_t$.
Suppose there is a polynomial $f$ in VALPOL$(s, b)$, which is not an element of the error locator ideal. Since VALPOL$(s)$ is equal to the error locator ideal, there must be a smallest (with respect to $<_g$) $k \in {\mathbb{N}_0}^2 \geq_g l$ such that $d_{f,s,k} \neq 0$.
But then span$(f)$=$k$-lpp$(f) \in \Delta(\text{VALPOL}(s)) = \Delta(G)$.
Therefore there is a $p \in \Delta(G)$, such that $k$=$p$+lpp$(f)$.
There are two possibilities. Either lpp$(f) \in \text{Ext}(\Delta(\text{G}))$ or lpp$(f) \in \Delta(G)$.
In each of these cases we can find a $q$ in Ext$(\Delta(\text{G}))$, such that lpp$(f) \leq_g q$. Therefore $k \leq_g p + q \in \{p + \text{ext} | p \in G, \text{ext} \in \text{Ext}(G), G \in \Delta_t\}$.
But this contradicts the assumption that $k \geq_g l >_g \max_{<_g}\{p + \text{ext} | p \in G, \text{ext} \in \text{Ext}(G), G \in \Delta_t\}$.

Therefore every $f \in$ VALPOL$(s, b)$ is an element of the error-locator ideal. $\square$

**Example 4.4** *Suppose we want to construct a 2-error correcting 2-dimensional BCH-code.*

$$\Delta_2 = \{\{(0,0)\}, \{(0,0),(1,0)\}, \{(0,0),(0,1)\}\}.$$

*The corresponding sets of exterior corners are:*
$\{(1,0),(0,1)\}$,$\{(2,0),(0,1)\}$ *and* $\{(0,2),(1,0)\}$.
$\max_{<_g}\{(1,0),(0,1),(2,0),(0,1),(3,0),(1,1),(0,2),(1,0),(1,1),(0,3)\} = (0,3)$.
*So we get that for* $l \geq_T (4,0)$, *we constructed an at least 2-error correcting code. So a 2 dimensional BCH-code with* $\delta = (4,0)$ *can correct two errors.*
*Suppose we want to construct a 3-error correcting 2-dimensional BCH-code.*
$\Delta_3 = \Delta_2 \cup \{\{(0,0),(1,0),(0,1)\}, \{(0,0),(1,0),(2,0)\}, \{(0,0),(0,1),(0,2)\}\}$.
$\max_{\leq_g} = (0,2)+(0,3) = (0,5)$. *So a 2-dimensional BCH-code is at least 3-error correcting if* $\delta \geq (6,0)$.

For general 2-dimensional cyclic codes, there are no efficient decoding algorithms. One could use Sakata's algorithm as far as the syndroms are known, and then guess for every cyclotomic coset, for which the syndroms are unknown, a syndrom. We can be sure that the guess was correct, if we reach the parameter $l$ and $|\Delta(F)| = m \leq t$ and we find $m$ common roots of $F$. If we get that $\Delta(F_i) > t$ for an intermediate step, we can discard the guess and take a new guess.
In general we don't know an intelligent way of guessing, which ensures that we don't need much try and error, until we find the correct syndroms.
Another disadvantage of 2-dimensional cyclic codes is, that the bound for $\delta$ and therefore also for the minimum distance of the code is not really good.
Both disadvantages can be handled by another family of codes, which we treat in the next section.

# Chapter 5

# Decoding 1-point algebraic geometric codes

## 5.1 Basic notations

In this chapter we only give a very short introduction to AG-codes. For details on AG-codes we refer to [Dei99, HLP98, Pre98, Duu93, FR93], for details on the algebraic geometric background we refer to [Mor91, Ful69]. We assume some familiarity with the basics of algebraic geometry.
Throughout this chapter let $\mathbb{F}_q$ denote the finite field with $q$ elements. Let $\mathcal{C}$ be a non-singular curve of genus $g$ in the 2-dimensional projective space $\overline{\mathbb{P}^2} := \mathbb{P}^2(\overline{\mathbb{F}_q})$. For two homogeneous polynomials $f$ and $g$ and $P \in \overline{\mathbb{P}}$ we let $m_P(F \cap g)$ denote the intersection multiplicity of $f$ and $g$ at $P$ and $m_P(f \cap \mathcal{C})$ the intersection multiplicity of $f$ and an homogeneous polynomial defining the curve $\mathcal{C}$ at $P$.

**Definition 5.1** *A* divisor *of $\mathcal{C}$ is an element of the free Abelian group generated by the points of $\mathcal{C}$. The group operation will be written additively. Thus a divisor of $\mathcal{C}$ is of the form*

$$D = \sum_{P \in \mathcal{C}} n_P(D)P,$$

*with $n_P(D) \in \mathbb{Z}$, and only finitely many $n_P \neq 0$.*
*The* degree *of a divisor $D$ is defined as*

$$\deg(D) := \sum_{P \in \mathcal{C}} n_P(D).$$

*The* support *of a divisor is defined as*

$$\operatorname{supp}(D) := \{P \in \mathcal{C} | n_P(D) \neq 0\}.$$

*A divisor is called* $\mathbb{F}_q$-rational *iff the points of the support of the divisor have coordinates in $\mathbb{F}_q$.*

Now we link the notion of a divisor with that of rational functions.

**Definition 5.2** *Let the irreducible curve $\mathcal{C}$ be defined by the homogeneous polynomial $h$. Let $f$ be a homogeneous polynomial $\in \mathbb{F}_q[x, y, z]$, which is relatively prime to $h$. Then the* zero divisor *or the* intersection cycle *of $f$, denoted by $(f)$ is defined as:*

$$(f) := \sum_{P \in \overline{\mathbb{P}^2}} m_P(\mathcal{C} \cap f)P.$$

*Let $\phi = \frac{f}{g}$ be a rational function over $\mathcal{C}$. Let $f$ and $g$ be relatively prime to $h$. The* order of $\phi$ at P, *denoted by $\nu_P(\phi)$, is defined as*

$$\nu_P(\phi) := m_P(\mathcal{C} \cap f) - m_P(\mathcal{C} \cap g).$$

*$P$ is called a* pole *iff $\nu_P(\phi) < 0$, and a* zero *iff $\nu_P(\phi) > 0$.*
*The* divisor of poles and zeros *or the* principal divisor, *denoted by $(\phi)$,is defined as*

$$(\phi) := (f) - (g) = \sum_{P \in \overline{\mathbb{P}^2}} (m_P(\mathcal{C} \cap f) - m_P(\mathcal{C} \cap g))P = \sum_{P \in \overline{\mathbb{P}^2}} \nu_P(\phi).$$

Let $D$ be a divisor of $\mathcal{C}$. $D \geq 0 :\Longleftrightarrow n_P(D) \geq 0$ for all points $P$ of $\mathcal{C}$.

**Definition 5.3** *Let $D$ be a divisor of $\mathcal{C}$. The space $\mathcal{L}(D)$ is defined as*

$$\mathcal{L}(D) := \{\phi | (\phi) + D \geq 0\} \cup 0.$$

**Theorem 5.1** *(Riemann-Roch Theorem).Let $D$ be a divisor on a curve $\mathcal{C}$ of genus $g$. Then*

$$\dim(\mathcal{L}(D)) = \deg(D) + 1 - g + i(D)$$

*where $i(D)$ is the* index of speciality *of $D$. $i(D)$ satisfies the following properties*

$$0 \leq i(D) \leq g \qquad \textit{and } i(D) = 0 \quad \textit{for } \deg(D) > 2g - 2.$$

From the Riemann-Roch Theorem we get the following theorem, which will turn out to be useful.

**Theorem 5.2** *Let $Q$,$P_1$,...,$P_n$ be $n+1$ different $\mathbb{F}_q$- rational points on a curve $\mathcal{C}$ of genus $g$ and let $m$ be a positive integer.*
*The function*

$$\begin{aligned} \textit{EV: } \mathcal{L}(mQ) &\rightarrow (\mathbb{F}_q)^n && (5.1)\\ f' &\mapsto (f'(P_1), ..., f'(P_n)) && (5.2) \end{aligned}$$

*is surjective, if $n \leq m - 2g + 1$.*

*Proof.* Since all points $(P_1, \ldots, P_n)$ have coordinates in $\mathbb{F}_q$, EV takes only values in $(\mathbb{F}_q)^n$. Further on it is easy to see that EV is linear.
From linear algebra we know that $\dim(\mathcal{L}(mQ)) = \dim(\mathrm{Image(EV)}) + \dim(\ker(\mathrm{EV}))$.
Let $D := mQ$. From the Riemann-Roch theorem we get that $\dim(\mathcal{L}(D)) = \deg(D) + 1 - g + i(D)$, but since $\deg(D) = m \geq n + 2g - 1 > 2g - 2$, $i(D) = 0$.
Since $\mathcal{L}(mQ - \sum_{i=1}^n P_n)$ is the set of all rational functions $\phi \in \mathcal{L}(D)$ which have at least a simple zero at all points $P_i$ for $i = 1, \ldots, n$,

$$\mathcal{L}(mQ - \sum_{i=1}^{n} P_n) = \ker(\mathrm{EV}).$$

Let $D' := mQ - \sum_{i=1}^n P_n$. From the Riemann-Roch theorem we get that

$$\dim(\mathcal{L}(D')) = m - n + 1 - g + i(D').$$

Since $\deg(D') = m - n \geq 2g - 1 > 2g - 2$, $\dim(\mathcal{L}(D')) = m - n + 1 - g$.
So we finally get that $\dim(\mathrm{Image(EV)}) = m + 1 - g - (m - n + 1 - g) = n$. Therefore EV is surjective (it is even bijective). $\square$

**Definition 5.4** *Let $D := \sum_{i=1}^n P_i$ be an $\mathbb{F}_q$-rational divisor and $G = mQ$, where $Q \in \mathcal{C}$ has coordinates in $\mathbb{F}_q$ and is no element of* $\mathrm{supp}(D)$. *Then the code $C_\Omega(mQ, D)$ defined by*

$$c = (c_1, ..., c_n) \in C_\Omega(mQ, D) :\Longleftrightarrow \sum_{j=1}^{n} \phi(P_j)c_j = 0 \qquad \textit{for all } \phi \in \mathcal{L}(mQ)$$

*is called a* 1-point AG-code *or a* 1-point primary Goppa code.

From the Riemann-Roch Theorem we get the following bounds on the parameters of $C_\Omega(mQ, D)$.

**Theorem 5.3** *Let $C_\Omega(mQ, D)$ be a code as defined in the last definition. $C_\Omega(mQ, D)$ is a linear code with length $n$, $k = n - m + g - 1 - i(mQ)$, $d \geq d_\Gamma := m - 2g + 2$.*

*Remark:* The bound $d_\Gamma$ is called the Goppa-designed minimum distance. If $m < 4g - 2$ this bound can be improved by the Feng Rao bound $d_{FR}$. For this bound (which we will define later) the inequality $d_{FR} \geq d_\Gamma$ holds. We will present an algorithm (based on Sakata's algorithm), which decodes 1-dimensional AG-codes up to the Feng-Rao bound.
We will need the following theorem which gives us a bound on the number of roots of a function in $\mathcal{L}(mQ)$.

**Theorem 5.4** *Let $\phi$ be a non-zero rational function in $\mathcal{L}(mQ)$. $\phi$ has at most $m$ zeros.*

*Proof.* Since any rational function $\phi$ has the same number of poles and zeros ( counted with multiplicity over the algebraic closure ), that is

$$\sum_{P \in \overline{\mathbb{P}^2}} \nu_P(\phi) = 0.$$

and $\phi$ has only at most a pole (the point $Q$) of order at most $m$, it can't have more than $m$ zeros. $\square$.

**Definition 5.5** *Let $Q$ be a point of the smooth curve $\mathcal{C}$ of genus $g$ over $\mathbb{F}_q$. An integer $n \geq 0$ is called a* pole number *or a* nongap *of $Q$, iff $\mathcal{L}(nQ) \neq \mathcal{L}((n-1)Q)$. Otherwise $n$ is called a* gap number *or a* Weierstrass gap *of $Q$.*

**Theorem 5.5** *(Weierstrass Gap Theorem) Let $P$ be a point on a smooth curve of genus $g > 0$. Then there are exactly $g$ gaps numbers $i_1 < ... < i_g$ of $P$.*

The set of all nongaps of $Q$ forms an additive sub-semigroup of $\mathbb{N}_0$. Let $a_1, ..., a_N$ be a minimal set of generators for this semigroup and let (for $j = 1, .., N$) $\Psi_j$ be a rational function with pole order $a_j$ at $Q$ and no other poles.
To any vector $\alpha = (\alpha_1, ..., \alpha_n)$ we define the corresponding function $f_\alpha := \Psi_j^{\alpha_j}$. This function only has a pole at the point $Q$. The order of this pole is denoted by $O(\alpha)$ and is equal to

$$O(\alpha) = \sum_{j=1}^{N} \alpha_j a_j.$$

The functions $f_\alpha$ for which $O(\alpha) \leq m$ span the space $\mathcal{L}(mQ)$, but they are not independent, since

$$O(\alpha) = O(\alpha') \Rightarrow f_\alpha = c f_{\alpha'} + g \qquad \text{where } c \in \mathbb{F}_q \wedge O_Q(g) < O(\alpha). \tag{5.3}$$

We will now show how to apply Sakata's algorithm to the problem of decoding 1-point AG-codes. For convenience (and to keep notation simpler) we restrict ourselves to the case, where the semigroup of nongaps of $Q$ can be generated by two co-prime integers $a$ and $b$. We will call the function of order $a$ at $Q$ and no other poles $\Psi$ and the function of order $b$ at $Q$ and no other poles $\Phi$.
This restriction to only 2 generators is only necessary, to be able to apply Sakata's algorithm in 2 dimensions. If one wants to decode 1-point AG-codes with k generators one needs Sakata's algorithm for k dimensions, which is a direct generalisation of the 2-dimensional algorithm. For details we refer to [Sak90, SJMJH95a].
If we restrict ourselves to 1-point codes, where the semigroup of nongaps can be generated by two integers, we still cover whole families of algebraic geometric codes, which turned out to have a very good performance. Among them there are famous codes like the Hermitian codes. For more examples of codes which can be decoded by Sakata's algorithm in 2 dimensions we refer to [SJMJH95b].

**Example 5.1** *(Hermitian codes) Let $n$ be a prime power. The curve $H_n := x^{n+1} + y^n z + yz^n$ over the field $\mathbb{F}_q = \mathbb{F}_{n^2}$ is called the Hermitian curve.*
*$H_n$ is smooth, $g = \frac{n(n-1)}{2}$. There are $n^3$ affine $\mathbb{F}_{n^2}$-rational points and a single $\mathbb{F}_{n^2}$-rational point $P_\infty = (0:1:0)$ at infinity.*
*The semigroup of nongaps at the point $P_\infty = (0:1:0)$ is spanned by the two co-prime number $n$ and $n+1$. A rational function of pole order $n$ is the rational function $\Psi := \frac{x}{z}$, a rational function of pole order $n+1$ is the rational function $\Phi := \frac{y}{z}$. The codes obtained by the Hermitian curve and the point $P_\infty$ are called the Hermitian codes.*
*We now define a Hermitian code over $H_2$ and show how encoding can be done using a generator matrix.*

*We define the 1-point AG code* $C_\Omega(mP_\infty, D)$ *over the smooth curve* $H_2 = x^3 + y^2z + yz^2$ *over the field* $\mathbb{F}_4$. $D$ *denotes the sum of all* $2^3 = 8$ *affine* $\mathbb{F}_4$*-rational points of* $H_2$.
*So* $D = \sum_{i=1}^{8} P_i$ *where*

$$\begin{array}{lll} P_1 = (0:0:1) & P_2 = (0:1:1) & P_3 = (1:\alpha:1) \\ P_4 = (1:\alpha^2:1) & P_5 = (\alpha:\alpha:1) & P_6 = (\alpha:\alpha^2:1) \\ P_7 = (\alpha^2:\alpha:1) & P_8 = (\alpha^2:\alpha^2:1) & \end{array}$$

$P_\infty := (0:1:0)$ *and we set* $m := 4$.
*The genus of the curve is 1. The parameters of this code are:*

$$n = 8, k = 8 - 4 - 1 + 1 = 4, d \geq d_\Gamma = 4 - 2 + 2 = 4.$$

*Hence we constructed an at least 1-error correcting code.*
*The semigroup of nongaps is generated by the two co-prime integers 2 and 3. A rational function with a pole of order 2 at the point* $P_\infty$ *is* $\Phi := \frac{x}{z}$, *a rational function with a pole of order* 3 *at the point* $P_\infty$ *is* $\Psi := \frac{y}{z}$. *The space* $\mathcal{L}(4P_\infty)$ *is spanned by the rational functions* $f_i$, $i = 1, ..., 4$.

| | | | | |
|---|---|---|---|---|
| *function* | $1$ | $\Psi$ | $\Phi$ | $\Phi^2$ |
| *order of pole at* $P_\infty$ | 0 | 3 | 2 | 4 |

$H = (f_i(P_j))_{i=1,...,4;j=1,...,8}$ *is a parity check matrix for* $C_\Omega(4P_\infty, D)$.

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}$$

*We get a generator matrix* $G$ *by finding a basis for the nullspace of* $H$.

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & \alpha \\ 0 & 1 & 0 & 1 & 0 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

*Now we suppose that we want to transmit the message* $(1, \alpha, 1, 1)$. *We encode the message by multiplying it by* $G$.

$$G \cdot (1, \alpha, 1, 1) = \left[1,\ \alpha,\ 1,\ \alpha,\ 1,\ \alpha,\ 0,\ \alpha^2\right].$$

## 5.2 The Basic Algorithm

Let $C_\Omega(mQ, D)$ be a 1-point AG code for which the semigroup of nongaps at $Q$ is generated by two co-prime integers $a$ and $b$.
Let $r \in \mathbb{F}_q^n$ be the received word, $e \in \mathbb{F}_q^n$ the error vector. Let $e_{\text{num}}$ be the number of errors which occurred during the transmission, $E = \{i_1, ..., i_{e_{\text{num}}}\}$ the error positions and let $t := \lfloor \frac{d_\Gamma - 1}{2} \rfloor$.

Just as in the previous section we define a syndrom array.
The values

$$s_{i,j} := \sum_{k=1}^{n} e_k \Psi^i(P_k)\Phi^j(P_k)$$

are called the syndroms of the received word $r$. The syndroms can be calculated from the received word, if $O((i,j)) = ai + bj \leq m$ since then $s_{i,j} := \sum_{k=1}^{n} r_k \Psi^i(P_k)\Phi^j(P_k)$. Otherwise the syndroms are called unknown at the beginning. The syndroms are not independent. Because of 5.3 we have that

$$O((i,j)) = O((i',j')) \Rightarrow s_{i,j} = c s_{i',j'} + \sum_{(O((k,l))<O((i,j)))} c_{(k,l)} s_{k,l}.$$

To be able to distinguish between dependent and independent syndroms we choose for every possible order $o$ a vector $b_o$ with $O(b_o) = o$ and denote by $B := \{b_o | o \text{ is a nongap of } Q\}$.

**Definition 5.6** *For two vectors and respectively two power products we define an admissible total ordering with respect to the two co-prime numbers $a$ and $b$.*
*Let $u = (u_1, u_2)$, $v = (v_1, v_2)$ be two elements of $\mathbb{N}_0^2$.*

$$u <_O v :\Longleftrightarrow au_1 + bu_2 < av_1 + bv_2 \vee (au_1 + bu_2 = av_1 + bv_2 \wedge u_2 < v_2)$$

*Remark:* This admissible ordering, which is defined according to the nongap structure of the curve, will turn out to be the appropriate ordering for the application of Sakata's algorithm to 1-point AG-codes.
For a rational function $\overline{f} = \sum_{q=(q_1,q_2)\in S(f)} f_q \Psi^{q_1}\Phi^{q_2}$ and its corresponding polynomial $f = \sum_{q=(q_1,q_2)\in S(f)} f_q x^{q_1} y^{q_2}$ we get the following theorem which gives us a hint how to use Sakata's algorithm for decoding.

**Theorem 5.6** *The function $\overline{f} = \sum_{q=(q_1,q_2)\in S(f)} f_q \Psi^{q_1}\Phi^{q_2}$ vanishes on all error positions, i.e,*

$$\overline{f}(P_l) = \sum_{q=(q_1,q_2)\in S(f)} f_q \Psi(P_l)^{q_1}\Phi(P_l)^{q_2} = 0 \qquad \text{if } l \in E$$

*if and only if $f = \sum_{q=(q_1,q_2)\in S(f)} f_q x^{q_1} y^{q_2} \in \mathrm{VALPOL}(s,b)$ for a $b$ for which $O(b) \geq e_{\mathrm{num}} + 2g - 1 + O(\mathrm{lpp}(f))$.*

*Proof.* By $g_{(i,j)}$ we denote the rational function $\Psi^i\Phi^j$.

$$\sum_{(i,j)\in S(f)} f_{(i,j)} s_{(i,j)+p-\mathrm{lpp}(f)} = \sum_{(i,j)\in S(f)} f_{(i,j)} \sum_{k\in E} g_{(i,j)+p-\mathrm{lpp}(f)}(P_k) e_k \tag{5.4}$$

$$= \sum_{k\in E} e_k g_{p-\mathrm{lpp}(f)}(P_j) \sum_{(i,j)\in S(f)} f_{(i,j)} g_{(i,j)}(P_k). \tag{5.5}$$

So if $\sum_{(i,j)\in S(f)} f_{(i,j)} \Psi^i(P_j)\Phi^j(P_k) = 0$ for all $k \in E$, i.e the rational function $\overline{f}$ vanishes on the error positions, the corresponding polynomial fulfills the recursion relation for all $p \in \mathbb{N}_0^2$.

On the other hand the function

$$\begin{aligned} \text{EV: } L(aQ) &\rightarrow (\mathbb{F}_q)^{e_{\text{num}}} && (5.6)\\ f' &\mapsto (f'(P_{i_1}), ..., f'(P_{i_{e_{\text{num}}}})) && (5.7) \end{aligned}$$

is surjective (due to the Riemann-Roch theorem, see Theorem 5.2) if $a - 2g + 1 \geq e_{\text{num}}$. So if the polynomial $f$ is valid for all $p$ such that $O(p - \text{lpp}(f)) \leq 2g + e_{\text{num}} - 1$, we can be sure that the corresponding rational function has to vanish on all error positions. This is the case if $f$ is valid for all $p$ such that $O(p) \leq 2g + e_{\text{num}} + O(\text{lpp}(f)) - 1$. So if $f \in \text{VALPOL}(s, b)$ for a $b$ with $O(b) \geq 2g + e_{\text{num}} + O(\text{lpp}(f))$, $\overline{f}$ has to vanish on the error positions. □

We know the syndroms only up to the order $m$. The next question, we want to answer is, how many errors we can correct, if we apply Sakata's algorithm for the known syndroms. To do this, if we want to apply the last theorem, we need a bound for $O(\text{lpp}(f))$, for an $f$ generating the syndrom array up to the first unknown syndrom.
The next theorem gives us a bound on $\Delta(\text{VALPOL}(s, b))$ for any $b$.

**Theorem 5.7** *Let $b \in {\mathbb{N}_0}^2$.*

$$|B \cap \Delta(\text{VALPOL}(s, b))| \leq e_{\text{num}}.$$

*Proof.* Let $R$ denote the ring of rational functions which have no pole outside $Q$, and let $I \subseteq R$ be the ideal of those functions vanishing on the error positions $P_{i_1}$,...,$P_{i_{e_{\text{num}}}}$.
Then the dimension of $R/_I$, as a vector space over $\mathbb{F}_q$ is equal to $e_{\text{num}}$.
For each $p \in B \cap \Delta(\text{VALPOL}(s, b))$ we take a polynomial $\sigma_p$ with $\text{lpp}(\sigma_p) = p$, the corresponding rational function $\overline{\sigma_p}$ and denote by $[\overline{\sigma_p}]$ the equivalence class of $\overline{\sigma_p}$ in $R/_I$.
$\overline{\sigma_p} \notin I$, because otherwise $\sigma_p$ would be an element of VALPOL($s$), which is not possible because $\text{lpp}(\sigma_p) \in \Delta(\text{VALPOL}(s, b))$.
Any linear combination of $\overline{\sigma_p}$, $p \in B \cap \Delta(\text{VALPOL}(s, b))$ can't be zero, because rational function of different pole order are linearly independent. So the leading power product of every linear combination is an element of $\Delta(\text{VALPOL}(s, b))$. Therefore every linear combination of $\overline{\sigma_p}$ can't be an element of $I$.
Therefore $|B \cap \Delta(\text{VALPOL}(s, b))| \leq \dim(R/_I) = e_{\text{num}}$. □

As a consequence of the last theorem, we get the next theorem.

**Theorem 5.8** *If $e_{\text{num}}$ errors occurred, we can find for every $b \in \mathbb{N}_0^2$ an element $f(x, y)$ of VALPOL$(s, b)$ such that*

$$O(\text{lpp}(f)) \leq e_{\text{num}} + g.$$

*Here $g$ denotes again the genus of the curve $\mathcal{C}$.*

*Proof.* Suppose that for every $f(x, y) \in \text{VALPOL}(s, b)$, $O(\text{lpp}(f)) > e_{\text{num}} + g$. Then $\{p | p \in {\mathbb{N}_0}^2 \wedge O(p) \leq e_{\text{num}} + g\} \subseteq \Delta(\text{VALPOL}(s, b))$.

Because of the last theorem we get that the number of vectors with different pole numbers in $\Delta(\text{VALPOL}(s,b))$ can't exceed $e_{\text{num}}$. Since there can only be $g$ gap numbers, and we have that all vectors of order $0, 1, ..., e_{\text{num}} + g$ are elements of $\Delta(\text{VALPOL}(s,b))$, we arrive at a contradiction. Thus there must exist a function $f(x,y) \in \text{VALPOL}(s,b)$ with $O(\text{lpp}(f)) \leq e_{\text{num}} + g$. □

*Remark:* We know all syndroms up to the order $m$. We can use Sakata's algorithm to find a minimal set of polynomials generating the syndrom array up to the value first value $k$ with $O(k) = m+1$. From Theorem 5.6 we get that the corresponding rational functions vanish on the error positions, if $e_{num} \leq m - 2g + 1 - O(\text{lpp}(f))$. Since there is an $f \in \text{VALPOL}(s,k)$ with $O(\text{lpp}(f)) \leq e_{num} + g$ we get that if $e_{num} \leq \frac{m-3g+1}{2}$ we can find the error positions as roots of the corresponding rational function $\overline{f}$. This leads to the next theorem.

**Theorem 5.9** *Let $C_\Omega(mQ, D)$ be a 1-point AG-code over a smooth curve of genus $g$. Let $k$ be the first vector (with respect to $<_O$), such that $O(k) > m$.*
*If the number of errors satisfies $e_{num} \leq \frac{m-3g+1}{2}$, there is at least a polynomial $f \in$ $\text{VALPOL}(s,k)$ with $O(\text{lpp}(f)) \leq e_{\text{num}} + g$. The corresponding rational function $\overline{f}$ vanishes on the error positions.*

Thus we can find a rational function let's call it $\overline{f}$ vanishing at all error positions by the application of Sakata's algorithm if $e_{num} \leq \frac{m-3g+1}{2}$. Let $R$ denote the set $\{i | \overline{f}(P_i) = 0\}$. To find the error values, we solve the equation

$$H \cdot x = s \tag{5.8}$$

with the restrictions that

$$x_i = 0 \qquad \text{for all } i \notin R. \tag{5.9}$$

Let $\{g_1, ..., g_{n-k}\}$ be a basis for $\mathcal{L}(mQ)$. $H$ denotes the parity check matrix

$$H = (g_i(P_j))_{1 \leq i \leq n-k, 1 \leq j \leq n}$$

and $s$ the syndrom vector $(s_i)_{1 \leq i \leq n-k}$ with $s_i = \sum_{j=1}^{n} e_j g_i(P_j)$.
Obviously the error vector $e$ solves the equations, and fulfills the restrictions, since all error positions are in R. The next theorem tells us that this is the only solution to the system, satisfying the restrictions.

**Theorem 5.10** *Let $C_\Omega(mQ, D)$ be a 1-point AG-code over a smooth curve of genus $g$, let $e_{num} \leq \frac{m-3g+1}{2}$ and let $f$ be a polynomial with minimal $O(\text{lpp}(f)) \in \text{VALPOL}(s,k)$.*

*The equation 5.8 with the restrictions 5.9 has the unique solution $e$, the actual error vector.*

*Proof.* Obviously the error vector $e$ is a solution to the system. Suppose there were two solutions $e$ and $e'$ with $e \neq e'$. Then $H \cdot (e - e') = 0$ and $e - e' \in C_\Omega(mQ, D)$. Let $d$ denote the Hamming distance. $d(e - e', 0) = d(e, e') \geq d_\Gamma = m - 2g + 2$.
On the other hand we get from Theorem 5.4 that the number of roots of $f$ can't exceed $e_{\text{num}} + g$, because $f \in \mathcal{L}((e_{\text{num}} + g)Q)$.

Since $e_i$ and $e'_i$ are both zero if $i \notin R$, they can only be different at at most $e_{\text{num}} + g$ positions.
Therefore $d(e,e') \leq e_{\text{num}} + g \leq m - 3g + 1 + g = m - 2g + 1 < m - 2g + 2$.
Thus we arrive at a contradiction and there is a unique solution, namely the error vector $e$. $\square$

So we can correct $e_{num} \leq \frac{m-3g+1}{2}$ errors applying Sakata's algorithm directly without estimating new syndroms. The resulting algorithm, we will call the *Basic Algorithm*.

### 5.2.1 Summary of the Basic Algorithm

Consider the 1-point AG-code $C_\Omega(mQ, D)$ over a curve of genus $g$, whose semi-group of nongaps is generated by the two co-prime integers $a$ and $b$. Let $D = \sum_{i=1}^{n} P_i$. Let $\Psi$ be a rational functional with a pole of order $a$ at $Q$ and no other poles and $\Phi$ be a rational function with a pole of order $b$ at $Q$ and no other poles.
**Initialization - once for a given code**

1. Find a basis $\{g_1, ..., g_{n-k}\}$ for $\mathcal{L}(mQ, D)$.
2. Calculate the parity check matrix $H$, i.e:
$$H = (g_i(P_j))_{i=1,\dots,n-k;j=1,\dots,n}.$$
3. For all $v$ with $O(v) \leq m$ and all $1 \leq i \leq n$ calculate
$$\Psi^{v_1}(P_i)\Phi^{v_2}(P_i).$$

**Decoding**
Let $w$ be the received word.

1. Calculate $s_v$ for all $v$ with $O(v) \leq m$, that is compute
$$s_v = \sum_{i=1}^{n} w_i \Psi^{v_1}(P_i)\Phi^{v_2}(P_i).$$
*Remark:* Because of the calculations during the initialization, the syndroms can be computed without evaluating rational functions.
2. If all syndroms are zero, no error occurred and the received word is the sent code word.
3. Let $s$ be the array containing the calculated syndroms, i.e:
$$s_{ij} := s_{(i,j)}$$
Let $l$ be the first (with respect to $<_O$) vector, such that $O(l) > m$. Use Sakata's algorithm to calculate a minimal set of polynomials $F$ generating $s$ up to the value $l$, with respect to $<_O$, i.e
$$F := \text{Sakata's algorithm}(s, l, <_o).$$

4. Find a polynomial $f \in F$ with $O(\text{lpp}(f))$ is minimal.

5. Calculate the error locations R, i.e

$$R = \{1 \leq i \leq n | \overline{f}(P_i) = 0\},$$

where $\overline{f}$ denotes the corresponding rational function of $f$.

6. Find the error values $e_i$ by solving the linear system

$$H \cdot e = s \tag{5.10}$$

with the restrictions that

$$e_i = 0 \qquad \text{for all } i \notin R. \tag{5.11}$$

$c := w - e$ is the nearest neighbor code word of $w$.

**Example 5.2** *We consider once again a Hermitian code over $H_2$. Let $D$ be the divisor of the $2^3$ affine $\mathbb{F}_4$-rational points. Let $P_\infty := (0:1:0)$. We consider the one-point code $C_\Omega(6P_\infty, D)$. The designed distance is $d_\Gamma = 6 - 2g + 2 = 6$, therefore we constructed an at least 2-error correcting code. Applying the Basic Algorithm we can correct $\frac{m-3g+1}{2} = \frac{6-3+1}{2} = 2$ errors. This is one of the few cases, in which we don't lose error-correcting capacity using only the Basic Algorithm.*
*As a basis for $\mathcal{L}(6P_\infty)$ we choose $\{1, y, y^2, x, xy, x^2\}$.*
*As a parity check matrix we get:*

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 \\ 0 & 1 & \alpha^2 & \alpha & \alpha^2 & \alpha & \alpha^2 & \alpha \\ 0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 \\ 0 & 0 & \alpha & \alpha^2 & \alpha^2 & 1 & 1 & \alpha \\ 0 & 0 & 1 & 1 & \alpha^2 & \alpha^2 & \alpha & \alpha \end{bmatrix}.$$

*Let the received word be*

$$w = \left[\alpha, \alpha, \alpha, \alpha^2, 0, \alpha, 1, 1\right].$$

*The known part of the syndrom array is:*

$$s = \begin{bmatrix} \alpha^2 & \alpha^2 & 0 \\ \alpha & 0 & ?_{2,3} \\ 0 & ?_{3,2} & ?_{3,3} \end{bmatrix}.$$

*We apply Sakata's algorithm according to the order $<_O$ up to the value $(2,1)$ with $O(2,1) = 7$.*
*Applying Sakata's algorithm we get that a minimal set of polynomials generating $s$ up to the value $(2,1)$ is*

$$F = [\alpha\, x + y,\, \alpha + \alpha^2\, x + x^2]$$

*The polynomial of minimal is $f(x,y) = \alpha x + y$, with $O(\mathrm{lpp}(f)) = 3 \leq t+g = 3$. We know that this polynomial has to vanish on the error positions. The roots of $f(x,y)$ are $P_1, P_3$ and $P_6$, therefore $R = \{1,3,6\}$.*
*To get the error values we have to solve the equation*

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & \alpha & \alpha^2 \\ 0 & \alpha^2 & \alpha \\ 0 & 1 & \alpha \\ 0 & \alpha & 1 \\ 0 & 1 & \alpha \end{bmatrix} \cdot \begin{bmatrix} e_1 \\ e_3 \\ e_6 \end{bmatrix} = \begin{bmatrix} \alpha^2 \\ \alpha^2 \\ 0 \\ \alpha \\ 0 \\ 0 \end{bmatrix}.$$

*The unique solution is $e_1 = 0, e_3 = 1$ and $e_6 = \alpha$. Thus the error vector $e$ is $[0,0,1,0,0,\alpha,0,0]$ and the sent code word is $c = w - e$.*

By the Basic Algorithm, we are able to correct $\lfloor \frac{m-3g+1}{2} \rfloor$ errors, but since the minimum distance of the code is at least $d_\Gamma = m - 2g + 2$, this result is not satisfying. If we knew the syndroms up to the order $o = t + 2g - 1 + (t+g) = 2t + 3g - 1$ we could could apply Sakata's algorithm for all known syndroms, and determine the error values as in the Basic Algorithm.
The next goal is to calculate these unknown syndroms.

## 5.3 Estimating additional syndroms

Sakata's algorithm provides a minimal set of polynomials generating the array $s$ up to the first value $l$, with $O(l) > m$. We now want to use this minimal polynomial set to determine the syndroms of order $m+1$.

Suppose more generally that we know all syndroms up to the order $o-1 \geq m$ and that we constructed by Sakata's algorithm a minimal set of polynomials $F$ in VALPOL$(s,l)$, where $l$ denotes the first value such that $O(l) = o$. By $\Delta$ we denote $\Delta(F)$. By $U_o = \{u | O(u) = o\}$ the set of all vectors of order $o$.
We choose the minimal set of polynomials such that for every $f \in F$, $\mathrm{lpp}(f) \in B$. Because of 5.3 this can be reached.
Let $f$ be an element of $F$. If there is a $u \in U_o$ such that $\mathrm{lpp}(f) \leq u$, we can use $f$ to give a 'guess' for $s_u$. If $f$ is also valid at $u$ we have that

$$d_{f,s,u} = \sum_{(i,j) \in S_f} f_{(i,j)} s_{u - \mathrm{lpp}(f) + (i,j)} = 0$$

and therefore

$$s_u = -\frac{1}{\mathrm{lc}(f)} \sum_{(i,j) \in S_f \setminus \{\mathrm{lpp}(f)\}} f_{(i,j)} s_{u - \mathrm{lpp}(f) + (i,j)}.$$

In this situation we can calculate $s_u$, since all other syndroms are of lower order and are known.

If $f$ isn't valid at $u$, there can't be any polynomial $f'$ with $\text{lpp}(f') \leq u - \text{lpp}(f)$ and the set $\Delta' := \Delta(F')$ for a minimal set of polynomials $F' \in VALPOL(s,k)$ with $O(k) \geq o+1$ contains all vectors $p$ such that $p \leq u - \text{lpp}(f)$.

For every $f$ in $F$ we can determine how many new independent vectors would be introduced to the set $\Delta'$ if $f$ was not valid. For every $f \in F$, we look whether there is a $u \in U_o$, such that $\text{lpp}(f) \leq u$. If this is not the case $f$ is not used for voting.
If $f$ is not valid at $u$, $\Delta'$ contains the set $\{p|p \leq u - \text{lpp(f)}\}$. An element of this set is an element of $\Delta' \setminus \Delta$, if and only if it was in the exterior of $\Delta$.
So if $f$ was not valid at $u$, the new elements in $\Delta'$ would be

$$K_f := \{p|p \leq u - \text{lpp}(f) \wedge (\exists f' \in \text{VALPOL}(s,l) : \text{lpp}(f') \leq p)\}.$$

Since we are only interested in new elements, which belong to the set $B$ and for some reasons, which will become clearer later, only for such elements for which we can find an element in $B$ such that the order of the sum is equal to $o$, we define for $j \in \mathbb{N}$

$$K(j) := \{p \in B | \exists \gamma : O(\gamma) = j, p \leq \gamma \wedge \gamma - p \in B\}.$$

and for every $f \in F$

$$K'_f := K_f \cap K(o).$$

**Definition 5.7** *Let $U_o$ denote the set of all vectors of order $o$. For every $f$ in $VALPOL(s,l)$ we define:*
*$f$ is* used for voting *iff there is a $u \in U_o$ such that*

- $u \geq \text{lpp}(f)$
- $u - \text{lpp}(f) \in B$

*For every $f$ in $VALPOL(s,l)$ which is used for voting we define by*

$$\text{vote}_{f,u} := -\frac{1}{\text{lc}(f)} \sum_{(i,j) \in S_f \setminus \{\text{lpp}(f)\}} f_{(i,j)} s_{u - \text{lpp}(f) + (i,j)}$$

*the* vote of $f$ for $s_u$.

*Let $b \in B$ with $O(b) = o$. Since $O(b) = O(u) = o$, $s_u = cs_b + \sum_{(O((k,l))<O(u))} c_{(k,l)} s_{k,l}$.*
*By*

$$\text{vote}_f := \frac{1}{c}(\text{vote}_{f,u} - \sum_{(O((k,l))<O(u))} c_{(k,l)} s_{k,l})$$

*we define the* vote of f.

*Let $w_1, ...w_p$ the different votes of all polynomials in $VALPOL(s,l)$ which are used for voting.*
*For $i = 1, ..., p$ let $L_i$ denote the union of all sets $K'_f$, for which* $\text{vote}_f = w_i$.

*By $|L_i|$ we define the* number of votes for $w_i$.
*By $K'$ we denote $L_1 \cup \ldots \cup L_p$ and define by $|K'|$ the* (total) number of votes.

**Example 5.3** *We consider the Hermitian code over $H_4$ and set $m = 25$. Suppose that $e_{\text{num}} \leq t = 7$.*

*The known part of the syndrom array is:*

$$s = \begin{bmatrix} 1 & \alpha^6 & \alpha & 1 & \alpha^2 & \alpha^4 & \alpha^{12} & S_{8,1} & S_{9,1} & \\ \alpha^3 & \alpha^8 & \alpha^6 & \alpha^{11} & \alpha^9 & \alpha^{12} & S_{7,2} & & & \\ \alpha^7 & \alpha^6 & \alpha^{14} & \alpha^{10} & S_{5,3} & S_{6,3} & & & & \\ 1 & \alpha^{14} & \alpha^8 & S_{4,4} & S_{5,4} & & & & & \\ \alpha^7 & \alpha^9 & S_{3,5} & S_{4,5} & & & & & & \\ \alpha^2 & S_{2,6} & & & & & & & & \\ S_{1,7} & & & & & & & & & \\ & & & & & & & & & \end{bmatrix}.$$

*The first unknown syndrome is $s_{(4,2)}$ with $O(4,2) = 26$. Sakata's algorithm gives us a minimal set of polynomial $F$ generating $s$ up to the value $(4,2)$. $U_{26} = \{(4,2)\}$.*

$$\begin{aligned} \text{“}F\text{”} = [&\alpha^3\,x\,y^2 + \alpha^{11} + \alpha^9\,x + y + \alpha^3\,x^2 + \alpha^{10}\,x^3 + \alpha^5\,y^2 + \alpha^4\,y\,x + y^3 + \alpha^{12}\,y\,x^2, \\ &x\,y^2 + \alpha^7 + \alpha^{10}\,x + \alpha^{11}\,y + \alpha^{14}\,x^3 + \alpha^{14}\,y^2 + \alpha^{11}\,y\,x + \alpha^3\,y\,x^2, \\ &\alpha^{12} + \alpha^3\,x + \alpha^{11}\,y + x^2 + \alpha\,x^3 + \alpha^3\,y^2 + \alpha^2\,y\,x + y\,x^2, \\ &\alpha^3 + \alpha^{11}\,x + \alpha^4\,y + \alpha^5\,x^2 + \alpha^5\,x^3 + x^4 + \alpha^{10}\,y\,x + \alpha^9\,y\,x^2] \end{aligned}.$$

*We visualize $\Delta(F)$ in Figure 5.1. The elements in $B$ are marked gray, the vectors which are not in $B$ are marked black.*
*We see that $|\Delta(F) \cap B| = 7 = t$. For every $f$ in $F$ we calculate $K'_f$.* $\text{lpp}(f_1) = (0,3) \not\leq (4,2)$ *so $f_1$ can't be used for voting.*
$\text{lpp}(f_2) = (1,2)$, $\text{span}(f_2) = (4,2) - (1,2) = (3,0)$ *but since $(3,0) \in \Delta(F)$ there would be no new vectors in the delta set if the vote of $f_2$ was wrong and $K'_{f_2} = \emptyset$.*
$\text{lpp}(f_3) = (2,1)$, $\text{span}(f_3) = (4,2) - (2,1) = (2,1)$, $(2,1) \in B$,$(2,1) \notin \Delta(F)$ *and* $(4,2) - (2,1) = (2,2) \in B$, *therefore $(2,1) \in K'_{f_3}$. But it is the only element because all other vectors smaller than $(2,1)$ are already elements in $\Delta(F)$.*
$\text{lpp}(f_4) = (4,0)$, *span*$(f_4) = (0,2) \in \Delta(F)$, *therefore $K'_{f_4} = \emptyset$.*

*In this example we seem to be lucky. If we don't take the vote of the function $f_3$, the size of the $\Delta(F)$ increases by 1, which can't be true if not more the 7 errors occurred. Thus we have to take the vote of $f_3$ which is 0 and we get that $s_{(4,2)} = 0$.*

Let $w_1$,...,$w_p$ be the <u>different</u> votes of the polynomials $f \in F$. Let $g$ be the number such the number of votes for $w_g$ is maximal. The surprising result (which is due to Feng and Rao) is, that $w_g$ is always the true value of $s_b$. To proof this we need to define the Feng-Rao bound.

**Definition 5.8** *The* Feng-Rao bound of the code $C_\Omega(mQ, D)$ *(denoted by $d_{FR}(C)$) is defined as:*

$$d_{FR}(C) := \min(\{|K(o)|| o > m\}).$$

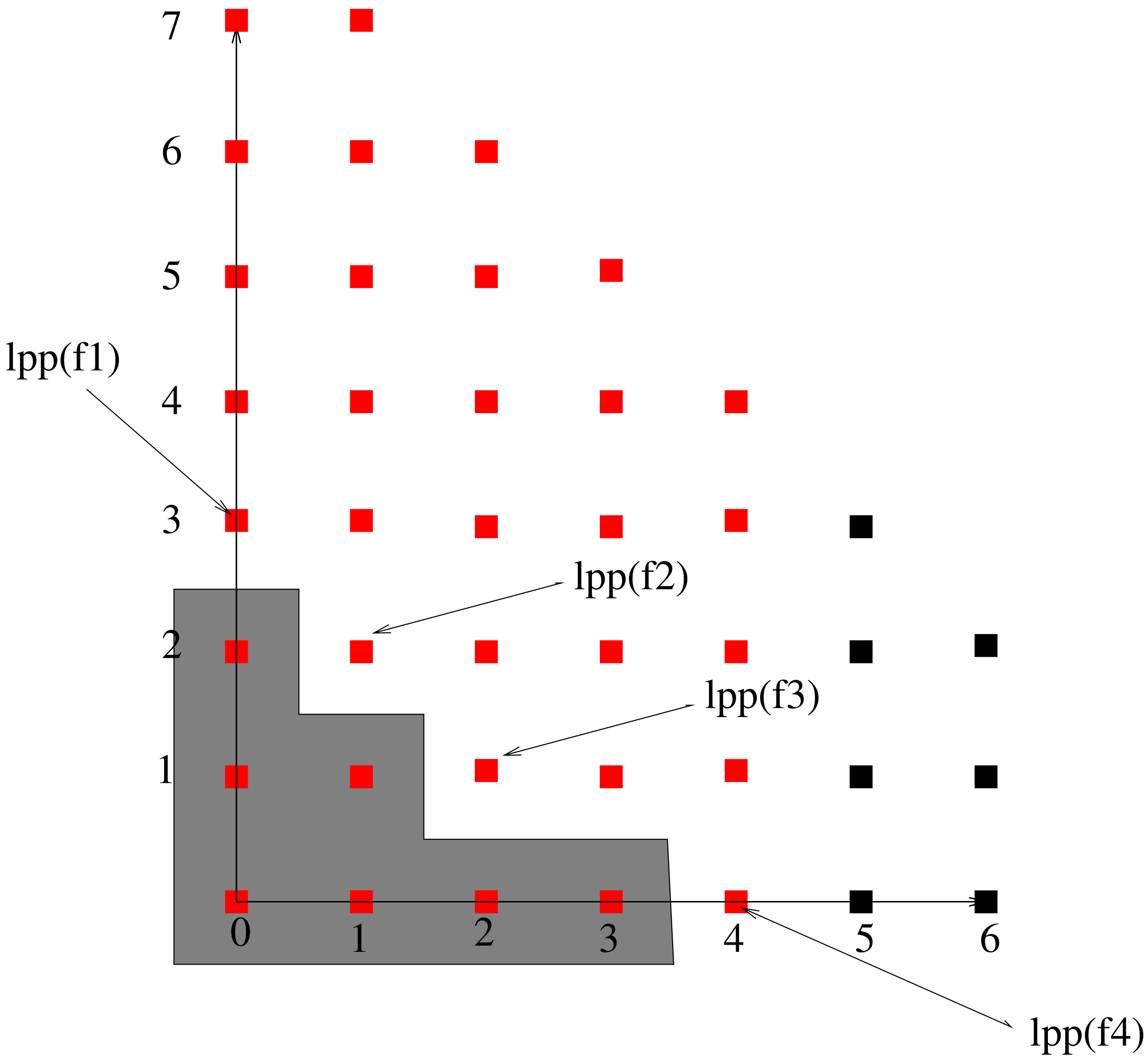



Figure 5.1: Visualization of $\Delta(F)$, Example 5.3

It was proved by Feng and Rao that

$$d_{FR} \geq m - 2g + 2 = d_\Gamma.$$

**Theorem 5.11** *Suppose that the number of errors* $e_{\mathrm{num}}$ *satisfies*

$$e_{\mathrm{num}} \leq t = \lfloor \frac{d_{FR} - 1}{2} \rfloor$$

*and let* $g$ *be the number such that* $|L_g|$ *is maximal. Then we have that*

$$s_b = w_g.$$

To proof this theorem we need the following Lemma.

**Lemma 5.1**
$$|K'| \geq |K(o)| - 2|\Delta|.$$

*Proof.*(Lemma) We construct an injective mapping from $K(o) \setminus (K' \cup \Delta)$ to $|\Delta|$. Let $x \in K(o) \setminus (K' \cup \Delta)$. Since $x \in K(o)$, $x \in B$ and there is a unique vector $p_x$ such that $O(p_x) = o, p_x \geq x$ and $p_x - x \in B$. Since $x \in K(o)$ and $x \notin K' \cup \Delta$, there is no polynomial $f_i \in F$ such that $x \leq p_x - \text{lpp}(f_i)$. Consequently we get that $p_x - x \geq 0$ and $p_x - x \ngeq \text{lpp}(f_i)$ for all $f_i \in F$. Thus $p_x - x \in \Delta$. The mapping :

$$\text{f: } K(o) \setminus (K' \cup \Delta) \rightarrow \Delta \tag{5.12}$$
$$x \mapsto p_x - x \tag{5.13}$$

is injective, because if $f(x) = f(y)$ then $O(p_x) - O(x) = O(p_y) - O(y)$, but then $O(x) = O(y)$ which means that $x = y$, since both $x$ and $y$ are in $K(o)$ and therefore in $B$.

Therefore $|K(o) \setminus (K' \cup \Delta)| \leq |\Delta|$. But since $|K(o)| - (|K'| + |\Delta|) \leq |K(o) \setminus (K' \cup \Delta)|$, $|K'| \geq |K(o)| - 2|\Delta|$. □

*Proof.*(Theorem 5.11) Suppose first that $s_b$ was different from all values $w_1, ..., w_p$. Then the next delta set which we denote by $\Delta'$, is increased at least by $|K'|$.
But then, using the Lemma we have

$$|\Delta'| \geq |\Delta| + |K'| \geq |K(o)| - |\Delta| \geq d_{FR} - t > t.$$

This can't be true, since by Theorem 5.7 $|\Delta'| \leq t$.
So one of the values $w_1, ...w_p$ must be correct, without restriction assume that $w_1$ is the correct one.
Set $\overline{L}_1 := L_2 \cup ... \cup L_p$. Since $w_2, ..., w_p$ are different from $s_b$, the $\Delta$-set will increase with at least by $\overline{L}_1$.
Therefore by Theorem 5.7 we have

$$|\Delta| + |\overline{L}_1| \leq t.$$

Using the definition of the Feng-Rao bound and the Lemma we get that

$$|\overline{L}_1| \leq t - |\Delta| < \frac{d_{FR}}{2} - |\Delta| \leq \frac{1}{2}|K'|.$$

Since $\overline{L}_1 \cup L_1 = K'$, we can conclude that $|L_1| > \frac{1}{2}|K'|$. But this gives us a general insight. If we, for any $j = 2, ..., p$, put
$$\overline{L}_j := \bigcup_{i \neq j} L_i$$
then $L_1 \subseteq \overline{L}_j$ and therefore $|\overline{L}_j| > \frac{1}{2}|K'|$. So we can conclude that the correct value $w_l$ corresponds to the maximum value of $|L_l|$. □

By majority voting we can determine new originally unknown syndroms up to the order $2t + 3g - 1$. We know that a polynomial of the minimal set $F$ generating s up to the first value of order $2t + 3g$, and whose leading power product is of order less or equal to $t + g$ has to vanish on the error positions. From exactly the same observations as in the proof of Theorem 5.10, we get that we can determine the error vector by solving the a linear system using all known syndroms.

### 5.3.1 Summary of the Extended Algorithm

Consider the 1-point AG-code $C_\Omega(mQ, D)$ over a curve of genus $g$, whose semi-group of nongaps is generated by the two co-prime integers $a$ and $b$. Let $D = \sum_{i=1}^{n} P_i$. Let $\Psi$ be a rational function with a pole of order $a$ at $Q$ and no other poles and $\Phi$ be a rational function with a pole of order $b$ at $Q$ and no other poles.
**Initialization - once for a given code**

1. Find a basis $\{g_1, ..., g_{n-k}\}$ for $\mathcal{L}(mQ, G)$.

2. Calculate the parity check matrix $H$, i.e:

$$H = (g_i(P_j))_{i=1,\dots,n-k;j=1,\dots,n}.$$

3. For all $v$ with $O(v) \leq m$ and all $1 \leq i \leq n$ calculate

$$\Psi^{v_1}(P_i)\Phi^{v_2}(P_i)$$

4. For all $o \leq 2t + 3g - 1$ choose exactly a vector $b_o$ such that $O(b_o) = o$, denote by $B = \{b_o | o \leq 2t + 3g - 1 \text{ and } o \text{ is a non-gap of } Q\}$.

5. For all $(i, j)$ with $O(i, j) \leq 2t + 3g - 1$ and $(i, j) \notin B$: Express $\Psi^i\Phi^j$ as linear combination of some $\Psi^k\Phi^l$ with $(k, l) \in B$ and $O((k, l)) \leq O((i, j))$.

**Decoding**
Let $w$ be the received word.

1. Calculate $s_v$ for all $v$ with $O(v) \leq m$, that is compute

$$s_v = \sum_{i=1}^{n} w_i \Psi^{v_1}(P_i)\Phi^{v_2}(P_i).$$

   *Remark:* Because of the calculations during the initialization, the syndroms can be computed without evaluating rational functions.

2. If all syndroms are zero, no error occurred and the received word is the sent code word.

3. Let $s$ be the array containing the calculated syndroms, i.e:

$$s_{ij} := s_{(i,j)}$$

   Let $l$ be the first (with respect to $<_O$) vector, such that $O(l) > m$. Use Sakata's algorithm to calculate a minimal set of polynomials $F$ generating $s$ up to the value $l$, with respect to $<_O$,i.e

$$F := \text{Sakata's algorithm}(s, l, <_o).$$

4. o:=m+1;
5. While $o \leq 2t + 3g - 1$ do
   (a) Using the linear combination found in Initialization step 5), normalize $F$ such that for each $f \in F$, lpp(f) $\in$ B.
   (b) For each $f \in F$ which can be used for voting determine $\text{vote}_f$ and $K'_f$.
   (c) $s_o$ is the vote $w_i$ such that $|L_i|$ is maximal.
   (d) Determine all other syndroms of order $o$ using the linear combinations found in Initialization step 5).
   (e) Run Sakata's algorithm for the new known syndroms to update F.
   (f) $o := o + 1;$

   enddo
6. Find a polynomial f $\in F$ for which $O(\text{lpp}(f))$ is minimal.
7. Calculate the error locations R, i.e
$$R = \{1 \leq i \leq n | \overline{f}(P_i) = 0\},$$
   where $\overline{f}$ denotes the corresponding rational function of $f$.
8. Find the error values $e_i$ by solving the linear system
$$\overline{H} \cdot e = \overline{s} \tag{5.14}$$
   with the restrictions that
$$e_i = 0 \qquad \text{for all } i \notin R. \tag{5.15}$$
   $\overline{H}$ denotes the matrix $\overline{H} = (\psi^{b_{o1}} \phi^{b_{o2}}(P_k))_{o<2t+3g,k=1,..,n}$ and $\overline{s} = (s_{b_o})_{o<2t+3g}$.
   $c := w - e$ is the nearest neighbor code word of $w$.

A disadvantage of the Extended Algorithm is, that we have to solve the error value system, i.e a linear system.
To avoid having to solve a linear system, we could go on calculating all syndroms $s_{i,j}$ for $0 \leq i \leq \mathbb{F}_q - 2$ and $0 \leq i \leq \mathbb{F}_q - 2$.
If we know all these syndroms, we can calculate the error values by a simple summation formular.
For details of this approach, which was also implemented in the tutorial implementation we refer to [SJH95].

# Chapter 6

# A tutorial implementation

## 6.1 Summary of the implementation

The purpose of this implementation is to allow the user to experiment with the codes presented in this diploma thesis, in the environment of the Maple computer algebra system. This tutorial implementation was developed by Harald Deischinger and the author during their work on their respective diploma theses. It not only contains an implementation of Sakata's approach to the decoding of 1-point AG-codes, but due to the work done by Harald Deischinger [Dei99] also an implementation of Duursma's approach. This enables the reader to apply both methods to the same code and compare their behaviour and their complexity.
This tutorial implementation is distributed as a part of the CASA package, developed by the Research Institute for Symbolic Computation (RISC-Linz) at the University of Linz under the direction of Franz Winkler [HHW99]. CASA may be obtained from the CASA home page

`http://www.risc.uni-linz.ac.at/software/casa`

or by writing to `casa@risc.uni-linz.ac.at`.

**Example 6.1** *(How to start the tutorial implementation:)*

```
> restart;with(casa);
            |__|
            |  |        Welcome to CASA 2.4.3 for Maple V.5
     |   /\|  |/\
    /=\__|  []  |
   /           \_       Copyright (C) 1990-1999 by Research Institute
   |             \      for Symbolic Computation (RISC-Linz), the
   \   CASA 2.4.3 |     University of Linz, A-4040 Linz, Austria.
    |             |
   _|     |||     |     For help type '?casa' or '?casa,<topic>'.
 __/      |||     |_
```

[*BCH2*, *BCHDecode*, *CyclicEncode*, *DivAbsDeg*, *DivAdd*, *DivBasisL*, *DivDeg*, *DivGe*, *DivGeNeg*, *DivLe*, *DivPretty*, *DivSub*, *DivSupp*, *Genus*, *GoppaDecode*, *GoppaEncode*, *GoppaFunctional*, *GoppaPrepareDu*, *GoppaPrepareSV*, *GoppaPrepareSa*, *GoppaPrimary*, *InPolynomial*, *NormalPolynomial*, *OutPolynomial*, *PolynomialRoots*, *RatEv*, *RatMapEv*, *RatMul*, *RatOrder*, *RatOrders*, *RatOrdersRat*, *RatPower*, *RationalPoints*, *SakataDecode*, *SubsPolynomial*, *finiteCurve*, *finiteField*, *finiteRational*, *init*, *makeDivisor*, *mapOutPolynomial*, *mapSubsPolynomial*]

```
>
```

After the installation of the package use the Maple call `with(casa)` and the full functionality of the implementation is available. Since this is a tutorial implementation you can determine how much information should be displayed during the computation. With the assignment `infolevel['casa/finite']:=level`, you can determine the output behaviour of the implementation. Higher values of `level` give more information. A value of 0 disables any message, except for warnings and errors. A level of 9 prints (almost) all information, except for bigger matrices.

### 6.1.1 Finite fields

The first step is to create the finite field $\mathbb{F}_q$ over which we would like to work. Maple supports finite fields using `RootOf`, but this method is not very convenient. So the first part of the implementation (done by Deischinger Harald), was to supply an easier interface to work with finite fields. The maple call `Fq:=finiteField(q)` prepares the necessary functions to work with the finite field $\mathbb{F}_q$. The returned table provides methods for conversion and some useful tools.
You can think of the function `finiteField(q)` as a constructor of the class `finiteField`. The returned table can `Fq` can be interpreted as an instance of this class.
A detailed description of all methods supplied by this object can be found in the online documentation. The two most important methods are:

```
> F16:=finiteField(16);
```

$$F16 := Field$$

```
> F16['In'](alpha);
```

$$\mathrm{RootOf}(\_Z^4 + \_Z + 1)$$

```
> a:=F16['In'](alpha^3+alpha+1);
```

$$a := \mathrm{RootOf}(\_Z^4 + \_Z + 1)^3 + \mathrm{RootOf}(\_Z^4 + \_Z + 1) + 1$$

```
> b:=F16['Inv'](a);
```

$$b := \mathrm{RootOf}(\_Z^4 + \_Z + 1)^2 + 1$$

```
> F16['Out'](b);
```

$$\alpha^8$$

```
> F16['Out'](a);
```

$$\alpha^7$$

```
>
```

Figure 6.1: Examples show how to use methods for a finite field

- `Fq['In'](x)`: This function converts the element `x` of the finite field `Fq` to Maple representation. This is done by replacing every occurrence of `alpha` in `x` by the `RootOf` representation of the primitive element. To get to know the `RootOf` representation of the primitive element `alpha`, one can use `Fq['In'](alpha)`.

- `Fq['Out'](x)`: This function takes an element in the `RootOf` representation, and tries to express it as a power of the primitive element `alpha`. If this is not possible (for example `x` is a variable) it is returned unchanged.

**Example 6.2** *Examples for how to work with finite fields can be found in Figure 6.1 or in the online-documentation.*

### 6.1.2 Polynomials over finite fields

Since we need a lot of computation in the domain of polynomials over finite fields, this implementation supplies basic operations for polynomials of two variables over a finite field. The functions supplied by the current implementation are:

- `InPolynomial(f,field)`: converts the polynomial `f` over the finite field `field` to `RootOf`-form.

- `OutPolynomial(f,field)`: converts the polynomial `f` in `RootOf`-form to `alpha`-form.

Generate the finite field of the coefficients of the polynomial

```
> GF16:=finiteField(16);
```

$$GF16 := Field$$

Generate the polynomial in RootOf form

```
> g:=InPolynomial(alpha*x+alpha^2*x^2*y,GF16);
```

$$g := \mathrm{RootOf}(\_Z^4 + \_Z + 1)\, x + \mathrm{RootOf}(\_Z^4 + \_Z + 1)^2\, x^2\, y$$

```
> SubsPolynomial([1,alpha],g,GF16);
```

$$\mathrm{RootOf}(\_Z^4 + \_Z + 1)^3 + \mathrm{RootOf}(\_Z^4 + \_Z + 1)$$

Find the roots of g over the finite field GF16. Print these roots in "nice" alpha form.

```
> map(GF16['mapOut'],PolynomialRoots(g,GF16));
```

$$[[0, 0], [0, 1], [0, \alpha], [0, \alpha^2], [0, \alpha^3], [0, \alpha^4], [0, \alpha^5], [0, \alpha^6], [0, \alpha^7], [0, \alpha^8], [0, \alpha^9], [0, \alpha^{10}], [0, \alpha^{11}], [0, \alpha^{12}], [0, \alpha^{13}], [0, \alpha^{14}], [1, \alpha^{14}], [\alpha, \alpha^{13}], [\alpha^2, \alpha^{12}], [\alpha^3, \alpha^{11}], [\alpha^4, \alpha^{10}], [\alpha^5, \alpha^9], [\alpha^6, \alpha^8], [\alpha^7, \alpha^7], [\alpha^8, \alpha^6], [\alpha^9, \alpha^5], [\alpha^{10}, \alpha^4], [\alpha^{11}, \alpha^3], [\alpha^{12}, \alpha^2], [\alpha^{13}, \alpha], [\alpha^{14}, 1]]$$

Generate the polynomial h.

```
> h:=InPolynomial((alpha^2*x*y+1)*x,GF16);
```

$$h := (\mathrm{RootOf}(\_Z^4 + \_Z + 1)^2\, x\, y + 1)\, x$$

Find the common roots of the polynomials g and h.

```
> map(GF16['mapOut'],PolynomialRoots([g,h],GF16));
```

$$[[0, 0], [0, 1], [0, \alpha], [0, \alpha^2], [0, \alpha^3], [0, \alpha^4], [0, \alpha^5], [0, \alpha^6], [0, \alpha^7], [0, \alpha^8], [0, \alpha^9], [0, \alpha^{10}], [0, \alpha^{11}], [0, \alpha^{12}], [0, \alpha^{13}], [0, \alpha^{14}]]$$

Print the expanded and collected product of g and h in alpha form.

```
> OutPolynomial(NormalPolynomial(g*h,GF16),GF16);
```

$$\alpha\, x^2 + \alpha^6\, y\, x^3 + \alpha^4\, y^2\, x^4$$

```
>
```

Figure 6.2: Examples for the calculations with polynomials over a finite field

- `PolynomialRoots(F,field)`: Calculates the common roots of all polynomials in the list of polynomials `F` over the finite field `field`.

**Example 6.3** *(Use of the methods for polynomials over finite fields) Examples for the use of the methods for polynomials over finite fields can be found in Figure 6.2.*

### 6.1.3 Smooth plane curves over finite fields

In order to keep the data associated with a plane curve over a finite field together the type `‘casa/finite/Curve‘` is supported. The function

```
C:=finiteCurve(f,Fq)
```

```
> H2:=finiteCurve(x^3+y^2+y, finiteField(4));
```

$$H2 := x^3 + y\,z^2 + y^2\,z$$

```
> Genus(H2);
```

$$1$$

```
> RationalPoints(H2);
```

$$[[0,0,1],[0,1,1],[1,\alpha,1],[1,\alpha^2,1],[\alpha,\alpha,1],[\alpha,\alpha^2,1],[\alpha^2,\alpha,1],[\alpha^2,\alpha^2,1],[0,1,0]]$$

```
>
```

Figure 6.3: Calculations with smooth plane curves over finite fields

is used for initialization, it takes as arguments the polynomial `f` defining the curve and the finite field `Fq` over which the curve is defined.
It is assumed that the curve is smooth. The system doesn't test this. Methods supplied for plane curves are:

- `Genus(C)`: This function calculates the genus of the curve `C`.
- `RationalPoints(C)`: Finds the `Fq`-rational points of the curve `C`. Points in the projective space are represented as a Maple list of length 3.

The application of these functions is shown in Figure 6.3

### 6.1.4 Rational Functions

For algebraic geometric codes, rational functions on a smooth plane curve are of importance. In this implementation a rational function is initialized by the call

```
phi:=finiteRational(f,curve)
```

The result is of type `casa/finite/Rational`. `f` will be homogenized if necessary. The most important functions to work with rational functions are implemented as:

- `RatOrder(P,phi)`: This function calculates the value $\nu_{\mathtt{P}}(\mathtt{phi})$
- `RatEv(phi,P)`: This function evaluates the rational function `phi` at the point `P`.
- `RatMul(phi1,phi2)`: This function multiplies the two rational functions `phi1` and `phi2`.
- `RatPower(phi,i)`: Takes the rational function `phi` to the power of `i`.

Examples can be found in Figure 6.4.

```
> phi:=finiteRational((y^3+y^2)/x^3,H2);
```

$$\phi := \frac{y^2 z + y^3}{x^3}$$

```
> RatOrder([0,0,1],phi);
```

$$3$$

```
> RatEv(phi,[0,1,1]);
```

$$1$$

```
>
```

Figure 6.4: Examples for the use of the methods for rational functions over plane algebraic curves

### 6.1.5 Divisors

To be able to construct general algebraic geometric codes, the data structure of a divisor is needed. We only need to construct one-point algebraic geometric codes, nevertheless (to be compatible to the rest of the implementation) we will use divisors.
A divisor is initialized using the function:

```
Di:=makeDivisor(Ps,nPs);
```

Here `Ps` is a list of points and `nPS` is a list of corresponding multiplicities.

- `DivPretty(Di)`: This functions is supplied for pretty printing a divisor as a string
- `DivAdd(D1,D2)`, `DivSub(D1,D2)`: These functions add/subtract two divisors
- `DivLe(D1,D2)`, `DivGe(D1,D2)`: These functions are used for the comparison of two divisors. `DivLe` tests if `D1` $\leq$ `D2` and `DivGe` tests for `D1` $\geq$ `D2`
- `DivDeg(Di)`: This function calculates the degree of the divisor `Di`.

Examples can be found in Figure 6.5.
For the construction of a basis of $\mathcal{L}(mQ)$, there is only a heuristic implementation. To construct a basis, only monomials are tested. This is efficient for most of the examples, especially for Hermitian curves and all the curves described in [SJMJH95b].
If this heuristic approach fails, a basis must be supplied by the user. For more details, please consult [Dei99].
Figure 6.6 shows a case in which the heuristic approach works.

### 6.1.6 2-dimensional BCH-codes

2-dimensional BCH codes can be constructed by the function `BCH2`. The arguments required are the domain of the code ($N_1$ and $N_2$), the parameter $\delta$ and the finite field $\mathbb{F}_q$.

```
> D1:=makeDivisor([[0,0,1],[0,1,1]],[1,-2]);
                                D1 := Divisor
> DivPretty(D1);
                        "+1 (0:0:1) -2 (0:1:1) "
> D2:=makeDivisor([[0,0,1],[1,1,0]],[2,1]);
                                D2 := Divisor
> DivLe(D1,D2);
                                    true
> DivPretty( DivAdd(D1,D2) );
                 "+3 (0:0:1) -2 (0:1:1) +1 (1:1:0) "
> DivPretty( DivSub(D1,D2) );
                 "-1 (0:0:1) -2 (0:1:1) -1 (1:1:0) "
> DivDeg(D2);
                                      3
>
```

Figure 6.5: Examples for divisors

```
> Di:=makeDivisor([[0,1,0]],[4]);
                                Di := Divisor
> DivPretty(Di);
                                "+4 (0:1:0) "
> DivBasisL(Di,H2,"monomial");
```

$$\left[1, \frac{y}{z}, \frac{x}{z}, \frac{x^2}{z^2}\right]$$

```
>
```

Figure 6.6: Example for the construction of $\mathcal{L}(4P_\infty)$ over $H_2$

The function `CyclicEncode` can be used for encoding. The two arguments required are the message which should be encoded and the code. The function `BCHDecode` can be used for decoding. Its parameters are the received word and the code. Details may again be found in the online documentation, and in the second example of the next section.

### 6.1.7 1-point AG-codes

The tutorial implementation supplies a method (`GoppaPrimary`) to construct primary Goppa codes. 1-point AG-codes are special primary Goppa codes, therefore you can use this method to construct a 1-point AG-code. The arguments required are the curve `C`, and two divisors `D` and `G`$=mQ$.
Encoding can be done by `GoppaEncode`. Its parameters are the message and the code.
For decoding, one has to initialize the method one would like to use for decoding. `GoppaPrepareSa`

prepares all preprocessing steps to use Sakata's approach to decoding. Its argument is the code and its result is a table of the results of all the computation which has to be done only once for a given code. `GoppaPrepareDu` initializes Duursma's method and if the genus of the curve is low one can also use `GoppaPrepareSV` to initialize the basic algorithm, which doesn't use majority voting for unknown syndroms.
After this initialization you can use `GoppaDecode` or `SakataDecode` to decode a received word. The parameters of this methods are the code, the result of the initialization, and the received word.
Details may again be found in the third example of the next section or in the online documentation.

## 6.2 Example sessions

To demonstrate the tutorial implementation, and to make the theory of 2-dimensional codes and AG-codes clearer, we include some example sessions. In all example sessions we first construct a code, then encode a message, make random errors and try to decode the received word.
As examples we choose a Reed-Solomon code treated as a special example of a two dimensional BCH-code, a 2-error correcting 2-dimensional BCH code, and a Hermitian code.
To the reader it might be interesting to compare the two most frequently used methods for decoding one-point AG-codes, i.e the application of Sakata's algorithm and Duursma's approach. For this reason, we decode the Hermitian code in both ways.

### 6.2.1 Reed-Solomon Code

The first example session is a Reed-Solomon code over $\mathbb{F}_{16}$ with designed minimum distance 5. We force the generator polynomial to vanish on 4 subsequent powers of $\alpha$, namely $1, \alpha, \alpha^2, \alpha^3$.
This code can be regarded as a 2-dimensional BCH-code of domain (1,15) and $\delta = (4, 0)$. The Maple output start on page 71.

### 6.2.2 2-dimensional BCH-code

We want to construct an at least 2-error correcting code, we choose $\delta = (4, 0)$. As a field we choose $\mathbb{F}_4$ and as a domain $(7, 7)$. The Maple output start on page 75.

### 6.2.3 Hermitian Code

As the last example we take an Hermitian code, defined over the curve

$$H_4 = V(x^5 + y^4z + yz^4)$$

over the field $\mathbb{F}_{16}$.
This is a code of length 64. We will show how to decode it using Duursma's approach and Sakata's approach. The Maple output starts on page 78.

# Example: Reed-Solomon code as special case of a 2-dimensional BCH-code

## Loading files:

```
> restart;with(casa);
           |__|
           |  |          Welcome to CASA 2.4.3 for Maple V.5
     |   /\|  |/\
    /=\__|  []  |
   /            \_       Copyright (C) 1990-1999 by Research Institute
   |              \      for Symbolic Computation (RISC-Linz), the
   \   CASA 2.4.3 |      University of Linz, A-4040 Linz, Austria.
    |             |
   _|     |||     |      For help type '?casa' or '?casa,<topic>'.
__/       |||     |_
```

[*BCH2*, *BCHDecode*, *CyclicEncode*, *DivAbsDeg*, *DivAdd*, *DivBasisL*, *DivDeg*, *DivGe*, *DivGeNeg*, *DivLe*, *DivPretty*, *DivSub*, *DivSupp*, *Genus*, *GoppaDecode*, *GoppaEncode*, *GoppaFunctional*, *GoppaPrepareDu*, *GoppaPrepareSV*, *GoppaPrepareSa*, *GoppaPrimary*, *InPolynomial*, *NormalPolynomial*, *OutPolynomial*, *PolynomialRoots*, *RatEv*, *RatMapEv*, *RatMul*, *RatOrder*, *RatOrders*, *RatOrdersRat*, *RatPower*, *RationalPoints*, *SakataDecode*, *SubsPolynomial*, *finiteCurve*, *finiteField*, *finiteRational*, *init*, *makeDivisor*, *mapOutPolynomial*, *mapSubsPolynomial*]

## determining the infolevel:

```
> infolevel[`casa/finite`]:=11;
```

$$\mathit{infolevel}_{\mathit{casa/finite}} := 11$$

## generate a 2 error correcting code:

### *generate the field over which the code should work*

```
> GF16:=finiteField(16);
```

$$\mathit{GF16} := \mathit{Field}$$

### *generate the code (delta>=(4,0))*

```
> C:=BCH2(1,15,[4,0],GF16);
```

"Common roots" = [[0, 0], [0, 1], [0, 2], [0, 3]]

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \alpha & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 & \alpha^7 & \alpha^8 & \alpha^9 & \alpha^{10} & \alpha^{11} & \alpha^{12} & \alpha^{13} & \alpha^{14} \\ 1 & \alpha^2 & \alpha^4 & \alpha^6 & \alpha^8 & \alpha^{10} & \alpha^{12} & \alpha^{14} & \alpha & \alpha^3 & \alpha^5 & \alpha^7 & \alpha^9 & \alpha^{11} & \alpha^{13} \\ 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} & 1 & \alpha^3 & \alpha^6 & \alpha^9 & \alpha^{12} \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^6 & \alpha^9 & \alpha^2 & \alpha^6 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^{12} & \alpha^8 & \alpha^5 & \alpha^7 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^{13} & 1 & \alpha^5 & \alpha^{11} \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^2 & \alpha^{11} & \alpha^7 & \alpha^6 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & \alpha^{12} & \alpha^9 & \alpha^{12} & \alpha^2 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & \alpha^8 & \alpha^8 & \alpha^{14} & \alpha^{11} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & \alpha^2 & \alpha^9 & \alpha^3 & \alpha^3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \alpha^9 & \alpha^7 & \alpha^8 & \alpha^{11} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \alpha^2 & \alpha^8 & 1 & \alpha^{10} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & \alpha & \alpha^{11} & \alpha^{11} & \alpha^{12} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & \alpha^3 & \alpha^{11} & 1 & \alpha^9 \end{bmatrix}$$

$[n, k] = 15, 11$

"Extension Field" = 16

$C := Cyclic/BCH2$

***Getting information about the code:***

You can get the dimension, the length, the information positions and much more information about the code.' G' means the generator matrix
'Delta' the parameter of the BCH code, 'ExtField' the extension field which contains both alpha and beta, 'Roots' are the common positions
of zeros of the code.

```
> eval(C['Delta']);eval(C['k']);eval(C['ExtField']['seq']
  );eval(C['Roots']);
```

[4, 0]

11

$[0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}]$

[[0, 0], [0, 1], [0, 2], [0, 3]]

## encode a message and make two errors:

```
> c:=CyclicEncode([seq(1,i=1..11)],C);
```

$$c := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, \alpha^{14}, 0, \alpha^7, \alpha^{10}]$$

```
> C['Field']['seq'];
```

$$[0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}]$$

```
> r:=evalm(c):
  R1:=rand(1..C['n']):
  R2:=rand(0..14):
  for i from 1 to 2 do
   p:=R1();print(p);r[p]:=c[p]+alpha^R2() : od:print(r);
>
```

$$7$$

$$13$$

$$[1, 1, 1, 1, 1, 1, 1 + \alpha^5, 1, 1, 1, 1, \alpha^{14}, \alpha^8, \alpha^7, \alpha^{10}]$$

## decode the message

```
> BCHDecode(r,C);
```

"Syndrom array" =

$$[\alpha^9, \alpha^{13}, 0, \alpha, ?_{1,5}, ?_{1,6}, ?_{1,7}, ?_{1,8}, ?_{1,9}, ?_{1,10}, ?_{1,11}, ?_{1,12}, ?_{1,13}, ?_{1,14}, ?_{1,15}]$$

"Sakata's algorithm:"

$$\text{"u"} = [0, 0], \text{"F"} = [y, x], \text{"G"} = [1]$$

$$\text{"u"} = [1, 0], \text{"F"} = [y, x + 1], \text{"G"} = [1]$$

$$\text{"u"} = [0, 1], \text{"F"} = [\alpha^4 + y, x + 1], \text{"G"} = [1]$$

$$\text{"u"} = [2, 0], \text{"F"} = [\alpha^4 + y, x + 1], \text{"G"} = [1]$$

$$\text{"u"} = [1, 1], \text{"F"} = [\alpha^4 + y, x + 1], \text{"G"} = [1]$$

$$\text{"u"} = [0, 2], \text{"F"} = [\alpha^8 + \alpha^4 y + y^2, x + 1], \text{"G"} = [\alpha^4 + y]$$

$$\text{"u"} = [3, 0], \text{"F"} = [\alpha^8 + \alpha^4 y + y^2, x + 1], \text{"G"} = [\alpha^4 + y]$$

$$\text{"u"} = [2, 1], \text{"F"} = [\alpha^8 + \alpha^4 y + y^2, x + 1], \text{"G"} = [\alpha^4 + y]$$

$$\text{"u"} = [1, 2], \text{"F"} = [\alpha^8 + \alpha^4 y + y^2, x + 1], \text{"G"} = [\alpha^4 + y]$$

$$\text{"u"} = [0, 3], \text{"F"} = [\alpha^3 + \alpha^{14} y + y^2, x + 1], \text{"G"} = [\alpha^4 + y]$$

$$\text{"u"} = [4, 0], \text{"F"} = [\alpha^3 + \alpha^{14} y + y^2, x + 1], \text{"G"} = [\alpha^4 + y]$$

"Minimal set generating the syndromarrayup to the b" = [4, 0]

"F" = $[\alpha^3 + \alpha^{14} y + y^2, x + 1]$

"common roots of F" = [[0, 6], [0, 12]]

"error vector" = $[0, 0, 0, 0, 0, 0, \alpha^5, 0, 0, 0, 0, 0, \alpha^8, 0, 0]$

"sent codeword" = $[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, \alpha^{14}, 0, \alpha^7, \alpha^{10}]$

[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]

# Example: 2D-BCH Codes

## Loading files:

```
> restart;with(casa);
```

```
          |__|
          |  |          Welcome to CASA 2.4.3 for Maple V.5
     |   /\|  |/\
    /=\__|  []  |
   /            \_      Copyright (C) 1990-1999 by Research Institute
   |              \     for Symbolic Computation (RISC-Linz), the
   \   CASA 2.4.3 |     University of Linz, A-4040 Linz, Austria.
    |             |
   _|     |||     |     For help type '?casa' or '?casa,<topic>'.
 __/      |||     |_
```

[*BCH2*, *BCHDecode*, *CyclicEncode*, *DivAbsDeg*, *DivAdd*, *DivBasisL*, *DivDeg*, *DivGe*, *DivGeNeg*, *DivLe*, *DivPretty*, *DivSub*, *DivSupp*, *Genus*, *GoppaDecode*, *GoppaEncode*, *GoppaFunctional*, *GoppaPrepareDu*, *GoppaPrepareSV*, *GoppaPrepareSa*, *GoppaPrimary*, *InPolynomial*, *NormalPolynomial*, *OutPolynomial*, *PolynomialRoots*, *RatEv*, *RatMapEv*, *RatMul*, *RatOrder*, *RatOrders*, *RatOrdersRat*, *RatPower*, *RationalPoints*, *SakataDecode*, *SubsPolynomial*, *finiteCurve*, *finiteField*, *finiteRational*, *init*, *makeDivisor*, *mapOutPolynomial*, *mapSubsPolynomial*]

## determining the infolevel:

```
> infolevel['casa/finite']:=10;
```

$$infolevel_{casa/finite} := 10$$

## generate a 2 error correcting code:

### *generate the field over which the code should work*

```
> GF4:=finiteField(4);
```

$$GF4 := Field$$

### *generate the code (delta>=(4,0))*

```
> C:=BCH2(7,7,[4,0],GF4);
```

"Common roots" = [[0, 0], [1, 0], [4, 0], [2, 0], [0, 1], [0, 4], [0, 2], [1, 1], [4, 4], [2, 2], [3, 0], [5, 0], [6, 0], [2, 1], [1, 4], [4, 2], [1, 2], [4, 1], [2, 4], [0, 3], [0, 5], [0, 6]]

$$[n, k] = 49, 27$$

"Extension Field" = 64

$$C := Cyclic/BCH2$$

## encode a message and make two errors:

```
> c:=CyclicEncode([seq(1,i=1..27)],C);
```

$c$ := [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```
> C['Field']['seq'];
```

$$[0, 1, \alpha, \alpha^2]$$

```
> r:=evalm(c):
  R1:=rand(1..C['n']):
  R2:=rand(0..2):
  for i from 1 to 2 do
   p:=R1();print(p);r[p]:=c[p]+alpha^R2(): od:print(r);
>
```

1

19

[$1 + \alpha^2$, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, $1 + \alpha$, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

## decode the message

```
> m:=BCHDecode(r,C);
```

$$\text{"Syndrom array"} = \begin{bmatrix} 1 & \alpha^{44} & \alpha^{11} & \alpha^{4} & \alpha^{50} & \alpha^{16} & \alpha \\ \alpha^{50} & \alpha^{16} & \alpha & ?_{2,4} & \alpha^{44} & ?_{2,6} & ?_{2,7} \\ \alpha^{44} & \alpha^{11} & \alpha^{4} & ?_{3,4} & \alpha^{16} & ?_{3,6} & ?_{3,7} \\ \alpha^{16} & ?_{4,2} & ?_{4,3} & ?_{4,4} & ?_{4,5} & ?_{4,6} & ?_{4,7} \\ \alpha^{11} & \alpha^{4} & \alpha^{50} & ?_{5,4} & \alpha & ?_{5,6} & ?_{5,7} \\ \alpha & ?_{6,2} & ?_{6,3} & ?_{6,4} & ?_{6,5} & ?_{6,6} & ?_{6,7} \\ \alpha^{4} & ?_{7,2} & ?_{7,3} & ?_{7,4} & ?_{7,5} & ?_{7,6} & ?_{7,7} \end{bmatrix}$$

"Sakata's algorithm:"

$$\text{"u"} = [0, 0], \text{"F"} = [y, x], \text{"G"} = [1]$$

$$\text{"u"} = [1, 0], \text{"F"} = [y, \alpha^{50} + x], \text{"G"} = [1]$$

$$\text{"u"} = [0, 1], \text{"F"} = [y + \alpha^{44}, \alpha^{50} + x], \text{"G"} = [1]$$

$$\text{"u"} = [2, 0], \text{"F"} = [y + \alpha^{44}, \alpha^{54} + x^2 + \alpha^{50} x], \text{"G"} = [\alpha^{50} + x]$$

$$\text{"u"} = [1, 1], \text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{54} + x^2 + \alpha^{50} x], \text{"G"} = [\alpha^{50} + x]$$

$$\text{"u"} = [0, 2], \text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{54} + x^2 + \alpha^{50} x], \text{"G"} = [\alpha^{50} + x]$$

$$\text{"u"} = [3, 0], \text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{18} + x^2 + \alpha^{27} x], \text{"G"} = [\alpha^{50} + x]$$

$$\text{"u"} = [2, 1], \text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{18} + x^2 + \alpha^{27} x], \text{"G"} = [\alpha^{50} + x]$$

$$\text{"u"} = [1, 2], \text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{18} + x^2 + \alpha^{27} x], \text{"G"} = [\alpha^{50} + x]$$

$$\text{"u"} = [0, 3], \text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{18} + x^2 + \alpha^{27} x], \text{"G"} = [\alpha^{50} + x]$$

$$\text{"u"} = [4, 0], \text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{18} + x^2 + \alpha^{27} x], \text{"G"} = [\alpha^{50} + x]$$

"Minimal set generating the syndromarrayup to the b" = [4, 0]

$$\text{"F"} = [y + \alpha^{18} + \alpha^{27} x, \alpha^{18} + x^2 + \alpha^{27} x]$$

"common roots of F" = [[0, 0], [2, 4]]

"error vector" = [$\alpha^2$, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, $\alpha$, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

"sent codeword" = [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]

$$m := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

```
>
```

# Example: Hermitian Code

In this example we construct a 7-error correcting Hermitian code over $H_4$.

## Initialization

```
>
> restart;with(casa);
```

```
              |__|
              |  |          Welcome to CASA 2.4.3 for Maple V.5
       |    /\|  |/\
      /=\__|  []  |
     /           \_         Copyright (C) 1990-1999 by Research Institute
     |             \        for Symbolic Computation (RISC-Linz), the
     \   CASA 2.4.3 |       University of Linz, A-4040 Linz, Austria.
      |             |
     _|     |||     |       For help type '?casa' or '?casa,<topic>'.
  __/       |||     |_
```

[BCH2, BCHDecode, CyclicEncode, DivAbsDeg, DivAdd, DivBasisL, DivDeg, DivGe, DivGeNeg, DivLe, DivPretty, DivSub, DivSupp, Genus, GoppaDecode, GoppaEncode, GoppaFunctional, GoppaPrepareDu, GoppaPrepareSV, GoppaPrepareSa, GoppaPrimary, InPolynomial, NormalPolynomial, OutPolynomial, PolynomialRoots, RatEv, RatMapEv, RatMul, RatOrder, RatOrders, RatOrdersRat, RatPower, RationalPoints, SakataDecode, SubsPolynomial, finiteCurve, finiteField, finiteRational, init, makeDivisor, mapOutPolynomial, mapSubsPolynomial]

## Construction of the field, the curve and the code

```
> infolevel[`casa/finite`]:=9;
```

$$infolevel_{casa/finite} := 9$$

```
> F16:=finiteField(16);
```

$$F16 := Field$$

```
> F16['In'](alpha);
```

$$\mathrm{RootOf}(\_Z^4 + \_Z + 1)$$

```
> H4:=finiteCurve(x^5+y^4+y,F16);
```

$$H4 := x^5 + y\,z^4 + y^4\,z$$

```
> g:=Genus(H4);
```

$$g := 6$$

```
> C:= GoppaPrimary(H4,"affine",25);
```

$$G = +25\ (0{:}1{:}0)$$

$$\begin{aligned} D =\ & [\alpha^5, \alpha^{14}, 1] + [\alpha^6, \alpha^2, 1] + [\alpha^2, \alpha^{14}, 1] + [\alpha^3, \alpha^8, 1] + [\alpha^2, \alpha^{12}, 1] \\ & + [1, \alpha, 1] + [1, \alpha^2, 1] + [\alpha^{13}, \alpha^7, 1] + [\alpha^5, \alpha^{12}, 1] + [\alpha, \alpha^6, 1] + [\alpha^2, \alpha^3, 1] \\ & + [\alpha^3, \alpha, 1] + [\alpha, \alpha^9, 1] + [\alpha, \alpha^7, 1] + [\alpha^{12}, \alpha^8, 1] + [\alpha^2, \alpha^{11}, 1] + [\alpha, \alpha^{13}, 1] \\ & + [\alpha^{13}, \alpha^{13}, 1] + [\alpha^{13}, \alpha^9, 1] + [0, \alpha^5, 1] + [\alpha^{11}, \alpha^{11}, 1] + [\alpha^{12}, \alpha^4, 1] + [0, 1, 1] \\ & + [0, \alpha^{10}, 1] + [1, \alpha^4, 1] + [\alpha^{13}, \alpha^6, 1] + [\alpha^{11}, \alpha^3, 1] + [\alpha^{10}, \alpha^{13}, 1] \\ & + [\alpha^{10}, \alpha^6, 1] + [\alpha^{11}, \alpha^{14}, 1] + [\alpha^6, \alpha^4, 1] + [\alpha^7, \alpha^{13}, 1] + [\alpha^6, \alpha^8, 1] \\ & + [\alpha^{12}, \alpha, 1] + [\alpha^{12}, \alpha^2, 1] + [\alpha^{11}, \alpha^{12}, 1] + [\alpha^9, \alpha, 1] + [\alpha^5, \alpha^3, 1] \\ & + [\alpha^{10}, \alpha^9, 1] + [\alpha^9, \alpha^8, 1] + [\alpha^8, \alpha^3, 1] + [\alpha^8, \alpha^{14}, 1] + [\alpha^4, \alpha^{13}, 1] \\ & + [\alpha^{14}, \alpha^{11}, 1] + [\alpha^{14}, \alpha^{12}, 1] + [\alpha^{14}, \alpha^{14}, 1] + [\alpha^8, \alpha^{11}, 1] + [0, 0, 1] \\ & + [\alpha^7, \alpha^6, 1] + [\alpha^5, \alpha^{11}, 1] + [\alpha^7, \alpha^9, 1] + [\alpha^3, \alpha^2, 1] + [\alpha^3, \alpha^4, 1] + [1, \alpha^8, 1] \\ & + [\alpha^4, \alpha^7, 1] + [\alpha^4, \alpha^9, 1] + [\alpha^9, \alpha^2, 1] + [\alpha^4, \alpha^6, 1] + [\alpha^7, \alpha^7, 1] + [\alpha^{14}, \alpha^3, 1] \\ & + [\alpha^6, \alpha, 1] + [\alpha^9, \alpha^4, 1] + [\alpha^8, \alpha^{12}, 1] + [\alpha^{10}, \alpha^7, 1] \end{aligned}$$

$$L(G) = \operatorname{span}\left(1, \frac{y}{z}, \frac{y^2}{z^2}, \frac{y^3}{z^3}, \frac{y^4}{z^4}, \frac{y^5}{z^5}, \frac{x}{z}, \frac{x\,y}{z^2}, \frac{x\,y^2}{z^3}, \frac{x\,y^3}{z^4}, \frac{x\,y^4}{z^5}, \frac{x^2}{z^2}, \frac{x^2\,y}{z^3}, \frac{x^2\,y^2}{z^4}, \frac{x^2\,y^3}{z^5}, \frac{x^3}{z^3}, \frac{x^3\,y}{z^4}, \frac{x^3\,y^2}{z^5}, \frac{x^4}{z^4}, \frac{x^4\,y}{z^5}\right)$$

$$[n, k, d_\Gamma] = 64, 44, 15$$

$$C := Goppa/primary$$

## Encode a message an make 7 errors

```
> w:=GoppaEncode([seq(1,i=1..C['k'])],C);
```

$$w := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$

```
>
> R2:=rand(1..C['n']):
  R3:=rand(0..15):
  for i from 1 to 7 do
     r:=R2();w[r]:=w[r]+alpha^R3(); od: print(w);
```

```
w:=
vector([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+alpha^13, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+alpha^5, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1+alpha^14, 1, 1, 1, 1, 1+alpha^5, 1, 1,
1+alpha^6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+alpha^5,
1+alpha^5, 1, 1]);
```

$$w := [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^{13}, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^5, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^{14}, 1, 1, 1, 1, 1+\alpha^5, 1, 1, 1+\alpha^6, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1+\alpha^5, 1+\alpha^5, 1, 1]$$

## Decoding the message using Sakata's approach

First we heve to initialize the decoding procedure.

```
>
> H:=GoppaPrepareSa(C):
```

$$\text{"a"} = 4$$

$$\text{"The corresponding function"} = \frac{x}{z}$$

$$\text{"b"} = 5$$

$$\text{"The corresponding function"} = \frac{y}{z}$$

$$\frac{x^5}{z^5} = \frac{y}{z} + \frac{y^4}{z^4}$$

$$\frac{x^5\, y}{z^6} = \frac{y^2}{z^2} + \frac{y^5}{z^5}$$

$$\frac{x^5\, y^2}{z^7} = \frac{y^3}{z^3} + \frac{y^6}{z^6}$$

$$\frac{x^5\, y^3}{z^8} = \frac{y^7}{z^7} + \frac{y^4}{z^4}$$

$$\frac{x^6}{z^6} = \frac{x\, y}{z^2} + \frac{x\, y^4}{z^5}$$

$$\frac{x^6\, y}{z^7} = \frac{x\, y^2}{z^3} + \frac{x\, y^5}{z^6}$$

$$\frac{x^6 y^2}{z^8} = \frac{x y^3}{z^4} + \frac{x y^6}{z^7}$$

$$\frac{x^6 y^3}{z^9} = \frac{x y^4}{z^5} + \frac{x y^7}{z^8}$$

$$\frac{x^7}{z^7} = \frac{x^2 y}{z^3} + \frac{x^2 y^4}{z^6}$$

$$\frac{x^7 y}{z^8} = \frac{x^2 y^5}{z^7} + \frac{x^2 y^2}{z^4}$$

$$\frac{x^7 y^2}{z^9} = \frac{x^2 y^6}{z^8} + \frac{x^2 y^3}{z^5}$$

$$\frac{x^8}{z^8} = \frac{x^3 y^4}{z^7} + \frac{x^3 y}{z^4}$$

$$\frac{x^8 y}{z^9} = \frac{x^3 y^5}{z^8} + \frac{x^3 y^2}{z^5}$$

$$\frac{x^9}{z^9} = \frac{x^4 y}{z^5} + \frac{x^4 y^4}{z^8}$$

"result" =

$$\begin{bmatrix} 1 & \frac{x}{z} & \frac{x^2}{z^2} & \frac{x^3}{z^3} & \frac{x^4}{z^4} & \frac{x^5}{z^5} & \frac{x^6}{z^6} & \frac{x^7}{z^7} & \frac{x^8}{z^8} & \frac{x^9}{z^9} \\ \frac{y}{z} & \frac{x\,y}{z^2} & \frac{x^2\,y}{z^3} & \frac{x^3\,y}{z^4} & \frac{x^4\,y}{z^5} & \frac{x^5\,y}{z^6} & \frac{x^6\,y}{z^7} & \frac{x^7\,y}{z^8} & \frac{x^8\,y}{z^9} & \frac{x^9\,y}{z^{10}} \\ \frac{y^2}{z^2} & \frac{x\,y^2}{z^3} & \frac{x^2\,y^2}{z^4} & \frac{x^3\,y^2}{z^5} & \frac{x^4\,y^2}{z^6} & \frac{x^5\,y^2}{z^7} & \frac{x^6\,y^2}{z^8} & \frac{x^7\,y^2}{z^9} & \frac{x^8\,y^2}{z^{10}} & \frac{x^9\,y^2}{z^{11}} \\ \frac{y^3}{z^3} & \frac{x\,y^3}{z^4} & \frac{x^2\,y^3}{z^5} & \frac{x^3\,y^3}{z^6} & \frac{x^4\,y^3}{z^7} & \frac{x^5\,y^3}{z^8} & \frac{x^6\,y^3}{z^9} & \frac{x^7\,y^3}{z^{10}} & \frac{x^8\,y^3}{z^{11}} & \frac{x^9\,y^3}{z^{12}} \\ \frac{y^4}{z^4} & \frac{x\,y^4}{z^5} & \frac{x^2\,y^4}{z^6} & \frac{x^3\,y^4}{z^7} & \frac{x^4\,y^4}{z^8} & \frac{x^5\,y^4}{z^9} & \frac{x^6\,y^4}{z^{10}} & \frac{x^7\,y^4}{z^{11}} & \frac{x^8\,y^4}{z^{12}} & \frac{x^9\,y^4}{z^{13}} \\ \frac{y^5}{z^5} & \frac{x\,y^5}{z^6} & \frac{x^2\,y^5}{z^7} & \frac{x^3\,y^5}{z^8} & \frac{x^4\,y^5}{z^9} & \frac{x^5\,y^5}{z^{10}} & \frac{x^6\,y^5}{z^{11}} & \frac{x^7\,y^5}{z^{12}} & \frac{x^8\,y^5}{z^{13}} & \frac{x^9\,y^5}{z^{14}} \\ \frac{y^6}{z^6} & \frac{x\,y^6}{z^7} & \frac{x^2\,y^6}{z^8} & \frac{x^3\,y^6}{z^9} & \frac{x^4\,y^6}{z^{10}} & \frac{x^5\,y^6}{z^{11}} & \frac{x^6\,y^6}{z^{12}} & \frac{x^7\,y^6}{z^{13}} & \frac{x^8\,y^6}{z^{14}} & \frac{x^9\,y^6}{z^{15}} \\ \frac{y^7}{z^7} & \frac{x\,y^7}{z^8} & \frac{x^2\,y^7}{z^9} & \frac{x^3\,y^7}{z^{10}} & \frac{x^4\,y^7}{z^{11}} & \frac{x^5\,y^7}{z^{12}} & \frac{x^6\,y^7}{z^{13}} & \frac{x^7\,y^7}{z^{14}} & \frac{x^8\,y^7}{z^{15}} & \frac{x^9\,y^7}{z^{16}} \end{bmatrix}$$

$$\mathrm{order}(\text{"result"}) = \begin{bmatrix} 0 & 4 & 8 & 12 & 16 & 20 & 24 & 28 & 32 & 36 \\ 5 & 9 & 13 & 17 & 21 & 25 & 29 & 33 & 37 & 41 \\ 10 & 14 & 18 & 22 & 26 & 30 & 34 & 38 & 42 & 46 \\ 15 & 19 & 23 & 27 & 31 & 35 & 39 & 43 & 47 & 51 \\ 20 & 24 & 28 & 32 & 36 & 40 & 44 & 48 & 52 & 56 \\ 25 & 29 & 33 & 37 & 41 & 45 & 49 & 53 & 57 & 61 \\ 30 & 34 & 38 & 42 & 46 & 50 & 54 & 58 & 62 & 66 \\ 35 & 39 & 43 & 47 & 51 & 55 & 59 & 63 & 67 & 71 \end{bmatrix}$$

Now we use Sakata's approach for decoding the received word w. We use the Extended Algorithm.

```
> readlib(profile);profile(SakataDecode);
  SakataDecode(w,C,H);
```

**proc**() ... **end**

$$\text{"S"} = \begin{bmatrix} \alpha^3 & \alpha^{11} & \alpha^9 & \alpha^{14} & \alpha^{13} & \alpha^7 & \alpha^2 & S_{8,1} & S_{9,1} \\ \alpha^9 & \alpha^7 & \alpha^7 & \alpha^3 & 0 & 1 & S_{7,2} & & \\ \alpha^{14} & \alpha^{11} & \alpha^5 & \alpha^4 & S_{5,3} & S_{6,3} & & & \\ \alpha & \alpha^{12} & \alpha^{13} & S_{4,4} & S_{5,4} & & & & \\ 1 & \alpha^{12} & S_{3,5} & S_{4,5} & & & & & \\ \alpha^3 & S_{2,6} & & & & & & & \\ S_{1,7} & & & & & & & & \\ & & & & & & & & \end{bmatrix}$$

"Sakata's algorithm for known syndroms:"

"Minimal set generating the syndromarray up to" = 25

$$\text{"F"} = [\alpha^9 x^2 + x^3 + \alpha^4 + \alpha^7 x + \alpha^4 y^2 x + \alpha^3 y x^2 + \alpha^7 y^2 + \alpha^2 y x + y^3,$$
$$\alpha^{13} x^2 + \alpha^{12} x^3 + \alpha^{12} + \alpha^{12} x + \alpha^2 y + y^2 x + \alpha^4 y x^2 + \alpha^4 y^2 + \alpha^3 y x,$$
$$\alpha^{14} x^2 + \alpha^8 x^3 + \alpha^{13} + \alpha x + \alpha^{12} y + y x^2 + \alpha^9 y^2 + \alpha^8 y x,$$
$$\alpha^3 x^2 + x^3 + x^4 + \alpha^6 x + \alpha^2 y + \alpha y^2 x + \alpha^3 y x^2 + \alpha^7 y x]$$

"Starting majority voting for sydroms of order" = 26

"The points of this order are" = [ [4, 2] ]

$$\text{"votes of the polynomialf"} = \alpha^9 x^2 + x^3 + \alpha^4 + \alpha^7 x + \alpha^4 y^2 x + \alpha^3 y x^2 + \alpha^7 y^2 + \alpha^2 y x + y^3$$

"votes of the polynomialf" =

$$\alpha^{13} x^2 + \alpha^{12} x^3 + \alpha^{12} + \alpha^{12} x + \alpha^2 y + y^2 x + \alpha^4 y x^2 + \alpha^4 y^2 + \alpha^3 y x$$

"Kf" = { }

$$\text{"vote of f"} = \alpha^5$$

$$\text{"votes of the polynomialf"} = \alpha^{14} x^2 + \alpha^8 x^3 + \alpha^{13} + \alpha x + \alpha^{12} y + y x^2 + \alpha^9 y^2 + \alpha^8 y x$$

"Kf" = { [2, 1] }

"vote of f" = 0

$$\text{"votes of the polynomialf"} = \alpha^3 x^2 + x^3 + x^4 + \alpha^6 x + \alpha^2 y + \alpha y^2 x + \alpha^3 y x^2 + \alpha^7 y x$$

"Kf" = { }

$$\text{"vote of f"} = \alpha^5$$

$$\text{"[votes,multiplicities]"} = ([\alpha^5, 0], [0, 1])$$

$$\text{"[votes,newpowerproducts]"} = ([\alpha^5, \{\ \}], [0, \{[2, 1]\}])$$

$$\text{"S"}_{5,3} = 0$$

"Running Sakata's algorithm for the new known syndroms"

"Minimal set generating the syndromarray up to the order" = 26

$$"F" = [\alpha^9 x^2 + x^3 + \alpha^4 + \alpha^7 x + \alpha^4 y^2 x + \alpha^3 y x^2 + \alpha^7 y^2 + \alpha^2 y x + y^3,$$
$$\alpha^{11} x^2 + \alpha^6 x^3 + \alpha^{12} + \alpha^8 x + \alpha y + y^2 x + \alpha^4 y x^2 + \alpha^8 y^2 + \alpha^4 y x,$$
$$\alpha^{14} x^2 + \alpha^8 x^3 + \alpha^{13} + \alpha x + \alpha^{12} y + y x^2 + \alpha^9 y^2 + \alpha^8 y x,$$
$$\alpha^9 x^2 + x^3 + x^4 + \alpha + x + \alpha^5 y + \alpha y^2 x + \alpha^3 y x^2 + \alpha^4 y^2 + \alpha^{11} y x]$$

"Starting majority voting for sydroms of order" = 27

"The points of this order are" = [[3, 3]]

$$"[votes,multiplicities]" = ([\alpha^{11}, 0], [\alpha^2, 2])$$

$$"S"_{4,4} = \alpha^2$$

"Running Sakata's algorithm for the new known syndroms"

"Minimal set generating the syndromarray up to the order" = 27

$$"F" = [\alpha^{12} x^2 + \alpha^2 x^3 + \alpha^4 + \alpha^5 x + \alpha^9 y + \alpha^4 y^2 x + \alpha^3 y x^2 + y^2 + \alpha^9 y x + y^3,$$
$$\alpha^{11} x^2 + \alpha^6 x^3 + \alpha^{12} + \alpha^8 x + \alpha y + y^2 x + \alpha^4 y x^2 + \alpha^8 y^2 + \alpha^4 y x,$$
$$\alpha^{14} x^2 + \alpha^8 x^3 + \alpha^{13} + \alpha x + \alpha^{12} y + y x^2 + \alpha^9 y^2 + \alpha^8 y x,$$
$$\alpha^9 x^2 + x^3 + x^4 + \alpha + x + \alpha^5 y + \alpha y^2 x + \alpha^3 y x^2 + \alpha^4 y^2 + \alpha^{11} y x]$$

"Starting majority voting for sydroms of order" = 28

"The points of this order are" = [[7, 0], [2, 4]]

$$"[votes,multiplicities]" = ([\alpha^9, 3], [\alpha^{14}, 0])$$

$$"S"_{8,1} = 1$$

$$"S"_{3,5} = \alpha^9$$

"Running Sakata's algorithm for the new known syndroms"

"Minimal set generating the syndromarray up to the order" = 28

$$"F" = [\alpha^{12} x^2 + \alpha^2 x^3 + \alpha^4 + \alpha^5 x + \alpha^9 y + \alpha^4 y^2 x + \alpha^3 y x^2 + y^2 + \alpha^9 y x + y^3,$$
$$\alpha^{11} x^2 + \alpha^6 x^3 + \alpha^{12} + \alpha^8 x + \alpha y + y^2 x + \alpha^4 y x^2 + \alpha^8 y^2 + \alpha^4 y x,$$
$$\alpha^{14} x^2 + \alpha^8 x^3 + \alpha^{13} + \alpha x + \alpha^{12} y + y x^2 + \alpha^9 y^2 + \alpha^8 y x,$$
$$\alpha x^2 + \alpha^{14} x^3 + x^4 + \alpha + \alpha^2 x + \alpha^8 y + \alpha y^2 x + \alpha^3 y x^2 + \alpha y x]$$

"Starting majority voting for sydroms of order" = 29

"The points of this order are" = [[6, 1], [1, 5]]

$$"[votes,multiplicities]" = ([\alpha^{10}, 4])$$

"Starting majority voting for sydroms of order" = 30

"The points of this order are" = [[5, 2], [0, 6]]

$$"[votes,multiplicities]" = ([\alpha^{11}, 5])$$

"Starting majority voting for sydroms of order" = 31

"The points of this order are" = [[4, 3]]

"[votes,multiplicities]" = ([$\alpha^{12}$, 6])

"Starting majority voting for sydroms of order" = 32

"The points of this order are" = [[8, 0], [3, 4]]

"[votes,multiplicities]" = ([$\alpha^{10}$, 7])

$$\text{"S"} = \begin{bmatrix} \alpha^3 & \alpha^{11} & \alpha^9 & \alpha^{14} & \alpha^{13} & \alpha^7 & \alpha^2 & 1 & \alpha^{12} \\ \alpha^9 & \alpha^7 & \alpha^7 & \alpha^3 & 0 & 1 & \alpha^{14} & & \\ \alpha^{14} & \alpha^{11} & \alpha^5 & \alpha^4 & 0 & \alpha^6 & & & \\ \alpha & \alpha^{12} & \alpha^{13} & \alpha^2 & \alpha^{12} & & & & \\ 1 & \alpha^{12} & \alpha^9 & \alpha^{10} & & & & & \\ \alpha^3 & \alpha^{10} & & & & & & & \\ \alpha^{11} & & & & & & & & \\ & & & & & & & & \end{bmatrix}$$

"poss. error positions" = [1, 2, 3, 4, 11, 18, 27, 35, 39, 41, 44, 47, 61, 62]

*Error Value System:*

$$
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & \alpha & \alpha^{3} & \alpha^{5} & \alpha^{7} & \alpha^{8} & \alpha^{9} & \alpha^{9} & \alpha^{10} & \alpha^{14} & \alpha^{14} \\
0 & 1 & \alpha^{5} & \alpha^{10} & \alpha^{9} & \alpha^{2} & \alpha^{12} & \alpha^{9} & \alpha^{12} & \alpha & \alpha^{8} & \alpha^{9} & \alpha^{3} & \alpha^{11} \\
0 & 0 & 0 & 0 & \alpha^{2} & \alpha^{6} & \alpha^{10} & \alpha^{14} & \alpha & \alpha^{3} & \alpha^{3} & \alpha^{5} & \alpha^{13} & \alpha^{13} \\
0 & 0 & 0 & 0 & \alpha^{10} & \alpha^{5} & \alpha^{2} & \alpha & \alpha^{5} & \alpha^{10} & \alpha^{2} & \alpha^{4} & \alpha^{2} & \alpha^{10} \\
0 & 1 & \alpha^{10} & \alpha^{5} & \alpha^{3} & \alpha^{4} & \alpha^{9} & \alpha^{3} & \alpha^{9} & \alpha^{2} & \alpha & \alpha^{3} & \alpha^{6} & \alpha^{7} \\
0 & 0 & 0 & 0 & \alpha^{3} & \alpha^{9} & 1 & \alpha^{6} & \alpha^{9} & \alpha^{12} & \alpha^{12} & 1 & \alpha^{12} & \alpha^{12} \\
0 & 0 & 0 & 0 & \alpha^{11} & \alpha^{8} & \alpha^{7} & \alpha^{8} & \alpha^{13} & \alpha^{4} & \alpha^{11} & \alpha^{14} & \alpha & \alpha^{9} \\
0 & 0 & 0 & 0 & \alpha^{4} & \alpha^{7} & \alpha^{14} & \alpha^{10} & \alpha^{2} & \alpha^{11} & \alpha^{10} & \alpha^{13} & \alpha^{5} & \alpha^{6} \\
0 & 1 & 1 & 1 & \alpha^{12} & \alpha^{6} & \alpha^{6} & \alpha^{12} & \alpha^{6} & \alpha^{3} & \alpha^{9} & \alpha^{12} & \alpha^{9} & \alpha^{3} \\
0 & 0 & 0 & 0 & \alpha^{4} & \alpha^{12} & \alpha^{5} & \alpha^{13} & \alpha^{2} & \alpha^{6} & \alpha^{6} & \alpha^{10} & \alpha^{11} & \alpha^{11} \\
0 & 0 & 0 & 0 & \alpha^{12} & \alpha^{11} & \alpha^{12} & 1 & \alpha^{6} & \alpha^{13} & \alpha^{5} & \alpha^{9} & 1 & \alpha^{8} \\
0 & 0 & 0 & 0 & \alpha^{5} & \alpha^{10} & \alpha^{4} & \alpha^{2} & \alpha^{10} & \alpha^{5} & \alpha^{4} & \alpha^{8} & \alpha^{4} & \alpha^{5} \\
0 & 0 & 0 & 0 & \alpha^{13} & \alpha^{9} & \alpha^{11} & \alpha^{4} & \alpha^{14} & \alpha^{12} & \alpha^{3} & \alpha^{7} & \alpha^{8} & \alpha^{2} \\
0 & 1 & \alpha^{5} & \alpha^{10} & \alpha^{6} & \alpha^{8} & \alpha^{3} & \alpha^{6} & \alpha^{3} & \alpha^{4} & \alpha^{2} & \alpha^{6} & \alpha^{12} & \alpha^{14} \\
0 & 0 & 0 & 0 & \alpha^{13} & \alpha^{14} & \alpha^{2} & \alpha^{7} & \alpha^{14} & \alpha^{7} & \alpha^{14} & \alpha^{4} & \alpha^{14} & \alpha^{7} \\
0 & 0 & 0 & 0 & \alpha^{6} & \alpha^{13} & \alpha^{9} & \alpha^{9} & \alpha^{3} & \alpha^{14} & \alpha^{13} & \alpha^{3} & \alpha^{3} & \alpha^{4} \\
0 & 0 & 0 & 0 & \alpha^{14} & \alpha^{12} & \alpha & \alpha^{11} & \alpha^{7} & \alpha^{6} & \alpha^{12} & \alpha^{2} & \alpha^{7} & \alpha \\
0 & 0 & 0 & 0 & \alpha^{7} & \alpha^{11} & \alpha^{8} & \alpha^{13} & \alpha^{11} & \alpha^{13} & \alpha^{11} & \alpha & \alpha^{11} & \alpha^{13} \\
0 & 1 & \alpha^{10} & \alpha^{5} & 1 & \alpha^{10} & 1 & 1 & 1 & \alpha^{5} & \alpha^{10} & 1 & 1 & \alpha^{10} \\
0 & 0 & 0 & 0 & \alpha^{7} & \alpha & \alpha^{14} & \alpha & \alpha^{11} & \alpha^{8} & \alpha^{7} & \alpha^{13} & \alpha^{2} & \alpha^{3} \\
0 & 0 & 0 & 0 & 1 & 1 & \alpha^{6} & \alpha^{3} & 1 & 1 & \alpha^{6} & \alpha^{12} & \alpha^{6} & 1 \\
0 & 0 & 0 & 0 & \alpha^{8} & \alpha^{14} & \alpha^{13} & \alpha^{5} & \alpha^{4} & \alpha^{7} & \alpha^{5} & \alpha^{11} & \alpha^{10} & \alpha^{12} \\
0 & 0 & 0 & 0 & \alpha & \alpha^{13} & \alpha^{5} & \alpha^{7} & \alpha^{8} & \alpha^{14} & \alpha^{4} & \alpha^{10} & \alpha^{14} & \alpha^{9} \\
0 & 1 & 1 & 1 & \alpha^{9} & \alpha^{12} & \alpha^{12} & \alpha^{9} & \alpha^{12} & \alpha^{6} & \alpha^{3} & \alpha^{9} & \alpha^{3} & \alpha^{6} \\
0 & 0 & 0 & 0 & \alpha & \alpha^{3} & \alpha^{11} & \alpha^{10} & \alpha^{8} & \alpha^{9} & 1 & \alpha^{7} & \alpha^{5} & \alpha^{14} \\
0 & 0 & 0 & 0 & \alpha^{9} & \alpha^{2} & \alpha^{3} & \alpha^{12} & \alpha^{12} & \alpha & \alpha^{14} & \alpha^{6} & \alpha^{9} & \alpha^{11}
\end{bmatrix}
* e =
\begin{bmatrix}
\alpha^{3} \\ \alpha^{11} \\ \alpha^{9} \\ \alpha^{9} \\ \alpha^{7} \\ \alpha^{14} \\ \alpha^{14} \\ \alpha^{7} \\ \alpha^{11} \\ \alpha \\ \alpha^{13} \\ \alpha^{3} \\ \alpha^{5} \\ \alpha^{12} \\ 1 \\ 0 \\ \alpha^{4} \\ \alpha^{13} \\ \alpha^{12} \\ \alpha^{3} \\ 0 \\ \alpha^{2} \\ \alpha^{9} \\ \alpha^{10} \\ \alpha^{11} \\ \alpha^{12} \\ \alpha^{10}
\end{bmatrix}
$$

$$e = [0, 0, 0, 0, \alpha^{13}, 0, \alpha^{5}, 0, \alpha^{14}, 0, \alpha^{5}, \alpha^{6}, \alpha^{5}, \alpha^{5}]$$

$c = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,$
$1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$

$[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,$
$1, 1, 1, 1, 1, 1, 1]$

Thus we succeeded in correcting the 7 errors. We now apply Duursma's approach to the same received word w, in order to be able to compare these two methods.
First we have to calculate the initialization steps.

```
> H2:=GoppaPrepareDu(C):
```

$$G + gQ = +31\ (0:1:0)$$

$$L(G+gQ) = \operatorname{span}\left(1, \frac{x}{z}, \frac{y}{z}, \frac{x^2}{z^2}, \frac{x\,y}{z^2}, \frac{y^2}{z^2}, \frac{x^3}{z^3}, \frac{x^2\,y}{z^3}, \frac{x\,y^2}{z^3}, \frac{y^3}{z^3}, \frac{x^4}{z^4}, \frac{x^3\,y}{z^4}, \frac{x^2\,y^2}{z^4}, \frac{x\,y^3}{z^4}, \frac{y^4}{z^4}, \frac{x^4\,y}{z^5}, \frac{x^3\,y^2}{z^5}, \frac{x^2\,y^3}{z^5}, \frac{x\,y^4}{z^5}, \frac{y^5}{z^5}, \frac{x^4\,y^2}{z^6}, \frac{x^3\,y^3}{z^6}, \frac{x^2\,y^4}{z^6}, \frac{x\,y^5}{z^6}, \frac{y^6}{z^6}, \frac{x^4\,y^3}{z^7}\right)$$

$\psi * \chi =$

$$
\begin{bmatrix}
1\,, & \frac{x}{z}\,, & \frac{y}{z}\,, & \frac{x^2}{z^2}\,, & \frac{x\,y}{z^2}\,, & \frac{y^2}{z^2}\,, & \frac{x^3}{z^3}\,, & \frac{x^2\,y}{z^3}\,, & \frac{x\,y^2}{z^3}\,, & \frac{y^3}{z^3}\,, & \frac{x^4}{z^4}\,, & \frac{x^3\,y}{z^4}\,, & \frac{x^2\,y^2}{z^4} \\
\frac{x}{z}\,, & \frac{x^2}{z^2}\,, & \frac{x\,y}{z^2}\,, & \frac{x^3}{z^3}\,, & \frac{x^2\,y}{z^3}\,, & \frac{x\,y^2}{z^3}\,, & \frac{x^4}{z^4}\,, & \frac{x^3\,y}{z^4}\,, & \frac{x^2\,y^2}{z^4}\,, & \frac{x\,y^3}{z^4}\,, & \frac{x^5}{z^5}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^3\,y^2}{z^5} \\
\frac{y}{z}\,, & \frac{x\,y}{z^2}\,, & \frac{y^2}{z^2}\,, & \frac{x^2\,y}{z^3}\,, & \frac{x\,y^2}{z^3}\,, & \frac{y^3}{z^3}\,, & \frac{x^3\,y}{z^4}\,, & \frac{x^2\,y^2}{z^4}\,, & \frac{x\,y^3}{z^4}\,, & \frac{y^4}{z^4}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^2\,y^3}{z^5} \\
\frac{x^2}{z^2}\,, & \frac{x^3}{z^3}\,, & \frac{x^2\,y}{z^3}\,, & \frac{x^4}{z^4}\,, & \frac{x^3\,y}{z^4}\,, & \frac{x^2\,y^2}{z^4}\,, & \frac{x^5}{z^5}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^2\,y^3}{z^5}\,, & \frac{x^6}{z^6}\,, & \frac{x^5\,y}{z^6}\,, & \frac{x^4\,y^2}{z^6} \\
\frac{x\,y}{z^2}\,, & \frac{x^2\,y}{z^3}\,, & \frac{x\,y^2}{z^3}\,, & \frac{x^3\,y}{z^4}\,, & \frac{x^2\,y^2}{z^4}\,, & \frac{x\,y^3}{z^4}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^2\,y^3}{z^5}\,, & \frac{x\,y^4}{z^5}\,, & \frac{x^5\,y}{z^6}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^3\,y^3}{z^6} \\
\frac{y^2}{z^2}\,, & \frac{x\,y^2}{z^3}\,, & \frac{y^3}{z^3}\,, & \frac{x^2\,y^2}{z^4}\,, & \frac{x\,y^3}{z^4}\,, & \frac{y^4}{z^4}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^2\,y^3}{z^5}\,, & \frac{x\,y^4}{z^5}\,, & \frac{y^5}{z^5}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^3\,y^3}{z^6}\,, & \frac{x^2\,y^4}{z^6} \\
\frac{x^3}{z^3}\,, & \frac{x^4}{z^4}\,, & \frac{x^3\,y}{z^4}\,, & \frac{x^5}{z^5}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^6}{z^6}\,, & \frac{x^5\,y}{z^6}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^3\,y^3}{z^6}\,, & \frac{x^7}{z^7}\,, & \frac{x^6\,y}{z^7}\,, & \frac{x^5\,y^2}{z^7} \\
\frac{x^2\,y}{z^3}\,, & \frac{x^3\,y}{z^4}\,, & \frac{x^2\,y^2}{z^4}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^2\,y^3}{z^5}\,, & \frac{x^5\,y}{z^6}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^3\,y^3}{z^6}\,, & \frac{x^2\,y^4}{z^6}\,, & \frac{x^6\,y}{z^7}\,, & \frac{x^5\,y^2}{z^7}\,, & \frac{x^4\,y^3}{z^7} \\
\frac{x\,y^2}{z^3}\,, & \frac{x^2\,y^2}{z^4}\,, & \frac{x\,y^3}{z^4}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^2\,y^3}{z^5}\,, & \frac{x\,y^4}{z^5}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^3\,y^3}{z^6}\,, & \frac{x^2\,y^4}{z^6}\,, & \frac{x\,y^5}{z^6}\,, & \frac{x^5\,y^2}{z^7}\,, & \frac{x^4\,y^3}{z^7}\,, & \\
\frac{y^3}{z^3}\,, & \frac{x\,y^3}{z^4}\,, & \frac{y^4}{z^4}\,, & \frac{x^2\,y^3}{z^5}\,, & \frac{x\,y^4}{z^5}\,, & \frac{y^5}{z^5}\,, & \frac{x^3\,y^3}{z^6}\,, & \frac{x^2\,y^4}{z^6}\,, & \frac{x\,y^5}{z^6}\,, & \frac{y^6}{z^6}\,, & \frac{x^4\,y^3}{z^7}\,, & , & \\
\frac{x^4}{z^4}\,, & \frac{x^5}{z^5}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^6}{z^6}\,, & \frac{x^5\,y}{z^6}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^7}{z^7}\,, & \frac{x^6\,y}{z^7}\,, & \frac{x^5\,y^2}{z^7}\,, & \frac{x^4\,y^3}{z^7}\,, & , & , & \\
\frac{x^3\,y}{z^4}\,, & \frac{x^4\,y}{z^5}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^5\,y}{z^6}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^3\,y^3}{z^6}\,, & \frac{x^6\,y}{z^7}\,, & \frac{x^5\,y^2}{z^7}\,, & \frac{x^4\,y^3}{z^7}\,, & , & , & , & \\
\frac{x^2\,y^2}{z^4}\,, & \frac{x^3\,y^2}{z^5}\,, & \frac{x^2\,y^3}{z^5}\,, & \frac{x^4\,y^2}{z^6}\,, & \frac{x^3\,y^3}{z^6}\,, & \frac{x^2\,y^4}{z^6}\,, & \frac{x^5\,y^2}{z^7}\,, & \frac{x^4\,y^3}{z^7}\,, & , & , & , & , &
\end{bmatrix}
$$

$$\mathrm{order}(\psi * \chi) = \begin{bmatrix} 0 & 4 & 5 & 8 & 9 & 10 & 12 & 13 & 14 & 15 & 16 & 17 & 18 \\ 4 & 8 & 9 & 12 & 13 & 14 & 16 & 17 & 18 & 19 & 20 & 21 & 22 \\ 5 & 9 & 10 & 13 & 14 & 15 & 17 & 18 & 19 & 20 & 21 & 22 & 23 \\ 8 & 12 & 13 & 16 & 17 & 18 & 20 & 21 & 22 & 23 & 24 & 25 & 26 \\ 9 & 13 & 14 & 17 & 18 & 19 & 21 & 22 & 23 & 24 & 25 & 26 & 27 \\ 10 & 14 & 15 & 18 & 19 & 20 & 22 & 23 & 24 & 25 & 26 & 27 & 28 \\ 12 & 16 & 17 & 20 & 21 & 22 & 24 & 25 & 26 & 27 & 28 & 29 & 30 \\ 13 & 17 & 18 & 21 & 22 & 23 & 25 & 26 & 27 & 28 & 29 & 30 & 31 \\ 14 & 18 & 19 & 22 & 23 & 24 & 26 & 27 & 28 & 29 & 30 & 31 & \\ 15 & 19 & 20 & 23 & 24 & 25 & 27 & 28 & 29 & 30 & 31 & & \\ 16 & 20 & 21 & 24 & 25 & 26 & 28 & 29 & 30 & 31 & & & \\ 17 & 21 & 22 & 25 & 26 & 27 & 29 & 30 & 31 & & & & \\ 18 & 22 & 23 & 26 & 27 & 28 & 30 & 31 & & & & & \end{bmatrix}$$

$$\frac{x^5}{z^5} = \frac{y}{z} + \frac{y^4}{z^4}$$

$$\frac{x^6}{z^6} = \frac{x\,y}{z^2} + \frac{x\,y^4}{z^5}$$

$$\frac{x^5\,y}{z^6} = \frac{y^2}{z^2} + \frac{y^5}{z^5}$$

$$\frac{x^7}{z^7} = \frac{x^2\,y}{z^3} + \frac{x^2\,y^4}{z^6}$$

$$\frac{x^6\,y}{z^7} = \frac{x\,y^2}{z^3} + \frac{x\,y^5}{z^6}$$

$$\frac{x^5\,y^2}{z^7} = \frac{y^3}{z^3} + \frac{y^6}{z^6}$$

Now we apply Duursma's approach to decoding the received word w.

```
> profile(GoppaDecode);GoppaDecode(w,C,H2);
```

$$S=\begin{bmatrix}
\alpha^3 & \alpha^{11} & \alpha^9 & \alpha^9 & \alpha^7 & \alpha^{14} & \alpha^{14} & \alpha^7 & \alpha^{11} & \alpha & \alpha^{13} & \alpha^3 & \alpha^5 \\
\alpha^{11} & \alpha^9 & \alpha^7 & \alpha^{14} & \alpha^7 & \alpha^{11} & \alpha^{13} & \alpha^3 & \alpha^5 & \alpha^{12} & \alpha^7 & 0 & \alpha^4 \\
\alpha^9 & \alpha^7 & \alpha^{14} & \alpha^7 & \alpha^{11} & \alpha & \alpha^3 & \alpha^5 & \alpha^{12} & 1 & 0 & \alpha^4 & \alpha^{13} \\
\alpha^9 & \alpha^{14} & \alpha^7 & \alpha^{13} & \alpha^3 & \alpha^5 & \alpha^7 & 0 & \alpha^4 & \alpha^{13} & \alpha^2 & 1 & ?_{4,13} \\
\alpha^7 & \alpha^7 & \alpha^{11} & \alpha^3 & \alpha^5 & \alpha^{12} & 0 & \alpha^4 & \alpha^{13} & \alpha^{12} & 1 & ?_{5,12} & ?_{5,13} \\
\alpha^{14} & \alpha^{11} & \alpha & \alpha^5 & \alpha^{12} & 1 & \alpha^4 & \alpha^{13} & \alpha^{12} & \alpha^3 & ?_{6,11} & ?_{6,12} & ?_{6,13} \\
\alpha^{14} & \alpha^{13} & \alpha^3 & \alpha^7 & 0 & \alpha^4 & \alpha^2 & 1 & ?_{7,9} & ?_{7,10} & ?_{7,11} & ?_{7,12} & ?_{7,13} \\
\alpha^7 & \alpha^3 & \alpha^5 & 0 & \alpha^4 & \alpha^{13} & 1 & ?_{8,8} & ?_{8,9} & ?_{8,10} & ?_{8,11} & ?_{8,12} & ?_{8,13} \\
\alpha^{11} & \alpha^5 & \alpha^{12} & \alpha^4 & \alpha^{13} & \alpha^{12} & ?_{9,7} & ?_{9,8} & ?_{9,9} & ?_{9,10} & ?_{9,11} & ?_{9,12} & \\
\alpha & \alpha^{12} & 1 & \alpha^{13} & \alpha^{12} & \alpha^3 & ?_{10,7} & ?_{10,8} & ?_{10,9} & ?_{10,10} & ?_{10,11} & & \\
\alpha^{13} & \alpha^7 & 0 & \alpha^2 & 1 & ?_{11,6} & ?_{11,7} & ?_{11,8} & ?_{11,9} & ?_{11,10} & & & \\
\alpha^3 & 0 & \alpha^4 & 1 & ?_{12,5} & ?_{12,6} & ?_{12,7} & ?_{12,8} & ?_{12,9} & & & & \\
\alpha^5 & \alpha^4 & \alpha^{13} & ?_{13,4} & ?_{13,5} & ?_{13,6} & ?_{13,7} & ?_{13,8} & & & & &
\end{bmatrix}$$

$$e * \phi = [\alpha^3, \alpha^{11}, \alpha^9, \alpha^9, \alpha^7, \alpha^{14}, \alpha^{14}, \alpha^7, \alpha^{11}, \alpha, \alpha^{13}, \alpha^3, \alpha^5, \alpha^{12}, 1, 0, \alpha^4, \alpha^{13}, \alpha^{12}, \alpha^3, ?_{21}, ?_{22}, ?_{23}, ?_{24}, ?_{25}, ?_{26}]$$

*estimating entries of order* $s = 26$

*no row-candidate for entry* $_{4,\,13}$

*no row-candidate for entry* $_{5,\,12}$

*no row-candidate for entry* $_{6,\,11}$

*no row-candidate for entry* $_{7,\,9}$

$$\textit{row-candidate for } S_{8,8} = \frac{x^2\,y}{z^3} + \alpha^{13} + \frac{\alpha\,x}{z} + \frac{\alpha^{12}\,y}{z} + \frac{\alpha^{14}\,x^2}{z^2} + \frac{\alpha^8\,x\,y}{z^2} + \frac{\alpha^9\,y^2}{z^2} + \frac{\alpha^8\,x^3}{z^3}$$

$$\textit{col-candidate for } S_{8,8} = \frac{x^2\,y}{z^3} + \alpha^{13} + \frac{\alpha\,x}{z} + \frac{\alpha^{12}\,y}{z} + \frac{\alpha^{14}\,x^2}{z^2} + \frac{\alpha^8\,x\,y}{z^2} + \frac{\alpha^9\,y^2}{z^2} + \frac{\alpha^8\,x^3}{z^3}$$

*row, col-estimate for* $S_{8,8} = 0, 0$

*no col-candidate for entry* $S_{9,7}$

*no col-candidate for entry* $S_{11,6}$

*no col-candidate for entry* $S_{12,5}$

*no col-candidate for entry* $S_{13,4}$

*votes* $= ([0, 1])$

*most frequent vote* $= 0$

$$\mathit{new\ entry}\ S_{4,\,13} = 0$$

$$\mathit{new\ entry}\ S_{5,\,12} = 0$$

$$\mathit{new\ entry}\ S_{6,\,11} = 0$$

$$\mathit{new\ entry}\ S_{7,\,9} = 0$$

$$\mathit{new\ entry}\ S_{8,\,8} = 0$$

$$\mathit{new\ entry}\ S_{9,\,7} = 0$$

$$\mathit{new\ entry}\ S_{11,\,6} = 0$$

$$\mathit{new\ entry}\ S_{12,\,5} = 0$$

$$\mathit{new\ entry}\ S_{13,\,4} = 0$$

$$\mathit{estimating\ entries\ of\ order}\ s = 27$$

$$\mathit{estimating\ entries\ of\ order}\ s = 28$$

$$\mathit{estimating\ entries\ of\ order}\ s = 29$$

$$\mathit{estimating\ entries\ of\ order}\ s = 30$$

$$\mathit{estimating\ entries\ of\ order}\ s = 31$$

$$S = \begin{bmatrix}
\alpha^{3} & \alpha^{11} & \alpha^{9} & \alpha^{9} & \alpha^{7} & \alpha^{14} & \alpha^{14} & \alpha^{7} & \alpha^{11} & \alpha & \alpha^{13} & \alpha^{3} & \alpha^{5} \\
\alpha^{11} & \alpha^{9} & \alpha^{7} & \alpha^{14} & \alpha^{7} & \alpha^{11} & \alpha^{13} & \alpha^{3} & \alpha^{5} & \alpha^{12} & \alpha^{7} & 0 & \alpha^{4} \\
\alpha^{9} & \alpha^{7} & \alpha^{14} & \alpha^{7} & \alpha^{11} & \alpha & \alpha^{3} & \alpha^{5} & \alpha^{12} & 1 & 0 & \alpha^{4} & \alpha^{13} \\
\alpha^{9} & \alpha^{14} & \alpha^{7} & \alpha^{13} & \alpha^{3} & \alpha^{5} & \alpha^{7} & 0 & \alpha^{4} & \alpha^{13} & \alpha^{2} & 1 & 0 \\
\alpha^{7} & \alpha^{7} & \alpha^{11} & \alpha^{3} & \alpha^{5} & \alpha^{12} & 0 & \alpha^{4} & \alpha^{13} & \alpha^{12} & 1 & 0 & \alpha^{2} \\
\alpha^{14} & \alpha^{11} & \alpha & \alpha^{5} & \alpha^{12} & 1 & \alpha^{4} & \alpha^{13} & \alpha^{12} & \alpha^{3} & 0 & \alpha^{2} & \alpha^{9} \\
\alpha^{14} & \alpha^{13} & \alpha^{3} & \alpha^{7} & 0 & \alpha^{4} & \alpha^{2} & 1 & 0 & \alpha^{2} & 1 & \alpha^{14} & \alpha^{6} \\
\alpha^{7} & \alpha^{3} & \alpha^{5} & 0 & \alpha^{4} & \alpha^{13} & 1 & 0 & \alpha^{2} & \alpha^{9} & \alpha^{14} & \alpha^{6} & \alpha^{12} \\
\alpha^{11} & \alpha^{5} & \alpha^{12} & \alpha^{4} & \alpha^{13} & \alpha^{12} & 0 & \alpha^{2} & \alpha^{9} & \alpha^{10} & \alpha^{6} & \alpha^{12} & \\
\alpha & \alpha^{12} & 1 & \alpha^{13} & \alpha^{12} & \alpha^{3} & \alpha^{2} & \alpha^{9} & \alpha^{10} & \alpha^{11} & \alpha^{12} & & \\
\alpha^{13} & \alpha^{7} & 0 & \alpha^{2} & 1 & 0 & 1 & \alpha^{14} & \alpha^{6} & \alpha^{12} & & & \\
\alpha^{3} & 0 & \alpha^{4} & 1 & 0 & \alpha^{2} & \alpha^{14} & \alpha^{6} & \alpha^{12} & & & & \\
\alpha^{5} & \alpha^{4} & \alpha^{13} & 0 & \alpha^{2} & \alpha^{9} & \alpha^{6} & \alpha^{12} & & & & &
\end{bmatrix}$$

$$e * \phi = [\alpha^{3}, \alpha^{11}, \alpha^{9}, \alpha^{9}, \alpha^{7}, \alpha^{14}, \alpha^{14}, \alpha^{7}, \alpha^{11}, \alpha, \alpha^{13}, \alpha^{3}, \alpha^{5}, \alpha^{12}, 1, 0, \alpha^{4}, \alpha^{13}, \alpha^{12}, \alpha^{3}, 0, \alpha^{2}, \alpha^{9}, \alpha^{10}, \alpha^{11}, \alpha^{12}]$$

$$\mathit{Nullspace(S)} = ([\alpha^{13}, \alpha, \alpha^{12}, \alpha^{14}, \alpha^{8}, \alpha^{9}, \alpha^{8}, 1])$$

$$\mathit{theta} = \alpha^{13} + \frac{\alpha x}{z} + \frac{\alpha^{12} y}{z} + \frac{\alpha^{14} x^{2}}{z^{2}} + \frac{\alpha^{8} x y}{z^{2}} + \frac{\alpha^{9} y^{2}}{z^{2}} + \frac{\alpha^{8} x^{3}}{z^{3}} + \frac{x^{2} y}{z^{3}}$$

$$\textit{error locations} = \{\, 11, 18, 27, 35, 39, 41, 44, 47, 61, 62 \,\}$$

$$\textit{Error Value System:}\quad \begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\alpha & \alpha^{3} & \alpha^{5} & \alpha^{7} & \alpha^{8} & \alpha^{9} & \alpha^{9} & \alpha^{10} & \alpha^{14} & \alpha^{14} \\
\alpha^{9} & \alpha^{2} & \alpha^{12} & \alpha^{9} & \alpha^{12} & \alpha & \alpha^{8} & \alpha^{9} & \alpha^{3} & \alpha^{11} \\
\alpha^{2} & \alpha^{6} & \alpha^{10} & \alpha^{14} & \alpha & \alpha^{3} & \alpha^{3} & \alpha^{5} & \alpha^{13} & \alpha^{13} \\
\alpha^{10} & \alpha^{5} & \alpha^{2} & \alpha & \alpha^{5} & \alpha^{10} & \alpha^{2} & \alpha^{4} & \alpha^{2} & \alpha^{10} \\
\alpha^{3} & \alpha^{4} & \alpha^{9} & \alpha^{3} & \alpha^{9} & \alpha^{2} & \alpha & \alpha^{3} & \alpha^{6} & \alpha^{7} \\
\alpha^{3} & \alpha^{9} & 1 & \alpha^{6} & \alpha^{9} & \alpha^{12} & \alpha^{12} & 1 & \alpha^{12} & \alpha^{12} \\
\alpha^{11} & \alpha^{8} & \alpha^{7} & \alpha^{8} & \alpha^{13} & \alpha^{4} & \alpha^{11} & \alpha^{14} & \alpha & \alpha^{9} \\
\alpha^{4} & \alpha^{7} & \alpha^{14} & \alpha^{10} & \alpha^{2} & \alpha^{11} & \alpha^{10} & \alpha^{13} & \alpha^{5} & \alpha^{6} \\
\alpha^{12} & \alpha^{6} & \alpha^{6} & \alpha^{12} & \alpha^{6} & \alpha^{3} & \alpha^{9} & \alpha^{12} & \alpha^{9} & \alpha^{3} \\
\alpha^{4} & \alpha^{12} & \alpha^{5} & \alpha^{13} & \alpha^{2} & \alpha^{6} & \alpha^{6} & \alpha^{10} & \alpha^{11} & \alpha^{11} \\
\alpha^{12} & \alpha^{11} & \alpha^{12} & 1 & \alpha^{6} & \alpha^{13} & \alpha^{5} & \alpha^{9} & 1 & \alpha^{8} \\
\alpha^{5} & \alpha^{10} & \alpha^{4} & \alpha^{2} & \alpha^{10} & \alpha^{5} & \alpha^{4} & \alpha^{8} & \alpha^{4} & \alpha^{5} \\
\alpha^{13} & \alpha^{9} & \alpha^{11} & \alpha^{4} & \alpha^{14} & \alpha^{12} & \alpha^{3} & \alpha^{7} & \alpha^{8} & \alpha^{2} \\
\alpha^{6} & \alpha^{8} & \alpha^{3} & \alpha^{6} & \alpha^{3} & \alpha^{4} & \alpha^{2} & \alpha^{6} & \alpha^{12} & \alpha^{14} \\
\alpha^{13} & \alpha^{14} & \alpha^{2} & \alpha^{7} & \alpha^{14} & \alpha^{7} & \alpha^{14} & \alpha^{4} & \alpha^{14} & \alpha^{7} \\
\alpha^{6} & \alpha^{13} & \alpha^{9} & \alpha^{9} & \alpha^{3} & \alpha^{14} & \alpha^{13} & \alpha^{3} & \alpha^{3} & \alpha^{4} \\
\alpha^{14} & \alpha^{12} & \alpha & \alpha^{11} & \alpha^{7} & \alpha^{6} & \alpha^{12} & \alpha^{2} & \alpha^{7} & \alpha \\
\alpha^{7} & \alpha^{11} & \alpha^{8} & \alpha^{13} & \alpha^{11} & \alpha^{13} & \alpha^{11} & \alpha & \alpha^{11} & \alpha^{13} \\
1 & \alpha^{10} & 1 & 1 & 1 & \alpha^{5} & \alpha^{10} & 1 & 1 & \alpha^{10} \\
\alpha^{7} & \alpha & \alpha^{14} & \alpha & \alpha^{11} & \alpha^{8} & \alpha^{7} & \alpha^{13} & \alpha^{2} & \alpha^{3} \\
1 & 1 & \alpha^{6} & \alpha^{3} & 1 & 1 & \alpha^{6} & \alpha^{12} & \alpha^{6} & 1 \\
\alpha^{8} & \alpha^{14} & \alpha^{13} & \alpha^{5} & \alpha^{4} & \alpha^{7} & \alpha^{5} & \alpha^{11} & \alpha^{10} & \alpha^{12} \\
\alpha & \alpha^{13} & \alpha^{5} & \alpha^{7} & \alpha^{8} & \alpha^{14} & \alpha^{4} & \alpha^{10} & \alpha^{14} & \alpha^{9} \\
\alpha^{9} & \alpha^{12} & \alpha^{12} & \alpha^{9} & \alpha^{12} & \alpha^{6} & \alpha^{3} & \alpha^{9} & \alpha^{3} & \alpha^{6} \\
\alpha & \alpha^{3} & \alpha^{11} & \alpha^{10} & \alpha^{8} & \alpha^{9} & 1 & \alpha^{7} & \alpha^{5} & \alpha^{14}
\end{bmatrix} * e = \begin{bmatrix}
\alpha^{3} \\ \alpha^{11} \\ \alpha^{9} \\ \alpha^{9} \\ \alpha^{7} \\ \alpha^{14} \\ \alpha^{14} \\ \alpha^{7} \\ \alpha^{11} \\ \alpha \\ \alpha^{13} \\ \alpha^{3} \\ \alpha^{5} \\ \alpha^{12} \\ 1 \\ 0 \\ \alpha^{4} \\ \alpha^{13} \\ \alpha^{12} \\ \alpha^{3} \\ 0 \\ \alpha^{2} \\ \alpha^{9} \\ \alpha^{10} \\ \alpha^{11} \\ \alpha^{12}
\end{bmatrix}$$

$$e = [\,\alpha^{13}, 0, \alpha^{5}, 0, \alpha^{14}, 0, \alpha^{5}, \alpha^{6}, \alpha^{5}, \alpha^{5}\,]$$

$c = [\,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,$
$1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1\,]$

$[\,1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,$
$1, 1, 1, 1, 1, 1, 1\,]$

Please remark the fact that the number of votes and the vote for the independent syndroms is equal in both methods. Moreover you can see, that the row candidate of $S_{8,8}$ is an element in F. This is not an incident, the elements of the actual F are candidates for the row and column candidates.

We now compare the time, used for decoding. In this case Sakata's approach is approximately 3 times as fast as Duursma's approach.

```
> showprofile(GoppaDecode,SakataDecode);
function           depth    calls     time    time%         bytes   bytes%
---------------------------------------------------------------------------
GoppaDecode            1        1  153.569    75.64     296681672    74.05
SakataDecode           1        1   49.470    24.36     103981468    25.95
---------------------------------------------------------------------------
total:                 2        2  203.039   100.00     400663140   100.00

>
```

# Bibliography


[Adá91] J. Adámek. *Foundations of Coding. Theory and Application of Error-Correcting Codes with an Introduction to Cryptography and Information Theory*, John Wiley and Sons (1991).

[Ber68] E. R. Berlekamp. *Algebraic Coding Theory*, McGraw-Hill, New York (1968).

[Buch65] B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*, Ph.D. Dissertation, Universität Innsbruck, Austria (1965).

[CRHT94] X. Chen, I. S. Reed, T. Helleseth, T. K. Truong. Use of Groebner Bases to Decode Binary Cyclic Codes up to the True Minimum Distance, *IEEE Transactions on Information Theory*, 40:1653-1663 (1994).

[Dei99] H. Deischinger. *Algebraic Geometric Codes*, Diploma thesis, JK-University Linz, Austria (1999).

[Duu93] I. M. Duursma. Majority Coset Decoding, *IEEE Transactions on Information Theory*, 39:1067-1070 (1993).

[FR93] G.-L. Feng, T. R. N. Rao. Decoding Algebraic-Geometric Codes up to the Designed Minimum Distance, *IEEE Transactions on Information Theory*, 39:37-45 (1993).

[Ful69] W. Fulton. *Algebraic Curves*, Mathematics Lecture Note Series, The Benjamin/Cummings Publishing Company (1969).

[HHW99] R. Hemmecke, E. Hillgarter, F. Winkler. The CASA System, In: *Handbook for Computer Algebra*, J. Grabmeier, E. Kaltofen, V. Weispfenning (eds.), Springer-Verlag (1999).

[HLP98] T. Høholdt, J. H. van Lint, R. Peelikan. Algebraic Geometry Codes, In: *Handbook of Coding Theory*, V. S. Pless, W. C. Huffman (eds.), Elsevier Science B. V. (1998).

[HLS95] C. Heegard, J. Little, K. Saints. Systematic Encoding via Groebner Bases for a Class of Algebraic-Geometric Goppa Codes, *IEEE Transactions on Information Theory*, 41:1752-1761 (1995).

[HP95] T. Høholdt, R. Pelikan. On the Decoding of Algebraic-Geometric Codes, *IEEE Transactions on Information Theory*, 41,No. 6:1589-1614 (1995).

[Imai77] H. Imai. A Theory of Two-Dimensional Cyclic Codes, *Information and Control*, 34:1-21 (1977).

[JLJH93] J. Justesen, J. Larsen, H. E. Jensen, T. Høholdt. Fast Decoding of Codes from Algebraic Plane Curves, *IEEE Transactions on Information Theory*, 38,No.1:111-119 (1993).

[Leo95] D. A. Leonhard. Error-Locator Ideals for Algebraic-Geometric Codes, *IEEE Transactions on Information Theory*, 41, No.3:819-824 (1995).

[Lit98] J. B. Little. Applications to Coding Theory, In: *Application of Computational Algebraic Geometry*, D. A. Cox, B. Sturmfeld (eds.), Proceedings of Symposia in Applied Mathematics Vol.53, American Mathematical Society, Providence, Rhode Island (1998).

[LP98] R. Lidl, G. Pilz. *Applied Abstract Algebra*, Undergraduate Texts in Mathematics, Springer-Verlag, second edition (1998).

[Mas69] J. L. Massey. Shift-Register Synthesis and BCH- Decoding, *IEEE Transactions on Information Theory*, 15,No.1:122-127 (1969).

[Mor91] C. Moreno. *Algebraic Curves over Finite Fields*, Cambridge University Press (1991).

[Pre98] O. Pretzel. *Codes and Algebraic Curves*, Number 8 in Oxford lecture series in mathematics and its applications, Oxford Science Publications (1998).

[Sak78] S. Sakata. General Theory of Doubly Periodic Arrays over an Arbitary Finite Field and Its Aplications, *IEEE Transactions on Information Theory*, 24,No.6:719-730 (1978).

[Sak81] S. Sakata. On Determining the Independent Point Set of Doubly Periodic Arrays and Encoding 2-Dimensional Cyclic Codes and Their Duals, *IEEE Transactions on Information Theory*, 27:556-565 (1981).

[Sak86] S. Sakata. Finding a Minimal Set of Linear Recurring Relations Capable of Generating a Given Finite Two-dimensional Array, *Journal of Symbolic Computation*, 5:321-337 (1988).

[Sak90] S. Sakata. Extension of the Berlekamp-Massey Algorithm to N Dimensions, *Information and Computation*, 84:207-239 (1990).

[Sak91] S. Sakata. Decoding Binary 2-D Codes by the 2-D Berlekamp-Massey Algorithm, *IEEE Transactions on Information Theory*, 37:1200-1203 (1991).

[Sak98] S. Sakata. Groebner Bases and Coding Theory, In: *Groebner Bases and Applications*, B. Buchberger, F. Winkler (eds.), London Mathematical Society Lecture Note Series, Cambridge University Press (1998).

[SH95] K. Saints, C. Heegard. Algebraic-Geometric Codes and Multidimensional Cyclic Codes: A Unified Theory and Algorithms for Decoding Using Groebner Bases, *IEEE Transactions on Information Theory*, 41:1733-1751 (1995).

[SJH95] S. Sakata, H. E. Jensen, T. Høholdt. Generalized Berlekamp-Massey Decoding of Algebraic-Geometric Codes up to Half the Feng-Rao Bound, *IEEE Transactions on Information Theory*, 41,No.5:1762-1768 (1995).

[SJMJH95a] S. Sakata, J. Justesen, Y. Madelung, H. E. Jensen, T. Høholdt. Fast Decoding of Algebraic-Geometric Codes up to the Designed Minimum Distance, *IEEE Transactions on Information Theory*, 41,No.5:1672-1677 (1995).

[SJMJH95b] S. Sakata, J. Justesen, Y. Madelung, H. E. Jensen, T. Høholdt. A Fast Decoding Method of AG Codes from Miura-Kamiya Curves $C_{ab}$ up to Half the Feng-Rao Bound, *Finite Fields and Their Applications*, I:83-101 (1995).

[Sug75] Y. Sugiyama, M. Kasahra, S. Hirasawa, T. Namekawa. A Method for Solving the Key Equation for Decoding Goppa Codes, *Information and Control*, 27:87-99 (1975).

[Sul95] M. E. O'Sullivan. Decoding of Codes Defined by a Single Point on a Curve, *IEEE Transactions on Information Theory*, 41,No.6:1709-1719 (1995).

[vL92] J. H. van Lint. *Introduction to Coding Theory*, Number 86 in Graduate Texts in Mathematics, Springer- Verlag, second edition (1992).

[Win96] F. Winkler. *Polynomial Algorithms in Computer Algebra*, Springer Verlag, Wien-New York (1996).

# Index