On the automorphism group of various Goppa codes

Submitted by Stephan Wesemeyer to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Mathematics in the Faculty of Science, August 1997.

I certify that all material in this thesis which is not my own work has been identified and that no material is included for which a degree has previously been conferred upon me.

. . . . . .

(Stephan Wesemeyer)

Contents

Maturity

A stationary sense... as I suppose,
I shall have, till my single body grows
    Inaccurate, tired;
Then I shall start to feel the backward pull
Take over, sickening and masterful -
    Some say, desired.
And this must be THE PRIME OF LIFE...I blink,
As if at pain; for it is pain, to think
    This pantomime
Of compensating act and counter-act,
Defeat and counterfeit, makes up, in fact,
    My ablest time.

(Philip Larkin)

Introduction

This thesis deals with the automorphism group of algebraic-geometric codes (AG codes), also known as Goppa codes after V.D. Goppa who first introduced them (see [Gop88]). Goppa realised that one can associate codes with certain divisors of algebraic function fields thus linking the relatively new branch of mathematics of error-correcting codes with the old discipline of algebraic geometry. This fruitful connection has been exploited ever since.

Many old codes were found to be algebraic-geometric codes themselves or subsets of AG codes (like the BCH-codes and Reed-Solomon codes, for example) thus allowing mathematicians to examine them afresh from an algebraic-geometric point of view. Also, and more importantly, many new codes were discovered whose characteristics were better in certain ways than other "old" codes.

In this thesis we approach algebraic geometry from an algebraic angle, i.e. via algebraic function fields, rather than from a geometric point of view. This has the advantage that we only need to assume some basic knowledge of algebraic field extensions as a foundation.

The core of the thesis is concerned with establishing that under certain conditions the automorphism group of AG codes is isomorphic to a certain subgroup of automorphisms of the associated function field. Stichtenoth proved this to be true for rational Goppa codes (see [Sti90]), i.e. codes associated with the rational function field. Xing then showed that similar results hold for a special class of elliptic codes [Xin95a] and Hermitian codes [Xin95b]. The methods we present in this thesis allow us to prove this isomorphism for a large class of elliptic and hyperelliptic codes, as well as another special class of function fields, and for certain codes associated with the Klein Quartic.

The first three chapters of the thesis give a concise introduction to the theory of error-correcting codes, function fields and the connection between the two. The results quoted in these chapters are all very well-known and most of the time the reader is referred to standard text books for their proofs. The intention was to remind the reader of the necessary theoretical background on which later results are based.

Chapter 4 revisits Stichtenoth's result for the genus 0 case (i.e. the rational function field case) and then introduces the main ideas which are used in the later chapters to prove similar results

for function fields of higher genus.

Chapters 5 to 8 use the methods developed in Chapter 4 to prove the desired isomorphism between the automorphism group of certain AG codes and the associated group of automorphisms of the underlying function field. In each chapter there are a few examples to illustrate the results. Chapter 7 uses methods introduced by Xing in [Xin95a] and combines them with the earlier results to obtain a slightly improved version of the result Xing presented in his paper.

Chapter 9 deals with the computational aspects of my work. Getting results in mathematics sometimes seems to involve quite a bit of psychology in the sense that one is more likely to persevere if one is convinced that a certain result ought to be true. In my case it did help that the empirical data I had collected with the help of my computer programs seemed to suggest that a result about the automorphism group of geometric Goppa codes similar to the one Stichtenoth proved in his paper [Sti90] should hold for hyperelliptic fields and the Klein Quartic as well.

Consequently, I was able to prove theorems that show that the generalised version of Stichtenoth's result holds for more cases than my computer routines could deal with. In fact, the type of codes that my program could handle were by no means codes with any practical application. Most of the time their dimension was less than 6 and their length was hardly ever longer than 20. Still, they provided me with enough data and confidence that we thought it worthwhile embarking on finding a proof.

The appendices contain printouts of my programs and a collection of examples of how to use them.

Remark. [Note to the reader] Readers who are well-versed in the fundamentals of both the theory of error-correcting codes and function fields can skim over the first three chapters glancing at them cursorily to familiarise themselves with the notation and go straight to Chapter 4.

Acknowledgements

I would like to thank my mother for giving me moral and financial support throughout my six years of studying in Britain without which I would have never been able to complete my studies.

I am also very grateful to my supervisor, John Cremona, for invaluable advice and patience with even the most trivial questions (and there were a few...).

Furthermore I would like to thank Ray and Jeremy for being two excellent office mates and friends who made my stay in Exeter highly enjoyable and whose computer knowledge I could not have done without.

Chapter 1

Coding Theory Background

1.1 Introduction

Coding theory is one of the more recent subjects to have made an entry into the world of mathematics. Its main aim is to help ensure that information can be sent from A to B without losing any of it. We shall be more formal about this later on. Transmitting data and making sure that it does not get distorted during transmission is of ever increasing importance. The Internet is one example of a medium where one has to make sure that huge blocks of information can be sent down the line without getting scrambled. The editor (emacs) which was used to write this thesis was downloaded from the Internet and if the program code had been corrupted only slightly on its way over the phone lines it would not have been possible to run it at all.

One of the most familiar examples of the use of coding theory is the encoding of music on a CD. Most people know from personal experience that on the old vinyl LP a single scratch can ruin the enjoyment of listening to that record because either the stylus jumps or there is an audible click, etc. However, most scratches on the surface of a CD do not affect the reproduction of the original sound stored on it. This is an example of coding theory at work. When the laser of a CD player comes across such a scratch, it will not be able to read the data original stored in that place. The error correction mechanism in a CD player, however, is still able to reconstruct from the distorted data the original information and the listener can still enjoy the divine beauty of Beethoven's ninth symphony, say.

Human language is another example of a code, even though we do not tend to think about it in that way. However, especially at dinner parties with a lot of background noise, two people are still capable of having a conversation although they might not catch every word their partner

says. This is essential due to the redundancy of the human language. The language contains many words that are not strictly necessary for the successful conveyance of information between people. However, this "padding" helps the listener to fill in the occasional word herself which was drowned by background noises like the ones encountered at such parties.

Written language is also a case in point. Spelling mistakes can be easily recognised and corrected by most readers. Only when the word is especially short and there are lots of similar looking words and more than one of those would make sense in the context will the reader be uncertain how to correct the misprint, e.g.: "Yesterdey I spoke to the bany manager". Did I speak to the *bank*, *band* or maybe *zany* manager ? But whatever he was, I spoke to him *yesterday*.

Yet another example, though probably less familiar, is the transmission of pictures of distant planets taken by satellites like the Voyager spacecraft. Because of the large distances involved the signal not only takes hours to reach earth but it is very weak on arrival having also been distorted by all sorts of cosmic radiation, magnetic interference, etc. Hence it is highly desirable that we are still capable of retrieving the original information despite the distortion of the signal as it would be very costly, if possible at all, to retransmit the data.

Having seen various examples of where codes are used we need to introduce some concepts and definition that will provide the subject with a sound theoretical footing.

As a general reference for coding theory the reader is referred to the standard text books by MacWilliams and Sloane [MS77], van Lint [vL82], and Roman [Rom92]. Our exposition in this chapter follows very closely that of Roman and van Lint.

1.2 Fundamentals

Obviously, when we want to transmit any sort of information we have to translate it into a format that we can send through the chosen channel of transmission. Usually the format is dictated by the channel itself, e.g. if we want to send data over the Internet we probably have to encode the data in some form of electronic signals whereas if we want to send data via a telephone we would use acoustic signals instead (that the telephone then translates the acoustic signals into electric ones does not concern us here). This leads to the following definitions :

Definition 1.2.1. *Let $\mathcal{A} = \{a_1, a_2, \ldots, a_q\}$ be a finite set of q distinct elements, called the* **code alphabet**.

In most applications $\mathcal{A}$ will be the finite field $\mathbb{F}_q$ of $q = p^r$ elements where p is a prime. Instead of giving a very general definition of a code we shall concentrate on so called block-codes.

Definition 1.2.2. *Let $\mathcal{A} = \{a_1, a_2, \ldots, a_q\}$ be a code alphabet and $\mathcal{A}^n$ be the set of all strings of length n over $\mathcal{A}$. Any non-empty subset C of $\mathcal{A}^n$ is called a* **q-ary block code**. *Each string in C is called a* **codeword**. *If $C \subset \mathcal{A}^n$ contains M codewords, then it is customary to say that C has* **length** *n and* **size** *M, or is an* **(n, M)-code**. *The* **rate** *of a q-ary (n, M)-code is*

$$R = \frac{\log_q M}{n} \tag{1.2.3}$$

Now we define what we mean by a communication channel. For our purposes it is sufficient to assume that a channel accepts codewords $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ from a code C of length n over a code alphabet $\mathcal{A}$. and outputs strings $\mathbf{d} = (d_0, d_1, \ldots, d_{n-1})$ of length n over the same code alphabet $\mathcal{A}$. Note that $\mathbf{d}$ need not be an element of our code C any more.

Definition 1.2.4. *A* **discrete memoryless channel** *consists of a code alphabet $\mathcal{A} = \{a_1, \ldots a_q\}$ and a set of* **channel probabilities**, *or* **transition probabilities**, *$p(a_j|a_i) \geq 0$ satisfying*

$$\sum_{j=1}^{q} p(a_j|a_i) = 1$$

for $1 \leq i \leq q$. Intuitively, we think of $p(a_j|a_i)$ as the probability that a_j is received, given that a_i is sent through the channel. Furthermore, if $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ and $\mathbf{d} = (d_0, d_1, \ldots, d_{n-1})$ are words of length n over $\mathcal{A}$, the probability $p(\mathbf{d}|\mathbf{c})$ that $\mathbf{d}$ is received, given that $\mathbf{c}$ is sent, is

$$p(\mathbf{d}|\mathbf{c}) = \prod_{j=0}^{n-1} p(d_i|c_i)$$

One of the most important discrete memoryless channels is the **binary symmetric channel**, which has a code alphabet $\{0, 1\}$ and channel probabilities

$$p(1|0) = p(0|1) = p \text{ and } p(0|0) = p(1|1) = 1 - p.$$

This leads to probably the most familiar diagram in coding theory.

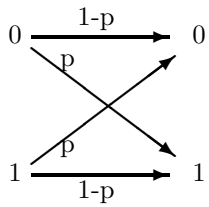


Figure 1.1: **A binary symmetric channel**

So what does it actually mean for an error to occur ? Obviously an error has occurred when $\mathbf{c}$ was input into the channel but $\mathbf{d} \neq \mathbf{c}$ has been received. We say that in this case a **word error**

has occurred.

A **decision scheme** is a function f from the set of output strings to set the of codewords including ∞, where ∞ represents the image of those output strings for which no corresponding codeword could be found. If we have a code C of length n over $\mathcal{A} = \{a_1, a_2, \dots, a_q\}$ then

$$f : \mathcal{A}^n \to C \cup \{\infty\}$$
$$d \mapsto f(d)$$

The idea behind this is that if $\mathbf{d}$ is received and $f(\mathbf{d}) \neq \infty$ then the decision scheme decides that $f(\mathbf{d})$ is the codeword that was sent. If $\infty \neq f(\mathbf{d})$ is not the codeword that was sent, we say that a **decision error**, or a **decoding error**, has been made.

To simplify matters we shall assume that for all $1 \leq i, j \leq q$

$$p(a_j|a_i) = \begin{cases} \frac{p}{q-1} & \text{if } j \neq i, \\ 1-p & \text{if } j = i. \end{cases}$$

This means that if an error occurs then it is independent of the a_i that was sent; thus any of the a_j with $j \neq i$ is as likely to be received as a result of an error as any other. Now it is easy to see that

$$(1-p)^n > (\frac{p}{q-1})(1-p)^{n-1} > (\frac{p}{q-1})^2(1-p)^{n-2} > \ldots > (\frac{p}{q-1})^{n-1}(1-p) > (\frac{p}{q-1})^n. \tag{1.2.5}$$

provided $\frac{p}{q-1} < (1-p) \iff p < \frac{q-1}{q}$. Obviously, any useful channel will have $p < \frac{1}{2}$, i.e. more than half of the time the transmission will be successful, otherwise no sensible communication is possible. (Note that the symmetric binary channel is an exception. If $p > \frac{1}{2}$ then $(1-p) < \frac{1}{2}$ and we assume that an error occurs every time and hence decode a received 1 as 0 and vice versa.) Supposing that $\mathbf{c} = (c_0, c_1, \dots, c_{n-1})$ was sent and $\mathbf{d} = (d_0, d_1, \dots, d_{n-1})$ received then a decision scheme based on (1.2.5) would decode $\mathbf{d}$ to be the codeword $\mathbf{c}'$ nearest to $\mathbf{d}$ in the sense that $\mathbf{c}'$ differs in as few positions as possible from $\mathbf{d}$. This concept leads to the definition of distance between codewords.

Definition 1.2.6. *Let $\mathcal{A} = \{a_1, a_2, \dots a_q\}$ be a code alphabet and $\mathbf{c}, \mathbf{c}' \in \mathcal{A}^n$ then the* **Hamming distance** *between $\mathbf{c} = (c_0, c_1, \dots, c_{n-1})$ and $\mathbf{c}' = (c'_0, c'_1, \dots, c'_{n-1})$ is defined to be*

$$d(\mathbf{c}, \mathbf{c}') = |\{i | c_i \neq c'_i \text{ for } 0 \leq i \leq n-1\}|.$$

It is immediate that d is a metric on $\mathcal{A}$ and thus $(\mathcal{A}^n, d)$ is a metric space. Supposing that d was received, our decision scheme to decode it can now be described as follows :

$$\text{Find } \mathbf{c} \in C \subset \mathcal{A}^n \text{ such that } d(\mathbf{c}, \mathbf{d}) = \min_{\mathbf{c}' \in C}\{d(\mathbf{c}', \mathbf{d})\}.$$

If there is more than one such $\mathbf{c}$ then we cannot decode $\mathbf{d}$; this is known as a **tie**. The decision scheme described above is known as **minimum distance decoding**.
Now assuming that we have an **encoder**, i.e. a means of translating a message m to a code word $\mathbf{c} \in C$, we have the following diagram :

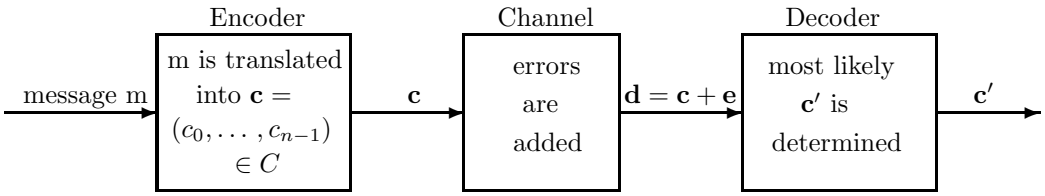


Figure 1.2: **The story so far**

Example 1.2.7. The most trivial example of a code is the binary repetition code. Here the information we want to transmit consists of the two values 0 and 1, i.e. $\mathcal{A} = \{0, 1\}$. Let $n > 0$ be odd.We associate with 1 (resp. 0) the codeword $(1, 1, \ldots, 1) \in \mathcal{A}^n$ (resp. $(0, 0, \ldots, 0) \in \mathcal{A}^n$). Now the most natural decision scheme in this case is the following :

$$f(\mathbf{d}) = \begin{cases} 1 \text{ if the majority of entries in } \mathbf{d} \text{ are ones} \\ 0 \text{ otherwise} \end{cases} .$$

(Note that we needed n to be odd to achieve the decoding of every possible $\mathbf{d}$.) The rate of this code is obviously rather poor. It is given by $R = \frac{1}{n}$. Nevertheless, if we assume a binary symmetric channel and an error probability of $p = 0.001$, say, then the probability that the decision scheme makes a decoding error is given by

$$\sum_{0 \le k < n/2} \binom{n}{k} (1-p)^k p^{n-k} \le p^{n/2}(1-p)^{n/2} \sum_{0 \le k < n/2} \binom{n}{k} < \big(p(1-p)\big)^{n/2} 2^n < (0.07)^n.$$

Hence as $n \to \infty$ this probability tends to 0, sadly so does the rate R of the channel.

On of the main aims in coding theory is to find codes which have a fairly high rate and allow the correction and/or detection of a reasonable number of errors. Shannon's Theorem, one of the milestones in coding theory, shows that this is not an illusive task (c.p. [vL82, pp22-29]).

1.3 t-Error-Correcting and t-Error-Detecting Codes

In implementing minimum distance decoding, the following concepts are useful.

Definition 1.3.1. *The* **minimum distance** *of a code C is defined to be*

$$d(C) = \min_{\substack{\mathbf{c},\mathbf{d}\in C \\ \mathbf{c}\neq\mathbf{d}}} d(\mathbf{c},\mathbf{d})$$

*An (n, M, d)-***code** *is a code of length n, size M, and minimum distance d.*

Note: In general, determining $d(C)$ is an arduous task because if C has M words one must check $\binom{M}{2}$ pairs of codewords to find it.

Definition 1.3.2. *A code C is t-***error-detecting** *if whenever at most t, but at least one, errors are made in a codeword, the resulting word is* not *a codeword. A code C is* **exactly** *t-***error-detecting** *if it is t-error-detecting but not $(t+1)$-error-detecting.*

We have the following result.

Theorem 1.3.3. *A code C is exactly t-error-detecting if and only if $d(C) = t + 1$.*

Proof.

$$\begin{aligned} d(C) = t+1 &\iff \exists\, \mathbf{c},\mathbf{d} \in C \text{ s.t. } d(\mathbf{c},\mathbf{d}) = t+1 \text{ and } \forall\, \mathbf{c}',\mathbf{d}' \in C \text{ with } \mathbf{c}' \neq \mathbf{d}'\ d(\mathbf{c}',\mathbf{d}') \geq t+1 \\ &\iff C \text{ is exactly } t\text{-error-detecting} \end{aligned}$$

□

If t errors occur in the transmission of a codeword, we will say that an error of **size** t has occurred.

Definition 1.3.4. *A code C is t-***error-correcting** *if minimum distance decoding is able to correct all errors of size t or less in an codeword, assuming that all ties are reported as errors. A code C is* **exactly** *t-***error-correcting** *if it is t-error-correcting but not $(t+1)$-error-correcting. (In other words, all errors of size t are corrected, but at least one error $t+1$ is decoded incorrectly.)*

Again there is a straightforward connection between the minimum distance of a code and its ability to correct t errors.

Theorem 1.3.5. *A code C is exactly t-error-correcting if and only if $d(C) = 2t+1$ or $d(C) = 2t+2$.*

Proof. obvious. □

As an immediate corollary we have:

Corollary 1.3.6. *$d(C) = d$ if and only if C is exactly $[(d-1)/2]$-error-correcting.*

1.4 The Packing and Covering Radii of a Code

We can get some further geometric insight into the error correcting capabilities of codes by introducing some further concepts.

Definition 1.4.1. *Let* $\mathbf{x}$ *be a word in* $\mathcal{A}^n$*, where* $|\mathcal{A}| = q$*, and let* $r \geq 0$ *be a real number. The* **sphere** *of radius* r *about* $\mathbf{x}$ *is the set*

$$S_q(\mathbf{x}, r) = \{\mathbf{y} \in \mathcal{A}^n | \, d(\mathbf{x}, \mathbf{y}) \leq r\}.$$

The **volume** $V_q(n, r)$ *of the sphere is independent of the centre and is given by*

$$V_q(n, r) = \sum_{k=0}^{[r]} \binom{n}{k} (q-1)^k$$

Definition 1.4.2. *Let* $C \subset \mathcal{A}^n$ *be a code. The* **packing radius** *of* C *is the* largest *integer* r *for which the spheres* $S_q(\mathbf{c}, r)$ *about each codeword* $\mathbf{c}$ *are disjoint. The* **covering radius** *of* C *is the* smallest *integer* s *for which the spheres* $S_q(\mathbf{c}, s)$ *about each codeword* $\mathbf{c}$ cover $\mathcal{A}^n$*, that is for which the union of the spheres* $S_q(\mathbf{c}, s)$ *is* $\mathcal{A}^n$*. We will denote the packing radius of* C *by* $pr(C)$ *and the covering radius by* $cr(C)$*.*

The following result shows the connection between error correction and the packing and covering radii.

Theorem 1.4.3. *Assuming that ties are always reported as errors, a code* C *is* t*-error-correcting if and only if the spheres* $S_q(\mathbf{c}, t)$ *about each codeword are disjoint.*

Proof. obvious □

Corollary 1.4.4. *Assuming that ties are always reported as errors, a code* C *is exactly* t*-error-decoding if and only if* $pr(C) = t$*.*

Another easy consequence of the above definition is the following theorem.

Theorem 1.4.5 (Sphere-Packing Bound). *A* t*-error-correcting* q*-ary* (n, M)*-code must satisfy*

$$M \cdot V_q(n, t) = M\big(1 + \binom{n}{1}(q-1) + \binom{n}{2}((q-1)^2 + \ldots + \binom{n}{t}(q-1)^t\big) \leq q^n.$$

Proof. clear. □

Definition 1.4.6. *A code* C *is said to be* **perfect** *if* $pr(C) = cr(C)$*. In words, a code* $C \subset \mathcal{A}^n$ *is perfect if their exists a number* r *for which the spheres* $S_q(\mathbf{c}, r)$ *about each codeword* $\mathbf{c}$ *are disjoint and cover* $\mathcal{A}^n$*.*

The size of a perfect code is uniquely determined by the length and the minimum distance. The following result illustrates this.

Theorem 1.4.7 (The Sphere-Packing Condition). *Let C be a q-ary (n, M, d)-code. Then C is perfect if and only if $d = 2t + 1$, i.e. d is odd, and*

$$M \cdot V_q(n, t) = q^n$$

that is,

$$M = q^n \Big/ \sum_{k=0}^{t} \binom{n}{k} (q-1)^k.$$

Proof. It is easy to see that the minimum distance d has to be odd. For if it were even, say $d = 2t + 2$, then there exist $\mathbf{c}$,$\mathbf{c}'$ such that $d(\mathbf{c}, \mathbf{c}') = 2t + 2$ and hence one can easily construct a word $\mathbf{d}$ with $d(\mathbf{c}, \mathbf{d}) = t + 1 = d(\mathbf{d}, \mathbf{c}')$. Now $\mathbf{d}$ lies in both $S_q(\mathbf{c}, t + 1)$ and $S_q(\mathbf{c}', t + 1)$. Therefore $pr(C) < t + 1$. On the other hand $\mathbf{d} \notin S_q(\mathbf{x}, t)$ for all $\mathbf{x} \in C$ by the triangle inequality. Thus $cr(C) > t$ which is a contradiction. Consequently a perfect code must have odd minimum distance. If C is perfect then $pr(C) = cr(C)$ and thus the set of M spheres of radius t about the codewords in C form a partition of $\mathcal{A}^n$. Hence, the sphere-packing condition holds.
Conversely, if the sphere-packing condition holds for some $(n, M, 2t + 1)$-code C, then since the spheres of radius t about each codeword are disjoint, and the sphere-packing condition implies that they cover $\mathcal{A}^n$, we have $pr(C) = cr(C) = t$. □

It is important to emphasise that the existence of numbers n, M, and t for which the sphere-packing condition holds does *not* imply that a code with these parameters exists. They are, in fact, rather rare. The next best thing is:

Definition 1.4.8. *A code C is said to be* **quasi-perfect** *if $cr(C) = pr(C) + 1$. In words, a code $C \subset \mathcal{A}^n$ is quasi-perfect if their exist a number r for which the spheres $S_q(\mathbf{c}, r)$ of radius r are disjoint and the spheres $S_q(\mathbf{c}, r + 1)$ of radius $r + 1$ cover $\mathcal{A}^n$.*

This concludes the more general approach to coding theory. In the remaining sections we are going to concentrate on so-called linear codes.

1.5 Linear Codes

We now turn to the problem of constructing codes which have some algebraic structure. The first idea is to take a group Q as a code alphabet, i.e. $\mathcal{A} = Q$, and to take a subgroup C of Q^n as a code. We shall, however, require a lot more structure. In the following $\mathcal{A} = \mathbb{F}_q$, where $\mathbb{F}_q$ is the finite field of $q = p^r$ (p prime) elements. Then $\mathcal{A}^n$ is an n-dimensional vector space (over $\mathbb{F}_q$).

Definition 1.5.1. *A* **linear code** C *is a linear subspace of* $\mathbb{F}_q^n$*. If* C *has dimension* k *then* C *is called an* $[n, k]$*-code.*

Obviously, the usual definition of the Hamming-distance d between two words in $\mathcal{A}^n$ carries over. Additionally, we have the concept of the **weight** of a codeword.

Definition 1.5.2. *Let* $C \subset \mathbb{F}_q^n$ *be an* (n, M)*-code. (Note: we do not require* C *to be linear). Then the* **weight** *of a codeword* $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in C$ *is defined by*

$$w(\mathbf{c}) = d(\mathbf{c}, \mathbf{0}) = |\{i|\ c_i \neq 0,\ 0 \leq i \leq n-1\}|$$

where $\mathbf{0} = (0, 0, \ldots, 0)$*. Furthermore the weight of the code* C *is defined to be*

$$w(C) = min\{w(\mathbf{c})|\ \mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}\}.$$

In the case of a linear code C the minimum distance and minimum weight of C are linked by the following result.

Theorem 1.5.3. *For a linear code* C *the minimum distance is equal to the minimum weight.*

Proof. If $\mathbf{c}, \mathbf{d} \in C$ then $\mathbf{c} - \mathbf{d} \in C$ by linearity. Thus $d(\mathbf{c}, \mathbf{d}) = d(\mathbf{c} - \mathbf{d}, \mathbf{0}) = w(\mathbf{c} - \mathbf{d})$. □

Note: An important consequence of this theorem is that, with a linear code C, one only needs to check $q^k = \binom{q^k}{1}$ codewords instead of $\binom{q^k}{2}$ pairs of codewords to determine the minimum distance $d(C)$.

From now on we shall use $[n, k, d]$-code as the notation for a k-dimensional linear code of length n with minimum distance d.

Because C is a linear subspace of $\mathbb{F}_q^n$ the following definition makes sense.

Definition 1.5.4. *A* **generator matrix** G *for a linear* $[n, k]$*-code,* C*, is a* k *by* n *matrix for which the rows are a basis of* C*.*

If G is a generator matrix for C then $C = \{\mathbf{a}G|\ \mathbf{a} \in \mathbb{F}_q^k\}$. We shall say that G is in **standard form** (often called reduced echelon form) if $G = (I_k\ P)$, where I_k is the k by k identity matrix. If G is in standard form then the first k symbols of a codeword are called **information symbols**. These can be chosen arbitrarily and then the remaining symbols, which are called **parity check symbols**, are determined. Using Definition 1.2.3 of the rate of a code it is immediate that the rate of a linear code is given by $R = k/n$, in accordance with the fact that k out of n symbols of an $[n, k]$-code carry information.

As far as error-correcting capability is concerned two codes C_1 and C_2 are equally good if C_2 is obtained by applying a fixed permutation of the symbols to all the codewords of C_1, i.e. there exist $\pi \in S_n$, the group of permutation on n elements, such that

$$\pi(C_1) = \{\pi(\mathbf{c}) = (c_{\pi(0)}, c_{\pi(1)}, \ldots, c_{\pi(n-1)}) | \, \mathbf{c} \in C_1\} = C_2.$$

We shall call such codes **equivalent**. In terms of the generator matrices of C_1 and C_2, G_1 and G_2 say, this means that

$$G_2 = G_1 \cdot P_\pi, \tag{1.5.5}$$

where P_π is the associated permutation matrix of $\pi \in S_n$. Sometimes the definition of equivalent codes is extended by allowing G_1 in (1.5.5) to be multiplied by a monomial matrix instead, i.e. a matrix with exactly one non-zero entry in each column and row. It is well known from linear algebra that every linear code is equivalent to a code with a generator matrix in standard form.

Definition 1.5.6. *Let C be a linear code of length n. The automorphism group of C is given by*

$$Aut(C) = \{\pi \in S_n | \pi(C) = C\}.$$

In terms of the generator matrix G of C this means that $\forall\, \pi \in Aut(C)$ the code C is also generated by GP_π where P_π is the permutation matrix corresponding to π.

Again, sometimes this definition is extended to monomial matrices instead (cf. [MS77, pp.229–238]).

We recall the following definition from linear algebra.

Definition 1.5.7. *Let $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1}), \mathbf{y} = (y_0, y_1, \ldots, y_{n-1}) \in \mathbb{F}_q^n$. The inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ is defined to be*

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=0}^{n-1} x_i y_i.$$

This leads to the following definition.

Definition 1.5.8. *If C is an $[n, k]$ code we define the* **dual code** $C^\perp$ *by*

$$C^\perp = \{\mathbf{y} \in \mathbb{F}_q^n \mid \langle \mathbf{x}, \mathbf{y} \rangle = 0, , \forall\ \mathbf{x} \in C\}.$$

The dual code $C^\perp$ is clearly a linear code, namely an $[n, n-k]$ code. It is worthwhile pointing out that in the case of a finite field, the subspaces C and $C^\perp$ can have an intersection larger than $\{\mathbf{0}\}$ and in fact we make the following definition.

Definition 1.5.9. *C is called* **self-dual** *(resp.* **self-orthogonal***) if $C = C^{\perp}$ (resp. $C \subseteq C^{\perp}$).*

Theorem 1.5.10. *Let C be a linear code. Then $C = (C^{\perp})^{\perp}$.*

Proof. Obviously, $C \subset (C^{\perp})^{\perp}$, by definition. The result now follows from the equality of dimensions. □

If $G = (I_k, P)$ is a generator matrix in standard form of the code C then $H = (-P^t, I_{n-k})$ (P^t is the transpose of P) is a generator matrix for $C^{\perp}$. This follows from the fact that H has the right size and rank and that $GH^t = 0$ implies that every codeword $\mathbf{a}G$ has inner product 0 with every row of H. in other words we have

$$\mathbf{x} \in C \iff \mathbf{x}H^t = \mathbf{0}. \tag{1.5.11}$$

In (1.5.11) we have $n - k$ linear equations which every codeword must satisfy.

If $\mathbf{y} \in C^{\perp}$ then the equation $\langle \mathbf{x}, \mathbf{y} \rangle = 0$ which holds for every $\mathbf{x} \in C$, is called a **parity check** (equation). H is called a **parity check matrix** of C.

Definition 1.5.12. *If C is a $[n, k]$ code with parity check matrix H then for every $\mathbf{x} \in \mathbb{F}_q^n$ we call $\mathbf{x}H^t$ the* **syndrome** *of $\mathbf{x}$.*

In (1.5.11) we saw that codewords are characterised by the syndrome $\mathbf{0}$. The syndrome is an important aid in decoding received vectors $\mathbf{x}$. Since C is a subgroup of $\mathbb{F}_q^n$ we can partition $\mathbb{F}_q^n$ into cosets of C. Two vectors $\mathbf{x}$ and $\mathbf{y}$are in the same coset iff they have the same syndrome ($\mathbf{x}H^t = \mathbf{y}H^t \iff (\mathbf{x} - \mathbf{y})H^t = \mathbf{0} \iff (\mathbf{x} - \mathbf{y}) \in C$). Therefore if a vector $\mathbf{x}$ is received for which the error pattern is $\mathbf{e}$ then $\mathbf{x}$ and $\mathbf{e}$ have the same syndrome. It follows that for minimum distance decoding of $\mathbf{x}$ one must choose a vector $\mathbf{e}$ of minimal weight in the coset which contains $\mathbf{x}$ and then decode $\mathbf{x}$ as $\mathbf{x} - \mathbf{e}$. The vector $\mathbf{e}$ is called the **coset leader**.

Here we see the first great advantage of introducing algebraic structure. For an $[n, k]$ code over $\mathbb{F}_q$ there are q^k codewords and q^n possible received messages. Let us assume that the rate of the code is reasonably high. The receiver needs to know q^{n-k} coset leaders corresponding to all possible syndromes. Now q^{n-k} is much smaller than q^n. If the code had no structure then for every possible received word $\mathbf{x}$ we would have to list the most likely transmitted word.

It is clear that if C has minimum distance $d(C) = 2t+1$ then every error pattern of weight $\leq t$ is the unique coset leader of some coset because two vectors with weight $\leq t$ have distance $\leq 2t$ and are therefore in different cosets (For if $\mathbf{e}' = \mathbf{e} + \mathbf{c}$ then $\mathbf{c} = \mathbf{e}' - \mathbf{e}$ with $w(\mathbf{c}) = d(\mathbf{e}', \mathbf{e}) \leq 2t$ and hence $\mathbf{e}' = \mathbf{e}$ because $d(C) = 2t + 1$). If C is perfect there are no other coset leaders and if C is quasi-perfect with $d(C) = 2t + 1$ then all coset leaders have weight $\leq t + 1$.

It will often turn out to be useful to adjoin one extra symbol to every codeword of a code C according to some natural rule. The most common of these is given in the following definition.

Definition 1.5.13. *If C is a code of length n over the code alphabet $\mathcal{A} = \mathbb{F}_q$ we define the* **extended code** $\overline{C}$ *by*

$$\overline{C} = \left\{(c_0, c_1, \ldots, c_{n-1}, c_n) \,|(c_0, c_1, \ldots, c_{n-1}) \in C, \sum_{i=0}^{n} c_i = 0\right\}.$$

If C is a linear code with generator matrix G and parity check matrix H then $\overline{C}$ has generator matrix $\overline{G}$ and parity check matrix $\overline{H}$, where $\overline{G}$ is obtained by adding a column to G in such a way that the sum of the columns of $\overline{G}$ is $\mathbf{0}$ and where

$$\overline{H} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ & & & & 0 \\ & & H & & 0 \\ & & & & \vdots \\ & & & & 0 \end{pmatrix}.$$

If C is a binary code with an odd minimum distance d then $\overline{C}$ has minimum distance $d+1$ since all weights and distances for $\overline{C}$ are even.

1.6 Hamming Codes - an Example

Let G be the k by n generator matrix of an $[n, k]$ code and assume that any two columns of G are linearly independent. The dual code $C^{\perp}$ has G as parity check matrix. If $\mathbf{c} \in C^{\perp}$ and if $\mathbf{e}$ is an error vector of weight 1, then the syndrome $(\mathbf{c} + \mathbf{e})G^t$ is a multiple of a column of G. Since this uniquely determines the column of G it follows that $C^{\perp}$ is a code which corrects at least one error. We now look at the case in which n is maximal (given k).

Definition 1.6.1. *Let $n = (q^k - 1)/(q - 1)$, where $q = p^r$, $r \geq 1$ and p a prime. The $[n, n-k]$* **Hamming code** *over $\mathbb{F}_q$ is a code for which the parity check matrix has columns which are pairwise linearly independent (over $\mathbb{F}_q$), i.e. the columns are a maximal set of pairwise linearly independent vectors.*

Lemma 1.6.2. *The minimum distance of a Hamming code is 3.*

Proof. Because any two columns of G are linearly independent, G does not contain the $\mathbf{0}$ column, nor are there any codewords of weight ≤ 2. Thus $d(C) \geq 3$. On the other hand because G consist of a maximal set of pairwise linearly independent columns which means that adding any two columns

results in a column vector that cannot be linearly independent from all the columns in G, hence it is a multiple of one of them. Thus giving rise to a codeword of weight 3. □

Theorem 1.6.3. *Hamming codes are perfect codes.*

Proof. Let C be the $[n, n-k]$ Hamming code over $\mathbb{F}_q$, where $n = (q^k - 1)/(q-1)$. If $\mathbf{c} \in C$ then

$$|S_q(\mathbf{c}, 1)| = 1 + n(q-1) = q^k.$$

Therefore the q^{n-k} disjoint spheres of radius 1 around the codewords of C contain $|C| \cdot q^k = q^n$ words, i.e. all possible words. Hence C is perfect. □

Example 1.6.4. The $[7, 4]$ binary Hamming code C has parity check matrix

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \tag{1.6.5}$$

Claim 1.6.6. *The extended code $\overline{C}$ corresponding to* (1.6.5) *is self-dual.*

Proof. The parity check matrix of $\overline{C}$ is given by

$$\overline{H} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix}.$$

It is now easy to see that $\overline{H}\,\overline{H}^t = \mathbf{0}$. □

1.7 Some General Properties of Linear Codes

Before looking at some more examples of linear codes we shall state and prove some of their many properties.

Theorem 1.7.1. *If H is the parity check matrix of a code of length n, then the code has dimension $n-k$ iff some k columns of H are linearly independent but no $k+1$ columns are. (Thus k is the rank of H).*

Proof. Clear. □

Theorem 1.7.2. *If H is the parity check matrix of a code of length n, then the code has minimum distance d iff every $d-1$ columns are linearly independent and some d columns are linearly dependent.*

Proof. There is a codeword $\mathbf{c}$ of weight $d \iff \mathbf{c}H^t = \mathbf{0}$ for some vector $\mathbf{c}$ of weight $d \iff$ some d columns are linearly dependent. □

Theorem 1.7.3. *Let C be a linear code then $Aut(C) = Aut(C^\perp)$.*

Proof. By Theorem 1.5.10 it is enough to show that $Aut(C) \subset Aut(C^\perp)$. Let $\mathbf{d} \in C^\perp$ and $\pi \in Aut(C)$ then for all $\mathbf{c} \in C$ we have $0 = \langle \pi^{-1}(\mathbf{c}), \mathbf{d} \rangle = \langle \mathbf{c}, \pi(\mathbf{d}) \rangle$, i.e. $\pi(\mathbf{d}) \in C^\perp$. □

It is worthwhile pointing out that $C_1 \subset C_2$ does not, in general, imply that $Aut(C_1) \supset Aut(C_2)$. In general, it is very difficult to determine the complete automorphism group of a code.

We shall now have a brief look at some of the bounds associated with codes. Given a code alphabet $\mathcal{A}$ of size q, we define an $(n, *, d)$-code as a code of length n and minimum distance d. We are interested in the maximal number of codewords (i.e. the largest M which can be put in the place of the $*$). An (n, M, d)-code which is not contained in an $(n, M+1, d)$-code is called **maximal**.

Definition 1.7.4. *Define $A(n, d)$ to be the maximum value of M for which an (n, M, d)-code exist. A code C such that $|C| = A(n, d)$ is called optimal.*

The study of the numbers $A(n, d)$ is considered to be the central problem in combinatorial coding theory.

Theorem 1.7.5 (The Singleton Bound). $A(n, d) \leq q^{n-d+1}$.

Proof. Let C be a q-ary (n, M, d)-code. Removing the last $(d-1)$ coordinates from all codewords, we obtain a $(n - d + 1, M, d')$-code $(d' \geq 1)$, i.e. the M codewords obtained this way are all different. Hence $M < q^{n-d+1}$. □

Consequently, for linear codes:

Corollary 1.7.6. *If C is a q-ary $[n, k, d]$ then $n - k \geq d - 1$.*

Note that codes with $n - k = d - 1$ are called **maximum distance separable** (abbreviated MDS). Theorems 1.4.5 and 1.7.5 have provided upper bounds on the size of a code with given minimum distance. The next theorem gives a lower bound for (non-linear) codes, which we will then modify to linear codes.

Theorem 1.7.7 (The Gilbert-Varshamov Bound). *For $n, d \in \mathbb{N}$ with $d \leq n$, we have*

$$A(n, d) \geq q^n / V_q(n, d-1).$$

Proof. Let the q-ary (n, M, d)-code C over $\mathcal{A}$ be maximal. This implies that there is no word in $\mathcal{A}^n$ with distance d or more to all the words of C. In other words: the spheres $S_q(\mathbf{c}, d-1)$, with $\mathbf{c} \in C$, cover $\mathcal{A}^n$. Thus $M \cdot V_q(n, d-1) \geq q^n$. □

The proof shows that a code which has at least $q^n/V_q(n, d-1)$ codewords can be constructed by simply starting with any codeword $\mathbf{c}_0$ and then consecutively adding new words which have distance at least d to the words which have been chosen before, until the code is maximal. Such a code has no structure. Surprisingly enough, the requirement that C is linear is not an essential restriction as the following theorem shows.

Theorem 1.7.8. *If $n, k, d \in \mathbb{N}$ satisfy $V_q(n, d-1) < q^{n-k+1}$ then an $[n, k, d]$-code exists.*

Proof. For $k = 0$ this is trivial. Let C_{k-1} be a $[n, k-1, d]$-code over $\mathbb{F}_q$. Since $|C_{k-1}|V_q(n, d-1) < q^n$, this code is not maximal. Hence there is a word $\mathbf{x} \in \mathcal{A}^n$ with distance $\geq d$ to all words of C_{k-1}. Let C_k be the code spanned by $C_k - 1$ and $\{\mathbf{x}\}$. Let $\mathbf{y} = a\mathbf{x} + \mathbf{c}$ (where $0 \neq a \in \mathbb{F}_q$, $\mathbf{c} \in C_{k-1}$) be a codeword in C_k. Then

$$w(\mathbf{y}) = w(a^{-1}\mathbf{y}) = w(\mathbf{x} + a^{-1}\mathbf{c}) = d(\mathbf{x}, -a^{-1}\mathbf{c}) \geq d$$

□

We finish this section by looking at the so-called asymptotic Gilbert-Varshamov bound. Set $\delta = d/n$ and consider the quantity

$$\alpha(\delta) = \limsup_{n\to\infty} \left(\frac{1}{n} \log_q A_q(n, \delta n) \right),$$

which is the limit superior of the maximum rate of a code of length n and minimum distance d, when $n \to \infty$ in such a way that $\delta = d/n$ remains constant.

We define the **entropy function** H_q on $(0, 1)$ by

$$H_q(\lambda) = -\lambda \log_q(\lambda) - (1-\lambda) \log_q(1-\lambda) \text{for } 0 < \lambda < 1.$$

Lemma 1.7.9. *Let $\theta = \frac{q-1}{q}$, $q \geq 2$. For $0 \leq \delta \leq \theta$, we have*

$$\lim_{n\to\infty} \left(\frac{1}{n} \log_q(V_q(n, [\delta n])) \right) = H_q(\delta) + \delta \log_q(q-1).$$

Proof. [Rom92, p.448] □

Now we can state the asymptotic Gilbert-Varshamov bound.

Theorem 1.7.10 (Asymptotic Gilbert-Varshamov Bound). *If $0 \leq \delta \leq (q-1)/q$ then*

$$\alpha(\delta) \geq 1 - H_q(\delta) - \delta \log_q(q-1).$$

Proof. By Lemma 1.7.9 and Theorem 1.7.7 we have

$$\alpha(\delta) = \limsup_{n\to\infty}[\frac{1}{n}\log_q A(n,\delta n) \geq \lim_{n\to\infty}(1 - \frac{1}{n}\log_q V_q(n,\delta n)) = 1 - H_q(\delta) - \delta\log_q(q-1).$$

□

The reason why we are interested in this bound is that is gives us some sort of measurement of how "good" or "bad" code are in the following sense.

Definition 1.7.11. *A family of codes over $\mathbb{F}_q$, for q fixed, is said to be* **good** *if it contains an infinite sequence of codes $C_1, C_2, \ldots$, where C_i is an $[n_i, k_i, d_i]$- code such that both the rate $R_i = k_i/n_i$ and d_i/n_i approach a nonzero limit as $i \to \infty$.*

It can be shown that BCH codes (described in Section 1.8.4) are bad in this sense, whereas classical Goppa codes, a generalisation of BCH codes (defined in Section 1.9), are good codes. The interested reader is referred to [MS77, p.269] and [vL82, pp. 110–111].

1.8 Cyclic Codes and BCH Codes

1.8.1 Definition of a cyclic code

In Definition 1.5.6 we defined the automorphism group $Aut(C)$ of a code C. In this section we shall study linear codes for which the automorphism group contains the cyclic group of order n, where n is the word length.

Definition 1.8.1. *A linear code is called* **cyclic** *if for all $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \in C$ it follows that $(c_{n-1}, c_0, c_1, \ldots, c_{n-2}) \in C$ also.*

The most important tool in our description of cyclic codes is the following isomorphism between $\mathbb{F}_q^n$ and a group of polynomials. The multiples of $x^n - 1$ form a principal ideal in the polynomial ring $\mathbb{F}_q[x]$. The residue class ring $\mathbb{F}_q[x]/(x^n - 1)$ has the set of polynomials

$$\{a_0 + a_x + \ldots + a_{n-1}x^{n-1} | \, a_i \in \mathbb{F}_q, 0 \leq i \leq n\}$$

as a system of representatives. Clearly $\mathbb{F}_q^n$ is isomorphic to this ring (considered only as an additive group). In the following we shall also use the multiplicative structure which we have now introduced, namely multiplication of polynomials $\mathrm{mod}(x^n - 1)$. From now on we make the following identification:

$$(a_0, a_1, \ldots, a_{n-1}) \in \mathbb{F}_q^n \leftrightarrow a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} \in \mathbb{F}_q[x]/(x^n - 1) \tag{1.8.2}$$

and we shall often speak of a codeword $\mathbf{c}$ as the codeword $c(x)$, using (1.8.2). Extending this, we interpret a linear code as a subset of $\mathbb{F}_q[x]/(x^n-1)$.

Theorem 1.8.3. *A linear code C in $\mathbb{F}_q^n$ is cyclic if and only if C is an ideal in $\mathbb{F}_q[x]/(x^n-1)$.*

Proof. If C is an ideal in $\mathbb{F}_q[x]/(x^n-1)$ and $c(x) = c_0 + c_1x + \ldots + c_{n-1}x^{n-1}$ is any codeword, then $xc(x)$ is also a codeword, i.e.

$$(c_{n-1}, c_0, c_1, \ldots, c_{n-2}) \in C.$$

Conversely, if C is cyclic, then for every codeword $c(x)$ the word $xc(x)$ is also in C. Therefore $x^jc(x)$ is in C for every j, and since C is linear $a(x)c(x)$ is in C for every polynomial $a(x)$. Hence C is an ideal. □

Convention 1.8.4. *We will only consider cyclic codes of length n over $\mathbb{F}_q$ with* $\gcd(n,q) = 1$.

Since $\mathbb{F}_q[x]/(x^n-1)$ is a principal ideal ring every cyclic code C consists of the multiples of a polynomial $g(x)$ which is the **monic** polynomial of lowest degree (i.e. not the zero polynomial) in the ideal.

This polynomial $g(x)$ is called the **generator polynomial** of the cyclic code. The generator polynomial is a divisor of x^n-1 (since otherwise the greatest common divisor of x^n-1 and $g(x)$ would be a polynomial in C of degree lower than the degree of $g(x)$). Let $x^n-1 = f_1(x)f_2(x)\cdots f_t(x)$ be the decomposition of x^n-1 into irreducible factors. Because of Convention 1.8.4 these factors are different. We can now find all cyclic codes of length n by picking (in all possible ways) one of the 2^t factors of x^n-1 as a generator polynomial $g(x)$ and defining the corresponding code to be the set of multiples of $g(x) \pmod{x^n-1}$.

Definition 1.8.1 guarantees that the automorphism group of a cyclic code C contains the cyclic group generated by the permutation

$$i \mapsto i+1 \pmod{n}.$$

However, since $a(x^q) = a(x)^q$ is in the same cyclic code as $a(x)$, we see that the permutation π_q defined by $\pi_q(i) = qi \pmod{n}$ (i.e. $x \mapsto x^q$) also maps a cyclic code onto itself. (Note that π_q is the Frobenius map.) If m is the order of $q \pmod{n}$ then the two permutations $i \mapsto i+1$ and π_q generate a group of order nm contained in $Aut(C)$.

1.8.2 Generator matrix and check polynomial

Let $g(x)$ be the generator polynomial of a cyclic code C of length n. If $g(x)$ has degree $n-k$ then the codewords $g(x), xg(x), \ldots, x^{k-1}g(x)$ clearly form a basis for C, i.e. C is an $[n,k]$-code. Hence,

if $g(x) = g_0 + g_1 x + \cdots + g_{n-k}x^{n-k}$ then

$$G = \begin{pmatrix} g_0 & g_1 & \cdots & g_{n-k} & 0 & 0 & \cdots & 0 \\ 0 & g_0 & \cdots & g_{n-k-1} & g_{n-k} & 0 & \cdots & 0 \\ 0 & 0 & \cdots & g_{n-k-2} & g_{n-k-1} & g_{n-k} & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & \cdots & & g_0 & g_1 & \cdots & g_{n-k} \end{pmatrix}$$

is a generator matrix for C. This means that we encode an information sequence $(a_0, a_1, \ldots, a_{k-1})$ as $\mathbf{a}G$ which is the polynomial

$$(a_0 + a_1 x + \cdots + a_{k-1}x^{k-1})g(x).$$

A more convenient form of the generator matrix is obtained by defining (for $n-k \leq i \leq n-1$), $x^i = g(x)q_i(x) + r_i(x)$ where $r_i(x)$ is a polynomial of degree $< n-k$. The polynomials $x^i - r_i(x)$ are codewords of C and form a basis for the code which yields a generator matrix of C in standard form (with I_k on the right, though).

Since $g(x)$ is a divisor of $x^n - 1$ there is a polynomial $h(x) = h_0 + h_1 x + \cdots + h_k x^k$ such that $g(x)h(x) = x^n - 1$ (in $\mathbb{F}_q[x]$). In the ring $\mathbb{F}_q/(x^n - 1)$ we have $g(x)h(x) = 0$, i.e. $g_0 h_i + g_1 h_{i-1} + \cdots + g_{n-k}h_{i-n+k} = 0$ for $i = 0, 1, \ldots, n-1$. It follows that

$$H = \begin{pmatrix} 0 & 0 & \cdots & 0 & h_k & h_{k-1} & \cdots & h_1 & h_0 \\ 0 & 0 & \cdots & h_k & h_{k-1} & \cdots & h_1 & h_0 & 0 \\ \vdots & & & & & & & & \vdots \\ h_k & h_{k-1} & \cdots & h_1 & h_0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

is a parity check matrix for the code C. We call $h(x)$ the **check polynomial** of C. The code C consists of all $c(x)$ such that $c(x)h(x) = 0$. By comparing G and H we see that the code with generator polynomial $h(x)$ is equivalent to the dual of C (obtained by reversing the order of the symbols).

1.8.3 Zeros of a cyclic code

Let $x^n - 1 = f_1(x) \cdots f_t(x)$ and let β_i be a zero of $f_i(x)$ in some extension field of $\mathbb{F}_q$. Then $f_i(x)$ is the minimal polynomial of β_i and therefore the cyclic code associated with $g(x) = f_i(x)$ is nothing but the set of polynomials $c(x)$ for which $c(\beta_i) = 0$. So in general a cyclic code can be specified by requiring that all codewords have prescribed zeros. In fact, it is sufficient to take one zero β_i of each irreducible factor f_i of the generator polynomial $g(x)$ and require that all codewords have these points as zeros (all in a suitable extension of $\mathbb{F}_q$). If we start with any

set $\{\alpha_1, \alpha_2, \ldots, \alpha_s\}$ and define a code C by $c(x) \in C$ iff $c(\alpha_i) = 0$ for $i = 1, 2, \ldots, s$, then the code C is cyclic and the generator polynomial of C is the least common multiple of the minimal polynomials of $\alpha_1, \alpha_2, \ldots, \alpha_s$. Suppose that all these zeros lie in $\mathbb{F}_{q^m}$ (which we can represent as a vector space $\mathbb{F}_q^m$). For every i we can consider the m by n matrix with the vector representations of $1, \alpha_i, \alpha_i^2, \ldots, (\alpha_i)^{n-1}$ as columns and put all these together to form the sm by n matrix H which has its entries in $\mathbb{F}_q$. Clearly $\mathbf{c}H^t = \mathbf{0}$, where $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$, means the same thing as $c(\alpha_i) = 0$ for $i = 1, 2, \ldots, s$. The rows of H are not necessarily independent. We may obtain a parity check matrix from H by deleting some of the rows. As an illustration of this way of describing cyclic codes we shall prove that binary (and many other) Hamming codes (cf. Definition 1.6.1) are (equivalent to) cyclic codes.

Theorem 1.8.5. *Let $n = (q^m - 1)/(q - 1)$ and let β be a primitive nth root of unity in $\mathbb{F}_{q^m}$. Furthermore, let* $\gcd(m, q - 1) = 1$. *The cyclic code*

$$C = \{c(x) | c(\beta) = 0\}$$

is equivalent to the $[n, n - m]$ Hamming code over $\mathbb{F}_q$.

Proof. Since

$$n = (q - 1)(q^{m-2} + 2q^{m-3} + \cdots + (m - 2)q + (m - 1)) + m,$$

we have $(n, q - 1) = (m, q - 1) = 1$. Therefore $\beta^{i(q-1)} \neq 1$ for $1 \leq i \leq n - 1$, i.e. $\beta^i \notin \mathbb{F}_q$ for $i = 1, 2, \ldots, n - 1$. It follows that the columns of the matrix H, which are the representations of $1, \beta, \beta^2, \ldots, \beta^{n-1}$ as vectors in $\mathbb{F}_q^m$, are pairwise linearly independent over $\mathbb{F}_q$. So H is the parity check matrix of an $[n, n - m]$-Hamming code. □

1.8.4 BCH codes

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

Definition 1.8.6. *A cyclic code of length n over $\mathbb{F}_q$ is called a* **BCH code of designed distance** *δ if its generator polynomial $g(x)$ is the least common multiple of the minimal polynomials of $\beta^l, \beta^{l+1}, \ldots, \beta^{l+\delta-2}$ for some l, where β is a primitive nth root of unity. Usually we shall take $l = 1$ (sometimes called a* **narrow-sense** *BCH code). If $n = q^m - 1$, i.e. β is a primitive element of $\mathbb{F}_{q^m}$ then the BCH code is called* **primitive**.

Remark 1.8.7. One of the simplest examples of BCH codes has its own name. A **Reed-Solomon code** (RS code) is a primitive BCH code of length $n = q-1$ over $\mathbb{F}_q$. The generator of such a code has the form $g(x) = \prod_{i=1}^{\delta-1}(x - \alpha^i)$ where α is primitive in $\mathbb{F}_q$.

The terminology "designed distance" is explained by the following theorem.

Theorem 1.8.8. *The minimum distance of a BCH code with designed distance δ is at least δ.*

Proof. In the same way as in Section 1.8.3 we form the $m(\delta-1)$ by n matrix H:

$$H = \begin{pmatrix} 1 & \beta^l & \beta^{2l} & \dots & \beta^{(n-1)l} \\ 1 & \beta^{l+1} & \beta^{2(l+1)} & \dots & \beta^{(n-1)(l+1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \beta^{l+\delta-2} & \beta^{2(l+\delta-2)} & \dots & \beta^{(n-1)(l+\delta-2)} \end{pmatrix}$$

where each entry is interpreted as a column vector of length m over $\mathbb{F}_q$. A word $\mathbf{c}$ is in the BCH code iff $\mathbf{c}H^t = \mathbf{0}$. The $m(\delta-1)$ rows of H are not necessarily independent. Consider any $\delta-1$ columns of H and let $\beta^{i_1 l}, \dots, \beta^{i_{\delta-1} l}$ be the top elements in the columns. The determinant of the submatrix of H obtained in this way is a Vandermonde determinant with value $\beta^{(i_1+\dots+i_{\delta-1})l}\Pi_{r>s}(\beta^{i_r} - \beta^{i_s}) \neq 0$, since β is a primitive nth root of unity. Therefore any $\delta-1$ columns of H are linearly independent and hence a codeword $\mathbf{c} \neq \mathbf{0}$ has weight $\geq \delta$. □

Remark 1.8.9. If l is fixed, BCH codes are **nested**, i.e. the code of designed distance δ_1 contains the code of designed distance δ_2 iff $\delta_1 \leq \delta 2$.

Theorem 1.8.8 is usually called the **BCH bound**. Finding the actual minimum distance of a BCH code is in general a hard problem. However, for a special class of BCH codes we have the following two results.

Theorem 1.8.10. *A BCH code C of length $n = q^m - 1$ and designed distance $\delta = q^h - 1$ over $\mathbb{F}_q$ has minimum distance δ.*

Proof. see [MS77, pp.260–261] □

and its corollary

Corollary 1.8.11. *The true minimum distance of a primitive BCH code C of designed distance δ over $\mathbb{F}_q$ is at most $q\delta - 1$.*

Proof. Define d_0 by $q^{h-1} \leq \delta < d_0 = q^h - 1$. Since BCH codes are nested, C contains the code of designed distance d_0, which by Theorem 1.8.10 has true minimum distance d_0. Therefore

$$d \leq d_0 \leq q\delta - 1.$$

□

In Section 1.8.1 we have already pointed out that the automorphism group of a cyclic code of length n over $\mathbb{F}_q$ not only contains the cyclic permutations but also π_q. For BCH codes we can prove much more. Consider a primitive BCH code C of length $n = q^m - 1$ over $\mathbb{F}_q$ with designed distance δ (i.e. $\alpha, \alpha^2, \ldots, \alpha^{\delta-1}$ are the prescribed zeros of the codewords, where α is a primitive element of $\mathbb{F}_{q^m}$).

We denote the **positions** of the symbols in the codewords by X_i $(i = 0, 1, \ldots, n-1)$, where $X_i = \alpha^i$. We extend the code to $\overline{C}$ by adding an overall parity check. We denote the additional position by ∞ and we make the obvious conventions concerning arithmetic with the symbol ∞. We represent the codeword $(c_0, c_1, \ldots, c_\infty)$ by $c_0 + c_1 x + \cdots + c_{n-1}x^{n-1} + c_\infty x^\infty$ and make the further conventions $1^\infty = 1, (\alpha^i)^\infty = 0$ for $i \not\equiv 0 \pmod{n}$.

We shall no show that $\overline{C}$ is invariant under the permutations of the **affine permutation group** $AGL(1, q^m)$ acting on the positions. This group consists of the permutations

$$P_{u,v}(X) = uX + v, \ (u \in \mathbb{F}_{q^m}, \, v \in \mathbb{F}_{q^m}, \, u \neq 0),$$

and it is 2-transitive. First observe that $P_{\alpha,0}$ is the cyclic shift on the positions of C and that it leaves ∞ invariant. Let $(c_0, c_1, \ldots, c_{n-1}, c_\infty) \in C$ and let $P_{u,v}$ yield the permuted word $(c'_0, c'_1, \ldots, c'_\infty)$. Then for $0 \leq k \leq d-1$ we have

$$\begin{aligned}\sum_i c'_i \alpha^{ik} &= \sum_i c_i (u\alpha^i + v)^k = \sum_i c_i \sum_{l=0}^{k} \binom{k}{l} u^l \alpha^{il} v^{k-l} \\ &= \sum_{l=0}^{k} \binom{k}{l} u^l v^{k-l} \sum_i c_i (\alpha^l)^i = 0\end{aligned}$$

because the inner sum is 0 for $0 \leq l \leq \delta - 1$ since $\mathbf{c} \in C$. So we have the following theorem.

Theorem 1.8.12. *The group of automorphisms of every extended primitive BCH code of length $n + 1 = q^m$ over $\mathbb{F}_q$ is $AGL(1, q^m)$.*

Corollary 1.8.13. *The minimum weight of a primitive binary BCH code is odd.*

Proof. Let C be such a code. We have shown that $Aut(\overline{C})$ is transitive on the positions. The same is true if we only consider the words of minimum weight in $\overline{C}$. So $\overline{C}$ has words of minimum weight with a 1 in the final check position. □

1.9 Classical Goppa Codes

1.9.1 Motivation

Consider once again the parity check matrix of a (narrow sense) BCH code as given in the proof of Theorem 1.8.8, i.e.

$$H = \begin{pmatrix} 1 & \beta & \beta^2 & \dots & \beta^{(n-1)} \\ 1 & \beta^2 & \beta^4 & \dots & \beta^{(n-1)2} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \beta^{\delta-1} & \beta^{2(\delta-1)} & \dots & \beta^{(n-1)(\delta-1)} \end{pmatrix},$$

where β is a primitive nth root of unity in $\mathbb{F}_{q^m}$ and each entry is interpreted as a column vector of length m over $\mathbb{F}_q$; (existence of β implies that $n|(q^m-1)$).

In Theorem 1.8.8 we proved that the minimum distance is at least δ by using the fact that any submatrix of H formed by taking $\delta-1$ columns is a Vandermonde matrix and therefore has determinant $\neq 0$. It is straightforward to check that the same argument works if we replace H by

$$\widehat{H} = \begin{pmatrix} h_0\beta_0 & h_1\beta_1 & \cdots & h_{n-1}\beta_{n-1} \\ \vdots & \vdots & & \vdots \\ h_0\beta_0^{\delta-1} & h_1\beta_1^{\delta-1} & \cdots & h_{n-1}\beta_{n-1}^{\delta-1} \end{pmatrix},$$

where $0 \neq h_j \in \mathbb{F}_{q^m}$ and the β_i are different non-zero elements of $\mathbb{F}_{q^m}$. If $h_j \in \mathbb{F}_q$ $(0 \leq j \leq n-1)$ then the factors h_j have no essential effect; the code is replaced by an equivalent code. However, if the h_j are elements of $\mathbb{F}_{q^m}$ then the terms $h_j\beta_j^t$ considered as column vectors over $\mathbb{F}_q$ can be very different from the original entries.

1.9.2 Goppa codes

Let $(c_0, c_1, \dots, c_{n-1})$ be a codeword in a BCH code with designed distance δ (word length n, β a primitive nth root of unity). Then, by definition $\sum_{i=0}^{n-1} c_i(\beta^j)^i = 0$ for $1 \leq j < d$. We wish to write this condition in another way. Observe that

$$\frac{z^n-1}{z-\beta^{-i}} = \sum_{k=0}^{n-1} z^k(\beta^{-i})^{n-1-k} = \sum_{k=0}^{n-1} \beta^{i(k+1)} z^k. \tag{1.9.1}$$

It follows that

$$\sum_{i=0}^{n-1} \frac{c_i}{z-\beta^{-i}} = \frac{z^{d-1}p(z)}{z^n-1} \tag{1.9.2}$$

for some polynomial $p(z)$.

If $g(z)$ is any polynomial and $g(\gamma) \neq 0$, we define $1/(z-\gamma)$ to be the unique polynomial mod $g(z)$ such that $(z-\gamma)\cdot 1/(z-\gamma) \equiv 1 \pmod{g(z)}$, i.e.

$$\frac{1}{(z-\gamma)} = \frac{-1}{g(\gamma)}\left(\frac{g(z)-g(\gamma)}{z-\gamma}\right). \tag{1.9.3}$$

These observations serve as preparation for the following definition.

Definition 1.9.4. *Let $g(z)$ be a (monic) polynomial of degree t over $\mathbb{F}_{q^m}$, $L = \{\gamma_0, \gamma_1, \ldots, \gamma_{n-1}\} \subset \mathbb{F}_{q^m}$, such that $|L| = n$ and $g(\gamma_i) \neq 0$ for $0 \leq i \leq n-1$. We define the* **Goppa code** *$\Gamma(L, g)$ with* **Goppa polynomial** *$g(z)$ to be the set of codewords $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ over $\mathbb{F}_q$ for which*

$$\sum_{i=0}^{n-1} \frac{c_i}{z-\gamma_i} \equiv 0 \pmod{g(z)}. \tag{1.9.5}$$

Observe that Goppa codes are linear. In order to establish the connection with Section 1.9.1 we try to find a suitable parity check matrix for $\Gamma(L, g)$. From (1.9.5) and (1.9.3) we see that

$$\left(\frac{1}{g(\gamma_0)}\cdot\frac{g(z)-g(\gamma_0)}{z-\gamma_0}, \ldots, \frac{1}{g(\gamma_{n-1})}\cdot\frac{g(z)-g(\gamma_{n-1})}{z-\gamma_{n-1}}\right),$$

with each entry interpreted as a column vector, is, in a sense, a parity check matrix. Let $h_j = g(\gamma_j)^{-1}$ and hence $h_j \neq 0$. If $g(z) = \sum_{i=0}^{t} g_i z^t$ then in the same way as (1.9.1) we have

$$\frac{g(z)-g(x)}{z-x} = \sum_{i+j\leq t-1} g_{i+j+1} x^j z^i.$$

Leaving out the factors z^i we find the parity check matrix for $\Gamma(L, g)$

$$\begin{pmatrix} h_0 g_t & \cdots & h_{n-1} g_t \\ h_0(g_{t-1} + g_t\gamma_0) & \cdots & h_{n-1}(g_{t-1} + g_t\gamma_{n-1}) \\ \vdots & \cdots & \vdots \\ h_0(g_1 + g_2\gamma_0 + \cdots + g_t\gamma_0^{t-1}) & \cdots & h_{n-1}(g_1 + g_2\gamma_{n-1} + \cdots + g_t\gamma_{n-1}^{t-1}) \end{pmatrix}.$$

By a linear transformation we find the matrix we want, namely

$$H = \begin{pmatrix} h_0 & \cdots & h_{n-1} \\ h_0\gamma_0 & \cdots & h_{n-1}\gamma_{n-1} \\ \vdots & \cdots & \vdots \\ h_0\gamma_0^{t-1} & \cdots & h_{n-1}\gamma_{n-1}^{t-1} \end{pmatrix}$$

(here we have used the fact that $g_t \neq 0$).

This does not have the full generality of the matrix $\widehat{H}$ of Section 1.9.1 where the h_j were arbitrary, since now we have $h_j = g(\gamma_j)^{-1}$.

Theorem 1.9.6. *The Goppa code $\Gamma(L, g)$ defined in Definition 1.9.4 has dimension $\geq n - mt$ and minimum distance $\geq t + 1$.*

Proof. This follows from the parity check matrix H in exactly the same way as for BCH codes. □

Chapter 2

Function Fields And Places

2.1 Introduction

In this chapter we present the basic definitions and results concerning algebraic function fields which are needed for the rest of the thesis. Our exposition follows very closely that of Stichtenoth [Sti91] and all the proofs can be found there.

The algebraic approach to algebraic function fields is more elementary than the approach via algebraic geometry: only some basic knowledge of algebraic field extensions, including some Galois theory is assumed. A second advantage is that some principal results of the theory (such as the Riemann-Roch Theorem) can be derived very quickly for function fields over an arbitrary constant field K. This facilitates the presentation of some applications of algebraic function fields to coding theory, which we will exhibit in Chapter 3.

Convention 2.1.1. *Throughout the whole chapter, K denotes an arbitrary field (unless stated otherwise).*

2.2 Places

Definition 2.2.1. *An* **algebraic function field** F/K **of one variable over** K *is an extension field $F \supseteq K$ such that F is a finite algebraic extension of $K(x)$ for some $x \in F$ which is transcendental over K.*

For brevity, we shall simply refer to F/K as a **function field**. Observe that the set $\tilde{K} = \{z \in F | \, z \text{ is algebraic over } K\}$ is a subfield of F, since sums, products, and inverses of algebraic elements are also algebraic. $\tilde{K}$ is called the **field of constants** of F/K. We have $K \subseteq \tilde{K} \subseteq F$, and it is

easily verified that $F/\tilde{K}$ is a function field over $\tilde{K}$. We say that K is **algebraically closed in** F (or K is the **full constant field of** F) if $\tilde{K} = K$. It can be shown that $\tilde{K}$ is a finite field extension of K, and hence we make the following assumption which is not critical to the theory:

Convention 2.2.2. *From here on, F/K will always denote an algebraic function field of one variable such that K is the full constant field of F/K.*

Remark 2.2.3. The elements of F which are transcendental over K can be characterised as follows: $z \in F$ is transcendental over K iff $[F : K(z)] < \infty$.

Example 2.2.4. The simplest example of an algebraic function field is the **rational function field**; F/K is called **rational** if $F = K(x)$ for some $x \in F$ transcendental over K. Any element $0 \neq z \in K(x)$ has a unique representation

$$z = a \cdot \prod_i p_i(x)^{n_i}, \tag{2.2.5}$$

in which $a \in K^*$, the $p_i(x) \in K[x]$ are monic, pairwise distinct irreducible polynomials and the $n_i \in \mathbb{Z}$.

An arbitrary function field F/K (which is not necessarily rational) is often represented as a simple algebraic field extension of a rational function field $K(x)$, i.e. $F = K(x, y)$ where $\varphi(y) = 0$ for some irreducible polynomial $\varphi(T) \in K(x)[T]$.

Definition 2.2.6. *A* **valuation ring** *of the function field F/K is a ring $\mathcal{O} \subseteq F$ with the following properties:*

(1) *$K \subsetneq \mathcal{O} \subsetneq F$, and*

(2) *for any $z \in F$, $z \in \mathcal{O}$ or $z^{-1} \in \mathcal{O}$.*

This definition is motivated by the following observation in the case of a rational function field $K(x)$; given an irreducible polynomial $p(x) \in K[x]$ consider the set

$$\mathcal{O}_{p(x)} = \left\{ \frac{f(x)}{g(x)} \middle| \, f(x), g(x) \in K[x],\, p(x) \nmid g(x) \right\}.$$

It is easily verified that $\mathcal{O}_{p(x)}$ is a valuation ring of $K(x)/K$. Note that if $q(x)$ is another irreducible polynomial, then $\mathcal{O}_{p(x)} \neq \mathcal{O}_{q(x)}$, in fact $\mathcal{O}_{p(x)} \not\subset \mathcal{O}_{q(x)}$ and $\mathcal{O}_{q(x)} \not\subset \mathcal{O}_{p(x)}$.

Proposition 2.2.7. *Let $\mathcal{O}$ be a valuation ring of the function field F/K. Then*

(a) *$\mathcal{O}$ is a local ring, i.e. $\mathcal{O}$ has a unique maximal ideal $P = \mathcal{O} \backslash \mathcal{O}^*$, where $\mathcal{O}^*$ denotes the group of units of $\mathcal{O}$.*

(b) *For* $0 \neq x \in F$, $x \in P \iff x^{-1} \notin \mathcal{O}$.

We have the following theorem:

Theorem 2.2.8. *Let* $\mathcal{O}$ *be a valuation ring of the function field* F/K *and* P *be its unique maximal ideal. Then*

(a) P *is a principal ideal.*

(b) *If* $P = t\mathcal{O}$ *then any* $0 \neq z \in F$ *has a unique representation of the form* $z = t^n u$ *for some* $n \in \mathbb{Z}$, $u \in \mathcal{O}^*$.

(c) $\mathcal{O}$ *is a principal ideal domain. More precisely, if* $P = t\mathcal{O}$ *and* I *is a non-trivial ideal of* $\mathcal{O}$ *then* $I = t^n\mathcal{O}$ *for some* $n \in \mathbb{N}$.

A ring having the above properties is called a **discrete valuation ring**. The reason for this terminology will become clear later.

Definition 2.2.9.

(1) *A* **place** P *of the function field* F/K *is the maximal ideal of some valuation ring* $\mathcal{O}$ *of* F/K. *Any element* $t \in P$ *such that* $P = t\mathcal{O}$ *is called a* **prime element** *for* P *(other notations are* **local parameter** *or* **uniformizing variable***).*

(2) $\mathbb{P}_F = \{P | \, P \text{ is a place of } F/K\}$.

If $\mathcal{O}$ is a valuation ring F/K and P its maximal ideal, then $\mathcal{O}$ is uniquely determined by P, namely $\mathcal{O} = \{z \in F | \, z^{-1} \notin P\}$, cf. Proposition 2.2.7(b). Hence $\mathcal{O}_P = \mathcal{O}$ is called the **the valuation ring of the place** P.

A second useful description of places is given in terms of valuations.

Definition 2.2.10. *A* **discrete valuation** *of* F/K *is a function* $v : F \to \mathbb{Z} \cup \{\infty\}$ *with the following properties:*

(1) $v(x) = \infty \iff x = 0$.

(2) $v(xy) = v(x) + v(y)$ *for all* $x, y \in F$.

(3) $v(x + y) \geq \min\{v(x), v(y)\}$ *for all* $x, y \in F$.

(4) *There exists an element* $z \in F$ *with* $v(z) = 1$.

(5) $v(a) = 0$ *for any* $a \in K$.

In this context, the symbol ∞ means some element not in $\mathbb{Z}$ such that $\infty + \infty = \infty + n = n + \infty = \infty$ and $\infty > m$ for all $m, n \in \mathbb{Z}$. It is obvious that v is surjective. Property (3) is called the **Strong Triangle Inequality**. A stronger version of this inequality can be derived from the axioms and is often very useful:

Lemma 2.2.11 (Strict Triangle Inequality). *Let v be a discrete valuation of F/K and $x, y \in F$ with $v(x) \neq v(y)$. Then $v(x + y) = \min\{v(x), v(y)\}$.*

Definition 2.2.12. *To any place $P \in \mathbb{P}_F$ we associate a function $v_P : F \to \mathbb{Z} \cup \{\infty\}$ which can be shown to be a discrete valuation of F/K: Choose a prime element t for P. Then every $0 \neq z \in F$ has a unique representation $z = t^n u$ with $u \in \mathcal{O}_P^*$ and $n \in \mathbb{Z}$. Define $v_P(z) = n$ and $v_P(0) = \infty$.*

Observe that this definition depends only on P, not on the choice of t. If $\tilde{t}$ is another prime element of P then $P = t\mathcal{O}_P = \tilde{t}\mathcal{O}_P$, so $t = \tilde{t}w$ for some $w \in \mathcal{O}_P^*$. Therefore $t^n u = (\tilde{t}^n w^n)u = \tilde{t}^n(w^n u)$ with $w^n u \in \mathcal{O}_P^*$.

Let P be a place of F/K and $\mathcal{O}_P$ be its valuation ring. Since P is a maximal ideal, the residue class ring $\mathcal{O}_P/P$ is a field. For $x \in \mathcal{O}_P$ we define $x(P) \in \mathcal{O}_P/P$ to be the residue class of x modulo P, for $x \in F \setminus \mathcal{O}_P$ we put $x(P) = \infty$ (note that the symbol ∞ has a different meaning here from the one used in Definition 2.2.10). It is easy to see that for a place $P \in \mathbb{P}_F$ we have that $K \subset \mathcal{O}_P$ and $K \cap P = \{0\}$. Therefore, there is a natural embedding of K into $F_P = \mathcal{O}_P/P$. This leads to the following definition.

Definition 2.2.13. *Let $P \in \mathbb{P}_F$.*

(1) *$F_P = \mathcal{O}_P/P$ is the* **residue class field** *of P. The map $x \to x(P)$ from F to $F_P \cup \{\infty\}$ is called the* **residue class map** *with respect to P.*

(2) *$\deg P = [F_P : K]$ is called the degree of P.*

It can be shown that the degree of P is always finite, i.e. F_P is a finite field extension of the field of constants K.

Remark 2.2.14. For the case when $\deg P = 1$, we have $F_P = K$, and the residue class map maps F to $K \cup \{\infty\}$. In particular, if K is an algebraically closed field, every place has degree one, so we can interpret an element $z \in F$ as a function

$$z : \begin{cases} \mathbb{P}_F & \longrightarrow K \cup \{\infty\} \\ P & \longmapsto z(P). \end{cases} \tag{2.2.15}$$

This is why F/K is called a **function field**. The elements of K, interpreted as functions in the sense of (2.2.15), are constant functions. For this reason K is called the **constant field** of F. Also, the following terminology is justified by (2.2.15):

Definition 2.2.16. *Let $z \in F$ and $P \in \mathbb{P}_F$. We say that P is a* **zero** *of z iff $v_P(z) > 0$ (i.e. $z(P) = 0$); P is a* **pole** *of z iff $v_P(z) < 0$ (i.e. $z(P) = \infty$). If $v_P(z) = m > 0$, P is a zero of z of* **order** *m; if $v_P(z) = -m < 0$, P is a* **pole** *of z of* **order** *m.*

2.3 Divisors

It can be shown that $\mathbb{P}_F \neq \emptyset$, in fact there are infinitely many places of F/K. Hence, the following definition makes sense:

Definition 2.3.1. *The (additively written) free abelian group which is generated by the places of F/K is denoted by $\mathcal{D}_F$ and called the* **divisor group** *of F/K.*

The elements of $\mathcal{D}_F$ are called **divisors** of F/K. In other words, a divisor is a formal sum

$$D = \sum_{P \in \mathbb{P}_F} n_P P \text{ with } n_P \in \mathbb{Z}, \text{ almost all}^\dagger\ n_P = 0.$$

The **support** of D is defined by

$$\operatorname{supp} D = \{P \in \mathbb{P}_F | \, n_P \neq 0\}.$$

It will often be found convenient to write

$$D = \sum_{P \in S} n_P P,$$

where $S \subseteq \mathbb{P}_F$ is a finite set with $S \supseteq \operatorname{supp} D$.

Two divisors $D = \sum n_P P$ and $D' = \sum n'_P P$ are added coefficient-wise:

$$D + D' = \sum_{P \in \mathbb{P}_F} (n_P + n'_P) P.$$

The zero element of the divisor group $\mathcal{D}_F$ is the divisor

$$0 = \sum_{P \in \mathbb{P}_F} r_P P, \text{ all } r_P = 0.$$

For $Q \in \mathbb{P}_F$ and $D = \sum n_P P \in \mathcal{D}_F$ we define $v_Q(D) = n_Q$, therefore

$$\operatorname{supp} D = \{P \in \mathbb{P}_F | \, v_P(D) \neq 0\} \quad \text{and} \quad D = \sum_{P \in \operatorname{supp} D} v_P(D) \cdot P.$$

[†] The terminology *almost all* means *all but finitely many.*

A partial ordering on $\mathcal{D}_F$ is defined by

$$D_1 \leq D_2 \iff v_P(D_1) \leq v_P(D_2) \text{ for all } P \in \mathbb{P}_F.$$

A divisor $D \geq 0$ is called **positive** (or **effective**). The **degree** of a divisor is defined by

$$\deg D = \sum_{P \in \mathbb{P}_F} v_P(D) \cdot \deg P$$

and this yields a homomorphism $\deg : \mathcal{D}_F \mapsto \mathbb{Z}$.

It can be shown that any non-zero element $x \in F$ has only finitely many zeros and poles in $\mathbb{P}_F$. Hence we have the following definition:

Definition 2.3.2. *Let $x \in F^*$ and denote by Z (resp. N) the set of zeros (poles) of x in $\mathbb{P}_F$. Then we define*

$$(x)_0 = \sum_{P \in Z} v_P(x)P \quad \text{, the } \textbf{zero divisor} \text{ of } x,$$

$$(x)_\infty = \sum_{P \in N} (-v_P(x))P \quad \text{, the } \textbf{pole divisor} \text{ of } x,$$

$$(x) = (x)_0 - (x)_\infty \quad \text{, the } \textbf{principal divisor} \text{ of } x.$$

Clearly $(x)_0 \geq 0$, $(x)_\infty \geq 0$ and

$$(x) = \sum_{P \in \mathbb{P}_F} v_P(x)P.$$

The elements $0 \neq x \in F$ which are constant are characterised by

$$x \in K \iff (x) = 0.$$

Definition 2.3.3. *Let*

$$\mathcal{P}_F = \{(x) | \, 0 \neq x \in F\}.$$

This is called the **group of principal divisors** *of F/K. It is clearly a subgroup of $\mathcal{D}_F$. The factor group*

$$\mathcal{C}_F = \mathcal{D}_F / \mathcal{P}_F$$

is called the **divisor class group**. *For a divisor $D \in \mathcal{D}_F$, the corresponding element in the factor group $\mathcal{C}_F$ is denoted by $[D]$, the* **divisor class** *of D. Let D_1 and D_2 be two divisors in $\mathcal{D}_F$. We shall say that D_1 is* **equivalent** *to D_2, written*

$$D_1 \sim D_2,$$

if $[D_1] = [D_2]$, i.e. $D_1 = D_2 + (x)$ for some $x \in F \setminus \{0\}$. This is easily verified to be an equivalence relation.

The next definition plays an important rôle in the theory of algebraic function fields.

Definition 2.3.4. *For a divisor $A \in \mathcal{D}_F$ we set*

$$\mathcal{L}(A) = \{x \in F | \, (x) \geq -A\} \cup \{0\}.$$

We shall summarise the properties of $\mathcal{L}(A)$ in the following theorem.

Theorem 2.3.5. *Let $A \in \mathcal{D}_F$. Then*

(a) *$x \in \mathcal{L}(A)$ iff $v_P(x) \geq -v_P(A)$ for all $P \in \mathbb{P}_F$.*

(b) *$\mathcal{L}(A) \neq \{0\}$ iff there exists a divisor $A' \sim A$ with $A' \geq 0$.*

(c) *$\mathcal{L}(A)$ is a finite dimensional vector space over K. The integer $\dim A = \dim \mathcal{L}(A)$ is called the* **dimension** *of the divisor A.*

(d) *If $A' \sim A$ then $\mathcal{L}(A') \cong \mathcal{L}(A)$.*

(e) *$\mathcal{L}(0) = K$.*

(f) *If $A < 0$ then $\mathcal{L}(A) = \{0\}$.*

The following theorem and its corollary will be used over and over again later on. The theorem states that an element $0 \neq x \in F$ has as many zeros as poles, provided they are counted properly.

Theorem 2.3.6. *Every principal divisor has degree zero. More precisely: let $x \in F \setminus K$ and $(x)_0$ and $(x)_\infty$ denote the zero and pole divisor of x. Then*

$$\deg(x)_0 = \deg(x)_\infty = [F : K(x)].$$

Corollary 2.3.7.

(a) *Let A, A' be divisors with $A \sim A'$. Then we have $\dim A = \dim A'$ and $\deg A = \deg A'$.*

(b) *If $\deg A < 0$ then $\dim A = 0$.*

(c) *For a divisor A of degree zero, the following are equivalent:*

(1) *A is principal.*

(2) *$\dim A \geq 1$.*

(3) *$\dim A = 1$.*

We will now give the definition of the genus of a function field which is one of the most important quantities associated with algebraic function fields. However, we need one further result first.

Proposition 2.3.8. *There is a constant $\gamma \in \mathbb{Z}$ such that, for all divisors $A \in \mathcal{D}_F$, the following holds:*

$$\deg A - \dim A \leq \gamma$$

Hence the following definition makes sense:

Definition 2.3.9. *The* **genus** *of F/K is defined by*

$$g = \max\{\deg A - \dim A + 1 \mid A \in \mathcal{D}_F\}.$$

We note that g is a non-negative integer. Finally in this section we have the following theorem.

Theorem 2.3.10 (Riemann's Theorem). *Let F/K be a function field of genus g.*

(a) *For any divisor $A \in \mathcal{D}_F$, $\dim A \geq \deg A + 1 - g$.*

(b) *There is an integer c, depending on F/K, such that*

$$dim A = \deg A + 1 - g$$

whenever $\deg A \geq c$.

In the next section we will introduce all the definitions and results needed to state the famous Riemann-Roch Theorem, one of its corollaries shows that the constants c of Theorem 2.3.10(b) can be taken to be $2g - 1$.

2.4 The Riemann-Roch Theorem

In this section, F/K denotes an algebraic function field of genus g.

Definition 2.4.1. *An* **adele** *of F/K is a mapping*

$$\alpha : \begin{cases} \mathbb{P}_F & \longrightarrow F, \\ P & \longmapsto \alpha_P, \end{cases}$$

such that $\alpha_P \in \mathcal{O}_P$ for almost all $P \in \mathbb{P}_F$. We regard an adele as an element of the direct product $\prod_{P \in \mathbb{P}_F} F$ and therefore use the notation $\alpha = (\alpha_P)_{P \in \mathbb{P}_F}$ or, even shorter, $\alpha = (\alpha_P)$. The set

$$\mathcal{A}_F = \{\alpha | \, \alpha \text{ is an adele of } F/K\}$$

is called the **adele space** *of F/K. It is regarded as a vector space over K in the obvious manner.*

The **principal adele** *of an element $x \in F$ is the adele all of whose components are equal to x (note that this definition makes sense because each element of F has only finitely many poles). This gives an embedding $F \hookrightarrow \mathcal{A}_F$. The valuations v_P of F/K extend naturally to $\mathcal{A}_F$ by setting $v_P(\alpha) = v_P(\alpha_P)$ (where α_P is the P-component of the adele α). By definition, $v_P(\alpha) \geq 0$ for almost all $P \in \mathbb{P}_F$.*

We also need the following definition.

Definition 2.4.2. *For $A \in \mathcal{D}_F$ we define*

$$\mathcal{A}_F(A) = \{\alpha \in \mathcal{A}_F \,|\, v_P(\alpha) \geq -v_P(A) \text{ for all } P \in \mathcal{P}_F\}.$$

Obviously, this is a K-subspace of $\mathcal{A}_F$.

Definition 2.4.3. *A* **Weil differential** *of F/K is a K-linear map $\omega : \mathcal{A}_F \to K$ vanishing on $\mathcal{A}_F(A) + F$ for some divisor $A \in \mathcal{D}_F$. We call*

$$\Omega_F = \{\omega \,|\, \omega \text{ is a Weil differential of } F/K\}$$

the **module of Weil differentials** *of F/K. For $A \in \mathcal{D}_F$ let*

$$\Omega(A) = \{\omega \in \Omega_F \,|\, \omega \text{ vanishes on } \mathcal{A}_F(A) + F\}.$$

We regard Ω_F as a K-vector space in the obvious manner (in fact, if ω_1 vanishes on $\mathcal{A}_F(A_1)+F$ and ω_2 vanishes on $\mathcal{A}_F(A_2) + F$ then $\omega_1 + \omega_2$ vanishes on $\mathcal{A}_F(A_3) + F$ for any divisor A_3 with $A_3 \leq A_1$ and $A_3 \leq A_2$, and $a\omega_1$ vanishes on $\mathcal{A}_F(A_1) + F$ for $a \in K$). Clearly, $\Omega_F(A)$ is a subspace of Ω_F.

The next definition allows us to regard Ω_F as a vector space over F, instead.

Definition 2.4.4. *For $x \in F$ and $\omega \in \Omega_F$ define $x\omega : \mathcal{A}_F \to K$ by*

$$(x\omega)(\alpha) = \omega(x\alpha).$$

It is easily checked that $x\omega$ is again a Weil differential of F/K. In fact, if ω vanishes on $\mathcal{A}_F(A)+F$ then $x\omega$ vanishes on $\mathcal{A}_F(A+(x))+F$. It turns out that Ω_F is a one-dimensional vector space over F. We are now going to associate with a Weil differential a divisor which, together with another definition, will then enable us to state the main theorem of this section. To this end, consider (for a fixed ω) the set of divisors

$$M(\omega) = \{A \in \mathcal{D}_F \,|\, \omega \text{ vanishes on } \mathcal{A}_F(A) + F\}.$$

Lemma 2.4.5. *Let $0 \neq \omega \in \Omega_F$. Then there is a uniquely determined divisor $W \in M(\omega)$ such that $A \leq W$ for any $A \in M(\omega)$.*

Because of this lemma, the next definition makes sense.

Definition 2.4.6.

(1) *The* **divisor** (ω) *of a Weil differential $\omega \neq 0$ is the uniquely determined divisor of F/K satisfying*

   (a) *ω vanishes on $\mathcal{A}_F((\omega)) + F$.*

   (b) *If ω vanishes on $\mathcal{A}_F(A) + F$ then $A \leq (\omega)$.*

(2) *For $0 \neq \omega \in \Omega_F$ and $P \in \mathbb{P}_F$ we define $v_P(\omega) = v_P((\omega))$.*

(3) *A place P is said to be a* **zero** *(resp. a* **pole***) of ω if $v_P(\omega) > 0$ (resp. $v_P(\omega) < 0$).*

(4) *A divisor W is called a* **canonical divisor** *of F/K if $W = (\omega)$ for some $\omega \in \Omega_F$.*

It follows immediately from the definitions that

$$\Omega_F(A) = \{\omega \in \Omega_F | \, \omega = 0 \text{ or } (\omega) \geq A\}.$$

We have the following result.

Proposition 2.4.7.

(a) *For $0 \neq x \in F$ and $0 \neq \omega \in \Omega_F$ we have $(x\omega) = (x) + (\omega)$.*

(b) *Any two canonical divisors of F/K are equivalent.*

A simple consequence of this proposition is that the canonical divisors of F/K form a whole class $[W]$ in the divisor class group $\mathcal{C}_F$; this divisor class is called the **canonical class** of F/K.

Theorem 2.4.8 (Riemann-Roch-Theorem). *Let W be a canonical divisor of F/K. Then, for any $A \in \mathcal{D}_F$,*

$$\dim A = \deg A + 1 - g + \dim(W - A).$$

Corollary 2.4.9. *For a canonical divisor W, we have*

$$\deg W = 2g - 2 \quad \textit{and} \quad \dim W = g.$$

Proposition 2.4.10. *A divisor B is canonical if and only if* $\deg B = 2g - 2$ *and* $\dim B \geq g$.

From Riemann's Theorem we already know that there is a constant c such that $\dim A = \deg A + 1 - g$ whenever $\deg A \geq c$. The next theorem shows that the constant can be chosen to be $2g - 1$.

Theorem 2.4.11. *If A is a divisor of F/K of degree $\geq 2g - 1$ then*

$$\dim A = \deg A + 1 - g.$$

This bound is best possible.

We conclude this section with an inequality for the dimension of an arbitrary divisor of degree $\leq 2g - 2$.

Theorem 2.4.12 (Clifford's Theorem). *For any divisor A with $0 \leq \deg A \leq 2g - 2$ we have*

$$\dim A \leq 1 + \frac{1}{2} \deg A$$

2.5 Algebraic Extensions of Function Fields

Every function field over K can be regarded as a finite field extension of a rational function field $K(x)$. This is one of the reasons why it is of interest to investigate field extensions F'/F of algebraic function fields. We shall quote all the relevant results concerning field extensions of algebraic function fields which will be needed in the remaining chapters. We first fix some notation for the rest of this section.

Convention 2.5.1. *F/K denotes an algebraic function field of one variable with full constant field K. We consider function fields F'/K' (where K' is the full constant field of F') such that $F' \supseteq F$ is an algebraic extension and $K' \supseteq K$.*

Definition 2.5.2.

(1) *An algebraic function field F'/K' is called an* **algebraic extension** *of F/K if $F' \supseteq F$ is an algebraic field extension and $K' \supseteq K$.*

(2) *The algebraic extension F'/K' of F/K is called a* **finite extension** *if $[F' : F] < \infty$.*

Definition 2.5.3. *Consider an algebraic extension of F'/K' of F/K. A place $P' \in \mathbb{P}_{F'}$ is said to* **lie over** *$P \in \mathbb{P}_F$ if $P \subseteq P'$. We also say that P' is an* **extension** *of P or that P* **lies under** *P', and we write $P'|P$.*

Proposition 2.5.4. *Let F'/K' be an algebraic extension of F/K. Suppose that P (resp. P') is a place of F/K (resp. F'/K'), and let $\mathcal{O}_P \subseteq F$ (resp. $\mathcal{O}_{P'} \subseteq F'$) denote the corresponding valuation ring, v_P (resp. $v_{P'}$) the corresponding discrete valuation. Then the following assertions are equivalent:*

(1) *$P'|P$.*

(2) *$\mathcal{O}_P \subseteq \mathcal{O}_{P'}$.*

(3) *There exists an integer $e \geq 1$ such that $v_{P'}(x) = e \cdot v_P(x)$ for all $x \in F$.*

Moreover, if $P'|P$ then

$$P = P' \cap F \text{ and } \mathcal{O}_P = \mathcal{O}_{P'} \cap F.$$

For this reason, P is also called the restriction of P' to F.

A consequence of the above proposition is that for $P'|P$ there is a canonical embedding of the residue class field $F_P = \mathcal{O}_P/P$ into the residue class field $F'_{P'} = \mathcal{O}_{P'}/P'$, given by

$$x(P) \mapsto x(P') \quad \text{for } x \in \mathcal{O}_P.$$

Therefore we can consider F_P as a subfield of $F'_{P'}$.

Proposition 2.5.5. *Let F'/K' be an algebraic extension of F/K.*

(a) *For each place $P' \in \mathbb{P}_{F'}$ there is exactly one place $P \in \mathbb{P}_F$ such that $P'|P$, namely $P = P' \cap F$.*

(b) *Conversely, every place $P \in \mathbb{P}_F$ has at least one, but only finitely many extensions $P' \in \mathbb{P}_{F'}$.*

The preceding results are the motivation for the following definition.

Definition 2.5.6. *Let F'/K' be an algebraic extension of F/K, and let $P' \in \mathbb{P}_{F'}$ be a place of F'/K' lying over $P \in \mathbb{P}_F$.*

(1) *The integer $e(P'|P) = e$ with*

$$v_{P'}(x) = e \cdot v_P(x) \text{ for any } x \in F$$

is called the **ramification index** *of P' over P.*

(2) *We say that $P'|P$ is* **ramified** *if $e(P'|P) > 1$, and $P'|P$ is* **unramified** *if $e(P'|P) = 1$.*

(3) *P is said to be* **totally ramified** *in F'/F if there is only one extension $P' \in \mathbb{P}_{F'}$ of P in F', and the ramification index is $e(P'|P) = [F' : F]$.*

(4) P' *is said to be* **tamely** *(resp.* **wildly***)* **ramified** *if* $e(P'|P) > 1$ *and* $\operatorname{char} K$ *does not divide* $e(P'|P)$ *(resp.* $\operatorname{char} K$ *divides* $e(P'|P)$*).*

(5) $f(P'|P) = [F'_{P'} : F_P]$ *is called the* **relative degree** *of* P' *over* P*.*

Proposition 2.5.7. *Let* F'/K' *be an algebraic extension of* F/K *and* P' *be a place of* F'/K' *lying over* $P' \in \mathbb{P}_F$*. Then*

(a) $f(P'|P) < \infty \iff [F' : F] < \infty$.

(b) *If* F''/K'' *is an algebraic extension of* F'/K' *and* $P'' \in \mathbb{P}_{F''}$ *is an extension of* P'*, then*

$$e(P''|P) = e(P''|P') \cdot e(P'|P),$$
$$f(P''|P) = f(P''|P') \cdot f(P'|P).$$

Theorem 2.5.8. *Let* F'/K' *be a finite extension of* F/K*,* P *a place of* F/K *and* $P_1, \ldots, P_m$ *all the places of* F'/K' *lying over* P*. Let* $e_i = e(P_i|P)$ *denote the ramification index and* $f_i = f(P_i|P)$ *the relative degree of* $P_i|P$*. Then*

$$\sum_{i=1}^{m} e_i f_i = [F' : F].$$

Corollary 2.5.9. *Let* F'/K' *be a finite extension of* F/K *and* $P \in \mathbb{P}_F$*. Then*

(a) $|\{P' \in \mathbb{P}_{F'} \mid P' \text{ lies over } P\}| \leq [F' : F]$.

(b) *If* $P' \in \mathbb{P}_{F'}$ *lies over* P *then* $e(P'|P) \leq [F' : F]$ *and* $f(P'|P) \leq [F' : F]$.

For the last two results of this section we assume that F/K is a function field of genus g with full constant field $K = \mathbb{F}_q$, i.e. K is the finite field of $q = p^r$ elements (p prime, $r \geq 1$). Because of applications in coding theory which will be described in the next chapter, we are interested in finding bounds on the number of places of degree one of a function field. The following theorems do just that. Denote by $N(F) = |\{P \in \mathbb{P}_F \mid \deg P = 1\}|$ the number of places in $F/\mathbb{F}_q$ of degree 1.

Theorem 2.5.10 (Hasse-Weil Bound). *The number* $N = N(F)$ *of places of* $F/\mathbb{F}_q$ *of degree one can be estimated by*

$$|N - (q+1)| \leq 2gq^{1/2}.$$

This bound can be sharpened slightly.

Theorem 2.5.11 (Serre Bound). *For a function field* $F/\mathbb{F}_q$ *of genus* g*, the number of places of degree one is bounded by*

$$|N - (q+1)| \leq g[2q^{1/2}].$$

2.6 Examples of Function Fields

In this section we will look at examples of function fields which will play a crucial rôle in the later chapters.

2.6.1 Rational function field

We have already come across the rational function field F/K, where $F = K(x)$ with x transcendental over K, in Example 2.2.4. Given an arbitrary monic, irreducible polynomial $p(x) \in K[x]$ consider the valuation ring

$$\mathcal{O}_{p(x)} = \left\{ \frac{f(x)}{g(x)} \middle| f(x), g(x) \in K[x], p(x) \nmid g(x) \right\} \tag{2.6.1}$$

of $K(x)/K$ with maximal ideal

$$P_{p(x)} = \left\{ \frac{f(x)}{g(x)} \middle| f(x), g(x) \in K[x], p(x) \mid f(x), p(x) \nmid g(x) \right\}. \tag{2.6.2}$$

In the particular case when $p(x)$ is linear, i.e. $p(x) = x - \alpha$ with $\alpha \in K$, we abbreviate and write

$$P_\alpha = P_{x-\alpha} \in \mathbb{P}_{K(x)}. \tag{2.6.3}$$

There is another valuation ring of $K(x)/K$, namely

$$\mathcal{O}_\infty = \left\{ \frac{f(x)}{g(x)} \middle| f(x), g(x) \in K[x], \deg f(x) \leq \deg g(x) \right\} \tag{2.6.4}$$

with maximal ideal

$$P_\infty = \left\{ \frac{f(x)}{g(x)} \middle| f(x), g(x) \in K[x], \deg f(x) < \deg g(x) \right\}. \tag{2.6.5}$$

This is called the **infinite** place of $K(x)$.

We shall summarise the main properties of the rational function field in the following proposition.

Proposition 2.6.6. *Let $F = K(x)$ be the rational function field.*

(a) *Let $P = P_{p(x)} \in \mathbb{P}_{K(x)}$ be the place defined by* (2.6.2)*, where $p(x) \in K[x]$ is an irreducible polynomial. Then $p(x)$ is a prime element for P, and the corresponding discrete valuation v_P can be described as follows: if $z \in K(x) \setminus \{0\}$ is written in the form $z = p(x)^n \cdot (f(x)/g(x))$ with $n \in \mathbb{Z}$, $f(x)$, $g(x) \in K[x]$, $p(x) \nmid f(x)$, $p(x) \nmid g(x)$, then $v_P(z) = n$. The residue class field $K(x)_P = \mathcal{O}_P/P$ is isomorphic to $K[x]/(p(x))$; an isomorphism given by*

$$\phi : \begin{cases} K[x]/(p(x)) & \longrightarrow K(x)_P \\ f(x) \pmod{p(x)} & \longmapsto f(x)(P). \end{cases}$$

Consequently, $\deg P = \deg p(x)$.

(b) *In the special case* $p(x) = x - \alpha$ *with* $\alpha \in K$*, the degree of* $P = P_\alpha$ *is one, and the residue class map is given by*

$$z(P) = z(\alpha) \text{ for } z \in K(x),$$

where $z(\alpha)$ *is defined as follows: write* $z = f(x)/g(x)$ *with coprime polynomials* $f(x), g(x) \in K[x]$*. Then*

$$z(\alpha) = \begin{cases} f(\alpha)/g(\alpha) & \text{if } g(\alpha) \neq 0, \\ \infty & \text{if } g(\alpha) = 0. \end{cases}$$

(c) *Finally, let* $P = P_\infty$ *be the infinite place of* $K(x)/K$ *defined by* (2.6.5)*. Then* $\deg P_\infty = 1$*. A prime element for* P_∞ *is* $t = 1/x$*. The corresponding discrete valuation* v_∞ *is given by*

$$v_\infty(f(x)/g(x)) = \deg g(x) - \deg f(x),$$

where $f(x), g(x) \in K[x]$*. The residue class map corresponding to* P_∞ *is determined by* $z(P_\infty) = z(\infty)$ *for* $z \in K(x)$*, where* $z(\infty)$ *is defined as usual: if*

$$z = \frac{a_n x^n + \ldots + a_0}{b_m x^m + \ldots + b_0} \text{ with } a_n, b_m \neq 0,$$

then

$$z(\infty) = \begin{cases} a_n/b_m & \text{if } n = m, \\ 0 & \text{if } n < m, \\ \infty & \text{if } n > m. \end{cases}$$

(d) K *is the full constant field of* $K(x)/K$*.*

Theorem 2.6.7. *There are no places of the rational function field* $K(x)/K$ *other than the places* $P_{p(x)}$ *and* P_∞*, defined by* (2.6.2) *and* (2.6.5)*.*

Proposition 2.6.8. *For a function field* F/K *the following conditions are equivalent:*

(1) F/K *is rational, i.e.* $F = K(x)$ *for some* x *which is transcendental over* K*.*

(2) F/K *has genus* 0*, and there is some divisor* $A \in \mathcal{D}_F$ *with* $\deg A = 1$*.*

2.6.2 Elliptic and hyperelliptic function fields

The simplest non-rational function fields are the elliptic function fields. We will also consider hyperelliptic function fields which in some sense (to be made more concise later) include the elliptic function fields as a special case.

First of all we define what we mean by an elliptic and a hyperelliptic function field.

Definition 2.6.9. *An algebraic function field F/K (K perfect) is said to be an* **elliptic** *function field if the following conditions hold:*

(1) *the genus of F/K is $g = 1$, and*

(2) *there exists a divisor $A \in \mathcal{D}_F$ with $\deg A = 1$.*

Definition 2.6.10. *An algebraic function field F/K (K perfect) is said to be a* **hyperelliptic** *function field if the following conditions hold:*

(1) *the genus of F/K is $g \geq 2$, and*

(2) *there exists a divisor $A \in \mathcal{D}_F$ with $\deg A = 2$ and $\dim A \geq 2$.*

Remark. Note that both definition are given in terms of an arbitary perfect field K. In all our applications we have that $K = \mathbb{F}_q$ which means that in the elliptic function field case condition (2) of Definition 2.6.9 always holds (cf. footnote 47 in [Sti91, p. 188]).

The reason why we consider elliptic function fields as a special case of hypperelliptic function fields is that elliptic function fields automatically satisfy condition (2) in Definition 2.6.10.

The following proposition describes hyperelliptic function fields in even characteristic.

Proposition 2.6.11. *Assume that* $\operatorname{char} \mathbb{F}_q = 2$.

(1) *Let $F/\mathbb{F}_q$ be a (hyper-)elliptic[‡] function field of genus g. Then there exist $x, y \in F$ such that $F = \mathbb{F}_q(x, y)$ and*

$$H(x, y) = y^2 + h(x)y + f(x) = 0 \tag{2.6.12}$$

with polynomials $h \in \mathbb{F}_q[x]$ and $f \in \mathbb{F}_q[x]$ such that all zeros of h in $\overline{\mathbb{F}_q}$ are simple zeros of f and ($\deg(h) \leq g$ and $\deg(f) = 2g + 1$) or ($\deg(h) = g + 1$ and $\deg(f) \leq 2g + 2$).

(2) *Conversely, let g be an integer such that $g \geq 1$.*
If $F = \mathbb{F}_q(x, y)$ and $y^2 + h(x)y + f(x) = 0$ with polynomials f and h as in (2.6.12) then $F/\mathbb{F}_q$ is a (hyper-)elliptic function field of genus g.

(3) *Let $F = \mathbb{F}_q(x, y)$ and $y^2 + h(x)y + f(x) = 0$ as in (2.6.12). Then the places of $\mathbb{F}_q(x)/\mathbb{F}_q$ which are ramified in $F/\mathbb{F}_q$ are*

all zeros of $h(x)$ if $\deg(h) = g + 1$,

all zeros of $h(x)$ and the pole of x if $\deg(h) \leq g$.

[‡] As an abbreviation we shall use **(hyper-)elliptic** to mean either elliptic or hyperelliptic

Proof. see [LeB96] and [Sti91, p.188 Proposition VI.1.3(b)] □

For (hyper-)elliptic function fields with odd characteristic we have the following.

Proposition 2.6.13. *Assume that* $\operatorname{char} \mathbb{F}_q \neq 2$.

(1) *Let* $F/\mathbb{F}_q$ *be a (hyper-)elliptic function field of genus* g. *Then there exist* $x, y \in F$ *such that* $F = \mathbb{F}_q(x, y)$ *and*

$$H(x, y) = y^2 - f(x) = 0 \tag{2.6.14}$$

where $f(x) \in \mathbb{F}_q[x]$ *is square-free of degree* $d = 2g + 1$ *or* $2g + 2$.

(2) *Conversely, if* $F = \mathbb{F}_q(x, y)$, *where* $y^2 = f(x) \in \mathbb{F}_q[x]$ *with a square-free polynomial* $f(x)$ *of degree* $d \geq 3$ *then* $F/\mathbb{F}_q$ *is (hyper-)elliptic of genus*

$$g = \left[\frac{d-1}{2}\right].$$

(3) *Let* $F = \mathbb{F}_q(x, y)$ *with* $y^2 = f(x)$ *as in* (2.6.14). *Then the places* $P \in \mathbb{P}_{\mathbb{F}_q(x)}$ *which ramify in* $F/\mathbb{F}_q(x)$ *are the following:*

$$\text{all zeros of } f(x) \text{ if } \deg f(x) \equiv 0 \pmod 2,$$
$$\text{all zeros of } f(x) \text{ and the pole of } x \text{ if } \deg f(x) \equiv 1 \pmod 2.$$

Proof. see [Sti91, p.188 Proposition VI.1.3(a) and p.194, Proposition VI.2.3] □

Remark. Note that if f is a quartic polynomial in (2.6.12) or (2.6.14) we can always reduce this to a cubic polynomial because of condition (2) in Definition 2.6.9. However, for our purposes, this is not necessary.

Let $F = K(x, y)$ be an elliptic function field over K. It is well-known that in the elliptic function field case the places of degree one form an abelian group (cf [Sti91, pp. 192–193] and [Sil86, pp. 55–59]). We denote this group by $H(K)$ and use $\oplus$ to denote the group operation.

Lemma 2.6.15. *Let* $F = K(x, y)$ *be an elliptic function field over* K, *where* K *is perfect. Let* $P_1, \ldots, P_r$ *be places of degree one, and* $c_1, \ldots, c_r \in \mathbb{Z}$. *Then*

$$\sum_{i=1}^{r} c_i P_i \text{ is a principal divisor} \iff \sum_{i=1}^{r} c_i = 0 \text{ and } \oplus \sum_{i=1}^{r} c_i P_i = 0 \text{ in the group } H(K)$$

Proof. Consider $\mathcal{D}_F$, the divisor group of F and the subgroup $\mathcal{P}_F = \{(x) | \, 0 \neq x \in F\}$, i.e. all principal divisors in F. Recall that the quotient group $\mathcal{C}_F = \mathcal{D}_F/\mathcal{P}_F$ is called the divisor class

group (cf. Definition 2.3.3). The class of A in the divisor class group $\mathcal{C}_F$ is denoted by $[A]$. Recall that for any two divisors A and B in $\mathcal{D}_F$ we have that $A \sim B$ if $A = B + (u)$ for some principal divisor $(u) \in \mathcal{P}_F$. Hence $A \sim B$ if and only if $A \in [B]$ if and only if $[A] = [B]$. By Corollary 2.3.7 equivalent divisors have the same degree and dimension, so the integers

$$\deg[A] := \deg A \quad \text{and} \quad \dim[A] := \dim A$$

are well-defined. We can now define the following subgroup of the divisor class group:

$$\mathcal{C}_F^0 := \{[A] \in \mathcal{C}_F |\ \deg[A] = 0\}.$$

This group is called the **group of divisor classes of degree zero**.
Stichtenoth [Sti91, pp.192–193] proves that in the elliptic function field case we have an isomorphism of abelian groups:

$$H(\mathbb{F}_q) \cong \mathcal{C}_F^0.$$

The isomorphism is given by:

$$\phi : \begin{cases} H(\mathbb{F}_q) & \to \mathcal{C}_F^0 \\ P & \mapsto [P - Q_\infty] \end{cases}$$

and the group operation on $H(\mathbb{F}_q)$ is given by:

$$P \oplus Q := \phi^{-1}(\phi(P) + \phi(Q)).$$

The result now follows from the above isomorphism. (cf. also [Mor91, p.190],[Sil86, p.67]). □

2.6.3 Tame cyclic extensions of the rational function field

We summarise some of the properties of function fields $F = K(x, y)$ which are defined by an equation

$$y^n = a \cdot \prod_{i=1}^{s} p_i(x)^{n_i} \tag{2.6.16}$$

with $s > 0$ pairwise distinct irreducible monic polynomials $p_i(x) \in K[x]$, $0 \neq a \in K$ and $0 \neq n_i \in \mathbb{Z}$. We shall assume that the following conditions hold:

$$\operatorname{char} K \nmid n, \text{ and } \gcd(n, n_i) = 1 \text{ for } 1 \leq i \leq s. \tag{2.6.17}$$

Note that (hyper-)elliptic function fields of characteristic $\neq 2$ are a special case of (2.6.16).

Proposition 2.6.18. *Suppose that $F = K(x, y)$ is defined by* (2.6.16) *and* (2.6.17). *Then we have:*

(a) *K is the full constant field of F, and $[F : K(x)] = n$. If K contains a primitive n-th root of unity, then $F/K(x)$ is cyclic.*

(b) *Let P_i (resp. P_∞) denote the zero of $p_i(x)$ (resp. the pole of x) in $K(x)$. The places $P_1, \ldots, P_s$ are totally ramified in $F/K(x)$. All places $Q_\infty \in \mathbb{P}_F$ with $Q_\infty | P_\infty$ have ramification index $e(Q_\infty|P_\infty) = n/d$ where*

$$d = \gcd\left(n, \sum_{i=1}^{s} n_i \cdot \deg p_i(x)\right). \tag{2.6.19}$$

No places $P \in \mathbb{P}_{K(x)}$ other than $P_1, \ldots, P_s, P_\infty$ ramify in $F/K(x)$. Because $\operatorname{char} K \nmid n$, *they are all tamely ramified.*

(c) *The genus of F/K is*

$$g = \frac{n-1}{2}\left(-1 + \sum_{i=1}^{s} \deg p_i(x)\right) - \frac{d-1}{2},$$

with d as in (2.6.19).

Because of Proposition 2.6.18 (a) and (b) the fields defined by (2.6.16) are called tame cyclic extensions of the rational function field.

2.6.4 Some elementary abelian p-extensions of $K(x)$, $\operatorname{char} K = p > 0$

In this example we consider a class of function fields F/K where $\operatorname{char} K = p > 0$.

Proposition 2.6.20. *Consider a function field $F = K(x, y)$ with*

$$y^q + \mu y = f(x) \in K[x], \tag{2.6.21}$$

where $q = p^s > 1$ is a power of p and $0 \neq \mu \in K$. Assume that $\deg f(x) = m > 0$ is coprime to p, and that all roots of $T^q + \mu T = 0$ are in K. Then the following holds:

(a) *$[F : K(x)] = q$, and K is the full constant field.*

(b) *$F/K(x)$ is Galois. The set $A = \{\gamma \in K \,|\, \gamma^q + \mu\gamma = 0\}$ is a subgroup of order q of the additive group of K. For all $\sigma \in Gal(F/K(x))$, there is a unique γ such that $\sigma(y) = y + \gamma$, and the map*

$$\begin{cases} Gal(F/K(x)) & \longrightarrow A, \\ \sigma & \longmapsto \gamma \end{cases}$$

is an isomorphism of $Gal(F/K(x))$ onto A.

(c) *The pole $P_\infty \in \mathbb{P}_{K(x)}$ of x in $K(x)$ has a unique extension $Q_\infty \in \mathbb{P}_F$, and $Q_\infty|P_\infty$ is totally ramified (i.e. $e(Q_\infty|P_\infty) = q$). Hence Q_∞ is a place of F/K of degree one.*

(d) *P_∞ is the only place of $K(x)$ which ramifies in $F/K(x)$.*

(e) *The genus of F/K is $g = (q-1)(m-1)/2$.*

(f) *The pole divisor of x (resp. y) is $(x)_\infty = qQ_\infty$ (resp. $(y)_\infty = mQ_\infty$).*

(g) *Let $r \geq 0$. Then the elements $x^i y^j$ with*

$$0 \leq i,\ 0 \leq j \leq q-1,\ qi + mj \leq r$$

form a basis of the space $\mathcal{L}(rQ_\infty)$ over K.

(h) *For all $\alpha \in K$, one of the following holds:*
Case (1). The equation $T^q + \mu T = f(\alpha)$ has q distinct roots in K.
Case (2). The equation $T^q + \mu T = f(\alpha)$ has no root in K.
In case (1) for each β with $\beta^q + \mu\beta = f(\alpha)$ there exists a unique place $P_{\alpha,\beta}$ such that $P_{\alpha,\beta}|P_\alpha$ and $y(P_{\alpha,\beta}) = \beta$. Hence P_α has q distinct extensions in $F/K(x)$ each of degree one.
In case (2), all extensions of P_α have degree > 1.

Proof. [Sti91, pp.200–201] □

The following is a special case of the above and will be used later on.

Definition 2.6.22. *The function field $H = \mathbb{F}_{q^2}(x, y)$ over $\mathbb{F}_{q^2}$ defined by*

$$y^q + y = x^{q+1}$$

is called the **Hermitian function field**.

We summarise all its relevant properties in the following lemma:

Lemma 2.6.23. *Let $H = \mathbb{F}_{q^2}(x, y)$ be a Hermitian function field over $\mathbb{F}_{q^2}$ then*

(a) *The genus of H is $g = q(q-1)/2$.*

(b) *H has $q^3 + 1$ places of degree one over $\mathbb{F}_{q^2}$, namely*

(1) *the common pole Q_∞ of x and y, and*

(2) *for each $\alpha \in \mathbb{F}_{q^2}$, there are q elements $\beta \in \mathbb{F}_{q^2}$ such that $\beta^q + \beta = \alpha^{q+1}$, and for all such pairs (α, β) there exists a unique place $P_{\alpha,\beta} \in \mathbb{P}_H$ of degree one with $x(P_{\alpha,\beta}) = \alpha$ and $y(P_{\alpha,\beta}) = \beta$.*

(c) *For $r \geq 0$, the elements $x^i y^j$ with $0 \leq i, 0 \leq j \leq q-1$ and $iq + j(q+1) \leq r$ form a basis of $\mathcal{L}(rQ_\infty)$.*

(d) *Denote by $Aut(H/\mathbb{F}_{q^2})$ the set of all automorphisms of H over $\mathbb{F}_{q^2}$ (i.e $\sigma(a) = a$ for all $\sigma \in Aut(H/\mathbb{F}_{q^2})$ and $a \in \mathbb{F}_{q^2}$). Let $\epsilon \in \mathbb{F}_{q^2} \setminus \{0\}$, $\delta \in \mathbb{F}_{q^2}$ and $\mu^q + \mu = \delta^{q+1}$. Then $\mu \in \mathbb{F}_{q^2}$, and there exists an automorphism $\sigma \in Aut(H/\mathbb{F}_{q^2})$ with*

$$\sigma(x) = \epsilon x + \delta \text{ and } \sigma(y) = \epsilon^{q+1} y + \epsilon \delta^q x + \mu. \tag{2.6.24}$$

The set of all automorphisms (2.6.24) of $H/\mathbb{F}_{q^2}$ constitutes a group $\Gamma \subseteq Aut(H/\mathbb{F}_{q^2})$ of order $q^3(q^2-1)$. Clearly $\sigma(Q_\infty) = Q_\infty$ for all $\sigma \in \Gamma$ and σ permutes the places $P_{\alpha,\beta}$ of H since they are the only places of degree one other than Q_∞.

Chapter 3

Function Fields And Codes

3.1 Introduction

In this section we shall show how we can design codes from function fields. These codes are called **geometric Goppa codes** or **algebraic geometric codes** (AG codes for short). They were first introduced by V.D. Goppa, cf. [Gop88].

Again our exposition of the material follows very closely that of Stichtenoth [Sti91]. As before all proofs can be found in his book.

3.2 Geometric Goppa Codes

Convention 3.2.1. *We shall fix some notation valid for the entire chapter:*

- $F/\mathbb{F}_q$ *is an algebraic function field of genus* g.
- $P_1, P_2, \ldots, P_n$ *are pairwise distinct places of* $F/\mathbb{F}_q$ *of degree one.*
- $D = P_1 + \cdots + P_n$.
- G *is a divisor of* $F/\mathbb{F}_q$ *such that* $\operatorname{supp} G \cap \operatorname{supp} D = \emptyset$.

Definition 3.2.2. *The* **geometric Goppa** *code* $\mathcal{C}_{\mathcal{L}}(D, G)$ *associated with the divisors* D *and* G *is defined by*

$$\mathcal{C}_{\mathcal{L}}(D, G) = \{(x(P_1), \ldots, x(P_n)) | \, x \in \mathcal{L}(G)\} \subseteq \mathbb{F}_q^n.$$

Lemma 3.2.3. *We consider the* **evaluation map** $ev_D : \mathcal{L}(G) \to \mathbb{F}_q^n$ *given by*

$$ev_D(x) = (x(P_1), \ldots, x(P_n)) \in \mathbb{F}_q^n \tag{3.2.4}$$

This map is $\mathbb{F}_q$-linear and $\mathcal{C}_{\mathcal{L}}(D,G)$ is the image of $\mathcal{L}(G)$ under this map. If, furthermore, $\deg G < n$ *then ev_D is a bijection onto $\mathcal{C}_{\mathcal{L}}(D,G)$.*

The next theorem shows why these codes are interesting: one can calculate (or at least estimate) their parameters n, k, and d by means of the Riemann-Roch Theorem, and one obtains a lower bound for their minimum distance in a very general situation.

Theorem 3.2.5. *$\mathcal{C}_{\mathcal{L}}(D,G)$ is an $[n,k,d]$-code with parameters*

$$k = \dim G - \dim(G-D) \quad \textit{and} \quad d \geq n - \deg G.$$

Corollary 3.2.6. *Suppose that the degree of G is strictly less than n. Then the evaluation map $ev_D : \mathcal{L}(G) \to \mathcal{C}_{\mathcal{L}}(D,G)$ is injective, and we have:*

(a) *$\mathcal{C}_{\mathcal{L}}(D,G)$ is an $[n,k,d]$-code with*

$$d \geq n - \deg G \quad \textit{and} \quad k = \dim G \geq \deg G + 1 - g.$$

Hence

$$k + d \geq n + 1 - g. \tag{3.2.7}$$

(b) *If, in addition, $2g - 2 < \deg G < n$, then $k = \deg G + 1 - g$.*

(c) *If $\{x_1, x_2, \ldots, x_k\}$ is a basis of $\mathcal{L}(G)$ then the matrix*

$$M = \begin{pmatrix} x_1(P_1) & x_1(P_2) & \cdots & x_1(P_n) \\ \vdots & \vdots & & \vdots \\ x_k(P_1) & x_k(P_2) & \cdots & x_k(P_n) \end{pmatrix}$$

is a generator matrix for $\mathcal{C}_{\mathcal{L}}(D,G)$.

We point out that the **lower** bound (3.2.7) for the minimum distance looks very similar to the **upper** Singleton Bound. Putting both bounds together we see that, for $\deg G < n$,

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

Note that $k + d = n + 1$ if F is a function field of genus $g = 0$. Hence the geometric Goppa codes constructed by means of a rational function field $\mathbb{F}_q(x)$ are always MDS codes.

In order to obtain a meaningful bound for the minimum distance of $\mathcal{C}_{\mathcal{L}}(D,G)$ by Theorem 3.2.5, we often assume that $\deg G < n$.

Definition 3.2.8. *The integer $\delta = n - \deg G$ is called the* **designed distance** *of the geometric Goppa code $\mathcal{C}_{\mathcal{L}}(D,G)$.*

Theorem 3.2.5 states that the minimum distance δ of a geometric Goppa code cannot be less than its designed distance. The question whether $\delta = d$ or $\delta < d$ is answered by the following remark.

Remark 3.2.9. Suppose that $\dim G > 0$ and $\delta = n - \deg G > 0$. Then $\delta = d$ if and only if there exists a divisor D' with $0 \leq D' \leq D$, $\deg D' = \deg G$ and $\dim(G - D) > 0$.

Yet another code can be associated with the divisors D and G. However, before we give that construction we need two more concepts from the theory of algebraic function fields. The next two definitions hold for any function field F/K.

Definition 3.2.10. *For $A \in \mathcal{D}_F$,*

$$i(A) = \dim A - \deg A + g - 1$$

is called the **index of speciality** *of A.*

In Section 2.4 we considered the diagonal embedding $F \hookrightarrow \mathcal{A}_F$ which maps $x \in F$ to the corresponding principal adele. Now we introduce, for any place $P \in \mathbb{P}_F$, another embedding $\iota_P : F \hookrightarrow \mathcal{A}_F$.

Definition 3.2.11. *Let $P \in \mathbb{P}_F$.*

(1) *For $x \in F$ let $\iota_P(x) \in \mathcal{A}_F$ be the adele whose P-component is x, and all other components are 0.*

(2) *For a Weil differential $\omega \in \Omega_F$ define its* **local component** *$\omega_P : F \to K$ by*

$$\omega_P(x) = \omega(\iota_P(x)).$$

Clearly, ω_P is a K-linear mapping.

Using local components of Weil differentials we make the following definition.

Definition 3.2.12. *Let G and $D = P_1 + \cdots + P_n$ be divisors as before. Then we define the code $C_\Omega(D, G) \subseteq \mathbb{F}_q^n$ by*

$$C_\Omega(D, G) = \{(\omega_{P_1}(1), \ldots, \omega_{P_n}(1)) | \, \omega \in \Omega_F(G - D)\}.$$

Our first result is analogous to Theorem 3.2.5.

Theorem 3.2.13. *$C_\Omega(D, G)$ is an $[n, k', d']$-code with parameters*

$$k' = i(G - D) - i(G) \quad \textit{and} \quad d' \geq \deg G - (2g - 2).$$

Under the additional hypothesis $\deg G > 2g-2$*, we have* $k' = i(G-D) \geq n+g-1-\deg G$*. If, moreover,* $2g-2 < \deg G < n$ *then*

$$k' = n+g-1-\deg G.$$

Actually, Goppa introduce the codes $C_\Omega(D,G)$ rather than $\mathcal{C}_\mathcal{L}(D,G)$. However, there is a very close relation between them.

Theorem 3.2.14. *The codes* $\mathcal{C}_\mathcal{L}(D,G)$ *and* $C_\Omega(D,G)$ *are dual to each other, i.e.*

$$C_\Omega(D,G) = \mathcal{C}_\mathcal{L}(D,G)^\perp.$$

Our next aim is to prove that $C_\Omega(D,G)$ can be represented as $\mathcal{C}_\mathcal{L}(D,H)$ with an appropriate divisor H. For this purpose, we need the following lemma.

Lemma 3.2.15. *There exists a Weil differential* η *such that*

$$v_{P_i}(\eta) = -1 \quad \textit{and} \quad \eta_{P_i}(1) = 1 \textit{ for } i = 1, \ldots, n.$$

Proposition 3.2.16. *Let* η *be a Weil differential*[‡] *such that* $v_{P_i}(\eta) = -1$ *and* $\eta_{P_i}(1) = 1$ *for* $i = 1, \ldots, n$*. Then*

$$\mathcal{C}_\mathcal{L}(D,G)^\perp = C_\Omega(D,G) = \mathcal{C}_\mathcal{L}(D,H) \textit{ with } H = D - G + (\eta).$$

Corollary 3.2.17. *Suppose there is a Weil differential* η *such that*

$$(\eta) = 2G - D \quad \textit{and} \quad \eta_{P_i}(1) = 1 \textit{ for } i = 1, \ldots, n.$$

Then the code $\mathcal{C}_\mathcal{L}(D,G)$ *is self-dual.*

3.3 Automorphisms of Geometric Goppa Codes

In Section 1.5 we defined what we mean by the automorphism group of a code, cf. Definition 1.5.6. In this section we study automorphisms of geometric Goppa codes that are induced by the automorphisms of the corresponding function field.

Let $F/\mathbb{F}_q$ be a function field and $Aut(F/\mathbb{F}_q)$ be the group of automorphisms of F over $\mathbb{F}_q$ (i.e. $\sigma(a) = a$ for $\sigma \in Aut(F/\mathbb{F}_q)$ and $a \in \mathbb{F}_q$). The group $Aut(F/\mathbb{F}_q)$ acts on $\mathbb{P}_F$ by setting

[‡]Proposition VII.1.2 in [Sti91, p.205] provides an easy way of finding such Weil differentials. However, it requires the concept of derivatives which we have not introduced and therefore we can only refer the interested reader to Stichtenoth's book.

$\sigma(P) = \{\sigma(x) | \, x \in P\}$. It is easy to see that the corresponding valuations v_P and $v_{\sigma(P)}$ are related as follows:

$$v_{\sigma(P)}(y) = v_P(\sigma^{-1}(y)) \text{ for all } y \in F. \tag{3.3.1}$$

Moreover, $\deg \sigma(P) = \deg P$ since σ induces an isomorphism of the residue class fields of P and $\sigma(P)$ given by $\sigma(z(P)) = \sigma(z)(\sigma(P))$. The action of $Aut(F/\mathbb{F}_q)$ on $\mathbb{P}_F$ extends to an action on the divisor group by setting

$$\sigma\left(\sum n_P P\right) = \sum n_P \sigma(P).$$

As before we consider divisors $D = P_1 + \ldots + P_n$ and G of $F/\mathbb{F}_q$ where $P_1, \ldots, P_n$ are distinct places of degree one and $\operatorname{supp} G \cap \operatorname{sup} D = \emptyset$.

Definition 3.3.2.

$$Aut_{D,G}(F/\mathbb{F}_q) = \{\sigma \in Aut(F/\mathbb{F}_q) | \, \sigma(D) = D \text{ and } \sigma(G) = G\}.$$

Observe that an automorphism $\sigma \in Aut_{D,G}(F\mathbb{F}_q)$ need not fix all places $P_1, \ldots, P_n$ but it yields a permutation of $P_1, \ldots, P_n$. From (3.3.1) it follows that

$$\sigma(\mathcal{L}(G)) = \mathcal{L}(G) \tag{3.3.3}$$

for $\sigma \in Aut_{D,G}(F/\mathbb{F}_q)$, because $\sigma(G) = G$. The next result shows that each $\sigma \in Aut_{D,G}(F/\mathbb{F}_q)$ induces an automorphism of the corresponding code $\mathcal{C}_{\mathcal{L}}(D, G)$.

Proposition 3.3.4.

(a) *$Aut_{D,G}(F/\mathbb{F}_q)$ acts on the code $\mathcal{C}_{\mathcal{L}}(D, G)$ by*

$$\sigma((x(P_1), \ldots, x(P_n))) = (x(\sigma(P)), \ldots, x(\sigma(P)))$$

(for $x \in \mathcal{L}(G)$). This yields a homomorphism from $Aut_{D,G}(F/\mathbb{F}_q)$ to $Aut(\mathcal{C}_{\mathcal{L}}(D, G))$.

(b) *If $n > 2g + 2$, the above homomorphism is injective, hence $Aut_{D,G}(F/\mathbb{F}_q)$ can be regarded as a subgroup of $Aut(\mathcal{C}_{\mathcal{L}}(D, G))$.*

3.4 Rational Geometric Goppa Codes

In this section we give examples of geometric Goppa codes associated with divisors of a rational function field. We will rediscover some of the codes discussed in Chapter I.

Definition 3.4.1. *A geometric Goppa code $\mathcal{C}_{\mathcal{L}}(D,G)$ associated with divisors G and D of a rational function field $\mathbb{F}_q(z)/\mathbb{F}_q$ is said to be* **rational** *(as before, it is assumed that $D = P_1 + \ldots + P_n$ with pairwise distinct places of degree one, and $\operatorname{supp} G \cap \operatorname{supp} D = \emptyset$).*

Observe that the length of $\mathcal{C}_{\mathcal{L}}(D,G)$ is bounded by $q+1$ because $\mathbb{F}_q(z)$ has only $q+1$ places of degree 1: the pole P_∞ of z and, for each $\alpha \in \mathbb{F}_q$, the zero P_α of $z-\alpha$ (see Proposition 2.6.6). The following results follow immediately from Section 3.2.

Proposition 3.4.2. *Let $C = \mathcal{C}_{\mathcal{L}}(D,G)$ be a rational Goppa code over $\mathbb{F}_q$, and let n, k, d be the parameters of C. Then we have:*

(a) $n \leq q+1$.

(b) $k = 0$ *iff* $\deg G < 0$, *and* $k = n$ *iff* $\deg G > n-2$.

(c) *For* $0 \leq \deg G \leq n-2$,

$$k = 1 + \deg G \quad \textit{and} \quad d = n - \deg G.$$

In particular, C is an MDS-code.

(d) *$C^\perp$ is also a rational geometric Goppa code.*

The next result gives an explicit description of the generator matrices for rational codes.

Proposition 3.4.3. *Let $C = \mathcal{C}_{\mathcal{L}}(D,G)$ be a rational geometric Goppa code over $\mathbb{F}_q$ with parameters n, k and d.*

(a) *If $n \leq q$ there exist pairwise distinct elements $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ and $v_1, \ldots v_n \in \mathbb{F}_q \setminus \{0\}$ (not necessarily distinct) such that*

$$C = \{(v_1 \cdot f(\alpha_1), v_2 \cdot f(\alpha_2), \ldots, v_n \cdot f(\alpha_n)) \mid f \in \mathbb{F}_q[z] \textit{ and } \deg f \leq k-1\}.$$

The matrix

$$M = \begin{pmatrix} v_1 & v_2 & \ldots & v_n \\ \alpha_1 v_1 & \alpha_2 v_2 & \ldots & \alpha_n v_n \\ \alpha_1^2 v_1 & \alpha_2^2 v_2 & \ldots & \alpha_n^2 v_n \\ \vdots & \vdots & & \vdots \\ \alpha_1^{k-1} v_1 & \alpha_2^{k-1} v_2 & \ldots & \alpha_n^{k-1} v_n \end{pmatrix} \tag{3.4.4}$$

is a generator matrix for C.

(b) *If $n = q + 1$, C has a generator matrix*

$$M = \begin{pmatrix} v_1 & v_2 & \dots & v_{n-1} & 0 \\ \alpha_1 v_1 & \alpha_2 v_2 & \dots & \alpha_{n-1} v_{n-1} & 0 \\ \alpha_1^2 v_1 & \alpha_2^2 v_2 & \dots & \alpha_{n-1}^2 v_{n-1} & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ \alpha_1^{k-1} v_1 & \alpha_2^{k-1} v_2 & \dots & \alpha_{n-1}^{k-1} v_{n-1} & 1 \end{pmatrix} \tag{3.4.5}$$

where $\mathbb{F}_q = \{\alpha_1, \dots, \alpha_{n-1}\}$ and $v_1, \dots, v_{n-1} \in \mathbb{F}_q \setminus \{0\}$.

Definition 3.4.6. *Let $\alpha = (\alpha_1, \dots, \alpha_n)$ where the α_i are distinct elements of $\mathbb{F}_q$, and let $v = (v_1, \dots, v_n)$ where the v_i are non-zero (not necessarily distinct) elements of $\mathbb{F}_q$. Then the* **Generalised Reed Solomon** *code, denoted by $GRS_k(\alpha, v)$, consists of all vectors*

$$(v_1 \cdot f(\alpha_1), \dots, v_n \cdot f(\alpha_n))$$

with $f(z) \in \mathbb{F}_q[z]$ and $\deg f \leq k - 1$ (for a fixed $k \leq n$).

In the case $\alpha = (\beta, \beta^2, \dots, \beta^n)$ (where $n = q - 1$ and β a primitive n-th root of unity) and $v = (1, 1, \dots, 1)$, $\mathrm{GRS}_k(\alpha, v)$ is a Reed Solomon code, cf. Remark 1.8.7. Obviously, $\mathrm{GRS}_k(\alpha, v)$ is an $[n, k]$-code, and Proposition 3.4.3(a) states that all rational geometric Goppa codes (over $\mathbb{F}_q$) of length $n \leq q$ are Generalised Reed Solomon codes. The converse is also true:

Proposition 3.4.7. *Every Generalised Reed Solomon code $GRS_k(\alpha, v)$ can be represented as a rational geometric Goppa code.*

In order to determine the dual of a rational code $C = \mathcal{C}_{\mathcal{L}}(D, G)$, we need (by Theorem 3.2.14 and Proposition 3.2.15) a Weil differential ω of $\mathbb{F}_q(z)$ such that

$$v_{P_i}(\omega) = -1 \quad \text{and} \quad \omega_{P_i}(1) = 1 \text{ for } i = 1, \dots, n. \tag{3.4.8}$$

Lemma 3.4.9. *Consider the rational function field $F = \mathbb{F}_q(z)$ and suppose we have n distinct elements $\alpha_1, \dots, \alpha_n \in \mathbb{F}_q$. Let $P_i \in \mathbb{P}_F$ be the zero of $z - \alpha_i$ and $h(z) = \prod_{i=1}^{n}(z - \alpha_i)$. Suppose y is an element of F such that $y(P_i) = 1$ for $i = 1, \dots, n$. Then there exists a Weil differential ω of $F/\mathbb{F}_q$ with the property* (3.4.8) *and the divisor*

$$(\omega) = (y) + (h'(z)) - (h(z)) - 2P_\infty$$

(where $h'(z) \in \mathbb{F}_q[z]$ is the derivative of the polynomial $h(z)$).

Next we would like to describe BCH codes and classical Goppa codes by means of rational geometric Goppa codes. To this end, we need the following concept:

Definition 3.4.10. *Consider an extension field $\mathbb{F}_{q^m}$ of $\mathbb{F}_q$ and a code C over $\mathbb{F}_{q^m}$ of length n. Then*

$$C|_{\mathbb{F}_q} = C \cap \mathbb{F}_q^n$$

is called the **subfield subcode** *of C (or the restriction of C to $\mathbb{F}_q$).*

$C|_{\mathbb{F}_q}$ is a code over $\mathbb{F}_q$. Its minimum distance cannot be less than the minimum distance of C, and for the dimension of $C|_{\mathbb{F}_q}$, we have the trivial estimate $\dim C|_{\mathbb{F}_q} \leq \dim C$. In general, this is a strict inequality.

Definition 3.4.11. *Let $n \mid q^m - 1$ and $\beta \in \mathbb{F}_{q^m}$ be a primitive n-th root of unity. Let $l \in \mathbb{Z}$ and $\delta \geq 2$. Define a code $C(n, l, \delta)$ over $\mathbb{F}_{q^m}$ by the generator matrix*

$$H = \begin{pmatrix} 1 & \beta^l & \beta^{2l} & \dots & \beta^{(n-1)l} \\ 1 & \beta^{l+1} & \beta^{2(l+1)} & \dots & \beta^{(n-1)(l+1)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & \beta^{l+\delta-2} & \beta^{2(l+\delta-2)} & \dots & \beta^{(n-1)(l+\delta-2)} \end{pmatrix}. \tag{3.4.12}$$

The code $C = C(n, l, \delta)^{\perp}|_{\mathbb{F}_q}$ is obviously a BCH code with designed distance δ as described in Section 1.8.4.

Proposition 3.4.13. *Let $n \mid q^m - 1$ and $\beta \in \mathbb{F}_{q^m}$ be a primitive n-th root of unity. Let $F = \mathbb{F}_{q^m}(z)$ be the rational function field over $\mathbb{F}_{q^m}$ and P_0 (resp. P_∞) be the zero (resp. pole) of z. For $i = 1, \dots, n$ denote by P_i the zero of $z - \beta^{i-1}$, and set $D_\beta = P_1 + \dots + P_n$. Suppose that $a, b \in \mathbb{Z}$ are integers such that $0 \leq a + b \leq n - 2$. Then we have*

(a) *$\mathcal{C}_{\mathcal{L}}(D_\beta, aP_0 + bP_\infty) = C(n, l, \delta)$ with $l = -a$ and $\delta = a + b + 2$ (where $C(n, l, \delta)$ is as in Definition 3.4.11).*

(b) *The dual of $\mathcal{C}_{\mathcal{L}}(D_\beta, aP_0 + bP_\infty)$ is given by*

$$\mathcal{C}_{\mathcal{L}}(D_\beta, aP_0 + bP_\infty)^{\perp} = \mathcal{C}_{\mathcal{L}}(D_\beta, rP_0 + sP_\infty)$$

with $r = -(a + 1)$ and $s = n - b - 1$. Hence the BCH code $C(n, l, \delta)^{\perp}|_{\mathbb{F}_q}$ is the restriction to $\mathbb{F}_q$ of the code $\mathcal{C}_{\mathcal{L}}(D_\beta, rP_0 + sP_\infty)$ with $r = l - 1$ and $s = n + 1 - \delta - l$.

(c) *The automorphism $\sigma \in Aut(F/\mathbb{F}_{q^m})$ given by $\sigma(z) = \beta^{-1}z$ leaves the places P_0 and P_∞ invariant, and we have*

$$\sigma(P_i) = P_{i+1} \ (i = 1, \dots, n-1) \quad \text{and} \quad \sigma(P_n) = P_1.$$

Hence, by Proposition 3.3.4, σ induces the following automorphism of the code $\mathcal{C}_{\mathcal{L}}(D_\beta, aP_0 + bP_\infty)$:

$$\sigma((c_1, \ldots, c_n)) = (c_2, \ldots, c_n, c_1)$$

showing that BCH codes are indeed cyclic codes.

In a similar way we obtain the classical Goppa codes.

Definition 3.4.14. *Let $L = \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}_{q_m}$ with $|L| = n$, and let $g(z) \in \mathbb{F}_{q^m}[z]$ be a polynomial of degree t such that $1 \leq t \leq n-1$ and $g(\alpha_i) \neq 0$ for all $\alpha_i \in L$. We define a code $C(L, g(z)) \subseteq (\mathbb{F}_{q^m})^n$ by the generator matrix*

$$H = \begin{pmatrix} g(\alpha_1)^{-1} & g(\alpha_2)^{-1} & \ldots & g(\alpha_n)^{-1} \\ \alpha_1 \cdot g(\alpha_1)^{-1} & \alpha_2 \cdot g(\alpha_2)^{-1} & \ldots & \alpha_n \cdot g(\alpha_n)^{-1} \\ \vdots & \vdots & & \vdots \\ \alpha_1^{t-1} \cdot g(\alpha_1)^{-1} & \alpha_2^{t-1} \cdot g(\alpha_2)^{-1} & \ldots & \alpha_n^{t-1} \cdot g(\alpha_n)^{-1} \end{pmatrix}. \tag{3.4.15}$$

Note that the matrix (3.4.15) is a special case of (3.4.4) (with $v_i = g(\alpha_i)^{-1}$), hence $C(L, g(z))$ and $C(L, g(z))^\perp$ are Generalised Reed Solomon codes. Now we give an explicit description of these codes as rational Goppa codes.

Proposition 3.4.16. *In addition to the notation of Definition 3.4.14, let P_i denote the zero of $z - \alpha_i$ (for $\alpha_i \in L$), P_∞ the pole of z and $D_L = P_1 + \ldots + P_n$. Let G_0 be the zero divisor of $g(z)$ (in the divisor group of the rational function field $F = \mathbb{F}_{q^m}(z)$). Then*

$$C(L, g(z)) = \mathcal{C}_{\mathcal{L}}(D_L, G_0 - P_\infty) = \mathcal{C}_{\mathcal{L}}(D_L, A - G_0)^\perp$$

and

$$\Gamma(L, g(z)) = \mathcal{C}_{\mathcal{L}}(D_L, G_0 - P_\infty)^\perp|_{\mathbb{F}_q} = \mathcal{C}_{\mathcal{L}}(D_L, A - G_0)|_{\mathbb{F}_q},$$

where the divisor A is determined as follows: Set

$$h(z) = \prod_{\alpha_i \in L} (z - \alpha_i) \quad \text{and} \quad A = (h'(z)) + (n-1)P_\infty.$$

The code $\Gamma(L, g(z)) = C(L, g(z))^\perp|_{\mathbb{F}_q}$ is the classical Goppa code with Goppa polynomial $g(z)$ as described in Section 1.9.

3.5 Hermitian Codes

In this section we discuss the properties of codes constructed by means of the Hermitian function field H (cf. Definition 2.6.22 and Lemma 2.6.23). First we recall some properties of the Hermitian

function field H. H is a function field over $\mathbb{F}_{q^2}$; it can be represented as

$$H = \mathbb{F}_{q^2}(x, y) \quad \text{with} \quad y^q + y = x^{q+1}.$$

The genus of H is $g = q(q-1)/2$, and H has $N = 1 + q^3$ places of degree one, namely

- the common pole Q_∞ of x and y, and
- for each pair $(\alpha, \beta) \in \mathbb{F}_{q^2} \times \mathbb{F}_{q^2}$ with $\beta^q + \beta = \alpha^{q+1}$, there is a unique place $P_{\alpha,\beta}$ of degree one such that $x(P_{\alpha,\beta}) = \alpha$ and $y(P_{\alpha,\beta}) = \beta$.

Observe that for all $\alpha \in \mathbb{F}_{q^2}$ there exist q distinct elements $\beta \in \mathbb{F}_{q^2}$ with $\beta^q + \beta = \alpha^{q+1}$, hence the number of places $P_{\alpha,\beta}$ is q^3.

Definition 3.5.1. *For $r \in \mathbb{Z}$ we define*

$$C_r = \mathcal{C}_{\mathcal{L}}(D, rQ_\infty), \tag{3.5.2}$$

where

$$D = \sum_{\beta^q + \beta = \alpha^{q+1}} P_{\alpha.\beta} \tag{3.5.3}$$

is the sum of all places of degree one (except Q_∞) of the Hermitian function field $H/\mathbb{F}_{q^2}$. The codes C_r are called **Hermitian** *codes.*

Hermitian codes are codes of length $n = q^3$ over the field $\mathbb{F}_{q^2}$. For $r \le s$, we have $C_r \subseteq C_s$. Obviously, if $r < 0$, $\mathcal{L}(rQ_\infty) = 0$ and therefore $C_r = 0$. For $r > q^3 + q^2 - q - 2 = q^3 + (2g - 2)$, Theorem 3.2.5 and Riemann-Roch yield

$$\begin{aligned} \dim C_r &= \dim(rQ_\infty) - \dim(rQ_\infty - D) \\ &= (r + 1 - g) - (r - q^3 + 1 - g) = q^3 = n. \end{aligned}$$

Hence $C_r = \mathbb{F}_{q^2}^n$ in this case, and it remains to study Hermitian codes with $0 \le r \le q^3 + q^2 - q - 2$.

Proposition 3.5.4. *Let C_r be an Hermitian code with $0 \le r \le q^3 + q^2 - q - 2$. Then*

(a) *The dual code of C_r is*

$$C_r^\perp = C_{q^3+q^2-q-2-r}.$$

Hence C_r is self-orthogonal if $2r \le q^3 + q^2 - q - 2$, and C_r is self-dual iff $r = (q^3 + q^2 - q - 2)/2$ (note that the latter can only occur when q is a power of 2).

(b) *The dimension of* C_r *is given by*

$$\dim C_r = \begin{cases} |I(r)| & \text{for } 0 \le r < q^3, \\ q^3 - |I(s)| & \text{for } q^3 \le r \le q^3 + q^2 - q - 2, \end{cases}$$

where $s = q^3 + q^2 - q - 2 - r$ *and* $I(r) = \{0 \le n \le r \,|\, \exists\ z \in H \text{ with } (z)_\infty = nQ_\infty\}$. *For* $q^2 - q - 2 < r < q^3$, *we have*

$$\dim C_r = r + 1 - q(q-1)/2.$$

(c) *The minimum distance* d *of* C_r *satisfies*

$$d \ge q^3 - r.$$

If $0 \le r < q^3$ *and both* r *and* $q^3 - r$ *are pole numbers*[†] *of* Q_∞, *then*

$$d = q^3 - r.$$

(d) *The automorphism group of* $Aut(C_r)$ *of the Hermitian code* C_r *contains a subgroup of order* $q^3(q^2-1)$.

[†]Let F/K be a function field. An integer $n \ge 0$ is called a **pole number** of a place $P \in \mathbb{P}_F$ iff there exists an element $x \in F$ with $(x)_\infty = nP$.

Chapter 4

Automorphisms Of AG Codes And A Map On $\mathcal{L}(G)$

4.1 Introduction

In the first three chapters we have set the scene for the heart of this thesis. The remaining part of it is concerned with finding the automorphism group of various Goppa codes. The starting point of this research was Stichtenoth's paper on the automorphism group of rational Goppa codes ([Sti90]). Because he covered the function fields of genus $g = 0$, the next natural step was to look at Goppa codes associated with function fields of higher genus. Xing ([Xin95a] and [Xin95b]) proved similar results for elliptic codes and Hermitian codes. However, his results are not as general as Stichtenoth's as they involve severe restrictions on the divisors D and G used to define the Goppa codes in question. This chapter starts off with a recapitulation of Stichtenoth's result in the case of rational function field and goes on to introduce some new ideas which will allow us to find the automorphism group of certain (hyper-)elliptic codes (see Chapter 5). We then develop these ideas further and find the automorphism group of codes associated with a special class of function fields which we call, for lack of a better name, "admissible"(see Chapter 6). Having obtained these results we then use them to prove and improve Xing's result in the elliptic function field case (see Chapter 7). The improved result may have been known to Xing as well. I dare venture to say that he chose to present a weaker result merely to make his paper more readable. Lastly, we look at the function field of the Klein Quartic and obtain similar results there (see Chapter 8).

We recall some notation from the previous chapters which will be used throughout this chapter.

- $\mathbb{F}_q$- finite field of $q = p^n$ elements, where p is a prime.

- $\overline{K}$- the algebraic closure of a field K.
- $F/\mathbb{F}_q$ - an algebraic function field of one variable over $\mathbb{F}_q$ such that $\mathbb{F}_q$ is algebraically closed in F.
- $(x)_0$, $(x)_\infty$ - the zero/pole divisor of $0 \neq x \in F$.
- (x) - the principal divisor of $0 \neq x \in F$, i.e. $(x) = (x)_0 - (x)_\infty$.
- v_P - the discrete valuation of F with respect to a place P of $F/\mathbb{F}_q$.
- g - the genus of $F/\mathbb{F}_q$.
- $\mathbb{P}_F$ - the set of all places of F.
- $\mathbb{P}_F^{(k)}$ - the set of places of degree k in F.
- $\mathcal{D}_F$ - the divisor group of F/K, i.e. the free abelian group generated by the places of F/K.
- For a place $P \in \mathbb{P}_F^{(1)}$ and $u \in F$ with $v_P(u) \geq 0$ we have
  $u(P)$ - the value of u at P, i.e. $u(P) := u + P \in \mathbb{F}_q$ because P has degree 1.
- If G is a divisor of the function field $F/\mathbb{F}_q$ then
  $\mathcal{L}(G) = \{x \in F \mid (x) \geq -G\} \cup \{0\}$ is a finite dimensional $\mathbb{F}_q$-vector space.
- $Aut(F/\mathbb{F}_q)$ - the set of $\mathbb{F}_q$-automorphisms of a function field F.
- $Aut_{D,G}(F/\mathbb{F}_q) = \{\sigma \in Aut(F/\mathbb{F}_q) \,|\, \sigma(D) = D \text{ and } \sigma(G) = G\}$.
- $ev_D : \mathcal{L}(G) \to \mathbb{F}_q^n$, such that $ev_D(f) = (f(P_1), f(P_2), \ldots, f(P_n))$ for all $f \in \mathcal{L}(G)$ (cf. Definition 3.2.3).

4.2 Stichtenoth's Result

The main result of his paper is that the automorphism group of a geometric Goppa code associated with a rational function field $F = \mathbb{F}_q(x)$ is isomorphic to a certain subgroup of $Aut(F/\mathbb{F}_q)$. It is well-known that $Aut(F/\mathbb{F}_q) = PGL(2, q)$, the projective linear group over $\mathbb{F}_q$. In his paper Stichtenoth studies the automorphism group of rational Goppa codes $\mathcal{C}_\mathcal{L}(D, G)$ which are defined by the divisors G and $D = P_1 + \ldots + P_n$, where $\operatorname{supp} G \cap \operatorname{supp} D = \emptyset$ and $P_i \in \mathbb{P}_F^{(1)}$ for $1 \leq i \leq n$.

Remark 4.2.1. Stichtenoth uses a slightly more general definition of $Aut_{D,G}(F/\mathbb{F}_q)$. It is given by

$$Aut'_{D,G}(F/\mathbb{F}_q) = \{\sigma \in Aut(F/\mathbb{F}_q) \,|\, \sigma(D) = D \text{ and } \sigma(G) \sim_D G\},$$

where $\sigma(G) \sim_D G$ iff there exists $u \in F$ s.t. $G = \sigma(G) + (u)$ and $u(P_i) = 1$ for $i = 1, \ldots, n$.

The next lemma shows that for certain divisors his definition and ours given in Definition 3.3.2 coincide.

Lemma 4.2.2. *Assume $G = G_0 - G_1$ such that $G_0 \geq 0$, $G_1 \geq 0$ and $\deg(G_0 + G_1) \leq n-1$. Then*

$$Aut'_{D,G}(F/\mathbb{F}_q) = Aut_{D,G}(F/\mathbb{F}_q)$$

Proof. [Sti90, p.116, Lemma 2.2] □

The main result of his paper is the following:

Theorem 4.2.3. *Let $\mathcal{C}_{\mathcal{L}}(D,G)$ be the rational Goppa code with $1 \leq \deg G \leq n-3$. Then we have $Aut(\mathcal{C}_{\mathcal{L}}(D,G)) = Aut_{D,G}(F/\mathbb{F}_q)$, i.e. every automorphism of $\mathcal{C}_{\mathcal{L}}(D,G)$ is induced by a projective linear transformation.*

The proof depends essentially on two lemmas:

Lemma 4.2.4. *Let $\mathcal{C}_{\mathcal{L}}(D,G)$ and $\mathcal{C}_{\mathcal{L}}(D,G')$ be rational Goppa codes of length $n \geq 3$. Suppose that $0 \leq \deg G = \deg G' \leq n-2$. Then $\mathcal{C}_{\mathcal{L}}(D,G) = \mathcal{C}_{\mathcal{L}}(D,G')$ iff $G \sim_D G'$.*

Proof. see [Sti90, pp.117–120]. □

Lemma 4.2.5. *Let $\mathcal{C}_{\mathcal{L}}(D,G) = \mathcal{C}_{\mathcal{L}}(D',G')$ be a rational Goppa code of length n with $1 \leq \deg G = \deg G' \leq n-3$. Then there is an automorphism $\sigma \in Aut(F/\mathbb{F}_q)$ such that $\sigma(D) = D'$.*

Proof. see [Sti90, pp.117–120]. □

Assuming these two lemmas we can now prove the theorem.

Proof. We have to show that every automorphism of $\mathcal{C}_{\mathcal{L}}(D,G)$ is induced by an element of $Aut_{D,G}(F/\mathbb{F}_q)$. So we consider a permutation $\pi \in S_n$ such that

$$\pi(\mathcal{C}_{\mathcal{L}}(D,G)) = \mathcal{C}_{\mathcal{L}}(D,G).$$

Since $\pi(\mathcal{C}_{\mathcal{L}}(D,G)) = \mathcal{C}_{\mathcal{L}}(\pi(D),G)$ we know from Lemma 4.2.5 that there exists $\sigma \in Aut(F/\mathbb{F}_q)$ with $\sigma(P_i) = P_{\pi(i)}$ for $i = 1, \ldots, n$. In particular we have $\sigma(D) = D$. On the other hand,

$$\begin{aligned} \mathcal{C}_{\mathcal{L}}(\sigma(D),\sigma(G)) &= \mathcal{C}_{\mathcal{L}}(D,G) \qquad \dagger \\ &= \mathcal{C}_{\mathcal{L}}(\pi(D),G) \\ &= \mathcal{C}_{\mathcal{L}}(\sigma(D),G). \end{aligned}$$

By Lemma 4.2.4 we conclude that $\sigma(G) \sim_D G$. This proves that $\sigma \in Aut'_{D,G}(F/\mathbb{F}_q)$. □

†This equality is a trivial consequence of the way automorphisms of $F/\mathbb{F}_q$ act on $\mathcal{C}_{\mathcal{L}}(D,G)$. For a proof see Lemma 4.3.2.

Obviously, it would be quite nice to find generalisations of Lemma 4.2.4 and Lemma 4.2.5 which hold for function fields of higher genus.

This has been achieved for Lemma 4.2.4. Using some ingenious methods C. Munuera and R. Pellikaan proved the following result.

Lemma. *Suppose $n > 2g + 2$. Let G and G' be two divisors of the same degree m in a function field of genus g. If $\mathcal{C}_{\mathcal{L}}(D, G)$ is not equal to 0 nor to $\mathbb{F}_q^n$ and $2g - 1 < m < n - 1$, then $\mathcal{C}_{\mathcal{L}}(D, G) = \mathcal{C}_{\mathcal{L}}(D, G')$ if and only if $G \sim_D G'$.*

Proof. see [MP93, p. 245] □

However, so far there has been no generalisation of Lemma 4.2.5 whose proof relies on the fact that $PGL(2, q)$ acts 3-transitively on the places of degree 1 of the rational function field. It is clear from that that if there is going to be a generalisation its proof will have to use very different methods because, in general, the automorphism group of a function field of higher genus does not act 3-transitively on its places of degree 1.

The main objective of this thesis is to provide some corroborative evidence in the form of theorems and examples to support the view that Stichtenoth's result in the case of rational function fields might have an analogue in function fields of higher genus, i.e. that under certain conditions all automorphisms of a geometric Goppa code $\mathcal{C}_{\mathcal{L}}(D, G)$ come from automorphisms of the underlying function field.

4.3 The Ingredients Needed....

This section will introduce all the necessary results which will be used later to determine the automorphism group of a big class of geometric Goppa codes.

We start off with an elementary result regarding field extensions.

Theorem 4.3.1. *Consider the following situation:*

$$\begin{array}{c} F = K(x, y) \\ \Big| \; H(x, y) = 0 \\ K(x) \\ \Big| \\ K \end{array}$$

where K is a field, x is transcendental over K, and $H(x, y) = 0$ is the minimal polynomial of y over $K(x)$. Suppose $\widetilde{x}$, $\widetilde{y} \in F \setminus K$ (i.e. $\widetilde{x}$, $\widetilde{y}$ are both transcendental) and $E(\widetilde{x}, \widetilde{y}) = 0$. Then the

following map defines a K-endomorphism of F:

$$\lambda : F/K \longrightarrow F/K$$
$$f(x,y) \longmapsto f(\widetilde{x},\widetilde{y})$$

Furthermore, if x and y are in the image of λ, then λ is as an automorphism of F/K.

Proof. Because $\widetilde{x}$ is transcendental we have the following isomorphisms:

$$\lambda_1 : K[x] \longrightarrow K[\widetilde{x}]$$
$$f(x) \longmapsto f(\widetilde{x})$$

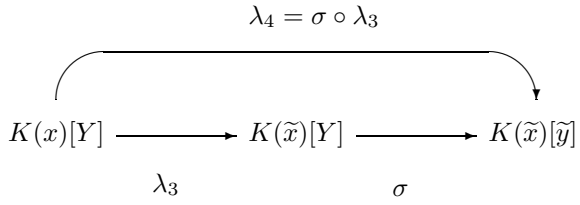
This isomorphism naturally extends to the field of fractions, i.e.

$$\lambda_2 : K(x) \longrightarrow K(\widetilde{x})$$
$$f(x) \longmapsto f(\widetilde{x})$$

is an isomorphism. This map in turn induces a ring-isomorphism:

$$\lambda_3 : K(x)[Y] \longrightarrow K(\widetilde{x})[Y]$$
$$f(x,Y) \longmapsto f(\widetilde{x},Y)$$

Finally we have the following ring-homomorphism:



where σ simply evaluates a rational function in $\widetilde{x}$ and Y at $\widetilde{y}$.
Now $ker(\lambda_4) = (H(x,Y))$ and hence we have

$$F = K(x)[Y]/\big(H(x,Y)\big) \cong K(\widetilde{x})[\widetilde{y}] = K(\widetilde{x})(\widetilde{y}) \subseteq F$$

Therefore the map λ induced by λ_4 is the desired field endomorphism. The last assertion of the theorem is obvious. □

As usual we assume that

- $F/\mathbb{F}_q$ is a function field of genus g,

- $D = P_1 + \ldots + P_n$, where the P_i are distinct places of degree one ($1 \leq i \leq n$), and
- G a divisor of $F/\mathbb{F}_q$ such that $\operatorname{supp} G \cap \operatorname{supp} D = \emptyset$.

The following three lemmas establish some basic facts concerning the action of $Aut(F/\mathbb{F}_q)$ on codes $\mathcal{C}_{\mathcal{L}}(D, G)$ and the spaces $\mathcal{L}(G)$ for certain divisors G.

Lemma 4.3.2. *For all $\sigma \in Aut(F/\mathbb{F}_q)$ we have*

$$\mathcal{C}_{\mathcal{L}}(D, G) = \mathcal{C}_{\mathcal{L}}(\sigma(D), \sigma(G)).$$

Proof. Let $\sigma \in Aut(F/\mathbb{F}_q)$. For a place P of degree 1 we have the following fact:

$$u(P) = \sigma(u)\big(\sigma(P)\big).$$

Because $\deg P = 1$, $u(P) =: a \in \mathbb{F}_q$, i.e. $u - a \in P$. Thus $\sigma(u - a) \in \sigma(P)$ which means that

$$\sigma(u)\big(\sigma(P)\big) = \sigma(a) = a = u(P).$$

We obviously have that $\sigma(\mathcal{L}(G)) = \mathcal{L}(\sigma(G))$. Furthermore, by the above fact, we know that for all $c \in \mathcal{C}_{\mathcal{L}}(D, G)$, there exists $u \in \mathcal{L}(G)$ such that

$$c = (u(P_1), u(P_2), \ldots, u(P_n)) = (\sigma(u)\big(\sigma(P_1)\big), \sigma(u)\big(\sigma(P_2)\big), \ldots, \sigma(u)\big(\sigma(P_n)\big))$$

Therefore, $\mathcal{C}_{\mathcal{L}}(D, G) = \mathcal{C}_{\mathcal{L}}(\sigma(D), \sigma(G))$. □

The next two lemmas are of crucial importance in the later sections when we determine the automorphism group of certain Goppa codes.

Lemma 4.3.3. *Let $\sigma \in Aut(F/\mathbb{F}_q)$. Let $G = kP > 0$ be a divisor of F with $\dim G > 1$. If $\sigma(\mathcal{L}(G)) = \mathcal{L}(G)$ then $\sigma(G) = G$.*

Proof. Because $\dim G > 1$, $\mathcal{L}(G) \setminus \mathbb{F}_q \neq \emptyset$, i.e. there exists $f \in \mathcal{L}(G)$ such that f has a pole at P and no other poles. Now $\mathcal{L}(G) = \sigma(\mathcal{L}(G)) = \mathcal{L}(\sigma(G))$. Furthermore, $\sigma(G) = k\sigma(P)$ and thus $\sigma(P) = P$ (for otherwise f would have a pole at $\sigma(P)$ as well). □

The next lemma is very similar to the previous one.

Lemma 4.3.4. *Let $\sigma \in Aut(F/\mathbb{F}_q)$ and suppose $G = \sum k_P P > 0$ is a divisor of F. Set $l := \max\{deg P \mid P \in \operatorname{supp} G\}$ and assume that $\deg G \geq 2g$. If $\sigma(\mathcal{L}(G)) = \mathcal{L}(G)$ then $\sigma(G) = G$.*

Proof. Again we note that $\mathcal{L}(G) = \sigma(\mathcal{L}(G)) = \mathcal{L}(\sigma(G))$. Suppose $\sigma(G) \neq G$ then there exists $P \in \operatorname{supp} G$ such that $v_P(G) \neq v_P(\sigma(G))$. By symmetry we can assume that $v_P(G) > v_P(\sigma(G))$.

Now for all $P \in \operatorname{supp} G$, we have $\deg(G-P) \geq \deg G - l \geq 0$. Therefore, by the Riemann-Roch Theorem and Clifford's Theorem, we see that

$$\dim(G-P) < \dim G.$$

Hence there exists $f \in \mathcal{L}(G) \setminus \mathcal{L}(G-P)$ with $-v_P(f) = v_P(G)$. Now $f \in \mathcal{L}(\sigma(G))$ as well, i.e $-v_P(f) \leq v_P(\sigma(G))$. But $-v_P(f) = v_P(G) > v_P(\sigma(G))$, a contradiction. Whence $\sigma(G) = G$. □

The next result is almost trivial but will prove useful later. It simply says that if two functions agree at enough places then they must be equal.

Lemma 4.3.5. *Let $u, v \in \mathcal{L}(G)$ for some divisor $G = G_0 - G_1$ where $G_0, G_1 \geq 0$. Set $r := \deg(G_0)$. Let $P_1, P_2, \ldots, P_n$ be n distinct places of degree 1 which are not in the support of G_0. If $u(P_i) = v(P_i)$ for $1 \leq i \leq n$ and $n > r$ then $u = v$.*

Proof. Because $\mathcal{L}(G)$ is a vector space $u - v \in \mathcal{L}(G)$. Hence $\deg((u-v)_\infty) \leq r$. But $u - v$ has at least $n > r$ zeros (because $(u-v)(P_i) = 0$ and hence $v_{P_i}(u-v) \geq 1$ for $1 \leq i \leq n$). Thus $u - v = 0$. □

We will now introduce the crucial ingredient for our results. In Chapter 3 we have seen that, if $\deg D > 2g+2$, $Aut_{D,G}(F/\mathbb{F}_q)$ can be embedded into $Aut(\mathcal{C}_\mathcal{L}(D,G))$ (cf. Proposition 3.3.4). We shall investigate under which conditions this embedding is actually surjective.

The idea is to associate to each automorphism π of the code $\mathcal{C}_\mathcal{L}(D,G)$ a linear automorphism, denoted by λ_π (or just λ if no confusion is likely to arise), of the associated function space $\mathcal{L}(G)$. We then prove that (under certain conditions) λ_π is the restriction to $\mathcal{L}(G)$ of an automorphism $\widetilde{\lambda} \in Aut(F/\mathbb{F}_q)$, in the sense that $\widetilde{\lambda}|_{\mathcal{L}(G)}$ is linear and agrees with λ_π on $\mathcal{L}(G)$ (as a $\mathbb{F}_q$-linear map). Finally, we show that $\widetilde{\lambda}$ is actually in $Aut_{D,G}(F/\mathbb{F}_q)$ and induces π via the embedding of $Aut_{D,G}(F/\mathbb{F}_q)$ into $Aut(\mathcal{C}_\mathcal{L}(D,G))$.

We make the following definition:

Definition 4.3.6. *Let $D = P_1 + \ldots + P_n$ and $D' = P_1' + \ldots + P_n'$ where $P_i, P_i' \in \mathbb{P}_F^{(1)}$. Suppose that G is any divisor with support disjoint from D and D' such that $\deg(G) < n$. If $\mathcal{C}_\mathcal{L}(D,G) = \mathcal{C}_\mathcal{L}(D',G)$ then we can define an $\mathbb{F}_q$-linear map $\lambda : \mathcal{L}(G) \to \mathcal{L}(G)$ by $\lambda = ev_{D'}^{-1} \circ ev_D$.*

Note that this definition makes sense because ev_D and $ev_{D'}$ are both $\mathbb{F}_q$-linear and bijective on $\mathcal{C}_\mathcal{L}(D,G)$ by Lemma 3.2.3. Pictorially, λ can be visualised as follows:

$$\begin{array}{ccc} \mathcal{C}_\mathcal{L}(D,G) & \xrightarrow{id} & \mathcal{C}_\mathcal{L}(D',G) \\ {\scriptstyle ev_D}\uparrow & & \downarrow{\scriptstyle ev_{D'}^{-1}} \\ \mathcal{L}(G) & \xrightarrow[\lambda = ev_{D'}^{-1} \circ ev_D]{} & \mathcal{L}(G) \end{array}$$

For all $f \in \mathcal{L}(G)$ we can characterise $\lambda(f) \in \mathcal{L}(G)$ by $\lambda(f)(P_i') = f(P_i)$ for $1 \leq i \leq n$.
We now give some easy consequences of the above definition. The next four lemmas will be used later.

Lemma 4.3.7. *With the notation of Definition 4.3.6.*
If $1 \in \mathcal{L}(G)$ *then* $\lambda(1) = 1$.

Proof. Since $1(P) = 1$ for all $P \in \mathbb{P}_F^{(1)}$ we know that $1(P_i') = 1(P_i) = 1 = \lambda(1)(P_i')$ for $1 \leq i \leq n$, i.e. $ev_{D'}(\lambda(1)) = ev_{D'}(1)$. By assumption $ev_{D'}$ is injective and hence it follows that $\lambda(1) = 1$. □

The following two lemmas will be used over and over again in the next chapters. They show that in the situation described in Definition 4.3.6 (with certain constraints on the degree of D and G) λ is also multiplicative, in a restricted sense, as well as linear.

Lemma 4.3.8. *With the notation of Definition 4.3.6.*
Suppose that $G > 0$ *and assume that* $n > 2\deg(G)$. *If* $f, g \in \mathcal{L}(G)$ *are such that* $fg \in \mathcal{L}(G)$ *also, then* $\lambda(fg) = \lambda(f)\lambda(g)$.

Proof. Obviously $\lambda(f), \lambda(g)$ and $\lambda(fg) \in \mathcal{L}(G)$; therefore, *a fortiori*, they are also in $\mathcal{L}(2G)$. Furthermore $\lambda(f)\lambda(g) \in \mathcal{L}(2G)$. Now $\lambda(f)\lambda(g)$ and $\lambda(fg)$ agree at all places of D of which there are at least $n > 2\deg(G)$. Thus by Lemma 4.3.5 we have $\lambda(fg) = \lambda(f)\lambda(g)$. □

Lemma 4.3.9. *With the notation of Definition 4.3.6.*
Suppose that $G > 0$ *and assume that* $n > 2\deg(G)$. *If* $f, f^k \in \mathcal{L}(G)$ *(*$k \geq 2$*), then* $\lambda(f^k) = \lambda(f)^k$. *Furthermore,* $\deg\left((\lambda(f))_\infty\right) \leq \frac{\deg G}{k}$.

Proof. By Lemma 4.3.8 we know that $\lambda(f^2) = \lambda(f)^2$. This implies that $\lambda(f)^2 \in \mathcal{L}(G)$. Thus by an obvious induction argument we find that

$$\lambda(f^k) = \lambda(f)^k.$$

The last statement follows from $\lambda(f)^k \in \mathcal{L}(G)$ since

$$\begin{aligned}\deg(G) &\geq \deg\left(\left(\lambda(f)^k\right)_\infty\right) \\ &= k\deg\left(\left(\lambda(f)\right)_\infty\right).\end{aligned}$$

□

The condition that $\deg((\lambda(f))_\infty) \leq \frac{\deg G}{k}$ will be used later in the construction of an automorphism of a function field.

The following lemma links λ to an automorphism of the function field. It will be used later to show that the map λ associated to a permutation $\pi \in Aut(\mathcal{C}_{\mathcal{L}}(D,G))$ can be used to construct an automorphism, $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$, of the function field which in turn induces the permutation π we started off with.

Lemma 4.3.10. *Let $F = \mathbb{F}_q(x,y)$ be a function field and $\sigma \in Aut(F/\mathbb{F}_q)$. Suppose we have that $\mathcal{C}_{\mathcal{L}}(D,G) = \mathcal{C}_{\mathcal{L}}(D',G)$ where $D = \sum_{i=1}^{n} P_i$, $D' = \sum_{i=1}^{n} P_i'$, and the support of G is disjoint from the support of D and D'. Let λ be the map described in Definition 4.3.6. Assume also that $x, y \in \mathcal{L}(G)$. If $\sigma|_{\mathcal{L}(G)} = \lambda$ and $\sigma(G) = G$ then $\sigma(P_i) = P_i'$ for $1 \leq i \leq n$.*

Proof. We have $\mathcal{C}_{\mathcal{L}}(D',G) = \mathcal{C}_{\mathcal{L}}(D,G) = \mathcal{C}_{\mathcal{L}}(\sigma(D),\sigma(G)) = \mathcal{C}_{\mathcal{L}}(\sigma(D),G)$ and $\sigma(f) = \lambda(f)$ for all $f \in \mathcal{L}(G)$. This implies that $\sigma(f)(\sigma(P_i)) = \lambda(f)(P_i')$ for all $f \in \mathcal{L}(G)$ and $1 \leq i \leq n$.

Now $F = \mathbb{F}_q(x,y)$ and if $P \in \mathbb{P}_F^{(1)}$ then the values $x(P)$ and $y(P)$ uniquely determine P. As $x, y \in \mathcal{L}(G)$ we find that

$$x(\sigma(P_i)) = x(P_i') \quad \text{and} \quad y(\sigma(P_i)) = y(P_i')$$

and hence $\sigma(P_i) = P_i'$ for $1 \leq i \leq n$. □

Every $\pi \in Aut(\mathcal{C}_{\mathcal{L}}(D,G))$ has a corresponding λ_π-map because $\mathcal{C}_{\mathcal{L}}(D,G) = \mathcal{C}_{\mathcal{L}}(\pi(D),G)$ and thus

$$\begin{aligned} \lambda_\pi : \mathcal{L}(G) &\longrightarrow \mathcal{L}(G) \\ f &\longmapsto \lambda_\pi(f) \\ \text{where } \lambda_\pi &= ev_{\pi(D)}^{-1} \circ ev_D \end{aligned}$$

i.e. $\lambda_\pi(f)\big(P_{\pi(i)}\big) = f\big(P_i\big)$ for $1 \leq i \leq n$.

The next two results are of theoretical interest and will not be used later. They just illustrate some further properties of the map λ. The following lemma shows that the composition of two automorphisms, π_1 and π_2 say, of the code corresponds to the composition of the associated maps λ_{π_1} and λ_{π_2}.

Lemma 4.3.11. *If π_1,$\pi_2 \in Aut(\mathcal{C}_{\mathcal{L}}(D,G))$ then $\lambda_{\pi_1} \circ \lambda_{\pi_2} = \lambda_{\pi_1 \circ \pi_2}$.*

Proof. Let $f \in \mathcal{L}(G)$. By the definition of λ we have

$$f(P_i) = \lambda_{\pi_1\pi_2}(f)(P_{\pi_1 \circ \pi_2(i)}) \tag{4.3.12}$$

where $1 \leq i \leq n = \deg(D)$.
Similarly, we have

$$f(P_i) = \lambda_{\pi_2}(f)\big(P_{\pi_2(i)}\big) \tag{4.3.13}$$

Now

$$\lambda_{\pi_2}(f)(P_j) = \lambda_{\pi_1}(\lambda_{\pi_2}(f))\big(P_{\pi_1(j)}\big) \tag{4.3.14}$$

where $1 \leq j \leq n$.

Setting $j = \pi_2(i)$ in (4.3.14) we get

$$\lambda_{\pi_2}(f)\big(P_{\pi_2(i)}\big) = \lambda_{\pi_1}(\lambda_{\pi_2}(f))\big(P_{\pi_1(\pi_2(i))}\big) \tag{4.3.15}$$

where $1 \leq i \leq n$.

Combining (4.3.12),(4.3.13) and (4.3.15) gives

$$\lambda_{\pi_1\pi_2}(f)\big(P_{\pi_1\circ\pi_2(i)}\big) = f(P_i) = \lambda_{\pi_1}(\lambda_{\pi_2}(f))\big(P_{\pi_1(\pi_2(i))}\big)$$

for $1 \leq i \leq n$.

Because of the injectivity of λ it follows that $\lambda_{\pi_1\pi_2}(f) = \lambda_{\pi_1}(\lambda_{\pi_2}(f))$ for all $f \in \mathcal{L}(G)$. □

Lemma 4.3.16. *With the notation of Definition 4.3.6.*
Suppose that $G = G_0 - G_1$, where $G_0, G_1 \geq 0$. Set $m := \deg(G_0)$ and assume that $n > m$.
If $u \in \mathcal{L}(G)$ is such that $(u)_0 = P_{i_1} + P_{i_2} + \ldots + P_{i_m}$, where $1 \leq i_1 < i_2 < \ldots < i_m \leq n$ then $(\lambda(u))_0 = P'_{i_1} + P'_{i_2} + \ldots + P'_{i_m}$.

Proof. Because $\lambda(u)(P'_{i_k}) = 0$ for $1 \leq k \leq m$ we know that $(\lambda(u))_0 \geq P'_{i_1} + P'_{i_2} + \ldots + P'_{i_m}$ but $\deg((\lambda(u))_\infty) \leq m = \deg(G_0)$. Thus $(\lambda(u))_0 = P'_{i_1} + P'_{i_2} + \ldots + P'_{i_m}$. □

The last lemma just says that if a function $u \in \mathcal{L}(G)$ has exactly $m = \deg G_0$ distinct simple zeros at places in the support of D then its image under λ will also have exactly m simple zeros, at the corresponding places of degree one in the support of D'.

Chapter 5

Automorphisms Of Elliptic And Hyperelliptic Codes

5.1 Introduction

In this section we shall describe how one can use the map λ introduced in Chapter 4 to determine the automorphism group of a large class of codes associated with (hyper-)elliptic function fields.

5.2 Elliptic and Hyperelliptic Codes

First we remind the reader of some results about (hyper-)elliptic function fields. We have to differentiate between (hyper-)elliptic function fields in even characteristic and (hyper-)elliptic function fields in odd characteristic (cf. Section 2.6.2).

- Let $F/\mathbb{F}_q$ be a (hyper-)elliptic function field. Then there exist x, y such that $F = \mathbb{F}_q(x, y)$ where the defining equation of F is given by (2.6.12) in $\operatorname{char} F = 2$ (resp. by (2.6.14) in $\operatorname{char} F \neq 2$), i.e.

$$\begin{aligned} y^2 + h(x)y = f(x) \qquad & \text{if char} = 2, \\ y^2 = f(x) \qquad & \text{if char} \neq 2. \end{aligned}$$

- In characteristic 2 the places of $P \in \mathbb{P}_{\mathbb{F}_q(x)}$ which are ramified in F are

$$\begin{aligned} &\text{all zeros of } h(x) \text{ if } \deg(h) = g + 1, \\ &\text{all zeros of } h(x) \text{ and the pole of } x \text{ if } \deg(h) \leq g. \end{aligned}$$

- In characteristic $\neq 2$ the places of $P \in \mathbb{P}_{\mathbb{F}_q(x)}$ which are ramified in F are

$$\text{all zeros of } f(x) \text{ if } \deg f(x) \equiv 0 \pmod 2,$$
$$\text{all zeros of } f(x) \text{ and the pole of } x \text{ if } \deg f(x) \equiv 1 \pmod 2.$$

With the notations as above we make the following trivial observations. Denoting the place (or places) above the pole of x by Q_∞ (or $Q_{\infty,1}, Q_{\infty,2}$ in the split case), we have:

(1) $(x)_\infty = 2Q_\infty$, and $(y)_\infty = (2g+1)Q_\infty$, if the pole of x is ramified in $F/\mathbb{F}_q(x)$.

(2) $(x)_\infty = Q_{\infty,1} + Q_{\infty,2}$, and $(y)_\infty \leq (g+1)(Q_{\infty,1} + Q_{\infty,2})$, if the pole of x splits in $F/\mathbb{F}_q(x)$;

(3) $(x)_\infty = Q_\infty$, $(y)_\infty \leq (g+1)Q_\infty$ if the pole of x stays inert in $F/\mathbb{F}_q(x)$.

We will use the following notation from now on:

$$D_\infty = \begin{cases} Q_\infty & \text{if the pole of } x \text{ ramifies, i.e. } \deg D_\infty = 1 \\ Q_{\infty,1} + Q_{\infty,2} & \text{if the pole of } x \text{ splits, i.e. } \deg D_\infty = 2 \\ Q_\infty & \text{if the pole of } x \text{ stays inert, i.e. } \deg D_\infty = 2. \end{cases}$$

5.2.1 Fixing the notation for the rest of this section

- $F/\mathbb{F}_q$ - a (hyper-)elliptic function field of genus g over $\mathbb{F}_q$ defined by an equation $H(x,y) = 0$ given by (2.6.12) or (2.6.14).
- $H(\mathbb{F}_q) = \mathbb{P}_F^{(1)} \cup \operatorname{supp}(D_\infty)$, where $\mathbb{P}_F^{(1)}$ is the set of all places of degree 1 in F.
- For all $P \in \mathbb{P}_F^{(1)}$ define $x_P := x(P)$ and $y_P := y(P)$, and identify $P = (x_P, y_P)$. Note that P is uniquely determined by the values x_P and y_P.
- For all $P \in \mathbb{P}_F^{(1)}$ denote by $\overline{P}$ the conjugate of P under the non-trivial automorphism ϕ of $F/\mathbb{F}_q(x)$, where

$$\phi(x) = x \quad \text{and} \quad \phi(y) = \begin{cases} -y & \text{if } \operatorname{char} F \neq 2, \\ y + h(x) & \text{if } \operatorname{char} F = 2. \end{cases}$$

Hence

$$\overline{P} = \begin{cases} (x_P, -y_P) & \text{if } \operatorname{char} F \neq 2, \\ (x_P, y_P + h(x_P)) & \text{if } \operatorname{char} F = 2. \end{cases}$$

- $H_r(\mathbb{F}_q)$ - the set of places $P \in H(\mathbb{F}_q) \setminus \mathrm{supp}(D_\infty)$ ramified in $F/\mathbb{F}_q(x)$. If $P \in H_r(\mathbb{F}_q)$ then

$$(x - x_P) = \begin{cases} 2P - 2D_\infty & \text{if the pole of } x \text{ ramifies in } F/\mathbb{F}_q, \\ 2P - D_\infty & \text{otherwise.} \end{cases}$$

  Note also that if $P \in H_r(F/\mathbb{F}_q)$ is a ramified place then $y_P = 0$.

- $H_s(\mathbb{F}_q)$ - the set of places $P \in H(\mathbb{F}_q) \setminus \mathrm{supp}(D_\infty)$ split in $F/\mathbb{F}_q(x)$. If $P \in H_s(\mathbb{F}_q)$ then

$$(x - x_P) = \begin{cases} P + \overline{P} - 2D_\infty & \text{if the pole of } x \text{ ramifies,} \\ P + \overline{P} - D_\infty & \text{otherwise.} \end{cases}$$

5.2.2 The automorphism group of elliptic and hyperelliptic codes

The following lemma and its corollaries will be used later in Proposition 5.2.6 to construct a basis for the space $\mathcal{L}(G)$ for a certain type of divisor G.

Lemma 5.2.1. *Let $P = (x_P, y_P) \in H_s(\mathbb{F}_q)$. Suppose $z \in F$ is such that $v_P(z) \geq 0$. Then for all $k \geq 0$ there exist unique $a_i \in \mathbb{F}_q$ $(0 \leq i \leq k)$ such that*

$$v_P(z - \sum_{i=0}^{k} a_i (x - x_P)^i) \geq k + 1.$$

Remark 5.2.2. Note that because $P \in H_s(\mathbb{F}_q)$, $v_P(x - x_P) = 1$ and hence $\sum_{i=0}^{k} a_i (x - x_P)^i$ is the beginning of the power series expansion of z in terms of the local uniformizing parameter $(x - x_P)$. In the completion of the valuation ring at P one could write $z = \sum_{i=0}^{\infty} a_i (x - x_P)^i$ (cf. [Sti91, pp. 143–145]).

Proof. We define the a_i inductively. Because $v_P(z) \geq 0$ we can set $a_0 = z(P) \in \mathbb{F}_q$ and hence $v_P(z - a_0) \geq 1$. Now suppose that

$$v_P\left(z - \sum_{i=0}^{k-1} a_i (x - x_P)^i\right) \geq k.$$

Because $v_P(x - x_P) = 1$ we have

$$v_P\left(\frac{z - \sum_{i=0}^{k-1} a_i (x - x_P)^i}{(x - x_P)^k}\right) \geq 0. \tag{5.2.3}$$

So there exists a unique $a_k = ((z - \sum_{i=0}^{k-1} (x - x_P)^i)/((x - x_P)^k))(P)$ (note that this makes sense because of (5.2.3)) such that

$$v_P\left(\frac{z - \sum_{i=0}^{k-1} a_i (x - x_P)^i}{(x - x_P)^k} - a_k\right) \geq 1.$$

Thus

$$v_P\left(\frac{z-\sum_{i=0}^{k} a_i(x-x_P)^i}{(x-x_P)^k}\right) \geq 1,$$

and hence

$$v_P\left(z-\sum_{i=0}^{k} a_i(x-x_P)^i\right) \geq k+1.$$

□

Corollary 5.2.4. *Let $P=(x_P,y_P)\in H_s(\mathbb{F}_q)$; then for $1\leq i\leq g+1$, there exist functions f_{iP} such that $f_{iP}=y-h_i(x)$ where $\deg h_i(x)\leq(i-1)$, $v_P(f_{iP})\geq i$ and $v_{\overline{P}}(f_{iP})=0$.*
In other words, we can find a function that only involves y and x to the power of at most $i-1$ which has at least i zeros at P and none at $\overline{P}$.

Proof. Note that $v_P(y)=0$ and apply Lemma 5.2.1. This yields functions

$$f_{iP}=y-y_P-(\sum_{j=1}^{k-1} a_i(x-x_P)^i) \quad \text{with} \quad v_P(f_{iP})\geq i,$$

where $1\leq i\leq g+1$.

Furthermore, by way of construction, $v_{\overline{P}}(f_{iP})=0$, since $f_{iP}(\overline{P})=y_{\overline{P}}-y_P\neq 0$. □

The above method has the advantage that it is easy to implement on a computer. Without loss of generality we may assume that $x_P=0$ (otherwise we do a linear shift of coordinates). By Remark 5.2.2 we see that what we are looking for is an "approximation" of y in terms of polynomials $h_i(x)\in\mathbb{F}_q[x]$, where $\deg h_i(x)\leq i$ for $0\leq i\leq g+1$ and $v_P(y-h_i(x))\geq i+1$. This means that

$$\begin{cases} h_i(x)^2+h(x)h_i(x) & \text{in char } F=2,\\ h_i^2(x) & \text{in char } F\neq 2 \end{cases}$$

has to agree with $f(x)$ in all their lower degree terms (up to degree i), where

$$\begin{cases} y^2+h(x)y=f(x) & \text{in char } F=2,\\ y^2=f(x) & \text{in char } F\neq 2 \end{cases}$$

are the defining equations of the (hyper-)elliptic function field.

We look at the case $\operatorname{char} F=2$ and leave the $\operatorname{char} F\neq 2$ to the reader.
The defining equation of the (hyper-)elliptic function field is given by (2.6.12), $y^2+h(x)y=f(x)$. Suppose $P=(0,y_P)\in H_s(\mathbb{F}_q)$, $h(x)=c_0+c_1x+\ldots+c_{g+1}x^{g+1}$, and $f(x)=d_0+d_1x+\ldots+d_{2g+2}x^{2g+2}$, where $c_i,d_j\in\mathbb{F}_q$ $(0\leq i\leq g+1,\ 0\leq j\leq 2g+2)$ and g is the genus of F. Because

$P \in H_s(\mathbb{F}_q)$ we also note that $c_0 \neq 0$. We are going to define $h_i(x) \in \mathbb{F}_q[x]$ such that $\deg h_i(x) \leq i$ and $v_p(y - h_i(x)) \geq i + 1$ $(0 \leq i \leq g + 1)$.
$i = 1$: Put $h_1(x) = e_0$. By the above reasoning we have to choose e_0 such that

$$e_0^2 + c_0 e_0 = d_0.$$

Because $P = (0, y_P)$ this means that $e_0 = y_P$.
$i = 2$: Put $h_2(x) = y_P + e_1 x$. Then we have

$$h_1(x)^2 + h(x)h_1(x) = y_P^2 + c_0 y_P + (c_0 e_1 + c_1 y_P)x + \text{higher degree terms},$$

and we need

$$(c_0 e_1 + c_1 y_P) = d_1.$$

Because $c_0 \neq 0$ we can solve for e_1.

So suppose we have computed $h_{i-1}(x)$. Set $h_i(x) = h_{i-1}(x) + e_i x^i$. We then have

$$h_i(x)^2 + h(x)h_{i-1}(x) + h(x)e_i x^i = d_0 + d_1 x + \ldots + d_{i-1}x^i + (c_0 e_i + r)x^i + \text{higher degree terms},$$

where r is a sum of products of c_j and e_k, where $1 \leq j \leq i$ and $0 \leq k \leq i - 1$, i.e. it is a known quantity by the inductive hypothesis. Hence we can solve

$$(c_0 e_i + r) = d_i$$

for e_i. This completes the induction.

Corollary 5.2.5. *With the notation of Corollary 5.2.4. Let $P = (x_P, y_P) \in H_s(\mathbb{F}_q)$. For $1 \leq i \leq g + 1$ set $g_{i\overline{P}} := \dfrac{f_{iP}}{(x - x_P)^i}$; then*

$$v_{\overline{P}}(g_{i\overline{P}}) = -i \quad \textit{and} \quad v_\infty(g_{i\overline{P}}) \geq \begin{cases} -(2g + 1 - 2i) & \textit{if the pole of } x \textit{ ramifies,} \\ -(g + 1 - i) & \textit{otherwise.} \end{cases}$$

Here $v_\infty = v_{Q_\infty}$ if the pole of x is either ramified or inert in $F/\mathbb{F}_q$. If the pole of x is split then v_∞ represents either valuation $v_{Q_{\infty,1}}$ or $v_{Q_{\infty,2}}$.

Proof. Obviously we have

$$v_P(f_{iP}) \geq i \text{ and } v_{\overline{P}}(f_{iP}) = 0 \quad \text{by Corollary 5.2.4;}$$

$$v_\infty(f_{iP}) \geq \begin{cases} -(2g + 1) & \text{if the pole of } x \text{ is ramified,} \\ -(g + 1) & \text{otherwise;} \end{cases}$$

$$\left((x - x_P)^i\right) = \begin{cases} i(P + \overline{P}) - 2iD_\infty & \text{if the pole of } x \text{ is ramified,} \\ i(P + \overline{P}) - iD_\infty & \text{otherwise;} \end{cases}$$

and hence the result follows.

Note that in both cases when $i = g+1$ we actually have

$$\left(g_{(g+1)\overline{P}}\right)_\infty = \left(\frac{f_{(g+1)P}}{(x-x_P)^{g+1}}\right)_\infty = (g+1)\overline{P}.$$

□

Proposition 5.2.6. *Let J be a (possibly empty) subset of $\mathbb{P}_F^{(1)} \setminus \mathrm{supp}(D_\infty)$. Suppose that G is a divisor of the form:*

$$G = n_\infty D_\infty + \sum_{Q\in J} n_Q Q$$

where $n_Q \geq 1$ for all $Q \in J$ and $n_\infty \geq \begin{cases} 2g+2 & \text{if the pole of } x \text{ ramifies} \\ g+1 & \text{otherwise} \end{cases}$.

Setting $m := g+1$, a basis of $\mathcal{L}(G)$ is given by the following elements

$$\begin{aligned}\mathcal{L}(G) =& \langle x^\alpha y^\beta |\ \alpha \geq 0, \beta \in \{0,1\} \text{ and } x^\alpha y^\beta \in \mathcal{L}(n_\infty D_\infty),\\ & \left(\frac{1}{x-x_Q}\right)^\alpha \left(\frac{y}{x-x_Q}\right)^\beta |\ 2\alpha+\beta \leq n_Q, \alpha \geq 0, \beta \in \{0,1\}, Q \in J \cap H_r(\mathbb{F}_q),\\ & \left(g_{mQ}\right)^\alpha \left(g_{\beta Q}\right) |\ m\alpha+\beta \leq n_Q, \alpha \geq 0, 0 \leq \beta < m, Q \in J \cap H_s(\mathbb{F}_q)\rangle.\end{aligned}$$

Proof. Case 1 - The pole of x is ramified:

First of all, we check that all the functions are actually in $\mathcal{L}(G)$. This is obvious for the functions of the form $x^i y^j$. Note that there are

$$\dim(\mathcal{L}(n_\infty D_\infty)) = n_\infty + 1 - g$$

such functions (by Riemann-Roch and $n_\infty \geq 2g+2$).

Now looking at $Q \in J \cap H_r(\mathbb{F}_q)$ we note that

$$\begin{aligned}\left(\frac{y}{x-x_Q}\right)_\infty =& (2g-1)D_\infty + Q \text{ and}\\ \left(\frac{1}{x-x_Q}\right)_\infty =& 2Q.\end{aligned}$$

For $1 \leq i \leq n_Q$ we can write $i = 2\alpha+\beta$ with $\alpha \geq 0$ and $\beta \in \{0,1\}$. Thus

$$v_Q\left(\left(\frac{1}{x-x_Q}\right)^\alpha \left(\frac{y}{x-x_Q}\right)^\beta\right) = \begin{cases} -(2\alpha+1) & \text{if } \beta = 1 \\ -2\alpha & \text{if } \beta = 0 \end{cases}$$

and

$$v_\infty\left(\left(\frac{1}{x-x_Q}\right)^\alpha \left(\frac{y}{x-x_Q}\right)^\beta\right) = \begin{cases} (2\alpha-(2g-1)) & \text{if } \beta = 1 \\ 2\alpha & \text{if } \beta = 0 \end{cases}$$

and as $n_\infty \geq 2g+2$ all n_Q of these functions are in $\mathcal{L}(G)$. Finally we look at $Q \in J \cap H_s(\mathbb{F}_q)$. For $1 \leq i \leq n_Q$, we can write $i = m\alpha + \beta$ with $\alpha \geq 0$ and $0 \leq \beta < m$. Thus by Corollary 5.2.5 and setting $g_{0Q} := 1$ we have

$$v_Q\Big(\big(g_{mQ}\big)^\alpha\big(g_{\beta Q}\big)\Big) = -(m\alpha+\beta)$$

and

$$v_\infty\Big(\big(g_{mQ}\big)^\alpha\big(g_{\beta Q}\big)\Big) \geq \begin{cases} -(2g+1-2\beta) & \text{if } \beta \neq 0 \\ 0 & \text{otherwise.} \end{cases}$$

We see that all n_Q of these functions are in $\mathcal{L}(G)$ because $n_\infty \geq 2g+2$. Since

$$\deg G = n_\infty + \sum_{Q \in J} n_Q \geq 2g+2,$$

it follows by Riemann-Roch that

$$\dim \mathcal{L}(G) = n_\infty + \sum_{Q \in J} n_Q + 1 - g.$$

Hence, we have exactly the right number of functions and it only remains to show that they are linearly independent. This follows immediately from the "Strict-Triangle-Inequality"-property of v_Q for all $Q \in J \cup \operatorname{supp} D_\infty$.

Case 2 - The pole of x is inert or split:

Again we check that all the functions are actually in $\mathcal{L}(G)$. This is obvious for the functions of the form $x^i y^j$. Note that there are

$$\dim(\mathcal{L}(n_\infty D_\infty)) = 2n_\infty + 1 - g$$

such functions (by Riemann-Roch and $n_\infty \geq g+1$).

Now looking at $Q \in J \cap H_r(\mathbb{F}_q)$ we note that

$$\begin{aligned} Q \leq \Big(\frac{y}{x-x_Q}\Big)_\infty &\leq gD_\infty + Q \quad \text{and} \\ \Big(\frac{1}{x-x_Q}\Big)_\infty &= 2Q. \end{aligned}$$

For $1 \leq i \leq n_Q$ we can write $i = 2\alpha + \beta$ with $\alpha \geq 0$ and $\beta \in \{0,1\}$. Thus

$$v_Q\Big(\Big(\frac{1}{x-x_Q}\Big)^\alpha\Big(\frac{y}{x-x_Q}\Big)^\beta\Big) = \begin{cases} -(2\alpha+1) & \text{if } \beta = 1 \\ -2\alpha & \text{if } \beta = 0 \end{cases}$$

and

$$v_\infty\Big(\Big(\frac{1}{x-x_Q}\Big)^\alpha\Big(\frac{y}{x-x_Q}\Big)^\beta\Big) \geq \begin{cases}\alpha - g & \text{if } \beta = 1\\ \alpha & \text{if } \beta = 0\end{cases}$$

and as $n_\infty \geq g+1$ all n_Q of these functions are in $\mathcal{L}(G)$. (Note that we adopt the same definition of v_∞ as given in Corollary 5.2.5, i.e. v_∞ represents either of the $v_{Q_\infty,i}$ $(i=1,2)$ in the split case and v_{Q_∞} in the inert case.)

Finally we look at $Q \in J \cap H_s(\mathbb{F}_q)$.
For $1 \leq i \leq n_Q$ and we can write $i = m\alpha + \beta$ with $\alpha \geq 0$ and $0 \leq \beta < m$. Thus by Corollary 5.2.5 and setting $g_{0Q} := 1$ we have

$$v_Q\Big(\big(g_{mQ}\big)^\alpha\big(g_{\beta Q}\big)\Big) = \begin{cases}-(m\alpha+\beta) & \text{if } \beta \neq 0\\ -m\alpha & \text{otherwise}\end{cases}$$

and

$$v_\infty\Big(\big(g_{mQ}\big)^\alpha\big(g_{\beta Q}\big)\Big) \geq \begin{cases}-(g+1-\beta) & \text{if } \beta \neq 0\\ 0 & \text{otherwise.}\end{cases}$$

We see that because $n_\infty \geq g+1$ all n_Q of these functions are in $\mathcal{L}(G)$. Since

$$\deg G = 2n_\infty + \sum_{Q\in J} n_Q \geq 2g+2,$$

it follows by Riemann-Roch that

$$\dim \mathcal{L}(G) = 2n_\infty + \sum_{Q\in J} n_Q + 1 - g.$$

Hence, we have exactly the right number of functions and it only remains to show that they are linearly independent. This again follows immediately from the "Strict-Triangle-Inequality"-property of v_Q for all $Q \in J \cup \operatorname{supp} D_\infty$. □

Remark 5.2.7. The observant reader will have noticed that n_∞ did not need to be as big as it was stipulated in the lemma. However, we need the condition on n_∞ given in the previous lemma for the proof of the next theorem. The reason why we chose the value of n_∞ the way we did was to make sure that y and x^{g+1} are in $\mathcal{L}(G)$.

We are now in a position to formulate the main result of this section

Theorem 5.2.8. *Let the notation and G be as in Proposition 5.2.6. Let $I \subseteq H(\mathbb{F}_q) \setminus \operatorname{supp}(G)$. Set $D = \sum_{P\in I} P$, $n := \deg(D)$ and $t := \deg(G)$. If $n > \max\{2t, 2g+2\}$ then*

$$Aut(\mathcal{C}_\mathcal{L}(D,G)) \cong Aut_{D,G}(F/\mathbb{F}_q).$$

Proof. By assumption $n > 2g + 2$, hence by Proposition 3.3.4, $Aut_{D,G}(F/\mathbb{F}_q)$ can be regarded as a subgroup of $Aut(\mathcal{C}_{\mathcal{L}}(D,G))$. We are going to show that it is actually isomorphic to the whole group. So let $\pi \in Aut(\mathcal{C}_{\mathcal{L}}(D,G))$; then because $\mathcal{C}_{\mathcal{L}}(D,G)) = \mathcal{C}_{\mathcal{L}}(\pi(D),G)$ we can consider the corresponding map λ_π (see Definition 4.3.6). We are going to show that from λ_π we can construct an automorphism $\widetilde{\lambda_\pi} \in Aut(F/\mathbb{F}_q)$ which preserves $\mathcal{L}(G)$ and whose restriction to $\mathcal{L}(G)$ coincides with λ_π. We then prove that $\widetilde{\lambda_\pi}$ is actually in $Aut_{D,G}(F/\mathbb{F}_q)$. For simplicity we write λ instead of λ_π.

For $i = 0, 1$, define α_i to be such that $y^i x^{\alpha_i} \in \mathcal{L}(G)$ but $y^i x^{\alpha_i+1} \notin \mathcal{L}(G)$. Note that $\alpha_1 \leq \alpha_0$. By Lemma 4.3.9 we know that

$$\lambda(x^l) = \lambda(x)^l \text{ for } 1 \leq l \leq \alpha_0.$$

Thus from Lemma 4.3.8 it follows that

$$\lambda(yx^l) = \lambda(y)\lambda(x)^l \text{ for } 1 \leq l \leq \alpha_1.$$

Furthermore by Lemma 4.3.7 we have

$$\lambda(1) = 1.$$

Using the same method, i.e. making repeated use of Lemma 4.3.8 and Lemma 4.3.9 it is easy to see that

$$\lambda\left(\left(\frac{1}{x - x_Q}\right)^\alpha \left(\frac{y}{x - x_Q}\right)^\beta\right) = \lambda\left(\frac{1}{x - x_Q}\right)^\alpha \lambda\left(\frac{y}{x - x_Q}\right)^\beta$$

for $2\alpha + \beta \leq n_Q$, $\alpha \geq 0$, $\beta \in \{0, 1\}$, and $Q \in J \cap H_r(\mathbb{F}_q)$; and

$$\lambda\left((g_{mQ})^\alpha (g_{\beta Q})\right) = \lambda\left(g_{mQ}\right)^\alpha \lambda\left(g_{\beta Q}\right)$$

for $m\alpha + \beta \leq n_Q$, $\alpha \geq 0$, $0 \leq \beta < m$, and $Q \in J \cap H_s(\mathbb{F}_q)$.

A simple calculation establishes that

$$\lambda\left(\frac{1}{x - x_Q}\right) = \frac{1}{\lambda(x) - x_Q} \quad \text{and} \quad \lambda\left(\frac{y}{x - x_Q}\right) = \frac{\lambda(y)}{\lambda(x) - x_Q}.$$

Next we show that for $Q \in \text{supp}(G) \cap H_s(\mathbb{F}_q)$ and $1 \leq i \leq \min\{n_Q, g + 1\}$ we have

$$\begin{aligned}
\lambda(g_{iQ}) =& \lambda\left(\frac{f_{i\overline{Q}}}{(x - x_Q)^i}\right) \\
=& \lambda\left(\frac{y - p_i(x)}{(x - x_Q)^i}\right) \\
=& \frac{\lambda(y) - p_i(\lambda(x))}{(\lambda(x) - x_Q)^i}
\end{aligned}$$

where $\deg(p_i(x)) \leq i-1$. For $1 \leq i \leq \min\{n_Q, g+1\}$ it follows (by Lemma 4.3.9 and the linearity of λ) that:

$$\lambda\big((x-x_Q)^i\big) = (\lambda(x)-x_Q)^i \quad \text{and} \quad \lambda(y-p_i(x)) = \lambda(y) - p_i(\lambda(x))$$

(Note that the given conditions on n_∞ imply that $(x-x_Q)^i,\, y-p_i(x) \in \mathcal{L}(G)$).
Hence by Lemma 4.3.8 we find

$$\lambda\Big((x-x_Q)^i(g_{iQ})\Big) = (\lambda(x)-x_Q)^i\lambda\big(g_{iQ}\big) = \lambda\big(y-p_i(x)\big).$$

Thus

$$\lambda\big(g_{iQ}\big) = \frac{\lambda(y)-p_i(\lambda(x))}{(\lambda(x)-x_Q)^i}.$$

As every element $\tilde{h}(x,y) \in \mathcal{L}(G)$ can be written as a linear combination of the above elements, we have shown that $\lambda(\tilde{h}(x,y)) = \tilde{h}(\lambda(x),\lambda(y))$.

Finally, we show that λ determines the desired automorphism. Because of the conditions on n_∞, we see that y and x^{g+1} are both in $\mathcal{L}(G)$. Hence $\lambda(y), \lambda(x) \in \mathcal{L}(G)$ and also, by Lemma 4.3.9, $\deg\left((\lambda(x))_\infty\right) \leq t/(g+1)$. We will now show that $\lambda(y)$ and $\lambda(x)$ satisfy $H(\lambda(x),\lambda(y)) = 0$, the defining equation of the (hyper-)elliptic function field given by (2.6.12) or (2.6.14).

First we look at the case where $\operatorname{char} F = 2$:

In this case, the defining equation of the (hyper-)elliptic function field given by (2.6.12) is

$$H(\lambda(x),\lambda(y)) = \lambda(y)^2 + h(\lambda(x))\lambda(y) + f(\lambda(x)) = 0.$$

Note that

$$\deg(\lambda(y))_\infty \leq t \quad \text{and} \quad \deg\left(h(\lambda(x))\right)_\infty \leq (g+1)\deg(\lambda(x))_\infty = t.$$

Thus $\lambda(y)^2, h(\lambda(x))\lambda(y)$ are both in $\mathcal{L}(2G)$ and so is their sum. Whence

$$\deg\left(\lambda(y)^2 + h(\lambda(x))\lambda(y)\right)_\infty \leq 2t.$$

We also have that

$$\deg\left(f(\lambda(x))\right)_\infty \leq (2g+2)\deg(\lambda(x))_\infty \leq 2t.$$

Furthermore, because $y^2 + h(x)y = f(x)$ we observe that

$$(\lambda(y)^2 + h(\lambda(x))\lambda(y))(P) = f(\lambda(x))(P)$$

for all $P \in \operatorname{supp} D$, where $|\operatorname{supp} D| > 2t$.

Since $\lambda(y)^2 + h(\lambda(x))\lambda(y)$ and $f(\lambda(x))$ are both elements of $\mathcal{L}(2G)$, we can apply Lemma 4.3.5 to deduce that $H(\lambda(x), \lambda(y)) = 0$.

Now we look at the case when $\operatorname{char} F \neq 2$:

We have to show that $\lambda(y)$ and $\lambda(x)$ satisfy $H(\lambda(x), \lambda(y)) = \lambda(y)^2 - f(\lambda(x)) = 0$, the defining equation of the (hyper-)elliptic function field given by (2.6.14). We know that

$$\deg\left(\lambda(y)^2\right)_\infty < 2t \quad \text{and} \quad \deg\left(f(\lambda(x))\right)_\infty = 2(g+1)\deg\left(\lambda(x)\right)_\infty \leq 2t.$$

Furthermore, because $y^2 = f(x)$ we have that

$$\lambda(y)^2(P) = f(\lambda(x))(P) \quad \text{for all } P \in \operatorname{supp} D,$$

where $|\operatorname{supp} D| > 2t$. Now $\lambda(y)^2$ and $f(\lambda(x))$ are both elements of $\mathcal{L}(2G)$. Therefore, we can apply Lemma 4.3.5 to deduce that $H(\lambda(x), \lambda(y)) = 0$.

So in either case $H(\lambda(x), \lambda(y)) = 0$ and thus by Theorem 4.3.1, there exists an endomorphism, $\widetilde{\lambda}$ say, of the function field such that $\widetilde{\lambda}(x) = \lambda(x)$ and $\widetilde{\lambda}(y) = \lambda(y)$. Because of the way λ acts on $\mathcal{L}(G)$, it is clear that the restriction of $\widetilde{\lambda}$ to $\mathcal{L}(G)$ coincides with λ. Now because x and y are in the image of λ and hence of $\widetilde{\lambda}$, $\widetilde{\lambda}$ must actually be an automorphism.

The only thing that remains to be shown is that $\widetilde{\lambda}$ is actually in $Aut_{D,G}(F/\mathbb{F}_q)$, i.e. that $\widetilde{\lambda}(D) = D$ and $\widetilde{\lambda}(G) = G$. The latter follows from Lemma 4.3.4, thus Lemma 4.3.10 applies and the result now follows. □

5.3 The Elliptic Case $g = 1$ - An Example

In the elliptic function field case we can prove a slightly more general theorem by making use of the group structure which can be imposed on the places of degree 1 (cf. Section 2.6.2). Let this group be denoted by $H(\mathbb{F}_q)$ and let $\oplus$ denote the group operation. Note that in the elliptic case we may assume that $d = 3$ because over a finite field every function field of genus 1 has a place of degree 1, which we may take to be Q_∞. Whence we have $D_\infty = Q_\infty$.

Theorem 5.3.1. *Let $I, J \subseteq H(\mathbb{F}_q)$ such that $I \cap J = \emptyset$. Suppose further that the divisors D and G are given by*

$$D = \sum_{P \in I} P \text{ and } G = \sum_{Q \in J} n_Q Q, \text{ where } n_Q \geq 1 \text{ and } 4 \leq \max\{n_Q | \, Q \in J\}.$$

Let $n := \deg D$ and $t := \deg G$. If $n > 2t$ then

$$Aut(\mathcal{C}_\mathcal{L}(D, G)) \cong Aut_{D,G}(F/\mathbb{F}_q).$$

Proof. Choose $Q \in J$ such that $n_Q = \max\{n_{Q'} | \, Q' \in J\}$. Because of the group structure of $H(\mathbb{F}_q)$ we know that there exists $\sigma \in Aut(F/\mathbb{F}_q)$ such that $\sigma(P) = P \ominus Q$ for all $P \in H(\mathbb{F}_q)$. Lemma 4.3.2 yields that $\mathcal{C}_{\mathcal{L}}(D,G) = \mathcal{C}_{\mathcal{L}}(\sigma(D), \sigma(G))$ and we have

$$Aut_{D,G}(F/\mathbb{F}_q) = \sigma^{-1} Aut_{\sigma(D),\sigma(G)}(F/\mathbb{F}_q)\sigma.$$

Hence we can work with $\sigma(D)$ and $\sigma(G)$ instead. So without loss of generality we might as well assume that $D_\infty \in J$ and that $n_\infty = \max\{n_Q | Q \in J\}$. Note that $4 \leq \max\{n_Q | \, Q \in J\}$ implies that $t \geq 4$ and thus $n > 4 = 2g + 2$. Now Theorem 5.2.8 applies. □

The following are immediate corollaries of the above theorem.

Corollary 5.3.2. *Let $J \subseteq H(\mathbb{F}_q) \setminus \{D_\infty\}$ with $n := |J|$. If*

$$D = \sum_{P \in J} P \text{ and } G = mD_\infty \text{ where } 8 \leq 2m < n \text{ then } Aut(\mathcal{C}_{\mathcal{L}}(D,G)) \cong Aut_{D,G}(F/\mathbb{F}_q).$$

Proof. Immediate from Theorem 5.3.1. □

Remark 5.3.3. If we further stipulate that $n \geq 10$ then the result also holds for $m = 3$ (see Example 6.3.7 and Chapter 7).

Remark 5.3.4. This result is similar to Xing's [Xin95a] (cf. Section 7). On the one hand, it is more general than his because we allow D to consist of any collection of places $P \neq D_\infty$ of degree one, whereas he requires $D = \sum_{P \in H(\mathbb{F}_q) \setminus \{D_\infty\}} P$. However, because the proof of Corollary 5.3.2 does not make use of the underlying group structure of the elliptic curve we have a worse bound for m than he does (by a factor of 2 approximately). See Chapter 7 for a more detailed exposition.

To obtain a larger number of automorphisms of the code it is obvious that we need to choose the divisors D and G carefully, i.e. in such a way that $Aut_{G,D}(F/\mathbb{F}_q)$ contains many elements. The next corollary gives an example of this.

Corollary 5.3.5. *Let $J = H(\mathbb{F}_q) \setminus H_r(\mathbb{F}_q)$ and $I = H_r(\mathbb{F}_q)$. Put $D = \sum_{P \in J} P$ and $G = \sum_{Q \in I} Q$. Set $n := \deg D$ and $t := \deg G$. Choose $m \geq 4$. If $n > 2mt$ then*

$$Aut(\mathcal{C}_{\mathcal{L}}(D, mG)) \cong Aut_{D,mG}(F/\mathbb{F}_q).$$

Proof. Immediate from Theorem 5.3.1. □

Remark 5.3.6. We note that $|Aut_{D,mG}(F/\mathbb{F}_q)| = t|Aut_{D,mD_\infty}(F/\mathbb{F}_q)|$ where $t = 1, 2, 4$ is the number of ramified places of F. Hence the maximum number of automorphisms we can have in the above case is 48. (If $\operatorname{char} F = 3$, we can have up to 12 Q_∞-fixing automorphisms. If F has four ramified places, $Q_1, \ldots, Q_4$ say, then we also have the four automorphisms $\sigma_i(P) = P \oplus Q_i$, $1 \leq i \leq 4$.)

Example 5.3.7. Let $H(x,y) = y^2 - (x^3 - x) = 0$ define the elliptic function field $F/\mathbb{F}_{81}$. It is easy to establish that the function field defined by this elliptic curve has 64 places of degree 1. The j-invariant of the elliptic curve is 0. Therefore we know that there are 12 Q_∞-fixing automorphisms (cf. [Sil86, Proposition 1.2 (c) in Appendix A]). It is also obvious that $x^3 - x = x(x-1)(x-2)$ over $\mathbb{F}_{81}$ and thus $H(\mathbb{F}_q)$ has four ramified places. Hence for $4 \leq m \leq 7$ the codes described in Corollary 5.3.5 have an automorphism group of size 48. Note that in this case the code $\mathcal{C}_\mathcal{L}(D, 7G)$ is an $[60, 28, d]$-code, where $32 \leq d \leq 33$.

Remark 5.3.8. Obviously, we can choose the places of any subgroup $H' < H(\mathbb{F}_q)$ as the support of the divisor G in Corollary 5.3.5, letting D be the sum of the remaining places of degree one. Thus we have that the automorphisms of $F/\mathbb{F}_q$ fixing $G = m\sum_{P\in H'} P$ also fix D. Provided that $\deg D > 2m \deg G$, we find that $Aut(\mathcal{C}_\mathcal{L}(D, mG)) \cong Aut_{D,mG}(F/\mathbb{F}_q)$.

5.4 A Hyperelliptic Code in char 2

We shall give yet another example of how to apply the map λ to determine the automorphism group of a geometric Goppa code. This time we make use of an example of a self-dual code that Stichtenoth gave in his paper on self-dual codes [Sti88, pp. 206–207, Example 4.1.1].

We consider the hyperelliptic function field $F/\mathbb{F}_{q^2}$ defined by

$$y^2 + y = x^{q+1} \tag{5.4.1}$$

where $q = 2^h$. By Proposition 2.6.11 it follows that the genus of F is $g = q/2$. Furthermore, it is easy to check that in this case all places of $\mathbb{F}_{q^2}(x)$ of degree 1 split in $F/\mathbb{F}_{q^2}(x)$ except the pole, Q_∞, of x which ramifies in $F/\mathbb{F}_{q^2}(x)$. Hence F has exactly $1 + 2q^2$ places of degree one.

Theorem 5.4.2. *Let $F/\mathbb{F}_{q^2}$ be the hyperelliptic function field defined by* (5.4.1)*. Let*

$$D = (x^{q^2} + x)_0 = \sum_{P\in\mathbb{P}_F^{(1)}\setminus\{Q_\infty\}} P.$$

Suppose that $q + 2 \leq m \leq q^2 - 1$. Then

$$Aut_{D,mQ_\infty} \cong Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty)).$$

Proof. Note that $\deg D = 2q^2$ and $2g + 2 = q + 2$ and hence the result follows immediately from Theorem 5.2.8. □

In the course of his example 4.1.1 Stichtenoth proves the following:

Lemma 5.4.3. *Their exists a differential η which has the properties*

$$(\eta) = (2q^2 + q - 2)Q_\infty - D \quad \textit{and} \quad \eta_P(1) = 1$$

for all P in the support of D.

Proof. See [Sti88, p.207]. □

This leads to the following lemma:

Lemma 5.4.4. *With the notation of Theorem 5.4.2.*

$$\mathcal{C}_\mathcal{L}(D, mQ_\infty)^\perp = \mathcal{C}_\mathcal{L}(D, kQ_\infty)$$

where $k = 2q^2 + q - (m + 2)$.

Proof. By Proposition 3.2.16 we see that

$$\mathcal{C}_\mathcal{L}(D, mQ_\infty)^\perp = \mathcal{C}_\mathcal{L}(D, H) \quad \text{with } H = D - mQ_\infty + (\eta).$$

Hence $H = D - mQ_\infty + (2q^2 - 2 + q)Q_\infty - D = (2q^2 + q - (m + 2))Q_\infty$. □

For any linear code C we have $Aut(C^\perp) = Aut(C)$ and $C = (C^\perp)^\perp$ thus the above lemma now allows us to prove the following result.

Theorem 5.4.5. *Let $F/\mathbb{F}_{q^2}$ be the hyperelliptic function field defined by* (5.4.1)*. Let*

$$D = (x^{q^2} + x)_0 = \sum_{P \in \mathbb{P}_F^{(1)} \setminus \{Q_\infty\}} P.$$

Suppose that $q + 2 \leq m \leq q^2 - 1$ or $q^2 + q - 1 \leq m \leq 2q^2 - 4$. Then

$$Aut_{D,mQ_\infty} \cong Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty)).$$

Proof. Immediate from Theorem 5.4.2 and the previous lemma. □

Chapter 6

A Special Class Of Function Fields

6.1 Introduction

In this chapter we investigate a class of algebraic function fields which is more general than the (hyper-)elliptic fields considered in the previous chapter. For want of a better name, we will call this class of function fields admissible function fields. We will show that for a special class of codes associated with these function fields, similar results hold with respect to the group of code automorphisms.

6.2 Preliminaries

The following definition describes the function fields we are interested in.

Definition 6.2.1. *Let $F/\mathbb{F}_q$ be a function field such that*

- *F is not rational.*
- *there exists a place Q_∞ of degree 1 and two elements $x, y \in F$ such that $(x)_\infty = kQ_\infty$, $(y)_\infty = lQ_\infty$ and $k, l \geq 1$.*
- *For $m \geq 0$, the elements $x^i y^j$ with $0 \leq i$, $0 \leq j \leq k-1$ and $ki + lj \leq m$ form a basis of the space $\mathcal{L}(mQ_\infty)$.*

Because there does not seem to exist a specific name for this type of function fields in the literature, we shall call, for the purposes of this thesis, a function field F satisfying the above conditions **admissible**.

The following list of function fields satisfying the above conditions shows that our definition is not vacuous:

- (hyper-)elliptic function fields $F/\mathbb{F}_q$ given by $y^2 = f(x)$ (see (2.6.14)) or $y^2 + h(x)y = f(x)$ (see (2.6.12)), where the pole of x ramifies in $F/\mathbb{F}_q$ (here, $k = 2$ and $l = 2g + 1$).
- all function fields of the form $F = K(x, y)$ with

$$y^n + \mu y = f(x) \in K[x]$$

  where $n = p^s > 1$ is a power of p and $0 \neq \mu \in K$. Moreover, we assume that $\deg f =: d > 0$ is prime to p, and that all roots of $T^n + \mu T = 0$ are in K (here, $k = n$ and $l = d$), (cf. Proposition 2.6.20).
- Hermitian function fields (special case of the above).

From the above list of examples it can be seen that this class of function fields is fairly broad.

We are now going to prove some easy consequences of the above definition.

Lemma 6.2.2. *With the notation of Definition 6.2.1.*
If F is an admissible function field then $\min\{k, l\} \geq 2$ *and* $\gcd(k, l) = 1$.

Proof. If $\min\{k, l\} = 1$, say $(x)_\infty = Q_\infty$, then $[F : \mathbb{F}_q(x)] = \deg((x)_\infty) = 1$. Hence $F = \mathbb{F}_q(x)$ is rational, which is a contradiction to the stipulated assumptions, proving the first assertion. Let $g = g(F)$ be the genus of F. By the Riemann-Roch-Theorem we know that for $m \geq 2g - 1$ we have that

$$\dim(mQ_\infty) = \deg(mQ_\infty) + 1 - g = m + 1 - g.$$

Thus looking at

$$\mathcal{L}(mQ_\infty) \subset \mathcal{L}((m + 1)Q_\infty) \subset \mathcal{L}((m + 2)Q_\infty)$$

where $m \geq 2g - 1$, we see that the dimension goes up by 1 each time. Hence there are elements $x^{i_1}y^{j_1}$ and $x^{i_2}y^{j_2}$ with $0 \leq i_1, i_2$ and $0 \leq j_1, j_2 \leq k - 1$ such that $ki_1 + lj_1 = m + 1$ and $ki_2 + lj_2 = m + 2$. Now any common factor of k and l divides $m + 1$ and $m + 2$ and thus divides 1. Therefore $\gcd(k, l) = 1$. □

Lemma 6.2.3. *With the notation of Definition 6.2.1.*
If F is an admissible function field then $F = \mathbb{F}_q(x, y)$ and

$$y^k + g(x, y) = b_l x^l + b_{l-1}x^{l-1} + \ldots + b_0 \tag{6.2.4}$$

with $b_l \neq 0$ *and* $b_r \in \mathbb{F}_q$ *for* $0 \leq r \leq l$ *and*
$g(x,y) = a_{k-1}y^{k-1}h_{k-1}(x) + a_{k-2}y^{k-2}h_{k-2}(x) + \ldots + a_1 y h_1(x)$
where $a_r \in \mathbb{F}_q$, $h_r \in \mathbb{F}_q[x]$ *and* $k(\deg h_r(x)) + rl \leq lk$ *for* $1 \leq r \leq k-1$.

Proof. We know that $[F : \mathbb{F}_q(x)] = \deg\big((x)_\infty\big) = k$ and $[F : \mathbb{F}_q(y)] = \deg\big((y)_\infty\big) = l$. Now we also have the following inclusions

$$\mathbb{F}_q(x) \subset \mathbb{F}_q(x,y) \subset F$$

and

$$\mathbb{F}_q(y) \subset \mathbb{F}_q(x,y) \subset F.$$

By the multiplicity of dimensions we see that $[F : \mathbb{F}_q(x,y)]$ divides both k and l. Thus it must be 1 because $(k,l) = 1$ by Lemma 6.2.2.
Thus we immediately see that y^k can be written as an $\mathbb{F}_q[x]$-linear combination of $1, y, y^2, \ldots, y^{k-1}$. We observe that $y^k \in \mathcal{L}(klQ_\infty)$. Therefore

$$y^k = s_0(x) + s_1(x)y + \ldots s_{k-1}(x)y^{k-1} \tag{6.2.5}$$

where $s_0(x), \ldots s_{k-1}(x) \in \mathbb{F}_q[x]$.
Because both sides of (6.2.5) must have the same number of poles we know that

$$v_{Q_\infty}(y^k) = -kl = v_{Q_\infty}(s_0(x) + s_1(x)y + \ldots + s_{k-1}(x)y^{k-1}).$$

Since $\gcd(k,l) = 1$, it follows that $\deg s_0(x) = l$. Setting $g(x,y) := -(s_1(x)y + \ldots + s_{k-1}(x)y^{k-1})$ yields the desired form. □

6.3 Codes Associated with Admissible Function Fields

In this section we determine the automorphism group of a certain class of codes associated with admissible function fields. First we fix some notation:

- Let F be an admissible function field with parameters k, l.
- Let $J \subseteq \mathbb{P}_F^{(1)} \setminus \{Q_\infty\}$.
- Set $n := |J|$.
- Let $D = \sum_{P \in J} P = P_1 + P_2 + \ldots + P_n$
- Let $D' = \sum_{r=1}^{n} P_{\pi(r)}$ for some $\pi \in S_n$

The following lemma is a rather technical inequality but is needed in the proof of the next theorem.

Lemma 6.3.1. *If $\beta \geq \gamma \geq 2$ and $l \geq k \geq \beta + 1$, β,k,l $\in \mathbb{N}$ then*

$$k(l + \frac{k-1}{\beta}) \geq \gamma(l + \frac{k-1}{\gamma - 1}).$$

Proof. Because $\beta \geq \gamma \geq 2$ and $l \geq k$ it is clear that

$$(\beta - \gamma + 1)(\beta(\gamma - 1)l - k + 1) \geq 0.$$

Adding $\beta\gamma((\gamma - 1)l + k - 1)$ to both sides one gets

$$(\beta + 1)(\gamma - 1)(\beta l + k - 1) \geq \beta\gamma((\gamma - 1)l + k - 1)$$

and hence

$$k(\gamma - 1)(\beta l + k - 1) \geq \beta\gamma((\gamma - 1)l + k - 1).$$

Now dividing both sides by $\beta(\gamma - 1)$ yields the result. □

We are now in a position to exhibit some of the properties of the map λ in connection with codes associated with admissible function fields.

Theorem 6.3.2. *With the notation of Definition 6.2.1:*
Let $l > k$ and $m \geq l$ so that x and y are in $\mathcal{L}(mQ_\infty)$. Suppose that $\mathcal{C}_\mathcal{L}(D, mQ_\infty) = \mathcal{C}_\mathcal{L}(D', mQ_\infty)$. Define $\lambda : \mathcal{L}(mQ_\infty) \to \mathcal{L}(mQ_\infty)$ as described in Definition 4.3.6. If the following conditions are satisfied

(1) $-v_{Q_\infty}(\lambda(x)) \leq c$ *(where $c \leq m$ since $\lambda(x) \in \mathcal{L}(mQ_\infty)$);*

(2) $n > \max\{m + c, k(l + \frac{k-1}{\beta}), l \min\{c, k(1 + \frac{k-1}{m-k+1}\}\}$ *where $\beta = \min\{k - 1, r \mid y^r \in \mathcal{L}(mQ_\infty)\}$.*

then $v_{Q_\infty}(\lambda(x)) = -k$ and $v_{Q_\infty}(\lambda(y)) = -l$.
Moreover, λ is the restriction to $\mathcal{L}(mQ_\infty)$ of an automorphism of the underlying function field.

Remark 6.3.3. For $1 \leq i \leq \beta$ we define $\alpha_i := [\frac{m-il}{k}]$. Then we obviously have

$$\begin{aligned}\mathcal{L}(mQ_\infty) = \langle 1, x, x^2, \ldots, x^{\alpha_0},\\ y, xy, x^2y, \ldots, x^{\alpha_1}y,\\ \vdots\\ y^\beta, xy^\beta, \ldots, x^{\alpha_\beta}y^\beta\rangle.\end{aligned}$$

For convenience we write v for v_{Q_∞}.

Proof. The outline of the proof is the following:
As before, the main thrust of the proof is to show that λ acts multiplicatively on $\mathcal{L}(mQ_\infty)$, in the sense that if $x^iy^k \in \mathcal{L}(mQ_\infty)$ then $\lambda(x^iy^j) = \lambda(x)^i\lambda(y)^k \in \mathcal{L}(mQ_\infty)$. In the process of establishing this, we will be able to refine the estimates of the degrees of the pole divisors of $\lambda(x)$ and $\lambda(y)$. The only thing we know about them to start with is that they can be approximated by

$$k \leq \deg((\lambda(x))_\infty) \leq c \quad \text{and} \quad k \leq \deg((\lambda(y))_\infty) \leq m.$$

With the revised estimates and the conditions on n (the number of places in D), we will then proceed to show that $\lambda(x)$ and $\lambda(y)$ satisfy the defining equation (6.2.4) of the function field F and hence determine an endomorphism, $\widetilde{\lambda}$, of $F/\mathbb{F}_q$ which agrees with λ on $\mathcal{L}(mQ_\infty)$. Because x and y are in the image of λ and hence of $\widetilde{\lambda}$, it follows that $\widetilde{\lambda}$ is an automorphism of $F/\mathbb{F}_q$ proving that $v(\lambda(x)) = -k$ and $v(\lambda(y)) = -l$.

First we show that $\lambda(x^i) = \lambda(x)^i$ for $1 \leq i \leq \alpha_0$:
Let μ be such that $\mu c \leq m < (\mu+1)c \leq m+c$. We know that $\deg((\lambda(x))_\infty) \leq c$, hence $\deg((\lambda(x)^2)_\infty) \leq 2c \leq m+c$. $\lambda(x^2)$ and $\lambda(x)^2$ are both in $\mathcal{L}((m+c)Q_\infty)$ and they agree at $n > m+c$ places, so (by Lemma 4.3.5) $\lambda(x^2) = \lambda(x)^2$ which in turns shows that $\deg((\lambda(x)^2)_\infty) \leq 2c$. Now continuing in this way we find that

$$\lambda(x^\nu) = \lambda(x)^\nu$$

for $1 \leq \nu \leq \mu$. If $\mu = \alpha_0$, we deduce that $-v(\lambda(x)) \leq \min\{c, [\frac{m}{\alpha_0}]\}$.

Otherwise, we have that $\mu < \alpha_0$ and thus $x^{\mu+1} \in \mathcal{L}(mQ_\infty)$. Because $\deg(\lambda(x)^{\mu+1})_\infty \leq m+c$ we find that

$$\lambda(x^{\mu+1}) = \lambda(x)^{\mu+1}$$

and hence that $m \geq -v(\lambda(x^{\mu+1})) = -(\mu+1)v(\lambda(x))$. Hence $-v(\lambda(x)) \leq \frac{m}{\mu+1} < c$. Now by Lemma 4.3.5 (with $\mathcal{L}((m+c)Q_\infty)$) we see that

$$\lambda(x^{\mu+2}) = \lambda(x)^{\mu+1}\lambda(x)$$

because $-v(\lambda(x)^{\mu+2}) \leq (\mu+2)\frac{m}{\mu+1} = m + \frac{m}{\mu+1} < m+c < n$. As $-v(\lambda(x^{\mu+2})) \leq m$ we deduce (as before) that $-v(\lambda(x)) \leq \frac{m}{\mu+2}$. Continuing this process we find that $-v(\lambda(x)) \leq \frac{m}{\alpha_0}$.

So, in either case, we have that

$$k \leq -v(\lambda(x)) \leq \min\{c, [\frac{m}{\alpha_0}]\}. \tag{6.3.4}$$

Provided that $\alpha_r > 0$ $(1 \leq r \leq \beta)$ and using the same technique as above, we also find that

$$\lambda(y^r x) = \lambda(y^r)\lambda(x) \text{ for } 1 \leq r \leq \beta$$

(because $n > m + c$ and $\lambda(y^r x)$, $\lambda(y^r)\lambda(x) \in \mathcal{L}((m + c)Q_\infty)$, thus Lemma 4.3.5 applies). This in turn implies that $-v(\lambda(y^r)) \leq m + v(\lambda(x))$. Hence we can look at the element $\lambda(y^r x^2)$ and deduce that

$$\lambda(y^r x^2) = \lambda(y^r)\lambda(x)^2 \text{ for } 1 \leq r \leq \beta.$$

Continuing like this we find that

$$\lambda(y^r x^s) = \lambda(y^r)\lambda(x)^s \quad \text{for } 1 \leq r \leq \beta \text{ and } 0 \leq s \leq \alpha_r.$$

Because of (6.3.4) we know that

$$-v(\lambda(y^r)) \leq m - k\alpha_r \text{ for } 1 \leq r \leq \beta.$$

Thus $-v(\lambda(y)) \leq m - k\alpha_1 \leq l + k - 1$ as $\alpha_1 := [\frac{m-l}{k}] \geq \frac{m-l-k+1}{k}$. If $\beta = 1$ this is the best we can do. Otherwise we can assume that $\beta \geq 2$ and hence $-v(\lambda(y^2)) \leq m - k\alpha_2 \leq 2l + k - 1$. Now because of Lemma 6.3.1 we know that $n > 2(l + k - 1) > 2l + k - 1$. Whence $\lambda(y^2)$, $\lambda(y)^2$ are both in $\mathcal{L}(2(l + k - 1)Q_\infty)$. Moreover, they agree at n places and thus by Lemma 4.3.5 we find that

$$\lambda(y^2) = \lambda(y)^2.$$

It follows that $-v(\lambda(y)) \leq l + \frac{k-1}{2}$. Using this improved estimate for $v(\lambda(y))$, we can now repeat the above process for $\lambda(y^3)$ to obtain $-v(\lambda(y)^3) \leq 3l + k - 1$, i.e. $-v(\lambda(y)) \leq l + \frac{k-1}{3}$. Continuing like this, i.e. using the improved estimates for $v(\lambda(y))$, Lemma 6.3.1 and Lemma 4.3.5, we arrive at the following inequality:

$$-v(\lambda(y)) \leq l + \frac{k-1}{\beta}. \tag{6.3.5}$$

Observing that $\alpha_0 = [\frac{m}{k}] \geq \frac{m-k+1}{k}$ and thus $[\frac{m}{\alpha_0}] \leq \frac{km}{m-k+1}$, we also have

$$-v(\lambda(x)) \leq \min\{c, [\frac{m}{\alpha_0}]\} = \min\{c, k(1 + \frac{k-1}{m-k+1})\}$$

which, together with (6.3.5), allows us to complete the proof.
Replacing x (resp. y) by $\lambda(x)$ (resp. $\lambda(y)$) in the defining equation (6.2.4) of the admissible function field we see that

$$\Big(\lambda(y)^k + g(\lambda(x), \lambda(y))\Big)(P_i') = \Big(b_l\lambda(x)^l + b_{l-1}\lambda(x)^{l-1} + \ldots + b_1\lambda(x) + b_0\Big)(P_i').$$

The function $\lambda(y)^k + g(\lambda(x), \lambda(y))$ on the left has at most $k(l + \frac{k-1}{\beta})$ poles (because of the way $g(x, y)$ is defined - cf. Lemma 6.2.3) and the function $b_l\lambda(x)^l + b_{l-1}\lambda(x)^{l-1} + \ldots + b_1\lambda(x) + b_0$ on the right has at most $lk(1 + \frac{k-1}{m-k+1})$ poles. Because $n > max\{m + c, k(l + \frac{k-1}{\beta}), lk(1 + \frac{k-1}{m-k+1})\}$ we can apply Lemma 4.3.5 and see that the two sides are equal. Hence λ preserves the defining equation of the admissible function field and thus (by Theorem 4.3.1) $\lambda(x)$ and $\lambda(y)$ determine an endomorphism, $\widetilde{\lambda}$ say, of it which agrees with λ on $\mathcal{L}(mQ_\infty)$. Because x, y are in the image of $\widetilde{\lambda}$, it is actually an automorphism also proving that $v(\lambda(x)) = k$ and $v(\lambda(y)) = l$. □

Using the above result we can now determine the automorphism group of a large class of codes associated with these function fields.

Theorem 6.3.6. *With the notation of Definition 6.2.1.*
Let $F/\mathbb{F}_q$ be an admissible function field over $\mathbb{F}_q$ of genus g where $l > k$. Assume that $m \geq l$. Let $D = \sum_{P \in J} P$ where $J \subseteq \mathbb{P}_F^{(1)} \setminus \{Q_\infty\}$. If $n > \max\{2g + 2, 2m, k(l + \frac{k-1}{\beta}), lk(1 + \frac{k-1}{m-k+1})\}$ then

$$Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty)) \cong Aut_{D,mQ_\infty}(F/\mathbb{F}_q).$$

Proof. By Proposition 3.3.4, $Aut_{D,G}(F/\mathbb{F}_q)$ can be regarded as a subgroup of $Aut(\mathcal{C}_\mathcal{L}(D, G))$. We are going to show that it is actually isomorphic to the whole group.
So let $\pi \in Aut(\mathcal{C}_\mathcal{L}(D, G))$; then because $\mathcal{C}_\mathcal{L}(D, G)) = \mathcal{C}_\mathcal{L}(\pi(D), G)$ we can consider the corresponding map λ_π (cf. Definition 4.3.6). We are going to show that λ_π is the restriction of an automorphism $\widetilde{\lambda_\pi} \in Aut_{D,G}(F/\mathbb{F}_q)$ to $\mathcal{L}(G)$. For simplicity we write λ instead of λ_π.

We apply Theorem 6.3.2 with $c = m$ which shows that λ yields an automorphism, $\widetilde{\lambda}$, of the function field F which agrees with λ on $\mathcal{L}(mQ_\infty)$. Whence, by Lemma 4.3.3, $\widetilde{\lambda}(G) = G$ (note that $\dim(mQ_\infty) \geq 3$ because $x, y \in \mathcal{L}(mQ_\infty)$). Furthermore, by Lemma 4.3.10, we also have $\widetilde{\lambda}(D) = \pi(D)$ showing that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$. □

Example 6.3.7. Specialising to the elliptic function field case we can give another proof of Corollary 5.3.2.
Let $J \subseteq \mathbb{P}_F^{(1)} \setminus \{Q_\infty\}$, set $n := |J|$. Suppose that

$$D = \sum_{P \in J} P \text{ and } G = mQ_\infty \text{ where } m \geq 3.$$

If $n \geq \max\{10, 2m + 1\}$ then

$$Aut(\mathcal{C}_\mathcal{L}(D, G)) \cong Aut_{D,G}(F/\mathbb{F}_q).$$

Proof. In this case $k = 2$, $l = 3$, $g = 1$, $\beta = 1$ and $c = m \geq 3$.
For Theorem 6.3.6 to hold we need $n > \max\{4, 2m, 8, 6(1 + \frac{1}{m-1})\}$. □

6.4 The Automorphism Group of Hermitian Codes - An Example

Here we consider the famous Hermitian codes associated with the function field $F/\mathbb{F}_{q^2}$ (where q is a power of a prime) given by the Hermitian curve equation

$$y^q + y = x^{q+1}. \tag{6.4.1}$$

All the relevant results about this class of codes can be found in [Xin95b] and [Sti91].
In his paper [Xin95b] Xing has given an exhaustive description of the automorphism groups of Hermitian codes; thus the following result is not new, but illustrates how the map λ can be used to obtain partial results without much computation.
First we recall some of the properties of the Hermitian function field (cf. [Sti91, p.203]).

- The genus of the Hermitian function field $F/\mathbb{F}_{q^2}$ is $g = q(q-1)/2$.
- F has $q^3 + 1$ places of degree one over $\mathbb{F}_{q^2}$, namely
  (1) the common pole Q_∞ of x and y, and
  (2) for each $\alpha \in \mathbb{F}_{q^2}$, there are q elements $\beta \in \mathbb{F}_{q^2}$ such that $\beta^q + \beta = \alpha^{q+1}$, and for all such pairs (α, β) there is a unique place $P_{\alpha,\beta} \in \mathbb{P}_F^{(1)}$ with $x(P_{\alpha,\beta}) = \alpha$ and $y(P_{\alpha,\beta}) = \beta$.
- $(x)_\infty = qQ_\infty$ and $(y)_\infty = (q+1)Q_\infty$.
- For $m \geq 0$, the elements $x^i y^j$ with $0 \leq i$, $0 \leq j \leq q-1$ and $iq + j(q+1) \leq m$ form a basis of $\mathcal{L}(mQ_\infty)$.

Using Theorem 6.3.6 one can easily deduce the following result.

Theorem 6.4.2. *Let $F/\mathbb{F}_{q^2}$ be the Hermitian function field defined by (6.4.1). Let $J = \mathbb{P}_F^{(1)} \setminus \{Q_\infty\}$ and $D = \sum_{P \in J} P$. If $q > 2$ and $q+1 \leq m \leq [\frac{q^3-1}{2}]$, then the automorphism group of the Hermitian code $\mathcal{C}_\mathcal{L}(D, mQ_\infty)$ is given by*

$$Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty)) \cong Aut_{D,mQ_\infty}(F/\mathbb{F}_{q^2}).$$

Proof. Because $q > 2$ and the fact that a Hermitian code is of length $n = q^3$, it is easy to check that

$$q^3 > \max\{2q^2, q(q+1)(1 + \frac{q-1}{2})\}.$$

(Note that the requirement $q^3 > 2q^2$ is the reason why we had to exclude the case $q = 2$.)
Thus as long as $n = q^3 > 2m$ as well, Theorem 6.3.6 applies and the result follows. □

Because the dual of a Hermitian Code $\mathcal{C}_{\mathcal{L}}(D, mQ_\infty)$ is given by $\mathcal{C}_{\mathcal{L}}(D, mQ_\infty)^\perp = \mathcal{C}_{\mathcal{L}}(D, (q^3+q^2-q-2-m)Q_\infty)$ we also find that

Corollary 6.4.3. *If $q > 2$ and $\frac{q^3+1}{2} + q^2 - q - 2 \leq m \leq q^3 + q^2 - 2q - 3$ then*

$$Aut(\mathcal{C}_{\mathcal{L}}(D, mQ_\infty)) \cong Aut_{D,mQ_\infty}(F/\mathbb{F}_{q^2}).$$

Proof. Note that for any linear code C we have $Aut(C) = Aut(C^\perp)$, where $C^\perp$ is the dual of C. Thus, for m in the given range, we notice that $\mathcal{C}_{\mathcal{L}}(D, mQ_\infty)^\perp = \mathcal{C}_{\mathcal{L}}(D, sQ_\infty)$ where $s = q^3 + q^2 - q - 2 - m$. It is trivial to check that $q + 1 \leq s \leq \frac{q^3-1}{2}$ and hence Theorem 6.4.2 applies. □

For $0 \leq m \leq q$ and for $m \geq q^3 + q^2 - 2q - 2$ the automorphism group of $\mathcal{C}_{\mathcal{L}}(D, mQ_\infty)$ can be determined to be either S_{q^3} or $AGL(2, q^2) \otimes S_q^{q^2}$ (see [Xin95b, Proof of the Main Theorem - (i) and (ii)]). So the only cases which are not covered by the above theorem are when $\frac{q^3-1}{2} < m < \frac{q^3+1}{2} + q^2 - q - 2$ which is a gap of $(q^2 - q - 2)$ (when q is odd) or a gap of $(q^2 - q - 1)$ (when q is even).

Chapter 7

A Paper Of Xing's Revisited

7.1 Introduction

We shall study in more detail a special class of elliptic codes which were discussed by Xing in [Xin95a]. We recall that an elliptic function field $F = \mathbb{F}_q(x, y)/\mathbb{F}_q$ is given by the following equation:

$$H(x, y) \equiv y^2 + h(x)y + f(x) = 0 \tag{7.1.1}$$

where $h(x) = a_1x + a_2$ (a_1, $a_2 \in \mathbb{F}_q$), and $f(x) \in \mathbb{F}_q[x]$ with $\deg f = 3$ (cf. Section 5.2). We shall use the notation introduced in Chapter 5:

- Let Q_∞ be the common pole of x and y. Note that in the elliptic function field case $H(\mathbb{F}_q) = \mathbb{P}_F^{(1)}$. Furthermore, we know that the places of degree one of an elliptic function field $F/\mathbb{F}_q$ form a finite abelian group, denoted by $H(\mathbb{F}_q)$ (cf. Section 2.6.2). Conventionally, the place at infinity, Q_∞, acts as the zero element of this group. Let $\oplus$ denote the group operation on $H(\mathbb{F}_q)$.

- Let $H(\mathbb{F}_q) = \{Q_\infty, P_1, P_2, \ldots, P_{2r+t}\}$ (with $t = 0, 1, 3$) and suppose that $P_{2i-1} = \ominus P_{2i}$ for $1 \leq i \leq r$ and $H_r(\mathbb{F}_q) = \{P_{2r+1}, \ldots, P_{2r+t}\}$.

- Let $H(\mathbb{F}_q)[2] = H_r(\mathbb{F}_q) \cup \{Q_\infty\}$, in other words it contains all the ramified places of degree one. It is easy to see that $H(\mathbb{F}_q)[2]$ is the 2-torsion part of the abelian group $H(\mathbb{F}_q)$.

- Furthermore, all $\mathbb{F}_q$-automorphisms of F fixing the place at infinity form a finite group whose order divides 24 (cf. [Sil86, pp.325-326]).

Xing [Xin95a] proved the following theorem concerning the automorphism group of elliptic codes.

Theorem. *Let the divisor* $D = \sum_{P \in H(\mathbb{F}_q) \setminus \{Q_\infty\}} P$. *If* $|H(\mathbb{F}_q)| \geq 27$ *and* $5 \leq m \leq |H(\mathbb{F}_q)| - 9$, *then*

$$Aut(\mathcal{C}_{\mathcal{L}}(D, G)) \cong Aut_{D,mQ_\infty}(F/\mathbb{F}_q)$$

Proof. see [Xin95a, pp. 4068–4071]. □

By looking in more detail at Xing's proof we were able to improve his result slightly. The method we use is mainly due to Xing and involves looking at the different cases where the 2-torsion subgroup of the elliptic curve in question, $H(\mathbb{F}_q)[2]$, has size $|H(\mathbb{F}_q)[2]| = (t+1) = 1,2$ or 4. Consequently, we were able to show (see Theorem 7.4.1) that we only require $H(\mathbb{F}_q) \geq 19 - t$ in our version. Moreover, m is allowed to range over $3 \leq m \leq |H(\mathbb{F}_q)| - (9-t)$. The last step of the proof of our version of Xing's result also uses results from the previous chapters.

In the next sections we state and prove all the preliminary results needed for our version of Xing's result which will eventually be proved in Section 7.4.

7.2 Preliminary Results - Group Theoretical Lemmas

The following results are all rather tedious and trivial. They will be required later on in order to make use of the group structure of $H(\mathbb{F}_q)$.

Definition 7.2.1. *Let* G *be any finite abelian group. Let* $G[k]$ *denote the* k*-torsion subgroup of* G, *i.e.* $G[k] = \{g \in G | \, kg = 0\}$.

The first lemma just says that as long as a finite group G is big enough then we can find three pairwise distinct elements not in the 2-torsion adding up to a 2-torsion element.

Lemma 7.2.2. *Let* G *be a finite abelian group such that* $|G| > 3|G[2]| + 2$. *If* $g \in G[2]$ *then there exists* $h_1, h_2, h_3 \in G \setminus G[2]$ *such that* $\{\pm h_i\} \cap \{\pm h_j\} = \emptyset$ *for* $1 \leq i < j \leq 3$ *and* $h_1 + h_2 + h_3 = g$.

Proof. First of all we observe that with out loss of generality we can assume that $g = 0$. If h_1, h_2, h_3 satisfy the conditions stipulated in the Lemma and $h_1 + h_2 + h_3 = 0$ then $h_i + g$ ($i = 1, 2, 3$ and $g \in G[2]$) will also satisfy the conditions and we have

$$(h_1 + g) + (h_2 + g) + (h_3 + g) = g.$$

Choose $h_1 \in G \setminus G[2]$. Pick $h_2 \in G \setminus (G[2] \cup \{h_1, -2h_1\} \cup (-h_1 + G[2]) \cup \{h \in G | 2h = -h_1\})$. Note that we can find such a h_2 because $|G[2]| = |(-h_1 + G[2]| = |\{h \in G | 2h = -h_1\}|$ and $|G| > 3G[2] + 2$. Let $h_3 = -(h_1 + h_2)$. Then h_1, h_2, h_3 satisfy the conditions of the Lemma because

if $h_3 \in G[2]$ then $h_2 \in -h_1 + G[2]$: contradicting the choice of h_2;
if $h_3 = h_1$ then $h_2 = g - 2h_1$: contradicting the choice of h_2;
if $h_3 = -h_1$ then $h_2 = g$: contradicting the choice of h_2;
if $h_3 = h_2$ then $2h_2 = -h_1 + g$: contradicting the choice of h_2;
if $h_3 = -h_2$ then $h_1 = 0$: contradicting the choice of h_1.
Finally $h_2 \neq \pm h_1$, and $h_2 \notin G[2]$ by way of choice of h_2. □

The next two lemmas, like the previous one, are about finding elements in a finite group G which have to add up to an element in the 2-torsion of G while satisfying certain conditions.

Lemma 7.2.3. *Let G be a finite abelian group such that $|G| \geq 14$ and $G[2] = \{0, g\}$. Suppose that $g_1, g_2 \in G \backslash G[2]$ are such that $\pm g_1 \neq \pm g_2$. There exists two elements $h_1, h_2 \in G \backslash (G[2] \cup \{\pm g_1, \pm g_2\})$ such that $\pm h_1 \neq \pm h_2$ and $h_1 + h_2 = g$.*

Proof. Choose $h_1 \in G \setminus (G[2] \cup \{\pm g_1, \pm g_2\} \cup \{g \pm g_1, g \pm g_2\} \cup \{h \in G | 2h = g\})$. For this to be possible we need the above set to be non-empty, i.e. $|G| > 2 + 4 + 4 + 2 = 12$.
Put $h_2 = h_1 - g$. Then h_1 and h_2 are the desired elements because:
if $h_2 \in G[2]$ then $h_1 \in G[2]$: contradicting the choice of h_1;
if $h_2 = h_1$ then $g = 0$: contradicting $g \neq 0$;
if $h_2 = -h_1$ then $2h_1 = g$: contradicting the choice of h_1;
if $h_2 = \pm g_i$ then $h_1 = g \pm g_i$ $(i = 1, 2)$: contradicting the choice of h_1. □

Lemma 7.2.4. *Let G be a finite abelian group. Assume that $|G| \geq 15$ and $G[2] = \{0\}$. Suppose also that $g_1, g_2 \in G \setminus \{0\}$ such that $\pm g_1 \neq \pm g_2$. For $2 \leq k \leq |G| - 7$ we can find pairwise distinct elements $h_1, h_2, \ldots, h_k \in G \setminus \{0, \pm g_1, \pm g_2\}$ such that $\sum_{j=1}^{k} h_j = 0$.*

Proof. We look at the two cases k odd and k even.
k even:

In this case the result is trivial. Obviously, $|G \setminus (G[2] \cup \{\pm g_1, \pm g_2\})| = |G| - 5 > k$. Hence one can choose $\frac{k}{2}$ elements $h_1, h_2, \ldots, h_{\frac{k}{2}}$ satisfying $\pm h_i \neq \pm h_j$, $1 \leq i < j \leq \frac{k}{2}$. Now define $h_{\frac{k}{2}+t} = -h_t$ for $1 \leq t \leq \frac{k}{2}$. One sees immediately that these elements satisfy the conditions of the lemma.
k odd:

Here we have to find three elements $h_1, h_2, h_3 \in G \setminus \{0, \pm g_1, \pm g_2\}$ such that $\pm h_i \neq \pm h_j$, $1 \leq i < j \leq 3$, and $h_1 + h_2 + h_3 = 0$. Choose $h_1 \in G \setminus \{0, \pm g_1, \pm g_2\}$. Now choose

$$h_2 \in G \setminus (\{0, \pm h_1, \pm g_1, \pm g_2\} \cup \{-2h_1\} \cup \{h \in G | 2h = -h_1\} \cup \{-h_1 \pm g_1, -h_1 \pm g_2\}).$$

For this to be possible we need $|G| > 7 + 1 + 1 + 4 = 13$. Put $h_3 = -(h_1 + h_2) \neq 0$. We check that the 3 elements do satisfy the necessary conditions:

Suppose $h_3 = h_1$ then $h_2 = -2h_1$: contradicting the choice of h_2;
Suppose $h_3 = -h_1$ then $h_2 = 0$: contradicting the choice of h_2;
Suppose $h_3 = h_2$ then $2h_2 = -h_1$: contradicting the choice of h_2;
Suppose $h_3 = -h_2$ then $h_1 = 0$: contradicting the choice of h_1;
Suppose $h_3 = \pm g_{1,2}$ then $h_2 = -h1 \mp g_{1,2}$: contradicting the choice of h_2.
The other conditions are obvious from the choice of h_2 and h_1. Having found these elements we proceed as before selecting the remaining $(k-3)$ elements from the set $G\backslash\{0, \pm g_1, \pm g_2, \pm h_1, \pm h_2, \pm h_3\}$, i.e. we need $0 \leq (k-3) \leq |G| - 11$. □

The following lemma will play a role in the proof of Lemma 7.3.3.

Lemma 7.2.5. *Let $r \geq 0$. If $G = \{0, h_1, h_2, \ldots, h_{2r}, h_{2r+1}, h_{2r+2}, h_{2r+3}\}$ is a finite abelian group such that $|G[2]| = 4$ and*

$$h_{2i-1} + h_{2i} = h_{2r+1}, \qquad \text{for } 1 \leq i \leq r$$

then $h_{2r+2} + h_{2r+3} = h_{2r+1}$.

Proof. First of all we observe that

$$\sum_{k=1}^{2r+3} h_k = \sum_{g \in G[2]} g = 0.$$

Then we note that

$$\sum_{k=1}^{2r+3} h_k = rh_{2r+1} + h_{2r+1} + h_{2r+2} + h_{2r+3} = 0$$

Thus we have that

$$(r+1)h_{2r+1} = -(h_{2r+2} + h_{2r+3}) \tag{7.2.6}$$

Now looking at the image of h_{2r+1} in $G/G[2]$ we note that

$$\frac{2r+4}{4}(h_{2r+1} + G[2]) = G[2]$$

because $|G/G[2]| = (2r+4)/4$. Thus $((2r+4)/4)h_{2r+1} \in G[2]$ and hence $(2r+4)/2)h_{2r+1} = 0$, i.e.

$$(r+2)h_{2r+1} = 0 \tag{7.2.7}$$

Subtracting (7.2.6)from (7.2.7) yields $h_{2r+1} = h_{2r+2} + h_{2r+3}$. □

7.3 Main Results

First of all we introduce some notation which will be used throughout this section:

- Let $F/\mathbb{F}_q$ be the elliptic function field defined by $y^2 + h(x)y + f(x) = 0$ (cf. (7.1.1)).
- Let $\oplus$ be the group operation defined on the places of degree one in F.
- As usual let $H(\mathbb{F}_q) = \mathbb{P}_F^{(1)} = \{Q_\infty, P_1, P_2, \ldots, P_{2r+t}\}$. Assume that $P_{2i-1} \oplus P_{2i} = Q_\infty$ for $1 \leq i \leq r$ and $H(\mathbb{F}_q)[2] = \{Q_\infty, P_{2r+1}, \ldots, P_{2r+t}\}$.
- Let $(P_1', P_2', \ldots, P_{2r+t}')$ be a permutation of $(P_1, P_2, \ldots, P_{2r+t})$.
- Let $D = \sum_{i=1}^{2r+t} P_i$ and $D' = \sum_{i=1}^{2r+t} P_i'$ be two divisors (note that they are equal as divisors).
- We may also assume that $P_{2i-1} = (x_i, y_i)$ for $1 \leq i \leq r$ and $P_{2i} = (x_i, -y_i - h(x_i))$ with $y_i \neq -y_i - h(x_i)$. We also have that $P_{2r+j} = (x_{r+j}, 0)$, $1 \leq j \leq t$.
- A basis of $\mathcal{L}(mQ_\infty)$ is given by $\{1, x, \ldots, x^{[\frac{m}{2}]}, y, yx, \ldots, yx^{[\frac{m-3}{2}]}\}$.

Recall that in the elliptic function field case there is a very nice result linking a principal divisors only involving places of degree one to a certain sum in $H(\mathbb{F}_q)$, i.e. suppose $J = \sum_{i=1}^{k} P_i$ where $P_i \in H(\mathbb{F}_q)$ for $1 \leq i \leq k$. Then J is principal if and only if $deg(J) = 0$ and $\oplus \sum_{i=1}^{k} P_i = 0$ (cf. Lemma 2.6.15).

This result allows us to prove the following claim which will be used over and over again in the lemmas following it:

Claim 7.3.1. *If $\mathcal{C}_\mathcal{L}(D, mQ_\infty) = \mathcal{C}_\mathcal{L}(D', mQ_\infty)$, where $2 \leq m \leq |H(\mathbb{F}_q)| - 1$, and there are indices $i_1 < i_2 < \ldots < i_m$ such that $\oplus \sum_{j=1}^{m} P_{i_j} = 0$, then $\oplus \sum_{j=1}^{m} P'_{i_j} = 0$ as well.*

Proof. There exists $w \in H$ satisfying $(w) = \sum_{j=1}^{m} P_{i_j} - mQ_\infty$ (by Lemma 2.6.15), hence it follows that there exists $w' \in \mathcal{L}(mQ_\infty)$ such that $w(P_i) = w'(P_i')$, $i = 1, 2, \ldots, 2r+t$, so $(w')_0 \geq \sum_{j=1}^{m} P'_{i_j}$. On the other hand we have $(w')_\infty \leq mQ_\infty$, therefore the divisor (w') is equal to $\sum_{j=1}^{m} P'_{i_j} - mQ_\infty$. Thus our claim follows by applying Lemma 2.6.15 once more. □

Remark 7.3.2. The observant reader will have realised that the above Lemma is just a special case of Lemma 4.3.16.

The following results are closely related to Lemma 2 in [Xin95a, pp.4064–4068]. They use the same methods but take advantage of the additional information of the size of the 2-torsion subgroup of $H(\mathbb{F}_q)$. In essence, the results prove the following:
if $\mathcal{C}_\mathcal{L}(D, mQ_\infty) = \mathcal{C}_\mathcal{L}(D', mQ_\infty)$ then $P_{2i-1} \oplus P_{2i} = Q_\infty \Rightarrow P'_{2i-1} \oplus P'_{2i} = Q_\infty$ for $1 \leq i \leq r$.

7.3.1 Case 1: $H(\mathbb{F}_q)[2] = \{Q_\infty, P_{2r+1}, P_{2r+2}, P_{2r+3}\}$

Lemma 7.3.3. *Suppose* $|H(\mathbb{F}_q)| \geq 16$ *and* $4 \leq m \leq |H(\mathbb{F}_q)| - 6$. *If* $\mathcal{C}_{\mathcal{L}}(D, mQ_\infty) = \mathcal{C}_{\mathcal{L}}(D', mQ_\infty)$ *then* $P'_{2i-1} \oplus P'_{2i} = Q_\infty$ *for* $1 \leq\ i \leq r$, *and* $H(\mathbb{F}_q)[2] = \{Q_\infty, P'_{2r+1}, P'_{2r+2}, P'_{2r+3}\}$.

Proof. Case m is even:

For any indices $1 \leq i \neq j \leq r$ we can find indices $i_1, \ldots, i_{\frac{m}{2}-1} \in \{1, 2, \ldots, r\} \setminus \{i, j\}$ such that

$$P_{2i-1} \oplus P_{2i} \oplus \sum_{k=1}^{\frac{m}{2}-1} (P_{2i_k-1} \oplus P_{2i_k}) = Q_\infty$$

and

$$P_{2j-1} \oplus P_{2j} \oplus \sum_{k=1}^{\frac{m}{2}-1} (P_{2i_k-1} \oplus P_{2i_k}) = Q_\infty$$

(Note that this can be done because $1 \leq \frac{m}{2} - 1 \leq r - 2$.)

Applying the claim to the two equations above, one gets

$$P'_{2i-1} \oplus P'_{2i} \oplus \sum_{k=1}^{\frac{m}{2}-1} (P'_{2i_k-1} \oplus P'_{2i_k}) = Q_\infty \tag{7.3.4}$$

and

$$P'_{2j-1} \oplus P'_{2j} \oplus \sum_{k=1}^{\frac{m}{2}-1} (P'_{2i_k-1} \oplus P'_{2i_k}) = Q_\infty. \tag{7.3.5}$$

(7.3.4)$\ominus$(7.3.5) gives

$$P'_{2i-1} \oplus P'_{2i} = P'_{2j-1} \oplus P'_{2j}$$

Let $P'_{2i-1} \oplus P'_{2i} = Q$ $(1 \leq i \leq r)$, it is obvious that $P'_k \neq Q$ for any $1 \leq k \leq 2r$. Otherwise one would have $P_{2[\frac{k+1}{2}]-1} = Q_\infty$ or $P_{2[\frac{k+1}{2}]} = Q_\infty$. We claim that Q is exactly the zero element Q_∞. Suppose that this is false. Then $Q \in \{P'_{2r+1}, \ldots, P'_{2r+3}\}$. We may assume that $Q = P'_{2r+1}$. By (7.3.4) we immediately get $(m/2)Q = Q_\infty$, i.e. $mQ = Q_\infty$. By Lemma 7.2.2 there exists 3 indices $1 \leq i < j < k \leq 2r$ such that $[\frac{i+1}{2}], [\frac{j+1}{2}], [\frac{k+1}{2}]$ are pairwise distinct and

$$P_i \oplus P_j \oplus P_k = P_{2r+1}$$

(Note: this is where we need $|H(\mathbb{F}_q)| \geq 16$)

We now choose indices $l_1, \ldots, l_{\frac{m}{2}-2} \in \{1, 2, \ldots, r\} \setminus \{[\frac{i+1}{2}], [\frac{j+1}{2}], [\frac{k+1}{2}]\}$. Applying the claim to

$$P_i \oplus P_j \oplus P_k \oplus P_{2r+1} \oplus \sum_{h=1}^{\frac{m}{2}-2} (P_{2l_h-1} \oplus P_{2l_h}) = Q_\infty,$$

we obtain

$$P'_i \oplus P'_j \oplus P'_k \oplus P'_{2r+1} \oplus \sum_{h=1}^{\frac{m}{2}-2} (P'_{2l_h-1} \oplus P'_{2l_h}) = Q_\infty. \tag{7.3.6}$$

Now for any $1 \leq i \leq 2r$, it is easy to see that the following two sets are equal:

$$\{P_i, \ominus P_i\} = \{P_{2[\frac{i+1}{2}]-1}, P_{2[\frac{i+1}{2}]}\}, \qquad \{P'_i, Q \ominus P'_i\} = \{P'_{2[\frac{i+1}{2}]-1}, P'_{2[\frac{i+1}{2}]}\}. \tag{7.3.7}$$

Hence the equality

$$(\ominus P_i) \oplus (\ominus P_j) \oplus (\ominus P_k) \oplus P_{2r+1} \oplus \sum_{h=1}^{\frac{m}{2}-2} (P_{2l_h-1} \oplus P_{2l_h}) = Q_\infty,$$

implies

$$(Q \ominus P'_i) \oplus (Q \ominus P'_j) \oplus (Q \ominus P'_k) \oplus P'_{2r+1} \oplus \sum_{h=1}^{\frac{m}{2}-2} (P'_{2l_h-1} \oplus P'_{2l_h}) = Q_\infty. \tag{7.3.8}$$

(7.3.6) $\oplus$ (7.3.8) yields $2(\frac{m}{2}-2)Q \oplus 3Q \oplus 2Q = (m+1)Q = Q_\infty$. We already have $mQ = Q_\infty$, thus $Q = Q_\infty$. This contradicts our assumption and hence proves the lemma in the case m even.

Case m is odd:

Note that $\sum_{k=2r+1}^{2r+3} P_k = Q_\infty$, thus for any 2 indices $1 \leq i \neq j \leq r$ we can choose $(m-5)/2$ indices in $\{1, 2, \ldots, r\} \setminus \{i, j\}$ such that

$$P_{2i-1} \oplus P_{2i} \oplus \sum_{h=1}^{\frac{m-5}{2}} (P_{2i_h-1} \oplus P_{2i_h}) \oplus P_{2r+1} \oplus P_{2r+2} \oplus P_{2r+3} = Q_\infty.$$

(Note: this is possible since $0 \leq \frac{m-5}{2} \leq r-4 < r-2$)

Hence we get

$$P'_{2i-1} \oplus P'_{2i} \oplus \sum_{h=1}^{\frac{m-5}{2}} (P'_{2i_h-1} \oplus P'_{2i_h}) \oplus P'_{2r+1} \oplus P'_{2r+2} \oplus P'_{2r+3} = Q_\infty \tag{7.3.9}$$

and similarly

$$P'_{2j-1} \oplus P'_{2j} \oplus \sum_{h=1}^{\frac{m-5}{2}} (P'_{2i_h-1} \oplus P'_{2i_h}) \oplus P'_{2r+1} \oplus P'_{2r+2} \oplus P'_{2r+3} = Q_\infty$$

Therefore, as before, we obtain $P'_{2i-1} \oplus P'_{2i} = P'_{2j-1} \oplus P'_{2j}$ for $1 \leq i, j \leq r$. Let $P'_{2i-1} \oplus P'_{2i} = Q$. Again, we observe that $Q \neq P'_k$, $k = 1, 2, \ldots, 2r$ so that $Q = Q_\infty$ or $Q \in \{P'_{2r+1}, P'_{2r+2}, P'_{2r+3}\}$. We want to show that $Q = Q_\infty$. Suppose that this is false then we may assume that $Q = P'_{2r+1}$. By Lemma 7.2.5 we know that $P'_{2r+2} \oplus P'_{2r+3} = P'_{2r+1}$. So (7.3.9) yields: $Q + (\frac{m-5}{2})Q + 2Q = (\frac{m+1}{2})Q = Q_\infty$, i.e. $(m+1)Q = Q_\infty$. Now we choose indices $1 \leq i < j < k \leq 2r$ such that

$[\frac{i+1}{2}], [\frac{j+1}{2}], [\frac{k+1}{2}]$ are pairwise distinct and $P_i \oplus P_j \oplus P_k = Q_\infty$. This is possible by Lemma 7.2.2. Now we choose indices $i_1, i_2, \ldots, i_{\frac{m-3}{2}} \in \{1, 2, \ldots, r\} \setminus \{[\frac{i+1}{2}], [\frac{j+1}{2}], [\frac{k+1}{2}]\}$ and consider

$$P_i \oplus P_j \oplus P_k \oplus \sum_{h=1}^{\frac{m-3}{2}} (P_{2i_h-1} \oplus P_{2i_h}) = Q_\infty.$$

(Note: this is possible since $1 \leq \frac{m-3}{2} \leq r-3$)

The above equation yields

$$P'_i \oplus P'_j \oplus P'_k \oplus \sum_{h=1}^{\frac{m-3}{2}} (P'_{2i_h-1} \oplus P'_{2i_h}) = Q_\infty. \tag{7.3.10}$$

Applying the claim to

$$(\ominus P_i) \oplus (\ominus P_j) \oplus (\ominus P_k) \oplus \sum_{h=1}^{\frac{m-3}{2}} (P_{2i_h-1} \oplus P_{2i_h}) = Q_\infty.$$

and using (7.3.7) we obtain

$$(Q \ominus P'_i) \oplus (Q \ominus P'_j) \oplus (Q \ominus P'_k) \oplus \sum_{h=1}^{\frac{m-3}{2}} (P'_{2i_h-1} \oplus P'_{2i_h}) = Q_\infty. \tag{7.3.11}$$

(7.3.10)$\oplus$(7.3.11) yields:

$$3Q \oplus (m-3)Q = mQ = Q_\infty.$$

So $(m+1)Q = Q_\infty$ and $mQ = Q_\infty$, hence $Q = Q_\infty$, giving a contradiction. □

7.3.2 Case 2: $H(\mathbb{F}_q)[2] = \{Q_\infty, P_{2r+1}\}$

Lemma 7.3.12. *Suppose* $|H(\mathbb{F}_q)| \geq 14$ *and* $4 \leq m \leq |H(\mathbb{F}_q)|-4$. *If* $\mathcal{C}_\mathcal{L}(D, mQ_\infty) = \mathcal{C}_\mathcal{L}(D', mQ_\infty)$ *then* $P'_{2i-1} \oplus P'_{2i} = Q_\infty$ *for* $1 \leq\ i \leq r$, *and* $H(\mathbb{F}_q)[2] = \{Q_\infty, P'_{2r+1}\}$.

Proof. Case m is even:

Exactly the same proof as in Lemma 7.3.3 works.

Case m is odd:

By Lemma 7.2.3 we know that for any 2 indices $1 \leq i \neq j \leq r$ there exists distinct indices $i_1, i_2 \in \{1, 2, \ldots, r\} \setminus \{i, j\}$ such that $P_{i_1} \oplus P_{i_2} = P_{2r+1}$. (N.B.: this is where we need $|H(\mathbb{F}_q)| \geq 14$.) So now we choose $\frac{m-5}{2}$ distinct indices $i_3, i_4, \ldots, i_{\frac{m-5}{2}} \in \{1, 2, \ldots, r\} \setminus \{i, j, i_1, i_2\}$ and get

$$P_{2i-1} \oplus P_{2i} \oplus P_{2r+1} \oplus P_{i_1} \oplus P_{i_2} \oplus \sum_{k=3}^{2+\frac{m-5}{2}} (P_{2k-1} \oplus P_{2k}) = Q_\infty.$$

Applying the claim we get

$$P'_{2i-1} \oplus P'_{2i} \oplus P'_{2r+1} \oplus P'_{i_1} \oplus P'_{i_2} \oplus \sum_{k=3}^{2+\frac{m-5}{2}} (P'_{2k-1} \oplus P'_{2k}) = Q_\infty. \tag{7.3.13}$$

Similarly we obtain

$$P'_{2j-1} \oplus P'_{2j} \oplus P'_{2r+1} \oplus P'_{i_1} \oplus P'_{i_2} \oplus \sum_{k=3}^{2+\frac{m-5}{2}} (P'_{2k-1} \oplus P'_{2k}) = Q_\infty.$$

Subtracting one equation from the other we obtain as before

$$P'_{2i-1} \oplus P'_{2i} = P'_{2j-1} \oplus P'_{2j} = Q$$

We want to show that $Q = Q_\infty$. If this is false then as in Lemma 7.3.3 we can assume that $Q = P'_{2r+1}$. From (7.3.7) and $(\ominus P_{i_1}) \oplus (\ominus P_{i_2}) \oplus P_{2r+1} = Q_\infty$ we deduce that

$$P'_{2i-1} \oplus P'_{2i} \oplus P'_{2r+1} \oplus (Q \ominus P'_{i_1}) \oplus (Q \ominus P'_{i_2}) \oplus \sum_{k=3}^{2+\frac{m-5}{2}} (P'_{2k-1} \oplus P'_{2k}) = Q_\infty. \tag{7.3.14}$$

Now (7.3.13)$\oplus$(7.3.14) yields:

$$2Q + 2Q + 2Q + (m-5)Q = (m+1)Q = Q_\infty$$

But by Lemma 7.2.2 we can choose three indices $i, j, k \in \{1, 2, \ldots, 2r\}$ such that $[\frac{i+1}{2}], [\frac{j+1}{2}], [\frac{k+1}{2}]\}$ are all distinct and

$$P_i \oplus P_j \oplus P_k = Q_\infty.$$

We now choose $\frac{m-3}{2}$ distinct indices $i_1, i_2, \ldots, i_{\frac{m-3}{2}} \in \{1, 2, \ldots, r\} \setminus \{[\frac{i+1}{2}], [\frac{j+1}{2}], [\frac{k+1}{2}]\}$ and get

$$P_i \oplus P_j \oplus P_k \oplus \sum_{k=1}^{\frac{m-3}{2}} (P_{2i_k-1} \oplus P_{2i_k}) = Q_\infty.$$

Applying the claim we obtain

$$P'_i \oplus P'_j \oplus P'_k \oplus \sum_{k=1}^{\frac{m-3}{2}} (P'_{2i_k-1} \oplus P'_{2i_k}) = Q_\infty \tag{7.3.15}$$

and from

$$(\ominus P_i) \oplus (\ominus P_j) \oplus (\ominus P_k) \oplus \sum_{k=1}^{\frac{m-3}{2}} (P_{2i_k-1} \oplus P_{2i_k}) = Q_\infty$$

and (7.3.7) we get

$$(Q \ominus P'_i) \oplus (Q \ominus P'_j) \oplus (Q \ominus P'_k) \oplus \sum_{k=1}^{\frac{m-3}{2}} (P'_{2i_k-1} \oplus P'_{2i_k}) = Q_\infty \tag{7.3.16}$$

(7.3.15)$\oplus$(7.3.16) yields

$$3Q + (m-3)Q = mQ = Q_\infty.$$

Hence $(m+1)Q = Q_\infty$ and $mQ = Q_\infty$ implying $Q = Q_\infty$ a contradiction. Therefore the result follows. □

7.3.3 Case 3: $H(\mathbb{F}_q)[2] = \{Q_\infty\}$

Lemma 7.3.17. *Suppose* $|H(\mathbb{F}_q)| \geq 15$ *and* $4 \leq m \leq |H(\mathbb{F}_q)|-5$. *If* $\mathcal{C}_{\mathcal{L}}(D, mQ_\infty) = \mathcal{C}_{\mathcal{L}}(D', mQ_\infty)$ *then* $P'_{2i-1} \oplus P'_{2i} = Q_\infty$ *for* $1 \leq i \leq r$, *and* $H(\mathbb{F}_q)[2] = \{Q_\infty\}$.

Proof. Case m is even:
Following the proof of the even case in Lemma 7.3.3 we obtain

$$P'_{2i-1} \oplus P'_{2i} = P'_{2j-1} \oplus P'_{2j}$$

Let $P'_{2i-1} \oplus P'_{2i} = Q$ $(1 \leq i \leq r)$, it is obvious that $P'_k \neq Q$ for any $1 \leq k \leq 2r$. Otherwise one would have $P_{2[\frac{k+1}{2}]-1} = Q_\infty$ or $P_{2[\frac{k+1}{2}]} = Q_\infty$. Thus $Q = Q_\infty$.
Case m is odd:

By Lemma 7.2.4 for any two indices $1 \leq i \neq j \leq r$ we can find $m-2$ pairwise distinct indices $i_1, i_2, \ldots, i_{m-2} \in \{1, 2, \ldots, 2r\} \setminus \{2i-1, 2i, 2j-1, 2j\}$ such that

$$\oplus \sum_{k=1}^{m-2} P_{i_k} = Q_\infty$$

(Note that this can only be done when $|H(\mathbb{F}_q)| \geq 15$)
Applying the claim to

$$P_{2i-1} \oplus P_{2i} \oplus \sum_{k=1}^{m-2} P_{i_k} = Q_\infty$$

we get

$$P'_{2i-1} \oplus P'_{2i} \oplus \sum_{k=1}^{m-2} P'_{i_k} = Q_\infty \tag{7.3.18}$$

and similarly

$$P'_{2j-1} \oplus P'_{2j} \oplus \sum_{k=1}^{m-2} P'_{i_k} = Q_\infty. \tag{7.3.19}$$

So again we have

$$P'_{2i-1} \oplus P'_{2i} = P'_{2j-1} \oplus P'_{2j} \neq P'_k, k = 1, 2, \ldots, 2r.$$

Thus

$$P'_{2i-1} \oplus P'_{2i} = Q_\infty$$

as desired. □

7.4 Xing's Result Improved

Now we are in a position to state and prove the main theorem. We recall some of the notation used in the previous section:

- Let $F/\mathbb{F}_q$ be the elliptic function field defined by $y^2 + h(x)y + f(x) = 0$ (cf. (7.1.1)).
- Let $\oplus$ be the group operation defined on the places of degree one in F.
- Let $H(\mathbb{F}_q) = \mathbb{P}_F^{(1)} = \{Q_\infty, P_1, P_2, \ldots, P_{2r+t}\}$. Assume that $P_{2i-1} \oplus P_{2i} = Q_\infty$ for $1 \leq i \leq r$ and $H(\mathbb{F}_q)[2] = \{Q_\infty, P_{2r+1}, \ldots, P_{2r+t}\}$.
- We may also assume that $P_{2i-1} = (x_i, y_i)$ for $1 \leq i \leq r$ and $P_{2i} = (x_i, -y_i - h(x_i))$ with $y_i \neq -y_i - h(x_i)$. We also have that $P_{2r+j} = (x_{r+j}, 0)$, $1 \leq j \leq t$.
- A basis of $\mathcal{L}(mQ_\infty)$ is given by $\{1, x, \ldots, x^{[\frac{m}{2}]}, y, yx, \ldots, yx^{[\frac{m-3}{2}]}\}$.

Theorem 7.4.1. *Suppose $D = \sum_{P \in H(\mathbb{F}_q) \setminus \{Q_\infty\}} P$. If $|H(\mathbb{F}_q)| \geq (19 - t)$ and $3 \leq m \leq |H(\mathbb{F}_q)| - (9 - t)$, then*

$$Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty)) \cong Aut_{D,mQ_\infty}(\mathbb{F}_q(E)).$$

Proof. Obviously, if we permute the coordinates of a linear code, then the automorphism group of the new linear code is isomorphic to that of the original code. Thus we may as well assume that the notation introduced above is still valid in this set-up, i.e. $D = \sum_{i=1}^{2r+t} P_i$, where $P_{2i-1} \oplus P_{2i} = Q_\infty$ for $1 \leq i \leq r$. It is obvious by Proposition 3.3.4 that different elements of $Aut_{D,mQ_\infty}(\mathbb{F}_q(E))$ induce different automorphisms of $\mathcal{C}_\mathcal{L}(D, mQ_\infty)$. Conversely, if $\pi \in Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty)) \subseteq S_{2r+t}$, that is

$$\begin{aligned}\mathcal{C}_\mathcal{L}(D, mQ_\infty) &= \{(x(P_1), x(P_2), \ldots, x(P_{2r+t})) \mid x \in \mathcal{L}(mQ_\infty)\} \\ &= \{(x(P_{\pi(1)}), x(P_{\pi(2)}), \ldots, x(P_{\pi(2r+t)})) \mid x \in \mathcal{L}(mQ_\infty)\} = \mathcal{C}_\mathcal{L}(\pi(D), mQ_\infty),\end{aligned}$$

then we will prove that π can be induced by some element in $Aut_{D,mQ_\infty}(H/\mathbb{F}_q)$.

The case $m = 3$ is covered by Corollary 6.3.7. The proof we are about to give will hold for $4 \leq m \leq |H(\mathbb{F}_q)| - (9 - t)$.

Let $P'_i = P_{\pi(i)}$, $i = 1, 2, \ldots, 2r+t$, and $D' = \sum_{i=1}^{2r+t} P'_i$, then $\mathcal{C}_{\mathcal{L}}(D, mQ_\infty) = \mathcal{C}_{\mathcal{L}}(D', mQ_\infty)$. Consider the map λ from $\mathcal{L}(mQ_\infty)$ onto itself as described in Definition 4.3.6. The idea is to show that $-v_{Q_\infty}(\lambda(x)) = 2$ which will then allow us to apply Theorem 6.3.2 to deduce that λ is the restriction of an automorphism, $\widetilde{\lambda}$, of F such that $\lambda \in Aut_{D,mQ_\infty}(F/\mathbb{F}_q)$.

Now, by Lemma 7.3.3, Lemma 7.3.12, and Lemma 7.3.17 we know that $P'_{2i-1} \oplus P'_{2i} = Q_\infty$ for any $1 \leq i \leq r$. Let $P'_{2i-1} = (x'_i, y'_i)$ for $1 \leq i \leq r$, and $P'_{2r+j} = (x'_{r+j}, y'_{r+j})$, $j = 1, \ldots, t$, then

$$P'_{2i} = (x'_i, -y'_i - h(x'_i)) \quad \text{and} \quad y'_i \neq -y'_i - h(x'_i)$$

for $i = 1, 2, \ldots, r$. For any $f(x) \in \mathbb{F}_q[x]$ with $deg(f) \leq [\frac{m}{2}]$, one can find $p(x), q(x) \in \mathbb{F}_q[x]$ with $deg(p) \leq [\frac{m}{2}]$, $deg(q) \leq [\frac{m-3}{2}]$ such that $f(x)(P_i) = (p(x) + yq(x))(P'_i)$ for all $1 \leq i \leq 2r+t$ (N.B.: $p(x) + yq(x) = \lambda(f(x))$). Thus we have

$$f(x_i) = f(x)(P_{2i-1}) = (p(x) + yq(x))(P'_{2i-1}) = p(x'_i) + y'_i q(x'_i) \tag{7.4.2}$$

and

$$f(x_i) = f(x)(P_{2i}) = (p(x) + yq(x))(P'_{2i}) = g(x'_i) + (-y'_i - h(x'_i))q(x'_i). \tag{7.4.3}$$

Combining these two equalities, one finds that $q(x'_i) = 0$, $i = 1, 2, \ldots, r$. This implies that $q(x) = 0$, since $deg(q) \leq [\frac{m-3}{2}] < r$. It follows from (7.4.2), (7.4.3)) and $q(x) = 0$ that the following two rational Goppa codes are equal:

$$C(x_1 + x_2 + \ldots + x_{r+t}, [\frac{m}{2}]x_\infty) = C(x'_1 + x'_2 + \ldots + x'_{r+t}, [\frac{m}{2}]x_\infty),$$

where x_∞ corresponds to the place at infinity of $\mathbb{F}_q(x)$. Now Lemma 4.2.5 shows that there exists an element τ in the projective linear group $PGL(2, q)$ such that $\tau(x_i) = x'_i$ for $i = 1, 2, \ldots, r+t$. Hence

$$C(x'_1 + x'_2 + \ldots + x'_{r+t}, [\frac{m}{2}]x_\infty) = C(x_1 + x_2 + \ldots + x_{r+t}, [\frac{m}{2}]x_\infty) =$$
$$C(\tau(x_1) + \tau(x_2) + \ldots + \tau(x_{r+t}), \tau([\frac{m}{2}]x_\infty)) = C(x'_1 + x'_2 + \ldots + x'_{r+t}, [\frac{m}{2}]\tau(x_\infty)).$$

Now by Lemma 4.2.4 in conjunction with Lemma 4.2.2 we have that $\tau(x_\infty) = x_\infty$, i.e. τ is an affine linear transformation. Thus there are two elements $a, b \in \mathbb{F}_q$ such that $x'_i = ax_i + b$ for all $1 \leq i \leq r+t$.

Rephrasing the above in terms of the map $\lambda : \mathcal{L}(mQ_\infty) \to \mathcal{L}(mQ_\infty)$ as described in Definition 4.3.6 yields the following:

$$\lambda(x) = ax + b \quad \text{and}$$
$$\big((\lambda(x))_\infty\big) = 2Q_\infty,$$

i.e. $-v_{Q_\infty}(\lambda(x)) = 2$. Now we can apply Theorem 6.3.2 with $c = 2$ to see that λ is actually the restriction of an automorphism $\widetilde{\lambda}$ provided that $\deg D > \max\{m+2, 8, 6\}$. Now by Lemma 4.3.3 we conclude that $\widetilde{\lambda}(Q_\infty) = Q_\infty$ and thus by Lemma 4.3.10 we have $\widetilde{\lambda} \in Aut_{D,mQ_\infty}(F/\mathbb{F}_q)$. □

This chapter has illustrated why we had the rather esoteric condition that $-v_{Q_\infty}(\lambda(x)) \leq c$ in Theorem 6.3.2. In the above theorem we were able to show that $-v_{Q_\infty} = 2$ which then enabled us to prove the result for a bigger range of m because with $c = 2$ Theorem 6.3.2 gave us a lower bound for the number of places required in the support of D.

Obviously, in the above set-up we always have $-v_{Q_\infty}(\lambda(x)) \leq m$ but finding ways of reducing this bound means that the number of places needed in the support of the divisor D decreases. Therefore, it is highly desirable to obtain a bound for $-v_{Q_\infty}(\lambda(x))$ which is as low as possible because it means that we also have that $Aut_{D,mQ_\infty} \cong Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty))$ for some m in the range from $(\deg D)/2$ to $\deg D$.

Chapter 8

Codes Associated With The Klein Quartic

8.1 Introduction

In this chapter we show that the methods developed in the previous chapters can also be applied to Goppa codes associated with the Klein Quartic. We follow Stichtenoth's description in [Sti91, p.199].

8.2 The Function Field Associated with the Klein Quartic

Throughout this chapter we assume that $\operatorname{char} \mathbb{F}_q \neq 7$. Consider the function field $F = \mathbb{F}_q(z, y)$ defined by

$$z^3 + y^3 z + y = 0. \tag{8.2.1}$$

F is the function field of the Klein Quartic. After multiplying (8.2.1) by y^6 and setting $x = -y^2 z$, we can write $F = \mathbb{F}_q(x, y)$ where

$$y^7 = \frac{x^3}{1 - x}.$$

Because $\operatorname{char} \mathbb{F}_q \neq 7$, F is not rational and we have

- the genus of $F/\mathbb{F}_q$ is $g = 3$.
- $\mathbb{F}_q$ is the full constant field of F.

- exactly 3 places of $\mathbb{F}_q(x)$ ramify in $F/\mathbb{F}_q$, namely the pole P_∞ of x, the zero P_1 of x, and the zero P_2 of $x-1$. Denote the places lying above them by Q_∞, Q_1, and Q_2 respectively.
- Q_∞, Q_1, and Q_2 are all places of degree 1, i.e. their ramification index is 7.
- if $\mathbb{F}_q$ contains a seventh root of unity, ζ say, then $F/\mathbb{F}_q(x)$ is cyclic with Galois group isomorphic to the cyclic group generated by ζ.
- $\alpha = \begin{cases} x \mapsto \frac{x-1}{x} \\ y \mapsto \frac{-x}{y^3} \end{cases}$ is an automorphism of order 3 of $F/\mathbb{F}_q$ which permutes Q_∞, Q_1, Q_2 cyclically, i.e. $\alpha(\alpha(Q_\infty)) = \alpha(Q_1) = Q_2$.

The above notation will be used throughout the rest of this chapter.

We consider a certain subclass of Goppa codes associated with the Klein Quartic. The codes we are interested in initially are those whose divisors G are given by $G = k_0Q_\infty + k_1Q_1 + k_2Q_2 > 0$ with $k_i \in \mathbb{N}_0 \setminus \{1,2,3,4\}$, $i = 0,1,2$ (note $\mathbb{N}_0 = \{0,1,2,\dots\}$). Eventually we will deal with codes whose divisors $0 < G = k_0Q_\infty + k_1Q_1 + k_2Q_2$ are such that $\deg G \geq 5$ (by the Riemann-Roch Theorem it follows that $\dim G = \deg G - 2$).

8.3 The Easy Case

First we determine a basis for $\mathcal{L}(G)$ where $G = k_0Q_\infty + k_1Q_1 + k_2Q_2 > 0$ with $k_i \in \mathbb{N}_0 \setminus \{1,2,3,4\}$, $i = 0,1,2$. This will then allow us to determine the automorphism group of a certain class of Goppa codes associated with those divisors G. Later on, in Section 8.4 and Section 8.5, we will deal with divisors G, where $\deg G \geq 5$ and $supp(G) \subseteq \{Q_\infty, Q_1, Q_2\}$.

Lemma 8.3.1. *If $G = k_0Q_\infty + k_1Q_1 + k_2Q_2 > 0$ with $k_i \in \mathbb{N}_0 \setminus \{1,2,3,4\}$ then $\mathcal{L}(G)$ has a basis consisting of powers and products of the following elements:*

$$w_1 = \frac{x}{y^2} \qquad w_2 = \frac{x}{y} \qquad w_3 = x$$
$$z_1 = \alpha(w_1) = \frac{-1}{y} \qquad z_2 = \alpha(w_2) = \frac{x}{y^4} \qquad z_3 = \alpha(w_3) = \frac{x-1}{x}$$
$$v_1 = \alpha^2(w_1) = \frac{y^3}{x} \qquad v_2 = \alpha^2(w2) = \frac{-y^5}{x^2} \qquad v_3 = \alpha^2(w_3) = \frac{1}{1-x}$$

Proof. It is easy to see that

$$(x) = 7Q_1 - 7Q_\infty$$
$$(y) = 3Q_1 - 2Q_\infty - Q_2$$
$$(1-x) = 7Q_2 - 7Q_\infty$$

and hence

$$(w_1) = 2Q_2 + Q_1 - 3Q_\infty \qquad (w_2) = Q_2 + 4Q_1 - 5Q_\infty \qquad (w_3) = 7Q_1 - 7Q_\infty.$$

The divisors of v_i and z_i can be worked out immediately by using the fact that α permutes Q_∞, Q_1, Q_2 cyclically. We also note the following:

$$\begin{aligned}(w_1v_1) &= (y) &&= 3Q_1 - 2Q_\infty - Q_2\\ (w_1z_1) &= (\frac{-x}{y^3}) &&= 3Q_2 - 2Q_1 - Q_\infty\\ (v_1z_1) &= (\frac{-y^2}{x}) &&= 3Q_\infty - 2Q_2 - Q_1\end{aligned}$$

Because $\alpha(\mathcal{L}(G)) = \mathcal{L}(\alpha(G))$, we can assume , without loss of generality, k_0 is the maximum of the $k_i (i = 0, 1, 2)$. We distinguish 4 cases:
$k_1 = k_2 = 0,\ k_0 \geq 5$:

$$\mathcal{L}(k_0Q_\infty) = \langle w_1^{\beta_1} w_2^{\beta_2} w_3^{\beta_3} | \beta_i \geq 0\ , 3\beta_1 + 5\beta_2 + 7\beta_3 \leq k_0 \rangle$$

$k_2 = 0,\ k_0 \geq k_1 \geq 5$:

$$\begin{aligned}\mathcal{L}(k_0Q_\infty + k_1Q_1) = \langle & w_1^{\beta_1} w_2^{\beta_2} w_3^{\beta_3}\ | \beta_i \geq 0,\ 3\beta_1 + 5\beta_2 + 7\beta_3 \leq k_0,\\ & z_1^{\gamma_1} z_2^{\gamma_2} z_3^{\gamma_3} \quad | \gamma_i \geq 0,\ 3\gamma_1 + 5\gamma_2 + 7\gamma_3 \leq k_1,\\ & w_1z_1,\ (w_1z_1)^2,\ w_1(w_1z_1)\rangle\end{aligned}$$

$k_1 = 0,\ k_0 \geq k_2 \geq 5$:

$$\begin{aligned}\mathcal{L}(k_0Q_\infty + k_2Q_2) = \langle & w_1^{\beta_1} w_2^{\beta_2} w_3^{\beta_3}\ | \beta_i \geq 0,\ 3\beta_1 + 5\beta_2 + 7\beta_3 \leq k_0,\\ & v_1^{\gamma_1} v_2^{\gamma_2} v_3^{\gamma_3} \quad | \gamma_i \geq 0,\ 3\gamma_1 + 5\gamma_2 + 7\gamma_3 \leq k_2,\\ & w_1v_1,\ (w_1v_1)^2,\ v_1(w_1v_1)\rangle\end{aligned}$$

$k_0 \geq k_i \geq 5$ for $i = 1, 2$:

$$\begin{aligned}\mathcal{L}(k_0Q_\infty + k_1Q_1 + k_2Q_2) = \langle & w_1^{\beta_1} w_2^{\beta_2} w_3^{\beta_3}\ |\ \beta_i \geq 0,\ 3\beta_1 + 5\beta_2 + 7\beta_3 \leq k_0,\\ & v_1^{\gamma_1} v_2^{\gamma_2} v_3^{\gamma_3} \quad |\ \gamma_i \geq 0,\ 3\gamma_1 + 5\gamma_2 + 7\gamma_3 \leq k_1,\\ & z_1^{\delta_1} z_2^{\delta_2} z_3^{\delta_3} \quad |\ \delta_i \geq 0,\ 3\delta_1 + 5\delta_2 + 7\delta_3 \leq k_2,\\ & w_1z_1,\ (w_1z_1)^2,\ v_1z_1,\ (v_1z_1)^2,\ w_1v_1, (w_1v_1)^2\rangle\end{aligned}$$

Because of the way α acts we can always assume that we are in one of the above cases. It is now an easy but tedious exercise to show that the elements listed above do span the various $\mathcal{L}(G)$.

In general they do not form a basis of the $\mathcal{L}(G)$ because there are different elements whose pole divisors are the same, i.e. $(w_1^5)_\infty = 15Q_\infty = (w_2^3)_\infty$ but $w_1^5 \neq w_2^3$. To obtain a basis we just collect all those elements in $\mathcal{L}(G)$ which do have different pole divisors. □

For completeness we also note that

$$w_3 = w_2^2 w_1^{-1} \qquad v_3 = v_2^2 v_1^{-1} \qquad z_3 = z_2^2 z_1^{-1}.$$

This will be used in the proof of the next theorem to show that the λ acts multiplicatively on the spaces $\mathcal{L}(G)$ whose bases we determined in the previous lemma.

Remark. The elements w_i, v_i, z_i which were introduced in the previous lemma will be used throughout the rest of this chapter.

With the above description of $\mathcal{L}(G)$ and the methods used in the previous chapters it is now fairly straightforward to prove the following result.

Theorem 8.3.2. *Let $F/\mathbb{F}_q$ be the function field of the Klein Quartic. Let $G = k_0Q_\infty + k_1Q_1 + k_2Q_2 > 0$ with $k_i \in \mathbb{N}_0 \setminus \{1,2,3,4\}$, $i = 0,1,2$. Put $D = \sum_{P\in J} P$, where $J \subset \mathbb{P}_F^{(1)} \setminus \{Q_\infty, Q_1, Q_2\}$. Suppose that $n = \deg D > \max\{25, 2\deg G\}$; then*

$$Aut_{D,G}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D,G)).$$

Proof. As the genus of F is $g = 3$ and $n > 25$, we know from Proposition 3.3.4 that $Aut_{D,G}(F/\mathbb{F}_q)$ can be regarded as a subgroup of $Aut(\mathcal{C}_\mathcal{L}(D,G))$. We need to show that this embedding is surjective.

Recall that $\mathcal{C}_\mathcal{L}(D,G) = \mathcal{C}_\mathcal{L}(\alpha(D),\alpha(G))$ (by Lemma 4.3.2) and hence

$$Aut_{D,G}(F/\mathbb{F}_q) = \alpha^{-1} Aut_{\alpha(D),\alpha(G)}(F/\mathbb{F}_q)\alpha.$$

Because α cycles Q_∞, Q_1, Q_2 around, without loss of generality, we can assume that $k_0 = \max\{k_0,k_1,k_2\}$. As before we need to show that each λ which is associated with an automorphism of the code agrees with the restriction to $\mathcal{L}(G)$ of an automorphism of the function field. So suppose that $\pi \in Aut(\mathcal{C}_\mathcal{L}(D,G))$ and denote by λ the associated map from $\mathcal{L}(G)$ onto itself. We will sketch the proofs for the three different cases.

Case 1: $G = k_0Q_\infty$:

We observe that $F = \mathbb{F}_q(w_1, w_2)$ and as before, by Lemma 4.3.8, Lemma 4.3.9 and the linearity of λ, it is a trivial exercise to show that the linear map λ is also multiplicative, in a restricted sense, on $\mathcal{L}(k_0Q_\infty)$, i.e if $f, g \in \mathcal{L}(k_0Q_\infty)$ such that $fg \in \mathcal{L}(k_0Q_\infty)$ then $\lambda(fg) = \lambda(f)\lambda(g)$. However,

because $w_3 = w_2^2 w_1^{-1}$ we also have to show that $\lambda(w_3) = \lambda(w_2)^2\lambda(w_1)^{-1}$. Now $w_3 w_1 = w_2 w_2$ might not be in $\mathcal{L}(k_0 Q_\infty)$ but we certainly have that $w_3 w_1 = w_2 w_2 \in \mathcal{L}(2k_0 Q_\infty)$ and hence it makes sense to consider $f_1 = \lambda(w_3)\lambda(w_1)$ and $f_2 = \lambda(w_2)^2$ both of which are also in $\mathcal{L}(2k_0 Q_\infty)$. Because f_1 and f_2 agree at $2k_0 + 1$ places we can deduce by Lemma 4.3.5 that $\lambda(w_3)\lambda(w_1) = f_1 = f_2 = \lambda(w_2)^2$ and thus $\lambda(w_3) = \lambda(w_2)^2\lambda(w_1)^{-1}$.

It remains to show that λ also preserves the defining equation of $F = \mathbb{F}_q(w_1, w_2)$ which is easily seen to be given by

$$w_2 w_1 - w_2^3 = w_1^5. \tag{8.3.3}$$

Now when $k_0 = 5$ we know that $\deg\big(\lambda(w_1)\big)_\infty \leq 5$ and $\deg\big(\lambda(w_2)\big)_\infty \leq 5$ and thus in order to deduce that (8.3.3) is preserved by λ, we need at least 26 places (because $(\lambda(w_2)\lambda(w_1) - \lambda(w_2)^3)_\infty \leq 15Q_\infty$ and $(\lambda(w_1)^5)_\infty \leq 25Q_\infty$, so for Lemma 4.3.5 to apply we need these two functions to agree at a minimum of 26 places). The remaining cases are proved in a similar fashion using the fact that as soon as $k_0 > 5$ we have an improved estimate for $v_{Q_\infty}(\lambda(w_1))$ (since $\lambda(w_1^2) = \lambda(w_1)^2 \in \mathcal{L}(k_0 Q_\infty)$) and if $k_0 > 7$ we have also an improved estimate for $v_{Q_\infty}(\lambda(w_2))$ (since $\lambda(w_1 w_2) = \lambda(w_1)\lambda(w_2) \in \mathcal{L}(G)$).

Since λ preserves (8.3.3), we know, by Theorem 4.3.1, that $\lambda(w_1)$ and $\lambda(w_2)$ determine an endomorphism, $\widetilde{\lambda}$, of the function field. Obviously, $\widetilde{\lambda}$ agrees with λ on $\mathcal{L}(G)$ and hence w_1 and w_2 are in the image of $\widetilde{\lambda}$. Thus it is actually an automorphism of the function field. Moreover, by Lemma 4.3.4 and Lemma 4.3.10, it is in $Aut_{D,G}(F/\mathbb{F}_q)$.

Case 2: $G = k_0 Q_\infty + k_1 Q_1$:

As before it is straightforward to check that λ acts also multiplicatively, in a restricted sense, on $\mathcal{L}(G)$. Furthermore it is also easy to see that $F = \mathbb{F}_q(w_1, z_1)$. Now we express w_2, z_2 in terms of w_1, z_1 and check whether λ preserves these expressions. Note that then w_3 and z_3 can be dealt with in exactly the same way as in the case $\mathcal{L}(k_0 Q_\infty)$. We observe that

$$w_2 z_1 = -w_1 \qquad\qquad z_2 = (w_1 z_1) z_1$$

and using Lemma 4.3.8 we see immediately that λ preserves these identities and thus acts multiplicative on the whole of $\mathcal{L}(k_0 Q_\infty + k_1 Q_1)$. Now we also observe that the defining equation of $F = \mathbb{F}_q(w_1, z_1)$ is given by

$$(w_1 z_1)^3 + z_1^2 - w_1 = 0. \tag{8.3.4}$$

Because $(w_1 z_1)^2 = w_1^2 z_1^2$ is in $\mathcal{L}(k_0 Q_\infty + k_1 Q_1)$ we know that $\deg(\lambda(w_1)\lambda(z_1)) \leq \left[\frac{k_0 + k_1}{2}\right]$ and thus (8.3.4) will be preserved by λ as long as $\deg D \geq 2(k_0 + k_1) + 1$ using Lemma 4.3.5. So as before,

because w_1, z_1 are in $\mathcal{L}(G)$, λ determines an automorphism, $\widetilde{\lambda}$, of the function field. Now we can apply Lemma 4.3.4 (note that the conditions on k_i imply that $\deg G \geq 10 > 8 = 2g + 1 + 1$) to deduce that $\widetilde{\lambda}(G) = G$. Whence, by Lemma 4.3.10, it follows that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$. The case $\mathcal{L}(k_0Q_\infty + k_2Q_2)$ is completely analogous.

Case 3: $G = k_0Q_\infty + k_1Q_1 + k_2Q_2$:
Again we use w_1, z_1 as the generators of F. We observe that

$$v_1(w_1z_1) = -1 \qquad\qquad v_2(w_1^2z_1) = 1.$$

Thus by applying Lemma 4.3.8 twice we have

$$\lambda(v_1(w_1z_1)) = \lambda(v_1)\lambda(w_1z_1) = \lambda(v_1)\lambda(w_1)\lambda(z_1) = -1$$

and similarly

$$\lambda(v_2(w_1^2z_1)) = \lambda(v_2)\lambda(w_1)^2\lambda(z_1) = 1.$$

Rearranging we see that

$$\lambda(v_1) = -\frac{1}{\lambda(w_1)\lambda(z_1)} \qquad\qquad \lambda(v_2) = \frac{1}{\lambda(w_1)^2\lambda(z_1)}$$

by applying Lemma 4.3.8 several times. Note that w_3, z_3 and v_3 are dealt with as before. As in the previous case the defining equation is given by (8.3.4) and λ preserves it for the same reasons as given above. So as before, because w_1 and z_1 are in $\mathcal{L}(k_0Q_\infty)$, λ determines an automorphism, $\widetilde{\lambda}$, of the function field. Now by Lemma 4.3.3 and Lemma 4.3.10 we see that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$. □

8.4 The General Case

Using the same techniques and notation of the previous sections we can even prove the following, more general result.

Theorem 8.4.1. *Let $F/\mathbb{F}_q$ be the function field of the Klein quartic. Let $0 < G = k_0Q_\infty + k_1Q_1 + k_2Q_2$ with $k_i \geq 0$ $(i = 0, 1, 2)$ and $\deg G \geq 6$. Let $D = \sum_{P\in J} P$ where $J \subset \mathbb{P}_F^{(1)} \setminus \{Q_\infty, Q_1, Q_2\}$. If $|J| = \deg D \geq \max\{40, 2\deg G + 1\}$ then*

$$Aut_{D,G}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D, G)).$$

Proof. The proof is rather tedious. It involves looking at each individual case determining a basis for $\mathcal{L}(G)$, the defining equation for the function field and the number of places required to make the

method work. We shall prove a few cases in detail and then just provide tables for the remaining ones and leave the reader to check the details.

N.B.: The elements which will be used to describe a basis for the individual $\mathcal{L}(G)$ are the ones that were introduced in Lemma 8.3.1, i.e

$$(w_1) = (\frac{x}{y^2}) = 2Q_2 + Q_1 - 3Q_\infty \quad (w_2) = (\frac{x}{y}) = Q_2 + 4Q_1 - 5Q_\infty \qquad (w_3) = (x) = 7Q_1 - 7Q_\infty$$
$$(v_1) = (\frac{y^3}{x}) = 2Q_1 + Q_\infty - 3Q_2 \quad (v_2) = (\frac{-y^5}{x^2}) = Q_1 + 4Q_\infty - 5Q_2 \quad (v_3) = (\frac{1}{1-x}) = 7Q_\infty - 7Q_2$$
$$(z_1) = (\frac{-1}{y}) = 2Q_\infty + Q_2 - 3Q_1 \quad (z_2) = (\frac{x}{y^4}) = Q_\infty + 4Q_2 - 5Q_1 \qquad (z_3) = (\frac{x-1}{x}) = 7Q_2 - 7Q_1,$$

where $v_i = \alpha(z_i) = \alpha(\alpha(w_i))$ for $i = 1, 2, 3$.

As before we can regard $Aut_{D,G}(F/\mathbb{F}_q)$ as a subgroup of $Aut(\mathcal{C}_\mathcal{L}(D, G))$ by Proposition 3.3.4. We are going to show that it constitutes the whole group. Again because of the action of α on Q_∞, Q_1, and Q_2 we can assume that $k_0 = \max\{k_0, k_1, k_2\}$. Let $\pi \in Aut(\mathcal{C}_\mathcal{L}(D, G))$ and denote by λ the map associated with it (cf. Definition 4.3.6). We have already dealt with the case $k_1 = k_2 = 0$ and $k_0 \geq 5$. Thus the next case is $1 \leq k_1 \leq 4$, $6 \leq k_0 + k_1 \leq 9$ and $k_2 = 0$.

Case $G = 5Q_\infty + Q_1$:

It is a trivial exercise to show that $\mathcal{L}(5Q_\infty + Q_1) = \langle 1, w_1, w_2, w_1^2 z_1 \rangle$. We now note that $w_1 = x/y^2$ and $w_2 = x/y$ are generating elements of F, i.e. $F = \mathbb{F}_q(w_1, w_2) = \mathbb{F}_q(x, y)$. F is defined by the following equation:

$$w_1^5 = w_2 w_1 - w_2^3. \tag{8.4.2}$$

Next we have to show that $w_1^2 z_1$ can be expressed in terms of w_1 and w_2 and that this expression is preserved under the map λ.

It is obvious that $\deg((\lambda(w_1))_\infty) \leq 6$, $\deg((\lambda(w_2))_\infty) \leq 6$ and $\deg(\lambda(w_1^2 z_1))_\infty \leq 6$. A straightforward calculation shows that $w_1^2 z_1 = w_1^3 w_2^{-1}$. Thus looking at $(w_1^2 z_1) w_2 = w_1^3$ and using Lemma 4.3.5 together with the fact that $\deg D \geq 40$, we find that $\lambda(w_1^2 z_1)\lambda(w_2) = \lambda(w_1)^3$ and hence $\lambda(w_1^2 z_1) = \lambda(w_1)^3 \lambda(w_2)^{-1}$.

In order to apply Lemma 4.3.5 to deduce that

$$\lambda(w_1)^5 = \lambda(w_1)\lambda(w_2) - \lambda(w_2)^3$$

we need at least 31 places in the support of D.

This now shows that $\lambda(w_1)$ and $\lambda(w_2)$ determine an automorphism, $\widetilde{\lambda}$, of the function field F (because of Theorem 4.3.1 and the fact that w_1 and w_2 are in the image of $\widetilde{\lambda}$). We now apply Lemma 4.3.4 and Lemma 4.3.10 to deduce that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$.

Using the same methods for divisors $G = k_0 Q_\infty + Q_1$, we can summarise the results in a table. The first column describes the divisor $G = k_0 Q_\infty + k_1 Q_1 + k_2 Q_2$. The second column lists all the basis elements of $\mathcal{L}(G)$. Going down one row in the table increase the dimension of $\mathcal{L}(G)$ by one, thus it is sufficient to list only the element which is added to the previous set of basis elements. The third column gives the two generators of F which lie in $\mathcal{L}(G)$, e.g. $F = \mathbb{F}_q(w_1, w_2)$ in the above example. In the fifth column we write $(\lambda(x), \leq n)$ instead of $\deg((\lambda(x))_\infty) \leq n$, i.e. we give an upper bound for the degree of the pole divisor of the image of our chosen generators under the map λ. These estimates are then used to work out the minimum number of places required in the support of D (sixth column) to ensure that the defining equation of our function field (fourth column) is preserved by λ and that all basis elements can be expressed in terms of the chosen generators.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(5, 1, 0)$	$1, w_1, w_2,$ $w_1^2 z_1$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_2), \leq 6)$	31
$(6, 1, 0)$	$\ldots, w_1^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_2), \leq 7)$	22
$(7, 1, 0)$	$\ldots, w_3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_2), \leq 8)$	25
$(8, 1, 0)$	$\ldots, w_1 w_2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 9)$	28
$(9, 1, 0)$	$\ldots, w_1^3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_2), \leq 10)$	31
$(> 9, 1, 0)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 7)$	$2 \deg G + 1$

To complete the proof we must look at the remaining divisors G. Consequently, we have to provide several more tables similar to the one above. The format of them remains the same throughout. Occasionally, we may need a different defining equation for the function field in order to reduce the number of places required in the support of D to achieve the desired result. In those cases we go through an example that requires the use of that defining equation before we use it in our table.

Case $G = 4Q_\infty + 2Q_1$:

It is easy to see that $\mathcal{L}(4Q_\infty + 2Q_1) = \langle 1, w_1, w_1 z_1, w_1^2 z_1 \rangle$. We now note that $w_1 = x/y^2$ and $w_1^2 z_1 = -x^2/y^5$ are generating elements of F, i.e. $F = \mathbb{F}_q(w_1, w_1^2 z_1) = \mathbb{F}_q(x, y)$. F is defined by

the following equation:

$$w_1^5 = (w_1^2 z_1)^2 - w_1 (w_1^2 z_1)^3. \tag{8.4.3}$$

Next we have to show that $w_1 z_1$ can be expressed in terms of w_1 and $w_1^2 z_1$ (this is obvious) and that this expression is preserved under the map λ.

It is obvious that $\deg((\lambda(w_1))_\infty) \leq 6$, $\deg((\lambda(w_1 z_1))_\infty) \leq 6$ and $\deg(\lambda(w_1^2 z_1))_\infty \leq 6$. We have $w_1 z_1 = (w_1^2 z_1) w_1^{-1}$. Thus looking at $(w_1 z_1) w_1 = w_1^2 z_1$ and using Lemma 4.3.5 together with the fact that $\deg D \geq 40$, we find that $\lambda(w_1 z_1)\lambda(w_1) = \lambda(w_1^2 z_1)$ and hence $\lambda(w_1 z_1) = \lambda(w_1^2 z_1)\lambda(w_1)^{-1}$.

In order to apply Lemma 4.3.5 to deduce that

$$\lambda(w_1)^5 = \lambda(w_1^2 z_1)^2 - \lambda(w_1)\lambda(w_1^2 z_1)^3$$

we need at least 31 places in the support of D.

This now shows that $\lambda(w_1)$ and $\lambda(w_1^2 z_1)$ determine an automorphism, $\widetilde{\lambda}$, of the function field F (because of Theorem 4.3.1 and the fact that w_1 and $w_1^2 z_1$ are in the image of $\widetilde{\lambda}$). We now apply Lemma 4.3.4 and Lemma 4.3.10 to deduce that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$.

This leads to the following table:

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4, 2, 0)$	$1, w_1 z_1, w_1, w_1^2 z_1$	$w_1, w_1^2 z_1$	(8.4.3)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1^2 z_1), \leq 6)$	31
$(5, 2, 0)$	$\ldots, w_2$	$w_1, w_1^2 z_1$	(8.4.3)	$(\lambda(w_1), \leq 7)$ $(\lambda(w_1^2 z_1), \leq 7)$	36
$(6, 2, 0)$	$\ldots, w_1^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	25
$(7, 2, 0)$	$\ldots, w_3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 9)$	28
$(8, 2, 0)$	$\ldots, w_1 w_2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 10)$	31
$(9, 2, 0)$	$\ldots, w_1^3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_2), \leq 11)$	34
$(> 9, 2, 0)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	$2 \deg G + 1$

Note that after the first two cases we use a different set of generators and a different equation in order to keep the number of places in the support of D low.

In order to produce another table we need to look at another special case first.

Case $G = 3Q_\infty + 3Q_1$:

It is a trivial exercise to show that $\mathcal{L}(3Q_\infty + 3Q_1) = \langle 1, w_1, z_1, w_1z_1\rangle$. We now note that $w_1 = x/y^2$ and $z_1 = -1/y$ are generating elements of F, i.e. $F = \mathbb{F}_q(w_1, z_1) = \mathbb{F}_q(x, y)$. F is defined by the following equation:

$$(w_1z_1)^3 + z_1^2 - w_1 = 0. \tag{8.4.4}$$

It is obvious that $\deg((\lambda(w_1))_\infty) \leq 6$, $\deg((\lambda(z_1))_\infty) \leq 6$ and $\deg(\lambda(w_1z_1))_\infty \leq 6$. By Lemma 4.3.8 one sees immediately that $\lambda(w_1z_1) = \lambda(w_1)\lambda(z_1)$.

In order to apply Lemma 4.3.5 to deduce that

$$(\lambda(w_1)\lambda(z_1))^3 + \lambda(z_1)^2 - \lambda(w_1) = 0$$

we need at least 19 places in the support of D. (We only need 19 places because $\lambda(w_1z_1)^3 = (\lambda(w_1)\lambda(z_1))^3$ and $\deg(\lambda(w_1z_1)^3)_\infty \leq 18$.)

This now shows that $\lambda(w_1)$ and $\lambda(z_1)$ determine an automorphism, $\widetilde{\lambda}$, of the function field F. We now apply Lemma 4.3.4 and Lemma 4.3.10 to deduce that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$.

We obtain the following two tables:

The next table caters for divisors G, where $G = k_0Q_\infty + 3Q_1$ and $k_0 \geq 3$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3, 3, 0)$	$1, z_1, w_1z_1$ w_1	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 6)$ $(\lambda(z_1), \leq 6)$	19
$(4, 3, 0)$	$\ldots, w_1^2z_1$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 7)$ $(\lambda(z_1), \leq 7)$	22
$(5, 3, 0)$	$\ldots, w_2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 8)$ $(\lambda(z_1), \leq 8)$	25
$(6, 3, 0)$	$\ldots, w_1^2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 4)$ $(\lambda(z_1), \leq 9)$	28
$(7, 3, 0)$	$\ldots, w_3$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 5)$ $(\lambda(z_1), \leq 10)$	31
$(8, 3, 0)$	$\ldots, w_1w_2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 11)$	34
$(9, 3, 0)$	$\ldots, w_1^3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 12)$	37
$(> 9, 3, 0)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	$2\deg G + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 4Q_1$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4,4,0)$	$1, z_1, w_1z_1$ $(w_1z_1)^2, w_1, w_1^2z_1$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 8)$ $(\lambda(z_1), \leq 8)$	17
$(5,4,0)$	$\ldots, w_2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 9)$ $(\lambda(z_1), \leq 9)$	19
$(6,4,0)$	$\ldots, w_1^2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 5)$ $(\lambda(z_1), \leq 10)$	21
$(7,4,0)$	$\ldots, w_3$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 5)$ $(\lambda(z_1), \leq 11)$	23
$(8,4,0)$	$\ldots, w_1w_2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 6)$ $(\lambda(z_1), \leq 12)$	25
$(9,4,0)$	$\ldots, w_1^3$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 4)$ $(\lambda(z_1), \leq 13)$	27
$(10,4,0)$	$\ldots, w_2^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 7)$	22
$(>10,4,0)$	$\ldots, w_1^2w_2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 9)$	$2\deg G + 1$

It is worth noting that in the first few cases we use the fact that $(w_1z_1)^2 \in \mathcal{L}(G)$ to deduce that $\deg(\lambda(w_1)\lambda(z_1))_\infty \leq \deg G/2$ which helps to reduce the number of places needed.

Now we will look at the case where $G = k_0Q_\infty + k_2Q_2$, with $k_0 \geq k_2 > 0$ and $k_0 + k_2 \geq 6$.

The reason why we have to look at these divisors separately is the following lemma.

Lemma 8.4.5. *There is no automorphism of the function field associated with the Klein Quartic that preserves Q_∞ and swaps over Q_1 and Q_2.*

Proof. Suppose β was such an automorphism then because $(y) = 3Q_1 - 2Q_\infty - Q_2$ we would have

$$(\beta(y)) = 3Q_2 - 2Q_\infty - Q_1. \tag{8.4.6}$$

We are going to show that there is no such function as $\beta(y)$.

We observe that for $G = 4Q_\infty + Q_1$ we have that $\mathcal{L}(G) =< 1, w_1, w_1^2z_1 >$. We note that

$$(1) = 0$$
$$(w_1) = 2Q_2 + Q_1 - 3Q_\infty$$
$$(w_1^2z_1) = 5Q_2 - Q_1 - 4Q_\infty.$$

These three elements actually form a basis as can be seen from the fact that their pole divisors are all different. Now by (8.4.6) it follows that $\beta(y) \in \mathcal{L}(G)$. However, by the Strict Triangle Inequality (cf. Lemma 2.2.11), $\beta(y)$ cannot be expressed as a linear combination of $1, w_1, w_1^2 z_1$ which is a contradiction. Hence there is no function $\beta(y)$ and thus no automorphism β. □

Again, before we can summarise the results in the form of tables, we need to look at a special case first. This special case is not directly related to the sort of divisors G we are interested in. However, all we need from it is the defining equation of the function field which is given in terms of w_1 and $w_1 v_1$ both of which are in $\mathcal{L}(3Q_\infty + Q_2)$.

Case $G = 3Q_\infty + 2Q_1 + Q_2$:

It is a trivial exercise to show that $\mathcal{L}(3Q_\infty + 2Q_1 + Q_2) = \langle 1, w_1, , w_1 z_1, w_1 v_1 \rangle$. We now note that $w_1 = x/y^2$ and $w_1 v_1 = y$ are generating elements of F, i.e. $F = \mathbb{F}_q(w_1, w_1 v_1) = \mathbb{F}_q(x, y)$. F is defined by the following equation:

$$w_1^3 + w_1 (w_1 v_1)^3 - (w_1 v_1) = 0. \tag{8.4.7}$$

We leave it to the reader to show that $w_1 z_1 = x/y^3 = w_1 (w_1 v_1)^{-1}$ and that this expression is preserved under the map λ.

Obviously $\deg((\lambda(w_1))_\infty) \leq 6$, $\deg((\lambda(w_1 v_1))_\infty) \leq 6$ and $\deg(\lambda(w_1 z_1))_\infty \leq 6$. In order to apply Lemma 4.3.5 to deduce that

$$\lambda(w_1)^3 + \lambda(w_1)\lambda(w_1 v_1)^3 - \lambda(w_1 v_1) = 0$$

we need at least 25 places in the support of D. This now shows that $\lambda(w_1)$ and $\lambda(w_1 v_1)$ determine an automorphism, $\widetilde{\lambda}$, of the function field F. We now apply Lemma 4.3.4 and Lemma 4.3.10 to deduce that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$.

We now obtain the following tables:

The first one deals with divisors $G = k_0 Q_\infty + Q_2$, $k_0 \geq 5$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(5, 0, 1)$	$1, w_1v_1, w_1$ w_2	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(6, 0, 1)$	$\ldots, w_1^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_1v_1), \leq 7)$	22
$(7, 0, 1)$	$\ldots, w_3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	25
$(8, 0, 1)$	$\ldots, w_1w_2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 9)$	28
$(9, 0, 1)$	$\ldots, w_1^3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_2), \leq 10)$	31
$(> 9, 0, 1)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 7)$	$2 \deg G + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 2Q_2$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4, 0, 2)$	$1, w_1v_1, w_1,$ $(w_1v_1)^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 3)$	19
$(5, 0, 2)$	$\ldots, w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 7)$ $(\lambda(w_1v_1), \leq 3)$	22
$(6, 0, 2)$	$\ldots, w_1^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_1v_1), \leq 4)$	17
$(7, 0, 2)$	$\ldots, w_3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_1v_1), \leq 4)$	17
$(8, 0, 2)$	$\ldots, w_1w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_1v_1), \leq 5)$	21
$(9, 0, 2)$	$\ldots, w_1^3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_1v_1), \leq 5)$	19
$(> 9, 0, 2)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	$2 \deg G + 1$

Before we proceed with the next table we need to look at yet another special case.

Case $G = 3Q_\infty + 3Q_2$:

It is straightforward to show that $\mathcal{L}(3Q_\infty + 3Q_2) = \langle 1, v_1, w_1, , w_1v_1\rangle$. We now note that $w_1 = x/y^2$ and $v_1 = y^3/x$ are generating elements of F, i.e. $F = \mathbb{F}_q(w_1, v_1) = \mathbb{F}_q(x, y)$. F is defined by the following equation:

$$(w_1v_1)^3 + w_1^2 - v_1 = 0. \tag{8.4.8}$$

It is obvious that $\deg((\lambda(w_1))_\infty) \leq 6$, $\deg((\lambda(w_1v_1))_\infty) \leq 6$ and $\deg(\lambda(v_1))_\infty \leq 6$. A straightforward calculation using Lemma 4.3.8 shows that $\lambda(w_1v_1) = \lambda(w_1)\lambda(v_1)$.

In order to apply Lemma 4.3.5 to deduce that

$$(\lambda(w_1)\lambda(v_1))^3 + \lambda(w_1)^2 - \lambda(v_1) = 0$$

we need at least 19 places in the support of D (we use the fact that $(\lambda(w_1v_1))^3 = (\lambda(w_1)\lambda(v_1))^3$). This now shows that $\lambda(w_1)$ and $\lambda(v_1)$ determine an automorphism, $\widetilde{\lambda}$, of the function field F. We now apply Lemma 4.3.4 and Lemma 4.3.10 to deduce that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$.

We have now reached a stage where we can present a huge number of tables dealing with almost all the remaining cases.

The next table caters for divisors G, where $G = k_0Q_\infty + 3Q_2$ and $k_0 \geq 3$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3, 0, 3)$	$1, v_1, w_1v_1$ w_1	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 6)$ $(\lambda(v_1), \leq 6)$	19
$(4, 0, 3)$	$\ldots, (w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 7)$ $(\lambda(v_1), \leq 7)$	15
$(5, 0, 3)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 8)$ $(\lambda(v_1), \leq 8)$	17
$(6, 0, 3)$	$\ldots, w_1^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 4)$ $(\lambda(v_1), \leq 9)$	13
$(7, 0, 3)$	$\ldots, w_3$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 5)$ $(\lambda(v_1), \leq 10)$	16
$(8, 0, 3)$	$\ldots, w_1w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 5)$ $(\lambda(v_1), \leq 11)$	16
$(9, 0, 3)$	$\ldots, w_1^3$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 4)$ $(\lambda(v_1), \leq 12)$	19
$(> 9, 0, 3)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	$2 \deg G + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 4Q_2$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4, 0, 4)$	$1, v_1, w_1v_1^2$ $w_1v_1, w_1, (w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 8)$ $(\lambda(v_1), \leq 8)$	17
$(5, 0, 4)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 9)$ $(\lambda(v_1), \leq 9)$	19
$(6, 0, 4)$	$\ldots, w_1^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 5)$ $(\lambda(v_1), \leq 10)$	16
$(7, 0, 4)$	$\ldots, w_3$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 5)$ $(\lambda(v_1), \leq 11)$	16
$(8, 0, 4)$	$\ldots, w_1w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 6)$ $(\lambda(v_1), \leq 12)$	19
$(9, 0, 4)$	$\ldots, w_1^3$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 6)$ $(\lambda(v_1), \leq 13)$	19
$(10, 0, 4)$	$\ldots, w_2^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 7)$	22
$(> 10, 0, 4)$	$\ldots, w_1^2w_2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 9)$	$2 \deg G + 1$

Divisors of the form $G = k_0Q_\infty + k_iQ_i$ where $i = 1, 2$ and $k_0 \geq k_i \geq 5$ have been dealt with in Theorem 8.3.2. So now we will look at divisors G with $\text{supp}(G) = \{Q_\infty, Q_1, Q_2\}$.

The next table deals with divisors G of the form $G = k_0Q_\infty + Q_\infty + Q_2$, where $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4,1,1)$	$1, w_1v_1, w_1,$ $w_1^2z_1$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(5,1,1)$	$\ldots, w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 7)$ $(\lambda(w_1v_1), \leq 7)$	29
$(6,1,1)$	$\ldots, w_1^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_2), \leq 8)$	25
$(7,1,1)$	$\ldots, w_3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 9)$	28
$(8,1,1)$	$\ldots, w_1w_2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 10)$	31
$(9,1,1)$	$\ldots, w_1^3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 3)$ $(\lambda(w_2), \leq 11)$	34
$(> 9,1,1)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	$2\deg G + 1$

The next table deals with divisors G, where $G = k_0Q_\infty + 2Q_1 + Q_2$ and $k_0 \geq 3$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3,2,1)$	$1, w_1z_1, w_1v_1,$ w_1	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(4,2,1)$	$\ldots, w_1^2z_1$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 7)$ $(\lambda(w_1v_1), \leq 7)$	29
$(5,2,1)$	$\ldots, w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 8)$ $(\lambda(w_1v_1), \leq 8)$	33
$(6,2,1)$	$\ldots, w_1^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 9)$	28
$(7,2,1)$	$\ldots, w_3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 10)$	31
$(8,2,1)$	$\ldots, w_1w_2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 11)$	34
$(9,2,1)$	$\ldots, w_1^3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 12)$	37
$(> 9,2,1)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	$2\deg G + 1$

The next table caters for divisors G, where $G = k_0 Q_\infty + 3Q_1 + Q_2$ and $k_0 \geq 3$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3,3,1)$	$1, z_1, w_1 z_1,$ $w_1 v_1, w_1,$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 7)$ $(\lambda(z_1), \leq 7)$	22
$(4,3,1)$	$\ldots, w_1^2 z_1$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 8)$ $(\lambda(z_1), \leq 8)$	25
$(5,3,1)$	$\ldots, w_2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 9)$ $(\lambda(z_1), \leq 9)$	28
$(6,3,1)$	$\ldots, w_1^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 10)$	31
$(7,3,1)$	$\ldots, w_3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 11)$	34
$(8,3,1)$	$\ldots, w_1 w_2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_2), \leq 12)$	37
$(9,3,1)$	$\ldots, w_1^3$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 13)$	40
$(> 9,3,1)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 9)$	$2 \deg G + 1$

The next table caters for divisors G, where $G = k_0 Q_\infty + 4Q_1 + Q_2$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4, 4, 1)$	$1, z_1, v_1 z_1^2, w_1 z_1,$ $(w_1 z_1)^2, w_1, w_1, w_1^2 z_1$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 9)$ $(\lambda(z_1), \leq 9)$	19
$(5, 4, 1)$	$\ldots, w_2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 10)$ $(\lambda(z_1), \leq 11)$	21
$(6, 4, 1)$	$\ldots, w_1^2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 5)$ $(\lambda(z_1), \leq 11)$	23
$(7, 4, 1)$	$\ldots, w_3$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 6)$ $(\lambda(z_1), \leq 12)$	25
$(8, 4, 1)$	$\ldots, w_1 w_2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 6)$ $(\lambda(z_1), \leq 13)$	27
$(9, 4, 1)$	$\ldots, w_1^3$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 4)$ $(\lambda(z_1), \leq 14)$	29
$(10, 4, 1)$	$\ldots, w_2^2$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 7)$	26
$(> 10, 4, 1)$	$\ldots, w_1^2 w_2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 9)$	$2 \deg G + 1$

Remark 8.4.9. For divisors G, where $G = k_0 Q_\infty + k_1 Q_1 + k_2 Q_2$ and $k_0 \geq k_1 \geq 5$ and $1 \leq k_2 \leq 4$, we follow the proof of Theorem 8.3.2 in the case $G = k_0 Q_\infty + k_1 Q_1$ with the assumption that $\deg D \geq 2 \deg G + 1$. The cases where $k_0 \geq k_2 \geq 5$ and $1 \leq k_1 \leq 4$ are dealt with in the same way.

The next tables deal with divisors G of the form $G = k_0 Q_\infty + k_1 Q_1 + 2Q_2$, where $k_0 = \max\{k_1, 2\}$ and $k_0 + k_1 + 2 \geq 6$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3,1,2)$	$1, v_1z_1, w_1v_1,$ w_1	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(4,1,2)$	$\ldots, (w_1v_1)^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 7)$ $(\lambda(w_1v_1), \leq 3)$	22
$(5,1,2)$	$\ldots, w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 8)$ $(\lambda(w_1v_1), \leq 4)$	25
$(6,1,2)$	$\ldots, w_1^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_1v_1), \leq 4)$	17
$(7,1,2)$	$\ldots, w_3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_1v_1), \leq 5)$	21
$(8,1,2)$	$\ldots, w_1w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_1v_1), \leq 5)$	21
$(9,1,2)$	$\ldots, w_1^3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_1v_1), \leq 6)$	23
$(>9,1,2)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_2), \leq 8)$	$2\deg G + 1$

Before we can produce another table we need to go through the motions once more in order to introduce yet another pair of generators of F together with their defining equation.

Case $G = 2Q_\infty + 2Q_1 + 2Q_2$:

It is a trivial exercise to show that $\mathcal{L}(2Q_\infty + 2Q_1 + 2Q_2) = \langle 1, v_1z_1, w_1z_1, w_1v_1\rangle$. We now note that $w_1z_1 = x/y^3$ and $w_1v_1 = y$ are generating elements of F, i.e. $F = \mathbb{F}_q(w_1z_1, w_1v_1) = \mathbb{F}_q(x,y)$. F is defined by the following equation:

$$(w_1z_1)^3(w_1v_1)^2 + (w_1z_1)(w_1v_1)^3 + 1 = 0. \tag{8.4.10}$$

Obviously $\deg((\lambda(w_1z_1))_\infty) \leq 6$, $\deg((\lambda(w_1v_1))_\infty) \leq 6$ and $\deg(\lambda(v_1z_1))_\infty \leq 6$. We need to show that $v_1z_1 = y^2/x = (w_1z_1)^{-1}(w_1v_1)^{-1}$ is preserved under the map λ. Because $|\operatorname{supp}(D)| > 40$ we can apply Lemma 4.3.5 to

$$\lambda(v_1z_1)\lambda(w_1z_1)\lambda(w_1v_1) = 1.$$

(Actually, we only need $|\operatorname{supp}(D)| > 19$ because the left hand side of the above equation has at most 18 poles.)

In order to apply Lemma 4.3.5 to deduce that

$$\lambda(w_1z_1)^3(w_1v_1)^2 + \lambda(w_1z_1)\lambda(w_1v_1)^3 + 1 = 0$$

we need at least 31 places in the support of D. This now shows that $\lambda(w_1z_1)$ and $\lambda(w_1v_1)$ determine an automorphism, $\widetilde{\lambda}$, of the function field F. We now apply Lemma 4.3.4 and Lemma 4.3.10 to deduce that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$.

The next table caters for divisors G, where $G = k_0Q_\infty + 2Q_1 + 2Q_2$ and $k_0 \geq 2$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(2,2,2)$	$1, v_1z_1, w_1z_1,$ w_1v_1	w_1z_1, w_1v_1	(8.4.10)	$(\lambda(w_1z_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	31
$(3,2,2)$	$\ldots, w_1$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 7)$ $(\lambda(w_1v_1), \leq 7)$	29
$(4,2,2)$	$\ldots, w_1^2z_1$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 8)$ $(\lambda(w_1v_1), \leq 4)$	25
$(5,2,2)$	$\ldots, w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 9)$ $(\lambda(w_1v_1), \leq 4)$	27
$(6,2,2)$	$\ldots, w_1^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_1v_1), \leq 5)$	21
$(6,2,2)$	$\ldots, w_3,$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_1v_1), \leq 5)$	23
$(8,2,2)$	$\ldots, w_1w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(9,2,2)$	$\ldots, w_1^3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_1v_1), \leq 6)$	23
$(> 9,2,2)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 9)$	$2\deg G + 1$

Note that because $(w_1v_1)^2 = y^2$ is an element of $\mathcal{L}(4Q_\infty + 2Q_1)$ we can deduce that

$$\deg\left((\lambda(w_1v_1))_\infty\right) \leq \left[\frac{\deg(G)}{2}\right]$$

as soon as $k_0 \geq 4$ in the above table.

The next table caters for divisors G, where $G = k_0Q_\infty + 3Q_1 + 2Q_2$, $k_0 \geq 3$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3,3,2)$	$1, z_1, v_1z_1,$ w_1z_1, w_1v_1, w_1	z_1, w_1z_1	(8.4.4)	$(\lambda(z_1), \leq 8)$ $(\lambda(w_1z_1), \leq 8)$	25
$(4,3,2)$	$\ldots, w_1^2z_1$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 9)$ $(\lambda(w_1v_1), \leq 4)$	28
$(5,3,2)$	$\ldots, w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 10)$ $(\lambda(w_1v_1), \leq 5)$	31
$(6,3,2)$	$\ldots, w_1^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_1v_1), \leq 5)$	21
$(7,3,2)$	$\ldots, w_3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1v_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(8,3,2)$	$\ldots, w_1w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(9,3,2)$	$\ldots, w_1^3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 4)$ $(\lambda(w_1v_1), \leq 7)$	26
$(>9,3,2)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 10)$	$2\deg G + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 4Q_1 + 2Q_2$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4,4,2)$	$1, z_1, v_1z_1^2, v_1z_1, w_1z_1,$ $(w_1z_1)^2, w_1, w_1^2z_1$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 10)$ $(\lambda(z_1), \leq 10)$	21
$(5,4,2)$	$\ldots, w_2$	w_1, z_1	(8.4.4)	$(\lambda(w_1), \leq 11)$ $(\lambda(z_1), \leq 11)$	23
$(6,4,2)$	$\ldots, w_1^2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(7,4,2)$	$\ldots, w_3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 6)$ $(\lambda(w_1v_1), \leq 6)$	25
$(8,4,2)$	$\ldots, w_1w_2$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 7)$ $(\lambda(w_1v_1), \leq 7)$	29
$(9,4,2)$	$\ldots, w_1^3$	w_1, w_1v_1	(8.4.7)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_1v_1), \leq 7)$	27
$(>9,4,2)$	$\ldots, w_2^2, \ldots$	w_1, w_2	(8.4.2)	$(\lambda(w_1), \leq 5)$ $(\lambda(w_2), \leq 10)$	$2\deg G + 1$

The next tables deal with divisors G, where $G = k_0Q_\infty + k_1Q_1 + 3Q_2$, with $k_0 = \max\{k_1, 3\}$, $k_1 \geq 1$, and $k_0 + k_1 + 3 \geq 7$. They all use the following observation.

Remark 8.4.11. In each case the generators used are w_1 and v_1 together with the defining equation (8.4.8):

$$(w_1v_1)^3 + w_1^2 - v_1 = 0.$$

Because $\{w_1, v_1, (w_1v_1)\} \subset \mathcal{L}(3Q_\infty + 3Q_2)$ we can deduce that $\lambda(w_1v_1) = \lambda(w_1)\lambda(v_1)$. Furthermore, as soon as $4Q_\infty + 3Q_2 \leq G$, $(w_1v_1)^2 \in \mathcal{L}(G)$ as well and hence by Lemma 4.3.9 we find that

$$\deg\left((\lambda(w_1)\lambda(v_1))_\infty\right) \leq \deg(G)/2.$$

This then allows us to apply Lemma 4.3.5 to

$$(\lambda(w_1)\lambda(v_1))^2 + \lambda(w_1)^2 - \lambda(v_1) = 0$$

provided that $\deg D \geq 2\deg(G) + 1$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3, 1, 3)$	$1, v_1z_1, v_1,$ w_1, w_1v_1	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 7)$ $(\lambda(v_1), \leq 7)$	22
$(4, 1, 3)$	$\ldots, w_1^2z_1$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 8)$ $(\lambda(v_1), \leq 8)$	17
$(5, 1, 3)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 9)$ $(\lambda(v_1), \leq 9)$	19
$(> 5, 1, 3)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 6)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G) + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 2Q_1 + 3Q_2$ and $k_0 \geq 3$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3, 2, 3)$	$1, v_1z_1, v_1,$ w_1z_1, w_1v_1, w_1	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 8)$ $(\lambda(v_1), \leq 8)$	25
$(4, 2, 3)$	$\ldots, (w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 9)$ $(\lambda(v_1), \leq 9)$	19
$(5, 2, 3)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 10)$ $(\lambda(v_1), \leq 10)$	21
$(> 5, 2, 3)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 6)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G) + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 3Q_1 + 3Q_2$, $k_0 \geq 3$. Remark 8.4.11 still applies here.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(3,3,3)$	$1, z_1, v_1z_1, v_1$ w_1z_1, w_1v_1, w_1	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 9)$ $(\lambda(v_1), \leq 9)$	28
$(4,3,3)$	$\ldots, (w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 10)$ $(\lambda(v_1), \leq 10)$	21
$(5,3,3)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 11)$ $(\lambda(v_1), \leq 11)$	23
$(> 5,3,3)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 7)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G)+1$

The next table caters for divisors G, where $G = k_0Q_\infty + 4Q_1 + 3Q_2$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4,4,3)$	$1, z_1, v_1z_1^2,$ $v_1z_1, v_1, w_1z_1,$ $(w_1v_1), w_1, (w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 11)$ $(\lambda(v_1), \leq 11)$	23
$(5,4,3)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 12)$ $(\lambda(v_1), \leq 12)$	25
$(> 5,4,3)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 7)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G)+1$

The last cases we have to look at are for divisors $G = k_0Q_\infty + k_1Q_\infty + 4Q_2$, where $k_0 \geq 4$ and $1 \leq k_1 \leq 4$. Remark 8.4.11 still applies.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4,1,4)$	$1, v_1z_1, v_1,$ $v_1^2w_1, w_1v_1, w_1z_1$ $(w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 9)$ $(\lambda(v_1), \leq 9)$	19
$(5,1,4)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 10)$ $(\lambda(v_1), \leq 10)$	21
$(> 5,1,4)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 6)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G)+1$

The next table caters for divisors G, where $G = k_0Q_\infty + 2Q_1 + 4Q_2$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4, 2, 4)$	$1, v_1z_1, v_1,$ $(v_1z_1)^2, w_1z_1, w_1v_1,$ $w_1, (w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 10)$ $(\lambda(v_1), \leq 10)$	21
$(5, 2, 4)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 11)$ $(\lambda(v_1), \leq 11)$	23
$(> 5, 2, 4)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 7)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G) + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 3Q_1 + 4Q_2$, $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4, 3, 4)$	$1, z_1, v_1z_1, v_1$ $(v_1z_1)^2, w_1z_1, w_1v_1$ $w_1, (w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 11)$ $(\lambda(v_1), \leq 11)$	23
$(5, 3, 4)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 12)$ $(\lambda(v_1), \leq 12)$	25
$(> 5, 3, 4)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 7)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G) + 1$

The next table caters for divisors G, where $G = k_0Q_\infty + 4Q_1 + 4Q_2$ and $k_0 \geq 4$.

(k_0, k_1, k_2)	Basis of $\mathcal{L}(G)$	Generators	Equation	Degree	Places needed
$(4, 4, 4)$	$1, z_1, v_1z_1^2,$ $v_1z_1, v_1, (v_1z_1)^2,$ $w_1z_1, w_1v_1, w_1,$ $(w_1v_1)^2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 12)$ $(\lambda(v_1), \leq 12)$	25
$(5, 4, 4)$	$\ldots, w_2$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 13)$ $(\lambda(v_1), \leq 13)$	27
$(> 5, 4, 4)$	$\ldots, w_1^2, \ldots$	w_1, v_1	(8.4.8)	$(\lambda(w_1), \leq 8)$ $(\lambda(v_1), \leq \deg G)$	$2\deg(G) + 1$

Divisors of the form $G = k_0Q_\infty + k_1Q_1 + k_2Q_2$ with $k_i \geq 5$ $(i = 0, 1, 2)$ have been dealt with in Theorem 8.3.2. This completes the proof. □

8.5 The Remaining Cases and Examples

This section deals with divisors $G = k_0Q_\infty + k_1Q_1 + k_2Q_2$, where $\deg(G) = 5$, $k_0 = \max\{k_0, k_1, k_2\}$ and k_1, k_2 not both zero. This is the last case where the expected result holds because, in general, once $\deg G < 5$ it follows that $\mathcal{L}(G)$ does not contain two generators of the function field which is an essential requirement of our method.

At the end of this section we summarise all the results into one theorem and give a couple of examples of codes associated with the Klein Quartic.

Lemma 8.5.1. *Let $G = k_0Q_\infty + k_1Q_1 + k_2Q_2$ be such that $\deg(G) = 5$ and let $D = \sum_{P\in J} P$, where $J \subset \mathbb{P}_F^{(1)} \setminus \{Q_\infty, Q_1, Q_2\}$. If $\deg D > 36$ then*

$$Aut_{D,G}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D, G)).$$

Proof. As usual, without loss of generality, we can assume that $k_0 = \max\{k_0, k_1, k_2\}$ and k_1, k_2 not both zero. (Note that the case $k_0 = 5$ has already been dealt with in Theorem 8.3.2.)

By Proposition 3.3.4 we can regard $Aut_{D,G}(F/\mathbb{F}_q)$ as a subgroup of $Aut(\mathcal{C}_\mathcal{L}(D, G))$. Now let $\pi \in Aut(\mathcal{C}_\mathcal{L}(D, G))$. We need to show that π is induced by an element of $Aut_{D,G}(F/\mathbb{F}_q)$. Denote by λ the map from $\mathcal{L}(G)$ onto itself as described in Definition 4.3.6.

There are seven possibilities for our divisor G. We shall sketch the proof for each case.

Case $G = 2Q_\infty + Q_1 + 2Q_2$:

It is easy to check that $\mathcal{L}(2Q_\infty + Q_1 + 2Q_2) = \langle 1, v_1z_1, w_1v_1\rangle$. Now $v_1z_1 = y^2/x$ and $w_1v_1 = y$ are generating elements of F which satisfy the following equation:

$$(w_1v_1)^3(v_1z_1)^2 + (w_1v_1)(v_1z_1)^3 + 1 = 0.$$

By Lemma 4.3.5 λ preserves this equation provided that $\deg D \geq 26$ (Note that $\deg((\lambda(v_1z_1))_\infty) \leq 5$ and similarly $\deg((\lambda(w_1v_1))_\infty) \leq 5$). Hence $\lambda(v_1z_1)$ and $\lambda(w_1v_1)$ determine an automorphism, $\widetilde{\lambda}$, of F. We need to show that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$. We cannot apply Lemma 4.3.4 now because the degree of G is too small. However, we still have that $\widetilde{\lambda}(\mathcal{L}(G)) = \mathcal{L}(\widetilde{\lambda}(G)) = \mathcal{L}(G)$. Now $(v_1z_1)_\infty = 2Q_2 + Q_1$ and $(w_1v_1)_\infty = 2Q_\infty + Q_2$. Because $v_1z_1, w_1v_1 \in \mathcal{L}(\widetilde{\lambda}(G))$ we know that $2Q_\infty + Q_1 + 2Q_\infty \leq \widetilde{\lambda}(G)$ and hence we must actually have equality, i.e. $\widetilde{\lambda}(G) = G$. We can now apply Lemma 4.3.10 to complete the proof in this case.

Case $G = 2Q_\infty + 2Q_1 + Q_2$: entirely analogous to the previous case.

Case $G = 3Q_\infty + 2Q_1$:

This time $\mathcal{L}(3Q_\infty + 2Q_1) = \langle 1, w_1z_1, w_1\rangle$, and $w_1z_1 = x/y^3$ and $w_1 = x/y^2$ satisfy:

$$w_1^5 = w_1^2(w_1z_1)^2 - w_1^4(w_1z_1)^3.$$

Hence $\lambda(w_1)$ and $\lambda(w_1 z_1)$ determine an automorphism, $\widetilde{\lambda}$, of F provided that $\deg D > 35$. Because $w_1, w_1 z_1 \in \mathcal{L}(\widetilde{\lambda}(G))$, $(w_1)_\infty = 3Q_\infty$ and $(w_1 z_1)_\infty = 2Q_1 + Q_\infty$ we know that $3Q_\infty + 2Q_1 \leq \widetilde{\lambda}(G)$ and hence $\widetilde{\lambda}(G) = G$. Lemma 4.3.10 now completes the proof.

Case $G = 3Q_\infty + 2Q_2$:

We have $\mathcal{L}(3Q_\infty + 2Q_2) = \langle 1, w_1 v_1, w_1 \rangle$ and the defining equation we use is:

$$w_1^3 + w_1(w_1 v_1)^3 - (w_1 v_1) = 0.$$

So $\lambda(w_1)$ and $\lambda(w_1 v_1)$ determine an automorphism, $\widetilde{\lambda}$, of F provided that $\deg(D) > 21$. We need to show that $\widetilde{\lambda}(G) = G$. Now $w_1, w_1 v_1 \in \mathcal{L}(\widetilde{\lambda}(G))$. We note that $(w_1)_\infty = 3Q_\infty$, $(w_1 v_1)_\infty = 2Q_\infty + Q_2$ and thus $3Q_\infty + Q_2 \leq 3\widetilde{\lambda}(Q_\infty) + 2\widetilde{\lambda}(Q_2) = \widetilde{\lambda}(G)$ which implies that $\widetilde{\lambda}(G) = G$. Lemma 4.3.10 now shows that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$.

Cases $G = 4Q_\infty + Q_1$ and $G = 4Q_\infty + Q_2$: analogous to one of the above cases.

Case $G = 3Q_\infty + Q_1 + Q_2$:

This is the only slightly problematic case. Again $\mathcal{L}(3Q_\infty + Q_1 + Q_2) = \langle 1, w_1 v_1, w_1 \rangle$ with defining equation:

$$w_1^3 + w_1(w_1 v_1)^3 - (w_1 v_1) = 0.$$

So again we need $\deg D \geq 21$ for $\lambda(w_1)$ and $\lambda(w_1 v_1)$ to determine an automorphism, $\widetilde{\lambda}$, of the function field F. As before we see that $3Q_\infty + Q_2 \leq \widetilde{\lambda}(G) = 3\widetilde{\lambda}(Q_\infty) + \widetilde{\lambda}(Q_1) + \widetilde{\lambda}(Q_2)$. This implies that $\widetilde{\lambda}$ fixes Q_∞. But it is not immediate where Q_1 and Q_2 go under $\widetilde{\lambda}$. However looking at $(\widetilde{\lambda}(x)) = \widetilde{\lambda}(Q_1) + 7Q_\infty$, we see that $\widetilde{\lambda}(Q_1)$ is another totally ramified place and hence must be either Q_1 or Q_2. Thus $\widetilde{\lambda}(G) = G$. Lemma 4.3.10 now yields that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$. □

The previous lemma together with Theorem 8.4.1 yields the following theorem.

Theorem 8.5.2. *Let F be the function field associated with the Klein Quartic. Suppose that G is a divisor such that $\deg G \geq 5$ and $\mathrm{supp}(G) \subseteq \{Q_\infty, Q_1, Q_2\}$. Let $D = \sum_{P \in J} P$, where $J \subseteq \mathbb{P}_F^{(1)} \setminus \{Q_\infty, Q_1, Q_2\}$. If $\deg(D) \geq \max\{40, 2\deg G + 1\}$ then*

$$Aut_{D,G}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D, G)).$$

Proof. Clear. □

Remark 8.5.3. The proofs of the various results leading up to the last theorem actually show that the requirement $\deg G \geq \max\{40, 2\deg G + 1\}$ is not the best bound for our method to work. For most divisors G of small degree the number of places required in the support of D is smaller

than 40. The number of places in the support of D for an individual divisor G can be found in the various tables or worked out manually. Furthermore, empirical data suggest that the results hold for even fewer places than given in the table. The following examples provide some evidence for that.

Example 8.5.4. Let $F/\mathbb{F}_{19}$ be the function field associated with the Klein Quartic. It is easy to establish that F has 20 places of degree 1. With the same notation as before, we first look at the code associated with the divisors $G = 2Q_\infty + Q_1 + 2Q_2$ and $D = \sum_{P\in J} P$, where $J = \mathbb{P}_F^{(1)} \setminus \{Q_\infty, Q_1, Q_2\}$. We have already shown that $\mathcal{L}(2Q_\infty + Q_1 + 2Q_2) = \langle 1, v_1z_1, w_1v_1\rangle$. Hence the code $\mathcal{C}_\mathcal{L}(D, G)$ has length 17 and dimension 3. It is a straightforward check (provided one has the right software) that $Aut(\mathcal{C}_\mathcal{L}(D, G)) = \{id\}$, where id is the identity permutation, therefore, by Proposition 3.3.4, $Aut_{D,G}(F/\mathbb{F}_{19}) \cong \{id\}$ as well. Incidentally, this example also shows that the bounds we have given for our divisors D in the proof of the previous theorems and lemmas might not be best possible. For this case our proof required D to have at least 26 places in its support but here we only have 17 indicating that an improvement on the bounds for $\deg D$ might be possible.

Looking at $G = 2Q_\infty + 2Q_1 + 2Q_2$ we expect the associated code to have an automorphism group of at least 3 elements, namely all the powers of α because $\alpha(G) = G$ and $\alpha(D) = D$. Now $\mathcal{L}(G) = \langle 1, v_1z_1, w_1v_1, w_1z_1\rangle$. And again a computer check reveals that $Aut(\mathcal{C}_\mathcal{L}(D, G)) \cong \langle\alpha\rangle$. Again the number of places required in our proof, 31, is much bigger than the 17 places we actually had..

Lastly, we look at the case where $G = 2Q_\infty+2Q_2$. Now $\mathcal{L}(G) = \langle 1, w_1v_1\rangle$. And now it turns out that $|Aut(\mathcal{C}_\mathcal{L}(D, G))| = 72$. However, $Aut_{D,G}(F/\mathbb{F}_q) = \{id\}$ which can be deduced, for example, from looking at the code $\mathcal{C}_\mathcal{L}(D, 3Q_\infty + 3Q_2)$ whose group of automorphisms is trivial.

Example 8.5.5. Let $F = \mathbb{F}_{43}(x, y)$ be the function field associated with the Klein Quartic. It is straightforward to check that $|\mathbb{P}_F^{(1)}| = 80$. Furthermore, because $\mathbb{F}_{43}$ contains a seventh root of unity, we also have that $H = Gal(F/\mathbb{F}_{43}(x))$, the Galois group of F over $\mathbb{F}_{43}(x)$, is cyclic of order 7. Looking at the divisors $G = Q_\infty + Q_1 + Q_2$ and $D = \sum_{P\in J} P$, where $J = \mathbb{P}_F^{(1)} \setminus \{Q_\infty, Q_1, Q_2\}$. we find that, for $2 \le k \le 12$, $\mathcal{C}_\mathcal{L}(D, kG)$ is a Goppa code of length 77 and dimension $3k - 2$. For these values of k, Theorem 8.5.2 applies and we find that $Aut_{D,G}(F/\mathbb{F}_{43}) \cong Aut(\mathcal{C}_\mathcal{L}(D, G))$. If σ is a generator of H then $Aut_{D,G}(F/\mathbb{F}_{43}) = \langle\sigma, \alpha\rangle$, where α as before. Thus $|Aut(\mathcal{C}_\mathcal{L}(D, G))| = 21$.

Chapter 9

Computational Aspects And Conclusion

9.1 Introduction

This chapter is mainly concerned with the implementation of various routines which enabled us to compute numerous examples of (hyper-)elliptic codes and codes associated with the Klein Quartic using the computer algebra package MAPLE. Without the use of these routines it is very doubtful if any of the results in the previous chapters would have been obtained.

The algebra package MAPLE was chosen mainly because it was available on the departmental computers and seemed to have most of the features we required. Furthermore, most people in the department were familiar with MAPLE rather than any other algebra package that could have been considered (like Magma for example).

The program whose printout can be found in Appendix A allows the user to compute easily a great variety of elliptic codes and compute their group of automorphisms. It is also possible, though not as straightforward, to compute generator matrices for hyperelliptic codes and codes associated with function fields of the Klein Quartic.

We shall give a brief description of the methods we used and their limitations due to theoretical reasons and practical considerations.

Before going into more detail how we implemented the routines we fix some notation valid for the entire chapter:

- $F/\mathbb{F}_q$ is an algebraic function field of genus g.

- $\mathbb{P}_F^{(1)}$ denotes the set of all places of degree one in F.
- $P_1, P_2, \ldots, P_n$ are pairwise distinct places of $F/\mathbb{F}_q$ of degree one.
- $D = P_1 + \cdots + P_n$.
- G is a divisor of $F/\mathbb{F}_q$ such that $\operatorname{supp} G \cap \operatorname{supp} D = \emptyset$.
- $\mathcal{C}_{\mathcal{L}}(D, G)$ is the geometric Goppa code associated with the two divisors D and G.
- $Aut(F/\mathbb{F}_q)$ is the group of $\mathbb{F}_q$-automorphisms of F.
- $Aut_{D,G}(F/\mathbb{F}_q)$ denotes the group of $\mathbb{F}_q$-automorphisms of F fixing the divisors D and G.
- An $[n, k, d]$-code is a linear code of length n, dimension k, and minimum distance d (cf. Section 1.5).

9.2 The General Idea

There are two main parts to the program. The first one deals with how to compute a generator matrix of a geometric Goppa code. The second part finds the group of automorphisms of any linear code when given its generator matrix. Both parts have their own problems that need to be addressed.

The most difficult aspect when dealing with geometric Goppa codes $\mathcal{C}_{\mathcal{L}}(D, G)$ is how to find a basis for the vector space $\mathcal{L}(G)$. As soon as one has a basis of $\mathcal{L}(G)$, say $\mathcal{L}(G) = \langle x_1, x_2, \ldots, x_k \rangle$, then

$$M = \begin{pmatrix} x_1(P_1) & x_1(P_2) & \cdots & x_1(P_n) \\ \vdots & \vdots & & \vdots \\ x_k(P_1) & x_k(P_2) & \cdots & x_k(P_n) \end{pmatrix}$$

is a generator matrix for $\mathcal{C}_{\mathcal{L}}(D, G)$ (cf. Corollary 3.2.6).

The procedures dealing with this will be discussed in Section 9.3.

Assuming we have found such a basis and hence were able to compute M, how does one then determine $Aut(\mathcal{C}_{\mathcal{L}}(D, G))$?

It is very easy to see that knowing the generator matrix of a linear code C is all we need to find its group of automorphism:
Suppose we want to check whether $\pi \in S_n$ is an element of $Aut(C)$ where C is given by the generator matrix M. We first compute the **Reduced Echelon Form (REF)** of M obtaining M'. We then apply π to the columns of M' and reduce the matrix thus obtained to REF. If it

agrees with the M' then π is an automorphism of C. Thus going through all permutations in S_n and finding those that preserve the code is one way of finding the automorphisms of a code. Obviously, this approach by brute force is intractable as soon as n is sufficiently large because of the sheer number $(n!)$ of permutations we have to check. The method we implemented is slightly better and based on a back-tracking algorithm described by Leon [Leo84] which will be discussed in Section 9.4.

9.3 The Program - Function Fields

As mentioned before this part of the program is concerned with determining a generator matrix of a geometric Goppa code $\mathcal{C}_{\mathcal{L}}(D, G)$. We shall look at the various function fields for which we have written procedures which will enable us to compute the generator matrix of certain Goppa codes associated with them.

Remark 9.3.1. Throughout the following sections we will refer to various names of MAPLE routines that we wrote, e.g. "FindPlaces". All these routines can be found in the print-out of the MAPLE program in Appendix A.

9.3.1 The elliptic and hyperelliptic case

To make the computations fairly straightforward we restricted our attention to (hyper-)elliptic function fields $F/\mathbb{F}_q$, where

(1) $q = p$ is actually an odd prime, and

(2) $F = \mathbb{F}_p(x, y)$ with

$$y^2 = f(x), \qquad f(x) \in \mathbb{F}_p[x] \text{ square-free of degree } 2g + 1. \tag{9.3.2}$$

The reason why we restricted our attention to finite fields of prime order is due to MAPLE's "complicated" way of representing elements in a finite field extension of $\mathbb{F}_p$.

Furthermore, because p is an odd prime knowledge of $f(x)$ is all we need to determine all the places of degree one in F. We know from Proposition 2.6.13 that Q_∞, the common pole of x and y, is a ramified place of degree one. All one has to do to determine the rest of $\mathbb{P}_F^{(1)}$ is to evaluate $f(x)$ for each $x \in \mathbb{F}_p$ in turn : Let $\alpha \in \mathbb{F}_p$. There are three cases.

(1) If $f(\alpha) = 0$ then $(x-\alpha)$ is ramified in $F/\mathbb{F}_p(x)$, i.e. $(x-\alpha) = 2P_{\alpha,0} - 2Q_\infty$, with $\deg P_{\alpha,0} = 1$.

(2) If $f(\alpha) = \beta^2$, $\beta \in \mathbb{F}_p$ then $(x-\alpha)$ splits in $F/\mathbb{F}_p$, i.e. $(x-\alpha) = P_{\alpha,\beta} + P_{\alpha,-\beta} - 2Q_\infty$, with $\deg P_{\alpha,\beta} = \deg P_{\alpha,-\beta} = 1$.

(3) If $f(\alpha) = \beta^2$, $\beta \notin \mathbb{F}_p$ then $(x - \alpha)$ stays inert in $F/\mathbb{F}_p$, i.e. $(x - \alpha) = P_\alpha - 2Q_\infty$, with $\deg P_\alpha = 2$.

This is very easy to implement in MAPLE as it provides functions to compute the square root of numbers mod p.

Because we can only deal with finite fields $\mathbb{F}_p$, where p is a prime, it is not really a restriction to concentrate on odd characteristic only. After all, the Serre Bound (cf. Theorem 2.5.11) shows that the number of places of degree one in an elliptic function field $F/\mathbb{F}_2$ is at most 5 which does not allow the construction of any interesting codes.

To simplify the computation further, our MAPLE program can only deal with codes $\mathcal{C}_\mathcal{L}(D, G)$ whose divisor G is of a somewhat special form. Proposition 5.2.6 showed that for a certain divisors G a basis of $\mathcal{L}(G)$ can be computed easily. Looking at the proof of this proposition, one sees that it is very straightforward to determine a simple basis for divisors G of the form:

$$G = n_\infty Q_\infty + \sum_{Q \in J} n_Q Q \quad \text{where } n_Q \geq 1 \text{ for all } Q \in J \subset \mathbb{P}_F^{(1)} \text{ and } n_\infty \geq 2g - 1. \tag{9.3.3}$$

Remark 9.3.4. In the special case of the elliptic function field, we made use of the underlying group structure to compute a basis for any effective divisor G, where $\operatorname{supp} G \subset \mathbb{P}_F^{(1)}$ because by using a simple shift we can always assume that $Q_\infty \in \operatorname{supp}(G)$.

In summary, we have imposed the following restrictions on the codes we want to compute:

- Let $F/\mathbb{F}_p$ be a (hyper-)elliptic function field of genus g given by (9.3.2), where $p \neq 2$ is a prime.
- Let G be an effective divisors of F given by (9.3.3), such that $\operatorname{supp}(D) \cap \operatorname{supp}(G) = \emptyset$.

The restrictions on the divisor G are only imposed on the codes which can be computed automatically by our MAPLE program. There are, of course, routines, e.g. "CreateMatrix2", that will compute the generator matrix of any code $\mathcal{C}_\mathcal{L}(D, G)$, where $\operatorname{supp}(D) \cap \operatorname{supp}(G) = \emptyset$, provided that the user works out a basis for $\mathcal{L}(G)$ manually.

By Proposition 3.3.4 $Aut_{D,G}(F/\mathbb{F}_p)$ can be regarded as a subgroup of $Aut(\mathcal{C}_\mathcal{L}(D, G))$ provided that $\deg D > 2g + 2$. Having chosen G the way we did has the added advantage that we can actually find all elements of $Aut_{D,G}(F/\mathbb{F}_p)$ in this case. The way to go about it is to compute $Aut(F/\mathbb{F}_p)$ first and then check which automorphisms actually preserve G and D. The following lemma describes $Aut(F/k)$, where k is an arbitrary perfect field, in the case of an elliptic function field F/k.

The set $H(k)$ consisting of all places of degree one can be equipped with a group operation $\oplus$ which turns $H(k)$ into an abelian group with Q_∞ as its identity element (cf. Section 2.6.2).

Denote by $Aut_\infty(F/k)$ the group of Q_∞-fixing automorphisms of F whose action on $H(k)$ yields group automorphisms of $H(k)$.

Lemma 9.3.5. *Let $F/\mathbb{F}_p$ be an elliptic function field of genus 1. Then there exists a bijection between the two sets*

$$H(k) \times Aut_\infty(F/k) \widetilde{\rightarrow} Aut(F/k)$$
$$(P, \epsilon) \mapsto \tau_P \circ \epsilon$$

where τ_P is the translation $\tau_P(Q) = Q \oplus P$.

Proof. It is well-known (cf. [Sil86, p.68, Theorem 3.6]) that for a fixed $P \in \mathbb{P}_F^{(1)}$, the translation $\tau_P(Q) = Q \oplus P$, where $Q \in \mathbb{P}_F^{(1)}$, is an automorphism of the elliptic function field. Now if $\sigma \in Aut(F/k)$ then we see that

$$(\tau_{(\ominus\sigma(Q_\infty))} \circ \sigma) = \epsilon \in Aut_\infty(F/k).$$

Thus $\sigma = \tau_{\sigma(Q_\infty)} \circ \epsilon$. Note that the group structure is not a direct product, in fact the group structure is given by :

$$(P_1, \epsilon_1) \circ (P_2, \epsilon_2) = (P_1 \oplus \epsilon_1(P_2), \epsilon_1 \circ \epsilon_2).$$

□

For the hyperelliptic case with $g \geq 2$ the situation is as follows.

Lemma 9.3.6. *Let F/k (with k perfect) be a hyperelliptic function field of genus $g \geq 2$. Then $Aut(F/k)$ is isomorphic to an extension of a subgroup of $PGL(2, k)$ by a cyclic group of order 2.*

Proof. It is well-known (cf. [Sti91, p.195, Proposition VI.2.4.]) that there exists an x such that $k(x)$ is the only rational subfield of F with $[F : k(x)] = 2$. Now any automorphism, σ say, of F must send $k(x)$ to a rational subfield of F of the same co-dimension, i.e. to itself. Therefore σ restricts to $k(x)$ and the restriction of σ to $k(x)$ is an automorphism of $k(x)$. Now $Aut(k(x)/k) = PGL(2, p)$. Hence we have the following group homomorphism:

$$\alpha : \begin{cases} Aut(F/k) & \longrightarrow PGL(2, k) \\ \sigma & \longmapsto \sigma|_{k(x)}. \end{cases}$$

The kernel of α has size 2 (because $F/k(x)$ is an extension of degree 2 and hence there exists a non-trivial $k(x)$-automorphism of F). Thus $Aut(F/k)$ is isomorphic to an extension of a subgroup of $PGL(2, k)$ by a cyclic group of order 2. □

Remark. In our MAPLE program we obviously have $k = \mathbb{F}_p$.

We see that in the (hyper-)elliptic function field case, the group $Aut(F/\mathbb{F}_p)$ is very easy to compute. Observe that an automorphism, σ, of a (hyper-)elliptic function field is completely determined by the image of x and y under σ. The MAPLE routine "FindHyperAutos" determines all automorphisms of a (hyper-)elliptic function field and returns a list of pairs of the form $(\sigma(x), \sigma(y))$.

Now because both D and G consist of places of degree one which are uniquely determined by their values at x and y, computing the image of a place of degree one under an automorphism of the function field is a trivial exercise. Computing the images of places in the support of the divisors D and G is then used in "FindAutGD" to determine those automorphisms that actually fix D and G by checking whether the image of a place is still in the support of the divisor in question and has the right multiplicity as well. The routines which we implemented in MAPLE allow the user to do the following:

- Find all the places of degree 1 of the (hyper-)elliptic function field F given by (9.3.2) ("FindPlaces").
- Define divisors D and G whose support is a subset of $\mathbb{P}_F^{(1)}$ ("CreateD" and "CreateG").
- Find a basis of $\mathcal{L}(G)$ provided G is of the form given by (9.3.3)("FindBasisLG").
- Compute the generator matrix M of $\mathcal{C}_{\mathcal{L}}(D, G)$ ("CreateMatrix" (a basis for $\mathcal{L}(G)$ is computed automatically) and "CreateMatrix2"(the user provides a basis for $\mathcal{L}(G)$)).
- Calculate $Aut_{D,G}(F/\mathbb{F}_p)$ if required ("FindAutGD").
- Determine $Aut(\mathcal{C}_{\mathcal{L}}(D, G))$ ("FindAutGroup2", this actually calls a C++ program - see Section 9.4).

9.3.2 Codes associated with the Klein Quartic

The MAPLE program provides a routine called "Klein" which determines the places of degree one in the function field $F/\mathbb{F}_p$, where $7 \neq p$ is prime and $F = \mathbb{F}_p(x, y)$ with

$$y^7 = \frac{x^3}{1-x}.$$

Having computed all the places of degree one, the user then has to specify the divisor D (using "CreateD"). There is no routine which automatically determines a basis for $\mathcal{L}(G)$. In order to determine a code associated with the Klein Quartic the user has to compute a basis for $\mathcal{L}(G)$ manually and use "CreateMatrix2" to obtain the generator matrix of $\mathcal{C}_{\mathcal{L}}(D, G)$. Chapter 8 contains

various tables listing bases for various divisors G. Using any of those one can readily compute a generator matrix for such a code and then compute its group of automorphisms calling the routine "FindAutGroup2".

The routine "AutoTest" allows the user to apply the following automorphism α of $F/\mathbb{F}_p$ to a place of degree 1:

$$\alpha : \begin{cases} x \mapsto 1 - \frac{1}{x}, \\ y \mapsto -\frac{x}{y^3}. \end{cases}$$

This might be helpful in determining the automorphism group $Aut_{D,G}(F/\mathbb{F}_p)$ which the program cannot compute automatically.

9.3.3 Dual codes

Since the algorithm we use to compute the automorphism group of a linear code is very slow as soon as the code's dimension is greater than 6 (cf. Section 9.4), it is virtually impossible to determine the automorphism group for codes of high dimension. However, because $Aut(C) = Aut(C^\perp)$, where $C^\perp$ is the dual of a linear code C (see Theorem 1.7.3), it may be desirable to compute a generator matrix of the dual code rather than the original code and work out its group of automorphisms. The reason being that if C is a code of length n and dimension k, its dual will have length n and dimension $n - k$.

The routine "DualCode2" allows the user to compute the generator matrix $M^\perp$ of the dual of the geometric Goppa code $\mathcal{C}_\mathcal{L}(D, G)$. The generator matrix M' of $\mathcal{C}_\mathcal{L}(D, G)$ is first of all changed into standard form, i.e.

$$M = (I_k, P)$$

where I_k represents the $k \times k$-identity matrix and P the parity check part. This might involve permuting some of the columns of M which the routine, however, keeps track of. It is easy to see that

$$M^\perp = (-P^t, I_{n-k}).$$

is a generator matrix of the dual code (cf. Section 1.5). Undoing the column changes then yields a generator matrix for the dual of the code we started off with. This can then be used to compute the automorphism group of the dual code instead. Obviously this is only of any interest if the dimension of the dual code is sufficiently small (e.g. $n - k \leq 6$), for otherwise our algorithm for determining the code automorphisms will still be too slow.

9.4 The Program - Code Automorphisms

We have already mentioned that the method we use to compute the automorphism group of a linear code is based on Leon's backtracking algorithm described in [Leo84]. Here we shall give a short summary of how the algorithm works.

Assume that the generator matrix M of a linear code C is in standard form, i.e. $M = (I_k, P)$. Denote by $c_1, c_2, \ldots, c_n$ its n columns, i.e. $M = (c_1, c_2, \ldots, c_n)$. Recall that an element of $Aut(C)$ is a permutation $\pi \in S_n$ such that $\pi(M) = (c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(n)})$ is row-equivalent to M.

The idea of the algorithm is to build up an automorphism of the code from partial permutations. The way we represent a permutations, $\sigma \in S_n$, is as follows:

$$\sigma = (a_1, a_2, \ldots, a_n) = \begin{pmatrix} a_1 & a_2 & a_3 & \ldots & a_n \\ 1 & 2 & 3 & \ldots & n, \end{pmatrix}$$

where $a_i \in \{1, 2, \ldots, n\}$. This is non-standard in the sense that it is "upside down" but it makes the implementation easier later on.

The general idea is as follows:
A partial permutation $\pi_d = (a_1, \ldots, a_d)$ where $d < n$ is "good" if the matrix $N_d = (c_{a_1}, \ldots, c_{a_d})$ is row-equivalent to the matrix $M_d = (c_1, \ldots, c_d)$, i.e. the first d columns of M. In other words there exists elementary row operations which change N_d into M_d. We can now "extend" π_d by picking $a_{d+1} \in \{1, 2, \ldots, n\} \setminus \{a_1, \ldots, a_d\}$ and check whether $\pi_{d+1} = (a_1, \ldots, a_d, a_{d+1})$ is "good".

If, however, N_d is not row-equivalent to M_d (π_d is "bad") then all permutations of S_n starting with $(a_1, \ldots, a_d)$ cannot be automorphisms of the code C and thus do not need to be checked. This is called pruning the search tree. We then "backtrack" and check whether we can extend the partial permutation $\pi_{d-1} = (a_1, \ldots, a_{d-1})$ by selecting a $a'_d \in \{1, \ldots, n\} \setminus \{a_1, \ldots, a_{d-1}, a_d\}$ to find a "good" $\pi'_d = (a_1, \ldots, a_{d-1}, a'_d)$.

Figure 9.1 is an example how the backtracking algorithm works in practice.

For example, the partial permutation $(1, 3)$ in Figure 9.1 was "bad", i.e. columns 1 and 3 of the matrix could not be changed to columns 1 and 2 of the original matrix. Thus we do not need to check the permutations that start with $(1, 3)$. See Figure 9.2 for an implementation of the algorithm in pseudo-code.

Obviously the efficiency of the algorithm depends heavily on how quickly the function "Test" (see Figure 9.2) can decide whether or not a partial permutation is good or bad.

We shall now give a brief description of our implementation of the function "Test". We make use of the fact that the generator matrix M of the linear code C is in standard form, i.e. that the first k columns of M form the identity matrix I_k. In other words $I_k = (c_1, \ldots, c_k)$.

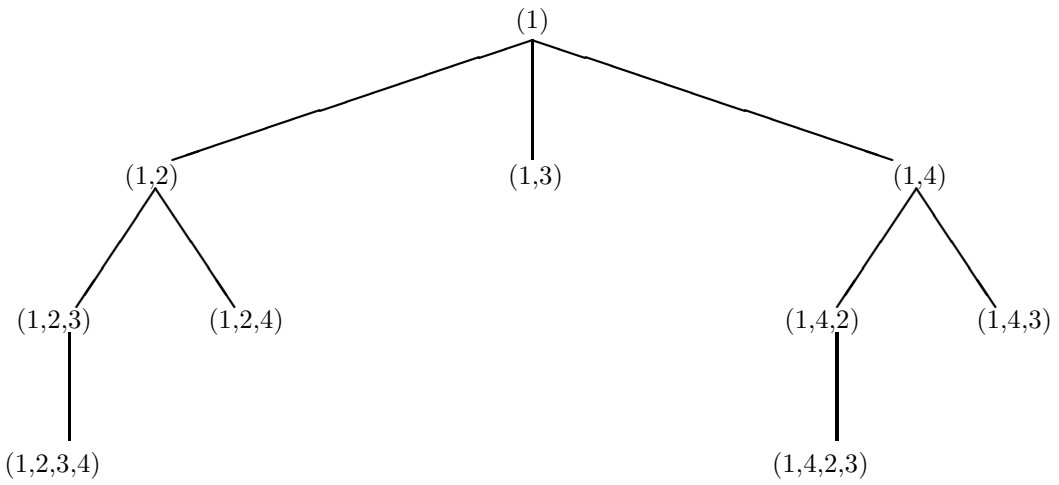


Figure 9.1: A typical search tree with pruning

To decide whether a partial permutation $\pi_d = (a_1, \ldots, a_d)$, where $d \leq k$ is "good", we simply reduce the matrix N_d to REF using Gaussian Elimination and check whether it agrees with M_d.

Suppose now that we have found k linearly independent columns, i.e. the partial permutation $\pi_k = (a_1, \ldots a_k)$ is "good". Let E be the matrix representing all the elementary row operations that change N_k into I_k. Deciding whether an extension of π_k is "good" is now very easy. Assume we extend π_k to $\pi_{k+1} = (a_1, \ldots, a_k, a_{k+1})$ then π_{k+1} is "good" provided that $Ec_{a_{k+1}} = c_{k+1}$. So in summary our implementation of "Test" does the following:
Suppose $\pi_d = (a_1, \ldots, a_d)$ is a partial permutation;

(1) if $d \leq k$ then reduce $N_d = (c_{a_1}, \ldots, c_{a_d})$ to REF and check whether it is equal to $M_d = (c_1, \ldots, c_d)$. If it is then the partial permutation is good. When $d = k$ and the partial permutation is good then "Test" keeps track of the elementary row operations used representing them by a matrix E.

(2) if $d > k$ then the partial permutation is good if $Ec_{a_d} = c_d$. Note that we assume here that the partial permutation $\pi_{d-1} = (a_1, \ldots, a_{d-1})$ was good.

Unfortunately, this approach of implementing "Test" is far from being fast. Even the C++ implementation is excruciatingly slow as soon as the dimension of the code is higher than 6. The reason for this will be explained later on. It is, however, very easy to encode.

Nevertheless our program provided enough data to formulate a hypothesis which was then turned into provable theorems. Figure 9.3 is a table of the running time of the C++ implementation of the algorithm for codes of dimension 4 to 7 and length from 16 to 24 (a dash, "-", indicates

$d = 1, \beta[1] = 0, checkflag[0..n] = 0$
WHILE $d > 0$ *do*
  REPEAT
    $\beta[d] = succ(\beta[d])$
  UNTIL $(checkflag[\beta[d]] = 0)$ **OR** $(\beta[d] = 0)$
  IF $(\beta[d] \neq 0)$ **THEN**
    REM $(\beta[1], \ldots, \beta[d])$ is a partial permutation
    $checkflag[\beta[d]] = 1$
    IF $(d < n)$ **THEN**
      IF $Test([\beta[1], \ldots, \beta[d])$ **THEN**
        REM partial permutation is ok
        $d = d + 1,\ \beta[d] = 0$
      ELSE
        REM partial permutation is bad
        $checkflag[\beta[d]] = 0$
      FI
    ELSE
      IF $Test([\beta[1], \ldots, \beta[n]])$ **THEN**
        REM we found an automorphism
        add $[\beta[1], \ldots, \beta[n]]$ to our list
      FI
      $checkflag[\beta[n]] = 0$
      $d = d - 1$
      $checkflag[\beta[d]] = 0$
    FI
  ELSE
    REM no more partial permutations
    $d = d - 1$
    $checkflag[\beta[d]] = 0$
  FI
OD

Figure 9.2: Pseudo-code for the backtracking algorithm due to Leon [Leo84]

that no timing was taken as it was obvious that it would take too long to run; the time format is $h\colon mm\colon ss$).

len/dim	16	18	20	22	24
4	0:00:08	0:00:13	0:00:23	0:00:37	0:00:58
5	0:01:41	0:03:34	0:06:54	0:12:31	0:21:32
6	0:22:04	0:54:45	-	-	-
7	4:19:27	-	-	-	-

Figure 9.3: Table of running times of the backtracking algorithm

The downfall of the backtracking algorithm is its generality. It can be used to determine the automorphism group of any $k \times n$ matrix of rank k. So why does it take so long?

The matrices we are interested in are generator matrices of linear codes which are in standard form, i.e. $M = (I_k, P)$. First of all we notice that the first k columns of our matrix are the columns of the identity matrix I_k. Thus as soon as we have found k linearly independent column vectors, we need to check all $k!$ permutations of them. Generally, one can say that picking k vectors at random usually means that they are also linearly independent. Because our generator matrix is a $k \times n$ matrix there are $\binom{n}{k}$ possible ways of choosing k columns.

So most of the time will be spent in the function "Test" reducing k random columns of M to I_k whilst keeping track of the elementary row operations used in a matrix, E say.

So once we have found a "good" partial permutation $\pi_k = (a_1, \dots, a_k)$ determining whether any of its extensions is "good" can be carried out very quickly. It only involves checking whether $Ec_{a_i} = c_i$ (for $k < i \leq n$). This will usually take negligible time because in general it will fail at the first column tested and we can backtrack.

Therefore we have the following estimates for the number of permutations we have to check:

$$\binom{n}{k} k!.$$

So for example taking $n = 16$ and $k = 4$ we find that we have to check roughly

$$\binom{16}{4} 4! = 43680$$

permutations. Increasing the length by 2 we have to check circa

$$\binom{18}{4} 4! = 73440$$

permutations, i.e. ≈ 1.68 times as many as before which corresponds to the time increase in the table. Similarly, increasing the dimension by 1 yields the following estimate of permutations that

need checking:

$$\binom{16}{5}5! = 524160 = 12\binom{16}{4}4!.$$

This time it is a 12-fold increase in permutations which tallies with the time listed in the table. This explains why the backtracking algorithm fails to be useful for codes of high dimension and length. As a final example we compare the number of permutations for $n = 16$ and $k = 6, 7$:

$$\binom{16}{7}7! = 57657600 = 10\binom{16}{6}6!$$

which roughly corresponds to the timings taken.

9.5 Examples

In this section we will illustrate how to use our MAPLE package in order to produce compute various Goppa codes and their groups of automorphisms. Most of the examples we are about to present are **not** covered by the theorems of the previous chapters. They are used to convince the reader that even though our theoretical results cover a broad range of geometric Goppa codes, they do not seem to be best possible.

9.5.1 The elliptic function field $F = \mathbb{F}_{17}(x, y)$ defined by $y^2 = x^3 - x$

Let $F/\mathbb{F}_{17}$ be the elliptic function field defined by $y^2 = x^3 - x$. We shall investigate various Goppa codes associated with this function field. The following is a short list of properties of F. We recall that a place of degree one in an elliptic function field is uniquely determined by its values at x and y. Hence we identify a place $P \in \mathbb{P}_F^{(1)}$ with $P = (x(P), y(P))$. Furthermore, recall that for all $P \in \mathbb{P}_F^{(1)}$ the translation $\tau_Q(P) = P \oplus Q$, where $Q \in \mathbb{P}_F^{(1)}$ is fixed, yields an automorphism of F.

- $x^3 - x = x(x-1)(x+1)$ and hence F has 4 ramified places over $\mathbb{F}_{17}(x)$. We denote them by Q_∞ (the common pole of x and y), $P_1 = (0, 0), P_2 = (1, 0)$ and $P_3 = (-1, 0)$.
- $|\mathbb{P}_F^{(1)}| = 16$, i.e. F has 12 places that are split over $\mathbb{F}_{17}(x)$. Denote them by

$$P_4 = (4, 3) \quad P_5 = (4, 14) \quad P_6 = (5, 1) \quad P_7 = (5, 16) \quad P_8 = (7, 8) \quad P_9 = (7, 9)$$
$$P_{10} = (10, 2) \quad P_{11} = (10, 15) \quad P_{12} = (12, 4) \quad P_{13} = (12, 13) \quad P_{14} = (13, 5) \quad P_{15} = (13, 12).$$

- There are 4 Q_∞-fixing automorphisms of F given by $< \alpha >$, where

$$\alpha : \begin{cases} x & \mapsto -x \\ y & \mapsto 4y \end{cases}$$

- Set $D = \sum_{i=4}^{15} P_i$ and let $G = k_0 Q_\infty + k_1 P_1 + k_2 P_2 + k_3 P_3$, where $k_i \geq 0$ $(i = 0, \ldots, 3)$.

Before we investigate the automorphism group of codes $\mathcal{C}_\mathcal{L}(D, G)$ (with divisors D and G defined as above), we take a look at the dual code $\mathcal{C}_\mathcal{L}(D, G)^\perp$.

Lemma 9.5.1. *Let D be as above. If $G = k_0 Q_\infty + k_1 P_1 + k_2 P_2 + k_3 P_3$, where $k_i \geq 0$ and $1 \leq \deg G \leq 11$ then*

$$\mathcal{C}_\mathcal{L}(D, G)^\perp = \mathbf{a}\mathcal{C}_\mathcal{L}(D, G'), \quad \text{where} \begin{cases} G' = 3(Q_\infty + P_1 + P_2 + P_3) - G, \\ \mathbf{a} = (u(P_4), \ldots, u(P_{15})), \text{ and} \\ u = \dfrac{(x-1)(x+1)}{(x^2+8x+2)(x^2+9x+2)}. \end{cases}$$

Observe that $u(P_i) \neq 0$ $(4 \leq i \leq 15)$. In other words the dual code of $\mathcal{C}_\mathcal{L}(D, G)$ is given by multiplying each code word of $\mathcal{C}_\mathcal{L}(D, G')$ by the vector $\mathbf{a}$.

Proof. We use Proposition VII.1.2[†] in [Sti91, p.205]. Putting

$$t = (x-4)(x-5)(x-7)(x-10)(x-12)(x-13) = x^6 + 12x^4 + 12x^2 + 1,$$

we find that $(t) = D - 12Q_\infty$ and $dt = (6x^5 + 14x^3 + 7x)dx$. Therefore

$$\begin{aligned} (dt) &= (x) + (x^2 + 8x + 2) + (x^2 + 9x + 2) + (dx) \\ &= 3P_1 + P_2 + P_3 + (x^2 + 8x + 2)_0 + (x^2 + 9x + 2)_0 - 13Q_\infty. \end{aligned}$$

Now by Proposition VII.1.2 in [Sti91, p.205] $\mathcal{C}_\mathcal{L}(D, G)^\perp = \mathcal{C}_\mathcal{L}(D, H)$, where $H = D - G + (dt) - (t)$ and we find that

$$\begin{aligned} H &= D - G + (dt) - (t) \\ &= (x^2 + 8 + 2)_0 + (x^2 + 9x + 2)_0 - (k_1 - 3)P_1 - (k_2 - 1)P_2 - (k_3 - 1)P_3 - (k_0 + 1)Q_\infty \\ &= \left(\frac{(x^2 + 8 + 2)(x^2 + 9x + 2)}{(x-1)(x+1)} \right) + (3 - k_0)Q_\infty + (3 - k_1)P_1 + (3 - k_2)P_2 + (3 - k_3)P_3 \\ &= -(u) + 3(Q_\infty + P_1 + P_2 + P_3) - G, \end{aligned}$$

where $u = \frac{(x-1)(x+1)}{(x^2+8+2)(x^2+9x+2)}$. Setting $G' = 3(Q_\infty + P_1 + P_2 + P_3) - G$ we find that

$$\mathcal{L}(G') \cong \mathcal{L}(H) \quad \text{where the isomorphism is given by } \phi : \begin{cases} \mathcal{L}(G') & \to \mathcal{L}(H) \\ z & \mapsto uz \end{cases}.$$

[†]Actually, the version we use is slightly simpler:

Proposition. *Let t be an element of F such that $v_{P_i}(t) = 1$ for $i = 1, \ldots, n$. Then the following holds.*

(a) *The differential $\eta := dt/t$ satisfies $\eta_{P_i}(1) = 1$ for $i = 1, \ldots, n$.*

(b) $\mathcal{C}_\mathcal{L}(D, G)^\perp = \mathcal{C}_\mathcal{L}(D, D - G + (dt) - (t))$.

Observe that $u(P_i) \neq 0$, in fact

$$\mathbf{a} = (u(P_4), \ldots, u(P_{15})) = (12, 12, 11, 11, 11, 11, 11, 11, 11, 11, 12, 12).$$

The result now follows. □

Remark. In the above example, it is is obvious that that $Aut_{D,G}(F/\mathbb{F}_q) = Aut_{D,G'}(F/\mathbb{F}_q)$. However, it is not true that $Aut_{D,G}(F/\mathbb{F}_q = Aut_{D,H}(F/\mathbb{F}_q)$ as can be easy seen by applying the automorphism

$$\sigma : \begin{cases} x \mapsto \frac{1}{x} \\ y \mapsto y \end{cases}$$

to the divisor H. Nevertheless, one checks easily that for all $\sigma \in Aut_{D,G}(F/\mathbb{F}_{17})$ we have that $\sigma(H) = H + (w)$, where (w) is a principal divisor such that $(w)(P) = 1$ for all $P \in \text{supp}(D)$. Hence we see that in this case we have that

$$Aut_{D,G}(F/\mathbb{F}_{17}) = Aut'_{D,H}(F/\mathbb{F}_{17}),$$

where $Aut'_{D,H}(F/\mathbb{F}_{17})$ is defined as in Remark 4.2.1.

Recall that by multiplying each coordinate of a code by a non-zero element of the ground field we obtain a so called equivalent code (cf. Section 1.5, note that here we use the extended definition of equivalence). It is obvious that equivalent codes have the same dimension and minimum distance. However, in general, they do not have isomorphic automorphism groups. Nevertheless, with the above set-up we see immediately that $Aut_{D,G}(F/\mathbb{F}_{17}) = Aut_{D,G'}(F/\mathbb{F}_{17})$ and the next examples show that our equivalent codes do have the same group of automorphisms.

Assume first that $G = m(Q_\infty + P_1 + P_2 + P_3)$, $m = 1, 2$ we find that

$$Aut_{D,G}(F/\mathbb{F}_{17}) =< \alpha^i \circ \tau_Q \mid \ 0 \leq i \leq 3, \text{ and } Q \in \{Q_\infty, P_1, P_2, P_3\} >,$$

i.e. $|Aut_{D,G}(F/\mathbb{F}_{17})| = 16$.

The following list of examples has been computed using our MAPLE program. The print-outs of the sessions in which these codes were computed can be found in Section C.1. (To avoid excessive waste of paper we only included a selection of these codes. The reader who wants to check all of them can run the short routine at the beginning of the print-outs).

Example 9.5.2. Let $G = m(Q_\infty + P_1 + P_2 + P_3)$ (with $m = 1, 2$) and D be as above then $\mathcal{C}_\mathcal{L}(D, G)$ is a code of length $\deg D = 12$, dimension $m \deg G = 4m$ and

$$Aut_{D,G}(F/\mathbb{F}_{17}) \cong Aut(\mathcal{C}_\mathcal{L}(D, G)).$$

Because $|Aut(\mathcal{C}_\mathcal{L}(D, G))| = 16$, a trivial consequence is that $\mathcal{C}_\mathcal{L}(D, G)$ is not cyclic.

When $G = Q_\infty + P_1 + P_2 + P_3$, then it is easy to see that $r = y/(x-1)$ and $s = y/(x-16)$ are both in $\mathcal{L}(G)$. Furthermore, $x = rs$ and $y = r^2s - r$, i.e. $F = \mathbb{F}_{17}(r,s)$, where $r^2s - r + rs^2 + s = 0$ is the defining equation of F over $\mathbb{F}_{17}(r)$. However to be able to construct an automorphism of the function field we would need at least 13 places in the support of D.

When $G = 2(Q_\infty + P_1 + P_2 + P_3)$ then x and $r = y/x$ are in $\mathcal{L}(G)$ and obviously also generate F. This time $F = \mathbb{F}_{17}(x,r)$ where r satisfies $r^2x - x^2 + 1 = 0$. Moreover, $r^2 \in \mathcal{L}(G)$ as well. However, the methods we have developed in previous chapters need at least 17 places of degree one in the support of D to be able to construct an automorphism of F. Using Lemma 9.5.1 we compute the dual code and see that $\mathcal{C}_\mathcal{L}(D,G)^\perp = \mathbf{a}\mathcal{C}_\mathcal{L}(D,(Q_\infty + P_1 + P_2 + P_3))$. However, we do not seem to be able to exploit this nice symmetry.

Example 9.5.3. We have computed all the groups of automorphisms of codes with divisor D as above and where $\operatorname{supp} G \subseteq \{Q_\infty, P_1, P_2, P_3\}$ and $3 \leq \deg G \leq 9$, most of which do not satisfy the conditions stipulated in the theorems dealing with (hyper-)elliptic codes (cf. Chapter 5), however in each case we found that $Aut_{D,G}(F/\mathbb{F}_{17}) \cong Aut(\mathcal{C}_\mathcal{L}(D,G)) = Aut(\mathcal{C}_\mathcal{L}(D,G)^\perp) \cong Aut_{D,G'}(F/\mathbb{F}_{17})$, where $G' = 3(Q_\infty + P_1 + P_2 + P_3) - G$.

When $\deg G = 10$ (resp. $\deg G = 11$) then we find that $\mathcal{C}_\mathcal{L}(D,G)$ can have many more automorphisms. Let $\mathbf{a} = (u(P_4), \ldots, u(P_{15}))$, where $u = ((x-1)(x+1))/((x^2+8x+2)(x^2+9x+2))$, thus $\mathbf{a} = (12,12,11,11,11,11,11,11,11,11,12,12)$.

Example 9.5.4. Let $G = 3Q_\infty + 3P_1 + 3P_2 + P_3$ and D as before, then

$$|Aut_{D,G}(F/\mathbb{F}_{17})| = 4 \quad \text{but} \quad |Aut(\mathcal{C}_\mathcal{L}(D,G))| = 128.$$

This was computed by looking at the dual code of $\mathcal{C}_\mathcal{L}(D,G)$. We know from Lemma 9.5.1 that the dual code of $\mathcal{C}_\mathcal{L}(D,G)$ is given by $\mathbf{a}\mathcal{C}_\mathcal{L}(D,H)$, where $H = 2P_3$. Therefore we have

$$\mathcal{L}(H) = u\langle 1, \frac{1}{(x+1)}\rangle.$$

This gives rise to the following generator matrix of $\mathcal{C}_\mathcal{L}(D,G)^\perp$

$$\mathbf{a}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 15 & 15 & 11 & 11 & 5 & 5 & 16 & 16 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 & 7 & 7 & 5 & 5 & 2 & 2 & 2 & 2 \\ 0 & 0 & 1 & 1 & 15 & 15 & 11 & 11 & 5 & 5 & 2 & 2 \end{pmatrix}$$

from which one can read off 64 permutations immediately (one of which is given by the field automorphism α^2). Hence, together with the 4 field automorphisms that fix D and G, we find all 128 automorphisms of the code.

The next example, however, shows that we can still have $Aut_{D,G}(F/\mathbb{F}_{17}) \cong Aut(\mathcal{C}_{\mathcal{L}}(D,G))$ despite $\deg G = 10$.

Example 9.5.5. Let $G = 4Q_\infty + 2P_1 + 2P_2 + 2P_3$ and D as before, then

$$Aut_{D,G}(F/\mathbb{F}_{17}) \cong Aut(\mathcal{C}_{\mathcal{L}}(D,G)) \quad \text{and} \quad |Aut(\mathcal{C}_{\mathcal{L}}(D,G))| = 4.$$

Again, the result was computed using the MAPLE program and actually looking at the dual of $\mathcal{C}_{\mathcal{L}}(D,G)$. In this case, we find that $\mathcal{C}_{\mathcal{L}}(D,G)^\perp = \mathbf{a}\mathcal{C}_{\mathcal{L}}(D,H)$, where

$$H = -Q_\infty + P_1 + P_2 + P_3$$
$$\mathcal{L}(H) = \langle \frac{1}{y}, \frac{x}{y} \rangle.$$

This time the generator matrix does not allow one to read off as many automorphisms as before:

$$\mathbf{a}\begin{pmatrix} 6 & 11 & 1 & 16 & 15 & 2 & 9 & 8 & 13 & 4 & 7 & 10 \\ 7 & 10 & 5 & 12 & 3 & 14 & 5 & 12 & 3 & 14 & 6 & 11 \end{pmatrix}$$
$$= \begin{pmatrix} 1 & 16 & 0 & 0 & 11 & 6 & 8 & 9 & 9 & 2 & 8 & 15 \\ 0 & 0 & 1 & 16 & 11 & 6 & 3 & 2 & 14 & 10 & 15 & 7 \end{pmatrix}.$$

Remark. The most obvious difference between the last two examples is that in the last one the two elements in $\mathcal{L}(H)$ are generators of the function field, whereas in the previous example they were not. So in the last case one could argue that it is conceivable that using the general idea of the methods described in the previous chapters (i.e. constructing an automorphism of the function field from a code automorphism by looking at what happens to the generators of the function field under the map λ associated with the automorphism of the code) this could still be carried out here but not in the case of Example 9.5.4 where there is only one generator.

As it turns out this function field also provides us with an example showing that even Xing's improved result (cf. Theorem 7.4.1) does not seem to be best possible.

We shall now look at the code $\mathcal{C}_{\mathcal{L}}(D,G)$ of $F/\mathbb{F}_{17}$, where

$$D = \sum_{i=1}^{15} P_i \quad \text{and} \quad G = mQ_\infty, \quad \text{for } 3 \le m \le 12. \tag{9.5.6}$$

We summarise the properties of these codes in the following lemma.

Lemma 9.5.7. *Let $\mathcal{C}_{\mathcal{L}}(D,G)$ be the code given by divisors D and G as in* (9.5.6)*. Then*

(a) $\dim G = \dim \mathcal{C}_{\mathcal{L}}(D,G) = m$;

(b) $\mathcal{C}_{\mathcal{L}}(D,G)^\perp = \mathcal{C}_{\mathcal{L}}(D,H)$ *where H is given by*

$$H = (x^4 + 8x^3 + 15x^2 + 14x + 14) + (x^4 + 9x^3 + 15x^2 + 3x + 14) + (15 - m)Q_\infty;$$

(c) $\mathcal{L}(H) = u\mathcal{L}((15-m)Q_\infty)$, *where*

$$u = 1/((x^4+8x^3+15x^2+14x+14)(x^4+9x^3+15x^2+3x+14)), \quad \textit{i.e.}$$
$$\mathcal{C}_\mathcal{L}(D, mQ_\infty)^\perp = \mathbf{a}\mathcal{C}_\mathcal{L}(D,(15-m))Q_\infty),$$

where $\mathbf{a} = (u(P_1), u(P_2), \ldots u(P_{15})) = (2,15,15,9,9,1,1,16,16,16,16,1,1,9,9)$*;*

(d) *For* $3 \leq m \leq 12$*, we have* $Aut_{D,G} \cong Aut(\mathcal{C}_\mathcal{L}(D,G))$.

Proof. (a) is clear and (c) follows directly from (b). For (b) we use once more Proposition VII.1.2 in [Sti91, p.205]. Putting

$$t = y(x-4)(x-5)(x-7)(x-10)(x-12)(x-13) = y(x^6+12x^4+12x^2+1),$$

we find that

$$(t) = D - 15Q_\infty,$$
$$dt = \frac{15}{2y}(x^4+8x^3+15x^2+14x+14)(x^4+9x^3+15x^2+3x+14)dx.$$

Because $H = D - G + (dt) - (t)$ and $(y) = (dx)$ in this set-up, the result now follows.

(d) follows from Theorem 7.4.1 for $3 \leq m \leq 10$ The remaining 2 cases were dealt with by computing the automorphism group explicitly using my MAPLE program (see Appendix C for the print-outs). □

In the above set-up Xing's improved result only holds for $3 \leq m \leq 10$, however, the result is still true for two more cases. Obviously, computing just one example is not enough. After having worked out numerous examples the result which we reckon might be true is the following:

Conjecture 9.5.8. *Let* $F/\mathbb{F}_q$ *be an elliptic function field. Let* $D = \sum_{P \in \mathbb{P}_F^{(1)} \setminus \{Q_\infty\}} P$*. Assume that* $\deg D \geq 6$*; then*

$$Aut_{D,mQ_\infty}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D, mQ_\infty)), \quad \textit{for } 3 \leq m \leq \deg D - 3.$$

The author is not aware of any counterexamples to the above conjecture.

9.5.2 Some hyperelliptic function fields associated with $y^2 = x^5 - x$

First of we shall study some hyperelliptic function fields $F = \mathbb{F}_q(x,y)$, where $y^2 = x^5 - x$ and $q \equiv 1 \pmod 8$. The following are some facts which can be easily verified:

- F has genus 2.

- $x^5 - x = x(x-1)(x+1)(x-\alpha)(x+\alpha)$, where $\alpha^2 = -1$ and hence F has 6 ramified places over $\mathbb{F}_q(x)$. We denote them by Q_∞ (the common pole of x and y), $P_1 = (0,0)$, $P_2 = (1,0)$, $P_3 = (-1,0)$, $P_4 = (\alpha,0)$ and $P_5 = (-\alpha,0)$.
- There are 8 Q_∞-fixing automorphisms of F given by $< \epsilon >$, where

$$\epsilon \begin{cases} x \mapsto \zeta^2 x \\ y \mapsto \zeta^5 y \end{cases}$$

  where ζ is an 8-th root of unity.

Suppose that $|\mathbb{P}_F^{(1)}| = n+1$ then F has $n-5$ places that are split over $\mathbb{F}_q(x)$. We denote them by $P_i = (x_i, y_i)$, where $6 \le i \le n$ and $P_{2k} = \overline{P}_{2k+1}$ for $3 \le k \le \frac{n-1}{2}$ (Note that $\overline{P}$ denotes the conjugate of P under the non-trivial $\mathbb{F}_q(x)$-isomorphism of F).

The codes we investigate first are the ones associated to the divisors

$$D = \sum_{i=6}^{n} P_i, \qquad \text{and} \qquad G = k_0 Q_\infty + \sum_{i=1}^{5} k_i P_i,$$

where $0 < G$ and $6 \le \deg G \le n-9$. Note that $\deg D = n-5$ and $5 \le \dim G \le n-10$.

First of all we look at the relation between $\mathcal{C}_\mathcal{L}(D,G)$ and its dual. To obtain a nice symmetrical result we have to assume that $n \equiv 3 \pmod 6$.

Lemma 9.5.9. *With D and G as above. Suppose that $n \equiv 3 \pmod 6$. Then*

$$\mathcal{C}_\mathcal{L}(D,G)^\perp = \mathbf{a}\mathcal{C}_\mathcal{L}(D,G'), \qquad \textit{where} \begin{cases} G' = (n-3)/6(Q_\infty + \sum_{i=1}^{5} P_i) - G, \\ \mathbf{a} = (u(P_6), \ldots, u(P_n)), \ \textit{and} \\ u \in \mathbb{F}_q(x). \end{cases}$$

Proof. The proof is analogous to the proof of Lemma 9.5.1. Again we use Proposition V.II.1.2. in [Sti91, p. 205]. Set

$$t := \prod_{i=3}^{\frac{n-1}{2}} (x - x(P_{2i})) = r(x), \qquad \text{hence} \qquad (t) = D - (n-5)Q_\infty.$$

Now $(dt) = (r'(x)) + (dx)$, where $r'(x)$ is the derivative of $r(x)$ and $(dx) = \sum_{i=1}^{5} -3Q_\infty$. Hence

$$\begin{aligned} H &= -G + (r'(x)) + (dx) + (n-5)Q_\infty \\ &= (n-8)Q_\infty + (\sum_{i=1}^{5} P_i) - G + (r'(x)) \\ &= \frac{n-3}{6}(Q_\infty + \sum_{i=1}^{5} P_i) - G - \frac{n-9}{6}(y) + (r'(x)). \end{aligned}$$

Thus

$$\mathcal{C}_{\mathcal{L}}(D,G)^{\perp} = \mathcal{C}_{\mathcal{L}}(D,H) = \mathbf{a}\mathcal{C}_{\mathcal{L}}(D,G'),$$

where $\mathbf{a} = (u(P_6), \ldots, u(P_n))$ and $u = y^{\frac{n-9}{6}}/r'(x)$. □

Example 9.5.10. Let $F = \mathbb{F}_{73}(x,y)$ be the hyperelliptic function field given by $y^2 = x^5 - x$. It is easy to verify that $|\mathbb{P}_F^{(1)}| = 70$, i.e. $n = 69 \equiv 3 \pmod 6$. Using the same notation as above it now follows that

$$\mathcal{C}_{\mathcal{L}}(D,G)^{\perp} = \mathbf{a}\mathcal{C}_{\mathcal{L}}(D,G'), \qquad \text{where} \begin{cases} G' = 11(Q_\infty + \sum_{i=1}^{5} P_i) - G, \\ \mathbf{a} = (u(P_6), \ldots, u(P_{69})), \text{ and} \\ u \in \mathbb{F}_q(x). \end{cases}$$

More explicitly, u is given by $u = y^{10}/(r_1(x)r_2(x)r_3(x))$, where

$$\begin{aligned} r_1(x) &= \left(x^2+14\right)\left(x^2+59\right)\left(x^4+34x^3+22x^2+8x+57\right) \\ r_2(x) &= \left(x^4+35x^2+43\right)\left(x^4+31x^3+51x^2+70x+57\right)\left(x^4+39x^3+22x^2+65x+57\right) \\ r_3(x) &= \left(x^4+42x^3+51x^2+3x+57\right)\left(x^4+38x^2+43\right)x^3 \end{aligned}$$

and $\mathbf{a}$ is given by

$$\begin{aligned} \mathbf{a} = (\ & 27\ 27\ 27\ 27\ 1\ 1\ 46\ 46\ 1\ 1\ 27\ 27\ 46\ 46\ 72\ 72 \\ & 27\ 27\ 46\ 46\ 46\ 46\ 46\ 46\ 27\ 27\ 27\ 27\ 72\ 72\ 46\ 46 \\ & 46\ 46\ 72\ 72\ 27\ 27\ 27\ 27\ 46\ 46\ 46\ 46\ 46\ 46\ 27\ 27 \\ & 72\ 72\ 46\ 46\ 27\ 27\ 1\ 1\ 46\ 46\ 1\ 1\ 27\ 27\ 27\ 27\ ) \end{aligned}$$

We will now show that we can determine the automorphism group of the following class of codes.

Lemma 9.5.11. *Let $F = \mathbb{F}_q(x,y)$ be the hyperelliptic function field given by $y^2 = x^5 - x$. With the notation as above , let $G = k(Q_\infty + \sum_{i=1}^{5} P_i)$ and $D = \sum_{P \in J} P$ where $J \subseteq \mathbb{P}_F^{(1)} \setminus \{Q_\infty, P_1, \ldots, P_5\}$. If $\deg D > \max\{2\deg G, 60\}$ then*

$$Aut_{D,G}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_{\mathcal{L}}(D,G)).$$

Proof. Obviously, if $k \geq 6$ then Theorem 5.2.8 applies. So the only cases we have to check are when $1 \leq k \leq 5$. We shall sketch the proof for each case.

Proposition 3.3.4 shows that $Aut_{D,G}(F/\mathbb{F}_q)$ is isomorphic to a subgroup of $Aut(\mathcal{C}_{\mathcal{L}}(D,kG))$. We need to show that every automorphism of the code is actually induced by an automorphism in $Aut_{D,G}(F/\mathbb{F}_q)$. So let $\pi \in Aut(\mathcal{C}_{\mathcal{L}}(D,kG))$. Consider the map λ_π from $\mathcal{L}(kG)$ onto itself as described in Definition 4.3.6. For simplicity we refer to λ_π as λ.

$k = 1$:

It is easy to verify that a basis of $\mathcal{L}(G)$ is given

$$B_{1G} = \left\{1, \frac{y}{x^2-1}, \frac{y}{x^2+1}, \frac{y}{x(x-1)}, \frac{1}{y}\right\}.$$

Put $r = y/(x^2-1)$ and $s = y/(x^2+1)$. Note that $rs = x$ and $r(r^2s^2-1) = y$. Thus we find that $F = \mathbb{F}_q(r,s)$ with

$$r^3s^2 - r^2s^3 - r - s = 0. \tag{9.5.12}$$

As usual we need to show that λ preserves the defining equation (9.5.12) and acts multiplicatively on the other elements of the basis.

We observe that

$$\frac{y}{x(x-1)} = \frac{rs+1}{s}.$$

Thus by Lemma 4.3.5 $\lambda(s)\lambda(y/x(x-1)) = \lambda(r)\lambda(s)+1$ provided that $\deg D > 12$ and hence $\lambda(y/x(x-1)) = (\lambda(r)\lambda(s)+1)/\lambda(s)$. Similarly

$$\frac{1}{y} = \frac{1}{r(rs-1)(rs+1)}.$$

So provided that $\deg G > 37$ Lemma 4.3.5 yields that $\lambda(1/y) = 1/(\lambda(r)(\lambda(r)\lambda(s)-1)(\lambda(r)\lambda(s)+1))$. Lastly we check that λ preserves the defining equation (9.5.12). This follows again from Lemma 4.3.5 provided that $\deg D > 30$.

Now by Theorem 4.3.1 and the fact that r and s are in the image of λ we deduce that λ gives rise to an automorphism, $\widetilde{\lambda}$ of the function field. By Lemma 4.3.4 and Lemma 4.3.10 we can now deduce that $\widetilde{\lambda} \in Aut_{D,G}(F/\mathbb{F}_q)$ and because $\widetilde{\lambda}|_{\mathcal{L}(G)} = \lambda$, it induces the code automorphism π we started off with.

In the remaining cases we shall just indicate how to prove that λ acts multiplicatively on $\mathcal{L}(kG)$ and that it preserves the defining equation of F. Once this has been established the fact that λ is induced by an automorphism of the function field follows from the argument given above.

$k = 2$:

This time a basis for $\mathcal{L}(2G)$ is given by

$$B_{2G} = B_{1G} \cup \left\{x, \frac{1}{x}, \frac{1}{x-1}, \frac{1}{x+1}, \frac{1}{x-\alpha}, \frac{1}{x+\alpha}\right\}.$$

This time we put $r = y/(x^2-1)$ and $s = x$. From this it is obvious that $F = \mathbb{F}_q(r,s)$. The defining equation of F in terms of r and s is

$$r^2s^2 - r^2 - s^3 - s = 0. \tag{9.5.13}$$

Now, as before, we need to check that λ acts multiplicatively on the basis elements and that it preserves (9.5.13). This is very straightforward and is left to the reader. The only thing which is worthwhile pointing out is that $r^2 \in \mathcal{L}(2G)$ and thus we can deduce by Lemma 4.3.9 that $\deg(\lambda(r))_\infty \leq 6$ which will reduce the number of places required in the calculations.

$k = 3$:

This time a basis for $\mathcal{L}(3G)$ is given by

$$B_{3G} = B_{2G} \cup \left\{ \frac{y}{x}, \frac{y}{x^2}, \frac{y}{(x-1)^2}, \frac{y}{(x+1)^2}, \frac{y}{(x-\alpha)^2}, \frac{y}{(x+\alpha)^2} \right\}.$$

Again we set $r = y/(x^2-1)$ and $s = x$. Because $r^3 \in \mathcal{L}(3G)$ we have that $\deg(\lambda(r))_\infty \leq 6$. Now the reader who went through the calculations for the case $k = 2$ knows that

$$\frac{1}{y} = \frac{1}{r(s^2-1)} \qquad \text{and hence} \qquad r(s^2-1)\frac{1}{y} = 1.$$

In order to deduce that $\lambda(r)(\lambda(s)^2-1)\lambda(1/y) = 1$ we need $\deg D > 6+18+18+18 = 60$, i.e. this explains the condition imposed on D in the statement of this lemma. The other elements and the defining equation can be dealt with as before.

$k = 4$:

A basis of $\mathcal{L}(4G)$ is given by

$$B_{4G} = B_{3G} \cup \left\{ x^2, \frac{1}{x^2}, \frac{1}{(x-1)^2}, \frac{1}{(x+1)^2}, \frac{1}{(x-\alpha)^2}, \frac{1}{(x+\alpha)^2} \right\}.$$

As before we set $r = y/(x^2-1)$ and $s = x$. Now $r^4, s^2 \in \mathcal{L}(4G)$ and hence $\deg(\lambda(r))_\infty \leq 6$ and $\deg(\lambda(s))_\infty \leq 12$. The reader is left to check the details.

$k = 5$:

Finally, a basis of $\mathcal{L}(5G)$ is given by

$$B_{5G} = B_{4G} \cup \left\{ y, \frac{y}{x^3}, \frac{y}{(x-1)^3}, \frac{y}{(x+1)^3}, \frac{y}{(x-\alpha)^3}, \frac{y}{(x+\alpha)^3} \right\}.$$

Again, we set $r = y/(x^2-1)$ and $s = x$ and this time we have $\deg(\lambda(r))_\infty \leq 6$ and $\deg(\lambda(s))_\infty \leq 15$. As before it is left to the reader to check that λ acts multiplicatively on $\mathcal{L}(5G)$ and that it preserves (9.5.13). This concludes the proof. □

Example 9.5.14. We have another look at the code defined in Example 9.5.10. Set $G = Q_\infty + \sum_{i=1}^{5} P_i$. Now Lemma 9.5.11 shows that

$$Aut_{D,kG}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D, kG)) \qquad \text{for } 1 \leq k \leq 5.$$

Furthermore, the function "FindAutGD" of our MAPLE program yields that $|Aut_{G,D}(F/\mathbb{F}_{73})| = 48$.

Looking at the automorphism which were computed in the above example one can then deduce the following lemma.

Lemma 9.5.15. *Let $F = \mathbb{F}_q(x,y)$ where $y^2 = x^5 - x$, $q \equiv 1 \pmod 8$. With the notation of Lemma 9.5.9, set $G = Q_\infty + \sum_{i=1}^{5} P_i$ and $D = \sum_{i=6}^{n}$. Then 48 divides the order of $Aut_{G,D}(F/\mathbb{F}_q)$.*

Proof. Obviously, $\langle \epsilon \rangle$ is a subgroup of $Aut_{D,G}(F/\mathbb{F}_q)$ of order 8. Similarly it is obvious that $\langle \sigma_1 \rangle$, where $\sigma_1(x) = 1/x$ and $\sigma_1(y) = y/x^3$, is a subgroup of order 2. Recall that ζ is an 8-th root of unity. Set $\beta = \zeta + \zeta^{-1}$, then $\beta^2 = 2$. We also have that $\alpha = \zeta^2$. We will show that

$$\langle \sigma_2 \rangle, \qquad \text{where } \sigma_2 : \begin{cases} x \mapsto \frac{x+1}{\alpha x - \alpha} \\ y \mapsto \frac{\beta^3 \zeta y}{(\alpha x - \alpha)^3} \end{cases}$$

is a subgroup of $Aut_{D,G}(F/\mathbb{F}_q)$ of order 3.

First of all we observe that σ_2 is indeed an automorphism of the function field. Applying σ_2 to $y^2 = x^5 - x$ yields

$$\begin{aligned}
& \sigma_2(y)^2 = \sigma_2(x)^5 - \sigma_2(x) \\
\iff \quad & \left(\frac{\beta^3\zeta y}{(\alpha x - \alpha)^3}\right)^2 = \left(\frac{x+1}{\alpha x - \alpha}\right)^5 - \left(\frac{x+1}{\alpha x - \alpha}\right) \\
\iff \quad & 8\alpha y^2 = (\alpha x - \alpha)(x+1)^5 - (\alpha x - \alpha)^5 (x+1) \\
\iff \quad & 8\alpha y^2 = 8\alpha x^5 - 8\alpha x \\
\iff \quad & y^2 = x^5 - x.
\end{aligned}$$

Furthermore, a simple but tedious calculation shows that $\sigma_2^3(x) = x$ and $\sigma_2^3(y) = y$. It is now straightforward to see that $\langle \epsilon, \sigma_1, \sigma_2 \rangle$ is a subgroup of order 48. □

We see that these codes have a fairly large group of automorphisms. If $q = 5^{2r}$, $r > 0$ then we have an additional automorphism σ_3 of order 5 with $\sigma_3(x) = x + 1$, $\sigma_3(y) = y$. Hence 240 divides $|Aut_{D,G}(F/\mathbb{F}_q)|$ in this case.

9.5.3 The Klein Quartic over $\mathbb{F}_q$ where $q \equiv 1 \pmod 7$

In this section we will have a look at a couple of Goppa codes associated with the Klein Quartic over finite fields $\mathbb{F}_q$ that contain a 7-th root of unity, i.e. $q \equiv 1 \pmod 7$.

We recall the following facts about function fields F associated with the Klein Quartic:

- $F = \mathbb{F}_q(x, y)$ where $y^7 = x^3/(1-x)$,
- the genus of $F/\mathbb{F}_q$ is $g = 3$,
- exactly 3 places of $\mathbb{F}_q(x)$ ramify in $F/\mathbb{F}_q$, namely the pole P_∞ of x, the zero P_1 of x, and the zero P_2 of $x-1$. Denote the places lying above them by Q_∞, Q_1, and Q_2 respectively. Their degree is 1 and consequently their ramification index is 7.
- if $\mathbb{F}_q$ contains a seventh root of unity, ζ say, then $F/\mathbb{F}_q(x)$ is cyclic with Galois group isomorphic to the cyclic group generated by ζ.
- $\alpha = \begin{cases} x \mapsto \frac{x-1}{x} \\ y \mapsto \frac{-x}{y^3} \end{cases}$ is an automorphism of order 3 of $F/\mathbb{F}_q$ which permutes Q_∞, Q_1, Q_2 cyclically, i.e. $\alpha(\alpha(Q_\infty)) = \alpha(Q_1) = Q_2$.

In Chapter 8 we gave a fairly comprehensive description of codes $\mathcal{C}_\mathcal{L}(D, G)$ whose automorphism group is isomorphic to the automorphism group $Aut_{D,G}(F/\mathbb{F}_q)$. However, the proofs usually required a high number of places in the support of D. We are going to give some examples where the result still holds even though the number of places is less than the one required in the proof.

Example 9.5.16. Let $F = \mathbb{F}_{29}(x, y)$ be the function field associated to the Klein Quartic. Using our Maple routine "Klein", we find that $|\mathbb{P}_F^{(1)}| = 24$. Set $G = k(Q_\infty + Q_1 + Q_2)$ $(k = 2, 3, 6)$ and $D = \sum_{P \in J} P$, where $P = \mathbb{P}_F^{(1)} \setminus \{Q_\infty, Q_1, Q_2\}$. Then our program found that

$$Aut_{G,D}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D, G)).$$

The cases $k = 4, 5$ could not be dealt with as they would have taken too long to compute.

9.6 Conclusion

In this thesis we have attempted and to a certain extent succeeded in generalising Stichtenoth's result on the automorphism group of rational Goppa codes (c.f. Section 4.2) to Goppa codes associated with various function fields of higher genus. There certainly seems to be a case for arguing that further investigation into the subject might yield a result along the following lines.

Conjecture. *Let $F/\mathbb{F}_q$ be a function field of genus g. Let D be a divisor in F with $\operatorname{supp} D \subset \mathbb{P}_F^{(1)}$. Suppose G is a divisors in F with $\operatorname{supp}(D) \cap \operatorname{supp}(G) = \emptyset$. Set $n = \deg D$. Then*

$$Aut_{D,G}(F/\mathbb{F}_q) \cong Aut(\mathcal{C}_\mathcal{L}(D, G)),$$

provided that $lb \leq \deg G \leq n - ub$, where lb and ub are constants independent of the chosen divisors D and G.

As a proof of the above seems to be out of reach at the moment, possible routes of future investigation could include :

- What other divisors G can we use ?
- Can we use the dual codes in a more promising way?
- Is there any way of avoiding the bound $2 \deg G < \deg D$?

The first question has probably a very straightforward answer. The methods introduced in the last chapters rely on the fact that $\mathcal{L}(G)$ contains two generators of the function field in question and that all other basis elements can be expressed in terms of these generators in a way that does not involve them to a high power. So, as long as $\mathcal{L}(G)$ satisfies these conditions, we should be able to use divisors G that do not necessarily consist of places of degree one.

The second and third question are harder and I have no idea yet how to tackle them. Obviously, if the divisors of the dual code are such that my theorems cater for them then using the dual code will help. This, however, will be the exception rather than the rule.

As for the third question, there does not seem to be a way of going beyond that range because of methods used, in particular Lemma 4.3.8 and Lemma 4.3.9 require $\deg D > 2 \deg G$.

Appendix A

MAPLE Program

The following pages contain the printouts of the various MAPLE programs that I wrote to the collect empirical data which eventually led me to the formulation of the results that are proved in chapters 4 to 8. They are public domain and anyone who is interested in using them can download them from

http://www.maths.ex.ac.uk/~stephan/Thesis.html

A.1 CodeGen2.txt

This is the main body of the program dealing with most aspects.

```
readlib(search);
readlib(readdata);
FindPlaces:=proc(FX:polynom,charp:integer)

# this procedure finds all the places of the hyperelliptic function field
# associated with the equation y^2=FX over F_{charp}. A list of all the
# places is returned.

  local Places,RPlaces,SPlaces,F,dF,loop,temp1,temp2,degFX;
  F:=FX mod charp;
  degF:=degree(F,X);
  if ((isprime(charp))=false) or (charp=2)or (charp=3) or
     (degF<3) or ((degF mod 2)=0) then
     ERROR(`wrong type of arguments(degree<>2n+1;n>0 or char<>p^r`);
  fi;
  dF:=diff(F,X) mod charp;
  dF:=unapply(dF,X);
  F:=unapply(F,X);
  RPlaces:=NULL;
  SPlaces:=NULL;
  for loop from 0 to (charp-1) do
    temp1:=F(loop) mod charp;
    temp2:=dF(loop) mod charp;
    if (temp1=0) then
        if (temp2=0) then
          ERROR(`F(X) is not square-free !!`,F,dF,loop);
        else
          RPlaces:=RPlaces,[loop,0];
        fi
    else
      temp2:=numtheory[msqrt](temp1,charp);
      if (temp2<> FAIL) then
        SPlaces:=SPlaces,[loop,temp2],[loop,(-temp2 mod charp)];
      fi;
    fi;
  od;
  Places:=[[`Qinf`],RPlaces,SPlaces];
end;

comma:=proc(a,b)
# this procedure is just an auxiliary function for zipper
# taking two elements and returning them as an exprseq
  RETURN(a,b)
end;

zipper:=proc(xlist:list,ylist:list)

# this procedure zips together two lists and returns
# a list of the form [xlist[1],ylist[1],xlist[2],ylist[2],...]
# NB. if one list is longer than the other the tail of the longer
# list is concatenated to the end.

  RETURN(eval(zip(comma,xlist,ylist,'NULL')));
end;

CreateD:=proc(Q:list,numbers:list)

# this procedure takes a list of places Q and a list of indices
# numbers and creates a divisor DivD:=seq(Q[i],i in numbers)

  local loop,temp;
  TestSet:={op(numbers)};
  if nops(TestSet)<>nops(numbers) then
    ERROR(`the places you specified aren't all distinct`);
  fi;
  temp:=NULL;
  for loop in numbers do
   temp:=temp,Q[loop];
  od;
  RETURN([temp]);
end;

CreateG:=proc(Q:list,numbers:listlist)

# Q is the list containing all the places of an elliptic
# function field and numbers is a list of pairs [mult,plno]
# giving the mult(iplicity) of the place Q[plno]. The divisor
# DivG=seq(i[1]*Q[i[2]], i in numbers) is returned.

 local temp,loop;
 temp:=NULL;
 for loop in numbers do
   temp:=temp,loop[1]*Q[loop[2]];
 od;
 RETURN([temp]);
end;

valplac:=proc(P:list,DivD)

# this proc. returns the multiplicity of the place P in the
# divisor DivD

  local loop;
  for loop in DivD do
    if type(loop,`*`) then
      if op(2,loop)=P then
        RETURN(op(1,loop));
      fi;
    elif loop=P then
      RETURN(1);
    fi;
  od;
  RETURN(0);
end;

DivIntersect:=proc(DivG,DivD)
```

```
# this procedure returns the intersection of
# two divisors, ie
# DivG intersect DivD:=[seq(min(vp(DivG),vp(DivD))*P)];

  local loop,temp,templ,test,Dmul;
  temp:=NULL;
  templ:=DivG;
  for loop in templ do
    if type(loop,`*`) then
      test:=op(2,loop);
      tmul:=op(1,loop);
    else
      test:=loop;
      tmul:=1;
    fi;
    Dmul:=valplac(test,DivD);
    if Dmul<>0 then
      temp:=temp,min(Dmul,tmul)*test;
    fi;
  od;
  RETURN([temp]);
end;

ConvertDiv:=proc(DivD,flag:integer)

# if flag=0 then
#   this procedure takes a divisor of the form
#   DivD=[Qinf,3*Q[1],5*Q[2],Q[3]] for example
#   and returns a list of the form
#   [[mul1,Qinf],[mul2,Q[1]],...], ie a containing
#   pairs where the first entry is the multiplicity of the
#   place which is the second entry
# else
#  it does the opposite

  local temp,temp2,loop;
  if flag=0 then
    temp:=DivD;
    temp2:=NULL;
    for loop in temp do
      if type(loop,`*`) then
         temp2:=temp2,[op(1,loop),op(2,loop)];
      else
        temp2:=temp2,[1,loop];
      fi;
    od;
    RETURN([temp2]);
  else
    temp2:=NULL;
    for loop in DivD do;
     temp2:=temp2,op(1,loop)*op(2,loop);
    od;
    RETURN([temp2]);
  fi;
end;

GetCoeffs:=proc(FX:polynom)

# extracts the coefficients of FX with respect to X

  local loop,cfstemp,degFX;
  cfstemp:=NULL;
  degFX:=degree(FX,X);
  for loop from 0 to degFX do
   cfstemp:=cfstemp,coeff(FX,X,loop);
  od;
  if cfstemp[degFX+1]<>1 then
   # one can always assume that the leading coefficient is 1
   ERROR(`Wrong type of polynomial-leading coefficient is not 1`);
  fi;
  RETURN(cfstemp);
end;

basis_k_Qinf:=proc(degF:integer,k:integer)

# this procedure determines the basis of L(k*Qinf)

  local xbasis,ybasis,basis:list;i,loop:integer;

  if k>=0 then

    # we have a non-trivial basis

    loop:=iquo(k,2);
    xbasis:=[seq(X^`i`,`i`=1..loop)];
    if k>=degF then
      loop:=iquo((k-degF),2);
      ybasis:=[seq(Y*(X^`i`),`i`=0..loop)];
    else
      ybasis:=[];
    fi;
    basis:=[1,op(zipper(xbasis,ybasis))];
  else
    basis:=[];
  fi;
  RETURN(basis);
end;

Find_kQ_Functions:=proc(FX:polynom,charp:integer,k:integer,P)

# this procedure will construct k functions that have pole order k at P
# and at most (d-2k) at \Qinf

local F,degF,xp,yp,f,QBasis,m,endl,loop,a,q,r;
F:=FX mod charp;
degF:=degree(F,X);
xp:=P[1];
yp:=P[2];
if yp<>0 then
  f:=-yp;
  QBasis:=(Y-f)/(X-xp);
  m:=(degF+1)/2;
  endl:=min(m,k);
  for loop from 2 to endl do;
    a:=Factor(FX-f^2) mod charp;
    a:=simplify(a/(X-xp)^(loop-1)) mod charp;
    a:=subs(X=xp,a) mod charp;
    a:=a/(-2*yp) mod charp;
    f:=f+a*(X-xp)^(loop-1);
    QBasis:=QBasis,((Y-f)/(X-xp)^loop);
  od;
  for loop from endl+1 to k do;
    q:=iquo(loop,m);
    r:=irem(loop,m);
    if r=0 then
      QBasis:=QBasis,(QBasis[m])^q;
    else
      QBasis:=QBasis,((QBasis[m])^q)*(QBasis[r]);
    fi;
  od;
else
  QBasis:=NULL;
  for loop from 1 to k do;
    q:=iquo(loop,2);
    r:=irem(loop,2);
    QBasis:=QBasis,((1/(X-xp))^q)*((Y/(X-xp))^r);
  od;
fi;
RETURN(map(modp,[QBasis], charp));
end;

FindBasisLG:=proc(FX:polynom,charp:integer,DivG)

# this procedure will determine a basis for the the vector space \LofG{G}
# where G is an effictive divisor whose support consists of places of degree
# one.

local Gbasis,degF,F,GList,maxP,loop,ShiftP,loop2,z,shift;
Gbasis:=NULL;
F:=FX mod charp;
degF:=degree(F,X);
if degF=3 then
  # we are in the elliptic case
  GList:=ConvertDiv(DivG,0);
  maxP:=0;
  # find the place with the highest multiplicity in G
  for loop in GList do;
    if loop[1]>maxP then
      maxP:=loop[1];
      ShiftP:=loop[2];
    fi;
  od;
  if ShiftP<>[Qinf] then
    GList:=ConvertDiv(ShiftDiv(FX,charp,DivG,[ShiftP[1],-ShiftP[2]]),0);
  fi;
  for `loop` in GList do;
    if loop[2]=[Qinf] then
      Gbasis:=Gbasis,op(basis_k_Qinf(3,loop[1]));
    else
      z:=(Y+loop[2][2])/(X-loop[2][1]);
      for loop2 from 1 to loop[1] do;
        Gbasis:=Gbasis,z^loop2;
      od;
    fi;
  od;
  Gbasis:=[Gbasis];
  if ShiftP<>[Qinf] then
   shift:=ComputeShift(FX,charp,ShiftP);
   Gbasis:=map(modp,subs({X=shift[1],Y=shift[2]},Gbasis),charp);
  fi;
  RETURN(Gbasis);
else
  # we are in the hyperelliptic case
  GList:=ConvertDiv(DivG,0);
  maxP:=0;
  maxQinf:=0;
  # find the place with the highest multiplicity in G
  for loop in GList do;
    if loop[2]=[Qinf] then
      maxQinf:=loop[1];
    elif loop[1]>maxP then
      maxP:=loop[1];
      ShiftP:=loop[2];
    fi;
  od;
  if maxQinf=0 or (maxP<>0 and maxQinf<(degF-2)) then
    ERROR(`Sorry wrong type of divisor for the hyperelliptic case`);
  else
    Gbasis:=op(basis_k_Qinf(degF,maxQinf));
    for loop in GList do;
      if loop[2]<>[Qinf] then;
        Gbasis:=Gbasis,op(Find_kQ_Functions(FX,charp,loop[1],loop[2]));
      fi;
```

```
    od;
  fi;
  RETURN([Gbasis]);
fi;
end;

AddPoints:=proc(FX:polynom,charp:integer,P1,P2)

# this procedure adds to points on the elliptic curve
# FX in characteristic charp

  local x1,x2,x3,y1,y2,y3,a2,a4,a6,lambda,v,test1,test2,cfs;
  if degree(FX mod charp,X)=3 then
     if P1=[Qinf] then
       RETURN(P2);
     elif P2=[Qinf] then
       RETURN(P1);
     else
       x1:=op(1,P1) mod charp;
       y1:=op(2,P1) mod charp;
       x2:=op(1,P2) mod charp;
       y2:=op(2,P2) mod charp;
       cfs:=GetCoeffs((FX mod charp));
       a2:=cfs[3];
       a4:=cfs[2];
       a6:=cfs[1];
       test1:=y1^2-(subs(X=x1,FX)) mod charp;
       test2:=y2^2-(subs(X=x2,FX)) mod charp;
       if (test1<>0)or(test2<>0) then
         ERROR(`Points not on the curve`);
       fi;
       if x1=x2 then
          test1:=y1+y2 mod charp;
          if test1=0 then
            RETURN([`Qinf`]);
          else
            lambda:=(3*x1^2+2*a2*x1+a4)/(2*y1) mod charp;
             v:=(-x1^3+a4*x1+2*a6)/(2*y1) mod charp;
          fi;
       else
          lambda:=(y2-y1)/(x2-x1) mod charp;
          v:=(y1*x2-y2*x1)/(x2-x1) mod charp;
       fi;
       x3:=lambda^2-a2-x1-x2 mod charp;
       y3:=-lambda*x3-v mod charp;
       RETURN([x3,y3]);
     fi;
  else
     ERROR(`F does not have degree 3`);
  fi;
end;

ShiftDiv:=proc(FX:polynom,charp:integer,DivD,P)

# this procedure applies alp:Places1->Places1 which is a
# translation by P,ie alp(Q)=Q+P to a divisor DivD
# FX and charp determine the polynomial of the elliptic
# function field and its characteristic

  local Dtemp1,NewD;
if degree(FX mod charp,X)=3 then
  if P<>[Qinf] then
    # we have to do a non-trivial shift
    Dtemp1:=ConvertDiv(DivD,0);
    NewD:=NULL;
    for loop in Dtemp1 do
      NewD:=NewD,[loop[1],AddPoints(FX,charp,loop[2],P)];
    od;
    RETURN(ConvertDiv([NewD],1));
  fi;
  RETURN(DivD);
else
  ERROR(`the polynomial does not have degree 3`);
fi;
end;

SigmaOfD:=proc(FX,charp,DivG,Sigma)

# this procedure computes Sigma(DivG) where DivG is any effectice divisor whose
# support consists of places of degree 1.
# Sigma is of the form [epsilon,power,P] and it acts on places of degree 1
# by Sigma(P')=(epsilon^power)(P') + P

local NewG,tempP,tempM,eps2,eps3,NewP;

if degree(FX mod charp,X)=3 then
  NewG:=NULL;
  for loop1 in DivG do;
    if type(loop1,`*`) then
      tempP:=op(2,loop1);
      tempM:=op(1,loop1);
    else
      tempP:=loop1;
      tempM:=1;
    fi;
    eps2:=(Sigma[1]^(2*Sigma[2])) mod charp;
    eps3:=(Sigma[1]^(3*Sigma[2])) mod charp;
    if tempP=[Qinf] then
      NewP:=[Qinf];
    else
      NewP:=[((eps2*tempP[1]) mod charp),((eps3*tempP[2]) mod charp)];
    fi;
    NewP:=AddPoints(FX,charp,NewP,Sigma[3]);
    NewG:=NewG,tempM*NewP;
  od;
  RETURN([NewG]);
else
  ERROR(`the polynomial doesn not have degree 3`);
fi;
end;

GetPermMat:=proc(sigma)

  # this function takes a permutation and finds the corresponding
  # permutation matrix.

  local P,l,loop,i;
  l:=nops(sigma);
  P:=linalg[matrix](l,l,0);
  i:=1;
  for loop in sigma do
   P[i,loop]:=1;
   i:=i+1;
  od;
  RETURN(P);
end;

ComputeShift:=proc(PX:polynom,charp:integer,Q)

# this procedure computes the functions FX,FY which represent the
# shift by the place Q, ie P=[g(X),h(Y)] then P+Q=[g(FX),h(FY)]

  local cfs,x,y,v,test;
if degree(PX mod charp,X)=3 then
  x:=Q[1];
  y:=Q[2];
  cfs:=GetCoeffs((PX mod charp));
  test:=y^2-(subs(X=x,PX)) mod charp;
  if (test<>0) then
     ERROR(`Place is not on the curve`);
  fi;
  lmda:=(Y-y)/(X-x);
  v:=(y*X-Y*x)/(X-x);
  FX:=simplify(expand(lmda^2-cfs[3]-x-X)) mod charp;
  FX:=simplify(subs(Y^2=PX,FX)) mod charp;
  FY:=simplify(expand(-lmda*FX-v)) mod charp;
  FY:=simplify(subs(Y^2=PX,FY)) mod charp;
  RETURN(FX,FY);
else
  ERROR(`the polynomial does not have degree 3`);
fi;
end;

EndoOfD:=proc(FX,charp,DivG,Endo)

# this procedure computes Endo(DivG) where DivG is any effectice divisor whose
# support consists of places of degree 1.
# Endo is of the form [n] and it acts on places of degree 1
# by Endo(P')=[n]*P, ie multiplication within the group

local NewG,tempP,tempM,NewP,n;
if degree(FX mod charp, X)=3 then
  NewG:=NULL;
  n:=op(Endo);
  for loop1 in DivG do;
    if type(loop1,`*`) then
      tempP:=op(2,loop1);
      tempM:=op(1,loop1);
    else
      tempP:=loop1;
      tempM:=1;
    fi;
    if tempP=[Qinf] then
      NewP:=[Qinf];
    else
      NewP:=tempP;
      for loop1 from 1 to (n-1) do;
        NewP:=AddPoints(FX,charp,NewP,tempP);
      od;
    fi;
    NewG:=NewG,tempM*NewP;
  od;
  RETURN([NewG]);
else
  ERROR(`the polynomial does not have degree 3`);
fi;
end;

EvalF:=proc(FX,charp:integer,BasisN,BasisD,P)
# evaluates BasisN/BasisD at P
local BDV,BNV,BNtemp,YCoeff;

  BDV:=subs(X=P[1],BasisD) mod charp;
  if BDV<>0 then
     # we can simply evaluate loop1 at loop2
     BDN:=subs({X=P[1],Y=P[2]},BasisN);
     RETURN (BDN/BDV mod charp);
  else
     # we have to do some work
     BNtemp:=expand(BasisN) mod charp;
     BNtemp:=simplify(BNtemp,{Y^2=FX})mod charp;
     BNtemp:=collect(BNtemp,Y);
     YCoeff:=coeff(BNtemp,Y,1);
     YCoeff:=2*YCoeff*Y-BNtemp;
```

```
    BNtemp:=expand(BNtemp*YCoeff) mod charp;
    BNtemp:=simplify(BNtemp,{Y^2=FX}) mod charp;
    BNtemp:=Factor(BNtemp) mod charp;
    BNtemp:=BNtemp/BasisD mod charp;
    BNtemp:=BNtemp/YCoeff mod charp;
    RETURN(subs({X=P[1],Y=P[2]},BNtemp)mod charp);
  fi;
end;

CreateMatrix:=proc(FX:polynom,charp:integer,DivG,DivD,ShiftP,flag)

# this procedure will return three different result depending on the way
# it is called.
# 1) If it's called with FX,charp,DivG,DivD only then it'll return
#    the basis of L(DivG) & a generator matrix of C(DivG,DivD).
# 2) if it's called with all parameters and flag=0 then it'll return
#    a basis of L(DivG) and a generator matrix of C(DivG,DivD) and C(DivG,DivD')
#    where DivD' is the translation of DivD by ShiftP
# 3) as 2) with flag<>0 then it'll return
#    a basis of L(DivG),a basis of L(DivG') and generator matrices for
#    C(DivG,DivD) and C(DivG',DivD) where DivG' is the translation of
#    DivG by ShiftP.


  local NoParms,NewG,NewD,BasisG,BasisNG,loop1,loop2,
        Dlist,NDlist,len1,len2,C1,C2,C1Max,C2Max,numSP,
        BasisD,BasisN,BDV,BNV,BNtemp,YCoeff;

  NoParms:=nargs;
  C1:=NULL;
  C2:=NULL;

# the following checks whether DivD is the sum of distinct places
# and whether supp(DivG) and supp(DivD) are disjoint

  Dlist:=ConvertDiv(DivD,0);
  for loop1 in Dlist do
    if loop1[1]<>1 then
      ERROR(`D<>sum(Pi:i in list)`);
    fi;
  od;
  if DivIntersect(DivG,DivD)<>[] then
      ERROR(`The supports of your G and D are not disjoint`);
  fi;

  BasisG:=FindBasisLG(FX,charp,DivG);
  len1:=nops(BasisG);
  len2:=nops(Dlist);

  if NoParms=4 then

    # just return basis of L(DivG) and generator matrix for C(DivG,DivD)

    for loop1 in BasisG do
      BasisN:=numer(loop1);
      BasisD:=denom(loop1);
      for loop2 in DivD do;
        C1:=C1,EvalF(FX,charp,BasisN,BasisD,loop2);
      od;
    od;
    C1Max:=linalg[matrix](len1,len2,[C1]);
    RETURN(BasisG,C1Max);
  elif ((NoParms=6) and (flag=0)) then
    # this part of the code is redundant and shouldn't be used anymore
    # return basis of L(DivG) and matrices for C(DivG,DivD) & C (DivG,DivD')
    numSP:=nops(ShiftP);
    if numSP=3 then
      NewD:=SigmaOfD(FX,charp,DivD,ShiftP);
    elif numSP=2 then
      NewD:=ShiftDiv(FX,charp,DivD,ShiftP);
    else
      NewD:=EndoOfD(FX,charp,DivD,ShiftP);
    fi;
    if DivIntersect(DivG,NewD)<>[] then
      ERROR(`The supports of G and the translated D are not disjoint`);
    fi;
    NDlist:=ConvertDiv(NewD,0);
    for loop1 in BasisG do
      BasisN:=numer(loop1);
      BasisD:=denom(loop1);
      for loop2 in DivD do
        C1:=C1,EvalF(FX,charp,BasisN,BasisD,loop2);
      od;
      for loop2 in NewD do
        C2:=C2,EvalF(FX,charp,BasisN,BasisD,loop2);
      od;
    od;
    C1Max:=linalg[matrix](len1,len2,[C1]);
    C2Max:=linalg[matrix](len1,len2,[C2]);
    print(`G =`, DivG);
    print(`D =`, DivD);
    print(`D' =`, NewD);
    print(`a basis of L(G) and matrices for C(G,D) & C(G,D') are :`);
    RETURN(BasisG,C1Max,C2Max);
  elif ((NoParms=6) and (flag<>0)) then

    # return basis of L(DivG),L(DivG') and matrices for C(DivG,DivD)
    #  & C(DivG',DivD)

    numSP:=nops(ShiftP);
    if numSP=3 then
      NewG:=SigmaOfD(FX,charp,DivG,ShiftP);
    elif numSP=2 then
      NewG:=ShiftDiv(FX,charp,DivG,ShiftP);
    else
      NewG:=EndoOfD(FX,charp,DivG,ShiftP);
    fi;
    if DivIntersect(DivG,NewD)<>[] then
      ERROR(`The supports of the translated G and D are not disjoint`);
    fi;
    BasisNG:=FindBasisLG(FX,charp,NewG);
    for loop1 in BasisG do
      BasisN:=numer(loop1);
      BasisD:=denom(loop1);
      for loop2 in DivD do
        C1:=C1,EvalF(FX,charp,BasisN,BasisD,loop2);
      od;
    od;
    for loop1 in BasisNG do
      BasisN:=numer(loop1);
      BasisD:=denom(loop1);
      for loop2 in DivD do
        C2:=C2,EvalF(FX,charp,BasisN,BasisD,loop2);
      od;
    od;
    C1Max:=linalg[matrix](len1,len2,[C1]);
    C2Max:=linalg[matrix](len1,len2,[C2]);
    print(`G =`, DivG);
    print(`G' =`, NewG);
    print(`D =`, DivD);
    print(`basis of L(G) & L(G') and matrices for C(G,D) & C(G',D) are :`);
    RETURN(BasisG,BasisNG,C1Max,C2Max);
  else
    ERROR(`wrong numbers of arguments `);
  fi;
end;

FindPerm:=proc(FX:polynom,charp:integer,Q:list,ShiftP:list)

# this function will return a permutation corresponding to the
# endomorphism sigma of the elliptic function field defined by FX
# of characteristic charp; where sigma is given by either
# sigma(Q)=1) [n]*Q
#          2) Q+P for some P
#          3) eps(Q)+P for some eps and P
# An expression of the form [[Qinf,1,3,4],[2,5,6],[7,8]] and the
# corresponding list with places will be returned.

  local Qset,i,loop,Qtable,Permlist,firstp,result1,result2,PermI,PermP,
        temp1,temp2;

  if ShiftP<>[Qinf] then
    NumOfOp:=nops(ShiftP);
    flag:=0;
    if NumOfOp=1 then
      if (op(ShiftP) mod (nops(Q))) <>1 then
        flag:=1;
      fi;
    elif NumOfOp=2 then
       flag:=2;
    else
      if ((((ShiftP[1]^ShiftP[2]) mod charp)<>1) or (ShiftP[3]<>[Qinf])) then
        flag:=3;
      fi;
    fi;
    if flag<>0 then
      Qset:=Q;
      i:=1;
      for loop in Qset do
       Qtable[loop]:=i;
       i:=i+1;
      od;
      Qtable[[Qinf]]:='Qinf';
      Permlist:=[seq(`i`,`i`=1..nops(Q))];
      firstp:=Permlist[1];
      Permlist:=subs(1=NULL,Permlist);
      result1:=firstp;
      result2:=Q[firstp];
      PermI:=NULL;
      PermP:=NULL;
      while Permlist<>[] do
        temp1:=Q[firstp];
        Qtable[temp1]:=0;
        if flag=1 then
          temp2:=op(EndoOfD(FX,charp,[temp1],ShiftP));
        elif flag=2 then
          temp2:=AddPoints(FX,charp,temp1,ShiftP);
        else
          temp2:=op(SigmaOfD(FX,charp,[temp1],ShiftP));
        fi;
        if Qtable[temp2]<> 0 then
          firstp:=Qtable[temp2];
          result1:=result1,firstp;
          result2:=result2,temp2;
          Permlist:=subs(firstp=NULL,Permlist);
        else
          PermI:=PermI,[result1];
          PermP:=PermP,[result2];
          firstp:=Permlist[1];
          Permlist:=subs(firstp=NULL,Permlist);
          result1:=firstp;
          result2:=Q[firstp];
        fi;
      od;
```

```
      RETURN(subs(1=‘Qinf‘,[PermI,[result1]]),[PermP,[result2]]);
    fi;
  fi;
  RETURN([identity],[identity]);
end;

jInvariant:=proc(FX:polynom,charp:integer)

  # this routine will compute the j-invariant of an elliptic curve
  # given in the form Y^2=FX where FX is a polynomial of the form
  # FX:=X^3+a2*X^2+a4*X+a6
  # This procedure uses the formula given in Silverman's 'The Arithmetic
  # of elliptic curves'.

  local cfs,a2,a4,a6,b2,b4,b6,b8,c4,delta,j;
if degree(FX mod charp,X)=3 then
   cfs:=GetCoeffs((FX mod charp));
   a2:=cfs[3];
   a4:=cfs[2];
   a6:=cfs[1];
   b2:=4*a2;
   b4:=2*a4;
   b6:=4*a6;
   b8:=4*a2*a6-a4^2;
   c4:=b2^2-24*b4;
   delta:=-b2^2*b8-8*b4^3-27*b6^2+9*b2*b4*b6;
   if ((delta mod charp)=0) then
     ERROR(‘curve is singular : F=‘,Factor(FX) mod charp);
   fi;
   j:=c4^3/delta mod charp;
   if (nargs<3) then
     if ((j=(1728 mod charp)) and (charp<>2) and (charp<>3)) then
       print(‘j is congruent to 1728 mod ‘, charp);
       if ((charp-1) mod 4)=0 then
         print(‘and there are 4 Qinf-fixing automorphisms‘);
         if ((a2<>0) or (a6<>0)) then
   WFX:=216^2*simplify(subs(X=((X-3*b2)/36),FX))mod charp;
           print(‘Warning !! Please use the following, equivalent
                  polynomial instead :‘,WFX);
         fi;
       else
         print(‘but there are only 2 Qinf-fixing automorphisms‘);
       fi;
     elif ((j=0) and (charp<>2) and (charp<>3)) then
       print(‘j is congruent to 0 mod ‘,charp);
       if ((charp-1) mod 6)=0 then
         print(‘and there are 6 Qinf-fixing automorphisms‘);
         if ((a2<>0) or (a4<>0)) then
           WFX:=216^2*simplify(subs(X=((X-3*b2)/36),FX))mod charp;
           print(‘Warning !! Please use the following, equivalent
                  polynomial instead :‘,WFX);
         fi;
       else
         print(‘but there are only 2 Qinf-fixing automorphisms‘);
       fi;
     else
       print (‘j is not congruent to 0 or 1728 mod ‘, charp);
       print (‘Hence there are only 2 Qinf-fixing automorphisms‘);
     fi;
   fi;
   RETURN(j);
else
  ERROR(‘the polynomial does not have degree 3‘);
fi;
end;

FindOrdPoint:=proc(FX:polynom,charp:integer,P1)

# This procedure determines the order of the point P1 on the curve
# given by FX in charp

local PointOrder,NewP1;

  PointOrder:=1;
  NewP1:=P1;
  while (NewP1<>[Qinf]) do;
    NewP1:=AddPoints(FX,charp,NewP1,P1);
    PointOrder:=PointOrder+1;
  od;
  RETURN(PointOrder);
end;

FindOrdD:=proc(FX:polynom,charp:integer,DivD)

# this procedure finds the order of all places of degree 1
# given by the  divisor DivD.
# note this assumes that DivD is sum of places of degree 1.
# it returns a list of pairs where the first entry is the
# multiplicity of the place and the second entry is its order
# ie FindOrdD(X^3-X,5,[2*[0,0]]) returns [2,2]

local DOrder,loop,tempD;

  tempD:=ConvertDiv(DivD,0);
  DOrder:=NULL;
  for loop in tempD do;
    DOrder:=DOrder,[loop[1],FindOrdPoint(FX,charp,loop[2])];
  od;
end;

FindHyperAutos:=proc(FX:polynom,charp:integer)

# this procedure finds all automorphisms of the (hyper)elliptic
# function field by looking at the action of PGL2(charp) on the
# projective line

local Xcoord,loop,yval,degFX,FofC,c,loop2,loop3,loop4,IsAuto,XcoordL,
      XcoordS,temp,result,btmp,testauto,testautoNum,testautoDen,ctmp,
      Dentmp,Numtmp,testcomp;

  Xcoord:=NULL;
  result:=NULL;

  # compute all possible X-coordinates. They represent the points of the
  # curve on the projective line

  for loop from 0 to charp-1 do;
    yval:=subs(X=loop,FX) mod charp;
    if yval=0 then
      Xcoord:=Xcoord,loop;
    fi;
  od;
  if [Xcoord]=[] then

    # only need to consider \Qinf-fixing autos of the form X->epsilonX

    for loop4 from 1 to charp-1 do;
      testauto:=loop4*X mod charp;
      NewFX:=simplify(expand(subs(X=testauto,FX))) mod charp;
      TestRem:=Rem(NewFX,FX,X) mod charp;
      if (TestRem=0) then
         TestQuo:=Quo(NewFx,FX,X) mod charp;
         Ycoeff:=numtheory[msqrt](TestQuo,charp);
         if Ycoeff=FAIL then
           print(testauto,TestQuo,‘Hmmmm‘);
         else
           result:=result,[testauto,Ycoeff*Y],[testauto,-Ycoeff*Y mod charp];
         fi;
      fi;
    od;
    RETURN([result]);
  fi;

  degFX:=degree(FX,X);
  c:=‘c‘;
  FofC:=simplify(c^(degFX+1)*(subs(X=1/c,FX))) mod charp;

  # FofC(c)=0 mod charp has to hold if we are to find an automorphism

  # Some special cases first. The following will always give us the
  # identity auto and the auto coming from the Galois group.
  # These are the Qinf-fixing autos.
  XcoordL:=[Xcoord];
  XcoordS:={Xcoord};
  Firstx:=XcoordL[1];
  for loop3 in XcoordL do;
    for loop4 from 1 to charp-1 do;
       btmp:=loop4*Firstx-loop3;
       testauto:=(X+btmp)/loop4 mod charp;
       testcomp:=loop4*X-btmp mod charp;
       IsAuto:=true;
       for loop in XcoordL do;
         temp:=subs(X=loop,testcomp) mod charp;
         if member(temp,XcoordS)=FALSE then

           # the matrix does not permute the Xcoord

           IsAuto:=false;
           break;
         fi;
       od;
       if IsAuto then

         # we might have an automorphism here...at least it permutes the
         # Xcoord. We need to check whether it preserves the defining
         # equation.

         NewFX:=simplify(expand(subs(X=testauto,FX))) mod charp;
         TestRem:=Rem(NewFX,FX,X) mod charp;
         if (TestRem=0) then
           TestQuo:=Quo(NewFX,FX,X) mod charp;
           Ycoeff:=numtheory[msqrt](TestQuo,charp);
           if Ycoeff=FAIL then
             print(testauto,TestQuo,‘Hmmmm‘);
           else
             result:=result,[testauto,Ycoeff*Y],[testauto,-Ycoeff*Y mod charp];
           fi
         fi;
       fi;
    od;
  od;

  # the above part seems to work alright.

  # now we deal with another special case in which a=0.
  # ie these are the automorphisms that send 0 to infinity
  # and infinity to something else in XCoord

  if (member(0,XcoordS)) then
    # there is a chance that we can find such autos...

    for loop2 from 1 to charp-1 do;
       for loop4 in XcoordL do;
         testauto:=loop2*X-loop4*loop2 mod charp;
```

```
      testcomp:=(1+loop4*loop2*X)/(loop2*X) mod charp;
      #if loop4=0 then
      #    testauto:=loop2*X mod charp;
      # else
      #    testauto:=loop2*X+(1/loop4) mod charp;
      # fi;
       IsAuto:=true;
       for loop in XcoordL do;
          if loop=0 then
            next;
          fi;
          temp:=subs(X=loop,testcomp) mod charp;
        #if temp=0 then
        #   next;
        #fi;
        #temp:=1/temp mod charp;
         if member(temp,XcoordS)=false then
           IsAuto:=false;
           break;
         fi;
       od;
       if IsAuto then
          # well, there is a chance we might have an auto here. At least
          # it permutes the Xcoord.

          NewFX:=((testauto^(degFX+1))*(subs(X=(1/testauto),FX))) mod charp;
          NewFX:=simplify(NewFX) mod charp;
          NewFX:=expand(NewFX) mod charp;
          TestRem:=Rem(NewFX,FX,X) mod charp;
          if (TestRem=0) then
             TestQuo:=Quo(NewFX,FX,X) mod charp;
             Ycoeff:=numtheory[msqrt](TestQuo,charp);
             if Ycoeff=FAIL then
                print(testauto,TestQuo,`Hmmmm`);
             else
                temp:=(Ycoeff*Y)/(testauto^((degFX+1)/2)) mod charp;
                result:=result,[1/testauto,temp],[1/testauto,-temp mod charp];
             fi;
          fi;
       fi;
    od;
  od;
fi;

# again the above part seems to work alright.
#Now we are looking at the remaing general matrices...

for ctmp from 1 to charp-1 do;
   temp:=subs(c=ctmp,FofC) mod charp;
   if (temp<>0) then
     # this c=ctmp won't give rise to an auto
     next;
   fi;
   for loop4 in XcoordL do;
     for btmp from 0 to charp-1 do;
       dtmp:=-loop4*ctmp mod charp;
       temp:=dtmp-(btmp*ctmp) mod charp;
       if (temp=0) then

         # this one isn't invertible

         next;
       fi;

       # well, here we go

       testauto:=(X+btmp)/(X*ctmp+dtmp) mod charp;
       testautoNum:=X+btmp mod charp;
       testautoDen:=X*ctmp+dtmp mod charp;
       testcomp:=(dtmp*X-btmp)/(1-ctmp*X) mod charp;
       IsAuto:=true;
       for loop in XcoordL do;
         if (((1-ctmp*loop) mod charp)=0) then
           next;
         fi;
         #Dentmp:=subs(X=loop,testautoDen) mod charp;
         #if (Dentmp=0) then
         #  next;
         #fi;
         temp:=subs(X=loop,testcomp) mod charp;
         if member(temp,XcoordS)=false then

           # no auto as it doesn't permute Xcoord
           IsAuto:=false;
           break;
         fi;
       od;
       if IsAuto then

         # well, this might be the one...

        # testauto:=testautoNum/testautoDen mod charp;
         NewFX:=((testautoDen^(degFX+1))*(subs(X=(testauto),FX))) mod charp;
         NewFX:=simplify(NewFX) mod charp;
         NewFX:=expand(NewFX) mod charp;
         TestRem:=Rem(NewFX,FX,X) mod charp;
         if (TestRem=0) then
            TestQuo:=Quo(NewFX,FX,X) mod charp;
            Ycoeff:=numtheory[msqrt](TestQuo,charp);
            if Ycoeff=FAIL then
               print(testauto,TestQuo,`Hmmmm`);
            else
               temp:=(Ycoeff*Y)/(testautoDen^((degFX+1)/2)) mod charp;
               result:=result,[testauto,temp],[testauto,-temp mod charp];
            fi;
         fi;
       fi;
     od;
   od;
 od;
 RETURN([result]);
end;


EvalPointHyperAuto:=proc(auto,P,charp)

# this procedure evaluates the image of P under the automorphism
# auto.

  local a,b,c,d,hypnum,hypdenom,test,test2;
   hypnumer:=numer(auto[1]);
   hypdenom:=denom(auto[1]);
   a:=coeff(hypnumer,X);
   b:=coeff(hypnumer,X,0);
   c:=coeff(hypdenom,X);
   d:=coeff(hypdenom,X,0);
   if P=[Qinf] then
     if c<>0 then
       RETURN([(-d/b mod charp),0]);
     else
       RETURN([Qinf]);
     fi;
   else
     test:=solve(auto[1]=P[1],X);
     if [test]=[] then
        RETURN([Qinf]);
     elif ((denom(test) mod charp)=0) then
         RETURN([Qinf]);
     else
       test:=test mod charp;
       test2:=solve(auto[2]=P[2],Y);
       test2:=subs(X=test,test2) mod charp;
       RETURN([test,test2]);
     fi;
   fi;
end;


FindAutGD:=proc(FX:polynom,charp:integer,Q,DivG,DivD)

# this procedure will determine the automorphism subgroup of F which
# fixes DivG and DivD (note this makes use of lemma 2.2 in
# Stichtenoth's paper) FX is the defining polynomial Q is the list
# containing all places of degree 1 DivG is a positive divisor whose
# support consists of places of degree 1 DivD is a sum of distict
# places of degree 1 In the elliptic function field case, the
# procedure returns a list of the form [[epsilon,power,P],...]  where
# epsilon is a root of unity(2,4,6th) power is the power of epsilon
# and P is the translation, ie if Sigma is an automorphism then
# Sigma(P')=(epsilon^power)(P')+P.  Otherwise it will return a list of
# "functions" [[f(x,y),g(x,y)],...]  which contain the field autos
# which send x->f(x,y) and y->g(x,y) and fix DivD, DivG.

local jInv,epsilon,NumEps,loop1,loop2,AutGD,TempSigma,TempG,TempD,
      NoAuto,P,DivGc,AutF,auto;

  if degree(FX,X)=3 then

    # We are in the elliptic case
    # first we determine how many Qinf-fixing autos there are

    jInv:=jInvariant(FX,charp,1);
    NumEps:=2;
    epsilon:=-1 mod charp;
    if (jInv=(1728 mod charp)) and (charp<>2) and (charp<>3) then
      if ((charp-1 mod 4)=0) then
        epsilon:=((numtheory[primroot](charp))^((charp-1)/4)) mod charp;
        NumEps:=4;
      fi;
    elif (jInv=0) and (charp<>2) and (charp<>3) then
      if ((charp-1 mod 6)=0) then
        epsilon:=((numtheory[primroot](charp))^((charp-1)/6)) mod charp;
        NumEps:=6;
      fi;
    fi;
    AutGD:=NULL;
    for loop1 from 1 to nops(Q) do;
      for loop2 from 0 to (NumEps-1) do;
        TempSigma:=[epsilon,loop2,Q[loop1]];
        TempG:=DivIntersect(DivG,SigmaOfD(FX,charp,DivG,TempSigma));
        if {op(TempG)}={op(DivG)} then
          TempD:=DivIntersect(DivD,SigmaOfD(FX,charp,DivD,TempSigma));
          if {op(TempD)}={op(DivD)} then
            AutGD:=AutGD,TempSigma;
          fi;
        fi;
      od;
    od;
    RETURN([AutGD]);
  else

    # We are in the hyperelliptic case.
    # The idea is to find all the autos of F/Fq and then apply them to
    # DivG and DivD. If they are both preserved by an auto then it
```

```
    # is actually in AutGD.

    AutGD:=NULL;
    AutF:=FindHyperAutos(FX,charp);
    for auto in AutF do;
      NoAuto:=0;
      DivGc:=ConvertDiv(DivG,0);
      for P in DivGc do;
        if P[1]<>valplac(EvalPointHyperAuto(auto,P[2],charp),DivG) then
          NoAuto:=1;
          break;
        fi;
      od;
      if NoAuto=1 then
         next;
      fi;
      for P in DivD do;
        if valplac(EvalPointHyperAuto(auto,P,charp),DivD)<>1 then
          NoAuto:=1;
          break;
        fi;
      od;
      if NoAuto=0 then
       AutGD:=AutGD,auto;
      fi;
    od;
    RETURN([AutGD]);
  fi;
end;

Position:=proc(x,l)

# this function returns the first i with l[i]=x if x in l
# 0 otherwise

local i;

  for i from 1 to nops(l) do
    if l[i]=x then
      RETURN(i);
    fi
  od;
  RETURN(0);
end;

MultiplyPerm:=proc(perm1:list,perm2:list)

# This function multiplies two permutations together
# It computes perm1 o perm2 = result

local length, result, loop;

 length:=nops(perm1);
 if length<>nops(perm2) then
    ERROR(`incompatible permutations`);
 else
    result:=NULL;
    for loop from 1 to length do
      result:=result,perm2[perm1[loop]];
    od;
 fi;
 RETURN([result]);
end;

FindPermOrd:=proc(perm:list)

# this procedure determines the order of a permutation

local ident,order,temp;

  ident:=[seq(`i`,`i`=1..nops(perm))];
  order:=1;
  temp:=perm;
  while temp<>ident do
    temp:=MultiplyPerm(temp,perm);
    order:=order+1;
  od;
  RETURN(order);
end;


identityPerm:=proc(length:integer)

# returns the identity permutation

local i;
 RETURN([seq(i,i=1..length)]);
end;

PermuteCols:=proc(M,perm:list)

# this procedure takes a matrix M and a permutation perm and computes
# esssentially M*P where P is the permutation matrix corresponding to perm.
# However the procedure uses column operations to achieve this.

local tempMat,loop1,loop2,NumOfCols;

  NumOfCols:=nops(perm);
  NumOfRows:=linalg[rowdim](M);
  tempMat:=copy(M);
  for loop1 from 1 to NumOfCols do;
    pj:=perm[loop1];
    for loop2 from 1 to NumOfRows do;
       tempMat[loop2,pj]:=M[loop2,loop1];
    od;
  od;
  RETURN(tempMat);
end;

FindPivots:=proc(M)

# this procedure determines the pivots of a matrix M which has been
# converted to REF

local pivotlist,loop1,loop2,MColDim,MRowDim;

  pivotlist:=NULL;
  MColDim:=linalg[coldim](M);
  MRowDim:=linalg[rowdim](M);
  loop1:=1;
  while (loop1<=MRowDim) do
    loop2:=loop1;
    while M[loop1,loop2]=0 do;
      if loop2=MColDim then
        #row of zeros has been encountered, ie no more pivots.
        RETURN([pivotlist]);
      else
        loop2:=loop2+1;
      fi;
    od;
    pivotlist:=pivotlist,loop2;
    loop1:=loop1+1;
  od;
  RETURN([pivotlist]);
end;

ConvertPerm:=proc(Perm)

# this procedure converts the old format of permutations to the new one
# and vica versa

local loop,tempPerm,lenPerm;

  lenPerm:=nops(eval(Perm));
  tempPerm:=array(Perm);
  for loop from 1 to lenPerm do
     tempPerm[Perm[loop]]:=loop;
  od;
  RETURN(convert(tempPerm,list));
end;

ChangeMatrix:=proc(CMat,pivots)

# this procedure swaps around columns so that CMat' will have pivots in the
# first r columns (r=nops(pivots)=Rankr)

local loop1,loop2,lenpiv,permlist,NewCMat;

  lenpiv:=nops(pivots);
  permlist:=NULL;
  NumOfRow:=linalg[rowdim](CMat);
  NumOfCol:=linalg[coldim](CMat);
  NewCMat:=copy(CMat);
  for loop1 from 1 to lenpiv do;
    pivtemp:=pivots[loop1];
    newperm:=array([seq(i,i=1..NumOfCol)]);
    if loop1<>pivtemp then
      newperm[loop1]:=pivtemp;
      newperm[pivtemp]:=loop1;
      NewCMat:=linalg[swapcol](NewCMat,loop1,pivtemp);
      permlist:=convert(newperm,list),permlist;
    fi;
  od;
  RETURN([[permlist],NewCMat]);
end;

ChangeAutoResult:=proc(Automorphisms,ColPerms)

# this procedure takes the permutations that were computed
# and applies the ColPerms neccessary to obtain the 'real' autos

local loop1,loop2,lenAut,lenCol,newpermlist,oldperm,newperm;

  lenAut:=nops(Automorphisms);
  lenCol:=nops(ColPerms);
  if lenCol=0 then
    RETURN(Automorphisms);
  fi;
  newpermlist:=NULL;
  for loop1 from 1 to lenAut do;
    newperm:=Automorphisms[loop1];
    for loop2 from 1 to lenCol do;
      tempperm:=ColPerms[loop2];
      newperm:=MultiplyPerm(newperm,tempperm);
      newperm:=MultiplyPerm(tempperm,newperm);
    od;
    newpermlist:=newpermlist,newperm;
  od;
  RETURN([newpermlist]);
end;

GetAutos:=proc (H,charp,Rankr,length)

# this procedure writes the data to a file, executes LeonTest and
```

```
# reads in the result
  local result;
  writeto (‘MapleMat.out‘);
  print(charp);
  print(Rankr);
  print(length);
  for loop1 from 1 to Rankr do;
    for loop2 from 1 to length do;
     print(H[loop1,loop2]);
    od;
  od;
  writeto(terminal);
  system (‘rm timeout.txt‘);
  system (‘(time LeonTest3 MapleMat.out MapleAutos.in) 2>&1 |cat >timeout.txt‘);
  tempresult1:=readdata(‘MapleAutos.in‘ , integer);
  loop1:=0;
  result:=NULL;
  tempresult2:=NULL;
  for loop2 in tempresult1 do;
    loop1:=loop1+1;
    if loop1=length then
      tempresult2:=tempresult2,loop2;
      result:=result,[tempresult2];
      loop1:=0;
      tempresult2:=NULL;
    else
      tempresult2:=tempresult2,loop2;
    fi;
  od;
  RETURN([result]);
end;

FindAutGroup2:=proc(CGMat,charp)

# this procedure sets everything up correctly so that GenPerm can calculate
# the automorphism group.

local H,pivots,result,Rankr,NumOfCols,temp;

  H:=Gaussjord(CGMat) mod charp;
  Rankr:=linalg[rank](H);
  NumOfCols:=linalg[coldim](H);
  pivots:=FindPivots(H);
  temp:=ChangeMatrix(H,pivots);
  H:=temp[2];
  result:=GetAutos(H,charp,Rankr,NumOfCols);
  result:=ChangeAutoResult(result,temp[1]);
  RETURN(result);
end;

extracttime:=proc()

# this procedure will extract the time it took for LeonTest to
# finish execution

local timestr,pos;
  timestr:=readline(‘timeout.txt‘);
  while timestr=0 or (search(timestr,‘real‘)=false) do;
    timestr:=readline(‘timeout.txt‘);
  od;
  timestr:=substring(timestr,5..length(timestr));
  RETURN(timestr);
end;


AnalyseOutput:=proc(FX,charp,DivQ,DivG,DivD,CodeC,Str1,Str2)

# this procedure just outputs certain values

local TempAutF,TempAutC;
#   if degree(FX mod charp,X)=3 then
      TemAutF:=FindAutGD(FX,charp,DivQ,DivG,DivD);
      print(subs(‘Aut‘.Str1.‘(F) is given by :‘));
      print(TemAutF);
      print(‘Its size is :‘,nops(TemAutF));
#   fi;
   print(subs(‘C(‘.Str2.‘)’s automorphism group is :‘));
   TemAutC:=FindAutGroup2(CodeC,charp);
   print(TemAutC);
   print(‘Its size is :‘,nops(TemAutC));
   print(‘Time needed(sec) :‘,extracttime());
end;

DualCode:=proc(CGen,charp)
# this procedure returns the generator matrix of the dual code of CGen

local C1,A,B,NumOfRow,NumOfCol,pivots,permlist,flag,loop;
 C1:=Gaussjord(CGen) mod charp;
 flag:=0;
 pivots:=FindPivots(C1);
 if pivots<>[seq(i,i=1..nops(pivots))] then
   C1:=ChangeMatrix(C1,pivots);
   permlist:=C1[1];
   C1:=C1[2];
   flag:=1;
 fi;
 NumOfRow:=linalg[rowdim](C1);
 NumOfCol:=linalg[coldim](C1);
 # get parity-check matrix A
 A:=linalg[submatrix](C1,1..NumOfRow,NumOfRow+1...NumOfCol);
 B:=linalg[diag](seq((-1 mod charp),i=1..NumOfRow));
 # compute -A and transpose it
 A:=map(modp,linalg[multiply](B,A),charp);
 A:=linalg[transpose](A);
 B:=linalg[diag](seq(1,i=1..(NumOfCol-NumOfRow)));
 A:=linalg[augment](A,B);
 A:=map(modp,A,charp);
 if flag=1 then
   for loop in permlist do;
     A:=PermuteCols(A,loop);
   od;
 fi;
 RETURN(A);
end;

DualCode2:=proc(FX,charp,DivG,DivD)
# this procedure will create the dual code of C(DivD,DivG)
# using simple matrix manipulation

local C1;
 C1:=Gaussjord(CreateMatrix(FX,charp,DivG,DivD)[2]) mod charp;
 RETURN(DualCode(C1,charp));
end;


DegreeDiv:=proc(DivD)

#This procedure returns the degree of the divisor DivD.

local NewD, loop,result;

  NewD:=ConvertDiv(DivD,0);
  result:=0;
  for loop in NewD do;
    result:=result+loop[1];
  od;
  RETURN(result);
end;

ChangeDiv:=proc(DivD,pivots)

#This routine will change DivD in such a way that the pivots of the
#matrix are in the first k position, where k=nops(pivots)

local degD,NewD1,NewD2,k,pos,loop;
  degD:=DegreeDiv(DivD);
  NewD1:=NULL;
  NewD2:=NULL;
  k:=nops(pivots);
  pos:=1;
  for loop from 1 to degD do;
     if pos<=k then
       if loop=pivots[pos] then
         NewD1:=NewD1,DivD[loop];
         pos:=pos+1;
       else
         NewD2:=NewD2,DivD[loop];
       fi;
     else
       NewD2:=NewD2,DivD[loop];
     fi;
  od;
  RETURN([NewD1,NewD2]);
end;


AnalyseCode:=proc(FX:polynom,charp,DivG,DivD)

# this procedure automates the analysis of a code
# depending on the number of parameters it is called with it’ll tell
# the user all the information which is required
# called with 4 arguments it determines the gen. mat. of C(DivG,DivD),
# its REF and
# called with 5 arguments it also determines the aut. group of C(DivG,DivD)
# called with 6 arguments it produces the REF’s of the two codes
# called with 7 arguments it also computes the autos


local NoParms,Code1,TemC1,DivQ,TemAutF,TemAutC,TemD,TemG,TimeUsed,numA5,
      degFX,genus,DimG,DualDim,TempC2,pivots,Str1,Str2;

  degFX:=degree(FX,X);
  genus:=(degFX-1)/2;
  DimG:=DegreeDiv(DivG)+1-genus;
  DualDim:=DegreeDiv(DivD)-DimG;

  NoParms:=nargs;
  DivQ:=FindPlaces(FX,charp);
  if (NoParms<=5) then
    print(‘G =‘,DivG);
    print(‘D =‘,DivD);
    print(‘a basis of L(G) & a generator matrix for C(G,D) are :‘);
    Code1:=CreateMatrix(FX,charp,DivG,DivD);
    print(Code1);
    print(‘REF of C(G,D)’s matrix is :‘);
    TemC1:=(Gaussjord(Code1[2]) mod charp);
    print(TemC1);
    if NoParms=5 then
       if DualDim<DimG then
          print(‘We will actually compute the automorphism group of the dual!‘);
          TempC2:=DualCode(TemC1,charp);
          print(‘The generator matrix of the dual is given by‘);
          print(TempC2);
          print(‘Its REF is given by‘);
```

```
          print((Gaussjord(TempC2) mod charp));
          AnalyseOutput(FX,charp,DivQ,DivG,DivD,TempC2,‘GD‘,‘G,D‘);
        else
          AnalyseOutput(FX,charp,DivQ,DivG,DivD,TemC1,‘GD‘,‘G,D‘);
        fi;
      fi;
    elif ((NoParms=6) or (NoParms=7)) then
      Code1:=CreateMatrix(FX,charp,DivG,DivD,args[5],args[6]);
      print(Code1);
      if (args[6]=0) then
        TemC1:=Gaussjord(Code1[2]) mod charp;
        print(‘REF of C(G,D)’s matrix is :‘);
        print(TemC1);
        if NoParms=7 then
          AnalyseOutput(FX,charp,DivQ,DivG,DivD,TemC1,‘GD‘,‘G,D‘);
        fi;
        TemC1:=Gaussjord(Code1[3]) mod charp;
        print(‘REF of C(G,D’)’s matrix is :‘);
        print(TemC1);
        if NoParms=7 then
          numA5:=nops(args[5]);
          if numA5=3 then
            TemD:=SigmaOfD(FX,charp,DivD,args[5]);
          elif numA5=2 then
            TemD:=ShiftDiv(FX,charp,DivD,args[5]);
          else
            TemD:=EndoOfD(FX,charp,DivD,args[5]);
          fi;
          AnalyseOutput(FX,charp,DivQ,DivG,TemD,TemC1,‘GD’‘,‘G,D’‘);
        fi;
      else
        TemC1:=Gaussjord(Code1[3]) mod charp;
        print(‘REF of C(G,D)’s matrix is :‘);
        print(TemC1);
        if NoParms=7 then
          AnalyseOutput(FX,charp,DivQ,DivG,DivD,TemC1,‘GD‘,‘G,D‘);
        fi;
        TemC1:=Gaussjord(Code1[4]) mod charp;
        print(‘REF of C(G’,D)’s matrix is :‘);
        print(TemC1);
        if NoParms=7 then
          numA5:=nops(args[5]);
          if numA5=3 then
            TemG:=SigmaOfD(FX,charp,DivG,args[5]);
          elif numA5=2 then
            TemG:=ShiftDiv(FX,charp,DivG,args[5]);
          else
            TemG:=EndoOfD(FX,charp,DivG,args[5]);
          fi;
          AnalyseOutput(FX,charp,DivQ,TemG,DivD,TemC1,‘G’D‘,‘G’,D‘);
        fi;
      fi;
    fi;
  fi;
end;

MultiplyMat:=proc(M,v)
#this procedure multiplies the columns c_i of the matrix M by v[i]

local NewMat,loop1,loop2,dimc,dimr,newv;

newv:=convert(v,vector);
dimc:=linalg[coldim](M);
dimr:=linalg[rowdim](M);
NewMat:=NULL;
for loop1 from 1 to dimr do;
 for loop2 from 1 to dimc do;
   NewMat:=NewMat,M[loop1,loop2]*newv[loop2];
 od;
od;
RETURN(linalg[matrix](dimr,dimc,[NewMat]));
end;
```

A.2 Klein.txt

This is the part of the program that deals with codes associated with the Klein Quartic.

```
Klein:=proc(charp:integer)

# this procedure will find the places of the
# Klein-Quartic function field associated with
# the equation y^7=(x^3)/(1-x). Note that we need
# charp<>7. Furthermore the places [Qinf], [0,0]
# and [1,0] are the only totally ramified places.

local result,loop1,loop2,temp1,temp2;

  result:=[Qinf],[0,0],[1,0];
  for loop1 from 2 to (charp-1) do;
     temp1:=(loop1^3)/(1-loop1) mod charp;
     temp2:=(Factor(X^7-temp1) mod charp);
     if type(temp2,‘*‘) then
       temp2:=[op(temp2)];
       for loop2 in temp2 do
           if degree(loop2,X)=1 then
             result:=result,[loop1,
                 (subs(X=0,loop2)mod charp)];
           fi;
       od;
     fi;
  od;
  RETURN([result]);
end;

CreateMatrix2:=proc(LofG,DivD,charp)

# this procedure will return a generator matrix of
# the code associated with LofG and DivD

  local BasisG,loop1,loop2,Dlist,len1,len2,C1,
        C1Max;

  C1:=NULL;

# the following checks whether DivD is the sum of
# distinct places

  Dlist:=ConvertDiv(DivD,0);
  for loop1 in Dlist do
    if loop1[1]<>1 then
      ERROR(‘D<>sum(Pi:i in list)‘);
    fi;
  od;
  BasisG:=LofG;
  len1:=nops(BasisG);
  len2:=nops(Dlist);

  # just return basis of L(DivG) and generator
  # matrix for C(DivG,DivD)

  for loop1 in BasisG do
    for loop2 in DivD do;
      C1:=C1,(subs({X=loop2[1],Y=loop2[2]},loop1)
          mod charp);
    od;
  od;
  C1Max:=linalg[matrix](len1,len2,[C1]);
  RETURN(BasisG,C1Max);
end;

AutoTest:=proc(P,charp)

# this procedure applies alpha(X)=1-1/X ,
# alpha(Y)=-X/Y^3 to the point P=[x1,y1]
# and returns alpha(P).

local tempx,tempy;
if P=[Qinf] then
  RETURN([1,0]);
elif P=[0,0] then
   RETURN([Qinf]);
elif P=[1,0] then
   RETURN([0,0]);
fi;
tempx:=1/(1-P[1]) mod charp;
tempy:=-tempx/P[2] mod charp;
tempy:=numtheory[mroot](tempy,3,charp);
RETURN([tempx,tempy]);
end;
```

A.3 FindH.txt

This is the program that computes the divisor H associated with the dual code of $\mathcal{C}_{\mathcal{L}}(D,G)$ provided certain conditions are satisfied.

```
FindH:=proc(F,charp,DivQ,DivG,DivD)

# This procedure returns a divisor H where C(D,H)
# is the dual of C(D,G). It is assumed that D
# either doesn’t contain any ramified
# points or it contains all of them except \Qinf.

local t,dt,lcdt,XValues, loop1,tflag,vtP;
 tflag:=0;
 if valplac([Qinf],DivD)<>0 then
    ERROR(‘wrong type of divisor‘);
 fi;
 XValuesR:=NULL;
 for loop1 from 2 to (degree(F,X)+1) do;
    TestP:=DivQ[loop1];
    if TestP[2]=0 then
      vtP:=valplac(TestP,DivD);
```

```
      if vtP=1 and tflag=0 then
        tflag:=1;
      elif vtP=0 and tflag=1 then
        ERROR(`wrong type of divisor`);
      fi;
      XValuesR:=XValuesR,TestP[1];
    else
      break;
    fi;
 od;
 XValuesR:=[XValuesR];
# print(XValuesR);
 XValues:=NULL;
 for loop1 in DivD do;
   xP:=loop1[1];
   yP:=-loop1[2] mod charp;
   if valplac([xP,yP],DivD)<>1 then
     ERROR(`wrong type of divisor`);
   fi;
   if yP<>0 then
     XValues:=XValues,loop1[1];
   fi;
 od;
 XValues:={XValues};
# print(XValues);
 t:=1;
 for loop1 in XValues do;
   t:=t*(X-loop1);
 od;
 t:=expand(t) mod charp;
# print(t);
 dt:=diff(t,X) mod charp;
 if tflag=1 then
    dt:=expand(2*F*dt+t*diff(F,X))/2 mod charp;
 fi;
 dtdeg:=degree(dt,X);
 lcdt:=coeff(dt,X,dtdeg);
 dt:=dt/lcdt mod charp;
 dt:=Factor(dt) mod charp;
# print(dt);
 DivH:=0;
 for loop1 in G do;
    DivH:=DivH-loop1;
 od;
# print(DivH);
 tdeg:=nops(XValues);
 if tflag=1 then
   tdeg:=tdeg+degree(F,X);
 fi;
 DivH:=DivH+dt+(2*tdeg-3)*[Qinf];
# print(DivH);
 if tflag=0 then
   for loop1 in XValuesR do;
      DivH:=DivH+[loop1,0];
    od;
 fi;
# print(DivH);
 RETURN(sort(DivH,X));
end;
```

Appendix B

C++ Program

This is the printout of the C++ program used to determine the automorphism group of a linear code over a finite field $\mathbb{F}_p$ given by its generator matrix. The code uses a number of matrix routines implemented by John Cremona.

```
//File LeonTest4.cc

# include <iostream.h>
# include <fstream.h>
# include "arith.h"
# include "matrix.h"

long pmod(long a,long p);        // Returns a mod p in (0..p-1)

int NewColUpdate(int* NewC,int flag, int* OldC);
void GenPerm(int n,char* FileName);

const int MAXSTACKLENGTH=30;
matrix _XStack[MAXSTACKLENGTH];
matrix CodeGenMat;
int _XStacklength=0;
int Rankr,n,charp;
const long *Old_entries,*New_entries,*Code_entries;

main(int argn,char** argv)
{
  if (argn!=3)
    {
      cerr<<"wrong number of arguments\n";
exit(1);
    }
  char test;
  ifstream in (argv[1]);
  if (!in)
    {
      cerr << "input file cannot be opened";
      exit( 1);
    }
  in >> charp;
  in >> Rankr;
  in >> n;
  _XStack[0]=idmat(Rankr);
  for (int i=1; i<=Rankr; i++)
    {
      _XStack[i]=_XStack[0];
    }
  CodeGenMat=matrix(Rankr,n);
  in >> CodeGenMat;
  Code_entries=CodeGenMat.get_mat();
  GenPerm(n,argv[2]);
}


inline long pmod (long a,long p)
{
 long c=a%p;
 if (c<0)
   {
     return (c+p);
   }
 return c;
}

int NewColUpdate(int flag, int col,long *OldMat, long *NewMat)
{
  int NewC1[Rankr+1],loop1,loop2,temp1,temp2,index;char test;
  long *trow, *tcol;

  //First entry in row col of CodeGenMat
  tcol=Code_entries+(col-1);
  if (flag<=Rankr)
    {
      if (flag==1)
{
  for (int i=1; i<=Rankr; i++)
    {
      NewC1[i]=*tcol;
      // Next entry of column col
      tcol+=n;
    }
}

      else
{
  trow=OldMat;
  for(loop1=1; loop1<=Rankr; loop1++)
    {
      temp1=0;
      for (loop2=1; loop2<=Rankr; loop2++)
{
  temp1=temp1+(*tcol)*(*trow);
                tcol+=n;
                trow++;
}
      NewC1[loop1]=pmod(temp1,charp);
      //Reset to first entry in column col of CodeGenMat
      tcol=Code_entries+(col-1);
    }
}
    }
  else
    {
      trow=OldMat;
      long *tcol2=Code_entries+(flag-1);

      for (loop1=1; loop1<=Rankr; loop1++)
{
          temp1=0;
  for (loop2=1; loop2<=Rankr; loop2++)
    {
      temp1=temp1+(*tcol)*(*trow);
              tcol+=n;
              trow++;
    }
  temp1=pmod(temp1,charp);
  if (temp1 != *tcol2)
    {
      return 0;
    }
  //Reset to first entry of column col of CodeGenMat
  tcol=Code_entries+(col-1);
        tcol2+=n;
}
      return 1;
    }

  // we have to convert NewC1 into a standard basis vecor ei where i=flag
  // changing OldMat into NewMat as we go along
  long *trow2,*trow3;
  loop1=flag;
  while ((loop1<=Rankr) && (NewC1[loop1]==0))
    {
      loop1++;
    }
  if (loop1>Rankr)
    {
      return 0;
    }
  else
    {
      if (loop1 != flag)
{
```

```
  NewC1[flag]=NewC1[loop1];

  //first entry of row flag of OldMat
         trow=OldMat+(flag-1)*Rankr;

  //first entry of row loop1 of OldMat
  trow2=OldMat+(loop1-1)*Rankr;

  //first entry of row flag of NewMat
  trow3=NewMat+(flag-1)*Rankr;

  for (loop2=1; loop2<=Rankr; loop2++)
    {
      temp1=((*trow)+(*trow2))%charp;
      *trow3=temp1;
             trow++;
      trow2++;
      trow3++;
    }
}
      else
{
  //first entry of row flag of OldMat
         trow=OldMat+(flag-1)*Rankr;

         //first entry of row flag of NewMat
         trow3=NewMat+(flag-1)*Rankr;

  for (loop2=1;loop2<=Rankr; loop2++)
    {
      *trow3=*trow;
              trow3++;
              trow++;
    }
}
      if (NewC1[flag]!=1)
{
  temp2=pmod(invmod(NewC1[flag],charp),charp);

  //first element of row flag of NewMat
  trow3=NewMat+(flag-1)*Rankr;

  for (loop1=1; loop1<=Rankr; loop1++)
    {
      temp1=pmod(((*trow3)*temp2),charp);
      *trow3=temp1;
             trow3++;
    }
}

      //first entry of row 1 ofOldMat
      trow=OldMat;

      //first entry of row flag of NewMat
      trow2=NewMat+(flag-1)*Rankr;

      //first entry of row 1 of NewMat
      trow3=NewMat;

      for (loop1=1; loop1<flag; loop1++)
{
  temp1=NewC1[loop1];
  if (temp1 != 0)
    {
      for (loop2=1; loop2<=Rankr; loop2++)
{
  temp2=pmod((*trow)-temp1*(*trow2),charp);
  *trow3=temp2;
                  trow++;
                  trow2++;
                  trow3++;
}

      //reset to first entry of row flag of NewMat
      trow2=NewMat+(flag-1)*Rankr;

    }
  else
    {
      for (loop2=1;loop2<=Rankr; loop2++)
{
  *trow3=*trow;
                  trow3++;
                  trow++;
}
    }
}

      //first entry of row (flag+1) of OldMat
      trow=OldMat+flag*Rankr;

      //first entry of row flag of NewMat
      trow2=NewMat+(flag-1)*Rankr;

      //first entry of row (flag+1) of NewMat
      trow3=NewMat+flag*Rankr;


      for (loop1=flag+1; loop1<=Rankr; loop1++)
{
  temp1=NewC1[loop1];
  if (temp1 != 0)
    {
      for (loop2=1; loop2<=Rankr; loop2++)
{
  temp2=pmod(((*trow)-temp1*(*trow2)),charp);
  *trow3=temp2;
                  trow++;
  trow2++;
                  trow3++;
}

      //reset to first entry of row flag of NewMat
      trow2=NewMat+(flag-1)*Rankr;

    }
  else
    {
      for (loop2=1;loop2<=Rankr; loop2++)
{
  *trow3=*trow;
  trow3++;
  trow++;
}
    }
}
      return 1;
    }
}
void GenPerm(int n,char* FileName)
{
  int d=1,beta[n+1],tempbeta,CheckFlag[n+1],testflag,i;
  long *tX1, *tX2;
  ofstream out(FileName);
  if (!out)
    {
      cerr << "cannot open output file\n";
      exit(1);
    }

  beta[1]=0;
  for (i=0; i<=n; i++)
    {
      CheckFlag[i]=0;
      beta[i]=0;
    }
  CheckFlag[0]=1;
  while (d>0)
    {
      tempbeta=beta[d];
      tempbeta++;
      while(tempbeta<=n && CheckFlag[tempbeta]==1)
{
  tempbeta++;
}
      if (tempbeta<=n)
{
  beta[d]=tempbeta;
  CheckFlag[tempbeta]=1;
          if (d<=Rankr)
    {
      tX1=_XStack[d-1].get_mat();
      tX2=_XStack[d].get_mat();
      testflag=NewColUpdate(d,tempbeta,tX1,tX2);
    }
  else
    {
      tX1=_XStack[Rankr].get_mat();
      tX2=_XStack[0].get_mat();
      testflag=NewColUpdate(d,tempbeta,tX1,tX2);
    }
  if (d<n)
    {
      if (testflag)
{
  d++;
  beta[d]=0;
}
      else
{
  CheckFlag[tempbeta]=0;
}
    }
        else
    {
      if (testflag)
{
  // here we'll collect all automorphisms in a list
  out<< " ";
                  for (i=1; i<=n; i++)
    {
      out<<beta[i]<<"\n";
    }
}
      CheckFlag[tempbeta]=0;
      d--;
      CheckFlag[beta[d]]=0;
    }
}
      else
{
  d--;
  CheckFlag[beta[d]]=0;
}
```

```
    }
}
```

Appendix C

MAPLE Sessions

The following are printouts of MAPLE sessions which illustrate how to use my MAPLE routines. Furthermore, they contain results that were quoted in Chapter 9.

C.1 Example 9.5.2 and Example 9.5.3

```
> F:=X^3-X;
                                    3
                                F := X  - X

------------------------------------------------------------------------------
> Q:=FindPlaces(F,17);
  Q := [[Qinf], [0, 0], [1, 0], [16, 0], [4, 3], [4, 14], [5, 1], [5, 16],

      [7, 8], [7, 9], [10, 2], [10, 15], [12, 4], [12, 13], [13, 5], [13, 12]]

------------------------------------------------------------------------------
> D:=CreateD(Q,[seq(i,i=5..16)]);
   D := [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15],

       [12, 4], [12, 13], [13, 5], [13, 12]]

------------------------------------------------------------------------------
> counter:=0;\
for looper.1 from 1 to 9 do;\
  for looper.2 from 0 to looper.1 do;\
    for looper.3 from 0 to looper.1 do;\
      for looper.4 from 0 to looper.1 do;\
        checkdegree:=(looper.1+looper.2+looper.3+looper.4);\
        if ((checkdegree>2) and (checkdegree<10)) then\
           Glist:=[looper.1,1];\
           for loopertmp from 2 to 4 do;\
             if looper.loopertmp>0 then\
                Glist:=Glist,[looper.loopertmp,loopertmp];\
             fi;\
           od;\
           counter:=counter+1;\
           print(`Example No.:`,counter);\
           G:=CreateG(Q,[Glist]);print(G);\
           AnalyseCode(F,17,G,D,1);\
        fi;\
      od;\
    od;\
  od;\
od;
                                  counter := 0

                                Example No.:, 1

                             [[Qinf], [1, 0], [16, 0]]

                          G =, [[Qinf], [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

 [12, 13], [13, 5], [13, 12]]

        a basis of L(G) & a generator matrix for C(G,D) are :

                     [ 1   1   1   1  1  1   1   1   1   1   1   1 ]
       Y      Y      [                                             ]
 [1, -----, ------], [ 1  16  13   4  7  10   4  13   5  12  16   1 ]
    X - 1  X - 16    [                                             ]
                     [ 4  13   3  14  1  16  11   6  16   1   4  13 ]

                        REF of C(G,D)'s matrix is :

                [ 1  0  0   1   5  13  3  15   0   1  6  12 ]
                [                                           ]
                [ 0  1  0   1   1   0  9   9   3  15  8  10 ]
                [                                           ]
                [ 0  0  1  16  12   5  6  11  15   2  4  13 ]

                          AutGD(F) is given by :

 [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                             Its size is :, 4

                      C(G,D)'s automorphism group is :

                 [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                     [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                     [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                     [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                             Its size is :, 4

                       Time needed(sec) :,         0.2

[...]

                              Example No.:, 4

                       [[Qinf], [0, 0], [1, 0], [16, 0]]

                    G =, [[Qinf], [0, 0], [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

 [12, 13], [13, 5], [13, 12]]

        a basis of L(G) & a generator matrix for C(G,D) are :

                                [ 1   1   1   1  1  1   1   1   1   1   1   1 ]
                                [                                             ]
         Y      Y               [ 5  12   7  10  6  11   7  10   6  11   3  14 ]
[1, Y/X, -----, ------], [                                                    ]
        X - 1  X - 16           [ 1  16  13   4  7  10   4  13   5  12  16   1 ]
                                [                                             ]
                                [ 4  13   3  14  1  16  11   6  16   1   4  13 ]

                           REF of C(G,D)'s matrix is :

                   [ 1  0  0   1  0   1  5  13   3  15  12  6 ]
                   [                                          ]
                   [ 0  1  0   1  0   1  6  12   7  11  16  2 ]
                   [                                          ]
                   [ 0  0  1  16  0   0  4  13  12   5  15  2 ]
                   [                                          ]
                   [ 0  0  0   0  1  16  3  14  13   4   9  8 ]
```

```
                        AutGD(F) is given by :

  [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]],

      [13, 0, [0, 0]], [13, 1, [0, 0]], [13, 2, [0, 0]], [13, 3, [0, 0]],

      [13, 0, [1, 0]], [13, 1, [1, 0]], [13, 2, [1, 0]], [13, 3, [1, 0]],

      [13, 0, [16, 0]], [13, 1, [16, 0]], [13, 2, [16, 0]], [13, 3, [16, 0]]]

                           Its size is :, 16

                    C(G,D)'s automorphism group is :

               [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                   [1, 2, 5, 6, 7, 8, 9, 10, 3, 4, 12, 11],

                   [1, 2, 7, 8, 9, 10, 3, 4, 5, 6, 11, 12],

                   [1, 2, 9, 10, 3, 4, 5, 6, 7, 8, 12, 11],

                   [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                   [2, 1, 6, 5, 8, 7, 10, 9, 4, 3, 11, 12],

                   [2, 1, 8, 7, 10, 9, 4, 3, 6, 5, 12, 11],

                   [2, 1, 10, 9, 4, 3, 6, 5, 8, 7, 11, 12],

                   [11, 12, 3, 4, 10, 9, 7, 8, 6, 5, 1, 2],

                   [11, 12, 6, 5, 3, 4, 10, 9, 7, 8, 2, 1],

                   [11, 12, 7, 8, 6, 5, 3, 4, 10, 9, 1, 2],

                   [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                   [12, 11, 5, 6, 4, 3, 9, 10, 8, 7, 1, 2],

                   [12, 11, 4, 3, 9, 10, 8, 7, 5, 6, 2, 1],

                   [12, 11, 8, 7, 5, 6, 4, 3, 9, 10, 2, 1],

                   [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                           Its size is :, 16

                      Time needed(sec) :,         0.8

[...]

                           Example No.:, 30

                  [2 [Qinf], 2 [0, 0], 2 [1, 0], 2 [16, 0]]

                G =, [2 [Qinf], 2 [0, 0], 2 [1, 0], 2 [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                      2             2                2
                     Y      Y      Y        Y        Y
            [1, X, Y/X, ----, -----, --------, ------, ---------],
                      2   X - 1        2  X - 16         2
                     X          (X - 1)          (X - 16)

                [  1   1   1   1   1   1   1   1   1   1   1   1 ]
                [                                               ]
                [  4   4   5   5   7   7  10  10  12  12  13  13 ]
                [                                               ]
                [  5  12   7  10   6  11   7  10   6  11   3  14 ]
                [                                               ]
                [  8   8  15  15   2   2  15  15   2   2   9   9 ]
                [                                               ]
                [  1  16  13   4   7  10   4  13   5  12  16   1 ]
                [                                               ]
                [  1   1  16  16  15  15  16  16   8   8   1   1 ]
                [                                               ]
                [  4  13   3  14   1  16  11   6  16   1   4  13 ]
                [                                               ]
                [ 16  16   9   9   1   1   2   2   1   1  16  16 ]

                           REF of C(G,D)'s matrix is :

                  [ 1  0  0  0  0  0  0   1  0   4   3   7 ]
                  [                                        ]
                  [ 0  1  0  0  0  0  0  16  0  13  13   9 ]
                  [                                        ]
                  [ 0  0  1  0  0  0  0  13  0   5   8  10 ]
                  [                                        ]
                  [ 0  0  0  1  0  0  0   4  0  12   1  16 ]
                  [                                        ]
                  [ 0  0  0  0  1  0  0  14  0   4   5  13 ]
                  [                                        ]
                  [ 0  0  0  0  0  1  0   3  0  13   4  13 ]
                  [                                        ]
                  [ 0  0  0  0  0  0  1   1  0   0   9   9 ]
                  [                                        ]
                  [ 0  0  0  0  0  0  0   0  1   1   9   9 ]

               We will actually compute the automorphism group of the dual!

                       The generator matrix of the dual is given by

                  [ 16  1   4  13   3  14  16  1   0  0  0  0 ]
                  [                                           ]
                  [ 13  4  12   5  13   4   0  0  16  1  0  0 ]
                  [                                           ]
                  [ 14  4   9  16  12  13   8  0   8  0  1  0 ]
                  [                                           ]
                  [ 10  8   7   1   4   4   8  0   8  0  0  1 ]

                                Its REF is given by

                  [ 1  0  0   8  0   8   6   2   7  1  12   6 ]
                  [                                           ]
                  [ 0  1  0   8  0   8  14  11   5  3  16   2 ]
                  [                                           ]
                  [ 0  0  1  16  0   0   4  13  12  5   4  13 ]
                  [                                           ]
                  [ 0  0  0   0  1  16   3  14  13  4  16   1 ]

                           AutGD(F) is given by :

      [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]],

          [13, 0, [0, 0]], [13, 1, [0, 0]], [13, 2, [0, 0]], [13, 3, [0, 0]],

          [13, 0, [1, 0]], [13, 1, [1, 0]], [13, 2, [1, 0]], [13, 3, [1, 0]],

          [13, 0, [16, 0]], [13, 1, [16, 0]], [13, 2, [16, 0]], [13, 3, [16, 0]]]

                                Its size is :, 16

                        C(G,D)'s automorphism group is :

                   [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                       [1, 2, 5, 6, 7, 8, 9, 10, 3, 4, 12, 11],

                       [1, 2, 7, 8, 9, 10, 3, 4, 5, 6, 11, 12],

                       [1, 2, 9, 10, 3, 4, 5, 6, 7, 8, 12, 11],

                       [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                       [2, 1, 6, 5, 8, 7, 10, 9, 4, 3, 11, 12],

                       [2, 1, 8, 7, 10, 9, 4, 3, 6, 5, 12, 11],

                       [2, 1, 10, 9, 4, 3, 6, 5, 8, 7, 11, 12],

                       [11, 12, 3, 4, 10, 9, 7, 8, 6, 5, 1, 2],

                       [11, 12, 6, 5, 3, 4, 10, 9, 7, 8, 2, 1],

                       [11, 12, 7, 8, 6, 5, 3, 4, 10, 9, 1, 2],

                       [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                       [12, 11, 5, 6, 4, 3, 9, 10, 8, 7, 1, 2],

                       [12, 11, 4, 3, 9, 10, 8, 7, 5, 6, 2, 1],

                       [12, 11, 8, 7, 5, 6, 4, 3, 9, 10, 2, 1],

                       [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                                Its size is :, 16

                          Time needed(sec) :,         0.7

                                Example No.:, 31

                                   [3 [Qinf]]

                                    G =, [3 [Qinf]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                        [ 1   1  1   1  1  1   1   1   1   1   1 ]
                        [                                         ]
                  [1, X, Y], [ 4   4  5   5  7  7  10  10  12  12  13  13 ]
                        [                                         ]
                        [ 3  14  1  16  8  9   2  15   4  13   5  12 ]

                           REF of C(G,D)'s matrix is :

                        [ 1  0  0  11  14   0  11  16  10  3  1  5 ]
                        [                                         ]
                        [ 0  1  0   6   1  15   1  13   0  7  8  4 ]
                        [                                         ]
```

```
                [ 0  0  1   1   3   3   6   6   8  8  9  9 ]

                          AutGD(F) is given by :

      [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                               Its size is :, 4

                        C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                               Its size is :, 4

                          Time needed(sec) :,         0.2

[...]

                               Example No.:, 43

                             [3 [Qinf], 3 [1, 0]]

                           G =, [3 [Qinf], 3 [1, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

             a basis of L(G) & a generator matrix for C(G,D) are :

                                 2         3
                           Y     Y         Y
             [1, X, Y, -----, --------, --------],
                      X - 1         2         3
                                (X - 1)   (X - 1)

                [ 1   1   1   1   1   1   1   1   1   1   1   1 ]
                [                                               ]
                [ 4   4   5   5   7   7  10  10  12  12  13  13 ]
                [                                               ]
                [ 3  14   1  16   8   9   2  15   4  13   5  12 ]
                [                                               ]
                [ 1  16  13   4   7  10   4  13   5  12  16   1 ]
                [                                               ]
                [ 1   1  16  16  15  15  16  16   8   8   1   1 ]
                [                                               ]
                [ 1  16   4  13   3  14  13   4   6  11  16   1 ]

                          REF of C(G,D)'s matrix is :

                 [ 1  0  0  0  0  0   7   6   8   6  0  4 ]
                 [                                         ]
                 [ 0  1  0  0  0  0   6   7   6   8  4  0 ]
                 [                                         ]
                 [ 0  0  1  0  0  0   0  13   4  15  5  3 ]
                 [                                         ]
                 [ 0  0  0  1  0  0  13   0  15   4  3  5 ]
                 [                                         ]
                 [ 0  0  0  0  1  0   1   8   9  10  5  1 ]
                 [                                         ]
                 [ 0  0  0  0  0  1   8   1  10   9  1  5 ]

                          AutGD(F) is given by :

      [[13, 0, [Qinf]], [13, 2, [Qinf]], [13, 0, [1, 0]], [13, 2, [1, 0]]]

                               Its size is :, 4

                        C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 7, 8, 6, 5, 3, 4, 10, 9, 1, 2],

                    [12, 11, 8, 7, 5, 6, 4, 3, 9, 10, 2, 1]]

                               Its size is :, 4

                          Time needed(sec) :,      2:15.9

[...]

                               Example No.:, 46

                          [3 [Qinf], 3 [1, 0], 3 [16, 0]]

                        G =, [3 [Qinf], 3 [1, 0], 3 [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

             a basis of L(G) & a generator matrix for C(G,D) are :

                                 2         3                 2         3
                           Y     Y         Y         Y       Y         Y
             [1, X, Y, -----, --------, --------, ------, ---------, ---------],
                      X - 1         2         3  X - 16          2         3
                                (X - 1)   (X - 1)          (X - 16)  (X - 16)

                [ 1   1   1   1   1   1   1   1   1   1   1   1 ]
                [                                               ]
                [ 4   4   5   5   7   7  10  10  12  12  13  13 ]
                [                                               ]
                [ 3  14   1  16   8   9   2  15   4  13   5  12 ]
                [                                               ]
                [ 1  16  13   4   7  10   4  13   5  12  16   1 ]
                [                                               ]
                [ 1   1  16  16  15  15  16  16   8   8   1   1 ]
                [                                               ]
                [ 1  16   4  13   3  14  13   4   6  11  16   1 ]
                [                                               ]
                [ 4  13   3  14   1  16  11   6  16   1   4  13 ]
                [                                               ]
                [ 16  16   9   9   1   1   2   2   1   1  16  16 ]
                [                                               ]
                [ 13   4  10   7   1  16   5  12  16   1  13   4 ]

                          REF of C(G,D)'s matrix is :

                    [ 1  0  0  0  0  0  0  0  0   3   3   6 ]
                    [                                       ]
                    [ 0  1  0  0  0  0  0  0  0   3   7   2 ]
                    [                                       ]
                    [ 0  0  1  0  0  0  0  0  0   8   0  13 ]
                    [                                       ]
                    [ 0  0  0  1  0  0  0  0  0   8  10   3 ]
                    [                                       ]
                    [ 0  0  0  0  1  0  0  0  0  14   6  10 ]
                    [                                       ]
                    [ 0  0  0  0  0  1  0  0  0  14   9   7 ]
                    [                                       ]
                    [ 0  0  0  0  0  0  1  0  0  10   9   5 ]
                    [                                       ]
                    [ 0  0  0  0  0  0  0  1  0  10  14   0 ]
                    [                                       ]
                    [ 0  0  0  0  0  0  0  0  1  16  11   6 ]

          We will actually compute the automorphism group of the dual!

                  The generator matrix of the dual is given by

                    [ 14  14  9   9   3   3   7  7   1  1  0  0 ]
                    [                                           ]
                    [ 14  10  0   7  11   8   8  3   6  0  1  0 ]
                    [                                           ]
                    [ 11  15  4  14   7  10  12  0  11  0  0  1 ]

                               Its REF is given by

                    [ 1  0  0   6   8  3   8  11  0   7  10  14 ]
                    [                                           ]
                    [ 0  1  0  11  12  0  10   7  4  14  15  11 ]
                    [                                           ]
                    [ 0  0  1   1   7  7   3   3  9   9  14  14 ]

                          AutGD(F) is given by :

      [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                               Its size is :, 4

                        C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                               Its size is :, 4

                          Time needed(sec) :,         0.2

[...]

                               Example No.:, 52

                          [3 [Qinf], [0, 0], [1, 0], [16, 0]]

                        G =, [3 [Qinf], [0, 0], [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

             a basis of L(G) & a generator matrix for C(G,D) are :

                                [ 1   1   1   1  1   1   1   1   1   1   1   1 ]
```

```
                                [                                       ]
                                [ 4   4   5   5  7   7  10  10  12  12  13  13 ]
                                [                                       ]
                     Y      Y   [ 3  14   1  16  8   9   2  15   4  13   5  12 ]
[1, X, Y, Y/X, -----, ------], [                                       ]
                   X - 1  X - 16[ 5  12   7  10  6  11   7  10   6  11   3  14 ]
                                [                                       ]
                                [ 1  16  13   4  7  10   4  13   5  12  16   1 ]
                                [                                       ]
                                [ 4  13   3  14  1  16  11   6  16   1   4  13 ]

                          REF of C(G,D)'s matrix is :

                     [ 1  0  0  0  0  15  0  12  16  11   3   6 ]
                     [                                         ]
                     [ 0  1  0  0  0  15  0  12   9   1   1   8 ]
                     [                                         ]
                     [ 0  0  1  0  0   3  0   6   2   6   4   5 ]
                     [                                         ]
                     [ 0  0  0  1  0   3  0   6   0   8  13  13 ]
                     [                                         ]
                     [ 0  0  0  0  1  16  0   0  14   3   8   9 ]
                     [                                         ]
                     [ 0  0  0  0  0   0  1  16  11   6   6  11 ]

                           AutGD(F) is given by :

     [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                              Its size is :, 4

                       C(G,D)'s automorphism group is :

                  [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                      [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                      [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                      [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                              Its size is :, 4

                        Time needed(sec) :,      2:32.2

[...]

                              Example No.:, 72

                    [3 [Qinf], 2 [0, 0], 2 [1, 0], 2 [16, 0]]

                  G =, [3 [Qinf], 2 [0, 0], 2 [1, 0], 2 [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                      2            2               2
                     Y     Y      Y       Y       Y
          [1, X, Y, Y/X, ----, -----, --------, ------, ---------],
                      2   X - 1        2  X - 16         2
                     X         (X - 1)          (X - 16)

                [  1   1   1   1   1   1   1   1   1   1   1   1 ]
                [                                               ]
                [  4   4   5   5   7   7  10  10  12  12  13  13 ]
                [                                               ]
                [  3  14   1  16   8   9   2  15   4  13   5  12 ]
                [                                               ]
                [  5  12   7  10   6  11   7  10   6  11   3  14 ]
                [                                               ]
                [  8   8  15  15   2   2  15  15   2   2   9   9 ]
                [                                               ]
                [  1  16  13   4   7  10   4  13   5  12  16   1 ]
                [                                               ]
                [  1   1  16  16  15  15  16  16   8   8   1   1 ]
                [                                               ]
                [  4  13   3  14   1  16  11   6  16   1   4  13 ]
                [                                               ]
                [ 16  16   9   9   1   1   2   2   1   1  16  16 ]

                          REF of C(G,D)'s matrix is :

                    [ 1  0  0  0  0  0  0  0  0  10   3   1 ]
                    [                                       ]
                    [ 0  1  0  0  0  0  0  0  0   7  13  15 ]
                    [                                       ]
                    [ 0  0  1  0  0  0  0  0  0  15   8   0 ]
                    [                                       ]
                    [ 0  0  0  1  0  0  0  0  0   2   1   9 ]
                    [                                       ]
                    [ 0  0  0  0  1  0  0  0  0   3   5  14 ]
                    [                                       ]
                    [ 0  0  0  0  0  1  0  0  0  14   4  12 ]
                    [                                       ]
                    [ 0  0  0  0  0  0  1  0  0   6   9   3 ]
                    [                                       ]
                    [ 0  0  0  0  0  0  0  1  0  11   0   6 ]

                    [                                       ]
                    [ 0  0  0  0  0  0  0  0  1   1   9   9 ]

          We will actually compute the automorphism group of the dual!

                  The generator matrix of the dual is given by

                [  7  10  2  15  14   3  11   6  16  1  0  0 ]
                [                                           ]
                [ 14   4  9  16  12  13   8   0   8  0  1  0 ]
                [                                           ]
                [ 16   2  0   8   3   5  14  11   8  0  0  1 ]

                              Its REF is given by

                [ 1  0  0   8   8   0  13  12  9  16   4  14 ]
                [                                           ]
                [ 0  1  0   8  14  11   5   3  0   8   2  16 ]
                [                                           ]
                [ 0  0  1  16  11   6   3  14  2  15  10   7 ]

                           AutGD(F) is given by :

     [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                              Its size is :, 4

                       C(G,D)'s automorphism group is :

                  [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                      [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                      [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                      [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                              Its size is :, 4

                        Time needed(sec) :,      0.3

[...]

                              Example No.:, 75

                          [3 [Qinf], 3 [0, 0]]

                        G =, [3 [Qinf], 3 [0, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                                [ 1   1   1   1   1   1   1   1   1   1   1   1 ]
                                [                                               ]
                                [ 4   4   5   5   7   7  10  10  12  12  13  13 ]
                     2     3    [                                               ]
                    Y     Y     [ 3  14   1  16   8   9   2  15   4  13   5  12 ]
[1, X, Y, Y/X, ----, ----], [                                               ]
                    2     3     [ 5  12   7  10   6  11   7  10   6  11   3  14 ]
                   X     X      [                                               ]
                                [ 8   8  15  15   2   2  15  15   2   2   9   9 ]
                                [                                               ]
                                [ 6  11   3  14  12   5   3  14  12   5  10   7 ]

                          REF of C(G,D)'s matrix is :

                    [ 1  0  0  0  0  0  16   3   9  10  11   7 ]
                    [                                         ]
                    [ 0  1  0  0  0  0   3  16  10   9   7  11 ]
                    [                                         ]
                    [ 0  0  1  0  0  0   4   0   4  16  15   6 ]
                    [                                         ]
                    [ 0  0  0  1  0  0   0   4  16   4   6  15 ]
                    [                                         ]
                    [ 0  0  0  0  1  0  13  16  13   0  14  16 ]
                    [                                         ]
                    [ 0  0  0  0  0  1  16  13   0  13  16  14 ]

                           AutGD(F) is given by :

     [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]],

         [13, 0, [0, 0]], [13, 1, [0, 0]], [13, 2, [0, 0]], [13, 3, [0, 0]]]

                              Its size is :, 8

                       C(G,D)'s automorphism group is :

                  [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                      [1, 2, 7, 8, 9, 10, 3, 4, 5, 6, 11, 12],

                      [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                      [2, 1, 8, 7, 10, 9, 4, 3, 6, 5, 12, 11],

                      [11, 12, 6, 5, 3, 4, 10, 9, 7, 8, 2, 1],
```

```
                    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                    [12, 11, 5, 6, 4, 3, 9, 10, 8, 7, 1, 2],

                    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                             Its size is :, 8

                       Time needed(sec) :,      3:01.1

                             Example No.:, 76

                       [3 [Qinf], 3 [0, 0], [16, 0]]

                      G =, [3 [Qinf], 3 [0, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

    [12, 13], [13, 5], [13, 12]]

          a basis of L(G) & a generator matrix for C(G,D) are :

                          2     3
                         Y     Y      Y
        [1, X, Y, Y/X, ----, ----, ------],
                          2     3   X - 16
                         X     X

            [ 1   1   1   1   1   1   1   1   1   1   1   1 ]
            [                                               ]
            [ 4   4   5   5   7   7  10  10  12  12  13  13 ]
            [                                               ]
            [ 3  14   1  16   8   9   2  15   4  13   5  12 ]
            [                                               ]
            [ 5  12   7  10   6  11   7  10   6  11   3  14 ]
            [                                               ]
            [ 8   8  15  15   2   2  15  15   2   2   9   9 ]
            [                                               ]
            [ 6  11   3  14  12   5   3  14  12   5  10   7 ]
            [                                               ]
            [ 4  13   3  14   1  16  11   6  16   1   4  13 ]

                    REF of C(G,D)'s matrix is :

             [ 1  0  0  0  0  0  0   2   1   1   8  10 ]
             [                                         ]
             [ 0  1  0  0  0  0  0   2   0   2  16   2 ]
             [                                         ]
             [ 0  0  1  0  0  0  0   4   2   1  10  11 ]
             [                                         ]
             [ 0  0  0  1  0  0  0   4  16   4   6  15 ]
             [                                         ]
             [ 0  0  0  0  1  0  0  12  15  15   2  11 ]
             [                                         ]
             [ 0  0  0  0  0  1  0  12   9   4  13   0 ]
             [                                         ]
             [ 0  0  0  0  0  0  1  16   9   8  14   3 ]

        We will actually compute the automorphism group of the dual!

                The generator matrix of the dual is given by

                [ 15  15  13  13   5   5   1  1  0  0  0  0 ]
                [                                           ]
                [ 16   0  15   1   2   8   8  0  1  0  0  0 ]
                [                                           ]
                [ 16  15  16  13   2  13   9  0  0  1  0  0 ]
                [                                           ]
                [  9   1   7  11  15   4   3  0  0  0  1  0 ]
                [                                           ]
                [  7  15   6   2   6   0  14  0  0  0  0  1 ]

                           Its REF is given by

                [ 1  0  0  0  0   1   5   4   2   7  11   3 ]
                [                                           ]
                [ 0  1  0  0  0  16  16   0  11   6   5  13 ]
                [                                           ]
                [ 0  0  1  0  0   9   6   9   1  11   0  14 ]
                [                                           ]
                [ 0  0  0  1  0   8  15  12  13   3  15   1 ]
                [                                           ]
                [ 0  0  0  0  1   1   5   5  13  13  15  15 ]

                         AutGD(F) is given by :

     [[13, 0, [Qinf]], [13, 2, [Qinf]], [13, 1, [0, 0]], [13, 3, [0, 0]]]

                             Its size is :, 4

                    C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 6, 5, 3, 4, 10, 9, 7, 8, 2, 1],

                    [12, 11, 5, 6, 4, 3, 9, 10, 8, 7, 1, 2]]

                             Its size is :, 4

                       Time needed(sec) :,        20.0

[...]

                             Example No.:, 78

                       [3 [Qinf], 3 [0, 0], 3 [16, 0]]

                      G =, [3 [Qinf], 3 [0, 0], 3 [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

    [12, 13], [13, 5], [13, 12]]

          a basis of L(G) & a generator matrix for C(G,D) are :

                          2     3                2          3
                         Y     Y      Y         Y          Y
        [1, X, Y, Y/X, ----, ----, ------, ---------, ---------],
                          2     3   X - 16          2          3
                         X     X            (X - 16)   (X - 16)

            [  1   1   1   1   1   1   1   1   1   1   1   1 ]
            [                                               ]
            [  4   4   5   5   7   7  10  10  12  12  13  13 ]
            [                                               ]
            [  3  14   1  16   8   9   2  15   4  13   5  12 ]
            [                                               ]
            [  5  12   7  10   6  11   7  10   6  11   3  14 ]
            [                                               ]
            [  8   8  15  15   2   2  15  15   2   2   9   9 ]
            [                                               ]
            [  6  11   3  14  12   5   3  14  12   5  10   7 ]
            [                                               ]
            [  4  13   3  14   1  16  11   6  16   1   4  13 ]
            [                                               ]
            [ 16  16   9   9   1   1   2   2   1   1  16  16 ]
            [                                               ]
            [ 13   4  10   7   1  16   5  12  16   1  13   4 ]

                    REF of C(G,D)'s matrix is :

                 [ 1  0  0  0  0  0  0  0  0  0  15   6   9 ]
                 [                                           ]
                 [ 0  1  0  0  0  0  0  0  0  0  15   2  13 ]
                 [                                           ]
                 [ 0  0  1  0  0  0  0  0  0  0  12   6   9 ]
                 [                                           ]
                 [ 0  0  0  1  0  0  0  0  0  0  12   0  15 ]
                 [                                           ]
                 [ 0  0  0  0  1  0  0  0  0  0   6  13  16 ]
                 [                                           ]
                 [ 0  0  0  0  0  1  0  0  0  0   6   3   9 ]
                 [                                           ]
                 [ 0  0  0  0  0  0  0  1  0  0   2  10   0 ]
                 [                                           ]
                 [ 0  0  0  0  0  0  0  0  1  0   2   7   3 ]
                 [                                           ]
                 [ 0  0  0  0  0  0  0  0  0  1  16   5  12 ]

            We will actually compute the automorphism group of the dual!

                    The generator matrix of the dual is given by

                   [  2   2   5  5  11  11  15  15   1  1  0  0 ]
                   [                                           ]
                   [ 11  15  11  0   4  14   7  10  12  0  1  0 ]
                   [                                           ]
                   [  8   4   8  2   1   8   0  14   5  0  0  1 ]

                               Its REF is given by

                    [ 1  0  0   7  0  6   8   3  8  11  14  10 ]
                    [                                         ]
                    [ 0  1  0  10  9  3   7  12  3   0   1   5 ]
                    [                                         ]
                    [ 0  0  1   1  2  2  14  14  6   6  11  11 ]

                             AutGD(F) is given by :

         [[13, 0, [Qinf]], [13, 2, [Qinf]], [13, 1, [0, 0]], [13, 3, [0, 0]]]

                                 Its size is :, 4

                        C(G,D)'s automorphism group is :

                    [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                        [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                        [11, 12, 6, 5, 3, 4, 10, 9, 7, 8, 2, 1],

                        [12, 11, 5, 6, 4, 3, 9, 10, 8, 7, 1, 2]]

                                 Its size is :, 4

                           Time needed(sec) :,         0.2
```

```
[...]

                          Example No.:, 80

                    [3 [Qinf], 3 [0, 0], [1, 0], [16, 0]]

                  G =, [3 [Qinf], 3 [0, 0], [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

           a basis of L(G) & a generator matrix for C(G,D) are :

                          2    3
                         Y    Y     Y       Y
           [1, X, Y, Y/X, ----, ----, -----, ------],
                          2    3   X - 1  X - 16
                         X    X

              [ 1   1   1   1   1   1   1   1   1   1   1   1 ]
              [                                               ]
              [ 4   4   5   5   7   7  10  10  12  12  13  13 ]
              [                                               ]
              [ 3  14   1  16   8   9   2  15   4  13   5  12 ]
              [                                               ]
              [ 5  12   7  10   6  11   7  10   6  11   3  14 ]
              [                                               ]
              [ 8   8  15  15   2   2  15  15   2   2   9   9 ]
              [                                               ]
              [ 6  11   3  14  12   5   3  14  12   5  10   7 ]
              [                                               ]
              [ 1  16  13   4   7  10   4  13   5  12  16   1 ]
              [                                               ]
              [ 4  13   3  14   1  16  11   6  16   1   4  13 ]

                       REF of C(G,D)'s matrix is :

               [ 1  0  0  0  0  0  0   2  0   2  12   6 ]
               [                                         ]
               [ 0  1  0  0  0  0  0   2  0   2  16   2 ]
               [                                         ]
               [ 0  0  1  0  0  0  0   4  0   3   1   3 ]
               [                                         ]
               [ 0  0  0  1  0  0  0   4  0   3   2   2 ]
               [                                         ]
               [ 0  0  0  0  1  0  0  12  0  13  11   2 ]
               [                                         ]
               [ 0  0  0  0  0  1  0  12  0  13  15  15 ]
               [                                         ]
               [ 0  0  0  0  0  0  1  16  0   0  16   1 ]
               [                                         ]
               [ 0  0  0  0  0  0  0   0  1  16  13   4 ]

        We will actually compute the automorphism group of the dual!

           The generator matrix of the dual is given by

              [ 15  15  13  13   5  5   1  1   0  0  0  0 ]
              [                                           ]
              [ 15  15  14  14   4  4   0  0   1  1  0  0 ]
              [                                           ]
              [  5   1  16  15   6  2   1  0   4  0  1  0 ]
              [                                           ]
              [ 11  15  14  15  15  2  16  0  13  0  0  1 ]

                        Its REF is given by

              [ 1  0  0   4  0  16  14  1   3   2   3   7 ]
              [                                           ]
              [ 0  1  0  13  0   1   7  3  10  11  13   9 ]
              [                                           ]
              [ 0  0  1   1  0   0   4  4  14  14  15  15 ]
              [                                           ]
              [ 0  0  0   0  1   1   5  5  13  13  15  15 ]

                        AutGD(F) is given by :

    [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]],

       [13, 0, [0, 0]], [13, 1, [0, 0]], [13, 2, [0, 0]], [13, 3, [0, 0]]]

                           Its size is :, 8

                     C(G,D)'s automorphism group is :

                  [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                      [1, 2, 7, 8, 9, 10, 3, 4, 5, 6, 11, 12],

                      [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                      [2, 1, 8, 7, 10, 9, 4, 3, 6, 5, 12, 11],

                      [11, 12, 6, 5, 3, 4, 10, 9, 7, 8, 2, 1],

                      [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                      [12, 11, 5, 6, 4, 3, 9, 10, 8, 7, 1, 2],

                      [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                           Its size is :, 8

                    Time needed(sec) :,         0.8

[...]

                          Example No.:, 85

                             [4 [Qinf]]

                           G =, [4 [Qinf]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

           a basis of L(G) & a generator matrix for C(G,D) are :

                          [  1   1  1   1   1   1   1   1   1   1   1   1 ]
                          [                                               ]
                          [  4   4  5   5   7   7  10  10  12  12  13  13 ]
                       2  [                                               ]
           [1, X, Y, X ], [  3  14  1  16   8   9   2  15   4  13   5  12 ]
                          [                                               ]
                          [ 16  16  8   8  15  15  15  15   8   8  16  16 ]

                       REF of C(G,D)'s matrix is :

              [ 1  0  0  11  0   3   9  14   4  14   3   7 ]
              [                                             ]
              [ 0  1  0   6  0  14  13   8   2   9  13   9 ]
              [                                             ]
              [ 0  0  1   1  0   0   8   8  14  14   7   7 ]
              [                                             ]
              [ 0  0  0   0  1   1   5   5  15  15  12  12 ]

                        AutGD(F) is given by :

     [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                           Its size is :, 4

                     C(G,D)'s automorphism group is :

                 [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                     [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                     [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                     [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                           Its size is :, 4

                    Time needed(sec) :,         0.7

[...]

                          Example No.:, 100

                    [4 [Qinf], 3 [1, 0], [16, 0]]

                  G =, [4 [Qinf], 3 [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

           a basis of L(G) & a generator matrix for C(G,D) are :

                                 2         3
                          2    Y       Y         Y         Y
           [1, X, Y, X , -----, --------, --------, ------],
                       X - 1          2         3  X - 16
                                (X - 1)   (X - 1)

              [ 1   1   1   1   1   1   1   1   1   1   1   1 ]
              [                                               ]
              [ 4   4   5   5   7   7  10  10  12  12  13  13 ]
              [                                               ]
              [ 3  14   1  16   8   9   2  15   4  13   5  12 ]
              [                                               ]
              [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
              [                                               ]
              [ 1  16  13   4   7  10   4  13   5  12  16   1 ]
              [                                               ]
              [ 1   1  16  16  15  15  16  16   8   8   1   1 ]
              [                                               ]
              [ 1  16   4  13   3  14  13   4   6  11  16   1 ]
              [                                               ]
              [ 4  13   3  14   1  16  11   6  16   1   4  13 ]

                       REF of C(G,D)'s matrix is :

                 [ 1  0  0  0  0  0  0  0   8  10   3  12 ]
```

```
[                                                  ]
[ 0  1  0  0  0  0  0  0  0  10   8  12   3 ]
[                                                  ]
[ 0  0  1  0  0  0  0  0  0  15   8   9  10 ]
[                                                  ]
[ 0  0  0  1  0  0  0  0  0   8  15  10   9 ]
[                                                  ]
[ 0  0  0  0  1  0  0  0  0  10   0   8   3 ]
[                                                  ]
[ 0  0  0  0  0  1  0  0  0   0  10   3   8 ]
[                                                  ]
[ 0  0  0  0  0  0  1  0  0  11   7   6   1 ]
[                                                  ]
[ 0  0  0  0  0  0  0  0  1   7  11   1   6 ]
```

We will actually compute the automorphism group of the dual!

The generator matrix of the dual is given by

```
[  9   7  2  9   7   0   6  10  1  0  0  0 ]
[                                          ]
[  7   9  9  2   0   7  10   6  0  1  0  0 ]
[                                          ]
[ 14   5  8  7   9  14  11  16  0  0  1  0 ]
[                                          ]
[  5  14  7  8  14   9  16  11  0  0  0  1 ]
```

Its REF is given by

```
[ 1  0  0  0   1   3  16   3   3   9  12   3 ]
[                                              ]
[ 0  1  0  0   3   1   3  16   9   3   3  12 ]
[                                              ]
[ 0  0  1  0   7  11  14   0  16  13  14   9 ]
[                                              ]
[ 0  0  0  1  11   7   0  14  13  16   9  14 ]
```

AutGD(F) is given by :

```
[[13, 0, [Qinf]], [13, 2, [Qinf]]]
```

Its size is :, 2

C(G,D)'s automorphism group is :

```
[
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11]
]
```

Its size is :, 2

Time needed(sec) :, 0.8

[...]

Example No.:, 121

```
[4 [Qinf], 2 [0, 0], 2 [16, 0]]

G =, [4 [Qinf], 2 [0, 0], 2 [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],
     [12, 13], [13, 5], [13, 12]]
```

a basis of L(G) & a generator matrix for C(G,D) are :

```
                      2              2
              2       Y      Y       Y
[1, X, Y, X , Y/X, ----, ------, ---------],
                    2   X - 16          2
                    X           (X - 16)

[  1   1   1   1   1   1   1   1   1   1   1   1 ]
[                                                ]
[  4   4   5   5   7   7  10  10  12  12  13  13 ]
[                                                ]
[  3  14   1  16   8   9   2  15   4  13   5  12 ]
[                                                ]
[ 16  16   8   8  15  15  15  15   8   8  16  16 ]
[                                                ]
[  5  12   7  10   6  11   7  10   6  11   3  14 ]
[                                                ]
[  8   8  15  15   2   2  15  15   2   2   9   9 ]
[                                                ]
[  4  13   3  14   1  16  11   6  16   1   4  13 ]
[                                                ]
[ 16  16   9   9   1   1   2   2   1   1  16  16 ]
```

REF of C(G,D)'s matrix is :

```
[ 1  0  0  0  0  0  0   3  0   9  12  11 ]
[                                        ]
[ 0  1  0  0  0  0  0  14  0   8   1   2 ]
[                                        ]
[ 0  0  1  0  0  0  0   8  0   1   7  13 ]
[                                        ]
[ 0  0  0  1  0  0  0   9  0  16   3  14 ]
```

```
[                                        ]
[ 0  0  0  0  1  0  0   4  0  13  10   9 ]
[                                        ]
[ 0  0  0  0  0  1  0  13  0   4   8   9 ]
[                                        ]
[ 0  0  0  0  0  0  1   1  0   0  12  12 ]
[                                        ]
[ 0  0  0  0  0  0  0   0  1   1  16  16 ]
```

We will actually compute the automorphism group of the dual!

The generator matrix of the dual is given by

```
[ 14   3   9   8  13   4  16  1   0  0  0  0 ]
[                                            ]
[  8   9  16   1   4  13   0  0  16  1  0  0 ]
[                                            ]
[  5  16  10  14   7   9   5  0   1  0  1  0 ]
[                                            ]
[  6  15   4   3   8   8   5  0   1  0  0  1 ]
```

Its REF is given by

```
[ 1  0  0   6  0   4   2  12  5   8  3  10 ]
[                                          ]
[ 0  1  0   6  0   4  16  15  6   7  4   9 ]
[                                          ]
[ 0  0  1  16  0   0  15   2  9   8  7  10 ]
[                                          ]
[ 0  0  0   0  1  16   2  15  4  13  8   9 ]
```

AutGD(F) is given by :

```
[[13, 0, [Qinf]], [13, 2, [Qinf]]]
```

Its size is :, 2

C(G,D)'s automorphism group is :

```
[
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11]
]
```

Its size is :, 2

Time needed(sec) :, 0.7

[...]

Example No.:, 138

```
[5 [Qinf]]

G =, [5 [Qinf]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],
     [12, 13], [13, 5], [13, 12]]
```

a basis of L(G) & a generator matrix for C(G,D) are :

```
                                                  [  1   1   1   1   1   1   1   1   1   1   1   1 ]
                                                  [                                                ]
                                                  [  4   4   5   5   7   7  10  10  12  12  13  13 ]
                    2                             [                                                ]
[1, X, Y, X , Y X], [  3  14   1  16   8   9   2  15   4  13   5  12 ]
                                                  [                                                ]
                                                  [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
                                                  [                                                ]
                                                  [ 12   5   5  12   5  12   3  14  14   3  14   3 ]
```

REF of C(G,D)'s matrix is :

```
[ 1  0  0  0  0  11  0   9  10   8   5   7 ]
[                                           ]
[ 0  1  0  0  0   6  5  13  13  15  11   9 ]
[                                           ]
[ 0  0  1  0  0  10  1   6  13  15   1   7 ]
[                                           ]
[ 0  0  0  1  0   7  7   2   1  16   6   0 ]
[                                           ]
[ 0  0  0  0  1   1  5   5  15  15  12  12 ]
```

AutGD(F) is given by :

```
[[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]
```

Its size is :, 4

C(G,D)'s automorphism group is :

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],
    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],
    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]
```

```
                        Its size is :, 4

                      Time needed(sec) :,       14.2

[...]

                        Example No.:, 158

                    [5 [Qinf], [0, 0], [1, 0], [16, 0]]

                   G =, [5 [Qinf], [0, 0], [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                    2              Y      Y
           [1, X, Y, X , Y X, Y/X, -----, ------],
                                  X - 1  X - 16

              [  1   1   1   1   1   1   1   1   1   1   1   1 ]
              [                                               ]
              [  4   4   5   5   7   7  10  10  12  12  13  13 ]
              [                                               ]
              [  3  14   1  16   8   9   2  15   4  13   5  12 ]
              [                                               ]
              [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
              [                                               ]
              [ 12   5   5  12   5  12   3  14  14   3  14   3 ]
              [                                               ]
              [  5  12   7  10   6  11   7  10   6  11   3  14 ]
              [                                               ]
              [  1  16  13   4   7  10   4  13   5  12  16   1 ]
              [                                               ]
              [  4  13   3  14   1  16  11   6  16   1   4  13 ]

                       REF of C(G,D)'s matrix is :

                [ 1  0  0  0  0  0  0   5  0   6   5  11 ]
                [                                        ]
                [ 0  1  0  0  0  0  0   5  0   6   1  15 ]
                [                                        ]
                [ 0  0  1  0  0  0  0   8  0  14   6   1 ]
                [                                        ]
                [ 0  0  0  1  0  0  0   8  0  14  12  12 ]
                [                                        ]
                [ 0  0  0  0  1  0  0   5  0  15   1  11 ]
                [                                        ]
                [ 0  0  0  0  0  1  0   5  0  15   6   6 ]
                [                                        ]
                [ 0  0  0  0  0  0  1  16  0   0  14   3 ]
                [                                        ]
                [ 0  0  0  0  0  0  0   0  1  16   7  10 ]

        We will actually compute the automorphism group of the dual!

                The generator matrix of the dual is given by

                [ 12  12   9  9  12  12   1  1   0  0  0  0 ]
                [                                           ]
                [ 11  11   3  3   2   2   0  0   1  1  0  0 ]
                [                                           ]
                [ 12  16  11  5  16  11   3  0  10  0  1  0 ]
                [                                           ]
                [  6   2  16  5   6  11  14  0   7  0  0  1 ]

                          Its REF is given by

                [ 1  0  0  10  0  14   9  14  1  12  14  10 ]
                [                                          ]
                [ 0  1  0   7  0   3  13   8  9  15   4   8 ]
                [                                          ]
                [ 0  0  1   1  0   0   2   2  3   3  11  11 ]
                [                                          ]
                [ 0  0  0   0  1   1  12  12  9   9  12  12 ]

                        AutGD(F) is given by :

     [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                        Its size is :, 4

                    C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                        Its size is :, 4

                      Time needed(sec) :,       0.7

                        Example No.:, 159

                    [5 [Qinf], [0, 0], [1, 0], 2 [16, 0]]

                   G =, [5 [Qinf], [0, 0], [1, 0], 2 [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                                                    2
                    2              Y      Y       Y
           [1, X, Y, X , Y X, Y/X, -----, ------, ---------],
                                  X - 1  X - 16          2
                                                 (X - 16)

              [  1   1   1   1   1   1   1   1   1   1   1   1 ]
              [                                               ]
              [  4   4   5   5   7   7  10  10  12  12  13  13 ]
              [                                               ]
              [  3  14   1  16   8   9   2  15   4  13   5  12 ]
              [                                               ]
              [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
              [                                               ]
              [ 12   5   5  12   5  12   3  14  14   3  14   3 ]
              [                                               ]
              [  5  12   7  10   6  11   7  10   6  11   3  14 ]
              [                                               ]
              [  1  16  13   4   7  10   4  13   5  12  16   1 ]
              [                                               ]
              [  4  13   3  14   1  16  11   6  16   1   4  13 ]
              [                                               ]
              [ 16  16   9   9   1   1   2   2   1   1  16  16 ]

                       REF of C(G,D)'s matrix is :

                [ 1  0  0  0  0  0  0  0  0  11   7  12 ]
                [                                       ]
                [ 0  1  0  0  0  0  0  0  0  11   3  16 ]
                [                                       ]
                [ 0  0  1  0  0  0  0  0  0   5  16   6 ]
                [                                       ]
                [ 0  0  0  1  0  0  0  0  0   5   5   0 ]
                [                                       ]
                [ 0  0  0  0  1  0  0  0  0   3   3  12 ]
                [                                       ]
                [ 0  0  0  0  0  1  0  0  0   3   8   7 ]
                [                                       ]
                [ 0  0  0  0  0  0  1  0  0  16   0  13 ]
                [                                       ]
                [ 0  0  0  0  0  0  0  1  0  16   3  10 ]
                [                                       ]
                [ 0  0  0  0  0  0  0  0  1  16   7  10 ]

        We will actually compute the automorphism group of the dual!

                The generator matrix of the dual is given by

                [  6   6  12  12  14  14  1   1   1  1  0  0 ]
                [                                           ]
                [ 10  14   1  12  14   9  0  14  10  0  1  0 ]
                [                                           ]
                [  5   1  11   0   5  10  4   7   7  0  0  1 ]

                          Its REF is given by

                [ 1  0  0  10  10   7  10  15   6   0  15  11 ]
                [                                            ]
                [ 0  1  0   7   0   3  13   8   9  15   4   8 ]
                [                                            ]
                [ 0  0  1   1  16  16   7   7  11  11  16  16 ]

                        AutGD(F) is given by :

                     [[13, 0, [Qinf]], [13, 2, [Qinf]]]

                        Its size is :, 2

                    C(G,D)'s automorphism group is :

[

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11]

]

                        Its size is :, 2

                      Time needed(sec) :,       0.2

[...]

                        Example No.:, 173

                           [6 [Qinf]]

                          G =, [6 [Qinf]]

D =,
```

```
[[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

    [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                                [  1   1  1   1   1   1   1   1   1   1   1   1 ]
                                [                                               ]
                                [  4   4  5   5   7   7  10  10  12  12  13  13 ]
                                [                                               ]
                 2       3      [  3  14  1  16   8   9   2  15   4  13   5  12 ]
   [1, X, Y, X , Y X, X ], [                                                    ]
                                [ 16  16  8   8  15  15  15  15   8   8  16  16 ]
                                [                                               ]
                                [ 12   5  5  12   5  12   3  14  14   3  14   3 ]
                                [                                               ]
                                [ 13  13  6   6   3   3  14  14  11  11   4   4 ]

                              REF of C(G,D)'s matrix is :

                    [ 1  0  0  0  0  11  0   9  10   8   5   7 ]
                    [                                          ]
                    [ 0  1  0  0  0   6  0   8   5   7   4   2 ]
                    [                                          ]
                    [ 0  0  1  0  0  10  0   5   8  10   3   9 ]
                    [                                          ]
                    [ 0  0  0  1  0   7  0  12   0  15   3  14 ]
                    [                                          ]
                    [ 0  0  0  0  1   1  0   0   7   7   5   5 ]
                    [                                          ]
                    [ 0  0  0  0  0   0  1   1   5   5  15  15 ]

                                 AutGD(F) is given by :

             [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                                    Its size is :, 4

                               C(G,D)'s automorphism group is :

                    [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                        [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                        [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                        [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                                    Its size is :, 4

                                  Time needed(sec) :,      2:08.0

[...]


                                    Example No.:, 187

                                [6 [Qinf], [0, 0], [1, 0], [16, 0]]

                              G =, [6 [Qinf], [0, 0], [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

    [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                      2       3        Y       Y
                [1, X, Y, X , Y X, X , Y/X, -----, ------],
                                          X - 1  X - 16

                    [  1   1   1   1   1   1   1   1   1   1   1   1 ]
                    [                                               ]
                    [  4   4   5   5   7   7  10  10  12  12  13  13 ]
                    [                                               ]
                    [  3  14   1  16   8   9   2  15   4  13   5  12 ]
                    [                                               ]
                    [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
                    [                                               ]
                    [ 12   5   5  12   5  12   3  14  14   3  14   3 ]
                    [                                               ]
                    [ 13  13   6   6   3   3  14  14  11  11   4   4 ]
                    [                                               ]
                    [  5  12   7  10   6  11   7  10   6  11   3  14 ]
                    [                                               ]
                    [  1  16  13   4   7  10   4  13   5  12  16   1 ]
                    [                                               ]
                    [  4  13   3  14   1  16  11   6  16   1   4  13 ]

                              REF of C(G,D)'s matrix is :

                   [ 1  0  0  0  0  0  0  0  0  15   5   4 ]
                   [                                       ]
                   [ 0  1  0  0  0  0  0  0  0  15   1   8 ]
                   [                                       ]
                   [ 0  0  1  0  0  0  0  0  0   8   6   0 ]
                   [                                       ]
                   [ 0  0  0  1  0  0  0  0  0   8  12  11 ]
                   [                                       ]
                   [ 0  0  0  0  1  0  0  0  0   7   1   4 ]
                   [                                       ]
                   [ 0  0  0  0  0  1  0  0  0   7   6  16 ]
                   [                                       ]
                   [ 0  0  0  0  0  0  1  0  0   5  14   1 ]
                   [                                       ]
                   [ 0  0  0  0  0  0  0  1  0   5   0  15 ]
                   [                                       ]
                   [ 0  0  0  0  0  0  0  0  1  16   7  10 ]

        We will actually compute the automorphism group of the dual!

                The generator matrix of the dual is given by

                [  2   2   9  9  10  10  12  12   1  1  0  0 ]
                [                                           ]
                [ 12  16  11  5  16  11   3   0  10  0  1  0 ]
                [                                           ]
                [ 13   9   0  6  13   1  16   2   7  0  0  1 ]

                              Its REF is given by

                [ 1  0  0  10   3   0  11  16  11  5  16  12 ]
                [                                           ]
                [ 0  1  0   7  16   2   1  13   0  6   9  13 ]
                [                                           ]
                [ 0  0  1   1  12  12  10  10   9  9   2   2 ]

                              AutGD(F) is given by :

             [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                                    Its size is :, 4

                               C(G,D)'s automorphism group is :

                    [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                        [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                        [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                        [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                                    Its size is :, 4

                                  Time needed(sec) :,      0.2

[...]

                                    Example No.:, 193

                                   [7 [Qinf]]

                                  G =, [7 [Qinf]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

    [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                                [  1   1  1   1   1   1   1   1   1   1   1   1 ]
                                [                                               ]
                                [  4   4  5   5   7   7  10  10  12  12  13  13 ]
                                [                                               ]
                                [  3  14  1  16   8   9   2  15   4  13   5  12 ]
                 2       3   2  [                                               ]
   [1, X, Y, X , Y X, X , Y X ], [ 16  16  8   8  15  15  15  15   8   8  16  16 ]
                                [                                               ]
                                [ 12   5  5  12   5  12   3  14  14   3  14   3 ]
                                [                                               ]
                                [ 13  13  6   6   3   3  14  14  11  11   4   4 ]
                                [                                               ]
                                [ 14   3  8   9   1  16  13   4  15   2  12   5 ]

                              REF of C(G,D)'s matrix is :

                   [ 1  0  0  0  0  0  0  0   8   6  15   7   3 ]
                   [                                             ]
                   [ 0  1  0  0  0  0  0  0   9   9   0   2   6 ]
                   [                                             ]
                   [ 0  0  1  0  0  0  0  0   1   9   4  11  10 ]
                   [                                             ]
                   [ 0  0  0  1  0  0  0  0  16  16   4  12  13 ]
                   [                                             ]
                   [ 0  0  0  0  1  0  0  0   3   2   3  16   0 ]
                   [                                             ]
                   [ 0  0  0  0  0  0  1  0  14   5   4   6   5 ]
                   [                                             ]
                   [ 0  0  0  0  0  0  0  1   1   5   5  15  15 ]

        We will actually compute the automorphism group of the dual!

                The generator matrix of the dual is given by

                [  9   8  16   1  14   3  16  1  0  0  0  0 ]
                [                                           ]
                [ 11   8   8   1  15  12  12  0  1  0  0  0 ]
                [                                           ]
```

```
[  2   0  13  13  14  13  12  0  0  1  0  0 ]
[                                           ]
[ 10  15   6   5   1  11   2  0  0  0  1  0 ]
[                                           ]
[ 14  11   7   4   0  12   2  0  0  0  0  1 ]
```

Its REF is given by

```
[ 1  0  0  0  0   2   0  12   4   7  14  11 ]
[                                           ]
[ 0  1  0  0  0   2   1  11   5   6  10  15 ]
[                                           ]
[ 0  0  1  0  0  12  15  12   1   8  11   8 ]
[                                           ]
[ 0  0  0  1  0  12  14  13  13  13   2   0 ]
[                                           ]
[ 0  0  0  0  1  16  14   3   1  16   9   8 ]
```

AutGD(F) is given by :

```
[[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]
```

Its size is :, 4

C(G,D)'s automorphism group is :

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],
 [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],
 [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],
 [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]
```

Its size is :, 4

Time needed(sec) :, 14.4

[...]

Example No.:, 200

[7 [Qinf], [0, 0], [16, 0]]

G =, [7 [Qinf], [0, 0], [16, 0]]

D =,

```
 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],
     [12, 13], [13, 5], [13, 12]]
```

a basis of L(G) & a generator matrix for C(G,D) are :

```
          2       3     2          Y
[1, X, Y, X , Y X, X , Y X , Y/X, -----],
                                  X - 16
```

```
[  1   1  1   1   1   1   1   1   1   1   1   1 ]
[                                               ]
[  4   4  5   5   7   7  10  10  12  12  13  13 ]
[                                               ]
[  3  14  1  16   8   9   2  15   4  13   5  12 ]
[                                               ]
[ 16  16  8   8  15  15  15  15   8   8  16  16 ]
[                                               ]
[ 12   5  5  12   5  12   3  14  14   3  14   3 ]
[                                               ]
[ 13  13  6   6   3   3  14  14  11  11   4   4 ]
[                                               ]
[ 14   3  8   9   1  16  13   4  15   2  12   5 ]
[                                               ]
[  5  12  7  10   6  11   7  10   6  11   3  14 ]
[                                               ]
[  4  13  3  14   1  16  11   6  16   1   4  13 ]
```

REF of C(G,D)'s matrix is :

```
[ 1  0  0  0  0  0  0  0  0  15   7   2 ]
[                                       ]
[ 0  1  0  0  0  0  0  0  0  15   8   1 ]
[                                       ]
[ 0  0  1  0  0  0  0  0  0   8  16   7 ]
[                                       ]
[ 0  0  0  1  0  0  0  0  0   8   0   6 ]
[                                       ]
[ 0  0  0  0  1  0  0  0  0   7   4   1 ]
[                                       ]
[ 0  0  0  0  0  1  0  0  0   7  14   8 ]
[                                       ]
[ 0  0  0  0  0  0  1  0  0   5  15   0 ]
[                                       ]
[ 0  0  0  0  0  0  0  1  0   5   2  13 ]
[                                       ]
[ 0  0  0  0  0  0  0  0  1  16   3  14 ]
```

We will actually compute the automorphism group of the dual!

The generator matrix of the dual is given by

```
[  2   2   9   9  10  10  12  12   1  1  0  0 ]
[                                             ]
[ 10   9   1   0  13   3   2  15  14  0  1  0 ]
[                                             ]
[ 15  16  10  11  16   9   0   4   3  0  0  1 ]
```

Its REF is given by

```
[ 1  0  0  16   0   7   3  16  8  11  12  11 ]
[                                           ]
[ 0  1  0   1   2  12   9  13  3   0  13  14 ]
[                                           ]
[ 0  0  1   1  12  12  10  10  9   9   2   2 ]
```

AutGD(F) is given by :

```
[[13, 0, [Qinf]], [13, 2, [Qinf]]]
```

Its size is :, 2

C(G,D)'s automorphism group is :

```
[
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11]
]
```

Its size is :, 2

Time needed(sec) :, 0.2

[...]

Example No.:, 203

[8 [Qinf]]

G =, [8 [Qinf]]

D =,

```
 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],
     [12, 13], [13, 5], [13, 12]]
```

a basis of L(G) & a generator matrix for C(G,D) are :

```
          2       3     2   4
[1, X, Y, X , Y X, X , Y X , X ],
```

```
[  1   1   1   1   1   1   1   1   1   1   1   1 ]
[                                               ]
[  4   4   5   5   7   7  10  10  12  12  13  13 ]
[                                               ]
[  3  14   1  16   8   9   2  15   4  13   5  12 ]
[                                               ]
[ 16  16   8   8  15  15  15  15   8   8  16  16 ]
[                                               ]
[ 12   5   5  12   5  12   3  14  14   3  14   3 ]
[                                               ]
[ 13  13   6   6   3   3  14  14  11  11   4   4 ]
[                                               ]
[ 14   3   8   9   1  16  13   4  15   2  12   5 ]
[                                               ]
[  1   1  13  13   4   4   4   4  13  13   1   1 ]
```

REF of C(G,D)'s matrix is :

```
[ 1  0  0  0  0  0  0  0   8  0   9  14  10 ]
[                                           ]
[ 0  1  0  0  0  0  0  0   9  0   8   4   8 ]
[                                           ]
[ 0  0  1  0  0  0  0  0   1  0  12  13  12 ]
[                                           ]
[ 0  0  0  1  0  0  0  0  16  0   5   8   9 ]
[                                           ]
[ 0  0  0  0  1  0  0  0   3  0   1   7   8 ]
[                                           ]
[ 0  0  0  0  0  0  1  0  14  0  16   9   8 ]
[                                           ]
[ 0  0  0  0  0  0  0  1   1  0   0   1   1 ]
[                                           ]
[ 0  0  0  0  0  0  0  0   0  1   1  13  13 ]
```

We will actually compute the automorphism group of the dual!

The generator matrix of the dual is given by

```
[ 9   8  16   1  14  3  16  1   0  0  0  0 ]
[                                           ]
[ 8   9   5  12  16  1   0  0  16  1  0  0 ]
[                                           ]
[ 3  13   4   9  10  8  16  0   4  0  1  0 ]
[                                           ]
[ 7   9   5   8   9  9  16  0   4  0  0  1 ]
```

Its REF is given by

```
[ 1  0  0   4  0  16   5  13  5   8  5  11 ]
[                                           ]
[ 0  1  0   4  0  16   6  12  6   7  1  15 ]
[                                           ]
[ 0  0  1  16  0   0   1  16  5  12  9   8 ]
```

```
                [                                    ]
                [ 0  0  0   0  1  16  14   3  1  16  9   8 ]

                          AutGD(F) is given by :

      [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                               Its size is :, 4

                          C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                               Its size is :, 4

                            Time needed(sec) :,         1.1

[...]

                               Example No.:, 205

                              [8 [Qinf], [1, 0]]

                             G =, [8 [Qinf], [1, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                      2        3     2   4    Y
                [1, X, Y, X , Y X, X , Y X , X , -----],
                                            X - 1

                    [  1   1   1   1   1   1   1   1   1   1   1   1 ]
                    [                                               ]
                    [  4   4   5   5   7   7  10  10  12  12  13  13 ]
                    [                                               ]
                    [  3  14   1  16   8   9   2  15   4  13   5  12 ]
                    [                                               ]
                    [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
                    [                                               ]
                    [ 12   5   5  12   5  12   3  14  14   3  14   3 ]
                    [                                               ]
                    [ 13  13   6   6   3   3  14  14  11  11   4   4 ]
                    [                                               ]
                    [ 14   3   8   9   1  16  13   4  15   2  12   5 ]
                    [                                               ]
                    [  1   1  13  13   4   4   4   4  13  13   1   1 ]
                    [                                               ]
                    [  1  16  13   4   7  10   4  13   5  12  16   1 ]

                             REF of C(G,D)'s matrix is :

                       [ 1  0  0  0  0  0  0  0  0  10  10   2 ]
                       [                                       ]
                       [ 0  1  0  0  0  0  0  0  0   7   8  16 ]
                       [                                       ]
                       [ 0  0  1  0  0  0  0  0  0  10   4  11 ]
                       [                                       ]
                       [ 0  0  0  1  0  0  0  0  0   7   0  10 ]
                       [                                       ]
                       [ 0  0  0  0  1  0  0  0  0  12  14   5 ]
                       [                                       ]
                       [ 0  0  0  0  0  1  0  0  0   5   2  11 ]
                       [                                       ]
                       [ 0  0  0  0  0  0  1  0  0  15   9   0 ]
                       [                                       ]
                       [ 0  0  0  0  0  0  0  1  0   2   9   1 ]
                       [                                       ]
                       [ 0  0  0  0  0  0  0  0  1   1  13  13 ]

                We will actually compute the automorphism group of the dual!

                        The generator matrix of the dual is given by

                    [  7  10   7  10   5  12  2  15  16  1  0  0 ]
                    [                                           ]
                    [  7   9  13   0   3  15  8   8   4  0  1  0 ]
                    [                                           ]
                    [ 15   1   6   7  12   6  0  16   4  0  0  1 ]

                                Its REF is given by

                    [ 1  0  0   4   0  16  5  13   5   8   5  11 ]
                    [                                           ]
                    [ 0  1  0   4   7   9  2  16  13   0  13   3 ]
                    [                                           ]
                    [ 0  0  1  16  15   2  7  10   3  14   8   9 ]

                             AutGD(F) is given by :

                          [[13, 0, [Qinf]], [13, 2, [Qinf]]]

                                Its size is :, 2

                          C(G,D)'s automorphism group is :

[

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12], [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11]

]

                                Its size is :, 2

                             Time needed(sec) :,         0.2

                                Example No.:, 206

                               [8 [Qinf], [0, 0]]

                              G =, [8 [Qinf], [0, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

            a basis of L(G) & a generator matrix for C(G,D) are :

                      2        3     2   4
                [1, X, Y, X , Y X, X , Y X , X , Y/X],

                    [  1   1   1   1   1   1   1   1   1   1   1   1 ]
                    [                                               ]
                    [  4   4   5   5   7   7  10  10  12  12  13  13 ]
                    [                                               ]
                    [  3  14   1  16   8   9   2  15   4  13   5  12 ]
                    [                                               ]
                    [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
                    [                                               ]
                    [ 12   5   5  12   5  12   3  14  14   3  14   3 ]
                    [                                               ]
                    [ 13  13   6   6   3   3  14  14  11  11   4   4 ]
                    [                                               ]
                    [ 14   3   8   9   1  16  13   4  15   2  12   5 ]
                    [                                               ]
                    [  1   1  13  13   4   4   4   4  13  13   1   1 ]
                    [                                               ]
                    [  5  12   7  10   6  11   7  10   6  11   3  14 ]

                             REF of C(G,D)'s matrix is :

                       [ 1  0  0  0  0  0  0  0  0  16  12  10 ]
                       [                                       ]
                       [ 0  1  0  0  0  0  0  0  0   1   6   8 ]
                       [                                       ]
                       [ 0  0  1  0  0  0  0  0  0  15   0  12 ]
                       [                                       ]
                       [ 0  0  0  1  0  0  0  0  0   2   4   9 ]
                       [                                       ]
                       [ 0  0  0  0  1  0  0  0  0  10   2   8 ]
                       [                                       ]
                       [ 0  0  0  0  0  1  0  0  0   7  14   8 ]
                       [                                       ]
                       [ 0  0  0  0  0  0  1  0  0   3   5   1 ]
                       [                                       ]
                       [ 0  0  0  0  0  0  0  1  0  14  13   0 ]
                       [                                       ]
                       [ 0  0  0  0  0  0  0  0  1   1  13  13 ]

                We will actually compute the automorphism group of the dual!

                        The generator matrix of the dual is given by

                    [ 1  16  2  15   7  10  14  3  16  1  0  0 ]
                    [                                         ]
                    [ 5  11  0  13  15   3  12  4   4  0  1  0 ]
                    [                                         ]
                    [ 7   9  5   8   9   9  16  0   4  0  0  1 ]

                                Its REF is given by

                    [ 1  0  0   4  12  4   3  15  0  13  11  5 ]
                    [                                         ]
                    [ 0  1  0   4  16  0   9   9  5   8   9  7 ]
                    [                                         ]
                    [ 0  0  1  16  14  3  10   7  2  15  16  1 ]

                             AutGD(F) is given by :

                [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                                Its size is :, 4

                          C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],
```

```
                    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                             Its size is :, 4

                          Time needed(sec) :,         0.2

                             Example No.:, 207

                               [9 [Qinf]]

                               G =, [9 [Qinf]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

             a basis of L(G) & a generator matrix for C(G,D) are :

                      2        3     2   4     3
             [1, X, Y, X , Y X, X , Y X , X , Y X ],

                [  1   1   1   1   1   1   1   1   1   1   1   1 ]
                [                                               ]
                [  4   4   5   5   7   7  10  10  12  12  13  13 ]
                [                                               ]
                [  3  14   1  16   8   9   2  15   4  13   5  12 ]
                [                                               ]
                [ 16  16   8   8  15  15  15  15   8   8  16  16 ]
                [                                               ]
                [ 12   5   5  12   5  12   3  14  14   3  14   3 ]
                [                                               ]
                [ 13  13   6   6   3   3  14  14  11  11   4   4 ]
                [                                               ]
                [ 14   3   8   9   1  16  13   4  15   2  12   5 ]
                [                                               ]
                [  1   1  13  13   4   4   4   4  13  13   1   1 ]
                [                                               ]
                [  5  12   6  11   7  10  11   6  10   7   3  14 ]

                       REF of C(G,D)'s matrix is :

                   [ 1  0  0  0  0  0  0  0  0  14  14  16 ]
                   [                                       ]
                   [ 0  1  0  0  0  0  0  0  0   3   4   2 ]
                   [                                       ]
                   [ 0  0  1  0  0  0  0  0  0   2  13   0 ]
                   [                                       ]
                   [ 0  0  0  1  0  0  0  0  0  15   8   4 ]
                   [                                       ]
                   [ 0  0  0  0  1  0  0  0  0   5   7   6 ]
                   [                                       ]
                   [ 0  0  0  0  0  1  0  0  0  12   9  10 ]
                   [                                       ]
                   [ 0  0  0  0  0  0  1  0  0   7   1   6 ]
                   [                                       ]
                   [ 0  0  0  0  0  0  0  1  0  10   0  12 ]
                   [                                       ]
                   [ 0  0  0  0  0  0  0  0  1   1  13  13 ]

          We will actually compute the automorphism group of the dual!

                  The generator matrix of the dual is given by

                  [ 3  14  15   2  12  5  10  7  16  1  0  0 ]
                  [                                         ]
                  [ 3  13   4   9  10  8  16  0   4  0  1  0 ]
                  [                                         ]
                  [ 1  15   0  13  11  7  11  5   4  0  0  1 ]

                              Its REF is given by

                  [ 1  0  0   4  16  0  8  10   4   9  13  3 ]
                  [                                         ]
                  [ 0  1  0   4  11  5  7  11   0  13  15  1 ]
                  [                                         ]
                  [ 0  0  1  16  10  7  5  12  15   2  14  3 ]

                              AutGD(F) is given by :

      [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                             Its size is :, 4

                        C(G,D)'s automorphism group is :

                  [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                      [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                      [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                      [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                             Its size is :, 4

                          Time needed(sec) :,         0.2

------------------------------------------------------------------------------
```

C.2 Example 9.5.4 and Example 9.5.5

```
> F:=X^3-X;
                                    3
                             F := X  - X

------------------------------------------------------------------------------
> Q:=FindPlaces(F,17);
  Q := [[Qinf], [0, 0], [1, 0], [16, 0], [4, 3], [4, 14], [5, 1], [5, 16],

      [7, 8], [7, 9], [10, 2], [10, 15], [12, 4], [12, 13], [13, 5], [13, 12]]

------------------------------------------------------------------------------
> D:=CreateD(Q,[seq(i,i=5..16)]);
   D := [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15],

       [12, 4], [12, 13], [13, 5], [13, 12]]

------------------------------------------------------------------------------
> G:=CreateG(Q,[[3,1],[3,2],[3,3],[1,4]]);
                 G := [3 [Qinf], 3 [0, 0], 3 [1, 0], [16, 0]]

------------------------------------------------------------------------------
> AnalyseCode(F,17,G,D,1);
                 G =, [3 [Qinf], 3 [0, 0], 3 [1, 0], [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]

             a basis of L(G) & a generator matrix for C(G,D) are :

                       2     3             2         3
                      Y     Y      Y      Y         Y         Y
        [1, X, Y, Y/X, ----, ----, -----, --------, --------, ------],
                       2     3   X - 1         2         3  X - 16
                      X     X           (X - 1)   (X - 1)

            [ 1   1   1   1   1   1   1   1   1   1   1   1 ]
            [                                               ]
            [ 4   4   5   5   7   7  10  10  12  12  13  13 ]
            [                                               ]
            [ 3  14   1  16   8   9   2  15   4  13   5  12 ]
            [                                               ]
            [ 5  12   7  10   6  11   7  10   6  11   3  14 ]
            [                                               ]
            [ 8   8  15  15   2   2  15  15   2   2   9   9 ]
            [                                               ]
            [ 6  11   3  14  12   5   3  14  12   5  10   7 ]
            [                                               ]
            [ 1  16  13   4   7  10   4  13   5  12  16   1 ]
            [                                               ]
            [ 1   1  16  16  15  15  16  16   8   8   1   1 ]
            [                                               ]
            [ 1  16   4  13   3  14  13   4   6  11  16   1 ]
            [                                               ]
            [ 4  13   3  14   1  16  11   6  16   1   4  13 ]

                       REF of C(G,D)'s matrix is :

                  [ 1  0  0  0  0  0  0  0  0   6  0   2 ]
                  [                                       ]
                  [ 0  1  0  0  0  0  0  0  0   6  0   2 ]
```

```
[                                  ]
[ 0  0  1  0  0  0  0  0  0  11  0   6 ]
[                                  ]
[ 0  0  0  1  0  0  0  0  0  11  0   6 ]
[                                  ]
[ 0  0  0  0  1  0  0  0  0   3  0   2 ]
[                                  ]
[ 0  0  0  0  0  1  0  0  0   3  0   2 ]
[                                  ]
[ 0  0  0  0  0  0  1  0  0  15  0   8 ]
[                                  ]
[ 0  0  0  0  0  0  0  1  0  15  0   8 ]
[                                  ]
[ 0  0  0  0  0  0  0  0  1  16  0   0 ]
[                                  ]
[ 0  0  0  0  0  0  0  0  0   0  1  16 ]
```

We will actually compute the automorphism group of the dual!

The generator matrix of the dual is given by

```
[ 11  11   6   6  14  14  2  2  1  1  0  0 ]
[                                          ]
[ 15  15  11  11  15  15  9  9  0  0  1  1 ]
```

Its REF is given by

```
[ 1  1  0  0   7   7   5   5  2  2  2  2 ]
[                                        ]
[ 0  0  1  1  15  15  11  11  5  5  2  2 ]
```

AutGD(F) is given by :

```
[[13, 0, [Qinf]], [13, 2, [Qinf]], [13, 1, [0, 0]], [13, 3, [0, 0]]]
```

Its size is :, 4

C(G,D)'s automorphism group is :

```
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

   [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 11],

   [1, 2, 3, 4, 5, 6, 7, 8, 10, 9, 11, 12],

   [1, 2, 3, 4, 5, 6, 7, 8, 10, 9, 12, 11],

   [1, 2, 3, 4, 5, 6, 8, 7, 9, 10, 11, 12],

[...]

   [12, 11, 6, 5, 4, 3, 9, 10, 7, 8, 2, 1],

   [12, 11, 6, 5, 4, 3, 9, 10, 8, 7, 1, 2],

   [12, 11, 6, 5, 4, 3, 9, 10, 8, 7, 2, 1],

   [12, 11, 6, 5, 4, 3, 10, 9, 7, 8, 1, 2],

   [12, 11, 6, 5, 4, 3, 10, 9, 7, 8, 2, 1],

   [12, 11, 6, 5, 4, 3, 10, 9, 8, 7, 1, 2],

   [12, 11, 6, 5, 4, 3, 10, 9, 8, 7, 2, 1]]
```

Its size is :, 128

Time needed(sec) :, 0.8

```
> G:=CreateG(Q,[[4,1],[2,2],[2,3],[2,4]]);
```

```
G := [4 [Qinf], 2 [0, 0], 2 [1, 0], 2 [16, 0]]
```

```
> AnalyseCode(F,17,G,D,1);
```

```
G =, [4 [Qinf], 2 [0, 0], 2 [1, 0], 2 [16, 0]]

D =,

 [[4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9], [10, 2], [10, 15], [12, 4],

     [12, 13], [13, 5], [13, 12]]
```

a basis of L(G) & a generator matrix for C(G,D) are :

$$\left[1,\ X,\ Y,\ X^2,\ Y/X,\ \frac{Y^2}{X^2},\ \frac{Y^2}{X-1},\ \frac{Y^2}{X^2-1},\ \frac{Y^2}{X-16},\ \frac{Y^2}{(X-1)^2},\ \frac{Y^2}{(X-16)^2}\right]$$

```
[  1   1   1   1   1   1   1   1   1   1   1   1 ]
[                                                ]
[  4   4   5   5   7   7  10  10  12  12  13  13 ]
[                                                ]
[  3  14   1  16   8   9   2  15   4  13   5  12 ]
[                                                ]
[ 16  16   8   8  15  15  15  15   8   8  16  16 ]
[                                                ]
[  5  12   7  10   6  11   7  10   6  11   3  14 ]
[                                                ]
[  8   8  15  15   2   2  15  15   2   2   9   9 ]
[                                                ]
[  1  16  13   4   7  10   4  13   5  12  16   1 ]
[                                                ]
[  1   1  16  16  15  15  16  16   8   8   1   1 ]
[                                                ]
[  4  13   3  14   1  16  11   6  16   1   4  13 ]
[                                                ]
[ 16  16   9   9   1   1   2   2   1   1  16  16 ]
```

REF of C(G,D)'s matrix is :

```
[ 1  0  0  0  0  0  0  0  0  10  0  15 ]
[                                      ]
[ 0  1  0  0  0  0  0  0  0   7  0   2 ]
[                                      ]
[ 0  0  1  0  0  0  0  0  0  15  0   9 ]
[                                      ]
[ 0  0  0  1  0  0  0  0  0   2  0   8 ]
[                                      ]
[ 0  0  0  0  1  0  0  0  0   3  0   9 ]
[                                      ]
[ 0  0  0  0  0  1  0  0  0  14  0   8 ]
[                                      ]
[ 0  0  0  0  0  0  1  0  0   6  0  11 ]
[                                      ]
[ 0  0  0  0  0  0  0  1  0  11  0   6 ]
[                                      ]
[ 0  0  0  0  0  0  0  0  1   1  0   0 ]
[                                      ]
[ 0  0  0  0  0  0  0  0  0   0  1   1 ]
```

We will actually compute the automorphism group of the dual!

```
                The generator matrix of the dual is given by

                [ 7  10  2  15  14  3  11   6  16  1   0  0 ]
                [                                           ]
                [ 2  15  8   9   8  9   6  11   0  0  16  1 ]

                            Its REF is given by

                [ 1  16  0   0  11  6  8   9  9   8   2  15 ]
                [                                           ]
                [ 0   0  1  16  11  6  3  14  2  15  10   7 ]

                          AutGD(F) is given by :

      [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                                 Its size is :, 4

                           C(G,D)'s automorphism group is :

                [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12],

                    [2, 1, 4, 3, 6, 5, 8, 7, 10, 9, 12, 11],

                    [11, 12, 10, 9, 7, 8, 6, 5, 3, 4, 2, 1],

                    [12, 11, 9, 10, 8, 7, 5, 6, 4, 3, 1, 2]]

                                 Its size is :, 4

                             Time needed(sec) :,         0.6
```

C.3 Lemma 9.5.7 (d)

```
> F:=X^3-X;
                                      3
                                F := X  - X
```

```
> Q:=FindPlaces(F,17);
  Q := [[Qinf], [0, 0], [1, 0], [16, 0], [4, 3], [4, 14], [5, 1], [5, 16],

      [7, 8], [7, 9], [10, 2], [10, 15], [12, 4], [12, 13], [13, 5], [13, 12]]
```

```
> D:=CreateD(Q,[seq(i,i=2..16)]);
    D := [[0, 0], [1, 0], [16, 0], [4, 3], [4, 14], [5, 1], [5, 16], [7, 8],

        [7, 9], [10, 2], [10, 15], [12, 4], [12, 13], [13, 5], [13, 12]]
```

```
> G:=CreateG(Q,[[11,1]]);
                                G := [11 [Qinf]]
```

```
> AnalyseCode(F,17,G,D,1);
                                G =, [11 [Qinf]]

D =,

    [[0, 0], [1, 0], [16, 0], [4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9],

        [10, 2], [10, 15], [12, 4], [12, 13], [13, 5], [13, 12]]

              a basis of L(G) & a generator matrix for C(G,D) are :

                    2        3     2   4     3   5     4
        [1, X, Y, X , Y X, X , Y X , X , Y X , X , Y X ],

            [ 1  1   1   1   1   1   1   1   1   1   1   1   1   1 ]
            [                                                     ]
            [ 0  1  16   4   4   5   5   7   7  10  10  12  12  13  13 ]
            [                                                     ]
            [ 0  0   0   3  14   1  16   8   9   2  15   4  13   5  12 ]
            [                                                     ]
            [ 0  1   1  16  16   8   8  15  15  15  15   8   8  16  16 ]
            [                                                     ]
            [ 0  0   0  12   5   5  12   5  12   3  14  14   3  14   3 ]
            [                                                     ]
            [ 0  1  16  13  13   6   6   3   3  14  14  11  11   4   4 ]
            [                                                     ]
            [ 0  0   0  14   3   8   9   1  16  13   4  15   2  12   5 ]
            [                                                     ]
            [ 0  1   1   1   1  13  13   4   4   4   4  13  13   1   1 ]
            [                                                     ]
            [ 0  0   0   5  12   6  11   7  10  11   6  10   7   3  14 ]
            [                                                     ]
            [ 0  1  16   4   4  14  14  11  11   6   6   3   3  13  13 ]
            [                                                     ]
            [ 0  0   0   3  14  13   4  15   2   8   9   1  16   5  12 ]

                          REF of C(G,D)'s matrix is :

               [ 1  0  0  0  0  0  0  0  0  0   1  0  11  10   0 ]
               [                                                  ]
               [ 0  1  0  0  0  0  0  0  0  0   7  0  11   4   9 ]
               [                                                  ]
               [ 0  0  1  0  0  0  0  0  0  0  13  0  16  10   5 ]
               [                                                  ]
               [ 0  0  0  1  0  0  0  0  0  0   6  0   5  12   6 ]
               [                                                  ]
               [ 0  0  0  0  1  0  0  0  0  0   6  0   5  16   2 ]
               [                                                  ]
               [ 0  0  0  0  0  1  0  0  0  0  15  0   3  14   8 ]
               [                                                  ]
               [ 0  0  0  0  0  0  1  0  0  0  15  0   3  11  11 ]
               [                                                  ]
               [ 0  0  0  0  0  0  0  1  0  0  12  0   8  15  12 ]
               [                                                  ]
               [ 0  0  0  0  0  0  0  0  1  0  12  0   8   5   5 ]
               [                                                  ]
               [ 0  0  0  0  0  0  0  0  0  1  16  0   0  11   6 ]
               [                                                  ]
               [ 0  0  0  0  0  0  0  0  0  0   0  1  16  12   5 ]

            We will actually compute the automorphism group of the dual!

                    The generator matrix of the dual is given by

               [ 16  10   4  11  11   2   2  5   5   1  1   0  0  0  0 ]
               [                                                       ]
               [  6   6   1  12  12  14  14  9   9   0  0   1  1  0  0 ]
               [                                                       ]
               [  7  13   7   5   1   3   6  2  12   6  0   5  0  1  0 ]
               [                                                       ]
               [  0   8  12  11  15   9   6  5  12  11  0  12  0  0  1 ]

                                Its REF is given by

               [ 1  0  0  0   1  16  11   4  10   2  12  15  12  11   7 ]
               [                                                       ]
               [ 0  1  0  0  12  14   5  12  16  10  11  13  11  14   0 ]
               [                                                       ]
               [ 0  0  1  0  14  13  11  11  10  12  16   5  14   0  12 ]
               [                                                       ]
               [ 0  0  0  1  16  12   5   6  11  10   7  14   3  13   4 ]

                                AutGD(F) is given by :

              [[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

                                       Its size is :, 4

                                 C(G,D)'s automorphism group is :

                      [[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],

                          [1, 2, 3, 5, 4, 7, 6, 9, 8, 11, 10, 13, 12, 15, 14],

                          [1, 3, 2, 14, 15, 13, 12, 10, 11, 9, 8, 6, 7, 5, 4],

                          [1, 3, 2, 15, 14, 12, 13, 11, 10, 8, 9, 7, 6, 4, 5]]

                                       Its size is :, 4

                                   Time needed(sec) :,         3.7
```

```
> G:=CreateG(Q,[[12,1]]);
                                G := [12 [Qinf]]
```

```
> AnalyseCode(F,17,G,D,1);
                                G =, [12 [Qinf]]

D =,

    [[0, 0], [1, 0], [16, 0], [4, 3], [4, 14], [5, 1], [5, 16], [7, 8], [7, 9],

        [10, 2], [10, 15], [12, 4], [12, 13], [13, 5], [13, 12]]

              a basis of L(G) & a generator matrix for C(G,D) are :

                    2        3     2   4     3   5     4   6
        [1, X, Y, X , Y X, X , Y X , X , Y X , X , Y X , X ],

            [ 1  1   1   1   1   1   1   1   1   1   1   1   1   1 ]
            [                                                     ]
```

```
[ 0  1  16   4   4   5   5   7   7  10  10  12  12  13  13 ]
[                                                          ]
[ 0  0   0   3  14   1  16   8   9   2  15   4  13   5  12 ]
[                                                          ]
[ 0  1   1  16  16   8   8  15  15  15  15   8   8  16  16 ]
[                                                          ]
[ 0  0   0  12   5   5  12   5  12   3  14  14   3  14   3 ]
[                                                          ]
[ 0  1  16  13  13   6   6   3   3  14  14  11  11   4   4 ]
[                                                          ]
[ 0  0   0  14   3   8   9   1  16  13   4  15   2  12   5 ]
[                                                          ]
[ 0  1   1   1   1  13  13   4   4   4   4  13  13   1   1 ]
[                                                          ]
[ 0  0   0   5  12   6  11   7  10  11   6  10   7   3  14 ]
[                                                          ]
[ 0  1  16   4   4  14  14  11  11   6   6   3   3  13  13 ]
[                                                          ]
[ 0  0   0   3  14  13   4  15   2   8   9   1  16   5  12 ]
[                                                          ]
[ 0  1   1  16  16   2   2   9   9   9   9   2   2  16  16 ]
```

REF of C(G,D)'s matrix is :

```
[ 1  0  0  0  0  0  0  0  0  0  0  0   8  10   4 ]
[                                                 ]
[ 0  1  0  0  0  0  0  0  0  0  0  0   7   4   3 ]
[                                                 ]
[ 0  0  1  0  0  0  0  0  0  0  0  0  11  10   6 ]
[                                                 ]
[ 0  0  0  1  0  0  0  0  0  0  0  0   4  12  13 ]
[                                                 ]
[ 0  0  0  0  1  0  0  0  0  0  0  0   4  16   9 ]
[                                                 ]
[ 0  0  0  0  0  1  0  0  0  0  0  0   9  14   0 ]
[                                                 ]
[ 0  0  0  0  0  0  1  0  0  0  0  0   9  11   3 ]
[                                                 ]
[ 0  0  0  0  0  0  0  1  0  0  0  0   6  15   9 ]
[                                                 ]
[ 0  0  0  0  0  0  0  0  1  0  0  0   6   5   2 ]
[                                                 ]
[ 0  0  0  0  0  0  0  0  0  1  0  0   3  11   2 ]
[                                                 ]
[ 0  0  0  0  0  0  0  0  0  0  1  0   3   0  13 ]
[                                                 ]
[ 0  0  0  0  0  0  0  0  0  0  0  1  16  12   5 ]
```

We will actually compute the automorphism group of the dual!

The generator matrix of the dual is given by

```
[  9  10   6  13  13   8   8  11  11  14  14   1  1  0  0 ]
[                                                        ]
[  7  13   7   5   1   3   6   2  12   6   0   5  0  1  0 ]
[                                                        ]
[ 13  14  11   4   8   0  14   8  15  15   4  12  0  0  1 ]
```

Its REF is given by

```
[ 1  0  15  0   7   7   6  16   7  12  14   5   1  11  0 ]
[                                                       ]
[ 0  1  16  0  15   1  11   1   6  15  12   8  14  14  5 ]
[                                                       ]
[ 0  0   0  1  16  12   5   6  11  10   7  14   3  13  4 ]
```

AutGD(F) is given by :

[[13, 0, [Qinf]], [13, 1, [Qinf]], [13, 2, [Qinf]], [13, 3, [Qinf]]]

Its size is :, 4

C(G,D)'s automorphism group is :

[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],

[1, 2, 3, 5, 4, 7, 6, 9, 8, 11, 10, 13, 12, 15, 14],

[1, 3, 2, 14, 15, 13, 12, 10, 11, 9, 8, 6, 7, 5, 4],

[1, 3, 2, 15, 14, 12, 13, 11, 10, 8, 9, 7, 6, 4, 5]]

Its size is :, 4

Time needed(sec) :, 0.4

C.4 Lemma 9.5.16

```
> Q:=Klein(29);
Q := [[Qinf], [0, 0], [1, 0], [3, 28], [3, 5], [3, 9], [3, 4], [3, 6], [3, 22],

    [3, 13], [14, 10], [14, 15], [14, 18], [14, 12], [14, 26], [14, 8],

    [14, 27], [20, 19], [20, 21], [20, 3], [20, 2], [20, 11], [20, 14], [20, 17]

    ]
```

```
> D:=CreateD(Q,[seq(i,i=4..24)]);
   D := [[3, 28], [3, 5], [3, 9], [3, 4], [3, 6], [3, 22], [3, 13], [14, 10],

       [14, 15], [14, 18], [14, 12], [14, 26], [14, 8], [14, 27], [20, 19],

       [20, 21], [20, 3], [20, 2], [20, 11], [20, 14], [20, 17]]
```

```
> G:=CreateG(Q,[[2,1],[2,2],[2,3]]);
                        G := [2 [Qinf], 2 [0, 0], 2 [1, 0]]
```

```
> w1:=X/Y^2;w2:=X/Y;v1:=Y^3/X;v2:=-Y^5/X^2;z1:=-1/Y;z2:=X/Y^4;
                                      X
                                w1 := ----
                                       2
                                      Y

                                       X
                                w2 := ---
                                      Y

                                       3
                                      Y
                                v1 := ----
                                       X

                                        5
                                       Y
                                v2 := - ----
                                         2
                                        X

                                z1 := - 1/Y

                                       X
                                z2 := ----
                                       4
                                      Y
```

```
> LofG:=[1,v1*z1,w1*z1,w1*v1];
                                    2
                                   Y        X
                      LofG := [1, - ----, - ----, Y]
                                   X       3
                                          Y
```

```
> C1:=CreateMatrix2(LofG,D,29);
                                  2
                                 Y        X
                    C1 := [1, - ----, - ----, Y], C1Max
                                 X       3
                                        Y
```

```
> Gaussjord(C1[2]) mod 29;
   [1, 0, 0, 0, 22, 21, 28, 4, 14, 10, 0, 12, 25, 7, 24, 8, 9, 7, 22, 14, 17]

    [0, 1, 0, 0, 1, 1, 22, 6, 23, 8, 14, 7, 5, 20, 1, 22, 27, 8, 17, 20, 17]

   [0, 0, 1, 0, 28, 7, 8, 4, 0, 17, 22, 15, 19, 25, 21, 7, 15, 5, 20, 12, 22]

   [0, 0, 0, 1, 8, 1, 1, 16, 22, 24, 23, 25, 10, 7, 13, 22, 8, 10, 0, 13, 3]
```

```
> FindAutGroup2(",29);print('The size of Aut(C) is :',nops("));
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21],

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   [16, 21, 18, 19, 20, 15, 17, 6, 1, 5, 7, 3, 4, 2, 8, 9, 11, 12, 13, 10, 14],
```

```
   [17, 15, 16, 20, 18, 19, 21, 1, 4, 2, 5, 6, 7, 3, 10, 14, 8, 11, 12, 9, 13],

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

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

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

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

                          The size of Aut(C) is :, 21

------------------------------------------------------------------------------
> LofG:=[1,z1,v1*z1,v1,w1*z1,w1*v1,w1];
                                  2    3
                                 Y    Y       X       X
              LofG := [1, - 1/Y, - ----, ----, - ----, Y, ----]
                                 X    X       3       2
                                             Y       Y

------------------------------------------------------------------------------
> C2:=CreateMatrix2(LofG,D,29);
                                2    3
                               Y    Y       X       X
           C2 := [1, - 1/Y, - ----, ----, - ----, Y, ----], C1Max
                               X    X       3       2
                                            Y       Y

------------------------------------------------------------------------------
> Gaussjord(C2[2]) mod 29;
 [ 1  0  0  0  0  0  0  12  16   4   0  14   7   6   6  16  12   7   4   0  14 ]
 [                                                                             ]
 [ 0  1  0  0  0  0  0   0   6  14   7  16   4  12   0   7   6   4  14  16  12   0 ]
 [                                                                             ]
 [ 0  0  1  0  0  0  0  16   7   6   4  12   0  14   0  14   6  12   7  16   4 ]
 [                                                                             ]
 [ 0  0  0  1  0  0  0   0   4  12  14   7   6  16  14   7  16   0   6   4  12 ]
 [                                                                             ]
 [ 0  0  0  0  1  0  0   7   0  14   6  16   4  12  16   4   0   6  12  14   7 ]
 [                                                                             ]
 [ 0  0  0  0  0  1  0   4   6  16  12   0  14   7   4  12  14  16   0   7   6 ]
 [                                                                             ]
 [ 0  0  0  0  0  0  1  14  12   0   7   6  16   4  12   0   7   4  14   6  16 ]

------------------------------------------------------------------------------
> FindAutGroup2(",29);print(`The size of Aut(C) is :`,nops("));
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21],

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

                          The size of Aut(C) is :, 21

------------------------------------------------------------------------------
>LofG:=[1,w1,w2,w1^2,v1,v2,v1^2,z1,z2,z1^2,w1*z1,(w1*z1)^2,v1*z1,(v1*z1)^2,
        w1*v1,(w1*v1)^2];

                          2    3       5    6
                 X       X    Y       Y    Y            X    1       X
  LofG := [1, ----, X/Y, ----, ----, - ----, ----, - 1/Y, ----, ----, - ----,
                2         4    X       2    2           4    2       3
               Y         Y             X    X           Y    Y       Y

        2       2    4
       X       Y    Y       2
```

```
       ----, - ----, ----, Y, Y ]
         6       X    2
        Y            X

------------------------------------------------------------------------------
> C3:=CreateMatrix2(LofG,D,29);
                          2    3       5    6                              2
                 X       X    Y       Y    Y            X    1       X    X
C3 := [1, ----, X/Y, ----, ----, - ----, ----, - 1/Y, ----, ----, - ----, ----,
                2         4    X       2    2           4    2       3    6
               Y         Y             X    X           Y    Y       Y    Y

              2    4
             Y    Y       2
           - ----, ----, Y, Y ],
             X    2
                  X

    C1Max

------------------------------------------------------------------------------
> Gaussjord(C3[2]) mod 29;
    [ 1  0  0  0  0  0  0  0  0  0  0  0  0  0   1  0  0  0   2  27  18  12 ]
    [                                                                       ]
    [ 0  1  0  0  0  0  0  0  0  0  0  0  0  0   1  0  0  0   4  16  26  13 ]
    [                                                                       ]
    [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0   1  0  0  0   5  16  12  26 ]
    [                                                                       ]
    [ 0  0  0  1  0  0  0  0  0  0  0  0  0  0   1  0  0  0  22  18  11   8 ]
    [                                                                       ]
    [ 0  0  0  0  1  0  0  0  0  0  0  0  0  0   1  0  0  0   6   3  23  27 ]
    [                                                                       ]
    [ 0  0  0  0  0  1  0  0  0  0  0  0  0  0   1  0  0  0   0   8   9  13 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0   1  0  0  0  17   7   1   5 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  1  0  0  0  0  0  0  28  0  0  0   8  27   5  18 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  28  0  0  0  23  19  26  19 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  28  0  0  0   6  16  27   9 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  28  0  0  0  25  12   1  20 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  28  0  0  0  27  21   1   9 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  28  0  0  0   5   4   5  15 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  0  0  0  0  0  1   0  1  0  0  18  11  10  18 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0   0  0  1  0  18  10  18  11 ]
    [                                                                       ]
    [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0   0  0  0  1  18  18  11  10 ]

------------------------------------------------------------------------------
> C3d:=DualCode(",29);
                                C3d := tempMat

------------------------------------------------------------------------------
> Gaussjord(C3d) mod 29;
     [1, 0, 0, 0, 8, 7, 28, 0, 15, 4, 6, 5, 13, 9, 4, 10, 7, 7, 4, 2, 15]

    [0, 1, 0, 0, 1, 1, 8, 0, 21, 1, 16, 13, 3, 12, 13, 20, 15, 26, 16, 0, 7]

    [0, 0, 1, 0, 28, 21, 22, 0, 7, 8, 4, 27, 9, 13, 6, 12, 14, 12, 9, 1, 9]

   [0, 0, 0, 1, 22, 1, 1, 0, 15, 16, 3, 13, 4, 24, 5, 15, 21, 12, 28, 25, 26]

     [0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 28, 28, 28, 28, 28, 28, 28]

------------------------------------------------------------------------------
> FindAutGroup2(",29);print(`The size of Aut(C) is :`,nops("));
[[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21],

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

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

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

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

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

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

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

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

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

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

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

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

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

   [15, 20, 21, 16, 17, 18, 19, 7, 5, 6, 3, 4, 2, 1, 14, 12, 9, 10, 8, 13, 11],
```

```
   [16, 21, 18, 19, 20, 15, 17, 6, 1, 5, 7, 3, 4, 2, 8, 9, 11, 12, 13, 10, 14],

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

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

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

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

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

                          The size of Aut(C) is :, 21

--------------------------------------------------------------------------------
>
```

Bibliography

[Gop88] V.D. Goppa. *Geometry and Codes*. Mathematics and its applications. Kluwer Academic Publishers, 1988.

[LeB96] D. LeBrigand. Quadratic algebraic function fields with ideal class number two. In Pellikaan, Perret, and Vladut, editors, *Arithmetic, Geometry and Coding Theory*, pages 105–125. de Gruyter, Berlin-New York, 1996.

[Leo84] J.S. Leon. Computing automorphism groups od combinatorial objects. In M. D. Atkinson, editor, *Computational Group Theory: proceedings of the LMS symposium on computational group theory*, pages 321–335. Academic Press, 1984.

[Mor91] C. Moreno. *Algebraic Curves over finite fields*. Cambridge University Press, 1991.

[MP93] C. Munuera and R. Pellikaan. Equality of geometric Goppacodes and equivalence of divisors. *Journal of Pure and Applied Algebra*, 90:229–252, 1993.

[MS77] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-Holland, 1977.

[Rom92] S. Roman. *Coding and Information Theory*. Graduate Texts in Mathematics 134. Springer-Verlag, 1992.

[Sil86] J. H. Silverman. *The Arithmetic of Elliptic Curves*. Springer Verlag, 1986.

[Sti88] H. Stichtenoth. Self-dual goppa codes. *Journal of Pure and Applied Algebra*, 55:199–211, 1988.

[Sti90] H. Stichtenoth. On automorphisms of geometric Goppa codes. *Journal of Algebra*, pages 113–121, 1990.

[Sti91] H. Stichtenoth. *Algebraic Function Fields and Codes*. Springer Verlag, 1991.

[vL82] J.H. van Lint. *Introduction to Coding Theory.* Graduate Texts in Mathematics 86. Springer-Verlag, 1982.

[Xin95a] C. Xing. Automorphism group of elliptic codes. *Communications in Algebra*, 23(11):4061–4072, 1995.

[Xin95b] C. Xing. On automorphism groups of the Hermitian codes. *IEEE Transactions on Information Theory*, 41(6):1629–1635, Nov 1995.