



FACHBEREICH 12 INFORMATIK UND MATHEMATIK
INSTITUT FÜR MATHEMATIK

Bachelorarbeit

Analyse der Implementierung von algebraisch-geometrischen Codes

Jens Heinrich
Studiengang: Mathematik
Matrikelnummer: 4626985
Frankfurt am Main
23. September 2019

Betreuer: Prof. Dr. Alex Küronya
Zweitprüfer: Max Hahn-Klimroth

Danksagungen

Ich möchte an dieser Stelle meinen Korrekturlesern für ihre Verbesserungsvorschläge danken, erwähnt seien hier Axel Dörner, Petra Schneider, Carina Reinecke, Kelly Koch, Klaudia Heinrich, Jessica Brockmann.

Danke geht auch an Eva Kaufholz-Soldat der Schreibberatung der Goethe-Universität. Ich möchte auch Nathalie Brunkhorst-Kanaan, Anna-Elisabeth Burghard und Sabine Berg danken.

Es sei der `sage-coding-theory@googlegroups.com` Liste für ihre Kommentare und Hilfe gedankt.

Ich danke Lena Glaser und Oliver Pellmann für ihre Kommentare und Verbesserungsvorschläge auch noch in letzter Minute.

Desweiteren möchte ich Professor Alex Küronya für die Betreuung danken und für die Geduld, die er mit mir bewiesen hat.

Meinen Partnern Paul Meyer-Bussian und Celina Korzen danke ich dafür, dass sie mich in dieser stressigen Zeit unterstützt haben.

Letzterer danke ich auch explizit für die Unterstützung mit den Grafiken.

Abstract

This thesis gives a quick introduction into codingtheory and algebra to define algebraic-geometric codes and proof the correctness of their implementation in the **SAGE**-project. Additionally a few tests on memory and time usage have been done for syndrome and nearest-neighbor decoding.

It requires some basic knowledge in mathematics, but aims to be readable for a beginner in codingtheory.

Zusammenfassung

Diese Arbeit beginnt mit Grundlagen der Codierungstheorie und der Algebra, um algebraisch-geometrische Codes zu definieren und die Korrektheit ihrer Implementierung im **SAGE**-Projekt zu zeigen. Es wurden auch Tests zum Zeit- und Speicherbedarf von Syndrom- und Nächster-Nachbar-Decodierung durchgeführt.

Das Verständnis der Arbeit benötigt einige Grundlagen der Mathematik, richtet sich aber dennoch auch an Anfänger auf dem Gebiet der Codierungstheorie.

Inhaltsverzeichnis

| | | |
|----------|--|-----------|
| 1 | Einleitung | 5 |
| 1.1 | Historie | 5 |
| 1.2 | Aktuell | 7 |
| 2 | Grundlagen | 8 |
| 2.1 | Grundlagen der Codierungstheorie | 8 |
| 2.2 | Algebraische Grundlagen | 13 |
| 3 | Codes | 25 |
| 3.1 | Darstellung | 25 |
| 3.2 | Dualer Code | 27 |
| 3.3 | Reed-Solomon Code | 29 |
| 3.4 | Goppa Code | 32 |
| 3.5 | Encoder und Decoder | 40 |
| 3.5.1 | Encoder | 40 |
| 3.5.2 | Decodierungsgrundlagen | 40 |
| 3.5.3 | Nächster Nachbar Decodierung | 41 |
| 3.5.4 | Syndromedecoding | 42 |
| 4 | Implementierung | 46 |
| 4.1 | Initialisierung | 47 |
| 4.2 | Generatormatrix | 51 |
| 4.3 | Prüfmatrix | 53 |
| 4.4 | Encoder und Decoder | 54 |
| 4.4.1 | Encoder | 55 |
| 4.4.2 | Nearest Neighbour | 56 |
| 4.4.3 | Syndromedecoder | 56 |
| 4.5 | Tests | 60 |
| 5 | Schlussfolgerung | 68 |

| | | |
|----------|--------------------------------|-----------|
| A | Mathematischer Anhang | 69 |
| A.1 | Homomorphiesatz | 69 |
| A.2 | Riemann-Roch Theorem | 70 |
| B | Werkzeuge | 71 |
| B.1 | Das Dokument | 71 |
| B.2 | Eigene Codefragmente | 75 |

Kapitel 1

Einleitung

Das Ziel dieser Arbeit ist, dem Lesereinen groben Einblick in die Codierungstheorie am Beispiel der Goppa Codes und der Implementierbarkeit anhand der bestehenden Implementierung zu geben.

1.1 Historie

Die Morsecodes von Samuel Morse und den nach ihm folgenden Tüftlern und Erfindern sind ein gutes Beispiel für das Encodieren (Buchstaben werden hier in Folgen von „lang-“, „kurz.“ und „Pause—“ umgewandelt) und nachfolgendes Decodieren (Folgen von „lang-“, „kurz.“ und „Pause—“ werden in Buchstaben umgewandelt).

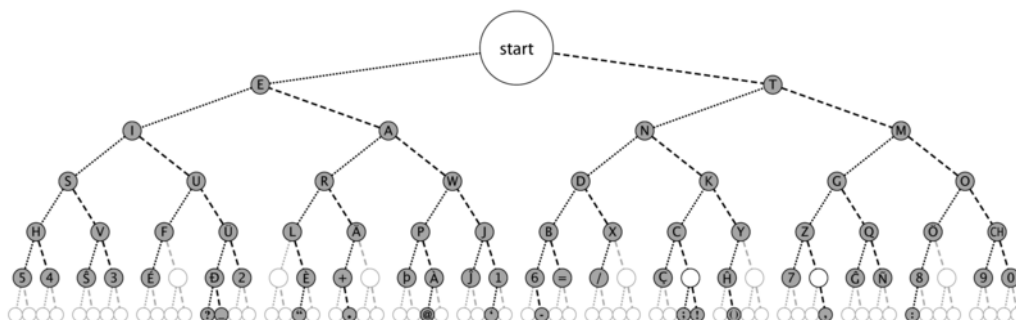
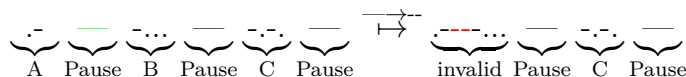


Abbildung 1.1: Morsecode Tree 

Später kamen der Wunsch und die Notwendigkeit dazu, Fehler in der Übertragung zu erkennen.

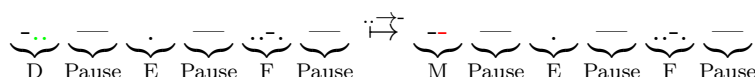
Teilweise können auch bei Morsecodes Fehler erkannt werden, so ist ABC mit einem

Fehler hier



nach Blick auf 1.1 kein zulässiges Codewort mehr und wird als falsch empfangenes Wort erkannt.

Dennoch ist der Morsecode kein Fehlererkennender Code, da z.B.



ein Fehler ist, aber nicht als solcher erkennbar ist.

Schon 1947 wurden laut [134] von Hamming die ersten Fehlerkorrigierende Codes entwickelt und 1950 in [54] veröffentlicht.

Diese in 3.5 beschriebenen Codes, geben im Rahmen der vorgesehenen Decodierung und der „Standarddarstellung“ die Errorlocator zurück.

Im Rahmen der Voyager Missionen in den 1970er Jahren wurden die damals noch experimentellen Reed-Solomon Codes eingesetzt, um Daten, wie z.B. das Bild 1.1 an die Erde zu übertragen.

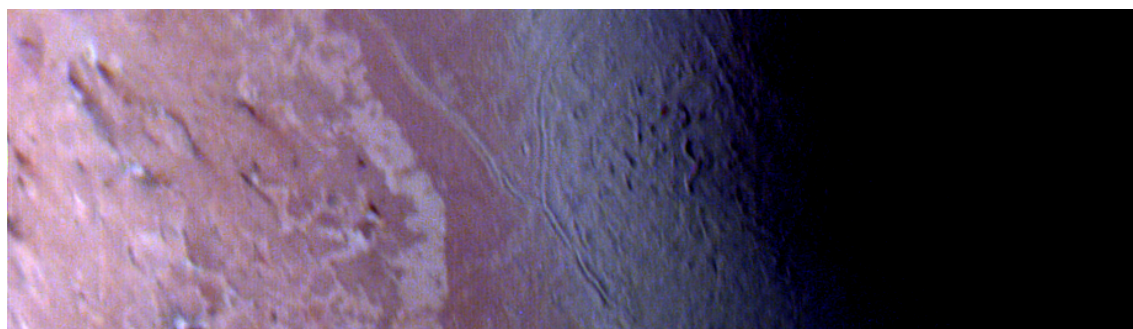


Abbildung 1.2: Voyager violet, green, and ultraviolet images of Triton were map projected into cylindrical coordinates and combined to produce this false color terrain map. Several compositionally distinct terrain and geologic features are portrayed. At center is a gray blue unit referred to as 'cantaloupe' terrain because of its unusual topographic texture. The unit appears to predate other units to the left. Immediately adjacent to the cantaloupe terrain, is a smoother unit, represented by a reddish color, that has been dissected by a prominent fault system. This unit apparently overlies a much higher albedo material, seen farther left. A prominent angular albedo boundary separates relatively undisturbed smooth terrain from irregular patches which have been derived from breakup of the same material. Also visible at the far left are diffuse, elongated streaks, which seem to emanate from circular, often bright centered features. The parallel streaks may represent vented particulate materials blown in the same direction by winds in Triton's thin atmosphere. The Voyager Mission is conducted by JPL for NASA's Office of Space Science and Applications. Courtesy NASA/JPL-Caltech.

Der damals verwendete Code wird in Abschnitt 3.3 als Beispiel ?? erklärt.

1.2 Aktuell

Heute ist man überall von fehlererkennende Codes und fehlerkorrigierende Codes umgeben; sie begleiten uns auf Schritt und Tritt z.B. in Form der ISBN-13.

1.1 Definition (ISBN-13[79, S. 1.1.1])

Die aus dem Jahr 2007 stammenden ISBN-13 Codes bestehen aus 5 Elementen nach [69, Abschnitt 5]:

- *GS1 Element*
- *Gruppennummer*
- *Verlagsnummer*
- *Titelnummer*
- *Prüfziffer*

bestehen aus 12 Ziffern und einem 13. Prüfzeichen (und sind so gewählt, dass sie mit der EAN übereinstimmen). Die 12 Zeichen bestehen aus einem Zeichen, welches die Region festlegt und 11 Zeichen, die sich aus einer in der Länge nicht festgelegten Verlagsnummer und einer Titelnummer zusammensetzen.

Damit $x_1x_2x_3x_4x_5x_6x_7x_8x_9x_{10}x_{11}x_{12}x_{13}$ eine legitime ISBN-13 ist, muss gelten [69, Anhang 1]:

$$x_{13} \equiv \sum_{i=1}^6 \underset{\text{Gewichtung}}{1} \cdot x_{2 \cdot i + 1} + \sum_{i=1}^6 \underset{\text{Gewichtung}}{3} \cdot x_{2 \cdot i} \pmod{10} \quad (1.1)$$

1.2.1 Beispiel

So lautet die ISBN von [69] 978929505512 und einsetzen in (1.1) ergibt:

$$(9 \cdot 18 \cdot 12 \cdot 15 \cdot 15 \cdot 11 \cdot 14 \cdot 1) + (7 \cdot 39 \cdot 39 \cdot 30 \cdot 35 \cdot 32 \cdot 3) = (34) + (96) = (130) \equiv 0 \pmod{10}$$

Weitere Beispiele für Fehlerkorrigierende Codes, mit denen man im Alltag in Kontakt kommt, sind IBAN und QR-Code.

Auch im Hintergrund sind wir von fehlerkorrigierende Codes umgeben, so ist jegliche Funktechnik anfällig für Übertragungsfehler und die Implementierungen von wi-fi.org und Mobilfunk haben entsprechende Fehlerkorrektur-Mechanismen.

Kapitel 2

Grundlagen

In den folgenden Abschnitten werden die nötigen mathematischen Grundlagen für die restliche Arbeit gelegt. Vieles wird Lesern, die schon Kontakt mit Algebra bzw. Codierungstheorie hatten, bekannt sein, dennoch wird werden die Grundlagen im Sinne einheitlicher Definitionen, der späteren Referenzierbarkeit und der Nachvollziehbarkeit ausgeführt.

2.1 Grundlagen der Codierungstheorie

Grundlegende Begriffe der Codierungstheorie übernehmen wir von Walker bzw. Pries und Malmskog, behalten die Notation aus dem Englischen und ergänzen abweichende Notationen in anderen Quellen, wenn möglich.

2.1 Definition (Code und Alphabet [73] [65, S. 1.1, 1.2,])

Ein Alphabet A ist eine Menge von Zeichen und ein Code \mathcal{C} von Länge n ist eine Teilmenge von A^n ; hier gilt $n \in \mathbb{N}$, aber es existieren auch Arbeiten, die sich mit Codes über unendlichen Körpern beschäftigen, z.B. [80]

Ein Element aus \mathcal{C} nennen wir Codewort und bezeichnen die Anzahl der Codewörter in \mathcal{C} mit $\#\mathcal{C}$. Um einen Begriff auch für Elemente von A^n zu haben, die kein Codewort sind, nennen wir eine beliebiges Element von A^n Wort.

In der Regel werden wir für A ein \mathbb{F}_q mit $q = p^m$ für p prim und $m \in \mathbb{N}$ wählen, also über Codes als Teilmengen von \mathbb{F}_q^n sprechen.

2.2 Definition (Systematische Codes [54])

Ein systematischer Code ist ein Code, bei dem jedes Codewort aus der gleichen Anzahl von Zeichen besteht.

Systematische Codes haben Vorteile gegenüber anderen Codes, da man bei der Übertragung keine zusätzlichen Zeichen zur Trennung benötigt. Ein Beispiel für einen nicht ist der Morse Code aus der Einleitung.

Im Folgenden wird in der Regel von linearen Codes gesprochen und Linearität ist bei Codes äquivalent zu Funktionen definiert:

2.3 Definition (linearer Code [73, S. 2.5])

Ein Code \mathcal{C} heisst linear, wenn er die folgenden drei Eigenschaften erfüllt:

- *Der Vektor $\vec{0} = (0, \dots, 0)$ ist ein Codewort in \mathcal{C}*
- *$\forall \vec{x} = (x_1, \dots, x_n) \in \mathcal{C}$ und $\vec{y} = (y_1, \dots, y_n) \in \mathcal{C}$ folgt, dass $\vec{x} + \vec{y} = (x_1 + y_1, \dots, x_n + y_n) \in \mathcal{C}$*
- *$\forall \vec{x} = (x_1, \dots, x_n) \in \mathcal{C}$ und αA folgt, dass $\alpha \vec{x} = (\alpha x_1, \dots, \alpha x_n) \in \mathcal{C}$*

2.1.1 Lemma

Jeder lineare Code hat eine Systematische Repräsentation.

Beweis. Sei \mathcal{C} ein linearer Code, dann ist $\mathcal{C} \stackrel{2.3}{\subset} A^n$. Über die natürliche Einbettung hat jedes Element c eine Repräsentation in A^n . □

Um verschiedene Codes vergleichen zu können, benötigen wir Kriterien, wie die im folgenden definierten *Kennziffern*.

2.4 Definition (Kennziffern von Codes [79, Definition 1.2.1] [73, Definition 2.6])

Sei \mathcal{C} ein linearer Code.

- *Die Dimension \mathcal{C} ist die minimale Anzahl der Codewörter, die nötig ist, um den vollständigen Code darzustellen, also die Mächtigkeit einer Basis von \mathcal{C} als Teilmenge von Alphabet^n .*
- *Eine Menge wie eben beschrieben nennt man Basis von \mathcal{C} .*
- *Die Länge n von \mathcal{C} ist die Dimension des Raums der Wörter und die Dimension k eines Codes (als Untervektorraum) nennen wir seine Parameter.*
- *Eine $k \times n$ Matrix, in deren Zeilen eine Basis von \mathcal{C} steht, Generatormatrix für \mathcal{C} .*

Einen Code mit Länge n und Dimension k bezeichnen wir als (n, k) -Code.

Für den Vergleich verschiedener Codewörter untereinander benutzen wir die folgende Definition:

2.5 Definition (Hammingdistanz und -gewicht [65, S. 1.3])

Für $\vec{x} = (x_1, \dots, x_n) \in \mathcal{C}$ und $\vec{y} = (y_1, \dots, y_n) \in A^n$ ist die Hamming - Distanz zwischen \vec{x} und \vec{y} definiert als

$$d(\vec{x}, \vec{y}) = \# \{i : x_i \neq y_i\} \quad (\text{hamming-distance})$$

Das Hamming -Gewicht (englisch weight) eines Vektors \vec{x} ist definiert als

$$\text{wt}(\vec{x}) = d(\vec{x}, \vec{0}) = \# \{i : x_i \neq 0\} \quad (\text{hamming-weight})$$

2.1.2 Lemma (Hamming Metrik)

Die Gleichung hamming-distance ergibt wie von Walker in [65, Exercise 1.4] als Aufgabe gestellt eine Metrik.

$$d(\vec{x}, \vec{y}) \geq 0 \text{ mit } d(\vec{x}, \vec{y}) = 0 \Leftrightarrow \vec{x} = \vec{y} \quad (\text{Positive Definitheit})$$

$$d(\vec{x}, \vec{y}) = d(\vec{y}, \vec{x}) \quad (\text{Symmetrie})$$

$$d(\vec{x}, \vec{y}) + d(\vec{y}, \vec{z}) \geq d(\vec{x}, \vec{z}) \quad (\text{Dreiecksungleichung})$$

Beweis. Angenommen, $\vec{x} \neq \vec{y}$ und $d(\vec{x}, \vec{y}) = 0$, dann gilt, aber nach (hamming-distance) gilt, dass $x_i = y_i \forall i$ und das ist ein Widerspruch zu $\vec{x} \neq \vec{y}$, also gilt $\vec{x} = \vec{y} \Rightarrow d(\vec{x}, \vec{y}) = 0$.
folgt, da $\vec{x} = \vec{y} \Rightarrow x_i = y_i$ gilt.

Da d „zählt“, ist die Aussage der (Positive Definitheit) gegeben.

Da $x_i \neq y_i \Leftrightarrow y_i \neq x_i$ gilt, folgt die Aussage von (Symmetrie)

Fallunterscheidung:

| | |
|---------------------------|--|
| $x_i = y_i = z_i :$ | $d_i(x_i, y_i) + d_i(y_i, z_i) = 0 + 0 \geq 0 = d_i(x_i, z_i)$ |
| $x_i = y_i \neq z_i :$ | $d_i(x_i, y_i) + d_i(y_i, z_i) = 0 + 1 \geq 1 = d_i(x_i, z_i)$ |
| $x_i \neq y_i = z_i :$ | $d_i(x_i, y_i) + d_i(y_i, z_i) = 1 + 0 \geq 1 = d_i(x_i, z_i)$ |
| $x_i = z_i \neq y_i :$ | $d_i(x_i, y_i) + d_i(y_i, z_i) = 1 + 1 \geq 2 = d_i(x_i, z_i)$ |
| $x_i \neq z_i \neq y_i :$ | $d_i(x_i, y_i) + d_i(y_i, z_i) = 1 + 1 \geq 2 = d_i(x_i, z_i)$ |

Da die Ungleichung eintragsweise gilt, gilt sie auch für die gesamte Funktion, schliesslich ist auch die Funktion hamming-distance eintragsweise definiert.

Mit dieser Eigenschaft von Codewörtern lässt sich wieder eine Eigenschaft des gesamten Codes definieren:

2.6 Definition (Minimaldistanz [73, S. 2.8])

Die Minimaldistanz eines Codes \mathcal{C} ist definiert als

$$d_{\min}(\mathcal{C}) = \min \{w(\vec{x}) : \vec{x} \in \mathcal{C}\} \quad (2.2)$$

2.1.3 Lemma ([65, Exercise 1.6])

Sei \mathcal{C} ein linearer Code dann gilt

$$d(a, b) \geq d_{\min}(\mathcal{C}) \quad \forall a, b \in \mathcal{C} \quad (2.3)$$

Beweis.

$$\begin{aligned} a, b \in \mathcal{C} &\stackrel{\mathcal{C}}{\Rightarrow} a - b = c \in \mathcal{C} \\ d(a, b) &= d(c + b, b) \\ &= d(c, 0) \\ &= \text{wt}(c) \\ &\geq d_{\min} \end{aligned}$$

□

Mit dieser Definition kann man den Begriff eines (n, k) -Codes aus 2.4 erweitern und für einen Code mit Länge n , Dimension k und Minimaldistanz d (n, k, d) -Codes schreiben.

2.7 Definition (Kugel in \mathbb{F}_q [73, S. 2.9])

Eine Kugel in \mathbb{F}_q^n um \vec{x} mit Radius r ist

$$\mathcal{B}_r(\vec{x}) = \{ \vec{y} \in \mathbb{F}_q^n \mid d(\vec{x}, \vec{y}) \leq r \} \quad (2.4)$$

Aus der Minimaldistanz (2.6) und der Definition 2.7 folgt, dass $d_{\min}/2$ Fehler erkannt und korrigiert werden können, indem man dem beliebigen Wort das Codewort mit geringster Distanz zuordnet, siehe 3.17.

2.1.4 Theorem (Shannonons Noisy Code Theorem [53, Theorem 11] [61, 53, 66])

Ein diskreter Kanal habe die Kapazität C und eine diskrete Quelle habe die Entropie H .

Wenn $H \leq C$, dann gibt es ein Kodierungssystem, sodass Ausgabe der Quelle über den Kanal mit beliebig niedriger Häufigkeit von Fehlern übertragen werden kann.

Wenn $H \leq C$, dann ist es möglich die Quelle so zu encodieren, dass die Äquivokation kleiner als $H - C + \epsilon$, wobei ϵ beliebig klein ist.

Es gibt keine Methode der ??, die eine Äquivokation $< H - C$ liefert.

Desweiteren können wir mit folgender Definition die *Gilbert-Varshamov Schranke* festlegen.

2.8 Definition (Anzahl der Punkte in einer Kugel [73, S. 2.12])
 $V_q(n, r)$ ist definiert als die Anzahl der Punkte von \mathbb{F}_q^n in einer Kugel mit Radius r um einen beliebigen Punkt von \mathbb{F}_q^n

Die folgende Abschätzung wird von Pries und Malmskog als Übungsaufgabe gestellt.

2.1.5 Lemma (Formel für die Anzahl der Punkte in einer Kugel [73, Problem Session 2 7.])

Sei $V_q(n, r)$ wie eben definiert, dann gilt

$$V_q(n, r) = \sum_{i=0}^r \binom{n}{i} (q-1)^i, \quad (2.5)$$

wobei

$$\binom{n}{k} = \frac{n!}{k! (n-k)!} \quad (2.6)$$

der übliche Binomialkoeffizient ist. A

Beweis. Da jede Kugel die nächst kleinere enthält, können wir folgende Hilfsformel definieren:

$$\bar{\mathcal{B}}_r(\vec{x}) := \left\{ \vec{y} \in \mathbb{F}_q^n \mid d(\vec{x}, \vec{y}) \leq r \right\} \quad (2.7)$$

Hierfür können wir die folgende Formel aufstellen, da die Hamming -Distanz eintragsweise definiert ist und wir für eine Distanz von r eine Abweichung an r von n Stellen brauchen, also $\binom{n}{r}$ Fehlerstellen und aufgrund der Mächtigkeit von \mathbb{F}_q für jede dieser Fehlerstellen $q-1$ mögliche Fehler haben. Da nur die Anzahl der unterschiedlichen Stellen relevant ist, können wir den Mittelpunkt „verlieren“:

$$\#\bar{\mathcal{B}}_r := \binom{n}{r} (q-1)^r \quad (2.8)$$

Mit der Hilfsformel (2.8) gilt:

$$V_q(n, r) = \sum_{i=0}^r \#\bar{\mathcal{B}}_i = \sum_{i=0}^r \binom{n}{i} (q-1)^i \quad \square$$

2.1.6 Theorem (Gilbert-Varshamov Schranke [73])

Für einen linearen Code mit maximaler Anzahl Codewörtern M von Länge n und Minimaldistanz d gilt

$$M = \frac{q^n}{V_q(n, d-1)}. \quad (2.9)$$

Für den Beweis dieser Schranke sei auf [55, Theorem 1] verwiesen.

2.9 Definition ([65, Definition 1.9] zyklische Codes)

Ein linearer Code heisst zyklisch, wenn für alle $(c_0, c_1, \dots, c_{n-1})$ auch $(c_1, c_2, \dots, c_{n-1}, c_0A) \in \mathcal{C}$

Sei

\mathcal{C} ein zyklischer Code und $R := \mathbb{F}_q[x] / (x^n - 1)$

dann können wir

\mathcal{C} mit $I_{\mathcal{C}} := \{c(x) := c_0 + c_1x + \dots + c_{n-1}x^{n-1} \in R_n \mid c := (c_0, c_1, \dots, c_{n-1}) \in \mathcal{C}\}$ identifizieren.

2.2 Algebraische Grundlagen

Im folgenden Abschnitt werden einige grundlegende Begriffe der Algebra eingeführt und auch gelegentlich Aussagen ohne Beweis eingeführt (für die Beweise wird jedoch auf andere Arbeiten verwiesen).

2.10 Definition (Algebraischer Abschlusses [65, S. 3.1, 3.3])

Ein Körper k ist algebraisch abgeschlossen, wenn jedes Polynom in $k[x]$ mindestens eine Nullstelle hat.

Ein algebraischer Abschluss von k ist ein Körper K mit $k \subseteq K$, mit folgenden Eigenschaften:

- K ist algebraisch abgeschlossen und
- Wenn L ein Körper ist, sodass $k \subseteq L \subseteq K$ und L algebraisch abgeschlossen ist, dann folgt $L = K$

Damit diese Definition nutzbar ist, müssen solche Abschlüsse existieren und daher benötigen wir das Theorem 2.2.1.

2.2.1 Theorem (Existenz des algebraischen Abschluss [65])

Jeder Körper hat einen bis auf Isomorphie eindeutigen algebraischen Abschluss.

Für den Beweis dieses Theorems sei z.B. auf [111] verwiesen.

Mit dieser Annahme kann man Polynome auf Körpern besser verstehen.

Das folgende Lemma wird von Walker als Übungsaufgabe gestellt und in dem Beweis des darauf folgenden Theorems 2.2.3 benötigt.

**2.2.2 Lemma** ([65, B.10])

Sei k ein Körper und sei $g(y) \in k[x]$.

1. *Sei $g(r) = 0$ für ein $r \in k$. Dann gibt es ein $f \in k$, sodass $g(x) = (x - r)f(x)$.*
2. *$g(x)$ hat höchstens $\deg(g)$ Nullstellen.*

Beweis. Da $k(x)$ ein euklidischer Ring ist, gibt es eine Darstellung von $g(x) = (x - r)f(x) + r(x)$ mit $\deg r < \deg(x - r) = 1$. Mit $0 = g(r) = (r - r)f(r) + r$ folgt $g(x) = (x - r)f(x)$.

Da jedes Abspalten einer Nullstelle den Grad der Funktion um 1 erniedrigt, kann $g(x)$ höchstens $\deg(g)$ Nullstellen haben. \square

2.2.3 Theorem (Vielfachheit von Nullstellen[65, S. 3.5])

Sei k ein algebraisch abgeschlossener Körper und sei $f(x) \in k[x]$ ein Polynom vom Grad n . Dann gibt es ein $u \in k^\times := k \setminus \{0\}$ und $\alpha_1, \dots, \alpha_n \in k$ (nicht notwendigerweise einzigartig), sodass $f(x) = u(x - \alpha_1) \cdots (x - \alpha_n)$. Insbesondere hat f genau n Nullstellen mit Vielfachheit in k .

Das Lemma 2.2.2 liefert die Grundlagen für den folgenden Beweis.

Beweis. Dieser Beweis lässt sich über Induktion nach dem Grad n der Funktion f .

Sei $n = 0$, dann ist f konstant und $f \in k^\times$. Somit gibt es keine Nullstellen. \checkmark

Angenommen man kann jedes Polynom von Grad n in der oben genannten Form schreiben und sei $f(x) \in k[x]$ mit Grad $n + 1$. Dann können wir mit der Aussage ?? folgern, dass es ein $g(x) \in k[x]$ mit $\deg(g) = n$ gibt.

Die Aussage folgt induktiv. \square

2.11 Definition (algebraische Kurve [65, S. 3.2])

Wenn $f(x, y)$ ein Polynom in zwei Variablen ist, dann definiert $f(x, y) = 0$ eine algebraische Kurve \mathcal{C}_f .

Sei k ein Körper, dann wird die Menge der Lösungen von $f(x, y) = 0$ über k mit $\mathcal{C}_f[k]$ bezeichnet.

Das folgende Beispiel wird von Walker als Übungsaufgabe gestellt und soll auch hier algebraische Kurven verständlicher machen.

2.2.4 Beispiel ([65, S. 3.6])

Finde alle rationalen Lösungen für $x^2 - 2y^2 = 1$.

Wir lösen dies graphisch

und betrachten die über $f : x^2 - 2y^2 - 1$ definierte Kurve \mathcal{C}_f .

Die folgenden Graphiken sind mit 1000 Datenpunkten gezeichnet.

(a)

(1, 0)

liegt auf der hyperbel und es gibt einen weiteren Punkt mit $y = 0$, nämlich der Punkt $(-1, 0)$.

(b) Sei L eine Gerade mit rationaler Steigung t durch den Punkt $(1, 0)$. Diese wird durch $y = (x, t) \mapsto tx - t$ beschrieben.

(c) Die Gleichung $f(x, p(x))$ hat genau 2 Lösungen (!sic Nullstellen) (x, y) , wobei $(1, 0)$ eine ist und die andere eine Lösung von $x^2 - 2y^2 = 1$ ist.

Beweis. Durch Einsetzen erhält man

$$\begin{aligned} f(x, p(x)) &= x^2 - 2(tx - t)^2 - 1 \\ &= x^2 - 2(t^2x^2 - 2t^2x + t^2) - 1 \\ &= x^2(1 - 2t^2) + x4t^2 - 2t^2 - 1 \end{aligned}$$

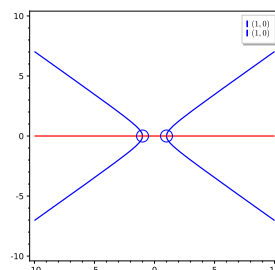


Abbildung 2.1: Schnitt der hyperbel mit der Gerade $y = 0$

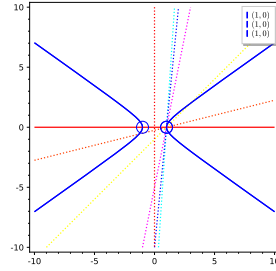


Abbildung 2.2: Geraden durch $(1, 0)$

und da eine Nullstelle betrachtet wird

$$0 = x^2 \underbrace{(1 - 2t^2)}_a + x \underbrace{4t^2}_b \underbrace{- 2t^2 - 1}_c$$

folgt mit der Mitternachtsformel

$$\begin{aligned} x_{1/2} &= -\frac{1}{2} \left(\frac{b \pm \sqrt{b^2 - 4ac}}{a} \right) \\ &= -\frac{1}{2} \left(\frac{4t^2 \pm \sqrt{16t^4 - 4(2t^2 - 1)(2t^2 + 1)}}{(1 - 2t^2)} \right) \end{aligned}$$

Wir sehen eine Bedingung an t : $t \neq \pm \frac{1}{\sqrt{2}}$

$$\begin{aligned} &= -\frac{1}{2} \left(\frac{2^2 t^2 \pm 2\sqrt{2^2 t^4 - 2t^2 + 1 - 2^2 t^4 - 2t^2}}{(1 - 2t^2)} \right) \\ &= \left(\frac{2t^2 \pm 1}{2t^2 - 1} \right) \end{aligned}$$

und es gilt

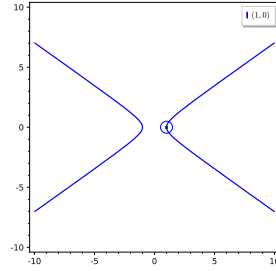
$$x_{1/2}, p(x_{1/2}) \tag{2.10}$$

ist eine rationale Lösung der Gleichung $x^2 - 2y^2 = 1$. \square

(d) $x(t) = \frac{2t^2+1}{2t^2-1}$ und $y(t) = p(x(t))$ definieren unendlich viele Lösungen für $x^2 - 2y^2 = 1$.

Beweis. $(x(t), y(t))$ ist eine Lösung, wie wir im vorherigen Schritt gesehen haben und da die Mächtigkeit von \mathbb{Z} unendlich ist, folgt die Aussage. \square

(e) $x(t), y(t)$ liefert alle rationalen Lösungen bis auf 2.

Abbildung 2.3: rationale Lösungen der Gleichung $x^2 - 2y^2 = 0$:**2.12 Definition** (Projektive Gerade [65])

Die projektive Gerade $\mathbb{P}^1(\mathbb{F}_q)$ ist definiert als

$$(\mathbb{F}_q^2 \setminus \{(0,0)\}) / \sim,$$

wobei $(X_0, Y_0) \sim (X_1, Y_1)$ genau dann, wenn $\exists \alpha \in \mathbb{F}_q^\times$, sodass $X_1 = \alpha X_0$ and $Y_1 = \alpha Y_0$

2.2.5 Lemma

Wenn man $(X_0 : Y_0)$ für die Äquivalenzklasse der Punkte (X_0, Y_0) schreibt, erhält man

$$\mathbb{P}^1(\mathbb{F}_q) = \{(\alpha : 1) | \alpha \in \mathbb{F}_q\} \cup \{(1 : 0)\}$$

\mathbb{P}^1 ist ein Kurve von Geschlecht 0 in \mathbb{P}^2 definiert durch die Gleichung $Z = 0$.

2.13 Definition (Projektive Ebene [65])

Sei k ein Körper. Die projektive Ebene $\mathbb{P}^2(k)$ ist definiert als

$$\mathbb{P}^2(k) := (k^3 \setminus \{(0,0,0)\}) / \sim,$$

wobei $(X_0, Y_0, Z_0) \sim (X_1, Y_1, Z_1)$ genau dann, wenn es ein $\alpha \in k^\times$ gibt, sodass $X_1 = \alpha X_0$, $Y_1 = \alpha Y_0$ und $Z_1 = \alpha Z_0$.

2.14 Definition (Projektiver Abschluss [65])

Sei k ein Körper, $f(x, y) \in \mathbb{F}^k[x, y]$ ein Polynom vom Grad d und \mathcal{C}_f die Kurve, die zu f assoziiert ist.

Der projektive Abschluss der Kurve \mathcal{C}_f ist $\hat{\mathcal{C}}_f := \{(X_0 : Y_0 : Z_0) \in \mathbb{P}^2 \mid F(X_0, Y_0, Z_0) = 0\}$, wobei $F(X, Y, Z) := Z^d f(X/Z, Y/Z) \in k[X, Y, Z]$ die Homogenisierung von f ist.

2.2.6 Theorem (Bezouts Theorem [65])

Wenn $f, g \in k[x, y]$ Polynome von Grad d und e , dann schneiden sich \mathcal{C}_f und \mathcal{C}_g in höchstens de Punkten. Desweiteren schneiden sich $\hat{\mathcal{C}}_f$ und $\hat{\mathcal{C}}_g$ in genau de Punkten von $\mathbb{P}^2(\bar{k})$, wenn die Punkte mit Vielfachheit gezählt werden.

[65]

2.15 Definition ([65])

$$R_n = \mathbb{F}_q[x] / \langle x^n - 1 \rangle$$

2.2.7 Theorem (Singleton Bound [65, Theorem 2.1] BZW. [63])

Sei \mathcal{C} ein linearer Code von Länge n , Dimension k und Minimumdistanz d_{\min} , dann $d \leq n - k + 1$.

Weitere wichtige Kennziffern für Codes werden von Lomont aus [66] übernommen.

2.16 Definition (Rate und Korrekturfähigkeit [66])

Die Rate \mathcal{R} eines fehlerkorrigierenden Codes $\mathcal{R} = \frac{k}{n}$ und die relative Korrekturfähigkeit $\delta = \frac{d}{n}$ liegen beide in $[0, 1]$.

Eine hohe Rate ermöglicht die Übertragung von mehr Informationen und eine hohe Korrekturfähigkeit ermöglicht die Korrektur von mehr Fehlern, aber eine Erhöhung des einen führt zu einer Erhöhung des anderen, wie das nächste Lemma zeigt.

2.2.8 Lemma ()

Korrekturfähigkeit und Rate hängen über den Singleton Bound ?? zusammen.

Beweis. Der Singleton-Bound liefert die Schranke $d \leq n - k + 1$ und durch Einsetzen folgt:

$$\delta = \frac{d}{n} \leq \frac{n - k + 1}{n} = \frac{n + 1}{n} - \frac{k}{n} = \frac{n + 1}{n} - \mathcal{R}$$

Durch Umstellen sieht man, dass die Summe nach oben beschränkt ist:

$$\delta + \mathcal{R} \leq \frac{n+1}{n}$$

□

2.17 Definition (Maximale Anzahl von Codewörtern[65])

Sei q eine Primpotenz und seien n, d natürliche Zahlen mit $d \leq n$. Dann ist die Anzahl $A_q(n, d)$ definiert als der maximale Wert von M , sodass es einen Code über \mathbb{F}_q von Länge n mit M Codewörtern und einer Minimaldistanz von d gibt.

2.2.9 Theorem (Plotkin Bound [65])

Sei $\Theta = 1 - \frac{1}{q}$. Dann $A_q(n, d) = 0$, wenn $d < \Theta n$ und

$$A_q(n, d) \leq \frac{d}{d - \Theta n}, \quad \text{für } d > \Theta n.$$

2.2.10 Beispiel ([65, Exercise 3.13])

Sei $f(x, y) = x^3 + x^2y - 3xy^2 - 3y^3 + 2x^2 - x + 5$, dann ist die Homogenisierung $F(X, Y, Z) = X^3 + X^2Y - 3XY^2 - 3Y^3 + 2X^2Z - XZ^2 + 5Z^3$. Damit sind die Lösungen der Gleichung $X^3 + X^2Y - 3XY^2 - 3Y^3 = Z(2X^2 - XZ + 5Z^2) = 0$ die Punkte in der Unendlichkeit, die auf dem Abschluss der ?? liegen.

Die Menge dieser Punkte ist

$$\{(X : 1 : 0) \mid X^3 + X^2 - 3X - 3 = 0\} = \left\{(-1 : 1 : 0), \left(-\frac{3}{2} : 1 : 0\right), \left(\frac{3}{2} : 1 : 0\right), \right\}.$$

Walker bezeichnet Kurven als „schön“, wenn sie eben und nichtsingulär sind, da einige Beweise dadurch einfacher werden.

Um eine Definition von singulär aufzustellen, wird die *formale Differenzierung* benötigt.

2.18 Definition (Formale Differenzierung[75, S. 11.2])

$$(f \pm g)' = (f)' \pm (g)' \quad (\text{Summenregel})$$

$$(f \cdot g)' = (f)' \cdot g + f \cdot (g)' \quad (\text{Produktregel})$$

$$\left(\frac{f}{g}\right)' = \frac{(f)' \cdot g - f \cdot (g)'}{g^2} \quad (\text{Quotientenregel})$$

$$(g \circ f)' = g'(f) \cdot f' \quad (\text{Kettenregel})$$

2.19 Definition (singuläre und nicht singuläre Punkte [65, Definition 4.1])

Sei k ein Körper und $f(x, y) \in k[x, y]$. Ein singulärer Punkt von \mathcal{C}_f ist ein Punkt $(x_0, y_0) \in \bar{k} \times \bar{k}$, sodass $f(x_0, y_0) = 0$ und sowohl $f_x(x_0, y_0) = 0$ als auch $f_y(x_0, y_0) = 0$.

Die Kurve \mathcal{C}_f ist nichtsingulär, wenn sie keine singulären Punkte hat.

Wenn $F(X, Y, Z)$ die Homogenisierung von $f(x, y)$ ist, dann ist $(X_0 : Y_0 : Z_0) \in \mathbb{P}^2(\bar{k})$ ein singulärer Punkt auf $\hat{\mathcal{C}}_f$, wenn der Punkt auf der Kurve liegt und alle partiellen Derivate dort verschwinden, zB wenn

$$\begin{aligned} F(X_0, Y_0, Z_0) &= F_X(X_0, Y_0, Z_0) \\ &= F_Y(X_0, Y_0, Z_0) \\ &= F_Z(X_0, Y_0, Z_0) \\ &= 0. \end{aligned}$$

Die Kurve $\hat{\mathcal{C}}_f$ ist nichtsingulär, wenn sie keine singulären Punkte hat.

2.2.11 Beispiel ([65, Exercise 4.2])

Sei $f(x, y) \in \mathbb{R}[x, y]$ und $(0, 0)$ nicht Singularität auf \mathcal{C}_f .

(a) Wenn $f_y(0, 0) \neq 0$, dann ist die Tangente von \mathcal{C}_f in $(0, 0)$ die Gerade $y = mx$, wobei $m = \frac{f_x(0, 0)}{f_y(0, 0)}$.

(b) Wenn $f_y(0, 0) = 0$, dann ist die Gerade $x = 0$ die Tangente an \mathcal{C}_f in $(0, 0)$.

Beweis. Um zu beweisen, dass es sich um eine Tangente handelt, muss gezeigt werden, dass die Ableitung übereinstimmt und dass der Wert der Funktion übereinstimmt.

$$f(0, 0) = 0$$

da die Kurve \mathcal{C}_f

und

$$y = \frac{f_x(0,0)}{f_y(0,0)}x \quad \text{durch Einsetzen}$$

$$0 = 0$$

bzw im Fall (b)

$$x = 0 \quad \text{durch Einsetzen}$$

$$0 = 0$$

Die Funktionswerte stimmen also überein
und es wird jetzt gezeigt, dass auch die Ableitungen übereinstimmen.

$$y = \frac{f_x(0,0)}{f_y(0,0)}x \quad |\delta x \qquad y = \frac{f_x(0,0)}{f_y(0,0)}x \quad |\delta y$$

$$0 = \frac{f_x(0,0)}{f_y(0,0)} \qquad 1 = \frac{f_x(0,0)}{f_y(0,0)}$$

□

2.2.12 Beispiel ([65])

Wir zeigen hier einige Beispiele für den Abschluss von Kurven in $\mathbb{R}[x, y]$ eingebettet in $\mathbb{R} \times \mathbb{R}$ über $f(x, y) = 0$
[65]

2.2.13 Beispiel (Eine Auswahl an Kurven [65, Exercise 4.4])

Die folgenden Graphiken sind mit $(x, y) \mapsto x^4 + y^4 - x^3 + y^2$ Datenpunkten gezeichnet.

- $f: (x, y) \mapsto x^6 + y^6 - xy$ 2.4
- $g: (x, y) \mapsto x^4 + y^4 - x^2y - xy^2$ 2.5
- $h: (x, y) \mapsto x^4 + y^4 - x^2$ 2.6
- $i: (x, y) \mapsto -x^3 + y^2$ 2.7

Unterscheidung: affine Punkte und Punkte in der Unendlichkeit

2.2.14 Beispiel ([65, Exercise 4.5])

Finde alle singulären Punkte auf den zu den Funktionen korrespondierenden Ebenenkurven:

- $f: (x, y) \mapsto 4x^2y^2 - (x^2 + y^2)^3$

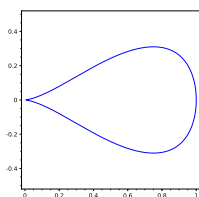


Abbildung 2.4:
Cusp

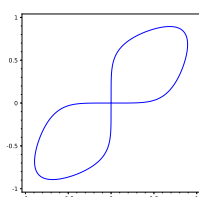


Abbildung 2.5:
Node

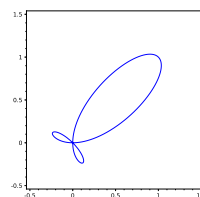


Abbildung 2.6:
Triplepoint

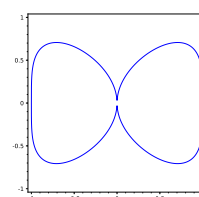


Abbildung 2.7:
Tachnode

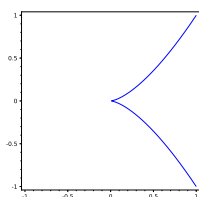


Abbildung 2.8:
 \mathcal{C}_f

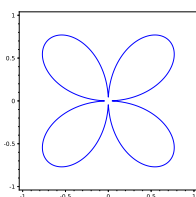


Abbildung 2.9:
 \mathcal{C}_g

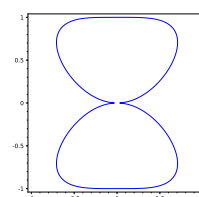


Abbildung 2.10:
 \mathcal{C}_h

- $g: (x, y) \mapsto -x^4 - y^4 + y^2$
- $h: ??$

Da die Kurven sich selber nur in $(0,0)$ schneiden, ist dieser Punkt jeweils ein guter Kandidat ist

2.20 Definition (absolut irreduzibel[65])

Curve \mathcal{C}_f heißt absolut irreduzibel, wenn $f(x, y) \in k[x, y]$ eine nichtsinguläre Kurve \mathcal{C}_f und $f = gh$ für $g, h \in \bar{k}[x, y]$, wobei \bar{k} der algebraische Abschluss von k ist, dann ist entweder $g \in \bar{k}$ oder $h \in \bar{k}$.

2.21 Definition (Elliptische Kurve[65])

Eine elliptische Kurve hat eine Gleichung der Form $y^2 = f(x)$, wobei $f(x)$ ein kubisches Polynom in x ohne mehrfache Nullstellen ist.

2.22 Definition (Genus [65])

Sei $f(x, y) \in k[x, y]$ ein Polynom vom Grad d , so dass $\hat{\mathcal{C}}_f$ nichtsingulär ist, dann ist das Geschlecht von \mathcal{C}_f (oder von $\hat{\mathcal{C}}_f$) definiert als

$$g := \frac{(d-1)(d-2)}{2}.$$

2.23 Definition (rationale Punkte [5.4,5.1])

Sei k ein Körper und sei \mathcal{C} die projektive Kurve, die von $F = 0$ definiert wird, wobei $F = F(X, Y, Z) \in [X, Y, Z]$ ein homogenes Polynom ist. Für einen beliebigen Körper K , der k enthält, ist ein K -rationaler Punkt auf \mathcal{C} als Punkt $(X_0 : Y_0 : Z_0) \in \mathbb{P}^2(K)$ mit $F(X_0, Y_0, Z_0) = 0$ definiert.

Die Menge aller K -rationalen Punkte auf \mathcal{C} schreiben wir als $\mathcal{C}(K)$.

Elemente von $\mathcal{C}(k)$ werden Punkte vom Grad Eins oder einfach rationale Punkte genannt. Ein Punkt vom Grad n auf \mathcal{C} über \mathbb{F}_q ist eine Menge $\mathcal{P} = \{P_0, \dots, P_{n-1}\}$ von n paarweise verschiedenen Punkten auf $\mathcal{C}(\mathbb{F}_{q^n})$, so dass $P_i = \sigma_{q,n}^i(P_0)$ für $i = 1, \dots, n-1$.

Sei $\mathbb{F}_q \subset \mathbb{F}_{q^n}$, dann ist der Frobenius Automorphismus $\sigma_{q,n} : \mathbb{F}_{q^n} \rightarrow \mathbb{F}_{q^n}$ definiert durch $\sigma_{q,n}(\alpha) = \alpha^q$.

2.24 Definition (Divisor[65, S. 5.7])

Sei \mathcal{C} eine Kurve, die über \mathbb{F}_q definiert ist.

Ein Divisor D auf \mathcal{C} über \mathbb{F}_q ist ein Element einer freien abelschen Gruppe auf der Menge der Punkte (von beliebigem Grad) auf \mathcal{C} über \mathbb{F}_q .

Unter dieser Voraussetzung ist jeder Divisor von der Form $D = \sum n_Q Q$ mit n_Q ganzzahlig und jedes Q ist ein Punkt (von beliebigem Grad) auf \mathcal{C} .

Wenn $n_Q \geq 0 \forall n_Q$, nennen wir D effektiv und schreiben $D \geq 0$. Wir definieren den Grad eines Divisors $D = \sum n_Q Q$ als $\deg D = \sum n_Q \deg Q$. Schliesslich definieren wir den Support eines Divisors $D = \sum n_Q Q$ als $\text{supp } D = \{Q | n_Q \neq 0\}$.

2.25 Definition (Hermitsche Kurve [65, S. 5.6])

Sei \mathcal{C} die projektive Ebenenkurve, die von $Y^q Z + YZ^q = X^{q+1}$ über dem Körper \mathbb{F}_{q^2} definiert ist, dann nennt man \mathcal{C} hermitsche Kurve

2.26 Definition (Raum der rationalen Funktionen vom Grad r [65])

$\forall r \geq 0 \quad L_r := \{f \in \mathbb{F}_q[x] \mid \deg(f) \leq r\} \cup \{0\}$

2.27 Definition (Raum der Linearen Funktionen [65, Definition 1.8, Definition 5.12])

$$\mathcal{L}_{k-1} = \{f \in \mathbb{F}_q[x] \mid \deg f \leq k-1\} \cup \{0\} \quad (2.11)$$

2.2.15 Lemma ([65, Exercise 6.2])

Wenn $P_\infty = (1 : 0)$ und $D = (k - 1) P_\infty$ gesetzt wird, dann gilt

$$\mathcal{L}(D) = \mathcal{L}_{k-1}, \quad (2.12)$$

wobei wir jedes Polynom $f(x) \in \mathbb{F}_q[x]$ von Grad d mit seiner Homogenisierung $Y^d f(X/Y) \in \mathbb{F}_q[X, Y]$ identifizieren.

Beweis. $\mathcal{L}(D) \subseteq \mathcal{L}_{k-1}$:
 $\mathcal{L}_{k-1} \subseteq \mathcal{L}(D)$:

□

2.28 Definition

$$\mathcal{L}(D) := \{f \in \mathbb{F}_q(\mathcal{C}) \mid |\operatorname{div}(f) + D| \geq 0\} \cup \{0\} \quad (2.13)$$

Und zum besseren Verständnis das folgende Lemma

2.2.16 Lemma ([65, Exercise 5.13])

Sei D ein Divisor auf einer nichtsingulären projektiven ebenen Kurve \mathcal{C} über \mathbb{F}_q , dann gilt:

- Wenn $\deg D \leq 0$, dann $\mathcal{L}(D) = \{0\}$
- $\mathbb{F} \subseteq \mathcal{L}(D)$ genau dann, wenn $D \geq 0$.

Kapitel 3

Codes

Im folgenden Kapitel werden einige Definitionen für Codes aufgestellt und deren Eigenschaften betrachtet.

3.1 Darstellung

Jetzt werden zuerst Darstellungsformen von Codes definiert und danach die Definition von MacKay von Hamming Codes auf lineare Codes erweitert.

3.1 Definition (Erzeugermatrix [67, S. 9] [81, S. 52])

Sei \mathcal{C} ein linearer Code, dann heißt eine Matrix G Erzeugermatrix von \mathcal{C} , wenn $\forall c \exists w \in A^k$, sodass gilt:

$$c = wG$$

Die Standardform einer Erzeugermatrix ist

$$G = [\mathcal{I}_k \mathcal{P}] \tag{3.1}$$

wobei \mathcal{I}_k die $k \times k$ Einheitsmatrix und \mathcal{P} eine $k \times (n - k)$ Matrix ist.

3.1.1 Beispiel (Erzeugermatrix)

Die Erzeugermatrix eines ?? ist

$$G = ??$$

Das folgende Lemma stammt von MacWilliams und Sloane und ist dort als Übungsaufgabe in Kapitel 9 Problem 31 gestellt.

3.1.2 Lemma ([59][Chapter 9 Prolem 31])

Sei C eine invertierbare $r \times r$ Matrix über \mathbb{F}_{q^m} und sei $\bar{\mathcal{H}} = C\mathcal{H}$, dann gilt $\mathcal{C}_{\bar{\mathcal{H}}} = \mathcal{C}_{\mathcal{H}}$.

Beweis. Ein Wort w ist in $\mathcal{C}_{\mathcal{H}}$ genau dann, wenn $\mathcal{H}w^{tr} = 0$.

Dann folgen diese Äquivalenzen:

$$\begin{aligned} \mathcal{H}w^{tr} &= 0 \\ \stackrel{C \text{ invertierbar}}{\Leftrightarrow} C\mathcal{H}w^{tr} &= C0 = 0 \end{aligned}$$

Also ist ein Wort, genau dann in $\mathcal{C}_{\bar{\mathcal{H}}}$, wenn es auch in $\mathcal{C}_{\mathcal{H}}$ ist. □

3.2 Definition (Prüfmatrix [67, S. 12])

Die Prüfmatrix zu den Annahmen aus 3.1 mit der Wahl aus (3.1) ist eine Matrix

$$H = \begin{bmatrix} -\mathcal{P}^T \mathcal{I}_{n-k} \end{bmatrix} \quad (3.2)$$

3.1.3 Beispiel (Prüfmatrix)

Die Prüfmatrix eines ?? ist

$$G = ?? \text{ mit } \mathcal{P} = ?? \#$$

3.1.4 Lemma

Die eben in 3.2 definierte Matrix H erfüllt für alle

$$c = wG$$

die Gleichung

$$Hc^T = 0.$$

Beweis. Seien H und c wie oben definiert, dann gilt

$$\begin{aligned} Hc^T &= H(wG)^T \\ &= HG^T w^T \end{aligned}$$

Und durch Einsetzen der Definitionen

$$\begin{aligned} &= \begin{bmatrix} -\mathcal{P}^T \mathcal{I}_{n-k} \end{bmatrix} [\mathcal{I}_k \mathcal{P}]^T w \\ &= \begin{bmatrix} -\mathcal{P}^T + \mathcal{P}^T \end{bmatrix} w \\ &= \vec{0} w \\ &= 0 \end{aligned}$$

□

3.2 Dualer Code

3.3 Definition (Standardskalarprodukt [1.6])

Auf dem Vektorraum \mathbb{F}_q^n ist das Standardskalarprodukt als

$$(\vec{v}, \vec{w}) = \sum_{i=0}^{n-1} \vec{v}_i \vec{w}_i \in \mathbb{F}_q$$

definiert.

Zwei Vektoren $\vec{v}, \vec{w} \in \mathbb{F}_q^n$ heißen orthogonal, wenn $(\vec{v}, \vec{w}) = 0$.

Dies ist äquivalent zu $\vec{v}^T \vec{w} = 0$.

Mit dieser Definition erschließt sich die folgende Klasse von Codes:

3.4 Definition (Dualer Code [79][1.6.1])

Sei \mathcal{C} ein (n, k) -Code, dann ist der dualer Code definiert als

$$\mathcal{C}^\perp = \{ \bar{c} \in \mathbb{F}_q^n \mid (c, \bar{c}) = 0 \forall c \in \mathcal{C} \},$$

wobei (c, \bar{c})

das eben in 3.3 definierte Standardskalarprodukt bezeichnet.

Ein Code mit $\mathcal{C} = \mathcal{C}^\perp$ heisst selbstdual.

Dass diese Definition einen Code ergibt, sehen wir durch folgendes Theorem 3.2.1

3.2.1 Theorem (Eigenschaften von dualen Codes [79, S. 1.6.2])

Sei \mathcal{C} ein (n, k) -Code mit Erzeugermatrix G und Prüfmatrix H .

1. Die Matrix H ist eine Erzeugermatrix und G ist eine Prüfmatrix des dualen Code \mathcal{C}^\perp .
2. Der duale Code ist ein $(n, n - k)$ -Code.
3. Es gilt $(\mathcal{C}^\perp)^\perp = \mathcal{C}$.

Beweis. Da jede Zeile von G zu einem Codewort c_i via

$$\vec{e}_i \cdot G = (0, \dots, 0, 1(i\text{-te}), 0, \dots, 0)^T \cdot G = c_i$$

korrespondiert, gilt $G\vec{v} = \vec{0}$, da für jede einzelne Zeile ?? gilt.

Wenn jetzt $G \cdot \vec{v}^T = 0$, dann gibt es für alle $\mathbb{F}_q^n \ni c \exists \vec{m} \in \mathbb{F}_q : \vec{m} \cdot G = c$ und mit $\vec{m} = \sum a_i \vec{e}_i$ folgt

$$(c, \vec{v}) \stackrel{??}{=} \sum c_i v_i = c \vec{v}^T \stackrel{??}{=} \vec{m} G \vec{v}^T \stackrel{\text{Annahme}}{=} \vec{m} \vec{0} = 0.$$

Aus

$$(G \cdot H^T)^T = H \cdot G^T = (0)^T = (0)$$

folgt, dass G eine Prüfmatrix von \mathcal{C}^\perp ist, somit gilt ??.

Da $H \in \mathcal{M}_{n-k,n}(\mathbb{F}_q)$ und H Erzeugermatrix für \mathcal{C}^\perp ist, folgt 2.

Aus der Definition eines dualen Codes folgt, dass $\mathcal{C} \subset (\mathcal{C}^\perp)^\perp$, mit der folgenden Überlegung:

Sei \bar{c} ein beliebiges Element aus \mathcal{C}^\perp , dann gilt nach der Definition von Dualität, dass für alle $c \in \mathcal{C}$ $(c, \bar{c}) = 0$ gilt. Da $G^\perp \in \mathcal{M}_{k,n}$ und alle Codewörter von $^\perp$ auch in \mathcal{C} liegen folgt 2.

□

3.5 Definition (Hamming Code [54])

Der binäre Hamming Code ist ein (n, k) -Code in $\mathbb{F}[2^n]$, der wie folgt definiert ist

Um Fehler zu erkennen, wird jeweils über alle Stellen in deren binärer Darstellung an der i -ten Stelle eine 1 steht summiert und geprüft, ob die Summe gerade oder ungerade ist; im ersten Fall ist die Prüfziffer gleich 0 und im zweiten gleich 1. Dies wird für alle $i = 1 \dots k$ wiederholt und $\sum p_i 2^i$ gibt die Fehlerstelle an. Diese Information muss auch in dem übermittelten Codewort encodiert werden, um die gewünschte Redundanz i zu erhalten, indem versichert wird, dass bei einem korrekten Codewort die $p_i = 0 \forall i$.

Damit die p_i unabhängig voneinander sind, wählen wir jeweils die 2^{i-1} -te Stelle im Codewort.

3.3 Reed-Solomon Code

Der von Reed und Solomon in 1960 entwickelte Code hat den Vorteil, dass ein Symbol durch eine Folge von Zeichen klassifiziert ist; dies ermöglicht eine bessere Burst-Correction und daher werden sie heute z. B. bei CDs eingesetzt.

Der eigentliche Code ist wie folgt definiert:

3.6 Definition (Reed-Solomon Codes [65, Definition 1.8])

Seien $\alpha_1, \dots, \alpha_{q-1}$ die Elemente von \mathbb{F}_q^\times und $1 \leq k \leq q-1$, dann ist der Reed-Solomon Code \mathcal{C}_{RS} definiert als

$$\mathcal{C}_{RS}(k, q) = \{(f(\alpha_1), \dots, f(\alpha_{q-1})) \mid f \in \mathcal{L}_{k-1}\}. \quad (3.3)$$

3.3.1 Lemma

Ein Reed-Solomon Code ist ein linearer Code.

Beweis. Da \mathcal{L}_{k-1} linear ist, ist für alle $f(\alpha_i)$ die Linearität gegeben und komponentenweise Linearität reicht für den Code aus. \square

3.3.2 Lemma

(Parameter eines Reed-Solomon Codes [65, Definition 1.8])

Ein $\mathcal{C}_{RS}(k, q)$ ist ein (n, k, d) -Code mit $n = q-1$ und $d = n - k + 1$.

Beweis. Mit der folgenden Rechnung ist die Länge des Codes gezeigt: $\mathcal{C}_{RS}(k, q) \subset \mathbb{F}^{q-1} = \mathbb{F}^n$

Da der Code das Bild einer linearen Transformation ist, kann die Dimension des Codes höchstens die Dimension des Urbilds, also k , sein. \square

Das folgende Beispiel ist aus [62].

1 Notation. Wir bedienen uns kurz der Notation und Konzepte von [60], passen aber einige Symbole an die in diesem Dokument bereits genutzten an:

- J ist die Anzahl der Bits pro Symbol (für einen anderen Körper als \mathbb{F}_2 funktioniert diese Definition bis auf das Wort Bits)
- n die Anzahl der Symbole pro Codewort
- k die Anzahl der Symbole, die Daten repräsentieren
- \mathcal{I} die Tiefe der Verwebung, es gilt also für eine Kette von $n \cdot \mathcal{I}$ Symbolen, die aus \mathcal{I} Codewörtern besteht, sind aufeinander folgende Symbole eines Codeworts durch \mathcal{I} Symbole anderer Codewörter getrennt

Mit der letzten Notation kann aus einem (n, k) -Code, der d Fehler korrigieren kann, ein $(\mathcal{I}n, \mathcal{I}k)$ -Code gebildet werden, der bis zu $\mathcal{I} \cdot d$ Fehler korrigieren kann, sofern sie als „Burst“, also konsekutiv auftreten.

3.3.3 Lemma

Sei \mathcal{C} ein beliebiger linearer Code, der d Fehler korrigieren kann, dann gibt es einen $(\mathcal{I} \cdot n, \mathcal{I} \cdot k)$ -Code, der $\mathcal{I}d$ konsekutive Fehler korrigieren kann.

Beweis. Sei \mathcal{C} ein Code wie oben, dann kann ein beliebiges Codewort als $(u_1 \dots u_J)$ mit $u_i \in \mathbb{F}$ geschrieben werden. Um Codewörter voneinander zu unterscheiden, wird das i -te Codewort aus \mathcal{C} wie folgt ergänzt: $(^i u_1 \dots ^i u_J)$ Ein Codewort aus $\bar{\mathcal{C}}$ wird dann geschrieben als $^1 u_1 \ ^2 u_1 \dots ^{??} u_1 \dots ^1 u_2 \ ^2 u_2 \dots ^{??-1} u_n \ ^{??} u_n$. Der so definierte Code hat $n^{??}$ Zeichen und durch „sortieren“ sieht man, dass es äquivalent zur Übertragung von $^{??}$ ist, somit also $k^{??}$ Datensymbole übertragen werden.

Wenn bei der Übertragung l konsekutive Fehler auftreten, dann ist jedes Codewort an höchstens $\lceil l/^{??} \rceil$ betroffen.

Für $l \leq d^{??}$ sind es somit $\lceil l/^{??} \rceil \leq \lceil d^{????} \rceil \leq d$ Fehler pro Codewort und diese können korrigiert werden.

□

3.3.4 Beispiel ([62])

Der Voyager Reed-Solomon Code ist ein $(255, 223)$ -Code über $\mathbb{F}[2^8]$ mit Verweigungstiefe von 4.

Das Erzeugerpolynom dieses Codes ist

$$\text{Erzeugerpolynom} = (x - \alpha)(x - \alpha^2) \dots (x - \alpha^{32})$$

Wobei α eine Nullstelle des primitiven Polynoms $x^8 + x^4 + x^3 + x^2 + 1$ von $\mathbb{F}[2^8]$ ist, wie in [60, Abschnitt V BZW. Table 4] dargestellt wird.

Die Definition lässt sich wie folgt verallgemeinern.

3.7 Definition (generalisierte Reed-Solomon Codes)

Sei $\mathcal{C}_{RS}(n, k, d)$ ein Reed-Solomon Code über \mathbb{F}_{q^m} und \vec{g} ein Vektor aus \mathbb{F}_{q^m} , dessen Einträge $\neq 0$ sind und der Template genannt wird.

Der generalisierte Reed-Solomon Code $\mathcal{C}_{GRS}(\vec{g})$ ist ein Code, der durch komponentenweise Multiplikation von \vec{g} mit den Codewörtern des Reed-Solomon Codes definiert ist. Also

$$\mathcal{C}_{GRS}(\vec{g}) = \{c \mid |c = \vec{g} c', c' \in \mathcal{C}_{RS}\},$$

wobei die i -te Komponente von $\vec{g} c'$ durch $g_i c'_i$ definiert ist.

3.3.5 Lemma (Parameter eines generalisierten Code)

Der in Definition 3.7 beschriebene Code ist wieder ein (n, k, d) -Code.

Beweis. Der Code ist ein linearer Code, da die Definition 2.3 komponentenweise ist. Jedes Codewort $c \in \mathcal{C}_{RS}$ ist auch in \mathcal{C}_{GRS} und $c' \vec{g} = c, c' \vec{g} = \bar{c} \in \mathcal{C}_{GRS}$ gilt $c \neq \bar{c} \Rightarrow c' \neq \bar{c}'$. Also ist die Anzahl n der Codewörter die gleiche.

Aus der gleichen Überlegung folgt, dass die Distanz $d(c, \bar{c}) = d(c', \bar{c}')$ und somit auch die d_{\min} .

Da jedes Element der Basis von \mathcal{C}_{RS} eine Entsprechung in \mathcal{C}_{GRS} hat, bleibt auch die Dimension k erhalten.

Damit ist \mathcal{C}_{GRS} ein (n, k, d) -Code. □

Aus Definition 3.7 lässt sich die folgende Klasse von Codes definieren.

3.8 Definition (Alternant Code [72, >Abschnitt 2.12])

Sei $\mathcal{C}_{GRS}(\vec{g})$

ein generalisierte Reed-Solomon Code wie in Definition 3.7, dann ist der Alternant Code \mathcal{C}_A (??) definiert als

$$\begin{aligned} \mathcal{C}_A(??) &= \mathcal{C}_{GRS}(??) \cap \mathbb{F}_q^n \\ &= \{c \mid c_i \in \mathbb{F}_q : c = \vec{g} c', c' \in \mathcal{C}_{RS}\}. \end{aligned}$$

Da alle $g_i \neq 0$, kann \vec{g}^{-1} mit Komponenten g_i^{-1} geschrieben werden. Damit ist

$$\mathcal{C}_A(\vec{g}) = \{c \mid c_i \in \mathbb{F}_q : \vec{g}^{-1} c = c', c' \in \mathcal{C}_{RS}\}.$$

3.3.6 Theorem ([72, Theorem 2.12.1])

Sei \mathcal{C}_{GRS} ein (n, K, D) -Code generalisierter Code über \mathbb{F}_{q^m} und \mathcal{C}_A ein (n, k, d) Untercode von \mathcal{C}_{GRS} über \mathbb{F}_q , dann gilt $D \leq d$ und $n - (d - 1)m \leq k \leq K$.

Beweis. Die Ungleichung $D \leq d$ folgt aus folgender Überlegung:

Angenommen $\exists c, \bar{c} \in \mathcal{C}_A$, so dass $d(c, \bar{c}) < D$, dann sind $c, \bar{c} \in \mathcal{C}_{GRS}$ und somit $D \leq d(c, \bar{c}) < D$ ✖.

Damit folgt $D + K \leq d + K$.

Da der Reed-Solomon Code den Singleton Bound 2.2.7 mit Gleichheit erfüllt ?? folgt $D + K = n + 1$ und für den Alternant Code $d + k \leq n + 1$ und somit $d + k \leq D + K$. Dann folgt $d + k \leq d + K$, also $k \leq K$.

Bleibt noch zu zeigen, dass $n - (d - 1)m \leq k$ gilt.

Der generalisierte Reed-Solomon Code ist ein linearer Code mit $n - K$ Prüfgleichungen über \mathbb{F}_{q^m} . Jede Prüfgleichung lässt sich als m Prüfgleichungen über \mathbb{F}_q darstellen. Da

diese $m(n - K)$ Prüfgleichungen nicht linearunabhängig sein müssen, folgt $(n - k) \leq m(n - K)$.

Da der Reed-Solomon Code den Singleton Bound 2.2.7 mit Gleichheit erfüllt, kann $D - 1 = n - K$ geschrieben werden und mit Einsetzen erhält man $(n - k) \leq m(D - 1)$ und durch die Ergebnisse von oben $n \leq k + m(D - 1) \leq k + m(d - 1)$. \square

3.4 Goppa Code

Goppa Codes können auf verschiedenen Wegen klassifiziert werden

Risse listet in [135] folgende Methoden, um Goppa Codes zu definieren:

1. Als lineare Codes die eine Relation bzgl einem Goppa Polynom erfüllen, wie in 3.9
2. Als alternant generalisierte Reed-Solomon Codes, wie in 3.10
3. Als Fourier-Transformation über endlichen Feldern, wie in 3.13
4. Als generalisierte Algebraische Codes über Kurven, wie in 3.14

3.9 Definition (Goppa Code [70, Definiton 6.10] [74, S. 2.70] [59, S. 12.3])

Sei $L = \{\alpha_1, \alpha_2, \dots, \alpha_n\}$, wobei $\alpha_i \in \mathbb{F}(q^m)$.

Sei $g(x) \in \mathbb{F}_{q^m}[x]$ das Goppa-Polynom, wobei $\forall \alpha \in L \ g(\alpha_i) \neq 0$.

Zu einem Vektor $\vec{a} = (a_1, a_2, \dots, a_n)$ mit Elementen aus $\mathbb{F}(q)$ wird die rationale Funktion

$$R_a(x) = \sum_{i=1}^n \frac{a_i}{x - \alpha_i}$$

assoziiert.

Dann besteht der Goppa Code $\mathcal{C}(L, g)$ aus allen Vektoren $a \in \mathbb{F}_q^n$ sodass

$$R_a(x) \equiv 0 \pmod{g(x)}. \quad (3.4)$$

Wenn $g(x) \equiv 1$ ist, dann bezeichnen wir $\mathcal{C}(L, g)$ als $\mathcal{C}(L)$.

Eine weitere Definition stammt von Roth, der die Definition von Goppa als Sonderfall der Definition 3.7 auffasst.

3.10 Definition (Goppa Code als Alternant Code [77, Section 5.6])

Goppa Codes sind ein $[n, k, d]$ \mathcal{C}_{GRS} über \mathbb{F} mit Spaltenmultiplikatoren die zu

den $??$ über

$$v_j = \frac{1}{\alpha_j} \forall 1 \leq j \leq n$$

in Verbindung stehen, wobei $f(x) \in \mathbb{F}[x]$ und $\deg f(x)$, so dass $f(\alpha_j) \neq 0 \forall 1 \leq j \leq n$

Blahut definiert die Goppa Codes aus den Alternant Codes wie folgt:

Sei \vec{g} das Template eines generalisierten Code, dann gibt es ein Template h , so dass $(1/n) g_i h_i = 1$. Damit ist $h_i = n g_i^{-1}$.

3.11 Definition (Faltungseigenschaft [72, Section 2.3])

Sei $C(x) = \sum_{j=0}^{n-1} V_j x^j$, dann gilt

$$v_i = \frac{1}{n} \sum_{j=1}^{n-1} V_j \omega^{-i \cdot j} = \frac{1}{n} C(\omega^{-i})$$

Die beiden Templates haben jeweils eine Fourier-Transformation G bzw. H , die sich als $G(x)$ bzw. $H(x)$ darstellen lassen.

Durch die Faltungseigenschaft der Fourier-Transformation erhält man aus $(1/n) g_i h_i = 1$ den Ausdruck $(1/n) G(x) H(x)$ der $g_i h_i = \frac{1}{n} G(\omega^{-i}) \frac{1}{n} H(\omega^{-i})$ erfüllt und $G(\omega^{-i}) = n g_i \neq 0$ und damit keine Nullstellen auf \mathbb{F}_{q^m} hat.

Mit dem erweiterten Euklidischen Algorithmus findet man Polynome $F(x)$ und $E(x)$, sodass

$$G(x) F(x) + (x^n - 1) G(x) = 1$$

und damit

$$G(x) F(x) = 1 \pmod{x^n - 1}.$$

3.12 Definition

Alternant Code [72, Section 2.13]

Sei $\omega \in \mathbb{F}_{q^m}$ von Ordnung n und $H(\omega^{-i}) \neq 0$ für $i = 0, \dots, n-1$ und wähle $j_0, n \in \mathbb{N}$. Der Alternant Code \mathcal{C}_A ist die Menge aller \vec{c} , deren Transformationen C zwei Bedingungen erfüllen:

$$C_j^q = C_{((qj))} \quad (3.5) \quad \text{und} \quad \sum_{k=0}^{n-1} H_{((j-k))} C_k = 0 \quad j = j_0, \dots, j_0 + 2t - 1. \quad d = e + f \quad (3.6)$$

Die Bedingung (3.5) legt fest, dass $\vec{c} \in \mathbb{F}_q^n$.

Die Bedingung (3.6) lässt sich schreiben als $C'(x) = H(x) C(x)$, BZW. als $C(x) = H(x) C'(x)$

Mit all dieser Sprache kann die folgende Definition getroffen werden.

3.13 Definition ([72, S. 2.13.1])

Der Goppa Code mit Distanz d ist ein Alternant Code mit Distanz d , wobei das $g_i = (1/m) G(\omega^{-i})$ und $\deg G = d - 1$

Diese drei Definitionen hängen durch die Konstruktionen wie beschrieben zusammen.

3.4.1 Theorem ([59, S. 2.13.2])

Für einen Goppa Code mit Goppa-Polynom g und definierender Menge $j_0, \dots, j_0 + d - 2$ ist \vec{c} genau dann ein Codewort, wenn

$$\sum_{i=1}^n c_i \frac{\omega^{ij}}{g(\omega^{-i})} = 0 \quad \forall j = j_0, \dots, j_0 + d - 2 \quad (3.7)$$

3.14 Definition (Algebraisch-geometrischer BZW. Goppa Code [65, Definition 6.3][73])

Sei \mathcal{C} eine glatte (also nichtsinguläre) projektive Kurve über \mathbb{F}_q . D eine Divisor von \mathcal{C} (z.B. $D = rP_\infty$, wobei P_∞ der Punkt in der Unendlichkeit von \mathcal{C} und r eine natürliche Zahl ist).

Sei $\mathcal{P} = \{P_1, \dots, P_n\}$ eine Menge von verschiedenen Punkten auf \mathcal{C} , die über \mathbb{F}_q definiert sind und nicht aus dem Erzeugendensystem von D stammen.

Eine Menge von Funktionen auf \mathcal{C} , die Polstellen nur in P_∞ mit Ordnung der Polstellen in ∞ höchstens r . Dann ist der algebraisch geometrische Code zu \mathcal{X} , \mathcal{P} und D definiert als

$$\mathcal{C}(\mathcal{C}, \mathcal{P}, D) := \{(f(P_1), \dots, f(P_n),) | f \in \mathcal{L}(D)\} \subset \mathbb{F}_q^n.$$

Anders gesagt, der algebraisch geometrische Code $\mathcal{C}(\mathcal{X}, \mathcal{P}, D)$ ist das Bild der Evaluierungsabbildung

$$\begin{aligned} ev : \mathcal{L}(D) &\rightarrow \mathbb{F}_q^n \\ f &\mapsto (f(P_1), \dots, f(P_n),) \end{aligned}$$

Die Reed-Solomon Codes aus ?? sind ein Sonderfall mit $\mathcal{C} = \mathcal{C}_{\{y=0\}}$ [65].

Mit diesem Gedanken im Hinterkopf ist verständlich, dass alle vier Definitionsarten vom Anfang des Abschnitts 3.4 für Goppa Codes zusammenhängen.

Sei $P_i = (\alpha_i : 1)$, $\mathcal{Q} = (1 : 0)$ und $D = P_1 + \dots + P_n$

Wenn wir E als Divisor der Nullstellen auf g auf der Projektive Gerade definieren, dann erhalten wir $\mathcal{C}(L, g) = \mathcal{C}^\perp(D, E - \mathcal{Q})$ und

$$c \in \mathcal{C}(L, g) \iff \sum \frac{c_i}{X - \alpha_i} dX \in \Omega(E - \mathcal{Q}_D).$$

Nach [59] und ?? kann die Prüfmatrix eines Goppa Code wie folgt bestimmt werden: Da der Term $X - \alpha_i$ das Polynom g nicht teilt, hat er ein Inverses in $\mathbb{F}_{q^m}[X]/g$. Dieses Inverse

$$(X - \alpha_i)^{-1} = -\frac{g(X) - g(\alpha_i)}{X - \alpha_i} g(\alpha_i)^{-1} \quad (3.8)$$

ist wirklich das Inverse von 3.4, da

$$\begin{aligned} (X - \alpha_i) \cdots (X - \alpha_i)^{-1} &\stackrel{3.8}{=} -(X - \alpha_i) \frac{g(X) - g(\alpha_i)}{X - \alpha_i} g(\alpha_i)^{-1} \\ &= -\frac{X - \alpha_i}{X - \alpha_i} \frac{g(X) - g(\alpha_i)}{g(\alpha_i)} \\ &= -1 \cdot \left(\frac{g(X)}{g(\alpha_i)} - \frac{g(\alpha_i)}{g(\alpha_i)} \right) \\ &= -1 \cdot \left(\frac{g(X)}{g(\alpha_i)} - 1 \right) \\ &= -g(X) g(\alpha_i)^{-1} + 1 \\ &\stackrel{g(\alpha_i)^{-1}}{\equiv} 1 \pmod{g(X)} \end{aligned}$$

Also ist $a \in \mathcal{C}(L, g)$ genau dann, wenn

$$\sum_{i=1}^n a_i \frac{g(X) - g(\alpha_i)}{X - \alpha_i} g(\alpha_i)^{-1} = 0 \quad (3.9)$$

als Polynom.

Wenn $g(X) = \sum_{i=0}^r g_i X^i$ mit $g_i \in \mathbb{F}_{q^m}$ und $g_r \neq 0$, dann

$$\begin{aligned} \frac{g(X) - g(\alpha_i)}{X - \alpha_i} &= g_r (z^{r-1} + z^{r-2}\alpha_i + \dots + z\alpha_i^{r-1}) \\ &\quad + g_{r-1} (z^{r-2} + z^{r-2}\alpha_i + \dots + z\alpha_i^{r-2}) \\ &\quad + \dots \\ &\quad + g_2 (z + \alpha_i) \\ &\quad + g_1 \end{aligned}$$

Durch Nullsetzen der Koeffizienten der $z^{r-1}, z^{r-2}, \dots, 1$ in 3.9 sehen wir, dass $a \in$

$\mathcal{C}(L, g)$ genau dann, wenn $Ha^{tr} = 0$, wobei

$$H = \begin{bmatrix} g_r g(\alpha_1)^{-1} & \dots & g_r g(\alpha_n)^{-1} \\ (g_{r-1} + \alpha_1 g_r) g(\alpha_1)^{-1} & \dots & (g_{r-1} + \alpha_n g_r) g(\alpha_n)^{-1} \\ \dots & \dots & \dots \\ (g_1 + \alpha_1 g_2 + \dots + \alpha_1^{r-1}) g(\alpha_1)^{-1} & \dots & (g_1 + \alpha_n g_2 + \dots + \alpha_n^{r-1}) g(\alpha_n)^{-1} \end{bmatrix}$$

(3.10)

$$= \begin{bmatrix} g_r g(\alpha_1)^{-1} & \dots & g_r g(\alpha_n)^{-1} \\ (g_{r-1} + \alpha_1 g_r) g(\alpha_1)^{-1} & \dots & (g_{r-1} + \alpha_n g_r) g(\alpha_n)^{-1} \\ \dots & \dots & \dots \\ (g_1 + \alpha_1 g_2 + \dots + \alpha_1^{r-1}) g(\alpha_1)^{-1} & \dots & (g_1 + \alpha_n g_2 + \dots + \alpha_n^{r-1}) g(\alpha_n)^{-1} \\ g(\alpha_1)^{-1} & & 0 \\ & g(\alpha_2)^{-1} & \\ & & \ddots \\ & & g(\alpha_n)^{-1} \end{bmatrix}$$

(3.11)

$$= \begin{bmatrix} g_r & 0 & 0 & \dots & 0 \\ g_{r-1} & g_r & 0 & \dots & 0 \\ g_{r-2} & g_{r-1} & g_r & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ g_1 & g_2 & g_3 & \dots & g_r \\ 1 & 1 & \dots & 1 & \\ \alpha_1 & \alpha_2 & \dots & \alpha_n & \\ \alpha_1^2 & \alpha_2^2 & \dots & \alpha_n^2 & \\ \dots & \dots & \dots & \dots & \dots \\ \alpha_1^{r-1} & \alpha_2^{r-1} & \dots & \alpha_n^{r-1} & \\ g(\alpha_1)^{-1} & & & & 0 \\ & g(\alpha_2)^{-1} & & & \\ & & \ddots & & \\ & & & g(\alpha_n)^{-1} & \end{bmatrix}$$

(3.12)

$$= \mathcal{C}\mathcal{X}\mathcal{Y}$$

(3.13)

Nach Lemma ??

$$\mathcal{H}' = \mathcal{X}\mathcal{Y} \quad (3.14)$$

$$= \begin{bmatrix} g(\alpha_1)^{-1} & \dots & g(\alpha_n)^{-1} \\ \alpha_1 g(\alpha_1)^{-1} & \dots & \alpha_n g(\alpha_n)^{-1} \\ \dots & \dots & \dots \\ \alpha_1^{r-1} g(\alpha_1)^{-1} & \dots & \alpha_n^{r-1} g(\alpha_n)^{-1} \end{bmatrix} \quad (3.15)$$

eine Prüfmatrix von $\mathcal{C}(L, g)$.

Dies ist nach [64] ein dualer Code zu ??.

3.4.2 Theorem (Parameter eines Goppa Code [65, S. 6.4] [73, Theorem 5.2])

Sei \mathcal{C} eine nicht singuläre projektive Ebenenkurve von Geschlecht g , die über \mathbb{F}_q definiert ist und sei $\mathcal{P} \subset \mathcal{C}(\mathbb{F}_q)$ eine Menge von n verschiedenen \mathbb{F}_q -rationalen Punkten auf \mathcal{C} und sei D ein Divisor auf \mathcal{C} , der $2g - 2 < \deg D < n$ erfüllt.

Dann ist der algebraisch geometrische Code $\mathcal{C}(\mathcal{C}, \mathcal{P}, D)$ ein linearer Code von Länge n , Dimension $k := \deg D + 1 - g$, und Minimaldistanz d , wobei $d \geq n - \deg D$.

Beweis.

$\mathcal{L}(D)$ ist ein Vektorraum über \mathbb{F}_q und ev ist eine lineare Transformation und erhält somit die Vektorraum-Eigenschaften. Daher ist $\mathcal{C}(\mathcal{X}, \mathcal{P}, D) \stackrel{??}{=} \text{ev}(\mathcal{L}(D))$ immer noch ein linearer Untervektorraum von \mathbb{F}_q^n und somit nach ?? ein linearer Code. Da das Alphabet $A = \mathbb{F}_q$, folgt mit der Definition ??, dass die Länge unseres Codes $n = \#\mathcal{P}$ ist.

Nachdem die Länge des Codes gezeigt ist, wird jetzt die Dimension des Codes betrachtet.

Das folgende Lemma wird von Walker als Übung gestellt.

3.4.3 Lemma ([65, Exercise A.23.a])

Seien A, B Vektorräume (oder Gruppen oder Ringe), und $\Phi : A \rightarrow B$ ein Homomorphismus. Dann gilt $\ker(\Phi) = \{x \in A \mid \Phi(x) = 0 \in B\}$.

Beweis. Es ist zu zeigen, dass für zwei Elemente aus $\ker(\Phi)$ die Verknüpfung wieder in $\ker(\Phi)$ liegt (mit einer anderen Operation als $+$ läuft der Beweis gleich ab): $a, b \in \ker(\Phi) \Rightarrow \Phi(a + b) \stackrel{\text{Homomorphismus}}{=} \Phi(a) + \Phi(b) \stackrel{\text{in } \ker(\Phi)}{=} 0_B + 0_B \stackrel{\text{Eigenschaften von } B}{=} 0_B \Rightarrow (a + b) \in \ker(\Phi)$. Ebenso läuft der Beweis für Skalare (bei Ringen und Vektorräumen): $a \in \ker(\Phi), \lambda \text{ Skalar} \Rightarrow \Phi(\lambda a) \stackrel{\text{Homomorphismus}}{=} \lambda \Phi(a) \stackrel{\text{in } \ker(\Phi)}{=} \lambda 0_B \stackrel{\text{Eigenschaften von } B}{=} 0_B \Rightarrow (\lambda a) \in \ker(\Phi)$.

□

Jetzt wird der zweite Teil der Aufgabe gezeigt:

3.4.4 Lemma ([65, Exercise A.23.b])

Seien A, B Vektorräume (oder Gruppen oder Ringe), und $\Phi : A \rightarrow B$ ein Homomorphismus.

Dann gilt Φ ist injektiv genau dann, wenn $\ker \Phi = \{0_A\}$.

Beweis. Angenommen $\ker \Phi = \{0_A, a\}$, $0_A \neq a$ und Φ injektiv, dann gilt $\Phi(a) = 0_B = \Phi(0_A) \xrightarrow{\text{injektiv}} a = 0_A \wedge a \neq 0_A$. Also folgt aus der Injektivität von Φ , dass $\ker \Phi = \{0_A\}$.

Wenn $\ker \Phi = \{0_A\}$ folgt $A/\ker \Phi \simeq \Phi(A) \subseteq B$

□

Damit folgt, dass ev (oder eine beliebige Abbildung) genau dann injektiv ist, wenn $\ker \text{ev} = \{0\}$, also $\ker \text{ev}$ trivial ist. Das $\dim \mathcal{C}(\mathcal{X}, \mathcal{P}, D) \stackrel{\text{Dimensionsformel}}{=} \dim \text{ev} \mathcal{L}(D) + \dim \ker \text{ev}$ folgt aus der Dimensionsformel.

Angenommen $\text{ev}(f) = 0$. Dann folgt aus der Definition ??, dass $f(P_1) = \dots = f(P_n) = 0$ gilt. Daraus folgt aber nach ??, dass alle P_i mindestens mit Koeffizient 1 in $\text{div}(f)$ vorkommen. Umgekehrt können die Punkte aus D nur mit negativem Faktor in $\text{div}(f)$ vorkommen. Da keines der P_i per Annahme in $\text{supp } D$ vorkommt, gilt $\text{div}(f) + D - P_1 - \dots - P_n \geq 0$.

Damit folgt aber auch,

dass $f \in \mathcal{L}(+D - P_1 - \dots - P_n)$.

Da nach 3.4.2 $\deg D < n$, hat der $D - P_1 - \dots - P_n$ negativen Grad und somit folgt nach

$\mathcal{L}(D - P_1 - \dots - P_n) = \{0\}$. Also ist $f = 0$ und der Kern ist trivial. $\Rightarrow \dim \mathcal{C} = \dim(D)$.

Durch die Wahl von $\deg D$ nach A.2.1 gilt die Gleichung ??, also $\dim \mathcal{L}(D) = \deg D + 1 - g$.

Als nächstes wird eine untere Schranke für d gezeigt:

Sei $\mathcal{C}(\mathcal{X}, \mathcal{P}, D) \ni w = (f(P_1), \dots, f(P_n)) = \text{ev}(f)$ ein Wort mit minimalem Gewicht $\text{wt}(w) = d$. Dann sind nach Definition ?? genau d der Koordinaten von w ungleich 0, also entsprechend $f(P_i) \neq 0$ für d der i und ebenso $f(P_i) = 0$ für $n - d$ der i .

Durch Wechsel der Nummerierung kann man annehmen, dass $f(P_{d+1}) = \dots = f(P_n) = 0$.

Wie oben folgt daraus, dass der $D - \text{div}(f) + D - P_{d+1} - \dots - P_n$ effektiv ist.

□

3.4.5 Lemma (Eigenschaften von $\mathcal{L}(D)$ [65, Exercise 5.13])

Sei D ein Divisor einer nichtsingulären projektiven Ebenenkurve \mathcal{C} , die über \mathbb{F}_q definiert ist.

Dann gilt:

- $\deg D \leq 0 \Rightarrow \mathcal{L}(D) = \{0\} \Leftrightarrow D \geq 0$

Beweis. Sei $\deg D \leq 0$, dann gilt für alle $f \in \mathcal{L}(D)$ $\operatorname{div} f \geq D$.

□

3.5 Encoder und Decoder

3.5.1 Encoder

Der Vorgang des Encodierens ist in der Regel ein relativ einfacher. Für Vektoren \vec{v} aus der Urbildmenge (also ist \vec{v} ein Daten-Wort) von G ist das zugehörige Codewort durch

$$\vec{v} \cdot G$$

gegeben.

Um beliebige Daten zu encodieren gibt es verschiedene Möglichkeiten, hier sei auf UTF-8, ASCII und ähnliche Standards verwiesen.

Der schwierige Teil, ist die Daten zu decodieren, wenn Fehler aufgetreten sind, was in den folgenden Abschnitten erklärt wird.

3.5.2 Decodierungsgrundlagen

Grundlegende Ideen der Decodierung sind die beiden folgenden Definitionen:

3.15 Definition (Minimumdistance Decoder [64][2.1])

Ein Minimumdistance Decoder eines Codes \mathcal{C} ist ein Decoder \mathcal{D} , der jedem Wort w das Codewort $\mathcal{D}(w)$ mit der geringsten Hamming -Distanz zuordnet.

3.16 Definition (Maximum Likelihood Decoder [64][2.1])

Ein Maximum Likelihood Decoder eines Codes \mathcal{C} ist ein Decoder \mathcal{D} , der jedem Wort w das Codewort $\mathcal{D}(w)$ mit der geringsten Fehlerwahrscheinlichkeit zuordnet.

Die Definition 3.16 ist die „natürliche“.

Sei Code \mathcal{C} ein Code, bei dem die Fehlerwahrscheinlichkeit für alle Stellen die gleiche ist, und die Wahrscheinlichkeit für alle Fehlerarten gleich ist.

Dann gilt:

3.5.3 Nächster Nachbar Decodierung

Nächster Nachbar Decodierung ist das, was man auch als BruteForce bezeichnen könnte, da man über alle Codewörter iteriert.

3.17 Definition (Nächster Nachbar Decodierung)

Die Nächster Nachbar Decodierungsmethode ist eine der einfachsten und ineffizientesten Methode, da $\#\mathcal{C}$ Operationen notwendig sind, außer das Wort r war bereits ein Codewort.

Diese Art der Decodierung ist nur bei $w(e) < d_{\min}/2$ sicher möglich.

Sei $w \in A^n$ die Eingabe, dann funktioniert diese Methode wie folgt:

Man berechnet für alle Codewort $c \in \mathcal{C}$ die Distanz $d(c, w)$ und wählt das c mit minimaler Distanz.

3.5.1 Lemma (Eigenschaften des Nächster Nachbar Decoder)

Sei $r = c + e$ mit $w(e) < d_{\min}/2$, dann gibt es genau ein $c \in \mathcal{C}$, sodass $r \in B_{d_{\min}/2}(c)$. Der Decoder kann also eine Entscheidung treffen.

Beweis. Sei $\bar{c} \in \mathcal{C}$ und $e \in A^n : w(e) < d_{\min}/2$, dann gilt für $\bar{c} = ??$:

$$\begin{aligned} d(r, \bar{c}) &= d(c + e, \bar{c}) \\ &= d(c + e, c) \\ &\stackrel{??}{=} d(e, 0) \\ &= wt(e) \\ &\leq d_{\min} \end{aligned} \tag{3.16}$$

und für $\bar{c} = ??$:

$$d(r, \bar{c}) = d(c + e, \bar{c}) \tag{3.17}$$

Nebenrechnung:

$$d(c, e) + d(\hat{c}, e) \geq d(c, \hat{c}) \quad | - d(e, c) \tag{3.18}$$

$$d(\hat{c}, e) \geq d(c, \hat{c}) - d(e, c) \tag{3.19}$$

und somit

$$d(c + e, \bar{c}) \geq d(c, \hat{c}) - d(e, c) \tag{3.20}$$

$$= d_{\min} - d(e, c) \tag{3.21}$$

$$> d_{\min} - d_{\min}/2 \tag{3.22}$$

$$= d_{\min}/2 \tag{3.23}$$

Also liegt r genau in einem der Bälle mit Radius $d_{\min}/2$ um eines der Codewörter. \square

3.5.4 Syndromedecoding

3.18 Definition (Vektorsyndrom, Syndrom [71][5.2.1])

Seien $\Phi, \Psi \in \mathbb{F}^n(D)$ und $\mathcal{P} := \sum_{i=1}^n P_i$.

Das Vektorsyndrom von Φ und einen Vektor $r = (r_1, r_2, \dots, r_n) \in \mathbb{F}^n$ ist definiert als

$$(\Phi \times r)_B := (\Phi(P_1) r_1, \Phi(P_2) r_2, \dots, \Phi(P_n) r_n)$$

Ein Syndrom wird als

$$(\Phi \cdot r)_B := \Phi(P_1) r_1 + \Phi(P_2) r_2 + \dots + \Phi(P_n) r_n$$

definiert.

3.5.2 Lemma (Syndrom Lemma [71, Lemma 5.2.3])

Sei das empfangene Wort $?? = c + e$, mit $c_i \in \{\bar{c} \in \mathcal{C} \mid \Phi \cdot c = 0 \forall \Phi \in \mathbb{F}^n(D)\}$ (also ein legitimes Codewort) und der Fehler $e \in \mathbb{F}^n$, dann ist $\Phi \cdot ?? = \Phi \cdot e \forall \Phi \in \mathbb{F}^n(D)$.

Beweis. $\Phi \cdot r = \Phi \cdot c + \Phi \cdot e = 0 + \Phi \cdot e = \Phi \cdot e$ Die erste Gleichheit folgt aus der Linearität von $??$ und die zweite aus der Bedingung an c .

□

3.19 Definition (Fehlerstelle, Errorlocator [71, Definition 5.2.4])

Sei $e = (e_1, e_2, \dots, e_n) \in \mathbb{F}^n$.

$e_i \neq 0$, ist P_i eine Fehlerstelle von e . Eine $0 \neq \Phi \in \mathbb{F}(\mathcal{C})$ ist ein Errorlocator für E , wenn $\Phi(P_i) = 0$ für alle Fehlerstellen P_i von e .

3.5.3 Lemma ([71, Lemma 5.2.6])

Wenn eine Funktion $\Theta \in \mathbb{F}(\mathcal{C})$ ein Errorlocator eines Fehlers e , dann gilt $\Theta \cdot e = 0$.

Beweis. Wenn Θ ein Errorlocator von $e = (e_1, e_2, \dots, e_n)$, dann gilt:

$$e_i \cdot \Theta_i = \begin{cases} e_i \cdot \Theta(x) = e_i \cdot 0 = 0 \forall x \in P_i & \text{wenn } e_i \neq 0, \\ 0 \cdot \Theta(x) = 0 & \text{sonst.} \end{cases}$$

□

3.5.4 Lemma (Existenz von Errorlocator [71, S. 5.2.7])

Sei $e \in \mathbb{F}^n$ mit $w(e) \leq t$ für ein $0 \neq t \in \mathbb{N}$. Sei A ein beliebiger Divisor mit einem Erzeugendensystem, das disjunkt zum Erzeugendensystem von B ist. Angenommen $l(A) > t$, dann gibt es einen Errorlocator in $??(A)$ für e .

Beweis. Sei $P_e := \{P_i | e_i \neq 0\}$ also die Menge aller Fehlerstellen von e . Sei $\{\Phi_i | i = 1, 2, \dots, l(A)\}$ eine Basis von $\mathcal{L}(A)$, dann ist

$$\Theta = \sum_{j=1}^{l(A)} \alpha_j \Phi_j \in ??(A)$$

nach Lemma 3.5.3 genau dann ein Errorlocator, wenn $\Theta(P_i = 0) \forall P_i \in P_e$.

Also ist die Suche nach einem Errorlocator äquivalent dazu das lineare Gleichungssystem

$$\Theta(P_i) = \sum_{j=1}^{l(A)} \alpha_j \phi_j(P_i) = 0 \text{ für } P_i \in P_e \quad (3.24)$$

zu lösen.

Dieses System hat $l(A)$ Unbekannte und höchstens $t (\geq w(e))$ Gleichungen; aus der Annahme $l(A) > t$ folgt, dass es eine Lösung $\neq 0$ gibt und diese korrespondiert zu einem Errorlocator.

□

3.5.5 Theorem ([71, Theorem 5.2.8])

Seien A und E Divisoren mit Erzeugendensystemen, die disjunkt zum Erzeugendensystem von \mathcal{P} sind, sodass

$$d(\mathcal{C}(\mathcal{P}, E)^\perp) > d(A) \quad (3.25)$$

Sei $\{\phi_i | i = 1, 2, \dots, l(E)\}$ eine Basis für $??(E)$. Sei $\Theta \in ??(A)$ und sei

$$I_\Theta := \{e \in \mathbb{F}^n | \Theta \text{ ist ein Errorlocator für } e\},$$

dann ist

$$\begin{aligned} f_\Theta : I_\Theta &\rightarrow \left\{ \left(\Phi_1 \cdot e, \Phi_2 \cdot e, \dots, \Phi_{l(E)} \cdot e \right) \mid e \in I_\Theta \right\} \\ e &\mapsto \left\{ \Phi_1 \cdot e, \Phi_2 \cdot e, \dots, \Phi_{l(E)} \cdot e \right\} \end{aligned} \quad (3.26)$$

eine Bijektion.

Beweis. Surjektiv: Das folgt aus der Definition, da $\{f_i \mid i = 1, 2, \dots, l(E)\}$ eine Basis für $??(E)$ ist und die e passend gewählt werden können.

Injektiv: Seien $e, e' \in I_\Theta$ und $f_\Theta(e) = f_\Theta(e')$. Es gilt $f_\Theta(e) - f_\Theta(e') = 0$, also $\Phi_i(e - e') = 0 \forall i = 1, 2, \dots, l(E)$. Da die Φ_i eine Basis von $??(E)$ bilden folgt $\Phi(e - e') = 0 \forall \Phi \in ??(E)$ und per Definition ?? ist somit $(e - e') \in \mathcal{C}(\mathcal{P}, E)^\perp$.

Wenn $e - e' \neq 0$, dann gilt durch die Annahmen 3.25 $w(e - e') \geq d(\mathcal{C}(\mathcal{P}, E)^\perp) > d(A)$. Das Θ ein Errorlocator sowohl von e als auch von e' ist und $\Theta \in ??(A)$ ist ein Widerspruch, wie wir gleich zeigen werden.

Es gilt für $\mathcal{P}_\Theta = \{P_i \mid \Theta(P) = 0\}$:

$$\Theta \in \mathcal{L}\left(A - \sum_{P \in \mathcal{P}_\Theta} P\right)$$

und somit

$$\mathcal{L}\left(A - \sum_{P \in \mathcal{P}_\Theta} P\right) \neq \emptyset \Rightarrow d\left(A - \sum_{P \in \mathcal{P}_\Theta} P\right) \geq 0.$$

Daraus folgt:

$$d(A) \geq d\left(\sum_{P \in \mathcal{P}_\Theta} P\right) = |\mathcal{P}_\Theta| \geq w(e - e')$$

Daraus folgt der Widerspruch und die Injektivität. □

Mit den Ergebnissen von Dai[71] und der Einschränkung auf lineare Codes, kann das folgende Lemma gezeigt werden.

3.5.6 Lemma (Syndromedecoding)

Sei $?? = \text{Codewort} + e$ mit c aus Code \mathcal{C} mit Prüfmatrix H und $e, ?? \in A^n$ gegeben und wenn $wt(e)$ klein genug ist, kann c rekonstruiert werden.

Beweis. Wenn $??$, Codewort, e wie oben, dann gilt aufgrund der Linearität der Matrixmultiplikation

$$H?? = H(e + c).$$

Wenn $wt(e)$ klein genug ist, folgt mit Lemma 3.5.4, dass ein Errorlocator existiert und, dass ein solcher zu einem linearen Gleichungssystem über einer Basis von $\mathcal{L}(A)$ korrespondiert. □

Binäre Codes haben in der Codierungstheorie den Vorteil, dass durch die Bestimmung der Fehlerstelle der Fehler bereits korrigierbar ist. Codes mit einer anderen Basis als 2 haben dagegen den Vorteil, dass entsprechend mehr Information übertragen werden kann, da der Raum der Wörter entsprechend größer ist.

Kapitel 4

Implementierung

In dieser Arbeit wird die Implementierung `goppa.py`[89], die unter anderem von Sluková, Sluková, Sluková, Sluková, Sluková [141, 138, 142, 143, 139, 140] im Rahmen von *GSoC/2019* [119] und Ion [125] im Rahmen von *GSoC/2019* [119] entwickelt wurde, betrachtet.

Desweiteren werden auch einige andere Codefragmente aus *sagemath-8.9.beta3* [**sagemath:8.9.beta3**, 90, 91, 92, 89, 93] betrachtet, da diese von den betrachteten Implementierungen genutzt werden.

Die Entscheidung für das **SAGE** Project ist darauf begründet, dass es ein Free and Open Source Software Projekt ist und es somit auch möglich ist, den Quelltext zu analysieren.

Im Folgenden werden einige Begriffe „missbraucht“, so wird nicht sauber zwischen Listen und Vektoren unterschieden, da diese sich in der Programmierung nur minimal unterscheiden.

Die Erklärungen beschränken sich auf die Funktionen des Moduls *coding*. Oft bestehen diese aus der mathematischen Operation, einigen Plausibilitäts-Checks, da die mathematischen Operationen sonst nicht definiert sind, und internen Operationen, die der Sprache geschuldet sind; so muss eine Erzeugermatrix G unveränderlich sein, da sich sonst bei einer Änderung der Matrix G alle von ihr abgeleiteten Codes ändern.

4.1 Initialisierung

4.1.1 Beispiel

Im Folgenden wird das wiederkehrende Beispiel aus der verwendet. Zuerst wird der Aufruf mit der Definition ?? unter Nutzung des Beispiels aus `sagemath-8.9.beta3[88]` verglichen.

Listing 4.1: Funktionsdefinition

```
sage: F = GF(2^6)
sage: R.<x> = F[]
sage: g = x^9 + 1
sage: L = [a for a in F.list() if g(a) != 0]
```

Das Polynom $g = x^9 + 1 \in \mathbb{F}_{2^6} = F$ und die Menge $L = \{a \in F \mid |g(a) \neq 0\}$ erzeugen nach 3.9 einen Goppa Code, da

Listing 4.2: EXAMPLE[94, S. 86–92]

```
sage: F = GF(2^6)
sage: R.<x> = F[]
sage: g = x^9 + 1
sage: L = [a for a in F.list() if g(a) != 0]
sage: C = codes.GoppaCode(g, L)
sage: C
[55, 16] Goppa code over GF(2)
```

$$F = \mathbb{F}_{2^6}[x] \ni g = x^9 + 1 \tag{4.1}$$

Die Beispiele werden im Text mit SageTeX `[tex:sagetex]` gesetzt.
??

Listing 4.3: Funktionsdefinition

```
sage: F8=GF(2**6)
sage: RR.<xx> = F8[]
sage: g = xx^9+1
```

Wir erzeugen also eine Polynom $g(x)$ über einem Körper in 2^6 Elementen (??).

Listing 4.4: Festlegen von L , wie in 3.9

```
sage: L = [a for a in F8.list() if g(a)!=0]
```

Diese Menge ist von folgender Gestalt:

$$L = ??$$

.

Listing 4.5: Aufruf von `GoppaCode`

```
sage: C = codes.GoppaCode(g, L)
```

Der Aufruf des Konstruktors funktioniert wie folgt:

Listing 4.6: `goppa.py`

```
def __init__(self, generating_pol, defining_set):

    self._field = generating_pol.base_ring().prime_subfield()
    self._length = len(defining_set)
    self._generating_pol = generating_pol
    self._defining_set = defining_set

    super(GoppaCode, self).__init__(self._field, self._length, "GoppaEncoder", "Syndrome
    ↪ ")

    if not generating_pol.is_monic():
        raise ValueError("generating polynomial must be monic")
    F = self._field
    if (not F.is_field() or not F.is_finite()):
        raise ValueError("generating polynomial must be defined over a finite field")
    for a in defining_set:
        if generating_pol(a) == 0:
            raise ValueError("defining elements cannot be roots of generating polynomial")
```

Es wird ein `CodeObjekt` angelegt und in den Schritten der Form (`self._`) werden Eigenschaften des Objektes gespeichert und der Rest sind eben solche Plau-si-bi-li-täts-prü-fun-gen, die bereits erwähnt wurden.

Listing 4.7: Methoden der Klasse

```
def __init__(self, generating_pol, defining_set):
def _repr_(self):
def _latex_(self):
def __eq__(self, other):
    """
def distance_bound(self):
```

Dieses `CodeObjekt` hat jetzt Funktionen der Klasse geerbt, aber diese werden hier nur analysiert, wenn sie benötigt werden.

Per Definition erbt die Klasse von der abstrakten Klasse `AbstractLinearCode` die folgenden Methoden und somit auch das `CodeObjekt`.

Listing 4.8: Methoden der abstrakten Klasse

```
class AbstractLinearCode(Module):
    def __init__(self, base_field, length, default_encoder_name, default_decoder_name):
    def __getstate__(self):
    def _repr_(self):
```

```

def __latex__(self):
def __an_element__(self):
def add_decoder(self, name, decoder):
def add_encoder(self, name, encoder):
def automorphism_group_gens(self, equivalence="semilinear"):
def ambient_space(self):
def assmus_mattson_designs(self, t, mode=None):
def base_field(self):
    r"""O3
    Return the base field of “self”.O32
@cached_method
def canonical_representative(self, equivalence="semilinear"):
def __contains__(self, v):
def characteristic(self):
def characteristic_polynomial(self):
def chinen_polynomial(self):
@cached_method
@cached_method
def decode_to_code(self, word, decoder_name=None, *args, **kwargs):
def decode_to_message(self, word, decoder_name=None, *args, **kwargs):
@cached_method
def decoders_available(self, classes=False):
def divisor(self):
def is_projective(self):
def dual_code(self):
def dimension(self):
def direct_sum(self, other):
    """
    Direct sum of the codes “self” and “other”
def juxtapose(self, other):
def u_u_plus_v_code(self, other):
def product_code(self, other):
def construction_x(self, other, aux):
def __eq__(self, right):
def __ne__(self, other):
def encode(self, word, encoder_name=None, *args, **kwargs):
def __call__(self, m):
@cached_method
def encoders_available(self, classes=False):
def extended_code(self):
    r"""
    Returns ‘self’ as an extended code.
def galois_closure(self, F0):
def __getitem__(self, i):
def generator_matrix(self, encoder_name=None, **kwargs):
def systematic_generator_matrix(self, systematic_positions=None):
@cached_method
def genus(self):
def __iter__(self):

```

```
@cached_method
def is_information_set(self, positions):
def is_permutation_automorphism(self,g):
def is_permutation_equivalent(self,other,algorithm=None):
def unencode(self, c, encoder_name=None, nocheck=False, **kwargs):
def weight_enumerator(self, names=None, bivariate=True):
@cached_method
def zeta_polynomial(self, name="T"):
def zeta_function(self, name="T"):
```

Auf die Methoden, die aus `Module` importiert werden, wird in dieser Arbeit nicht weiter eingegangen.

Die einzige Zeile, die etwas komplexer ist, ist Zeile 57 und für deren Verständnis werden die Zeilen 244 und 310 benötigt.

Listing 4.9: Methoden der Klasse

```
super(GoppaCode, self).__init__(self._field, self._length, "GoppaEncoder", "Syndrome
    ↪ ")
class GoppaCodeEncoder(Encoder):
GoppaCode._registered_encoders["GoppaEncoder"] = GoppaCodeEncoder
```

Gemeinsam überschreiben sie die folgende Methode:

Listing 4.10: Methoden der Klasse

```
def __init__(self, base_field, length, default_encoder_name, default_decoder_name):
    """
    self._registered_encoders["Systematic"] = LinearCodeSystematicEncoder
    self._registered_decoders["Syndrome"] = LinearCodeSyndromeDecoder
    self._registered_decoders["NearestNeighbor"] = LinearCodeNearestNeighborDecoder
    from sage.coding.information_set_decoder import LinearCodeInformationSetDecoder
    self._registered_decoders["InformationSet"] = LinearCodeInformationSetDecoder

    if not isinstance(length, (int, Integer)):
        raise ValueError("length must be a Python int or a Sage Integer")
    if length <= 0:
        raise ValueError("length must be a non-zero positive integer")
    if not base_field.is_field():
        raise ValueError("'base_field' must be a field (and {} is not one)".format(base_field)
            ↪ )
    if not default_encoder_name in self._registered_encoders:
        raise ValueError("You must set a valid encoder as default encoder for this code, by
            ↪ filling in the dictionary of registered encoders")
    if not default_decoder_name in self._registered_decoders:
        raise ValueError("You must set a valid decoder as default decoder for this code, by
            ↪ filling in the dictionary of registered decoders")

    self._length = Integer(length)
    self._default_decoder_name = default_decoder_name
```



```

self.__default_encoder_name = default_encoder_name
cat = Modules(base_field).FiniteDimensional().WithBasis().Finite()
facade_for = VectorSpace(base_field, self.__length)
self.Element = type(facade_for.an_element()) #for when we made this a non-facade
    ↪ parent
Parent.__init__(self, base=base_field, facade=facade_for, category=cat)

```

Das CodeObjekt `C` ist also ein `GoppaCode C ??` mit `GoppaCodeEncoder C.encoder()` ?? als Encoder und einem `LinearCodeSyndromeDecoder` als Decoder.

4.2 Generatormatrix

Die Generatormatrix des Code-Objekts wird wie folgt generiert:

Listing 4.11: `generator_matrix`

```

def generator_matrix(self):
    c = self.code()
    pmat = c.parity_check_matrix()
    aux = codes.from_parity_check_matrix(pmat)
    return aux.generator_matrix()

```

Die Implementierung der Klasse `GoppaCode` besitzt zwar keine explizite Beschreibung der Erzeugermatrix, aber eine Implementierung der Prüfmatrix, reicht aus, wie in ?? erklärt wird.

Dies reicht nach ?? aus, um die Erzeugermatrix zu definieren, da

$$H_C = G_{C^\perp} \quad \text{und} \quad G_{C^{\perp\perp}} = G_C. \quad \text{gilt.}$$

Denn `from_parity_check_matrix` ist in `coding/code_construction.py` wie folgt definiert.

Listing 4.12: `from_parity_check_matrix`

```

def from_parity_check_matrix(H):
    Cd = LinearCode(H)
    return Cd.dual_code()

```

Wobei

Listing 4.13: `dual_code`

```

def dual_code(self):
    return LinearCode(self.parity_check_matrix())

```

Unter der Annahme, dass `parity_check_matrix` die Prüfmatrix zurück gibt, gilt also:

$$\mathcal{C}(G)^\perp \quad (4.2)$$

Und `parity_check_matrix` liefert tatsächlich eine Prüfmatrix zurück, wie man hier sieht:

Listing 4.14: `parity_check_matrix`

```
def parity_check_matrix(self):
    G = self.generator_matrix()
    H = G.right_kernel()
    M = H.basis_matrix()
    M.set_immutable()
    return M
```

Wobei `generator_matrix(code)` die Erzeugermatrix G des übergebenen Codes zurückgibt.

$$\mathbf{G} = G(\mathcal{C}) \quad (4.3)$$

Der `right_kernel` H von G ist definiert als

$$H = \{x \mid \mathbf{G}x = 0\} = \{x \mid Gx = 0\} \quad (4.4)$$

und die Matrix \mathbf{M} , ist eine Matrix in Stufenform, die H erzeugt.
Damit gilt:

$$\mathbf{G}\mathbf{M}^{\mathbf{M} \subseteq \mathbf{H}} = 0 \quad (4.5)$$

Somit ist \mathbf{M} eine Prüfmatrix für \mathcal{C} .
Und somit

$$\mathbf{pmat} = H_{\mathcal{C}} \quad (4.6)$$

$$\mathbf{aux} = \mathcal{C} \left(\underbrace{H_{\mathcal{C}}}_{\mathbf{pmat}} \right)^\perp \quad (4.7)$$

$$\underbrace{G_{\mathcal{C}^\perp}} \quad (4.8)$$

und

$$G_{\mathcal{C}^{\perp\perp}} = G_{\mathcal{C}} \quad (4.9)$$

.

4.3 Prüfmatrix

Im folgenden Abschnitt wird die Implementierung der Berechnung der Prüfmatrix betrachtet.

Listing 4.15: `parity_check_matrix`

```
def parity_check_matrix(self):
    g = self._generating_pol
    F = g.base_ring()
    m = self.base_field().degree()
    n = self._length
    d = g.degree()
    alpha = F.primitive_element()
```

$g = g$
 $F = \mathbb{F}^m$
 $m = m$
 $n = \#L$
 $d = \deg g$

Das Polynom g ist wieder das Erzeugerpolynom für einen Goppa Code \mathbb{F} der Körper (präziser: der Ring), über dem g definiert ist und d der Grad von g .
 n ist die Länge des Codes (die Mächtigkeit der Menge L , über der der Code definiert wird) und m die Dimension von \mathbb{F} (die Anzahl der Kopien von F).
 α ist die Menge aller primitiven Elemente

Listing 4.16: `goppa.py`

```
D = self._defining_set
h = [(g(D[i]).inverse_of_unit()) for i in range(n)]
```

$D = (P_1, \dots, P_n)$ ist die Menge der Punkte über denen den Code definiert wird, wie auch in ?? (äquivalent zu L in [59]) und der Vektor \vec{h} enthält die Inversen $g(P_i)^{-1}$.

Listing 4.17: `goppa.py`

```
D = self._defining_set
h = [(g(D[i]).inverse_of_unit()) for i in range(n)]
```

Der `augment` Befehl dient dazu, aus einer $n \times k$ und einer $n \times j$ Matrix eine $n \times k + j$ Matrix zu bilden, die die beiden als Untermatrizen aufnimmt, siehe [136].
Der Befehl `_columnize` aus ?? transponiert ein Element aus $\mathbb{F}_{q^m}^n$.

Listing 4.18: `goppa.py`

```
def _columnize(element):  
    v = vector(element)  
    return v.column()
```

Diese Matrix ist die gleiche, die in 3.14 beschrieben wird.
Es gibt einige Änderungen des Codes, die aber in der betrachteten Version noch nicht inkludiert sind (siehe [108]), aber die Erklärung einfacher machen, daher werden sie hier eingebunden

Listing 4.19: `goppa.py`: paritycheckmatrix loop short

```
sage: C  
[55, 16] Goppa code over GF(2)  
sage: E = codes.encoders.GoppaCodeEncoder(C)
```

4.4 Encoder und Decoder

Die Klasse der Goppa Codes importiert aus *sagemath-8.9.beta3* 4.20 die Decoder `LinearCodeNearestNeighborDecoder`, `LinearCodeSyndromeDecoder` und `LinearCodeInformationSetDecoder`.

und aus [96] 4.21 den Encoder `GoppaCodeEncoder`.

Diese Arbeit betrachtet `GoppaCodeEncoder`, `LinearCodeNearestNeighborDecoder` und `LinearCodeSyndromeDecoder`.

Listing 4.20: `decoders_catalog`

```
_lazy_import('sage.coding.linear_code', ['LinearCodeNearestNeighborDecoder',  
                                          'LinearCodeSyndromeDecoder',  
                                          'LinearCodeInformationSetDecoder'])
```

Listing 4.21: `encoders_catalog`

```
#####
```

4.4.1 Encoder

Der allgemeine Encoder ist wie folgt definiert 4.22:

Listing 4.22: `encoder.py: class Encoder`

```
Every encoder class should inherit from this abstract class.
This is a default implementation which assumes that the message

def __call__(self, m):
    r"""
    Transforms an element of the message space into a codeword.
```

Dies entspricht der Definition aus ?? mit einem zusätzlichen `glsplausibility-check`.

Listing 4.23: `linear_code.py`

```
1,0,0,0,0,1,1,
0,1,0,0,1,0,1,
0,0,0,1,1,1,1
));
sage: f.close()
from sage.misc.temporary_file import tmp_filename
F = C.base_ring()
p = F.order() # must be prime and <11
Gr = [str(r)[1:-1].replace(" ", "") for r in C.generator_matrix().rows()]
sage: from sage.coding.information_set_decoder import
    ↳ LinearCodeInformationSetDecoder
sage: cl = LinearCodeInformationSetDecoder
sage: _explain_constructor(cl)
"The constructor requires the arguments ['number_errors'].\nIt takes the optional
    ↳ arguments ['algorithm'].\nIt accepts unspecified arguments as well.\nSee the
    ↳ documentation of sage.coding.information_set_decoder.
    ↳ LinearCodeInformationSetDecoder for more details."
```

Das eigentliche Encodieren findet wie folgt statt: Der Code wird zwischengespeichert, dann wird dessen Paritycheckmatrix als `pmat` zwischengespeichert. Die Paritycheckmatrix für den von `pmat` erzeugten Code als Erzeugermatrix ausgegeben. Dies ist nach er auch eine Erzeugermatrix. Diese ist nötig, da die allgemeine Methode für `generator_matrix` die Erzeugermatrix des Encoders zurück gibt.

4.4.2 Nearest Neighbour

Listing 4.24: label=lst:nearest-neighbor-decoder Klasse `LinearCodeNearestNeighborDecoder`

```
class LinearCodeNearestNeighborDecoder(Decoder):
    def __init__(self, code):
    def __eq__(self, other):
    def __repr__(self):
    def __latex__(self):
    def decode_to_code(self, r):
    def decoding_radius(self):
```

Diese Methode beruht darauf, über alle validen Codewörter zu iterieren, und jeweils das, mit der geringsten bisher bekannten Hammingdistanz zur Eingabe zu speichern und am Ende auszugeben. Die Mathematik wurde bereits in 3.5.3 betrachtet.

Dies geschieht hier:

Listing 4.25: label=lst:nearest-neighbor-decoder:decode-to-code Klasse `LinearCodeNearestNeighborDecoder`

```
def decode_to_code(self, r):
    c_min = self.code().zero()
    h_min = r.hamming_weight()
    for c in self.code():
        if (c-r).hamming_weight() < h_min:
            h_min = (c-r).hamming_weight()
            c_min = c
    c_min.set_immutable()
    return c_min
```

Der Decodierungsradius dieses Codes wird $\lfloor (d_{\min} - 1) / 2 \rfloor$ angegeben und dies stimmt mit 3.5.1 überein, da

$$d \leq \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor < \left\lfloor \frac{d_{\min}}{2} \right\rfloor$$

gilt.

4.4.3 Syndromedecoder

Listing 4.26: Klasse `LinearCodeSyndromeDecoder`

```
class LinearCodeSyndromeDecoder(Decoder):
    def __init__(self, code, maximum_error_weight=None):
    def __eq__(self, other):
    def __hash__(self):
```

```

def __repr__(self):
def __latex__(self):
@cached_method
def decode_to_code(self, r):
def maximum_error_weight(self):
def decoding_radius(self):
def syndrome_table(self):

```

Die Klasse der **LinearCodeSyndromeDecoder** kann beim Aufruf optional einen Wert für den maximalen Fehler übergeben bekommen, der festlegt bis zu welchem Grad die **lookup_table** vorbereitet wird.

Wenn der übergebene Wert höher ist, wird ein Fehler gemeldet und wenn er nicht übergeben wird, wird er als $n - k$ festgelegt, wobei der Code, über dem der Decoder definiert wird, ein n, k -Code ist.

Listing 4.27: Klasse **LinearCodeNearestNeighborDecoder**

```

def __init__(self, code, maximum_error_weight=None):
    n_minus_k = code.length() - code.dimension()
    if maximum_error_weight is None:
        self._maximum_error_weight = n_minus_k
    elif not isinstance(maximum_error_weight, (Integer, int)):
        raise ValueError("maximum_error_weight has to be a Sage integer or a Python int
        ↪ ")
    elif maximum_error_weight > n_minus_k:
        raise ValueError("maximum_error_weight has to be less than code's length minus
        ↪ its dimension")
    else:
        self._maximum_error_weight = maximum_error_weight
    super(LinearCodeSyndromeDecoder, self).__init__(code, code.ambient_space(), \
        code._default_encoder_name)
    self._lookup_table = self._build_lookup_table()

```

Da die **lookup_table** während der Initialisierung erzeugt wird, kann dieser Prozess die **SAGE** Umgebung zum Absturz bringen. Mehr dazu wird in Abschnitt 4.5 erklärt. Bei Betrachtung der dazugehörige Funktion **_build_lookup_table** fällt der Marker **@cached_method** auf. Dieser hält die Methode im Cache [144]:

Listing 4.28: label=lst:syndrome-decoder:lookuptable:init Initialisierung der Variablen

```

@cached_method
def _build_lookup_table(self):
    t = self._maximum_error_weight
    self._code_covering_radius = None
    self._code_minimum_distance = None
    self._decoder_type = copy(self._decoder_type)
    self._decoder_type.remove("dynamic")
    C = self.code()

```

```
n = C.length()
k = C.dimension()
H = C.parity_check_matrix()
F = C.base_ring()
l = list(F)
zero = F.zero()
```

Bis zu diesem Punkt wurden Variablen definiert und jetzt beginnt der Teil, der in den Tests ?? Probleme bereitet hat.

Listing 4.29: label=lst:syndrome-decoder:lookuptable:lookup Erzeugen der Tabelle

```
#Builds a list of generators of all error positions for all
#possible error weights
if zero in l:
    l.remove(zero)
# Remember to include the no-error-vector to handle codes of minimum
# distance 1 gracefully
zero_syndrome = vector(F,[F.zero()]*(n-k))
zero_syndrome.set_immutable()
lookup = { zero_syndrome : vector(F,[F.zero()]*n) }
error_position_tables = [cartesian_product([l]*i) for i in range(1, t+1)]
first_collision = True
#Filling the lookup table
for i in range(1, t+1):
    stop = True
    patterns = Subsets(range(n), i)
    basic = vector(F, n)
    for p in patterns:
        for error in error_position_tables[i-1]:
            ind = 0
            e = copy(basic)
            for pos in p:
                e[pos] = error[ind]
                ind += 1
```

Die Laufzeit in Schritten lässt sich wie folgt abschätzen:

Die äussere Kurve wird **maximum_error_weight** mal durchlaufen, da die obere Schranke für **range** ein echt kleiner von den Werten verlangt.

Die nächste Schleife benötigt im i -ten Schritt $\binom{n}{i}$ Schritte. Also sind es schon $\sum_{i=1}^t \binom{n}{i}$ Schritten, die zur Erstellung dieser **lookup_table** benötigt werden.

Die nächsttiefere Schleife läuft über das Kartesische Produkt von i Kopien von \mathbb{F}_q^\times ,

Die aktuelle Schrittzahl lautet: $\sum_{i=1}^t \binom{n}{i} \# \mathbb{F}^i$.

Die kleine Schleife im Inneren läuft wieder über i .

Damit sind es insgesamt

$$\sum_{i=1}^t \binom{n}{i} \# \mathbb{F}^i i$$

Schritte.

Dieses gesamte Konstrukt kann wie folgt dargestellt werden:

```
for p in Subsets(range(3),2):
    p
    for error in cartesian\_product([t]*2):
        error
        ind=0
        for pos in p:
            # iteriere \"uber den Fehler Vektor
            error[ind]
            ind+=1
```

Es wird also über alle Möglichkeiten i Fehler zu haben iteriert und die eindeutige Zuordnung wird dadurch sicher gestellt, dass bei Auftauchen einer Kollision die Decodierungsdistanz des Decoders verringert wird.

Listing 4.30: label=lst:nearest-neighbor-decoder Klasse
LinearCodeNearestNeighborDecoder

```
def __init__(self, code, maximum_error_weight=None):
    n_minus_k = code.length() - code.dimension()
    if maximum_error_weight is None:
        self._maximum_error_weight = n_minus_k
    elif not isinstance(maximum_error_weight, (Integer, int)):
        raise ValueError("maximum_error_weight has to be a Sage integer or a Python int
        ↪ ")
    elif maximum_error_weight > n_minus_k:
        raise ValueError("maximum_error_weight has to be less than code's length minus
        ↪ its dimension")
    else:
        self._maximum_error_weight = maximum_error_weight
    super(LinearCodeSyndromeDecoder, self).__init__(code, code.ambient_space(),\
        code._default_encoder_name)
    self._lookup_table = self._build_lookup_table()
```

Die Mathematik für diesen Decoder wurde bereits in ?? dargestellt, dennoch folgt eine kurze Analyse des Codes:

Sei \mathcal{C} ein Code mit Länge n über einem Körper mit l Elementen.

Der Algorithmus iteriert über alle Teilmengen von $1 \dots n$ mit Mächtigkeit i und alle möglichen Kombination von Ringelementen, also über

$$\sum_e \underbrace{l^e}_{\text{mögliche Fehler}} \underbrace{\binom{n}{e}}_{\text{mögliche Fehlerstellen}},$$

dies erklärt, warum bei den 4.5 durch den „OOM-Killer“ beendet wurden, mehr dazu hier 4.1.

4.5 Tests

Im Folgenden werden einige Tests beschrieben und die Testergebnisse graphisch dargestellt.

Die Testsysteme haben sich im Laufe der Arbeit gewandelt und das endgültige Testsystem hat die in 4.1 beschriebenen Spezifikationen:

Dennoch war auch hier der Arbeitsspeicher nicht ausreichend, um für einen grossen Goppa Code den `LinearCodeSyndromeDecoder` zu erstellen.

In Grafik 4.1 sieht man, dass der Speicherverbrauch abhängig von der Fehleranzahl, für die der Code designed ist, exponentiell ansteigt.

Und das Systemlog zeigt ??

Listing 4.31: Systemlog „killed process“

```
[Sa Sep 21 12:31:33 2019] [18947] 1001 18947 38150 84 29 0 0 free
[Sa Sep 21 12:31:33 2019] [28089] 1001 28089 32149 332 18 0 0 screen
[Sa Sep 21 12:31:33 2019] [28090] 1001 28090 28896 116 12 0 0 bash
[Sa Sep 21 12:31:33 2019] [30044] 1001 30044 46036 1137 44 0 0 python
[Sa Sep 21 12:31:33 2019] [34468] 1001 34468 38150 84 29 0 0 free
[Sa Sep 21 12:31:33 2019] [34601] 1001 34601 28871 112 13 0 0 bash
[Sa Sep 21 12:31:33 2019] [34617] 1001 34617 163986496 130632996 255686 0 0 python2
[Sa Sep 21 12:31:33 2019] [34850] 1001 34850 38150 84 30 0 0 free
[Sa Sep 21 12:31:33 2019] [34851] 1001 34851 26997 25 10 0 0 tail

[Sa Sep 21 12:31:33 2019] Out of memory: Kill process 34617 (python2) score 991 or sacrifice
    ↳ child
[Sa Sep 21 12:31:33 2019] Killed process 34617 (python2) total-vm:655945984kB, anon-rss
    ↳ :522531984kB, file-rss:0kB, shmem-rss:0kB
```

Das Module `memory_profiler` wurde zusätzlich mit `./sage -python -m easy_install memory_profiler` installiert, um die Funktion `%memit` zu nutzen. Diese führt eine Funktion mehrfach aus, um einen besseren Wert für die Speicherauslastung zu finden. Die Funktion `%timeit` misst nach dem selben Prinzip die benötigte Zeit.

| | |
|---------------------|---|
| CPU Model: | Intel(R) Xeon(R)CPU E5-2670 v2 @ 2.50GH |
| Socket: | 2 |
| Kerne (pro Socket) | 10 |
| Threads (pro Kern): | 2 |
| Arbeitsspeicher: | 512G |
| Betriebssystem: | CentOS Linux 7 (Core) |

Tabelle 4.1: Testsystem



Die folgenden Skripte (BZW. ältere Versionen von ihnen) haben die Daten[121] erzeugt:

Listing 4.32: `test_ipython.sage`: Test von Syndrome- und NearestNeighbor-Decoder

```
# https://stackoverflow.com/questions/252703/what-is-the-difference-between-pythons-list
#   ↳ -methods-append-and-extend
# https://ask.sagemath.org/question/9703/how-to-output-data-to-a-file/
# https://pypi.org/project/memory-profiler/
# https://stackoverflow.com/questions/19092812/measure-max-memory-usage-with-ipython
#   ↳ -like-timeit-but-memit
# https://stackoverflow.com/questions/10361206/how-to-run-an-ipython-magic-from-a-
#   ↳ script-or-timing-a-python-script

# load ipython extension
%load_ext memory_profiler
# tools
def random_error_true(code,max_error,quiet=false):
    if quiet!=false:
        print ("get random error for " +str(code) + " with max error " + str(max_error))
    while True:
        code_error = code.ambient_space().random_element(prob=max_error/code.length())
        if code_error.hamming_weight() <= max_error and code_error.hamming_weight()>0:
            break
    return code_error

def random_error(word,max_error,quiet=false):
    if quiet!=false:
        print ("get random error for " +str(word) + " with max error " + str(max_error))
    # https://ask.sagemath.org/question/9445/copy-vectors/
    word_length=word.length()
    word_error=copy([0]*word_length)
    error_set=Subsets(range(word_length),max_error).random_element()
    for entry in error_set:
        # get a random non zero Element
        while True:
            rand_entry=word.base_ring().random_element()
            if rand_entry!= 0:
                break
        word_error[entry]=word_error[entry]+rand_entry
    return vector(word_error)

def test_decode(decoder, max_error = 0,quiet=false):
    # https://stackoverflow.com/questions/4606919/in-python-try-until-no-error
    # do until
    while max_error==0:
        try:
```

```
        max_error = decoder.decoding_radius()
    except:
        pass
    if quiet!=false:
        print ("testing decoding for "+str(decoder))
    code = decoder.code()
    e = random_error(code,max_error)
    c = code.random_element()
    r = c+e
    if decoder.decode_to_code(r)==c:
        if quiet!=false:
            print "correctly decoded"
    else:
        raise sage.coding.decoder.DecodingError('Decoded incorrectly')

def test_decode_combined(code, decoder_type="nearestNeighbor", max_error_designed=5,
    ↪ max_error_test=5):
    if decoder_type=="nearestNeighbor":
        decoder=codes.decoders.LinearCodeNearestNeighborDecoder(code)
    elif decoder_type=="syndrome":
        decoder=codes.decoders.LinearCodeSyndromeDecoder(code,maximum_error_weight=
    ↪ max_error_designed)
    else:
        raise ValueError('decoder_type')
    try:
        test_decode(decoder,max_error=max_error_test)
    except DecodingError:
        raise DecodingError('Incorrect Decoding')

def test_suite(code, csvfile, max_error_test=5, max_error_design=5, runs=3):
    suitedate=time.strftime("%y-%m-%dT%H:%M+0200")
    csvwriter = csv.writer(csvfile, delimiter=',')
    # entries are defined as
    # decoder_name: syndrome,nearestNeighbor
    # type: mem, time
    # stage: init, run, combined
    # max_error_designed
    # max_error_test
    ROW=[rundate,"started run",suitedate,"started test_suite"]
    ROW.extend([0]*(runs+1))
    csvwriter.writerow(ROW)
    ROW=["decoder_name","type","stage","max_error_designed","max_error_test"]
    ROW.extend([0]*runs)
    csvwriter.writerow(ROW)
    csvfile.flush()

    # create empty list to store the decoders
    syndrome_decoders=[]
    for i in range(1,int(max_error_test)):
```

```

# syndrome
# mem
# init
row=["syndrome","mem","init",i,"0"]
try:
    mem=%memit -c -q -r {runs} -o decoder=codes.decoders.
        ↪ LinearCodeSyndromeDecoder(C, maximum_error_weight={i})
    MEM=mem.mem_usage
except DecodingError:
    MEM=[0]*runs
row.extend(MEM)
csvwriter.writerow(row)
csvfile.flush()
decoder=codes.decoders.LinearCodeSyndromeDecoder(C, maximum_error_weight=i)
syndrome_decoders.append(decoder)
# time
# init
row=["syndrome","time","init",i,"0"]
try:
    timing=%timeit -c -q -r {runs} -o decoder=codes.decoders.
        ↪ LinearCodeSyndromeDecoder(C, maximum_error_weight={i})
    TIME=timing.all_runs
except DecodingError:
    TIME=[0]*runs
row.extend(TIME)
csvwriter.writerow(row)
csvfile.flush()

# nearestNeighbor
# mem
# init
row=["nearestNeighbor","mem","init","0","0"]
try:
    mem=%memit -c -q -r {runs} -o decoder=codes.decoders.
        ↪ LinearCodeNearestNeighborDecoder(C)
    MEM=mem.mem_usage
except DecodingError:
    MEM=[0]*runs
row.extend(MEM)
csvwriter.writerow(row)
csvfile.flush()
decoder=codes.decoders.LinearCodeSyndromeDecoder(C, maximum_error_weight=i)
nearestNeighbor_decoder=decoder
# time
# init
row=["nearestNeighbor","time","init","0","0"]
try:
    timing=%timeit -c -q -r {runs} -o decoder=codes.decoders.
        ↪ LinearCodeSyndromeDecoder(C, maximum_error_weight={i})

```

```

        TIME=timing.all_runs
    except DecodingError:
        TIME=[0]*runs
    row.extend(TIME)
    csvwriter.writerow(row)
    csvfile.flush()

# Time Decoding
# syndrome
for decoder in syndrome_decoders:
    for i in range(1,max_error_test):
        # run
        row=["syndrome","mem","run",decoder.decoding_radius(),i]
        try:
            mem=%memit -c -q -r {runs} -o test_decode(decoder)
            MEM=mem.mem_usage
        except DecodingError:
            MEM=[0]*runs
        row.extend(MEM)
        csvwriter.writerow(row)
        csvfile.flush()
        # run
        row=["syndrome","time","run",decoder.decoding_radius(),i]
        try:
            timing=%timeit -c -q -r {runs} -o test_decode(decoder)
            TIME=timing.all_runs
        except DecodingError:
            TIME=[0]*runs
        row.extend(TIME)
        csvwriter.writerow(row)
        csvfile.flush()

# nearestNeighbor
decoder=nearestNeighbor_decoder
# mem
# run
row=["nearestNeighbor","mem","run",decoder.decoding_radius(),i]
try:
    mem=%memit -c -q -r {runs} -o test_decode(decoder)
    MEM=mem.mem_usage
except DecodingError:
    MEM=[0]*runs
row.extend(MEM)

# time
# run
row=["nearestNeighbor","time","run",decoder.decoding_radius(),i]
try:
    timing=%timeit -c -q -r {runs} -o test_decode(decoder)

```

```

        TIME=timing.all_runs
    except DecodingError:
        TIME=[0]*runs
    row.extend(TIME)
    csvwriter.writerow(row)
    csvfile.flush()

# Time All
for max_error_test in range(1,4):
    for max_error_designed in range(1,4):
        # mem
        row=["syndrome","mem","combined",max_error_designed,max_error_test]
        try:
            mem=%memit -c -q -r {runs} -o test_decode_combined(C,"syndrome")
            MEM=mem.mem_usage
        except DecodingError:
            MEM=[0]*runs
        row.extend(MEM)
        csvwriter.writerow(row)
        csvfile.flush()
        # time
        row=["syndrome","time","combined",max_error_designed, max_error_test]
        try:
            timing=%timeit -c -q -r {runs} -o test_decode_combined(C,"syndrome")
            TIME=timing.all_runs
        except DecodingError:
            TIME=[0]*runs
        row.extend(TIME)
        csvwriter.writerow(row)
        csvfile.flush()
    # mem
    row=["nearestNeighbor","mem","combined",max_error_designed,max_error_test]
    try:
        mem=%memit -c -q -r {runs} -o test_decode_combined(C,"nearestNeighbor")
        MEM=mem.mem_usage
    except DecodingError:
        MEM=[0]*runs
    row.extend(MEM)
    csvwriter.writerow(row)
    csvfile.flush()
    # time
    row=["nearestNeighbor","time","combined",max_error_designed,max_error_test]
    try:
        timing=%timeit -c -q -r {runs} -o test_decode_combined(C,"nearestNeighbor")
        TIME=timing.all_runs
    except DecodingError:
        TIME=[0]*runs
    # time
    row.extend(TIME)

```

Listing 4.33: Log the current memory usage of the system

```
MEMLOG=mem_ \$(date --iso-8601=minutes).log; echo "$(date) started  
→ before sage test" > $MEMLOG; free --seconds 60 --bytes >>  
→ $MEMLOG & tail -f $MEMLOG
```

```
        csvwriter.writerow(row)
        csvfile.flush()
    csvfile.close()

# Common settings
runs=3
# https://www.cyberciti.biz/faq/howto-get-current-date-time-in-python/
import time

# files
import csv

# error
from sage.coding.decoder import DecodingError

# Initialize small code
rundate=time.strftime("%y-%m-%dT%H:%M+0200")
F8=GF(2**3)
RR.<xx> = F8[]
g = xx^2+xx+1
L = [a for a in F8.list() if g(a)!=0]
C = codes.GoppaCode(g, L)

csvfile=open('./test_small_goppa_code_log'+rundate+'.csv','wb')

test_suite(code=C,csvfile=csvfile,runs=runs)

# Initialize big code
rundate=time.strftime("%y-%m-%dT%H:%M+0200")
F8=GF(2**6)
RR.<xx> = F8[]
g = xx^9+1
L = [a for a in F8.list() if g(a)!=0]
C = codes.GoppaCode(g, L)

csvfile=open('./test_big_goppa_code_log'+rundate+'.csv','wb')

test_suite(code=C,csvfile=csvfile)
```

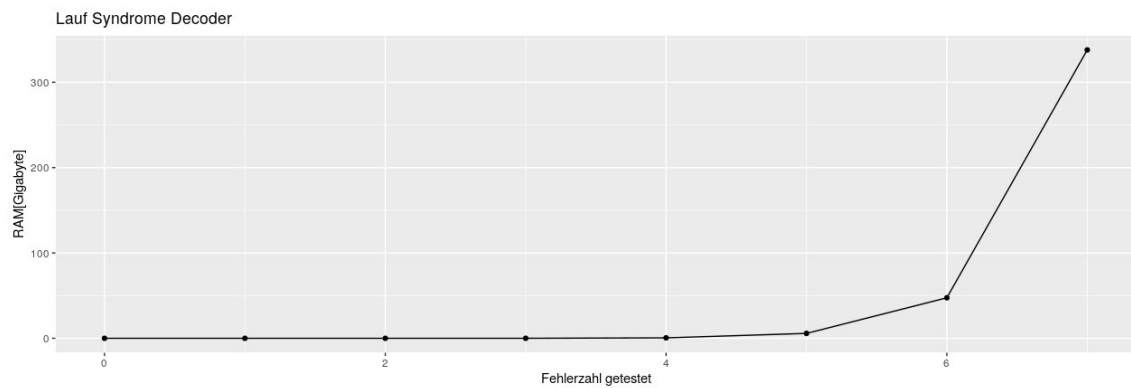



Abbildung 4.1: [Syndromedecoder: RAM - maximum_error_weight] Die Graphik zeigt die Auslastung der Arbeitsspeichers des Testsystems beim Anlegen der lookup_table

Kapitel 5

Schlussfolgerung

Mit den Ergebnissen von ?? und ?? ist ersichtlich, dass die Implementierung möglich und in diesem Fall auch korrekt ist.

Auf dieser Implementierung aufsetzend, können mathematisch-bewiesene Cryptosysteme aufgebaut werden; mehr dazu findet man z.B. in [97] [68, Kapitel 3] [140, McEliece Cryptosystem]

Für die Speicherprobleme aus dem Test sind bereits Verbesserungen geplant [126]. .



Anhang A

Mathematischer Anhang

Im Folgenden werden einige Beweise und Sätze aufgeführt, die als Hilfsmittel benötigt werden.

A.1 Homomorphiesatz

A.1.1 Theorem (Erster Isomorphie/ Homomorphiesatz [65, A.24])

Seien A und B Gruppen, Ringe oder Vektorräume und sei $\Phi : A \rightarrow B$ ein Homomorphismus. Dann gilt:

$$A / \ker \Phi \simeq \Phi(A).$$

Beweis. Sei alles wie eben definiert und sei $\pi : A \rightarrow A / \ker \Phi$ die natürliche Projektion, die jedes Element aus A auf seine Restklasse in $A / \ker \Phi$ abbildet und $\bar{\Phi} : A / \ker \Phi \rightarrow \Phi(A)$ die Einschränkung von Φ auf $A / \ker \Phi$. π ist surjektiv und $A / \ker \Phi \subseteq A$ (über diese Einbettung), also ist $\bar{\Phi} \circ \pi$ definiert. Die obere Sequenz ist exakt und das linke Dreieck kommutiert, da $\Phi \circ \iota (\ker \Phi) = \Phi(\ker \Phi) = 0$. Die gesuchte Abbildung existiert, über eine klassische Diagrammjagd. π ist surjektiv, also gibt

$$\begin{array}{ccccccc}
 0 & \longrightarrow & \ker \Phi & \xrightarrow{\quad \iota \quad} & A & \longrightarrow & A / \ker \Phi \longrightarrow 0 \\
 & & & \searrow 0 & \downarrow \pi & & \nearrow \exists! \bar{\Phi} \\
 & & & & \Phi(A) & &
 \end{array}$$

Abbildung A.1: Diagrammjagd
[131]

für alle $[a] \in A/\ker \Phi$ ein $a \in A$, sodass $\pi(a) = [a] \in A/\ker \Phi$. Betrachtet man jetzt $\Phi(a) \in \Phi(A)$ sehen sieht man die Existenz, der Abbildung $\text{Iso} : A/\ker \Phi \rightarrow \Phi(A)$. Sei jetzt $[a] = [b] \in A/\ker \Phi$ ein weiterer Vertreter der gleichen Restklasse, dann gilt $\text{Iso}([a]) \stackrel{\text{def } \pi}{=} \text{Iso}(\pi(a)) \stackrel{\text{def } \Phi}{=} \Phi(a) = \Phi(a) + 0 \stackrel{x \in \ker \Phi}{=} \Phi(a) + \Phi(x) \stackrel{\text{Homomorphismus}}{=} \Phi(a+x) \stackrel{a=b+x, x \in \ker \Phi}{=} \Phi(b) \stackrel{\text{def } \Phi}{=} \text{Iso}(\pi(b)) \stackrel{\text{def } \pi}{=} \text{Iso}([b])$. Da $\Phi : A \rightarrow \Phi(A)$ surjektiv ist, kann man diese Konstruktion auch umkehren und erhält die Isomorphie.

□

A.2 Riemann-Roch Theorem

A.2.1 Theorem (Riemann-Roch Theorem [65, S. 5.14])

Sei \mathcal{C} eine nicht singuläre projektive Ebenenkurve von Genus g , die über \mathbb{F}_q definiert ist und D ein Divisor auf \mathcal{C} . Dann $\dim \mathcal{L}(D) \geq \deg D + 1 - g$. Wenn auch $\deg D > 2g - 2$, dann

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

Für den Beweis dieses Theorems sei auf *Algebraic geometry* [58, S. IV.1] verwiesen.

Anhang B

Werkzeuge

Für diese Arbeit wurden desweiteren folgende Werkzeuge verwendet:

B.1 Das Dokument

Für das setzen dieses Dokuments wurden folgende Programme verwendet:

vim mit **vimplug:vimtex** als Editor.

für die Versionskontrolle, die es immer wieder möglich gemacht hat, bei Problemen auf einen funktionierenden Stand zu springen.

Gitea Version: 1.9.0 für das RemoteRepository und das sicherstellen, dass selbst im Falle eines fatalen Hardware Schadens nicht alles verloren ist.

Ein **git-hook** von mKaloer:pre-commit, der sicherstellt, dass nur sauberer Code im Commitet werden kann, einige Zeilen von [151]

Für die technische Entwicklung dieser Arbeit waren etliche Beiträge von [22] hilfreich, diese werden aber zu Gunsten der Übersichtlichkeit nur im Quellcode dieser Arbeit [120] verlinkt.

Auch die entsprechenden Anleitungen der \LaTeX Pakete waren eine grosse Hilfe; hier sei auf die Auflistung der Pakete verwiesen ??.

Ebenso sei auf die ausführlichen Anleitungen auf Deutsch [112] und Englisch [**sagemath:doc:en**], auf deren Einträge und Autoren der Kürze halber nicht einzeln eingegangen wird, verwiesen.

Zum schnellen Nachschlagen von Befehlen und groben Konzepten, waren [118] und [149], BZW. [150] sehr hilfreich.


Die Graphiken wurden mit Hilfe von *R: A Language and Environment for Statistical Computing*[133] und den Paketen [76, 100, 84, 99, 86, 83, 98]

- biber
- jabref

- FreeBSD
- tikz
- pdgplots
- latexmk
- git
- gnu-make

Die **vim** plugins **vimplug:gutentags** zum Springen durch den Code des **SAGE**-Projekts **vimtex** für die tolle Arbeitsumgebung **YouCompleteMe** für die Autovervollständigung im Besonderen

Abbildungsverzeichnis

| | | |
|------|---|----|
| 1.1 | Morsecode Tree  | 5 |
| 1.2 | Falschfarben Bild von Triton. Courtesy NASA/JPL-Caltech | 6 |
| 2.1 | Schnitt der hyperbel mit der Gerade $y = 0$: Plot mit mit <i>sagemath-8.9.beta3</i> | 15 |
| 2.2 | Geraden durch $??$: Plot mit mit <i>sagemath-8.9.beta3</i> | 16 |
| 2.3 | rationale Lösungen der Gleichung $x^2 - 2y^2 = 0$: Plot mit mit <i>sagemath-8.9.beta3</i> | 17 |
| 2.4 | Cusp: Plot \mathcal{C}_f für $f : ??$ mit <i>sagemath-8.9.beta3</i> | 22 |
| 2.5 | Node: Plot \mathcal{C}_g für $g : ??$ mit <i>sagemath-8.9.beta3</i> | 22 |
| 2.6 | Triplepoint: Plot \mathcal{C}_h für $h : ??$ mit <i>sagemath-8.9.beta3</i> | 22 |
| 2.7 | Tachnode: Plot \mathcal{C}_i für $i : ??$ mit <i>sagemath-8.9.beta3</i> | 22 |
| 2.8 | Plot \mathcal{C}_f für $f : ??$ mit <i>sagemath-8.9.beta3</i> | 22 |
| 2.9 | Plot \mathcal{C}_g für $g : ??$ mit <i>sagemath-8.9.beta3</i> | 22 |
| 2.10 | Plot \mathcal{C}_h für $h : ??$ mit <i>sagemath-8.9.beta3</i> | 22 |
| 4.1 | [Syndromedecoder: RAM - <code>maximum_error_weight</code>]Die Graphik zeigt die Auslastung der Arbeitsspeichers des Testsystems beim Anlegen der <code>lookup_table</code> | 67 |
| A.1 | Diagrammjagd | 69 |

Listings

| | | |
|------|--|----|
| 4.1 | Funktionsdefinition [94, S. 86–89] | 47 |
| 4.2 | EXAMPLE[94, S. 86–92] | 47 |
| 4.3 | Funktionsdefinition [89, Zeilen:39-42] | 47 |
| 4.4 | Festlegen von L [89, Zeilen:24] | 47 |
| 4.5 | [89, Zeilen: 43] Aufruf von GoppaCode | 48 |
| 4.6 | goppa.py | 48 |
| 4.7 | Methoden der KlasseGoppaCode[89, Zeilen:50, 70, 86, 102, 182, 220] | 48 |
| 4.8 | Methoden der KlasseAbstractLinearCode[93, Zeilen:336, 414, 565, 580, 625, 670, 686, 748, 810, 869, 881, 990, 1003, 1032, 1085, 1125, 1193, 1211, 1223, 1244, 1307, 1347, 1389, 1427, 1463, 1532, 1562, 1584, 1628, 1647, 1690, 1647, 1690, 1722, 1740, 1769, 1802, 1861, 1895, 1915, 1964, 2010, 2088, 2116, 2088, 2116, 2135, 2176, 2279, 2303, 2337, 2350, 2374, 2399, 2427, 2458, 2489, 3490, 3527, 3586, 3603, 3671,] | 48 |
| 4.9 | Überschreiben der __ini__ Methode[89, Zeilen:57, 244, 310] | 50 |
| 4.10 | abstrakte __init__ Methode[93, Zeilen:414-415, 540–563] | 50 |
| 4.11 | generator_matrix[89, Zeilen:285, 305–308] | 51 |
| 4.12 | from_parity_check_matrix [90, Zeilen:479, 499–500] | 51 |
| 4.13 | dual_code [93, Zeilen:1628, 1645] | 51 |
| 4.14 | parity_check_matrix [93, Zeilen:1308, 1341–1345] | 52 |
| 4.15 | paritycheckmatrix[89, S. 122, 151–157] | 53 |
| 4.16 | goppa.py | 53 |
| 4.17 | goppa.py | 53 |
| 4.18 | goppa.py | 54 |
| 4.19 | goppa_code.py [108, Zeilen:13-15] | 54 |
| 4.20 | Import der Decoder [95, Zeilen:70-72] | 54 |
| 4.21 | Import der Encoder [96, Zeilen:58] | 55 |
| 4.22 | encoder.py: class Encoder | 55 |
| 4.23 | linear_code.py | 55 |
| 4.24 | Übersicht LinearCodeNearestNeighborDecoder [93, Zeilen:4853, 4863, 4876, 4891, 4905, 4919, 4950] | 56 |

| | | |
|------|--|----|
| 4.25 | Klasse <code>LinearCodeNearestNeighborDecoder</code> <code>decode_to_code</code> [93, Zeilen:4919, 4941–4948] | 56 |
| 4.26 | Übersicht <code>LinearCodeSyndromeDecoder</code> [93, Zeilen:4413, 4540, 4580, 4596, 4610, 4624, 4638, 4761, 4799, 4816, 4831] | 56 |
| 4.27 | <code>__init__</code> des <code>LinearCodeSyndromeDecoder</code> [93, Zeilen:4540, 4567–4578] | 57 |
| 4.28 | Variablen der <code>_build_lookup_table</code> des <code>LinearCodeSyndromeDecoder</code> [93, Zeilen:4638–4639, 4691–4702] | 57 |
| 4.29 | Erzeugung der <code>_build_lookup_table</code> des <code>LinearCodeSyndromeDecoder</code> [93, Zeilen:4703–4725] | 58 |
| 4.30 | <code>__init__</code> des <code>LinearCodeSyndromeDecoder</code> [93, Zeilen:4540, 4567–4578] | 59 |
| 4.31 | <code>dmesg</code> : OOM Killer | 60 |
| 4.32 | Ipython Test | 61 |
| 4.33 | memory logging | 66 |
| B.2 | <code>scripty.sty</code> | 75 |
| B.1 | <code>remote_sage.sh</code> | 76 |

B.2 Eigene Codefragmente

Listing B.2: Dieses Skript lädt Code aus einem Git Repository, um per `lstinputlisting` aus dem Paket *listings* eingebunden zu werden.

```
%! TeX root=NONE
% this file is to contain reusable functions

% Additional document hooks
\RequirePackage{etoolbox}
\RequirePackage{todolist}

% robust commands
\RequirePackage{xparse}

% this is set as fallback
% Use New to warn if broken
\NewDocumentCommand{\GitRemote}{}{}
\NewDocumentCommand{\GitIdentifier}{}{}{master}

\NewExpandableDocumentCommand{\GitCheckout} {O{\GitIdentifier} O{\GitRemote} m}
  ↪ {\%
|"git archive --remote=#2 #1 #3 2>/dev/null | tar --extract --file - --to-stdout"
}

% empty lines ( without %) break this code
```

Listing B.1: Dieses Skript wird genutzt, um ein SAGE Code auf einem entfernten System auszuführen.

```
#!/bin/sh
# script to deploy sage file to remote host
# run it there
# and get the corresponding output back

# https://stackoverflow.com/questions/242538/unix-shell-script-find-out-which-directory-
#   ↳ the-script-file-resides
# Absolute path to this script, e.g. /home/user/bin/foo.csh
SCRIPT='readlink -f "$0"'
# Absolute path this script is in, thus /home/user/bin
SCRIPTPATH='dirname "$SCRIPT"'
echo $SCRIPTPATH

# input
SAGE_FILE="$1"

# default files
SAGE_RETURN="{ $SAGE_FILE.sagetex.s{cmd,out},sage-plots-for-$SAGE_FILE.tex}"


# setup
. $SCRIPTPATH/env.rc
# run



rsync ${RSYNCOPTS:} "build/$SAGE_FILE.sagetex.sage" "$REMOTE_HOST:
#   ↳ $REMOTE_PATH/$SAGE_FILE.sagetex.sage"
ssh ${SSHOPTS:} "$REMOTE_HOST" "$REMOTE_SAGE" "$REMOTE_PATH/
#   ↳ $SAGE_FILE.sagetex.sage"
rsync ${RSYNCOPTS:} --quiet --recursive "$REMOTE_HOST:$REMOTE_PATH/
#   ↳ $SAGE_RETURN" ./
```




```
\AtEndDocument{%
  \nocite{stack:tex:immediate-write}
  % https://tex.stackexchange.com/questions/20444/what-are-immediate-write18-and
  % ↪ -how-does-one-use-the
  \nocite{stack:overflow:git-in-another-folder}
  % https://stackoverflow.com/questions/3769137/use-git-log-command-in-another-
  % ↪ folder
  % the -C flag is useful to work as in another dir
  \nocite{stack:overflow:checkout-tags}
  % https://stackoverflow.com/questions/35979642/what-is-git-tag-how-to-create-
  % ↪ tags-how-to-checkout-git-remote-tags
  \nocite{texwelt:shellescape}
  % https://texwelt.de/wissen/fragen/15887/was-ist-write18-oder-shell-escape
  \nocite{man:gnu-tar}
  % https://www.gnu.org/software/tar/manual/html_node/Writing-to-Standard-
  % ↪ Output.html
  \nocite{stack:tex:input-from-web}
  % https://tex.stackexchange.com/questions/249567/is-it-possible-to-input-a-file-
  % ↪ from-the-web
  \nocite{stack:tex:shellescape-nested}
  % https://tex.stackexchange.com/questions/501640/shellescape-macro-nested-in-
  % ↪ another-macro?noredirect=1#comment1266953_501640

  \nocite{stack:tex:shellescape-nested:comment-new-vs-provide}
  % https://tex.stackexchange.com/questions/501640/shellescape-macro-nested-in-
  % ↪ another-macro/501873#501873
  \nocite{stack:tex:more-than-one-option}
  % https://tex.stackexchange.com/questions/29973/more-than-one-optional-argument
  % ↪ -for-newcommand
  \nocite{golatex:order-arguments}
  % https://golatex.de/viewtopic,p,92725.html
  \nocite{stack:tex:xparse-passing-arguments}
  % https://tex.stackexchange.com/questions/188313/passing-arguments-using-xparse
  \nocite{stack:tex:multiple-optional-arguments}
  % https://tex.stackexchange.com/questions/448542/how-to-define-a-command-with
  % ↪ -two-optional-arguments
  \nocite{stack:tex:behaviour-optional-args}
  % https://tex.stackexchange.com/questions/308/different-command-definitions-with-
  % ↪ and-without-optional-argument/2925#2925
}% end AtEndDocument
```

Literatur

- [53] C. E. Shannon. „A Mathematical Theory of Communication“. In: *Bell System Technical Journal* 27.4 (Okt. 1948), S. 379–423, 623–656. ISSN: 0005-8580. DOI: 10.1002/j.1538-7305.1948.tb00917.x.
- [54] Richard Wesley Hamming. „Error Detecting and Error Correction Codes“. In: (Apr. 1950). URL: <https://signallake.com/innovation/hamming.pdf> (besucht am 26.08.2019). 
- [55] E. N. Gilbert. „A Comparison of Signalling Alphabets“. In: 31.3 (Mai 1952), S. 504–522. DOI: 10.1002/j.1538-7305.1952.tb01393.x.
- [56] Irving S Reed und Gustave Solomon. „Polynomial codes over certain finite fields“. In: 8.2 (1960), S. 300–304.
- [57] Valerii Denisovich Goppa. „Codes associated with divisors“. In: *Problemy Peredachi Informatsii* 13.1 (1977), S. 33–39.
- [58] Robin Hartshorne. *Algebraic geometry*. Bd. 52. Graduate texts in mathematics. Springer, 1977, S. xvi+496. ISBN: 978-3-540-90244-7. DOI: 10.1007/978-1-4757-3849-0.
- [59] Florence Jessie MacWilliams und Neil James Alexander Sloane. *The theory of error-correcting codes*. Bd. 16. North Holland Publishing Co., 1977. ISBN: 0-444-85193-3.

- [60] Marvin Perlman und Ju-Ji Lee. „A comparison of conventional Reed-Solomon encoders and Berlekamp’s architecture“. In: *NASA Tech. Brief* 3610-81 (1982), S. 119.
- [61] Steven Roman. *Coding and Information Theory*. GTM 134. Berlin Heidelberg: Springer New York, 4. Juni 1992. 512 S. ISBN: 978-0-387-26456-1.
- [62] Robert J McEliece und Laif Swanson. *Reed-Solomon codes and the exploration of the solar system*. Piscataway, NJ: IEEE Press, 20. Aug. 1993. (Besucht am 18.09.2019).
- [63] H Stichtenoth. *Algebraic Function Fields and Codes*. New York: Springer-Verlag, 1993.
- [64] T. Hoholdt und R. Pellikaan. „On the decoding of algebraic-geometric codes“. In: 41.6 (Nov. 1995), S. 1589–1614. ISSN: 0018-9448. DOI: 10.1109/18.476214.
- [65] Judy L. Walker. *Codes and Curves*. Bd. 7. Student mathematical library. American Mathematical Society, 17. Mai 1999. ISBN: 978-0-8218-2628-7. DOI: 10.1090/stml/007. (Besucht am 05.09.2019).
-  [66] Chris Lomont. „Error Correcting Codes on Algebraic Surfaces“. 7. Sep. 2003. URL: <https://arxiv.org/pdf/math/0309123.pdf> (besucht am 02.04.2019).
- [67] David J. C. MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Pr., 1. Sep. 2003. ISBN: 0521642981.
-  [68] Key One Chung. „Goppa Codes“. English. Department of Mathematics, Iowa State University, Ames, Iowa 50011-2064, Dez. 2004. URL: <https://orion.math.iastate.edu/linglong/Math690F04/Goppa%20codes.pdf> (besucht am 09.07.2019).

- [69] ISBN-Agentur f. d. Bundesrepublik Deutschland. *ISBN-Handbuch - die Internationale Standard-Buchnummer ; [der ISO-Standard 2108]*. 1. Aufl. Frankfurt: ISBN-Agentur für die Bundesrepublik Deutschland, 2005. ISBN: 978-3-765-72781-8. (Besucht am 01.08.2019).
- [70] Todd K. Moon. *Error Correction Coding. Mathematical Methods and Algorithms*. Wiley, Mai 2005. DOI: 10.1002/0471739219.
- [71] Zhuo Jia Dai. „Algebraic Geometric Coding Theory“. ACCESSED:2019-04-16. 2006. URL: https://commons.wikimedia.org/wiki/File:Algebraic%5C_Geometric%5C_Coding%5C_Theory.pdf (besucht am 09.07.2019). 
- [72] Richard E. Blahut. *Algebraic Codes on Lines, Planes, and Curves*. Cambridge University Press, 2008. DOI: 10.1017/cbo9780511543401. (Besucht am 16.09.2019).
- [73] Rachel Pries und Beth Malmskog. „Maximal Curves and Application to Coding Theory“. 2008. URL: <https://www.math.colostate.edu/~pries/costaricalectures.pdf> (besucht am 09.07.2019). 
- [74] T Hohold, J van Lint und R Pellikaan. *Algebraic geometry codes*. 2011. Aufl. as 32 from Little2008. Elsevier, 20. Sep. 2011, S. 871–962. (Besucht am 09.07.2019).
- [75] Florian Modler und Martin Kreh. *Differenzierbarkeit*. Springer, 1. Jan. 2011. ISBN: 978-3-8274-2830-1. DOI: 10.1007/978-3-8274-2831-8_11.
- [76] Hadley Wickham. „The Split-Apply-Combine Strategy for Data Analysis“. In: 40.1 (2011), S. 1–29. URL: <http://www.jstatsoft.org/v40/i01/> (besucht am 21.09.2019). 
- [77] Ron M. Roth. *Introduction to Coding Theory*. Cambridge University Press, 18. Jan. 2012. 580 S. ISBN: 0521845041. (Besucht am 15.09.2019).

- [78] Nicola L. C. Talbot. *Using Latex to Write a PhD Thesis*. Dickimaw Books, 4. März 2013. 162 S. ISBN: 1909440027. (Besucht am 17.09.2019).
- [79] Irene I. Bouw. „Codierungstheorie, Vorlesungsskript“. 2014. URL: https://www.uni-ulm.de/fileadmin/website_uni_ulm/mawi.inst.100/vorlesungen/ss14/Codierungstheorie/CT14.pdf (besucht am 01.08.2019).
- [80] Jop Briët u.a. „Lower Bounds for Approximate LDC“. In: [abs/1402.6952](https://arxiv.org/abs/1402.6952) (2014). URL: <http://arxiv.org/abs/1402.6952> (besucht am).
- [81] San Ling und Chaoping Xing. *Coding Theory*. Cambridge University Press, 3. Juni 2014. 236 S. ISBN: 0521529239. (Besucht am 14.09.2019).
- [82] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. (Besucht am 21.09.2019).
- [83] Hadley Wickham. *gridExtra: Miscellaneous Functions for "Grid" Graphics*. R package version 2.3. 2017. URL: <https://CRAN.R-project.org/package=gridExtra> (besucht am 21.09.2019).
- [84] Hadley Wickham, Jim Hester und Romain Francois. *readr: Read Rectangular Text Data*. R package version 1.3.1. 2018. URL: <https://CRAN.R-project.org/package=readr> (besucht am 21.09.2019).
- [85] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.9.beta3)*. Version 8.9.beta3. 2019. URL: <https://sagemath.org> (besucht am 17.09.2019).
- [86] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.9.beta9)*. Version 8.9.beta9. 2019. URL: <https://sagemath.org> (besucht am 17.09.2019).

- [89] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [90] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [91] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [92] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [93] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [94] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [95] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [96] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version x.y.z)*. Version (x.y.z). 2019. URL: <https://sagemath.org> (besucht am 17. 09. 2019).



- [97] Thomas Risse. „How SAGE helps to implement Goppa Codes and McEliece PKCSs“. Apr. 2019.



- [98] Charlie Sharpsteen und Cameron Bracken. *tikzDevice: R Graphics Output in LaTeX Format*. R package version 0.12.3. 2019. URL: <https://CRAN.R-project.org/package=tikzDevice> (besucht am 21.09.2019).



- [99] Hadley Wickham und Lionel Henry. *tidyr: Tidy Messy Data*. R package version 1.0.0. 2019. URL: <https://CRAN.R-project.org/package=tidyr> (besucht am 21.09.2019).



- [100] Hadley Wickham u.a. *dplyr: A Grammar of Data Manipulation*. R package version 0.8.3. 2019. URL: <https://CRAN.R-project.org/package=dplyr> (besucht am 21.09.2019).



- [102] URL: https://vim.fandom.com/wiki/Increasing_or_decreasing_numbers (besucht am 23.09.2019).



- [103] URL: <https://dict.leo.org/englisch-deutsch/echelon%20matrix> (besucht am 31.08.2019).



- [104] URL: <https://tex.stackexchange.com/questions/52576/difference-between-bookmark-levels-greater-than-one-what-does-this-mean> (besucht am 23.09.2019).












- [105] URL: <https://tex.stackexchange.com/questions/226481/appendix-section-title> (besucht am 23.09.2019).



- [106] URL: <https://tex.stackexchange.com/questions/150492/how-to-use-itemize-in-table-environment> (besucht am 23.09.2019).



- [108] gh-927589452. *minor improvements*. URL: <https://trac.sagemath.org/ticket/28381> (besucht am 22.08.2019).

- [110] Heinrich Apfelmus. *Fun with Morse Code*. URL: <https://apfelmus.nfshost.com/articles/fun-with-morse-code.html> (besucht am 05.08.2019). 
- [111] Kevin Buzzard. *M345P11: Existence of algebraic closure of a field*. URL: <http://wwwf.imperial.ac.uk/~buzzard/maths/teaching/15Aut/M3P11/algclosure.pdf> (besucht am 05.09.2019). 
- [112] *Deutsch sprachige Anleitungen zu SAGE*. URL: <http://doc.sagemath.org/html/de/> (besucht am 08.09.2019). 
- [116] *git version 2.22.0*. URL: <https://git-scm.com/> (besucht am 23.09.2019). 
- [117] *Gitea Version: 1.9.0*. URL: <https://gitea.io/en-us/> (besucht am 23.09.2019). 
- [120] Jens Heinrich. *Analyse der Implementierung von algebraisch-geometrischen Codes*. URL: https://github.com/s0160187/bachelor_thesis (besucht am 21.09.2019). 
- [121] Jens Heinrich. *Gemessene Daten*. URL: https://github.com/s0160187/bachelor_thesis (besucht am 21.09.2019). 
- [122] *Homomorphiesatz - Mathepedia*. URL: <https://mathepedia.de/Homomorphiesatz.html> (besucht am 14.07.2019). 
- [126] J und Dimpanse. *When to protect the user - Implementing a sanity check for the syndrome decoder*. URL: <https://groups.google.com/d/msgid/sage-coding-theory/20190922081126.braabdrnez65ymjb%40deathbolt.927589452.space> (besucht am 23.09.2019). 



- [128] Roger Ludwig und Jim Taylor. *Voyager Telecommunications*. NASA. URL: https://voyager.gsfc.nasa.gov/Library/DeepCommo_Chapter3--141029.pdf (besucht am 18.09.2019).



- [130] *Morsecode tree*. URL: https://commons.wikimedia.org/wiki/File:Morse_code_tree3.png (besucht am 05.08.2019).



- [131] *MP: Kategorien (2) und Diagrammjagd (Matroids Matheplanet)*. URL: <https://matheplanet.de/matheplanet/nuke/html/article.php?sid=790> (besucht am 14.07.2019).



- [132] NASA/JPL-Caltech. URL: <https://photojournal.jpl.nasa.gov/tiff/PIA00060.tif> (besucht am 27.08.2019).



- [133] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. URL: <https://www.R-project.org> (besucht am).



- [134] *Richard W. Hamming. United States – 1968*. Additional Materials. Association for Computing Machinery. URL: http://amturing.acm.org/info/hamming_1000652.cfm (besucht am 17.09.2019).



- [135] Thomas Risse. „Generating Goppa Codes“. URL: <https://pdfs.semanticscholar.org/7b02/d4a611b27a11c323507ad31a7947f4c02b21.pdf> (besucht am 19.08.2019).



- [136] *Sage Quickstart for Linear Algebra — PREP Tutorials v8.8*. MAA PREP Workshop “Sage: Using Open-Source Mathematics Software with Undergraduates” (funding provided by NSF DUE 0817071). URL: <http://doc.sagemath.org/html/en/prep/Quickstarts/Linear-Algebra.html> (besucht am 21.08.2019).



- [138] Marketa Slukova. *Community Bonding and First Week*. URL: <https://medium.com/@em.slukova/gsoc-week-two-and-three-93364102338c> (besucht am 15.08.2019).

- [139] Markéta Sluková. *GSoC Final Coding Period*. URL: <https://medium.com/@em.slukova/gsoc-final-coding-period-22b671ddcae8> (besucht am 15.08.2019).



- [140] Markéta Sluková. *GSoC Final Report*. URL: <https://github.com/emes4/gsoc-finalreport> (besucht am 22.09.2019).



- [141] Markéta Sluková. *GSoC Rank Metric McEliece Cryptosystem*. URL: <https://medium.com/@em.slukova/gsoc-rank-metric-mceliece-cryptosystem-e28cd3701ba5> (besucht am 15.08.2019).



- [142] Markéta Sluková. *GSoC Second Coding Period*. URL: <https://medium.com/@em.slukova/gsoc-second-coding-period-eb3ebb179000> (besucht am 15.08.2019).



- [143] Markéta Sluková. *GSoC Week Two and Three - Markéta Sluková - Medium*. URL: <https://medium.com/@em.slukova/gsoc-week-two-and-three-93364102338c> (besucht am 15.08.2019).



- [144] William Stein u. a. *Cached Functions and Methods*. URL: <http://doc.sagemath.org/html/en/reference/misc/sage/misc/cachefunc.html> (besucht am 20.09.2019).



- [145] Dr Nicola Talbot. *Dickimaw Books*. URL: <https://www.dickimaw-books.com/software/bib2gls/> (besucht am 04.09.2019).



- [146] TeX-FAQ, Hrsg. *How to approach errors*. URL: <https://texfaq.org/FAQ-erroradvice> (besucht am 21.09.2019).



[52]

[52]

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Frankfurt am Main, den 23. September 2019,

(Jens Heinrich)