MASTER

Hamming-Metric Codes for Quantum Channels

Bachoríková, Lenka

Award date:
2022

Master Industrial and Applied Mathematics
Cryptography and Coding Theory

Hamming-Metric Codes for Quantum Channels

Master Thesis

Lenka Bachoríková
21 July 2022

Supervision:
E. Berardini
A. Ravagnani

Assessment Committee:
A. Abiad
E. Berardini
A. Ravagnani

Credits: 30

Abstract

In this thesis, CSS error-correcting codes are studied. The definition of a CSS code is generalized for any field with q elements, and it is shown that CSS codes are also error-correcting over these fields. Compared to the original definition by Calderbank and Shor, the requirements on dimension and minimum distance are relaxed. The main result of this thesis is an approximation of the number of error-correcting CSS codes which is based on a lower bound.

Contents

Introduction

Quantum error correction is the field of Coding Theory that deals with protecting quantum information. The importance of quantum error correction arose with the development of quantum computers, which are computers that operate with *qubits* instead of bits. They are much more efficient than classical computers for certain computational tasks, such as integer factorization [7]. However, compared to classical computers, quantum computers require error correction. Errors occur during quantum computing due to quantum noise and interference from the environment [1]. Therefore, quantum error-correcting codes were developed to correct these errors.

CSS codes are a family of quantum codes which are derived from two binary linear codes $C_2 \subseteq C_1$. They were first introduced by Calderbank and Shor in 1996 [4]. Around the same time, Steane worked on the same subject independently of Calderbank and Shor and he introduced these codes with a different definition [15]. This later turned out to be equivalent to the construction by Calderbank and Shor and therefore these codes are named after the initials of the three researchers. Some work has been undertaken on CSS codes, considering different families of linear codes from which the quantum codes are derived. To mention some of them, there are examples of CSS codes generated from linear Reed-Muller codes [5, 14], from Reed-Solomon codes [6], from algebraic-geometric codes [2] and from LDPC codes [10].

CSS codes are quantum error-correcting codes, which means that they were designed to correct quantum errors. Calderbank and Shor prove in [4, Theorem 1] that if the binary codes C_1 and $C_2^\perp$ with $C_2 \subseteq C_1$ satisfy three conditions, then the CSS code Q_{C_1,C_2} is error-correcting. The three conditions are namely that C_1 and $C_2^\perp$ must have the same length n, the same dimension $n-k$ and the same minimum distance d. More precisely, if these three conditions are satisfied, the Q_{C_1,C_2} can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors. Many existing examples of error-correcting CSS codes are based on a self-orthogonal linear code C, however, self-orthogonality is not required by Calderbank and Shor. Such CSS codes can be seen for example in [8, Example 13.1.6 and 13.1.7] or in the Steane code [15].

In this thesis we generalize the construction of CSS codes presented by Calderbank and Shor. We provide a relaxation of the conditions in [4, Theorem 1], more precisely, we drop the condition on the minimum distance of C_1 and $C_2^\perp$, and we generalize CSS codes over larger fields using Steane's definition. Moreover, we show that there exist CSS codes which are not based on a self-orthogonal code and therefore self-orthogonality is not a necessary condition for CSS error-correcting codes.

Since self-orthogonality is not a requirement, the number of CSS codes is larger than the number of self-orthogonal linear codes. Therefore, we can ask the following question: *How many CSS codes are there?* In this thesis we derive a lower bound for the number of CSS codes for some fixed parameters based on results from the paper by Byrne and Ravagnani [3]. Besides, we implement a program in the SageMath computer algebra system to count the exact number of CSS codes, in order to see if the lower bound is a good approximation for the number of CSS codes.

Outline of the thesis

The content of the thesis is organized in six chapters. We first introduce classical linear error-correcting codes in Chapter 1 and then we proceed with defining quantum errors and quantum error-correcting codes in Chapter 2. In Chapter 3 we introduce the CSS construction over the binary field, both by Calderbank and Shor, and by Steane, and then we generalize CSS codes over non-binary fields. At the end of this chapter we discuss how binary CSS codes correct quantum errors and whether this is the case also for CSS codes over non-binary fields. Finally, in the last two chapters we present the results of this thesis. In Chapter 4 we prove that counting the number of CSS codes is the same as counting the pairs of linear codes for these CSS codes, and we derive a lower bound for the number of error-correcting CSS codes. Then, in Chapter 5 some results of the lower bound from Chapter 4 are compared to the real numbers of the CSS codes found by the SageMath computer algebra system. The thesis ends with a conclusion.

Chapter 1

Classical linear codes

In this chapter we give some basic definitions and propositions on linear error-correcting codes. This is necessary so that we can define quantum codes which are derived from linear codes. The Chapter is split in three sections. In Section 1.1, we introduce the basic notation that will be used throughout the thesis. In Section 1.2 we provide definitions on linear codes. Most of these definitions are standard and can be found in *Introduction to Coding Theory* by van Lint [16] or in the *Coding Theory* lecture notes by Ravagnani [12]. Finally, in Section 1.3 we present some propositions related to counting or approximating the number of linear codes. These will be used to approximate the number of CSS codes in Chapter 4. The theorems and propositions in this section are from the paper by Byrne and Ravagnani [3].

1.1 Notation

Throughout the thesis we use the notation $\mathbb{F}_q$ for the field with q elements where q is a power of a prime number p. We denote the field of complex numbers by $\mathbb{C}$. Matrices and vectors are denoted by uppercase and lowercase letters respectively, the zero vector is denoted by $\mathbf{0}$ and the identity matrix by I. The i-th entry of a vector v is denoted as $(v)_i$. A transposed vector of a vector v is denoted by v^T. The inner product of two vectors v_1, v_2 is denoted by $\langle v_1, v_2 \rangle$. More notation will be introduced together with the definitions.

1.2 Linear codes

In this section we provide standard definitions on linear error-correcting codes according to *Introduction to Coding Theory* by van Lint [16] or the *Coding Theory* lecture notes by Ravagnani [12].

Definition 1.2.1. A *linear code* (or a *code*) C is a linear subspace of $\mathbb{F}_q^n$, where n is a positive integer. The vectors $c \in C$ are called *codewords*. The *dimension* of a code C is its dimension as a linear subspace over $\mathbb{F}_q$. If a linear code C_2 is a subspace of a linear code C_1, we denote it by $C_2 \subseteq C_1$ (or by $C_2 \subsetneq C_1$ if $C_2 \neq C_1$) and say that C_2 is a *subcode of* C_1.

Definition 1.2.2. A *generator matrix* G of a linear code C is a full rank matrix with entries in $\mathbb{F}_q$ such that its rows generate the codewords of C.

Generator matrices are not unique, one code can have multiple generator matrices. However, we can select the standard generator matrix of a code by putting the matrix into reduced row echelon form. Since reduced row echelon form is unique, every code will have one unique standard generator matrix, which we will from now on call *the* generator matrix.

Remark 1.2.3. Note that a linear code C can also be defined as the set $C = \left\{ vG \mid v \in \mathbb{F}_q^{1\times \dim(C)} \right\}$, where G is the generator matrix of C.

In the following definitions we state some basic properties of linear codes.

Definition 1.2.4. The *distance* between two codewords $c_1, c_2 \in C$ is defined as

$$d(c_1, c_2) := |\{i \mid (c_1)_i \neq (c_2)_i\}|.$$

Definition 1.2.5. Let us take a code $C \subseteq \mathbb{F}_q^n$. The *support* of a codeword c denoted by $\mathrm{supp}(c)$ is defined as follows

$$\mathrm{supp}(c) := \{i \in \{1, \dots, n\} \mid (c)_i \neq 0\}.$$

If the support of one codeword a is contained in the support of another codeword b, we will denote it by $a \leq b$.

Definition 1.2.6. The *weight* of a codeword $c \in C$, denoted $\mathrm{wt}(c)$, is the size of its support, i.e.

$$\mathrm{wt}(c) := |\mathrm{supp}(c)|.$$

Note that the distance between the zero vector and a codeword c is the same as the weight of c, since we have $d(\mathbf{0}, c) = |i \mid (c)_i \neq 0| = \mathrm{wt}(c)$. For this reason we use weight in the following definition.

Definition 1.2.7. The *minimum distance* of a code C is defined as

$$d(C) := \min_{\mathbf{0} \neq c \in C} \{\mathrm{wt}(c)\}.$$

In Proposition 1.3.3, we will need to know the number of codewords that are within a certain distance around a fixed codeword. This is given in the next definitions.

Definition 1.2.8. Let $x \in \mathbb{F}_q^n$ and let r be a positive integer. The *ball of radius r around a vector $v \in \mathbb{F}_q^n$* is the set

$$B(x, r) := \{y \in \mathbb{F}_q^n \mid d(x, y) \leq r\} \subseteq \mathbb{F}_q^n.$$

We set $\boldsymbol{b}(r) := |B(x, r)|$ for any vector $x \in \mathbb{F}_q^n$.

Remark 1.2.9. Note that for a positive integer r we have

$$\begin{aligned}
\boldsymbol{b}(r) &= |B(x, r)| \quad \text{for any } x \in \mathbb{F}_q^n \\
&= |B(\mathbf{0}, r)| \\
&= |\{y \in \mathbb{F}_q^n \mid d(\mathbf{0}, y) \leq r\}| \\
&= |\{y \in \mathbb{F}_q^n \mid \mathrm{wt}(y) \leq r\}| \\
&= \sum_{i=0}^{r} (q-1)^i \binom{n}{i}.
\end{aligned}$$

This is because for a fixed weight i, there are $\binom{n}{i}$ options for how the i number of nonzero elements can be positioned in the n entries of a codeword, and for each of these positions we have $(q-1)$ options, namely on each of the nonzero i positions there can be $q-1$ nonzero elements.

1.3 Number of linear codes

In this section we introduce some propositions about the number of linear codes. Most of the results are from Byrne and Ravagnani [3]. We will use these propositions in Chapter 4 to approximate the number of CSS codes.

Proposition 1.3.1. Let n, k be fixed positive integers such that $k \leq n$. Then there exist

$$\binom{n}{k}_q = \frac{(q^n - 1)(q^n - q) \dots (q^n - q^{k-1})}{(q^k - 1)(q^k - q) \dots (q^k - q^{k-1})}$$

codes $C \subseteq \mathbb{F}_q^n$ of dimension $\dim(C) = k$.

Proof. First we find the number of all linearly independent sets $\{v_0, v_1, \ldots, v_{k-1}\}$ of $\mathbb{F}_q^n$ consisting of k vectors. To construct such a set, we have $(q^n - 1)$ options to pick v_0, since there are q^n vectors in $\mathbb{F}_q^n$, hence $q^n - 1$ nonzero vectors. For v_1 we have exactly $q^n - (q-1) - 1 = q^n - q$ options, since we have to exclude $(q-1)$ vectors linearly dependent to v_0 and the zero vector. Similarly, for picking vector v_2 we have $q^n - q^2$ options, and so on. Since we are picking k vectors, the number of such sets is

$$(q^n - 1)(q^n - q) \ldots (q^n - q^{k-1}).$$

Note that this number describes the ordered sets, so for example $\{v_0, v_1, \ldots, v_{k-1}\}$ and $\{v_1, v_0, \ldots, v_{k-1}\}$ are counted as two different sets. Hence we still have to exclude the sets that consist of the same k vectors. There are $k!$ re-orderings of the k vectors v_i, hence we have that there are

$$M = \frac{(q^n - 1)(q^n - q) \ldots (q^n - q^{k-1})}{k!}$$

linearly independent subsets of $\mathbb{F}_q^n$ consisting of k elements.

Fix one code $C \subseteq \mathbb{F}_q^n$ of dimension k, then there are different linearly independent sets of vectors that span C, say N. Since C has dimension k and hence q^k elements, then by the same logic as above, we have that

$$N = \frac{(q^k - 1)(q^k - q) \ldots (q^k - q^{k-1})}{k!}.$$

Dividing M by N gives the number of generator sets, that generate a different k dimensional code, and this is exactly

$$\frac{(q^n - 1)(q^n - q) \ldots (q^n - q^{k-1})}{(q^k - 1)(q^k - q) \ldots (q^k - q^{k-1})},$$

which is the same as $\binom{n}{k}_q$. $\square$

The following two propositions are from the paper by Byrne and Ravagnani [3]. They approximate the number of codes that share the same fixed subcode.

Proposition 1.3.2 (Proposition 2.5 from [3])**.** Let $C_2 \subseteq \mathbb{F}_q^n$ be a linear code with $\dim(C_2) = k_2 < n$. Fix an integer k_1 such that $k_2 \leq k_1 \leq n$ and set $\mathcal{F}_{k_1} := \{C_1 \subseteq \mathbb{F}_q^n \mid C_2 \subseteq C_1, \dim(C_1) = k_1\}$. Then

$$|\mathcal{F}_{k_1}| = \binom{n - k_2}{k_1 - k_2}_q.$$

Proposition 1.3.3 (Theorem 5.1 from [3])**.** Let $C_2 \subseteq \mathbb{F}_q^n$ be a linear code with $\dim(C_2) = k_2$ such that we have $0 \leq k_2 < n$ and such that $d(C_2) \geq 2$. Let k_1, d be integers such that $k_2 \leq k_1 \leq n$ and $2 \leq d \leq d(C_2)$. We define $\mathcal{F}_{k_1} := \{C_1 \subseteq \mathbb{F}_q^n \mid C_2 \subseteq C_1, \dim(C_1) = k_1\}$ and $\mathcal{F}_{k_1,<d} := \{C_1 \in \mathcal{F}_{k_1} \mid d(C_1) < d\}$. Then

$$|\mathcal{F}_{k_1,<d}|/|\mathcal{F}_{k_1}| \leq \frac{q^{k_1} - q^{k_2}}{(q-1)(q^n - q^{k_2})}(\boldsymbol{b}(d-1) - 1).$$

In particular, the number of codes $C \subseteq \mathbb{F}_q^n$ of dimension k such that $d(C) < d$, where $0 \leq k \leq n$ and $d \geq 2$, is at most

$$\frac{q^k - 1}{(q^n - 1)(q - 1)} \binom{n}{k}_q (\boldsymbol{b}(d-1) - 1).$$

Linear codes play an important role in the construction of CSS codes, which are the main topic of this thesis. For that we first need to understand quantum errors and quantum error-correcting codes. This is the goal of the next chapter.

Chapter 2

Quantum codes

In this chapter we introduce the notation for describing quantum information and we define quantum codes. It is important to understand quantum codes before we move on to CSS codes, which are a family of quantum codes.

The content of this chapter is organized in four sections, which mostly follow the information in *Quantum error-correcting codes and their geometries* by Ball, Centelles and Huber [1]. In Section 2.1, qubits and their notation are explained. In Section 2.2, we define the generalization of qubits to non-binary fields, which are called qudits. A presentation of qudits can also be found in *Qudits and high-dimensional quantum computing* by Wang [17]. Section 2.3 is devoted to the different types of quantum errors, this is explained according to *Quantum computation and quantum information* by Nielsen and Chuang [9, Chapter 2]. Finally, Section 2.4 provides the definition of quantum codes and their properties. These are described according to *Self-dual codes and invariant theory* by Nebe, Rains and Sloane [8, Chapter 11].

2.1 Qubits

In this section we introduce qubits according to the theory in *Quantum error-correcting codes and their geometries* by Ball, Centelles and Huber [1, Chapter 1].

A *qubit* is a two-state quantum mechanical system. It can describe quantum mechanical phenomena such as the spin of an electron or light polarization. In reality, a continuum of different photon polarizations or spin-directions is possible, however for both of these phenomena we can only observe two outcomes. Namely, electron spin can be measured as "spin up" and "spin down" and light polarization as vertical or horizontal. Qubits can mathematically represent the outcomes of this phenomena.

Single qubits are represented by the so-called "*ket*" notation $|0\rangle, |1\rangle$. These symbols actually represent the standard basis vectors in $\mathbb{C}^2$ or in $\mathbb{F}_2^2$, namely

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad \text{and} \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{2.1}$$

Any quantum state can be represented as the linear combination over $\mathbb{C}$ of the two kets of the standard basis

$$|\psi\rangle = \psi_0 |0\rangle + \psi_1 |1\rangle, \tag{2.2}$$

where $\psi_0, \psi_1 \in \mathbb{C}$ are such that $|\psi_0|^2 + |\psi_1|^2 = 1$. Recall that the absolute value of a complex number is defined as $|\psi_i| = \psi_i \bar{\psi_i}$ where $\bar{\psi_i}$ is the *complex conjugate* of ψ_i. The values $|\psi_0|^2$ and $|\psi_1|^2$ are the *probabilities* with which we can measure outcomes $|0\rangle$ and $|1\rangle$, respectively. For instance, if we consider the state

$$|\psi\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle,$$

then outcomes $|0\rangle$ and $|1\rangle$ can be measured with equal probability $\frac{1}{2}$.

Associated to the ket notation we have the "*bra*" notation, denoted $\langle\cdot|$. The bra $\langle\alpha|$ is a row vector whose entries are the complex conjugates of the entries in $|\alpha\rangle$. For instance, we have

$$\langle 0| = \begin{bmatrix}1 & 0\end{bmatrix}, \qquad \text{and} \qquad \langle 1| = \begin{bmatrix}0 & 1\end{bmatrix}. \tag{2.3}$$

By putting the bra and the ket together, we obtain the so-called "*braket*" $\langle\cdot|\cdot\rangle$. This notation is equivalent to the scalar product of two vectors. For instance, we have

$$\langle 0|1\rangle = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}0 & 1\end{bmatrix}^T = 0.$$

The space where qubits live is called a *quantum Hilbert space.* To define this space in more detail, we first have to define what a Hilbert space is. This is done in the following definition.

Definition 2.1.1. A *Hilbert space* is a complex inner product space (i.e. a vector space provided with an inner product), that is also a complete metric space.

The vector space spanned by $|0\rangle$ and $|1\rangle$ such as in Equation 2.2 together with the braket $\langle .|.\rangle$ is a Hilbert space. This Hilbert space is denoted by H_2.

Just as bits form bit-strings, quantum information is stored in strings of qubits. This can be done by tensoring single qubits together, i.e. by doing the *cross product* of multiple qubits

$$|b_0\, b_1 \dots b_{n-1}\rangle := |b_0\rangle \otimes |b_1\rangle \otimes \cdots \otimes |b_{n-1}\rangle,$$

where $b_i \in \{0,1\}$. For example the 2-qubit vector $|10\rangle$ represents the following vector in $\mathbb{C}^4$ or in $\mathbb{F}_2^4$

$$\begin{aligned} |10\rangle &= |1\rangle \otimes |0\rangle \\ &= \begin{bmatrix}0 & 1\end{bmatrix}^T \otimes \begin{bmatrix}1 & 0\end{bmatrix}^T \\ &= \begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}^T, \end{aligned}$$

and the ket $|101\rangle$ consisting of 3 qubits represents the following vector in $\mathbb{C}^8$ or in $\mathbb{F}_2^8$

$$\begin{aligned} |101\rangle &= |1\rangle \otimes |0\rangle \otimes |1\rangle \\ &= \begin{bmatrix}0 & 0 & 1 & 0\end{bmatrix}^T \otimes \begin{bmatrix}0 & 1\end{bmatrix}^T \\ &= \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\end{bmatrix}^T. \end{aligned}$$

Note that an n-qubit vector represents a vector of length 2^n, which makes it an elements of $\mathbb{C}^{2^n}$ or $\mathbb{F}_2^{2^n}$. Similarly as before, any n-quantum state $|\psi\rangle$ can be represented as the linear combination of the 2^n kets $|x_0\rangle, \dots, |x_{2^n-1}\rangle$ over $\mathbb{C}$

$$|\psi\rangle = \psi_0 |x_0\rangle + \cdots + \psi_{2^n-1} |x_{2^n-1}\rangle, \tag{2.4}$$

where x_i is the binary representation of $i \in \{0,\dots,2^n-1\}$ and where $\psi_i \in \mathbb{C}$ are such that $\sum_{i=0}^{n-1} |\psi_i|^2 = 1$. Moreover, for two vectors $\alpha = [\alpha_0 \dots \alpha_{n-1}]$ and $\beta = [\beta_0 \dots \beta_{n-1}]$ the braket is computed as follows

$$\langle\alpha|\beta\rangle = \bar{\alpha}_0\beta_0 + \bar{\alpha}_1\beta_1 + \cdots + \bar{\alpha}_{n-1}\beta_{n-1}.$$

Let us denote the *standard basis vectors* of $\mathbb{C}^n$ (or $\mathbb{F}_2^n$) by $e_0, e_1, \dots, e_{n-1}$, where e_i denotes the vector of length n that has a 1 on the i-th position and 0 elsewhere. Namely, if $n = 2$ then we get

$$e_0 = \begin{bmatrix}1 & 0\end{bmatrix}^T, \qquad e_1 = \begin{bmatrix}0 & 1\end{bmatrix}^T.$$

Note that for a binary vector $x \in \mathbb{F}_2^n$, we have $|x\rangle = e_i \in \mathbb{F}_2^{2^n}$ where x is the binary representation of the integer $i \in \{0,\dots,2^n-1\}$.

Now we can finally define a quantum Hilbert space.

Definition 2.1.2. If we denote the binary representation of the integer i by x_i, then the *quantum Hilbert space* H_2^n is defined by basis vectors $|x_i\rangle$ for $i \in \{0,1,\dots,2^n-1\}$. This space is naturally a tensor product of n Hilbert spaces H_2.

Example 2.1.3. Let us take $n = 2$. Then the basis elements for the quantum Hilbert space H_2^2 are the vectors $|00\rangle, |01\rangle, |10\rangle, |11\rangle$, where we have:

$$\begin{aligned}|00\rangle &= [1000]^T,\\ |01\rangle &= [0100]^T,\\ |10\rangle &= [0010]^T,\\ |11\rangle &= [0001]^T.\end{aligned}$$

In the next section, we will generalize qubits and the quantum Hilbert space over non-binary fields.

2.2 Qudits

This section follows the theory in *Quantum error-correcting codes and their geometries* by Ball, Centelles and Huber [1, Chapter 5] and *Qudits and high-dimensional quantum computing* by Wang [17].

Quantum information can also be stored in so-called *qudits*. These are quantum mechanical systems that can attain more than two states, compared to qubits. Qudits provide an alternative to qubit-based computers. Advantages of using this technology include the reduction of the complexity of the circuits and the simplification of the experimental setup [17]. This opens up different ways for designing new quantum computers.

Qudits have a slightly different mathematical notation compared to qubits. Instead of considering vectors over the binary field $\mathbb{F}_2$, we take them over any field $\mathbb{F}_q$ with $q = p^r$ for a prime number p and some positive integer r. In this case, if we write $\mathbb{F}_q = \{b_0, b_1, \dots, b_{q-1}\}$, we have

$$|b_0\rangle = \begin{bmatrix}1\\0\\ \vdots\\0\\0\end{bmatrix}, \qquad |b_1\rangle = \begin{bmatrix}0\\1\\0\\ \vdots\\0\end{bmatrix}, \qquad \dots \qquad |b_{q-1}\rangle = \begin{bmatrix}0\\0\\ \vdots\\0\\1\end{bmatrix}.$$

So $|b_0\rangle, |b_1\rangle, \dots |b_{q-1}\rangle$ represent the standard basis vectors e_i in $\mathbb{F}_q^n$ or $\mathbb{C}^n$. The quantum states can be represented as

$$|\psi\rangle = \sum_{i=0}^{q-1} \psi_i \,|\, b_i \,\rangle, \tag{2.5}$$

where $\psi_i \in \mathbb{C}$ with $\sum_{i=0}^{q-1} |\psi_i|^2 = 1$. Similarly as for $\mathbb{F}_2$, the qudits $|b_i\rangle$ span a Hilbert space according to Equation 2.5 with braket being the inner product. This Hilbert space is denoted by H_q.

Quantum information is again stored in strings of the form

$$|b_{i_0}\, b_{i_1} \dots b_{i_{n-1}}\rangle := |b_{i_0}\rangle \otimes |b_{i_1}\rangle \otimes \dots \otimes |b_{i_{n-1}}\rangle.$$

Moreover, the bra and the braket notation remain the same as discussed in Section 2.1.

Remark 2.2.1. For $a, b \in \mathbb{F}_q^n$ we have that $|a\rangle = |b\rangle$ if and only if $a = b$ since $|a\rangle$ and $|b\rangle$ are standard basis vectors and hence are orthogonal.

We also generalize the quantum Hilbert space, which is the space of qudits. This is done in the definition below.

Definition 2.2.2. The *quantum Hilbert space* H_q^n is defined by basis vectors $|x_i\rangle$ for $i \in \{0, 1, \dots, 2^q - 1\}$. This space is naturally a tensor product of n Hilbert spaces H_q.

With the notion of qubits and qudits, we can now discuss quantum errors.

2.3 Quantum errors

In this section we discuss the different types of errors that can occur during quantum computations. We first introduce them for quantum codes over $\mathbb{F}_2$ and afterwards we expand this over $\mathbb{F}_q$. We follow the theory from *Quantum computation and quantum information* by Nielsen and Chuang [9, Chapter 2] and *Quantum error-correcting codes and their geometries* by Ball, Centelles and Huber [1, Chapter 5].

Quantum errors can be represented by matrices that are so-called *unitary*. Therefore we first introduce a couple of definitions from *Self-dual codes and invariant theory* by Nebe, Rains and Sloane [8, Chapter 11] before discussing the types of quantum errors.

Let us consider a quantum Hilbert space H_q^n. Let M be a linear operator on H_q^n. We define $M^\dagger$ to be the *Hermitian conjugate* of M if $\langle Mu|v\rangle = \langle u|M^\dagger v\rangle$ for any vectors $u, v \in H_q^n$. We say that M is *Hermitian* if $M^\dagger = M$. Note that for vectors and matrices whose elements are real numbers, the Hermitian conjugate $\dagger$ is the same as the transpose T. Therefore we will from now on only use the notation $\dagger$ in also for the transpose.

Definition 2.3.1. A linear operator M on a Hilbert space H_q^n is *unitary* if $MM^\dagger = M^\dagger M = I$.

Definition 2.3.2. A linear operator M is *locally unitary* if it can be written as a tensor product of unitary matrices acting independently, namely if $M = M_1 \otimes \cdots \otimes M_r$ where U_i are unitary matrices.

If we consider binary linear codes, flipping a bit from 0 to 1 (or the opposite) is the only type of error that can occur. In binary quantum codes however, we recognize two types of errors: the *bit-flip* error X and the *phase-flip* error Z.

The quantum bit-flip is an error type that is equivalent to the bit-flip error in linear codes. In binary quantum codes this corresponds to flipping between the kets $|0\rangle$ and $|1\rangle$. Consequently, we define the action X as

$$\begin{aligned} X : \mathbb{C}^2 &\longrightarrow \mathbb{C}^2. \\ |0\rangle &\longmapsto |1\rangle \\ |1\rangle &\longmapsto |0\rangle \end{aligned}$$

Let $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ be a qubit state with $\alpha_0, \alpha_1 \in \mathbb{C}$ such that $|\alpha_0|^2 + |\alpha_0|^2 = 1$. Applying the bit flip to the qubit state yields

$$X|\psi\rangle = \alpha_0 X|0\rangle + \alpha_1 X|1\rangle = \alpha_0|1\rangle + \alpha_1|0\rangle .$$

Note that the bit-flip action X can be represented by the following unitary matrix

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

which thus describes the action on all vectors in $\mathbb{C}^2$ or $\mathbb{F}_2^2$.

The phase-flip error has no classical analogy. The action Z in binary quantum codes acts only on the sign of the ket $|1\rangle$, so it can be defined as

$$\begin{aligned} Z : \mathbb{C}^2 &\longrightarrow \mathbb{C}^2. \\ |0\rangle &\longmapsto |0\rangle \\ |1\rangle &\longmapsto -|1\rangle \end{aligned}$$

Let us again take the qubit state $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$, then the phase-flip Z acts on the qubit state as follows

$$Z|\psi\rangle = \alpha_0 Z|0\rangle + \alpha_1 Z|1\rangle = \alpha_0|1\rangle - \alpha_1|0\rangle .$$

Similarly as before, note that this action can also be represented by the unitary matrix

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

which acts on vectors in $\mathbb{C}^2$ or $\mathbb{F}_2^2$.

Matrices X and Z are two out of the three so-called *Pauli matrices* X, Y and Z. These matrices are very useful in the study of quantum errors, since every quantum error (also non-unitary) can be written as a linear combination of these matrices. The Pauli matrices can be seen bellow in Equation 2.6.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{2.6}$$

The actions X and Z can also be extended over $\mathbb{F}_q$ with $q = p^r > 2$. The equivalent to the bit-flip error is the *dit-flip*. This can be represented by a unitary matrix that acts on a codeword $|x\rangle$ as a permutation of its entries. Namely, if we label the elements of $\mathbb{F}_q$ by $\{b_0, b_1, \dots, b_{q-1}\}$, then for each $b_i \in \mathbb{F}_q$ we can define a $q \times q$ matrix $X(b_i)$ that is obtained by permuting the columns of the $q \times q$ identity matrix by i positions to the right.

Example 2.3.3. If $q = 3$, then $b_0 = 0$, $b_1 = 1$, $b_2 = 2$ are the elements of $\mathbb{F}_3$. Hence we have

$$X(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad X(1) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \qquad X(2) = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

The equivalent to the phase-flip error over $\mathbb{F}_q$ is also called the *phase-flip* and it can be represented by a matrix $Z(b)$. We define $Z(b)$ to be a $q \times q$ diagonal matrix such that its i-th entry on the diagonal is $\omega^{\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}(ib)}$ for each $b \in \mathbb{F}_q$. Here $\omega = e^{2\pi i/p}$ is a primitive p-th root of unity, and the *trace* $\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}$ is defined as follows

$$\mathrm{Tr}_{\mathbb{F}_q/\mathbb{F}_p}(\alpha) := \sum_{i=0}^{n-1} \alpha^{p^i}.$$

Example 2.3.4. If we again consider $q = 3$, then we find

$$Z(0) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad Z(1) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \omega & 0 \\ 0 & 0 & \omega^2 \end{bmatrix} \qquad Z(2) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \omega^2 & 0 \\ 0 & 0 & \omega \end{bmatrix}$$

Note that over $\mathbb{F}_2$ the matrices $X(a)$ and $Z(b)$ give the same matrices as the matrices X and Z. It is very important to distinguish these two types of errors and know how they work, in order to be able to correct them. In Chapter 2.4 we will define quantum error correcting codes and discuss some of their properties.

2.4 Quantum error-correcting codes

With the use of definitions in Section 2.1 and Section 2.2 we can finally define quantum codes in this section. Then, we introduce some properties of quantum codes. In this section we follow the definition in *Self-dual codes and invariant theory* by Nebe, Rains and Sloane [8, Chapter 11]. Quantum codes are also defined in other literature, for instance in the original paper by Calderbank and Shor about CSS codes [4] or in *Quantum error-correcting codes and their geometries* by Ball, Centelles and Huber [1].

The definitions that we provide in Definition 2.4.1 and Definition 2.4.2 are standard definitions for a quantum code and error-correcting quantum code.

Definition 2.4.1. Let H_q^n be a quantum Hilbert space. A *quantum code Q* is any subspace of H_q^n.

Definition 2.4.2. Let H_q^n be a quantum Hilbert space. A *quantum t-error-correcting code Q* is a linear subspace of H_q^n of which t errors on qubits can be corrected. The *dimension* of Q is the dimension of the subspace.

We will now discuss some mathematical aspects of quantum error-correcting codes. The theory that follows is from Nebe, Rains and Sloane [8, Chapter 11] is slightly different than in other literature about quantum error-correcting codes, such as the theory in [1, 4, 13].

Let us consider a quantum Hilbert space $H_q^n = H_{q,1} \otimes \cdots \otimes H_{q,n}$. Let M be a linear operator on H_q^n. If B is the orthonormal basis of H_q^n, then the *trace* of M is defined to be

$$\mathrm{Tr}(M) := \sum_{|u\rangle \in B} \langle u| M |u\rangle .$$

Now let us write $H_q^n = V_1 \otimes V_2$ as a tensor product of two quantum Hilbert spaces. The *partial trace* Tr_{V_2} is the unique Hermitian operator on V_1 such that

$$\mathrm{Tr}(M(N \otimes I)) = \mathrm{Tr}(\mathrm{Tr}_{V_2}(M)N)$$

for all Hermitian operators N on V_1, M on H_q^n and I the identity matrix. We define similarly $\mathrm{Tr}_{V_1}(M)$ on V_2. Notice that $\mathrm{Tr}_{\emptyset} = I$ and $\mathrm{Tr}_{\{1,\dots,n\}} = \mathrm{Tr}$.

Definition 2.4.3. Let $\alpha \subseteq \{1,\dots,n\}$. Then α^C denotes the complement of α in $\{1,\dots,n\}$. We say that Q *can correct the erasure of* α if the partial trace $\mathrm{Tr}_{\alpha^C}(uu^\dagger)$ is independent of u for all unit vectors $u \in Q$. Here $\mathrm{Tr}_{\alpha^C}(uu^\dagger)$ is shorted for $\mathrm{Tr}_{H_{q,j_1} \otimes H_{q,j_2} \otimes \dots}$ with $\alpha^C = \{j_1, j_2, \dots\}$ and $H_q^n = H_{q,1} \otimes \cdots \otimes H_{q,n}$.

Definition 2.4.4. We say Q is *pure with respect to* α if for all unit vectors $u \in Q$, we have $\mathrm{Tr}_{\alpha^C}(uu^\dagger) = \lambda I$ for some scalar λ and the identity matrix I. We say Q *has minimal distance* $\geq d$ if for all α such that $|\alpha| < d$ the code Q can correct the erasure of α. We say Q *has minimal distance equal to* d if it has minimal distance $\geq d$ but not $\geq d+1$.

Note that Hermitian operator M can be uniquely written in the following way

$$M = \sum_{i,j,k,l} a_{ij,kl}(e_i \otimes f_j)(e_k \otimes f_l)^\dagger, \tag{2.7}$$

where $\{e_i\}_{i \in I}$ and $\{f_j\}_{j \in J}$ are the orthogonal bases for V_1 and V_2 respectively and $a_{ij,kl}$ are unique complex coefficients. Then the partial traces of M are of the following forms:

$$\mathrm{Tr}_{V_1}(M) = \sum_{i,j,l} a_{ij,il} f_j f_l^\dagger, \qquad \mathrm{Tr}_{V_2}(M) = \sum_{i,j,k} a_{ij,kj} e_i e_k^\dagger. \tag{2.8}$$

Example 2.4.5. (Example 13.1.4 from [8]) Take $H = H_1 \otimes H_2$ such that both $H_i = \mathbb{C}^2$. Denote their orthonormal bases by $\{e_i\}_{i=0}^1$ and $\{f_i\}_{i=0}^1$, respectively. Then we know that

$$\begin{aligned} e_0 = f_0 = |0\rangle = \begin{bmatrix} 1 & 0 \end{bmatrix}^\dagger, \\ e_1 = f_1 = |1\rangle = \begin{bmatrix} 0 & 1 \end{bmatrix}^\dagger. \end{aligned}$$

The basis elements for H are denoted by e_{ij} as follows

$$\begin{aligned} e_{00} = e_0 \otimes f_0 = \begin{bmatrix} 1 & 0 \end{bmatrix}^\dagger \otimes \begin{bmatrix} 1 & 0 \end{bmatrix}^\dagger = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^\dagger \\ e_{01} = e_0 \otimes f_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}^\dagger \otimes \begin{bmatrix} 0 & 1 \end{bmatrix}^\dagger = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix}^\dagger \\ e_{10} = e_1 \otimes f_0 = \begin{bmatrix} 0 & 1 \end{bmatrix}^\dagger \otimes \begin{bmatrix} 1 & 0 \end{bmatrix}^\dagger = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}^\dagger \\ e_{11} = e_1 \otimes f_1 = \begin{bmatrix} 0 & 1 \end{bmatrix}^\dagger \otimes \begin{bmatrix} 0 & 1 \end{bmatrix}^\dagger = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix}^\dagger \end{aligned}$$

Let the quantum code Q be $Q = \langle e_{00} + e_{11} \rangle$. We claim that the minimal distance of Q is ≥ 2. For that we need to check whether Q corrects erasure of all α such that $|\alpha| < 2$. That means we only need to check it for $\alpha = 1$, those are the subsets $\{1\}$ and $\{2\}$. Hence we need to compute $\mathrm{Tr}_{\{1\}^C}(uu^\dagger) = \mathrm{Tr}_{H_2}(uu^\dagger)$ and $\mathrm{Tr}_{\{2\}^C}(uu^\dagger) = \mathrm{Tr}_{H_1}(uu^\dagger)$ for all unit vectors $u \in Q$ and see if they both are proportional to identity.

First of all, note that Q is generated by one vector, so there is only one unit vector namely

$$u = \frac{1}{||e_{00} + e_{11}||}(e_{00} + e_{11}) = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix}^\dagger$$

for which we need to check the property above. Then we have

$$uu^\dagger = \frac{1}{2}\begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 1 & 0 & 0 & 1\end{bmatrix}.$$

Now we find the constants $a_{ij,kl}$ from Equation (2.7)

$$\begin{aligned}
uu^\dagger &= \sum_{i,j,k,l} a_{ij,kl}(e_{ij})(e_{kl})^\dagger \\
&= a_{00,00}\begin{bmatrix}1000\end{bmatrix}^\dagger\begin{bmatrix}1000\end{bmatrix} + a_{00,01}\begin{bmatrix}1000\end{bmatrix}^\dagger\begin{bmatrix}0100\end{bmatrix} + \cdots + a_{11,11}\begin{bmatrix}0001\end{bmatrix}^\dagger\begin{bmatrix}0001\end{bmatrix} \\
&= \frac{1}{2}\left(\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & 1\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 1 & 0 & 0 & 0\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\right)
\end{aligned}$$

Note that only the coefficients $a_{00,00} = a_{00,11} = a_{11,00} = a_{11,11} = \frac{1}{2}$, otherwise all other coefficients $a_{ij,kl} = 0$. We proceed with filling in $a_{ij,kl}$ into Equation (2.8),

$$\begin{aligned}
\mathrm{Tr}_{H_1}(uu^\dagger) &= \sum_{i,j,l} a_{ij,il} f_j f_l^\dagger \\
&= a_{00,00}\begin{bmatrix}1 & 0\end{bmatrix}^\dagger\begin{bmatrix}1 & 0\end{bmatrix} + a_{00,01}\begin{bmatrix}1 & 0\end{bmatrix}^\dagger\begin{bmatrix}0 & 1\end{bmatrix} + a_{01,00}\begin{bmatrix}0 & 1\end{bmatrix}^\dagger\begin{bmatrix}1 & 0\end{bmatrix} + \\
&+ a_{01,01}\begin{bmatrix}0 & 1\end{bmatrix}^\dagger\begin{bmatrix}0 & 1\end{bmatrix} + a_{10,10}\begin{bmatrix}1 & 0\end{bmatrix}^\dagger\begin{bmatrix}1 & 0\end{bmatrix} + a_{10,11}\begin{bmatrix}1 & 0\end{bmatrix}^\dagger\begin{bmatrix}0 & 1\end{bmatrix} + \\
&+ a_{11,10}\begin{bmatrix}0 & 1\end{bmatrix}^\dagger\begin{bmatrix}1 & 0\end{bmatrix} + a_{11,11}\begin{bmatrix}0 & 1\end{bmatrix}^\dagger\begin{bmatrix}0 & 1\end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix} + 0 + \cdots + 0 + \frac{1}{2}\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}.
\end{aligned}$$

Similarly,

$$\begin{aligned}
\mathrm{Tr}_{H_2}(uu^\dagger) &= \sum_{i,j,k} a_{ij,kj} e_i e_k^\dagger \\
&= \frac{1}{2}\begin{bmatrix}1 & 0\\ 0 & 0\end{bmatrix} + 0 + \cdots + 0 + \frac{1}{2}\begin{bmatrix}0 & 0\\ 0 & 1\end{bmatrix} \\
&= \frac{1}{2}\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}.
\end{aligned}$$

Hence both $\mathrm{Tr}_{H_1}(uu^\dagger)$ and $\mathrm{Tr}_{H_2}(uu^\dagger)$ are proportional to identity, so Q has minimal distance ≥ 2.

We can also show that Q has minimal distance not ≥ 3, which was not shown in the original example from [8]. Assume that $|\alpha| = 2$, hence $\mathrm{Tr}_{\{1,2\}^C}(uu^\dagger) = \mathrm{Tr}_\emptyset(uu^\dagger) = \lambda I$ for the unit vector u as above. However, we know that $\mathrm{Tr}_\emptyset = \mathrm{Id}$, which leads to a contradiction since $uu^\dagger$ cannot be written as λI. This shows that the minimal distance of Q is $d = 2$.

Chapter 3

Correcting quantum errors with classical codes: the CSS construction

The *CSS* codes, named after Calderbank, Shor and Steane, are a class of quantum error-correcting codes derived from classical linear codes. Their construction was first introduced by Calderbank and Shor [4] and independently by Steane [15]. The two constructions are different but turn out to be equivalent after a base change. In this chapter we introduce both definitions in Section 3.1 and Section 3.2, following the original papers [4, 15]. The original papers only use classical linear codes over the binary field $\mathbb{F}_2$ which gives CSS codes with qubits. In Section 3.3 we generalize this construction using codes defined over $\mathbb{F}_q$. Therefore the CSS codes will use qudits instead of qubits. Finally, in Section 3.4 we introduce the Theorem of Calderbank and Shor about error-correcting CSS codes and we generalize it to $\mathbb{F}_q$.

3.1 Definition by Calderbank and Shor

In this section we introduce the original construction of a CSS code by Calderbank and Shor [4]. This construction is performed using linear codes defined over the binary field $\mathbb{F}_2$. In this way, we obtain quantum codes using qubits, which were introduced in Section 2.1.

Definition 3.1.1. Let us take a quantum Hilbert space $H_2^n = H_2 \otimes \cdots \otimes H_2$ with basis vectors $|x_i\rangle$, where for any integer $i \in \{0, 1, \dots 2^n - 1\}$ we have that x_i is the binary representation of i. For a linear code $C \subseteq \mathbb{F}_2^n$ we define the *quantum Hilbert subspace* $H_C \subseteq H_2^n$ in the following way: For every codeword $c \in C$ we let $|c\rangle$ be a basis vector for H_C.

Let us take a quantum Hilbert subspace $H_{C_1} \subseteq H_2^n$. For every vector $w \in \mathbb{F}_2^n$ we define a quantum state as follows

$$|c_w\rangle = 2^{-\dim(C_1)/2} \sum_{c \in C_1} (-1)^{\langle c, w\rangle} |c\rangle .$$

where $\langle c, w\rangle$ denotes the inner product of vectors c and w. This construction has two important properties which will be shown in the following lemmas. The properties are stated in the paper by Calderbank and Shor [4]. In this section we will develop all the details of the proofs of these properties, which are not given in the paper.

Lemma 3.1.2. (Property 1) For vectors $w_1, w_2 \in \mathbb{F}_2^n$ and a codeword $c \in C \subseteq \mathbb{F}_2^n$, we have that

$$|c_{w_1}\rangle = |c_{w_2}\rangle \iff w_1 + w_2 \in C^\perp .$$

Proof. First, notice that by the definition of the dual code of C we know that $w_1 + w_2 \in C^\perp$ if and only if $\langle c, (w_1 + w_2)\rangle = 0$ for any $c \in C$. This is equivalent to $\langle c, w_1\rangle = \langle c, w_2\rangle$ for all $c \in C$. Finally, by the definition of $|c_w\rangle$, we have $\langle c, w_1\rangle = \langle c, w_2\rangle$ for all $c \in C$ if and only if $|c_{w_1}\rangle = |c_{w_2}\rangle$. □

In order to prove the second property, we need the following lemma.

Lemma 3.1.3. For a code $C \subseteq \mathbb{F}_2^n$ with a generator matrix G of size $k \times n$, and a vector $w \in \mathbb{F}_2^n$, we have

$$\sum_{c \in C} (-1)^{\langle c, w \rangle} = \sum_{v \in \mathbb{F}_2^k} (-1)^{vGw} = 0,$$

unless for all $c \in C$ we have $\langle c, w \rangle = 0$.

Proof. The first equality follows from the fact that every codeword $c \in C$ can be written as vG for some $v \in \mathbb{F}_2^k$. For proving the second equality, assume first that for all $v \in \mathbb{F}_2^k$ we have $vGw = 0$. Then

$$\sum_{v \in \mathbb{F}_2^k} (-1)^{vGw} = \sum_{v \in \mathbb{F}_2^k} 1 = 2^k \neq 0.$$

Now assume that there exists a vector v_1 such that $v_1 Gw \neq 0$, then in particular we must have $Gw \neq \mathbf{0}$. Let Gw have a 1 in the j-th position. Then if v_1 has a 1 in j-th position, i.e. $(v_1)_j = 1$, we get

$$v_1 Gw = \sum_{i=1}^{k} (v_1)_i \cdot (Gw)_i = \sum_{\substack{i=1 \\ i \neq j}}^{k} (v_1)_i (Gw)_i + (v_1)_j (Gw)_j = \sum_{\substack{i=1 \\ i \neq j}}^{k} (v_1)_i (Gw)_i + 1.$$

However, if we take a vector v_2 such that it has the exact same entries as v_1 apart of the j-th coordinate, which is $(v_2)_j = 0$, then we have

$$\begin{aligned}
v_2 Gw &= \sum_{i=1}^{k} (v_2)_i \cdot (Gw)_i \\
&= \sum_{\substack{i=1 \\ i \neq j}}^{k} (v_2)_i (Gw)_i + (v_2)_j (Gw)_j \\
&= \sum_{\substack{i=1 \\ i \neq j}}^{k} (v_2)_i (Gw)_i + 0 \\
&= \sum_{\substack{i=1 \\ i \neq j}}^{k} (v_1)_i (Gw)_i \\
&= v_1 Gw - 1.
\end{aligned}$$

Finally, the following chain of equalities concludes the proof:

$$\begin{aligned}
\sum_{v \in \mathbb{F}_2^k} (-1)^{vGw} &= \sum_{\substack{v \in \mathbb{F}_2^k \\ (v)_j = 1}} (-1)^{vGw} + \sum_{\substack{v \in \mathbb{F}_2^k \\ (v)_j = 0}} (-1)^{vGw} \\
&= \sum_{\substack{v \in \mathbb{F}_2^k \\ (v)_j = 1}} (-1)^{vGw} + \sum_{\substack{v \in \mathbb{F}_2^k \\ (v)_j = 1}} (-1)^{vGw-1} \\
&= \sum_{\substack{v \in \mathbb{F}_2^k \\ (v)_j = 1}} (-1)^{vGw} \left(1 + (-1)^{-1}\right) \\
&= \sum_{\substack{v \in \mathbb{F}_2^k \\ (v)_j = 1}} (-1)^{vGw} (1 - 1) \\
&= 0.
\end{aligned}$$

□

We are now ready to prove the second property.

Lemma 3.1.4. (Property 2) For vectors $w_1, w_2 \in \mathbb{F}_2^n$ and a codeword $c \in C \subseteq \mathbb{F}_2^n$, we have that

$$w_1 + w_2 \notin C^\perp \implies \langle c_{w_1} | c_{w_2} \rangle = 0$$

Proof. Let $\dim(C) = k$, and take $w_1, w_2 \in \mathbb{F}_2^n$ such that $w_1 + w_2 \notin C^\perp$, then

$$\begin{aligned} |c_{w_1}\rangle &= 2^{-\frac{k}{2}} \sum_{c_i \in C} (-1)^{\langle c, w_1 \rangle} |c_i\rangle, \\ |c_{w_2}\rangle &= 2^{-\frac{k}{2}} \sum_{c_j \in C} (-1)^{\langle c, w_2 \rangle} |c_j\rangle. \end{aligned}$$

Considering the braket of the two quantum states gives

$$\begin{aligned} \langle c_{w_1} | c_{w_2} \rangle &= \left\langle 2^{-\frac{k}{2}} \sum_{c_i \in C} (-1)^{\langle c_i, w_1 \rangle} |c_i\rangle \middle| 2^{-\frac{k}{2}} \sum_{c_j \in C} (-1)^{\langle c_j, w_2 \rangle} |c_j\rangle \right\rangle \\ &= \frac{1}{2^k} \left\langle \sum_{c_i \in C} (-1)^{\langle c_i, w_1 \rangle} |c_i\rangle \middle| \sum_{c_j \in C} (-1)^{\langle c_j, w_2 \rangle} |c_j\rangle \right\rangle \\ &= \frac{1}{2^k} \left(\sum_{i=1}^{2^k} \sum_{j=1}^{2^k} (-1)^{\langle c_i, w_1 \rangle + \langle c_j, w_2 \rangle} \langle c_i | c_j \rangle \right) \\ &= \frac{1}{2^k} \sum_{i=1}^{2^k} \Big((-1)^{\langle c_i, w_1 + w_2 \rangle} \langle c_i | c_i \rangle \Big) + \\ &\quad + \frac{1}{2^k} \sum_{\substack{i,j \\ i \neq j}}^{2^k} \Big((-1)^{\langle c_i, w_1 \rangle + \langle c_j, w_2 \rangle} \langle c_i | c_j \rangle + (-1)^{\langle c_j, w_1 \rangle + \langle c_i, w_2 \rangle} \langle c_j | c_i \rangle \Big). \end{aligned}$$

First, note that $\langle c_i | c_i \rangle = ||c_i||^2 = 1$ since every ket has norm 1. Secondly, note that $\langle c_i | c_j \rangle$ with $i \neq j$ is the inner product of two standard basis vectors in $\mathbb{F}_2^{2^n}$ (see Section 2.1). Hence $\langle c_i | c_j \rangle = \langle e_{i'} | e_{j'} \rangle = 0$ where i', j' are different since $i \neq j$. Therefore the second sum is equal to 0. Thus we are left with proving that

$$\langle c_{w_1} | c_{w_2} \rangle = \frac{1}{2^k} \left(\sum_{i=1}^{2^k} (-1)^{\langle c_i, w_1 + w_2 \rangle} \right) \tag{3.1}$$

is equal to zero. Since we assume $w_1 + w_2 \notin C_1^\perp$, that means that there exists a $c \in C$ for which $\langle c, w_1 + w_2 \rangle \neq 0$ and hence by Lemma 3.1.3 it follows that the sum in the right side of Equation (3.1) is 0. This concudes the proof. □

The two properties from Lemma 3.1.2 and Lemma 3.1.4 are crucial for finding the basis for a quantum Hilbert subspace. In order to do that, we first have to define cosets. This is possible since every code is an abelian group.

Definition 3.1.5. Let $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$ be two linear codes. We define $C_1/C_2 := \{x + C_2 \,|\, x \in C_2\}$ and we say that the elements of C_1/C_2 are the *cosets* of C_1.

Definition 3.1.6. Let $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$ be two linear codes. We say that two codewords w, w' are *equivalent* in C_1/C_2 if $w - w' \in C_2$.

In the following lemma we find the basis for the quantum Hilbert space H_{C_1} that we have defined at the beginning of this section.

Lemma 3.1.7. For a code $C \subseteq \mathbb{F}_2^n$ and vectors $w \in \mathbb{F}_2^n/C^\perp$, the quantum states $|c_w\rangle$ form a basis for H_C.

Proof. Let us take $w, w' \in \mathbb{F}_2^n$. If $w + w' \in C^\perp$ then by Lemma 3.1.2 we have that the codewords $|c_w\rangle$ and $|c_{w'}\rangle$ are equal. If $w + w' \notin C^\perp$, then by Lemma 3.1.4 the two codewords $|c_w\rangle$ and $|c_{w'}\rangle$ are orthogonal. That shows that all $|c_w\rangle$ with $w \in \mathbb{F}_2^n/C^\perp$ are orthogonal. Moreover, note that by the definition their norm is equal to 1 and hence the $|c_w\rangle$'s are orthonormal vectors. Finally, the codewords $|c_w\rangle$ for all $w \in C^\perp$ span the whole space H_C by definition, therefore we conclude that the $|c_w\rangle$ for $w \in \mathbb{F}_2^n/C^\perp$ form a basis for H_C. $\square$

After we have found the basis of H_{C_1}, we can finally proceed with the CSS construction. Let C_2 be a linear code such that $C_2 \subseteq C_1 \subseteq \mathbb{F}_2^n$. Following the construction by Calderbank and Shor, we define the quantum code to be the following

$$Q^{cs}_{C_1,C_2} := \{|c_w\rangle \mid w \in {C_2}^\perp\} = \{|c_w\rangle \mid w \in C_2^\perp/C_1^\perp\},$$

where the second equality holds since from Lemma 3.1.2 we know that $|c_w\rangle = |c_{w'}\rangle$ if and only if the sum of w and w' is in $C^\perp$.

Remark 3.1.8. Note that by definition of $Q^{cs}_{C_1,C_2}$ we have

$$|Q^{cs}_{C_1,C_2}| = |C_2^\perp/C_1^\perp| = |C_2^\perp|/|C_1^\perp|,$$

and hence $\dim(Q^{cs}_{C_1,C_2}) = \dim(C_2^\perp) - \dim(C_1^\perp) = (n - \dim(C_2)) - (n - \dim(C_1)) = \dim(C_1) - \dim(C_2)$.

Example 3.1.9. Let $C_1 \subseteq \mathbb{F}_2^5$ be a binary linear code with the following generator matrix

$$G_1 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}.$$

The code C_1 has parameters $[n, k, d] = [5, 3, 2]$. It consists of the codewords

$$C_1 = \{[00000], [10010], [01001], [11011], [00111], [10101], [01110], [11100]\}.$$

Let us take C_2 with generator matrix

$$G_2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}.$$

So $C_2 = \{[00000], [10010], [01001], [11011]\}$. Note that $C_2 \subseteq C_1$. To find the quantum code $Q^{cs}_{C_1,C_2}$, we find find $C_2^\perp$ and $C_1^\perp$. We have that

$$C_2^\perp = \{[0,0,0,0,0], [1,0,0,1,0], [0,1,0,0,1], [1,1,0,1,1], [0,0,1,0,0], [1,0,1,1,0], [0,1,1,0,1], [1,1,1,1,1]\},$$
$$C_1^\perp = \{[0,0,0,0,0], [1,0,1,1,0], [0,1,1,0,1], [1,1,0,1,1]\}.$$

Therefore, the elements of the set $C_2^\perp/C_1^\perp$ are

$$C_2^\perp/C_1^\perp = \{[0,0,0,0,0], [1,1,1,1,1]\},$$

where we leave out the $+C_1^\perp$ in the notation of the elements. The reason why we only have these two elements in $C_2^\perp/C_1^\perp$ is that $[0,0,0,0,0]$ is also equivalent to $[1,0,1,1,0], [0,1,1,0,1], [1,1,0,1,1]$ and $[1,1,1,1,1]$ is also equivalent to $[1,0,0,1,0], [0,0,1,0,0], [0,1,0,0,1]$. Therefore, the quantum code $Q^{cs}_{C_1,C_2}$ has dimension 1 and consists of exactly $2^{\dim(C_1)-\dim(C_2)} = 2^{3-2} = 2$ codewords, namely

$$|c_0\rangle = |c_{[00000]}\rangle = \frac{1}{\sqrt{8}}\left(|00000\rangle + |10010\rangle + |01001\rangle + |11011\rangle + |00111\rangle + |10101\rangle + |01110\rangle + |11100\rangle\right),$$
$$|c_1\rangle = |c_{[11111]}\rangle = \frac{1}{\sqrt{8}}\left(|00000\rangle + |10010\rangle + |01001\rangle + |11011\rangle - |00111\rangle - |10101\rangle - |01110\rangle - |11100\rangle\right).$$

3.2 Definition by Steane

A different definition of CSS codes is the construction proposed by Steane [15]. In this section we present the definition as formalised in [9]. Then we show the equivalence within Steane's construction and the one from Calderbank and Shor presented in Section 3.1.

Take linear codes $C_2 \subseteq C_1 \subseteq \mathbb{F}_2^n$ and let $w \in C_1$. We define the quantum state

$$|w + C_2\rangle := \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w + c\rangle \, ,$$

where the + in the ket notation denotes digit-wise addition modulo 2. For quantum codewords of this form a useful property holds, which is crucial in defining the CSS quantum code. In the proof of Lemma 3.2.1 we give the details on this property which is only stated and very briefly explained in the book [9].

Lemma 3.2.1. (Property) For linear codes $C_2 \subseteq C_1 \subseteq \mathbb{F}_2^n$ and vectors $w, w' \in C_1$, we have that

$$w + w' \in C_2 \iff |w + C_2\rangle = |w' + C_2\rangle \, .$$

Proof. Suppose $\bar{c} = w + w' \in C_2$. Then for all $c \in C_2$ we have that $w + c = w' + (\bar{c} + c)$ which implies that $|w + c\rangle = |w' + (\bar{c} + c)\rangle$. Therefore

$$\frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w + c\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w' + (\bar{c} + c)\rangle \, ,$$

which is by relabeling the same as

$$\frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w + c\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w' + c\rangle \, .$$

Hence, we find $|w + C_2\rangle = |w' + C_2\rangle$.

For the other direction, assume $w + w' \notin C_2$. Suppose for contradiction that there exist $c, c' \in C_2$ such that $w + c = w' + c'$. Then $w + w' = c + c' \in C_2$ which is a contradiction. Therefore there are no codewords $c, c' \in C_2$ such that $w + c = w' + c'$ and hence we must have that $|w + C_2\rangle \neq |w' + C_2\rangle$. □

By Lemma 3.2.1 it follows that the natural index set for $|w + C_2\rangle$ is the set of cosets $w \in C_1/C_2$ as defined in Definition 3.1.5. Naturally, the CSS code is then defined as follows

$$Q^s_{C_1,C_2} = \{|w + C_2\rangle \mid w \in C_1/C_2\}.$$

It follows that the number of quantum codewords in $Q^s_{C_1,C_2}$ is the same as $|C_1/C_2|$ and hence similarly as in Remark 3.1.8 we have that

$$\dim(Q^s_{C_1,C_2}) = \dim(C_1) - \dim(C_2).$$

Example 3.2.2. Let us take the same codes $C_2 \subseteq C_1 \subseteq \mathbb{F}_2^5$ as in Example 3.1.9. Then the codewords of $Q^s_{C_1,C_2}$ are the following:

$$|[00000] + C_2\rangle = \frac{1}{2}\left(|00000\rangle + |10010\rangle + |01001\rangle + |11011\rangle\right),$$
$$|[00111] + C_2\rangle = \frac{1}{2}\left(|00111\rangle + |10101\rangle + |01110\rangle + |11100\rangle\right).$$

The dimension of $Q^s_{C_1,C_2}$ is 1. Note that $Q^{cs}_{C_1,C_2}$ from Example 3.1.9 is not equal to the code $Q^s_{C_1,C_2}$ of this example.

As mentioned previously, Steane's definition of CSS code is equivalent to Canderbank and Shor's definition from Section 3.1. Before we prove this explicitly, we need to define what equivalence in the quantum world means. The quantum equivalence is defined in Definition 3.2.3, which is equivalent to the definition from to *Quantum computation* lecture notes by Preskill [11, Section 7.9.1].

Definition 3.2.3. Two quantum codes Q and Q' are equivalent if they differ by a locally unitary matrix and a permutation.

In the following proposition we will show that the Steane construction is equivalent to the Calderbank and Shor construction by showing that Q^{cs} and Q^{s} are equivalent.

Proposition 3.2.4. Let $C_2 \subseteq C_1 \subseteq \mathbb{F}_2^n$ be classical linear codes. Then $Q^{cs}_{C_1,C_2}$ is equivalent to $Q^{s}_{C_2^\perp,C_1^\perp}$.

Proof. Let us consider the quantum Hilbert subspace $H_{C_1} \subseteq H_2^n$. Let us take a CSS code $Q^{cs}_{C_1,C_2}$. We will apply the following change of the basis of H_{C_1}

$$\begin{aligned} |0\rangle &\longmapsto \frac{1}{\sqrt{2}}\left(|0\rangle + |1\rangle\right), \\ |1\rangle &\longmapsto \frac{1}{\sqrt{2}}\left(|0\rangle - |1\rangle\right) \end{aligned}$$

to the codewords of $Q^{cs}_{C_1,C_2}$. Recall that $Q^{cs}_{C_1,C_2} = \{|c_w\rangle \mid w \in C_2^\perp/C_1^\perp\}$. By [4, Section 3], applying the change of basis to the codewords $|c_w\rangle$ gives

$$|c_w\rangle = 2^{-\dim(C_1)/2}\sum_{c\in C_1}(-1)^{\langle c,w\rangle}|c\rangle \quad \longmapsto \quad |s_w\rangle = 2^{(\dim(C_1)-n)/2}\sum_{u\in C_1^\perp}|u+w\rangle .$$

Note that $\dim(C_1) - n = -\dim(C_1^\perp)$ and hence $2^{-\dim(C_1^\perp)/2} = \frac{1}{\sqrt{|C_1^\perp|}}$. Therefore we find that $|s_w\rangle$ is actually equal to $\left|w + C_1^\perp\right\rangle \in Q^{s}_{C_2^\perp,C_1^\perp}$. Moreover, note that the change of basis corresponds to the Hadamard matrix

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

which is unitary. Therefore, the linear operator that maps $|c_w\rangle$ to $|s_w\rangle$ composed of n of these matrices tensored, and hence the linear operator is locally unitary. This shows the equivalence between the two constructions. $\square$

Example 3.2.5. Take C_1, C_2 such as in Example 3.1.9. Then we have seen that $Q^{cs}_{C_1,C_2}$ contains two codewords, namely

$$\begin{aligned} |c_0\rangle &= \left|c_{[00000]}\right\rangle = \frac{1}{\sqrt{8}}\left(|00000\rangle + |10010\rangle + |01001\rangle + |11011\rangle + |00111\rangle + |10101\rangle + |01110\rangle + |11100\rangle\right), \\ |c_1\rangle &= \left|c_{[11111]}\right\rangle = \frac{1}{\sqrt{8}}\left(|00000\rangle + |10010\rangle + |01001\rangle + |11011\rangle - |00111\rangle - |10101\rangle - |01110\rangle - |11100\rangle\right). \end{aligned}$$

By applying the change of basis from Proposition 3.2.4, the codewords $|c_w\rangle \in Q^{cs}_{C_1,C_2}$ change to the following codewords

$$\begin{aligned} \left|s_{[00000]}\right\rangle &= \frac{1}{2}\left(|00000\rangle + |10010\rangle + |01001\rangle + |11011\rangle\right), \\ \left|s_{[11111]}\right\rangle &= \frac{1}{2}\left(|11111\rangle + |01101\rangle + |10110\rangle + |00100\rangle\right). \end{aligned}$$

Note that these codewords are equal to codewords in the Steane construction of CSS codes, namely

$$\begin{aligned} \left|s_{[00000]}\right\rangle &= \left|[00000] + C_1^\perp\right\rangle, \\ \left|s_{[11111]}\right\rangle &= \left|[11111] + C_1^\perp\right\rangle \end{aligned}$$

and that implies that under the change of basis from Proposition 3.2.4, the CSS code $Q^{cs}_{C_1,C_2}$ is the same code as $Q^{s}_{C_1',C_2'}$ with $C_1' = C_2^\perp$ and $C_2' = C_1^\perp$, which corresponds to the result in Proposition 3.2.4.

3.3 CSS codes over $\mathbb{F}_q$

In this section we generalize the CSS construction using classical linear codes defined over $\mathbb{F}_q$ for any $q = p^r$ with p a prime number and r a positive integer. Since Q^{cs} and Q^s are equivalent, we will denote both by Q and in what follows we will use the construction by Steane introduced in Section 3.2.

Definition 3.3.1. Take linear codes $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$ and let $w \in C_1$. Then we define the quantum state

$$|w + C_2\rangle \coloneqq \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w + c\rangle \,,$$

where the $+$ in the ket notation denotes digit-wise addition modulo p. Then the CSS code is defined as

$$Q_{C_1,C_2} \coloneqq \{|w + C_2\rangle \mid w \in C_1/C_2\}.$$

Similar results to the binary case hold when we move to $\mathbb{F}_q$. For instance, the reason for taking w from the cosets C_1/C_2 is the same as in the binary case, which was shown in Lemma 3.2.1. In the following lemma we generalize the property from Lemma 3.2.1 for linear codes over $\mathbb{F}_q$.

Lemma 3.3.2. (Property) Let $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$. For $w, w' \in C_1$ we have

$$w - w' \in C_2 \iff |c_w\rangle = |c_{w'}\rangle \,.$$

Proof. Assume $w - w' \in C_2$, so for some $\bar{c} \in C_2$ we have $\bar{c} = w - w'$. Then for all $c \in C_2$ we have that $w + c = w' + (\bar{c} + c)$ which immediately gives $|w + c\rangle = |w' + (\bar{c} + c)\rangle$. Therefore we find that

$$\frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w + c\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w' + (\bar{c} + c)\rangle \,,$$

which by relabeling is the same as

$$\frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w + c\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w' + c\rangle \,.$$

Hence, we find $|w + C_2\rangle = |w' + C_2\rangle$.

For the other direction, assume $w - w' \notin C_2$. Then suppose for contradiction that there are $c, c' \in C_2$ such that $w + c = w + c'$. Then $w - w' = c - c' \in C_2$ which is a contradiction. Therefore there are no codewords $c, c' \in C_2$ such that $w + c = w' + c'$ and hence we must have that $|c_w\rangle \neq |c_{w'}\rangle$. ☐

By the the previous lemma we know that counting quantum codewords $|c_w\rangle$ is the same as counting linear codewords $w \in C_1/C_2$. Hence the number of codewords in Q_{C_1,C_2} is $|Q_{C_1,C_2}| = |C_1|/|C_2| = q^{\dim(C_1)-\dim(C_2)}$. We conclude as in Remark 3.1.8 that the dimension of Q_{C_1,C_2} is $\dim(Q_{C_1,C_2}) = \dim(C_1) - \dim(C_2)$.

The definition of CSS codes provided in this section is the generalization of the Steane construction. We have previously shown in Proposition 3.2.4 that over the field $\mathbb{F}_2$, the Steane and the Calderbank and Shor constructions are equivalent. The construction by Calderbank and Shor can also be generalised to $\mathbb{F}_q$, and it can be shown that it is equivalent to the generalized Steane construction in Definition 3.3. However, showing it is beyond the scope of the thesis.

3.4 Correcting quantum errors with classical codes

In their pivotal paper [4], Calderbank and Shor prove that the CSS construction over $\mathbb{F}_2$ gives quantum error-correcting codes whose error correction capacity depends on a pair of two linear codes and their parameters. They present this in [4, Theorem 1]. In this section we will discuss which conditions are necessary for a quantum code to be error-correcting and which conditions of this theorem we can relax. Moreover, we will generalize the theorem to quantum CSS codes Q constructed from linear codes defined over $\mathbb{F}_q$, as in Proposition 3.4.2.

See the original theorem by Calderbank and Shor bellow in Theorem 3.4.1.

Theorem 3.4.1. *(Theorem 1 from [4]) Let C_1, C_2 be binary linear codes such that $C_2 \subseteq C_1 \subseteq \mathbb{F}_2^n$ and C_1,$C_2^\perp$ have both length n, dimension $n-k$ and minimum distance d. Then the quantum code Q_{C_1,C_2} can correct up to $t = \lfloor \frac{d-1}{2} \rfloor$ errors.*

The Calderbank and Shor construction presented in Section 3.1 gives a CSS code Q which is according to Theorem 3.4.1 a t-error-correcting code with $t = \lfloor \frac{d(C_1)-1}{2} \rfloor$. In Theorem 3.4.1 two conditions on linear codes C_1 and C_2 are posed. We will now discuss the importance of these conditions to show where they can be relaxed.

The first condition states that C_1 and C_2 are such that $C_2 \subseteq C_1 \subseteq \mathbb{F}_2^n$. This is a crucial part of the definition of the CSS code in Section 3.1 and Section 3.2, since $C_2 \subseteq C_1$ implies $C_1^\perp \subseteq C_2^\perp$ and then we can take the set $C_2^\perp / C_1^\perp$ to be the index set for each codeword $|c_w\rangle$. One could argue that instead of taking the dual of some code C_1 to be inside $C_2^\perp$, we can take any code $C \subseteq C_2^\perp$. In this case the natural index set would be $C_2^\perp / C$. However, this is only a matter of labeling as any code inside $C_2^\perp$ can be written as the dual of another code that contains C_2, namely as the dual of its dual code since in finite dimensional codes we have $(C^\perp)^\perp = C$ for every linear code $C \subseteq \mathbb{F}_2^n$.

The second condition states that C_1 must have the same parameters as $C_2^\perp$, that is same length, dimension and minimum distance. First, the equal length n follows directly from the condition of C_2 being a subcode of C_1. Secondly, the condition that $\dim(C_1) = \dim(C_2^\perp) = n-k$ is taken by Calderbank and Shor for simplicity, however they state in [4, Section 4] that it is not necessary. Therefore this condition can later be dropped. Finally, the minimum distance must satisfy $d(C_1) = d(C_2^\perp) = d$ for Q_{C_1,C_2} to correct $t = \lfloor \frac{d-1}{2} \rfloor$ errors. This condition can be relaxed to taking $d := \min\{d(C_1), d(C_2^\perp)\}$.

We relax Theorem 3.4.1 by dropping two conditions and by generalizing C_1 and C_2 to be defined over $\mathbb{F}_q$. The relaxed version of Theorem 3.4.1 is presented in the following Proposition 3.4.2.

Proposition 3.4.2. (Generalization of Theorem 3.4.1) Let C_1, C_2 be linear codes with $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$. Fix $d := \min\{d(C_1), d(C_2^\perp)\}$. Then, the quantum code Q_{C_1,C_2} can correct up to $t = \lfloor \frac{d-1}{2} \rfloor$ errors.

Before proving Proposition 3.4.2, we need the following lemma that is an extension to $\mathbb{F}_q$ of [4, Lemma 1] that is used in the proof of Theorem 3.4.1. The use of this lemma is not directly mentioned in this thesis, because it does not differ from how it is used in the original proof.

Lemma 3.4.3. Let $C \subseteq \mathbb{F}_q^n$ be a linear code and $e, E \in \mathbb{F}_q^n$ be two vectors such that $e \leq E$. Assume that $\mathrm{wt}(E) \leq d(C^\perp)$. Then there exists a codeword v_e such that $v_{e|_{\mathrm{supp}(E)}} = e$.

Proof. Let the projection map onto the support of E be

$$\begin{aligned} \pi_E : \mathbb{F}_q^n &\longrightarrow \mathbb{F}_q^n{}_{|_{\mathrm{supp}(E)}} \\ v &\longmapsto v_{|_{\mathrm{supp}(E)}} \end{aligned}$$

and assume that $\pi_{E|_C}$ is not surjective. Then we have

$$d' := \dim(\pi_E(C)) < \dim\left(\mathbb{F}_q^n{}_{|_{\mathrm{supp}(E)}}\right),$$

where $\mathbb{F}_q^n{}_{|_{\mathrm{supp}(E)}} = \{v \in \mathbb{F}_q^n \mid v_i = 0 \text{ if } i \notin \mathrm{supp}(E)\}$. That means that $\pi_E(C)$ is generated by d' number of standard basis vectors e_i, thus there is at least one $j \in \mathrm{supp}(E)$ such that for all $c \in \pi_E(C)$ we have $(c)_j = 0$. Note that since π_E is a projection, we have that if $i \in \mathrm{supp}(E)$ then $(c_{|_{\mathrm{supp}(E)}})_i = (c)_i$ and 0 otherwise, so this implies that for all $c \in C$ we have $(c)_j = 0$. Then we have

$$\langle e_j | c \rangle = \sum_{i=1}^{n} (e_j)_i \cdot (c)_i = (e_j)_j \cdot (c)_j = 0,$$

hence $e_j \in C^\perp$. Recall that we take e_j from $F_q^n{}_{|_{\mathrm{supp}(E)}}$ so we have $\mathrm{wt}(e_j) \leq \mathrm{wt}(E)$. By our assumption on the weight of E, we get $\mathrm{wt}(e_j) \leq \mathrm{wt}(E) < d(C^\perp)$ which is a contradiction. Therefore $\pi_{E|_C}$ is surjective and consequently for every e such that $e \leq E$ we can find $w \in C$ such that $w_{|_{\mathrm{supp}(E)}} = e$. □

Now we are ready to prove Proposition 3.4.2. The proof is very similar to the one of [4, Theorem 1], therefore we will not provide a full proof, but we will only focus on the parts that differ from the original proof by Calderbank and Shor. In what follows we will use the notation for CSS codes as defined in Definition 3.3.1.

Proof. Let us take $|w + C_2\rangle$ with $w \in C_1/C_2$ and let E be a vector in $\mathbb{F}_q$ such that there is a nonzero element from $\mathbb{F}_q$ on those qubits where the error happened and the other entries are zero. Then $\operatorname{supp}(E)$ represents the set of the indexes where the error occurred. Since we want to correct t errors, we assume $\operatorname{wt}(E) = t$. The proof is split into two parts: first we correct the bit-flip errors, secondly the phase-flip errors.

The codeword $|w + C_2\rangle$ with errors, denoted by the the action D, looks as follows

$$D\,|w + C_2, a_0\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{c\in C_2} \sum_{e\le E} |w + c + e\rangle \left| a_{c|_{\operatorname{supp}(E)},e} \right\rangle, \tag{3.2}$$

where $|a_i\rangle$ denotes the state of the environment with $|a_0\rangle$ being the initial state. To restore $|w + C_2\rangle$, note that $w \in C_1$ and every $c \in C_2 \subseteq C_1$, hence also $w + c \in C_1$. Then since $d \le d_1$ and $t = \lfloor \frac{d-1}{2} \rfloor$ we obtain $2t < d \le d_1$, which means the minimum distance between two codewords in C_1 is more than two times larger than t, which is the weight of E. Hence we can uniquely decode every $w+c+e$ to $w+c$. This corrects the bit-flip errors in $|w + C_2\rangle$. This is done by applying a unitary operation R_{bf} after which the quantum state looks as follows

$$R_{\mathrm{bf}} D\,|w + C_2, a_0\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{c\in C_2} \sum_{e\le E} |w + c\rangle \left| a_{c|_{\operatorname{supp}(E)},e} \right\rangle |A_e\rangle\,, \tag{3.3}$$

where $|A_c\rangle$ is the set of ancilla qubits where we record the error e.

In the second part, following the proof of [4, Theorem 1] we have to change the basis of $|w + C_2\rangle$ in order to be able to correct the phase-flip errors. This is because correcting the bit-flip error in the codeword with changed basis is actually the same as correcting the phase-flip error in the original basis. The change of basis looks as follows

$$|w + c\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{u\in\mathbb{F}_q^n/C_2} |u + C_2\rangle\,.$$

After applying the decoding and obtaining what is written in Equation (3.3), we apply substitution with the new basis and we get the following equation

$$R_{\mathrm{bf}}(D\,|w + C_2\rangle) = \frac{1}{|C_2|} \sum_{c\in C_2} \sum_{e\le E} \left| a_{c|_{\operatorname{supp}(E)},e} \right\rangle |A_e\rangle \sum_{u\in\mathbb{F}_q^n/C_2} |u + C_2\rangle\,. \tag{3.4}$$

Next, we show that there is at most one $w \in C_1/C_2$ such that we can correct $|u + C_2\rangle$ to $|w + C_2\rangle$ for every $u \in \mathbb{F}_q^n/C_2$. Assume $w_1, w_2 \in C_1/C_2$ are such that

$$\begin{aligned} u &= w_1 + e_1 + c_1 \\ u &= w_2 + e_2 + c_2 \end{aligned}$$

Then

$$\begin{aligned} e_1 + (q-1)e_2 &= q\cdot u - (w_1 + (q-1)w_2) - (c_1 + (q-1)c_2) \\ &= -(w_1 + (q-1)w_2) - (c_1 + (q-1)c_2) \in C_1. \end{aligned}$$

Note however that $\operatorname{wt}(e_1+e_2) \le \operatorname{wt}(e_1)+\operatorname{wt}(e_2) \le 2t$, and since we know $d_1 \ge d \ge 2t$, this shows that $e_1 = e_2$. That implies $w_1 + w_2 \in C_2$ and $|w_1 + C_2\rangle = |w_2 + C_2\rangle$. This shows that every $|u + C_2\rangle$ with $u \in \mathbb{F}_2^n/C_1^\perp$ in Equation (3.4) can be uniquely corrected to a $|w + C_2\rangle \in Q_{C_1,C_2}$, since there is at most one w with $d(w, u) < t$. Finally, we have corrected the bit-flip and phase-flip error, which shows that Q_{C_1,C_2} is a t-error-correcting quantum code. □

Chapter 4

Lower bound for the number of CSS codes

The CSS construction uses a pair of linear codes such that $C_2 \subseteq C_1 \subseteq \mathbb{F}_q$. We have also seen that the correction capacity of a CSS code is based on the minimum distances of C_1 and $C_2^\perp$. Namely, a CSS code can correct up to $\lfloor \frac{d-1}{2} \rfloor$ errors, where $d := \min(d(C_1), d(C_2^\perp))$. A direct way to obtain a CSS code that satisfies the necessary condition and that maximizes the correction capacity is to replace the pair (C_1, C_2) by $(C^\perp, C)$ with a self-orthogonal code C. For this reason most of the work on CSS codes has been focusing on self-orthogonal codes. Such CSS codes can be seen for example in [8, Example 13.1.6 and 13.1.7] or in the Steane code [15]. Even though these examples of CSS quantum codes use self-orthogonal codes, self-orthogonality is not stated as a necessary condition for the CSS construction. The construction also works for pairs $(C_1,\, C_2)$ where $C_1 \neq C_2^\perp$ with C_2 self-orthogonal. Some work has been undertaken in this direction, an example of CSS code derived from two LDPC codes C_1, C_2 such that $C_1 \neq C_2^\perp$ can be found in [10].

Using a self-orthogonal code is useful, because we only need one code and the minimum distance of the quantum code is the same as the one of the code we fixed. However, self-orthogonal codes are not dense, and considering only them is very restricting for the CSS construction. Therefore, it is natural to ask whether there are many pairs of codes which can be used to construct CSS codes without restricting to self-orthogonal codes. In this chapter we approximate the number of such codes. First, in Section 4.1, we investigate when two CSS codes Q_{C_1,C_2} and $Q_{C_1',C_2'}$ are equal. Then, in Section 4.2, we prove a lower bound on the number of CSS codes.

4.1 Equality for CSS codes

In this section we study when two CSS codes are equal. It is important to know the relation between two pairs of linear codes (C_1, C_2) and (C_1', C_2') when two CSS codes $Q_{C_1,C_2}, Q_{C_1',C_2'}$ are equal, so that we are able to count the number of CSS codes based on the number of pairs (C_1, C_2). We use the definition of CSS codes over $\mathbb{F}_q$ as given in Definition 3.3.1.

Proposition 4.1.1. Let $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$ and $C_2' \subseteq C_1' \subseteq \mathbb{F}_q^n$ be linear codes. Let us consider the CSS codes Q_{C_1,C_2}, $Q_{C_1',C_2'}$. Then the following holds

$$Q_{C_1,C_2} = Q_{C_1',C_2'} \iff C_1 = C_1' \text{ and } C_2 = C_2'.$$

Proof. The implication from right to left is obvious, therefore we will focus on proving the other direction.

We will first prove that $C_1 = C_1'$. Suppose by contradiction that $C_1 \neq C_1'$, then there is at least one codeword $\bar{w}$ such that $\bar{w} \in C_1 \backslash C_1'$ or $\bar{w} \in C_1' \backslash C_1$. Without loss of generality we can assume that $\bar{w} \in C_1 \backslash C_1'$. Now take $c = \mathbf{0} \in C_2 \subseteq C_1$, then $\bar{w} + c \notin C_1'$. Hence there exist no $w' \in C_1'$ and $c' \in C_2' \subseteq C_1'$ for which we would have that $\bar{w} + c = w' + c'$. Let us recall the definition of $|\bar{w} + C_2\rangle$

$$|\bar{w} + C_2\rangle = \frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |\bar{w} + c\rangle\,.$$

Since $\bar{w} + c \neq w' + c'$ for all $w' \in C_1'$ and $c' \in C_2'$, we conclude that $|\bar{w} + C_2\rangle \neq |w' + C_2'\rangle$ for any $w' \in C_2'$. This is a contradiction and hence we must have $C_1 = C_1'$.

Now we will show that $|C_2| = |C_2'|$. Since $Q_{C_1,C_2} = Q_{C_1',C_2'}$, that means that for every $|w + C_2\rangle \in Q_{C_1,C_2}$ there is a unique codeword $|w' + C_2'\rangle \in Q_{C_1',C_2'}$ such that $|w + C_2\rangle = |w' + C_2'\rangle$. Rewriting this equality gives

$$\frac{1}{\sqrt{|C_2|}} \sum_{c \in C_2} |w + c\rangle = \frac{1}{\sqrt{|C_2'|}} \sum_{c' \in C_2'} |w' + c'\rangle \,. \tag{4.1}$$

Recall that every ket is equal to a vector of the standard basis, hence the sums in Equation 4.1 contain $|C_2|$ and $|C_2'|$ distinct orthogonal vectors, respectively. Hence every $|w + c\rangle$ is equal to exactly one $|w' + c'\rangle$, implying by Remark 2.2.1 that every $w + c$ is equal to exactly one $w' + c'$. This implies that the two sums must contain the same number of elements, hence $|C_2| = |C_2'|$.

Finally, we prove that also $C_2 = C_2'$. Take $|w + C_2\rangle \in Q_{C_1,C_2}$ and $|w' + C_2'\rangle \in Q_{C_1,C_2'}$ such that $|w + C_2\rangle = |w' + C_2'\rangle$. Taking $c = \mathbf{0} \in C_2$ gives $|w + \mathbf{0}\rangle = |w' + c'\rangle$ for some $c' \in C_2'$, which then implies $w = w' + c'$ and hence that $w - w' \in C_2'$. Since $|w + C_2\rangle = |w' + C_2'\rangle$, for all $c \in C_2$ there exists a unique $c' \in C_2'$ such that $|w + c\rangle = |w' + c'\rangle$, which entails $c' - c = w - w' \in C_2'$. Therefore, we also have $c = c' - (c' - c) \in C_2'$ and thus $C_2 \subseteq C_2'$, which together with $|C_2| = |C_2'|$ implies $C_2 = C_2'$. □

By Proposition 4.1.1 it follows that different CSS codes correspond to different pairs of linear codes. More precisely, the number of t-error-correcting codes Q_{C_1,C_2} over $\mathbb{F}_q$ of length q^n and dimension k is the same as the number of linear codes (C_1, C_2) where $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$, $d(C_1), d(C_2^\perp) \geq d$, where $t = \lfloor \frac{d-1}{2} \rfloor$ and $\dim(C_1) - \dim(C_2) = k$. Recall that the requirement for the dimension follows by Remark 3.1.8 and minimum distance follows by Proposition 3.4.2. Hence the lower bound of the number of t-error-correcting CSS codes of dimension k should be derived for fixed parameters (q, n, k, d).

4.2 Approximating the number of CSS codes

In this section we will present a lower bound on the number of CSS codes. From Proposition 3.4.2 and the discussion after it, it follows that the number of t-error-correcting CSS codes Q_{C_1,C_2} over $\mathbb{F}_q$ with length q^n and dimension k is the same as the number of different pairs $(C_1,\, C_2)$ such that $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$ with $d(C_1), d(C_2^\perp) \geq d$ and $k = \dim(C_1) - \dim(C_2)$. Therefore we will focus on counting such pairs. Throughout this section we fix integers (q, n, k, d), where $1 \leq k \leq n$, $1 \leq d$ and $q > 0$ is a power of a prime.

4.2.1 Lower bounds for the number of codes C_1 and the number of codes C_2

To find the number of pairs of linear codes (C_1, C_2) with the properties we are interested in, we proceed in the following way. First, in Lemma 4.2.1, we approximate the number of linear codes C_1 with $d(C_1) \geq d$ for fixed (q, n, k, d). After that, in Lemma 4.2.2, we approximate the number of subcodes C_2 of a code C_1 for fixed (q, n, k, d) and fixed $\dim(C_1) = k_1$. Note that by fixing the dimension of C_1, we obtain by Remark 3.1.8 fixed dimension also for C_2, namely $\dim(C_2) = k_1 - k$.

Lemma 4.2.1. (Lower bound for the number of C_1's) Let q be the power of a prime, let n, k_1, d be integers such that $1 \leq k \leq n$ and $2 \leq d$. Set $\mathcal{F}_{C_1} := \{C_1 \subseteq \mathbb{F}_q^n \mid \dim(C_1) = k_1,\, d(C_1) \geq d\}$. Then we have

$$|\mathcal{F}_{C_1}| \geq \binom{n}{k_1}_q \left(1 - \frac{q^{k_1} - 1}{(q^n - 1)(q - 1)} \left(\sum_{i=0}^{d-1} (q-1) \binom{n}{i}_q - 1\right)\right).$$

Proof. By the second part of Proposition 1.3.3, we know that there are at most

$$\frac{q^{k_1} - 1}{(q^n - 1)(q - 1)} \binom{n}{k_1}_q (\boldsymbol{b}(d-1) - 1)$$

number of codes $C_1 \subseteq \mathbb{F}_q^n$ of dimension k_1 such that $d(C_1) < d$ where $0 \leq k_1 \leq n$ and $d \geq 2$. This proposition however approximates all the codes with minimum distance smaller than a fixed parameter d, but what we need is the exact opposite, namely the number of codes with minimum distance greater or equal to d. Therefore we will subtract this result from the number of all the k_1-dimensional codes in $\mathbb{F}_q^n$, which is by Proposition 1.3.1 equal to

$$\binom{n}{k_1}_q = \frac{(q^n-1)(q^n-q)\dots(q^n-q^{k_1-1})}{(q^{k_1}-1)(q^{k_1}-q)\dots(q^{k_1}-q^{k_1-1})}.$$

Note that by Remark 1.2.9 we have that $\boldsymbol{b}(d-1) = \sum_{i=0}^{d-1}(q-1)^i\binom{n}{i}$. Therefore the lower bound of linear codes C_1 with the desired properties is

$$|\mathcal{F}_{C_1}| \geq \binom{n}{k_1}_q \left(1 - \frac{q^{k_1}-1}{(q^n-1)(q-1)}\left(\sum_{i=0}^{d-1}(q-1)^i\binom{n}{i} - 1\right)\right).$$

□

Lemma 4.2.2. (Lower bound for the number of C_2's) Fix an integer $d \geq 2$ and let $C_1 \subseteq \mathbb{F}_q^n$ be a linear code with $\dim(C_1) = k_1$ and $d \leq d(C_1)$. Let us define the family of codes

$$\mathcal{F}_{C_2} \coloneqq \{C_2 \subseteq \mathbb{F}_q^n \,|\, C_2 \subseteq C_1,\, \dim(C_2) = k_2,\, d(C_2^\perp) \geq d\}.$$

Then we have that

$$|\mathcal{F}_{C_2}| \geq \binom{k_1}{k_1-k_2}_q \left(1 - \frac{q^{n-k_2}-q^{n-k_1}}{(q-1)(q^n-q^{n-k_1})}\left(\sum_{i=0}^{d-1}(q-1)^i\binom{n}{i} - 1\right)\right).$$

Proof. Fix $C_1 \subseteq \mathbb{F}_q^n$ with $\dim(C_1) = k_1$ and an integer $d \geq 2$. The family $\mathcal{F}_{C_2}$ is equal to the following set

$$\begin{aligned}\mathcal{F}_{C_2} &= \{C_2 \subseteq \mathbb{F}_q^n \,|\, C_2 \subseteq C_1,\, \dim(C_2) = k_2,\, d(C_2^\perp) \geq d\}\\ &= \{C_2^\perp \subseteq \mathbb{F}_q^n \,|\, C_1^\perp \subseteq C_2^\perp,\, \dim(C_2^\perp) = n-k_2,\, d(C_2^\perp) \geq d\}\\ &= \{C_2^\perp \subseteq \mathbb{F}_q^n \,|\, C_1^\perp \subseteq C_2^\perp,\, \dim(C_2^\perp) = n-k_2\} \setminus \{C_2^\perp \subseteq \mathbb{F}_q^n \,|\, C_1^\perp \subseteq C_2^\perp,\, \dim(C_2^\perp) = n-k_2,\, d(C_2^\perp) < d\}\\ &= \mathcal{F}_{n-k_2} \setminus \mathcal{F}_{n-k_2,<d}\end{aligned} \tag{4.2}$$

with the notation as in Proposition 1.3.3. This result implies that $|\mathcal{F}_{C_2}| = |\mathcal{F}_{n-k_2}| - |\mathcal{F}_{n-k_2,<d}|$. Recall that by Proposition 1.3.2 we have that

$$|\mathcal{F}_{n-k_2}| = \binom{n-(n-k_1)}{(n-k_2)-(n-k_1)}_q = \binom{k_1}{k_1-k_2}_q, \tag{4.3}$$

and by Proposition 1.3.3 we know that if $d \geq 2$ then

$$|\mathcal{F}_{n-k_2,<d}|/|\mathcal{F}_{n-k_2}| \leq \frac{q^{n-k_2}-q^{n-k_1}}{(q-1)(q^n-q^{n-k_1})}\left(\boldsymbol{b}(d-1)-1\right). \tag{4.4}$$

Equations (4.3) and (4.4) together allow to compute $|\mathcal{F}_{n-k_2,<d}|$ as follows

$$\begin{aligned}|\mathcal{F}_{n-k_2,<d}| &= (|\mathcal{F}_{n-k_2,<d}|/|\mathcal{F}_{n-k_2}|) \cdot |\mathcal{F}_{n-k_2}|\\ &\leq \frac{q^{n-k_2}-q^{n-k_1}}{(q-1)(q^n-q^{n-k_1})}\left(\boldsymbol{b}(d-1)-1\right) \cdot \binom{k_1}{k_1-k_2}_q.\end{aligned} \tag{4.5}$$

Finally, combining Equations (4.2),(4.3) and (4.5) we obtain the desired result

$$\begin{aligned}|\mathcal{F}_{C_2}| &\geq \binom{k_1}{k_1-k_2}_q - \left(\frac{q^{n-k_2}-q^{n-k_1}}{(q-1)(q^n-q^{n-k_1})}\left(\boldsymbol{b}(d-1)-1\right) \cdot \binom{k_1}{k_1-k_2}_q\right)\\ &= \binom{k_1}{k_1-k_2}_q \left(1 - \frac{q^{n-k_2}-q^{n-k_1}}{(q-1)(q^n-q^{n-k_1})}\left(\boldsymbol{b}(d-1)-1\right)\right)\\ &= \binom{k_1}{k_1-k_2}_q \left(1 - \frac{q^{n-k_2}-q^{n-k_1}}{(q-1)(q^n-q^{n-k_1})}\left(\sum_{i=0}^{d-1}(q-1)^i\binom{n}{i} - 1\right)\right).\end{aligned}$$

□

Before we prove a lower bound for the pairs of (C_1, C_2), we examine how each of the bounds in Lemma 4.2.1 and Lemma 4.2.2 behave for different parameters (q, n, k, d).

4.2.2 Examples of the bounds for the number of C_1 and C_2

The strategy of approximating the lower bound for the number of pairs (C_1, C_2) is to first approximate the number of codes C_1 and then multiply it with the approximation for the number of codes C_2. For this reason we first examine how the bound for the number of C_1's in Lemma 4.2.1 behaves for different parameters (q, n, k_1, d). Note that we here fix $k_1 = \dim(C_1)$ instead of $k = \dim(Q)$, which will be the fixed parameter in the final lower bound. We use a simple function called `lowerboundC1` in the computer algebra system SageMath to generate the results in the following examples. The algorithm can be found in the Appendix 6.

Recall that $d \geq 2$ and $1 \leq k_1 \leq n$. We skip the case when $k_1 = n$ because for that case $C_1 = \mathbb{F}_q^n$. We denote the number of codes C_1 with $d(C_1) \geq d$ by $|\mathcal{F}_{C_1}|$.

Example 4.2.3. Let us fix $q = 2$ and $n = 4$. We vary $1 \leq k_1 < n$ and look at the cases when $d = 2$ and $d = 3$, since for $d = 4$ the only code over $\mathbb{F}_2$ would be the repetition code. The lower bound from Lemma 4.2.1 of the number of codes for parameters (q, n, k_1, d) can be found bellow.

(q, n, k_1, d)	$\lvert\mathcal{F}_{C_1}\rvert$
$(2, 4, 1, 2)$	11
$(2, 4, 2, 2)$	7
$(2, 4, 3, 2)$	-13
$(2, 4, 1, 3)$	5
$(2, 4, 2, 3)$	-35
$(2, 4, 3, 3)$	-55

In Example 4.2.3 we see that when k_1 and d increase, the lower bound decreases and even attains a negative value. Negative values are not interesting for us, since they do not give a meaningful lower bound for the number of codes. Moreover, they can cause problems later on when we will multiply the number of codes C_1 with the number of codes C_2. If both of these values are negative, multiplying them would give a positive value that would be very likely incorrect. For this reason we want to focus on those cases where the lower bound is non-negative.

In the next example we look at what happens when we increase the value of q.

Example 4.2.4. Let us fix $q = 5$, $n = 4$ and $d = 2$ and 3. We vary k_1 such that $1 \leq k_1 < n$. Then we get the following outcomes from the lower bound.

(q, n, k_1, d)	$\lvert\mathcal{F}_{C_1}\rvert$
$(5, 4, 1, 2)$	152
$(5, 4, 2, 2)$	682
$(5, 4, 3, 2)$	32
$(5, 4, 1, 3)$	128
$(5, 4, 2, 3)$	-62
$(5, 4, 3, 3)$	-712

In Example 4.2.4 we see that again for higher values of k_1, d the lower bound gives a lower outcome. Note that also for $q = 5$ we have points where the lower bound is a negative number, however less of them than we had for $q = 2$.

Finally, let us have a look at what happens when we increase the length n.

Example 4.2.5. Let us fix $q = 5$, $k_1 = d = 2$ and then $k_1 = d = 3$. We will vary n such that $n \geq k_1$.

(q, n, k_1, d)	$\lvert\mathcal{F}_{C_1}\rvert$
$(5, 3, 2, 2)$	13
$(5, 5, 2, 2)$	19526
$(5, 4, 3, 3)$	-712
$(5, 6, 3, 3)$	1218360

In Example 4.2.5 we see that a larger value of n gives a larger value of the lower bound. Therefore we conclude that to avoid getting negative outputs, we want to focus on the cases when k_1, d are relatively small and n is relatively large.

Now we will briefly see what happens with the lower bound of the number of subcodes C_2 for those cases when the bound in Lemma 4.2.1 gave a negative outcome. Some instances are reported in Example 4.2.6.

Example 4.2.6. In this example we look at four cases when the number $|\mathcal{F}_{C_1}|$ was negative. We print $|\mathcal{F}_{C_2}|$ from Lemma 4.2.2 for the parameters (q, n, k_1, k_2, d).

(q, n, k_1, k_2, d)	$\lvert\mathcal{F}_{C_2}\rvert$
$(2, 4, 2, 1, 2)$	-1
$(2, 4, 3, 1, 2)$	-5
$(5, 4, 3, 1, 3)$	-137
$(5, 4, 3, 2, 3)$	3

In Example 4.2.6 we see that if we take parameters for which Lemma 4.2.1 gives a negative outcome, the outcome of Lemma 4.2.2 might also be negative. This is a problem since as mentioned before, multiplying the number of C_1's with the number of C_2's could in these cases give a large positive value, even though the real number of pairs could be 0. For this reason, we must restrict the lower bound for the number of pairs (C_1, C_2) for those cases when the bounds in Lemma 4.2.1 and Lemma 4.2.2 are both non-negative.

4.2.3 Lower bound for the number of CSS codes

In this section we combine the two results from Lemma 4.2.1 and Lemma 4.2.2 to give a lower bound on the number of t-error-correcting CSS codes. As mentioned before, we consider only the cases when both $|\mathcal{F}_{C_1}| \geq 0$ and $|\mathcal{F}_{C_2}| \geq 0$, and if that holds, we multiply them with each other.

Proposition 4.2.7. (Lower bound for the number of CSS codes) Let (q, n, k, d) be integers such that q is a power of a prime, $1 \leq k \leq n$ and $d \geq 2$. Set $t = \lfloor \frac{d-1}{2} \rfloor$. Suppose that both $|\mathcal{F}_{C_1}|$ and $|\mathcal{F}_{C_2}|$ from Lemma 4.2.1 and Lemma 4.2.2 are non-negative numbers. Then the number of t-error-correcting CSS codes Q_{C_1,C_2} of dimension k with $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$ and with $d(C_1), d(C_2^\perp) \geq d$, is at least

$$\sum_{k_1=k+1}^{n} \binom{n}{k_1}_q \binom{k_1}{k}_q \left(1 - \frac{q^{k_1}-1}{(q^n-1)(q-1)} (\boldsymbol{b}(d-1)-1)\right) \left(1 - \frac{q^{n-(k_1-k)} - q^{n-k_1}}{(q-1)(q^n - q^{n-k_1})} (\boldsymbol{b}(d-1)-1)\right).$$

Proof. From Proposition 4.1.1 it follows that counting the CSS codes Q_{C_1,C_2} with error correction capacity equals to $t = \lfloor \frac{d-1}{2} \rfloor$, corresponds to counting the pairs (C_1, C_2) such that $C_2 \subseteq C_1$ and $d(C_1), d(C_2^\perp) \geq d$. Therefore to obtain a lower bound for the number of Q_{C_1,C_2}, we can combine the results of Lemma 4.2.1 and Lemma 4.2.2.

By Lemma 4.2.1 we know that

$$|\mathcal{F}_{C_1}| \geq \binom{n}{k_1}_q \left(1 - \frac{q^{k_1}-1}{(q^n-1)(q-1)} (\boldsymbol{b}(d-1)-1)\right),$$

where $|\mathcal{F}_{C_1}|$ denotes the number of codes $C_1 \subseteq \mathbb{F}_q^n$ with $\dim(C_1) = k_1$ and $d(C_1) \geq d$. For each fixed k_1 which denotes the dimension of these C_1, it follows by Lemma 4.2.2 that

$$|\mathcal{F}_{C_2}| \geq \binom{k_1}{k_1-k_2}_q \left(1 - \frac{q^{n-k_2} - q^{n-k_1}}{(q-1)(q^n - q^{n-k_1})} (\boldsymbol{b}(d-1)-1)\right)$$

where $|\mathcal{F}_{C_2}|$ denotes the number of subcodes $C_2 \subseteq C_1$ with $\dim(C_2) = k_2$ and $d(C_2^\perp) \geq d$.

Fix k to be the dimension of the CSS codes that we want to count. Recall by Remark 3.1.8, if $\dim(C_1) = k_1$ and $\dim(C_2) = k_2$, then $k = k_1 - k_2$. The dimension of C_1 is moreover such that $k \leq k_1 \leq n$ but note that if we take $k = k_1$, then $k_2 = 0$, so we exclude this option. We therefore vary $k < k_1 \leq n$. Since we assume that both $|\mathcal{F}_{C_1}|$ and $|\mathcal{F}_{C_1}|$ are non-negative, we can conclude that the lower bound for the number of pairs of linear codes (C_1, C_2) where $C_2 \subseteq C_1 \subseteq \mathbb{F}_q^n$ and $d(C_1), d(C_2^\perp) \geq d$ is the following

$$\sum_{k_1=k+1}^{n} \binom{n}{k_1}_q \binom{k_1}{k}_q \left(1 - \frac{q^{k_1}-1}{(q^n-1)(q-1)}\left(\boldsymbol{b}(d-1)-1\right)\right)\left(1 - \frac{q^{n-(k_1-k)} - q^{n-k_1}}{(q-1)(q^n - q^{n-k_1})}\left(\boldsymbol{b}(d-1)-1\right)\right). \tag{4.6}$$

By Proposition 4.1.1 we know that counting the pairs (C_1, C_2) is the same as counting Q_{C_1,C_2}, hence Equation 4.6 is a lower bound for the number of t-error-correcting CSS codes with dimension k. $\square$

Remark 4.2.8. If we take $\dim(C_1) = k_1 = n$, we have that $C_1 = \mathbb{F}_q^n$ and therefore C_1 has minimum distance $d(C_1) = 1$. That implies that any subcode $C_2 \subseteq C_1$ will satisfy $d(C_2^\perp) \geq d = 1$. For this reason in Chapter 5 we will omit the case when $n = k_1$ and take the sum in Equation 4.6 from $k+1$ up until $n-1$.

In order to see if our lower bound is a good approximation of the actual number of CSS codes, we perform several experiments with the SageMath computer algebra system. We compare these results with our bound in the next chapter.

Chapter 5

Counting the number of CSS codes

In this chapter we will count the exact number of CSS codes Q_{C_1,C_2} using a program in the SageMath computer algebra system and then we will compare the results with the lower bound derived in Chapter 4. Again, counting t-error-correcting codes Q_{C_1,C_2} of dimension k where $t = \lfloor \frac{d-1}{2} \rfloor$ is by Proposition 4.1.1 the same as counting the pairs (C_1, C_2) with $C_2 \subseteq C_1$ and $d(C_1), d(C_2^{\perp}) \geq d$. That is why we focus on finding pairs of such linear codes. From now on when mentioning the pair of the codes C_1 and C_2 with these specific properties, we simply say C_1 and C_2 and do not state all the properties anymore.

For simplicity, the SageMath function for counting the number of CSS codes will only consider codes over the field $\mathbb{F}_p$ with p a prime number. The reason for this is that for $q = p^r$ it is more complicated to implement this, since then we work with polynomials instead of numbers.

In Section 5.1 we will explain the function for counting the number of CSS codes for $\mathbb{F}_p$. At the end of this section we will propose a solution on how to extend this algorithm for any field $\mathbb{F}_q$. Then, in Section 5.2 we will compare the results of the lower bound with the exact number of CSS codes.

5.1 Description of the SageMath code `countallpairs`

In this section we will describe the function that counts the number of pairs (C_1, C_2). The function is called `countallpairs` and can be seen below. It consists of two important functions, `countC1` that counts the number of codes C_1 and `countC2` which for each C_1 counts the number of subcodes C_2.

```
def countallpairs(p,n,k,d):
    count = 0
    for k1 in range(k+1,n)
        allmatrices = []
        midcount = 0
        allmatrices = countC1(p,n,k1,d)
        for G1 in allmatrices:
            midcount = midcount + countC2(p,n,k1-k,d,G1)
        print("For (k1,k2)=",k1,",",k1-k," we have number of pairs:",midcount)
        count = count + midcount
    print("The number of all pairs: ",count)
```

For each integer k_1 such that $k+1 \leq k_1 < n$, we want to find the codes C_1. As mentioned in Remark 4.2.8, we exclude the case where $k_1 = n$ because this would give $C_1 = \mathbb{F}_p^n$. Function `countC1` finds all the linear codes C_1 for a fixed dimension k_1. It goes through all linear codes $C \subseteq F_p^n$ with $\dim(C) = k_1$ and checks if $d(C), d(C^{\perp}) \geq d$, where p, n, k, d are fixed parameters. The reason behind checking if $d(C^{\perp}) \geq d$ is to optimize the running time of the program. Since we have the inclusion $C_2 \subseteq C_1$ we automatically have that $C_1^{\perp} \subseteq C_2^{\perp}$, which implies $d(C_1^{\perp}) \geq d(C_2^{\perp})$. By Proposition 3.4.2 we want to have $d(C_2^{\perp}) \geq d$, and that imposes the necessary condition onto $d(C_1^\perp)$ mentioned above. Example 5.1.1 in this section shows the difference between the number of codes C_1 with and without the condition $d(C_1^\perp) \geq d$.

Finding these codes boils down to finding their generator matrices. From Definition 1.2.2 we know that the generator matrix G of a code C is a full rank $k \times n$ matrix in the reduced row echelon form. The function `countC1` will therefore output a list of all the generator matrices in reduced row echelon form for all the linear codes with the desired properties.

Generating all matrices G of a certain form is done in the following way. We first generate all vectors of length n that consist of the elements in $\mathbb{F}_p$ in `list1`. Afterwards only elements with at least d nonzero entries are copied to `list2`, to prevent that the minimum distance $d(C)$ would be smaller than the fixed value d.

```
def countC1(p,n,k1,d):
    list1 = list(product(range(0,p),repeat=n))
    list2 = []
    for i in range(0,len(list1)):
        weight = 0
        for j in range(0,n):
            if list1[i][j] > 0:
                weight = weight + 1
        if weight >= d:
            list2.append(list(list1[i]))
```

After we have generated all the possible vectors that could generate the linear code C_1, we want to find all k_1-tuples of them and then turn these tuples into matrices G. The vectors are stored as `list2[i]` for all $i \in \{1, 2, \ldots, \text{length}(\texttt{list2})\}$, therefore finding all k_1-tuples of them is the same as finding all k_1-tuples of their iterations. This is done using the function `product`. Finally, we can create matrices G.

```
    rows = list(product(range(0,len(list2)),repeat = k1))
    length = len(rows[0])
    for r in rows:
        G = []
        for i in range(0,length):
            G.append(tuple(list2[r[i]]))
        G = matrix(GF(p),G)
        G = G.rref()
```

The last step is to check if the matrices satisfy the criteria in Proposition 3.4.2, namely whether it holds that $d(C), d(C^\perp) \geq d$. Then we return the list of all such matrices with the function `allmatrices`. This list is used in the fourth row of the function `countallpairs` that was presented at the beginning of this section.

```
        if G not in allmatrices:
            if G.rank() == k1:
                C = LinearCode(G)
                if C.minimum_distance() >= d:
                    Cperp = C.dual_code()
                    if Cperp.minimum_distance() >= d:
                        allmatrices.append(G)
    return allmatrices
```

In order to see how much the condition $d(C^\perp) \geq d$ optimizes our program, we also give a function `countC1noperp`, which is a copy of `countC1` but where we drop the criterion on the minimum distance of $C^\perp$. This function is presented in the Appendix 6. The following Example 5.1.1 shows the difference between the results from `countC1` and `countC1noperp` for some fixed parameters.

Example 5.1.1. We fix $(p, n, k, d) = (2, 4, 2, 2)$, then function `countC1` gives the number of codes C_1 with parameters $[p, n, k] = [2, 4, 2]$ and $d(C_1), d(C_1^{\perp}) \geq 2$ to be 9. We obtain the following generator matrices of these codes

$$G_1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad G_2 = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \qquad G_3 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$G_4 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \qquad G_5 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad G_6 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$G_7 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \qquad G_8 = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \qquad G_9 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix}.$$

If we drop the second requirement and run `countC1noperp`, we get that the number of codes C_1 is 13. Apart of matrices $G_1, \ldots, G_9$ we also obtain the following four matrices

$$G_{10} = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad G_{11} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad G_{12} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad G_{13} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}.$$

Each of the matrices $G_{10}, \ldots, G_{13}$ has no subspace C_2 satisfying $d(C_2^{\perp} \geq d)$. Hence we see that including the requirement $d(C_1^{\perp}) \geq d$ optimizes the running time of `countallpairs` by 4 cases for which we do not have to go over all its subspaces.

Note that for these parameters we have that $k = k_1$, which is the case that we otherwise exclude since then $\dim(C_2) = 0$. However, the purpose of this example is to show the usefulness of the condition $d(C_1^{\perp}) \geq d$. Moreover, we do not yet consider the subcode C_2, so for this reason it is not a problem to have $k = k_1$ in this case.

The full code for `countC1` can be found in the Appendix 6.

Now we return to the function `countallpairs` and run a for loop to go over all codes C_1 in `allmatrices`. We fix a code C_1 and can proceed with finding its subspaces C_2.

The function `countC2` fixes a code C_1 found in the previous function and checks all its subcodes, to find all C_2 with $d(C_2^{\perp}) \geq d$ and with $\dim(C_2) = k_1 - k$.

```
def countC2(p,n,k2,d,G1):
    count = 0
    allmatrices = []
    C1 = LinearCode(G1)
    codewords = list(C1)
    codewords.remove(codewords[0])
    generators = Combinations(codewords, k2)
```

The next step is to check if the obtained matrix is of full rank and if the linear code with this generator matrix satisfies the criteria for the minimum distance.

```
    for gens in generators:
        G2 = matrix(GF(p),gens).rref()
        C2 = LinearCode(G2)
        if G2 not in allmatrices:
            if subcode(C2,C1):
                if C2.dimension() == k2:
                    C2perp = C2.dual_code()
                    if C2perp.minimum_distance() >= d:
                        allmatrices.append(G2)
                        count = count + 1
    return count
```

The function then returns the number of subcodes C_2 for a fixed code C_1.

With the number of subcodes C_2 for each code C_1, we return to `countallpairs` where these results are summed. Finally, we return this value as the number of all pairs (C_1, C_2) with the desired properties.

5.1.1 SageMath code for $\mathbb{F}_q$

As mentioned at the beginning of this chapter, we implemented the function `countallpairs` only for codes over $\mathbb{F}_p$ for simplicity. This can however be extrended to codes over the field $\mathbb{F}_q$. If we want to do so, we need to change function `countC1`, since this is the place where all the codes and their generator matrices are generated. The main difference has to be done in the beginning of this function where we define `list1`. We define a field k of q elements representing the field $\mathbb{F}_q$. This field contains polynomials with the variable X. Hence the list `list1` would take polynomials instead of numbers between 0 and q.

```
    Fq = []
    k.<X> = GF(q,'X')
    for a in k:
        Fq.append(a)

    list1 = list(product(Fq,repeat=n))
```

A minor change has to be done in counting the weight of a codeword in function `countC1`, here we simply change $>$ to $\neq$.

```
            if list1[i][j] != 0:
                weight = weight + 1
```

There are more places where the code would have to be updated, however we will not go in depth of this, since we will only count the exact number of codes for codes over $\mathbb{F}_p$. For this the original code presented in Section 5.1 will be sufficient.

5.2 Results

In this section we compare the lower bound given in Proposition 4.2.7 with the outputs of the SageMath function `countallpairs` described in Section 5.1. The lower bound is implemented in SageMath as a function `lowerbound` and can be found in Appendix 6. Recall that to apply Proposition 4.2.7 we must have that both $|\mathcal{F}_{C_1}|, |\mathcal{F}_{C_2}| \geq 0$. For this reason the function `lowerbound` contains an if-condition, that will check whether this holds.

We first want to find out which parameters give us a nonzero lower bound, since those are the ones that we are interested in. For this we run the script below and we print all the nonzero outputs. Note that in the script we can vary parameters d, p and rng, where rng denotes the maximal value for n.

```
rng = 10
d = 3
p = 5
print("rng = ",rng,", d = ",d,", p = ",p)
for n in range(1,rng):
    for k in range(1,n):
        if countcodes(p,n,k,d)!=0:
            print("For (",p,",",n,",",k,",",d,") the lower bound: ", countcodes(p,n,k,d))
```

Some of the outputs of the script can be seen in Example 5.2.1.

Example 5.2.1. In the table below, we can see some outputs of the function `lowerbound` that approximates the number of CSS codes for fixed parameters (p, n, k, d).

(p, n, k, d)	`lowerbound`
$(2, 6, 1, 2)$	465
$(2, 7, 1, 2)$	16368
$(3, 3, 1, 2)$	1
$(3, 7, 2, 2)$	195052
$(5, 5, 2, 2)$	16927
$(5, 5, 3, 2)$	1
$(5, 7, 1, 3)$	5687456775

All the outputs for this script can be found in the Appendix 6. Note that the outputs in Appendix 6 show that that the smaller the value of p, the larger the lowest n value for which the `lowerbound` output is nonzero. For example, for $p = 2$, the first nonzero number is for $n = 6$. However, for $p = 3$ or $p = 5$, the first nonzero value is for $n = 3$. This indicates that for larger values of p the lower bound can be more precise.

Now we pick those parameters (p, n, k, d) for which we want to compare the lower bound to the exact number of codes. The code `countallpairs` is computationally very expensive and therefore we choose the parameters such that $n, p \leq 5$ and $d, k \leq 2$. Note that since we set $n \leq 5$, we automatically exclude results for $p = 2$, because of the previous reasoning. We also do not want the lower bound to be higher than 10 000, because that indicates that the code `countallpairs` will run for a long time.

We will provide some cases where we compare the lower bound to the exact number of codes. These are shown in the examples bellow.

Example 5.2.2. Let us fix $(p, n, k, d) = (3, 3, 1, 2)$. Then the function `lowerbound` outputs 1 as the lower bound for the number of CSS codes and the output from `countallpairs` is 4 CSS codes. Therefore we conclude that the lower bound is a good approximation for these parameters.

Example 5.2.3. Let us fix $(p, n, k, d) = (5, 3, 1, 2)$. Then we have that the function `lowerbound` outputs 39 and the output from `countallpairs` is 48 CSS codes. Therefore we conclude that the lower bound is a good approximation for these parameters.

Example 5.2.4. Let us take $(p, n, k, d) = (3, 5, 1, 2)$. Then we have that the `lowerbound` gives 4480 CSS codes and the output from `countallpairs` is 5360 CSS codes. The approximation is relatively far from the actual number of CSS codes for these parameters.

The results in the examples above show that the values from the function `lowerbound` are indeed lower than the exact number of CSS codes from the function `countallpairs`. We conclude that if $|\mathcal{F}_{C_1}|$, $|\mathcal{F}_{C_2}|$ are positive, then the lower bound is a good approximation of the exact number of the CSS codes.

There are many cases for which the function `lowerbound` gives 0. This is because of the if-condition in this function which checks whether both the numbers of C_1's and C_2's are non-negative. However, it is interesting to look at the number of CSS codes for some of these parameters, for which the lower bound was not positive, to see whether they are close to zero.

Example 5.2.5. Let us take $(p, n, k, d) = (2, 3, 2, 2)$. Then we have that the `lowerbound` gives 0 CSS codes and the output from `countallpairs` is 0 CSS codes. The approximated value 0 is therefore exact for these parameters.

Example 5.2.6. Let us take $(p, n, k, d) = (2, 4, 2, 2)$. Then we have that the `lowerbound` gives 0 CSS codes and the output from `countallpairs` is 1 CSS codes. The approximation 0 is therefore also very precise for these parameters.

Example 5.2.7. Let us take $(p, n, k, d) = (2, 4, 1, 2)$. Then we have that the `lowerbound` gives 0 CSS codes and the output from `countallpairs` is 6 CSS codes. Again, also for these parameters the value 0 is close to the real number of CSS codes.

We see that in the examples above, all the values of the numbers of the CSS codes are very close or equal to zero. Therefore, for small parameters such as those that were considered, we can conclude that the numbers of the CSS codes are also well approximated. This could however behave different for larger values of the parameters.

Chapter 6

Conclusion

In this thesis we studied CSS codes, which are a family of quantum error-correcting codes. The original definition of these codes was generalized to arbitrary finite fields $\mathbb{F}_q$ where q is a power of a prime and where the previous restrictions of the code dimensions were dropped. This new definition is given in Definition 3.3.1. Afterwards, we relaxed the original Theorem by Calderbank and Shor, where they show that CSS codes are quantum error-correcting codes. This was done by providing an alternative proof in Proposition 3.4.2. The main result of this thesis is an approximation of the number of CSS codes, which is presented as a lower bound in Proposition 4.2.7. The results of the approximation were compared to the exact numbers of CSS codes, which were computed by implementing a program in the SageMath computer algebra system. Because the SageMath program was computationally expensive, the numbers of codes that we compared were restricted to those with parameters $p, n \leq 5$ and $d, k \leq 2$. For these parameters, the lower bound approximates the number of CSS codes well.

This report also contains smaller results such as showing that CSS codes do not necessarily have to be based on self-orthogonal codes, even thought most of the existing examples are such. We have also shown that two CSS codes are equal if and only if their linear codes are equal in Proposition 4.1.1 which was crucial for proving the lower bound in Proposition 4.6.

Further research could include a couple of points. In the short-term, first, the SageMath program could be optimized to calculate the number of CSS codes in shorter amount of time. That way the results of the lower bound could be compared for codes with larger parameters. Secondly, the lower bound could be extended to approximate the number of codes closer to the real number of codes also for smaller values of q such as $q = 2$ or $q = 3$. Lastly, one can prove the equivalence of the Steane construction with the Calderbank and Shor construction over $\mathbb{F}_q$, that was not included in this thesis.

In the medium-term we can study whether there are different ways of using classical linear codes to correct quantum errors. In this thesis we only studied the CSS construction.

In the long term one can study whether there are different ways to approximate the number of CSS codes. The lower bound in this thesis is based on the paper by Byrne and Ravagnani [3]. However, using different theory could lead to different and maybe more accurate approximations of the number of CSS codes.

Bibliography

[1] S. Ball, A. Centelles, and F. Huber, *Quantum error-correcting codes and their geometries*, arXiv preprint arXiv:2007.05992, (2020).

[2] D. Bartoli, M. Montanucci, and G. Zini, *Ag codes and ag quantum codes from the ggs curve*, Designs, Codes and Cryptography, 86 (2018), pp. 2315–2344.

[3] E. Byrne and A. Ravagnani, *Partition-balanced families of codes and asymptotic enumeration in coding theory*, Journal of Combinatorial Theory, Series A, 171 (2020), p. 105169.

[4] A. R. Calderbank and P. W. Shor, *Good quantum error-correcting codes exist*, Physical Review A, 54 (1996), p. 1098.

[5] S. Kumar, R. Calderbank, and H. D. Pfister, *Reed-muller codes achieve capacity on the quantum erasure channel*, in 2016 IEEE International Symposium on Information Theory (ISIT), 2016, pp. 1750–1754.

[6] G. G. La Guardia, *Asymmetric quantum reed-solomon and generalized reed-solomon codes*, Quantum Information Processing, 11 (2012), pp. 591–604.

[7] N. D. Mermin, *Quantum computer science: an introduction*, Cambridge University Press, 2007.

[8] G. Nebe, E. M. Rains, and N. J. A. Sloane, *Self-dual codes and invariant theory*, vol. 17, Springer, 2006.

[9] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*, 2002.

[10] M. S. Postol, *A proposed quantum low density parity check code*, arXiv preprint quant-ph/0108131, (2001).

[11] J. Preskill, *Quantum computation*. `http://theory.caltech.edu/~preskill/ph229/notes/chap7.pdf`.

[12] A. Ravagnani, *Coding theory*. `https://a.ravagnani.win.tue.nl/coding%20th/Coding_theory_notes.pdf`.

[13] J. Roffe, *Quantum error correction: an introductory guide*, Contemporary Physics, 60 (2019), pp. 226–245.

[14] P. K. Sarvepalli and A. Klappenecker, *Nonbinary quantum reed-muller codes*, in Proceedings. International Symposium on Information Theory, 2005. ISIT 2005., IEEE, 2005, pp. 1023–1027.

[15] A. Steane, *Multiple-particle interference and quantum error correction*, Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 452 (1996), pp. 2551–2577.

[16] J. H. Van Lint, *Introduction to coding theory*, vol. 86, Springer Science & Business Media, 2012.

[17] Y. Wang, Z. Hu, B. C. Sanders, and S. Kais, *Qudits and high-dimensional quantum computing*, Frontiers in Physics, 8 (2020), p. 589504.

Appendix

Function for counting CSS codes

```
from itertools import permutations,product,combinations

#Parameters for the function
p = var("p")
n = var("n")
k = var("k")
d = var("d")

#Function countC1 generates and counts all codes C in F_p^n with dimension dim(C1)=k1
#such that k<k1<n and with minimal distance d(C) >= d (for some fixed d) and then checks
#if d(C^perp) >= d
def countC1(p,n,k1,d):
    allmatrices = []
    count = 0
    if (n>=k1):
        #List 1 contains all the vectors consisting of numbers between 0 and p of length
        list1 = list(product(range(0,p),repeat=n))
        list2 = []
        #Go over every element of list1 and check if weight is larger or equal to d, if
        #yes add it into list2
        for i in range(0,len(list1)):
            weight = 0
            for j in range(0,n):
                if list1[i][j] > 0:
                    weight = weight + 1
            if weight >= d:
                list2.append(list(list1[i]))

        #Now we have all the possible rows of matrix G stored in list2
        #We find all the k-tuples of elements of list2, which correspond to all matrices G
        rows = list(product(range(0,len(list2)),repeat = k1))

        #Contruction of matrix G
        length = len(rows[0])
        for r in rows:
            G = []
            for i in range(0,length):
                G.append(tuple(list2[r[i]]))
            G = matrix(GF(p),G)
            G = G.rref()
            #Check if G has not occured before and whether C satisfies the criteria
            if G not in allmatrices:
                if G.rank() == k1:
                    C = LinearCode(G)
                    if C.minimum_distance() >= d:
                        Cperp = C.dual_code()
                        if Cperp.minimum_distance() >= d:
                            allmatrices.append(G)
                            count = count + 1

    print("The number of codes:", count)
    return allmatrices
```

```
#Function subcode check if a code C2 is a subcode of the code C1
def subcode(C2,C1):
    for a in C2:
         if a not in C1:
            return 0
    return 1
```

```
#Function countC2 finds all subcodes C2 of a given code C1 such that d(C1^perp)>=d
def countC2(p,n,k2,d,G1):
    count = 0
    allmatrices = []

    #We generate all subspaces using all k2-tuples of all C1s codewords (skip the zero
    #codeword)
    C1 = LinearCode(G1)
    codewords = list(C1)
    codewords.remove(codewords[0])
    generators = Combinations(codewords, k2)

    #We create a generator matrix of the subcode C2 and check if it satisfies all the
    #criteria
    for gens in generators:
        G2 = matrix(GF(p),gens).rref()
        C2 = LinearCode(G2)
        if G2 not in allmatrices:
            if subcode(C2,C1):
                if C2.dimension() == k2:
                    C2perp = C2.dual_code()
                    if C2perp.minimum_distance() >= d:
                        count = count + 1
    return count
```

```
#Function countallpairs finds all pairs (C1,C2) for a quantum code Q_C1,C2
def countallpairs(p,n,k,d):
    count = 0
    #We vary the dimension of C1 as k2<=k1<=n and k=k1-k2
    for k1 in range(k+1,n)
        allmatrices = []
        midcount = 0
        allmatrices = countC1(p,n,k1,d)
        for G1 in allmatrices:
            midcount = midcount + countC2(p,n,k1-k,d,G1)
        print("For (k1,k2)=",k1,",",k1-k," we have number of pairs:",midcount)
        count = count + midcount
    print("The number of all pairs: ",count)
```

Additional functions

```
from itertools import permutations,product,combinations

#Parameters for the function
p = var("p")
n = var("n")
k = var("k")
d = var("d")

#Function countC1noperp generates and counts all codes C in F_p^n with minimal distance
#d(C) >= d (for some fixed d)
def countC1noperp(p,n,k1,d):
    allmatrices = []
    count = 0
    if (n>=k1):
        #List 1 contains all the vectors consisting of numbers between 0 and p of length
        list1 = list(product(range(0,p),repeat=n))
        list2 = []
        #Go over every element of list1 and check if weight is larger or equal to d, if
        #yes add it into list2
        for i in range(0,len(list1)):
            weight = 0
            for j in range(0,n):
                if list1[i][j] > 0:
                    weight = weight + 1
            if weight >= d:
                list2.append(list(list1[i]))

        #Now we have all the possible rows of matrix G stored in list2
        #We find all the k-tuples of elements of list2, which correspond to all matrices G
        rows = list(product(range(0,len(list2)),repeat = k1))

        #Contruction of matrix G
        length = len(rows[0])
        for r in rows:
            G = []
            for i in range(0,length):
                G.append(tuple(list2[r[i]]))
            G = matrix(GF(p),G)
            G = G.rref()

            #Check if G has not occured before and whether C satisfies the criteria
            if G not in allmatrices:
                if G.rank() == k1:
                    C = LinearCode(G)
                    if C.minimum_distance() >= d:
                        allmatrices.append(G)
                        count = count + 1

    print("The number of codes:", count)
    return allmatrices
```

Lower bound from Proposition 4.2.7

```
def lowerboundC1(q,n,k,d):
    b = 0
    for i in range(0,d):
        b = b + (q-1)^i*binomial(n,i)
    output = q_binomial(n,k,q)*(1-((q^k-1)/((q^n-1)*(q-1)))*(b-1))
    return output
```

```
def lowerboundC2(q,n,k1,k2,d):
    b = 0
    for i in range(0,d):
        b = b + (q-1)^i*binomial(n,i)
    output = q_binomial(k1,k1-k2,q)*(1-((q^(n-k2)-q^(n-k1))*(b-1))/((q-1)*(q^n-q^(n-k1))))
    return output
```

```
def lowerbound(q,n,k,d):
    s = 0
    for k1 in range(k+1,n):
        C1 = lowerboundC1(q,n,k1,d)
        C2 = lowerboundC2(q,n,k1,k1-k,d)
        if C1<0:
            C1 = 0
        if C2<0:
            C2 = 0
        s = s + C1*C2
    return s
```

Outputs for the script in Section 5.2

```
rng =  10 , d =  2 , p =  2
For ( 2 , 6 , 1 , 2 ) the lower bound: 465
For ( 2 , 7 , 1 , 2 ) the lower bound: 16368
For ( 2 , 8 , 1 , 2 ) the lower bound: 805434
For ( 2 , 8 , 2 , 2 ) the lower bound: 93345
For ( 2 , 9 , 1 , 2 ) the lower bound: 47670720
For ( 2 , 9 , 2 , 2 ) the lower bound: 30053280
```

```
rng =  10 , d =  2 , p =  3
For ( 3 , 3 , 1 , 2 ) the lower bound: 1
For ( 3 , 5 , 1 , 2 ) the lower bound: 4480
For ( 3 , 6 , 1 , 2 ) the lower bound: 313874
For ( 3 , 6 , 2 , 2 ) the lower bound: 195052
For ( 3 , 7 , 1 , 2 ) the lower bound: 30368331
For ( 3 , 7 , 2 , 2 ) the lower bound: 47674809
For ( 3 , 7 , 3 , 2 ) the lower bound: 6715800
For ( 3 , 8 , 1 , 2 ) the lower bound: 4348391200
For ( 3 , 8 , 2 , 2 ) the lower bound: 18534376992
For ( 3 , 8 , 3 , 2 ) the lower bound: 5440604240
For ( 3 , 8 , 4 , 2 ) the lower bound: 133840036
For ( 3 , 9 , 1 , 2 ) the lower bound: 962731541510
For ( 3 , 9 , 2 , 2 ) the lower bound: 9235671700006
For ( 3 , 9 , 3 , 2 ) the lower bound: 10627879161720
For ( 3 , 9 , 4 , 2 ) the lower bound: 56802358580
For ( 3 , 9 , 5 , 2 ) the lower bound: 1193920
```

```
rng =  10 , d =  2 , p =  5
For ( 5 , 3 , 1 , 2 ) the lower bound: 39
For ( 5 , 4 , 1 , 2 ) the lower bound: 2228
For ( 5 , 4 , 2 , 2 ) the lower bound: 224
For ( 5 , 5 , 1 , 2 ) the lower bound: 442853
For ( 5 , 5 , 2 , 2 ) the lower bound: 16927
For ( 5 , 5 , 3 , 2 ) the lower bound: 1
For ( 5 , 6 , 1 , 2 ) the lower bound: 118907250
For ( 5 , 6 , 2 , 2 ) the lower bound: 239688900
For ( 5 , 7 , 1 , 2 ) the lower bound: 59749225683
For ( 5 , 7 , 2 , 2 ) the lower bound: 354041153067
For ( 5 , 7 , 3 , 2 ) the lower bound: 134258656896
For ( 5 , 8 , 1 , 2 ) the lower bound: 60040603657936
For ( 5 , 8 , 2 , 2 ) the lower bound: 933524668543792
For ( 5 , 8 , 3 , 2 ) the lower bound: 1037515951813328
For ( 5 , 8 , 4 , 2 ) the lower bound: 74790836493164
For ( 5 , 9 , 1 , 2 ) the lower bound: 132084261969903470
For ( 5 , 9 , 2 , 2 ) the lower bound: 4795758878860333866
For ( 5 , 9 , 3 , 2 ) the lower bound: 14258034170791913112
For ( 5 , 9 , 4 , 2 ) the lower bound: 3025430261988613908
For ( 5 , 9 , 5 , 2 ) the lower bound: 41396246606782704
```

```
rng =  10 , d =  3 , p =  2
```

```
rng =  10 , d =  3 , p =  3
For ( 3 , 9 , 1 , 3 ) the lower bound: 1003811200
```

```
rng =  10 , d =  3 , p =  5
For ( 5 , 7 , 1 , 3 ) the lower bound: 5687456775
For ( 5 , 8 , 1 , 3 ) the lower bound: 6902181807192
For ( 5 , 8 , 2 , 3 ) the lower bound: 2560365976896
For ( 5 , 9 , 1 , 3 ) the lower bound: 59821776967508178
```