Matrix Codes and Subspace Designs

Chaoyi Lu

Supervisor: Eimear Byrne

08/03/2018

Abstract

We are interested in the existence of Assmus-Mattson rank-metric codes over finite fields, as such codes yield constructions of subspace designs. An Assmus-Mattson rank metric code is one whose dual code has few non-zero ranks. A first step towards tackling this problem is to check the existence of codes that hold designs known to be realizable. Given the parameters of a known subspace design, we compute the lists of rank weight distributions of possible codes associated with the design and apply, among other constraints, the rank metric MacWilliams identities to check existence.

1. Introduction

The Assmus-Mattson Theorem [13] is one of the well known results in the codes and designs theory. Based on this theorem, we have made several constructions of t-designs [9, 10, 13, 14, 15]. This theorem has also been the subject of some generalizations [1, 11, 18, 21, 25]. The Assmus-Mattson theorem states the conditions of a rank metric linear code holding a specific block as a subspace design. In particular, it shows that the supports of minimum rank metric weight codewords forms the blocks of a design if the number of non-zero weights of dual code is small.

The first time of the notion of a t-design appearing in the literature is 1970s [19] and some contructions of such designs can be found in the past papers [17, 23, 24]. A t-(n, d, λ) design (called blocks) over F_q is a collection of d-dimensional subspaces of F_q^n where every t-dimensional subspace of F_q^n is contained in the same number λ of elements of D [2]. During the last decade, there was a resurgence of interest in such designs, due in part to the fact that some subspace designs are optimal as constant weight subspace codes [2]. It has been shown that the subspace codes have the applications to error correction in network coding [12]. And [16] shows that non-trivial exist over any finite field and for every value of t, as long as the parameters n, d, λ are sufficiently large. The existence of such objects for various parameter sets are shown by several papers and most of them rely on assumptions of their automorphism groups [4, 5, 6, 7, 8].

In our project, we need to check the existence of possible Assmus-mattson codes which hold a specific design based on a table we take from [7] according to Assmus-Mattson Theorem and the main tool we use to detemine the non-existence of such codes is MacWilliams identities [12]. Due to the large amounts of works needing to be done, we are using python and school cluster to help our checking. In this report, we will first introduce some basic definitions, theorems, lemmas and examples we focus on in our project in section 2. Then we will put some sample codes for several specific examples and my main python codes along with the tables we focus on for checking the existence in section 3. Section 4 will show the results we get from our codes and some conclusions as well as some comments about these results up to now. We will also talk about what we will do in the future in this section.

2. Preliminaries

Finite Fields

In this section, we introduce the necessary background on finite fields. We refer the reader to [20] for further detail.

Definition 1. A **group** is a set G together with a binary operation on G such that the following three properties hold:

1. $*$ is **associative**; that is, for any $a, b, c \in G$, $a * (b * c) = (a * b) * c$.
2. There is an **identity (or unity)** element e in G such that for all $a \in G$, $a * e = e * a = a$.
3. For each $a \in G$, there exists an **inverse element** $a^{-1} \in G$ such that $a * a^{-1} = a^{-1} * a = e$.

If the group also satisfies:

- For all $a, b \in G$, $a * b = b * a$, then the group is called **abelian** (or commutative).

Definition 2. A **ring** $(R, +, \cdot)$ is a set R, together with two binary operations, denoted by $+$ and $\cdot$ such that:

1. R is an **abelian group** with respect to $+$.
2. $\cdot$ is **associative**,that is $a \cdot (b \cdot c) = (a \cdot b) \cdot c$, for all $a, b, c \in R$
3. The **distributive laws** hold; that is, for all a, b, c E R we have $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$

Definition 3.

1. A ring is called a ring with **identity** if the ring has a multiplicative identity, that is, if there is an element e such that $ae = ea = a$ for all $a \in R$.
2. A ring is called **commutative** if is commutative.
3. A ring is called an **integral domain** if it is a commutative ring with identity $e \neq 0$ in which $ab = 0$ implies $a = 0$ or $b = 0$.
4. A ring is called a **division ring (or skew field)** if the nonzero elements of R form a group under $\cdot$.
5. A **commutative division ring** is called a **field**.

Example 4. [3]

1. Q, R, C are fields.
2. The integers modulo p for some prime number p can be constructed as a finite field. i.e. we let $p = 7$ and then $Z_7 = \{0, 1, 2, 3, 4, 5, 6\}$ is a finite field. Also this field can be denoted as Z_7, $Z/7Z$, $GF(7)$ or F_7.
   It is known that every finite field has the order of a power of a prime number, and there exists a unique (up to isomorphism) finite field for each prime power. If q is a power of a prime number, then we can denote the finite field of order q as $GF(q)$, F_q. If $q = p^m$ where p is prime, we can also write $GF(p^m)$ or F_{p^m} and in this case we say that F_q has *characteristic* p. i.e. $px = x + x + x + \cdots + x$ for all $x \in F_q$.
3. Let $f(x) = x^3 + x + 1 \in F_2[x]$ and let α be a root of f in an extension of F_2(i.e. in a field containing F_2 as a subfield). In other words, α satisfies the relation $\alpha^3 = \alpha + 1$. Consider the following table:

0	0
1	1
α	α
α^2	α^2
α^3	$\alpha+1$
α^4	$\alpha^2+\alpha$
α^5	$\alpha^2+\alpha+1$
α^6	α^2+1
α^7	1

We the set $\{0,\alpha,\alpha^2,\ldots,\alpha^7=1\}$ is called $F_8\,(or\ GF(8))$, the finite field of order 8. It can be checked that the set above satisfies the field axioms, so it is an abelian additive group and forms a commutative multiplicative group when the zero element is removed. i.e. the multiplicative group $F_8^\times = F_8\backslash\{0\}$.
The column on the left represents the field elements in multiplicative notation. The column on the right represents the elements in additive form, using the relation provided by $f(\alpha)=0$.

4. Let $f(x)=x^2+x+2\in F_3[x]$ and α is a root of $f(x)$. Then the table of elements of $F_9=F_3(\alpha)$ is

0	0
1	1
α	α
α^2	$2\alpha+1$
α^3	$2\alpha+2$
α^4	2
α^5	2α
α^6	$\alpha+2$
α^7	$\alpha+1$
α^8	1

5. Let $f(x)=x^4+x+1\in F_2[x]$ and α is the root of $f(x)$. The table of elements of $F_{16}=F_2(\alpha)$ is

0	0
1	1
α	α
α^2	α^2
α^3	α^3
α^4	$\alpha+1$
α^5	$\alpha^2+\alpha$
α^6	$\alpha^3+\alpha^2$
α^7	$\alpha^3+\alpha+1$
α^8	α^2+1
α^9	$\alpha^3+\alpha$
α^{10}	$\alpha^2+\alpha+1$
α^{11}	$\alpha^3+\alpha^2+\alpha$
α^{12}	$\alpha^3+\alpha^2+\alpha+1$
α^{13}	$\alpha^3+\alpha^2+1$
α^{14}	α^3+1
α^{15}	1

Lemma 5. Let F be a finite field containing a subfield K with q elements. Then F has q^m elements, where $m=[F:K]$.

Theorem 6. Let F be a finite field. Then F has p^n elements, where the prime p is the characteristic of F and n is the degree of F over its prime subfield.

Lemma 7. If F is a finite field with q elements, then every $a\in F$ satisfies $a^q=a$.

Lemma 8. If F is a finite field with q elements and K is a subfield of F, then the polynomial $x^q - x$ in $K[x]$ factors in $F[x]$ as

$$x^q - x = \prod_{a \in F}(x - a)$$

Theorem 9. (Existence and Uniqueness of Finite Fields).
For every prime p and every positive integer n there exists a finite field with p^n elements. Any finite field with $q = p^n$ elements is isomorphic to the splitting field of $x^q - x$ over F_p.

Lemma 10. The finite field F_q^m is an F_q vector space of dimension m. If α is a root of an irreducible polynomial of degree m with coefficients in F_q, then the set $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ is a basis of F_q^m over F_q.

Rank metric codes

Generally speaking, a linear code is just a subspace of a vector space where the ambient vector space is equipped with a distance function. Classically, these are subspaces of F_q^n, endowed with the Hamming metric, but in recent years, there has been considerable interest in codes that are subspaces of matrices, for the rank metric. The reader is referred to [2] for further detail on this topic.

Definition 11. A **linear block code** of length n over F_{q^m} is a subspace of $F_{q^m}^n$. If C has dimension k over F_{q^m}, we say that C is an $[n, k]$ code. If furthermore C has minimum distance d we say that C is an $[n, k, d]$ code, or an F_{q^m} - $[n, k, d]$ code.
An F_q-$[n \times m, k]$ **linear matrix code** is a k-dimensional subspace of $F_q^{n \times m}$, we also say C is an F_q-$[n \times m, k, d]$ code if it has *minimum rank distance* d (Definition 14).

In this project, we focus on $[n \times m, k]$ matrix codes over F_q that are contructed from $[n, k]$ block codes over F_{q^m}. Let Γ be an arbitrary basis of F_{q^m} as a vector space over F_q.
Let $\Gamma(a) := [a_1, \ldots, a_m]$ be the representation of $a \in F_{q^m}$ as a vector of length m with coefficients in F_q, withrt the basis Γ.
Given any vector $v = [v_1, \ldots, v_n]^t \in F_{q^m}^n$, the matrix represetation of v is the $n \times m$ matrix:

$$\Gamma(v) := \begin{bmatrix} \Gamma(v_1) \\ \Gamma(v_2) \\ \vdots \\ \Gamma(v_n) \end{bmatrix} = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & v_{22} & \cdots & v_{2m} \\ \vdots & \vdots & \vdots & \vdots \\ v_{n1} & v_{n2} & \cdots & v_{nm} \end{bmatrix}.$$

Given a block code C, the corresponding matrix code of C with respect to Γ is defined as $\Gamma(C) := \{\Gamma(c) : c \in C\}$.

Example 12.

1. Example 4.3 continuted. For the finite field F_8 as showed above, this field has characteristic 2, since $1 + 1 = 0$ in this set. It also contains F_2 as a subfield. We could write $F_8 = F_2(\alpha)$ since it is obtained from F_2 by adjoining the root α to the field F_2. This means that we augment the set F_2 by α, and then perform all possible multiplications and additions until the set stabilizes. So
$$F_8 = F_2[\alpha] = \{a_0 + a_1\alpha + a_2\alpha^2 : a_i \in F_2\},$$
which is exactly the right column of the table in example 4.3.
In fact F_8 is an F_2 vector space, with basis $\{1, \alpha, \alpha^2\}$.
2. Example 4.4 continuted. For the finite field F_9, this field has characteristic 3 and we can write:
$$F_9 = F_3[\alpha] = \{a_0 + a_1\alpha : a_i \in F_3\},$$
and F_9 is an F_3 vector space with basis $\{1, \alpha\}$.
3. For example 4.5, the finite field F_{16} has characteristic 2, then we can write:
$$F_{16} = F_2[\alpha] = \{a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 : a_i \in F_2\},$$

and F_{16} is an F_2 vector space with basis $\{1, \alpha, \alpha^2, \alpha^3\}$.

4. Suppose that we have a codeword $[\alpha, \alpha^3, \alpha]^t \in F_{2^3}^3$ with $\Gamma = \{1, \alpha, \alpha^2\}$, then
$$\Gamma(\alpha) = \Gamma(0 \cdot 1 + 1 \cdot \alpha + 0 \cdot \alpha^2) = [0,1,0],$$
$$\Gamma(\alpha^3) = \Gamma(1 \cdot 1 + 1 \cdot \alpha + 0 \cdot \alpha^2) = [1,1,0],$$
Thus $\Gamma([\alpha, \alpha^3, \alpha]^t) = \Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha \end{bmatrix} = \begin{bmatrix} \Gamma(\alpha) \\ \Gamma(\alpha^3) \\ \Gamma(\alpha) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$
5. If the codeword is $[\alpha, \alpha^3, \alpha^6]^t \in F_{2^3}^3$ with the same basis as above, then
$$\Gamma(\alpha) = \Gamma(0 \cdot 1 + 1 \cdot \alpha + 0 \cdot \alpha^2) = [0,1,0],$$
$$\Gamma(\alpha^3) = \Gamma(1 \cdot 1 + 1 \cdot \alpha + 0 \cdot \alpha^2) = [1,1,0],$$
$$\Gamma(\alpha^6) = \Gamma(1 \cdot 1 + 0 \cdot \alpha + 1 \cdot \alpha^2) = [1,0,1],$$
Thus $\Gamma([\alpha, \alpha^3, \alpha^6]^t) = \Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha^6 \end{bmatrix} = \begin{bmatrix} \Gamma(\alpha) \\ \Gamma(\alpha^3) \\ \Gamma(\alpha^6) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$

Definition 13. A **distance function** on F_q^n is a metric on F_q^n. Such a function is a map
$$d : F_q^n \times F_q^n \to R$$
such that

1. $d(a,b) \geq 0$ for all $a, b \in F_q^n$, with equality if and only if $a = b$,
2. $d(a,b) = d(b,a)$ for all $a, b \in F_q^n$ (symmetry),
3. $d(a,b) = d(b,c) \geq d(a,c)$ for all $a, b, c \in F_q^n$ (the triangular inequality).

We then define the **weight** of a word in F_q^n to be its distance to the zero in F_q^n, i.e. $w(x) = d(x, 0)$.

Definition 14.
The **Rank distance** between a pair of matrices $X, Y \in F_q^{n \times m}$ is defined to be the rank of their difference:
$$d_{rk}(X,Y) := rk(X - Y).$$

Example 15.
We take the result of example 12.4 and 12.5 as our example here. We have
$$\Gamma([\alpha, \alpha^3, \alpha]^t) = \Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha \end{bmatrix} = \begin{bmatrix} \Gamma(\alpha) \\ \Gamma(\alpha^3) \\ \Gamma(\alpha) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$
and
$$\Gamma([\alpha, \alpha^3, \alpha^6]^t) = \Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha^6 \end{bmatrix} = \begin{bmatrix} \Gamma(\alpha) \\ \Gamma(\alpha^3) \\ \Gamma(\alpha^6) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$
so the rank distance between these two codewords $[\alpha, \alpha^3, \alpha]^t$ and $[\alpha, \alpha^3, \alpha^6]^t$ is
$$d_{rk}([\alpha, \alpha^3, \alpha]^t, [\alpha, \alpha^3, \alpha^6]^t)$$
$$= d_{rk}(\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha \end{bmatrix}, \begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha^6 \end{bmatrix}) = d_{rk}(\Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha \end{bmatrix}, \Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha^6 \end{bmatrix})$$
$$= rk(\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}) = rk(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}) = 1$$
where $\Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ has F_2-rank 2 and $\Gamma\begin{bmatrix} \alpha \\ \alpha^3 \\ \alpha^6 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ has rank 3.

Definition 16. A (linear) **matrix code** C is an F_q-subspace of $F^{n \times m}$. The **minimum distance** of C is
$$d_{rk}(C) := min\{d_{rk}(X,Y) : X, Y \in C, X \neq Y\} = min\{rk(X) : X \in C, X \neq 0\}$$

Definition 17. Let C be an $F_q - [n \times m, k, d]$ matrix code. We define the number

$$W_i(C) := |\{X \in C : rk(X) = i\}|,$$
which is the number of codewords of C of rank i. The (**rank**) **weight distribution** of C is the sequence ($W_i(C) : 1 \leq i \leq n$). We say that an integer i is a **non-zero weight** of C if $W_i(C) \neq 0$.

Definition 18. Let C be an F_q − $[n, k]$ code. A matrix G is called a **generator matrix** of C if G is a $k \times n$ matrix over F_q whose rows form a basis of C.
Therefore, $G \in F_q^n$ is a generator matrix for C if and only if
$$C = \{xG : x \in F_q^k\}$$
and G has rank k.

Defnition 19. Let C be an F_q - $[n, k]$ code. The dual code of C is defined by:
$$C^{\perp} = \{v \in F_q^n : c \cdot v = 0 \ \ \forall c \in C\}$$

Lemma 20. Let C be an F_q - $[n, k]$ code. Then $C^{\perp}$ is an F_q - $[n, n - k]$ code.

Definition 21. Let C be an F_q - $[n, k]$ code. An $(n - k) \times n$ matrix H over F_q is called a **parity check matrix (PCM)** for C if $Hc^{\perp} = 0$ for all $c \in C$.

Lemma 22. Let C be an F_q - $[n, k]$ code. Then H is a PCM for C if and only if H is a generator matrix for $C^{\perp}$.

Example 23.

1. The binary code $C = \{000, 110, 101, 011\}$ is a F_2 - $[3, 2, 2]$ code with generator matrix
$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$
2. Let $C = \{000, 110, 101, 011\} \subset\subset F_2^3$. Then $C^{\perp} = \{000, 111\}$. So C has parity check matrix $H = [111]$.
3. Let $s = 2$ and let $m = 2s$. Let $\{1, \alpha, \alpha^2, \alpha^3\}$ be a basis of $F_{2^{(m=4)}} = F_{16}$ over F_2. Let C be the F_{2^4}-$[4, 2, 2]$ rank metric code with parity check matrix
$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 \\ 1^{2^2} & (\alpha)^{2^2} & (\alpha^2)^{2^2} & (\alpha^3)^{2^2} \end{bmatrix} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 \\ 1 & \alpha^4 & \alpha^8 & \alpha^{12} \end{bmatrix}.$$
It can be checked that
$$W_0(C^{\perp}) = 1, \quad W_2(C^{\perp}) = \frac{(2^{2 \cdot 2} - 1)^2}{2^2 - 1} = 75, \quad W_4(C^{\perp}) = 2^{4 \cdot 2} - \frac{(2^{2 \cdot 2} - 1)^2}{2^2 - 1} - 1 = 180,$$
and that $W_i(C^{\perp}) = 0$ otherwise. So the weight distribution of $C^{\perp}$ is:
$$[1, 0, 75, 0, 180]$$

We refer readers to [3] for further detail relevent to Definition 18,19,20,21,22.

The Rank-Metric Assmus-Mattson Theorem

In this part, we will talk about the main theorem we base on in our project, but before that, we need to look at some necessary definitions and remarks first. More detail can be found in reference [2].

Defnition 24. Let $n \geq r \geq t$ be positive integers and let D be a collection of r-dimensional subspaces of F_q^n. We say that D forms (the blocks of) a t-**design** over F_q if every t-dimensional subspace of F_q^n is contained in the same number λ of elements of D. In this case we say that D is a t-(n, r, λ) design over F_q.
Designs over F_q are also known as **subspace designs** and as **designs over finite fields**. A t-$(n, r, 1)_q$ subspace design is called a q-Steiner system.

Example 25. Let $q = 2$ and for the designs $2\text{-}(n,3,7)_2$ where $n \equiv \pm 1 \bmod 6$, $n \geq 7$ with ambiant space F_{2^n}. The **"blocks"** are the F_2-subspaces of F_2^n of dimension 3 of the form:

$$\langle x, y, \frac{xy}{x+y} \rangle = \{a_x x + b_y y + c_{xy}\frac{xy}{x+y} : a_x, b_y, c_{xy} \in F_2\} \quad where\ x, y \in F_{2^n}^{\times}, x \neq y$$

Then any 2-dimensional space $\langle u, v \rangle := \{a_u u + a_v v : a_u, a_v \in F_2\}\ where\ u, v \in F_{2^n}^{\times}$ is contained in exactly 7 blocks of the form $\langle x, y, \frac{xy}{x+y} \rangle$.

This example is actually a theorem proved by S.Thomas 1987. It is one of the few constructions we know for subspace designs. Most of other constructions are found by computer but this one is not. We will refer readers to [24] for more detail.

Definition 26. Given a vector $v = [v_1, \ldots, v_n] \in F_{q^m}$, $\sigma(v)$ is the **column space** of the $n \times m$ matrix.

Definition 27. We say that a subspace $U < F_q^n$ is a **support** of a code C if $\sigma(x) = U$ for some codeword $x \in C$.

Remarks 28. A way to construct a design from an AM code here
We construct a design from an AM F_{q^m}-$[n,k,d]$ code as follows:

$$D(C) := \{\sigma(\Gamma(c)) : c \in C, rk(\Gamma(c)) = d\}.$$

If C is an AM code, then its dual code also has a subspace design held by the supports of its minimum weight codewords.

$$D(C^{\perp}) := \{\sigma(\Gamma(c)) : c \in C^{\perp}, rk(\Gamma(c)) = d^{\perp}\}.$$

where Γ is an arbitrary basis of F_{q^m}. We call these two designs as d-supports of $\Gamma(C)$ and $d^{\perp}$-supprts of $\Gamma(C^{\perp})$. The basis of this is the following theorem.

Theorem 29. The Rank-Metric Assmus-Mattson Theorem
Let C be an F_{q^m}-$[n,k,d]$ code. Let $1 \leq t < d$ be an integer, and assume that

$$|\{1 \leq i \leq n - t\ :\ W_i(C^{\perp}) \neq 0\}| \leq d - t.$$

Let $d^{\perp}$ be the minimum distance of $C^{\perp}$. Then for every F_q-basis of F_{q^m} the d-supports of $\Gamma(C)$ and the $d^{\perp}$-supports of $\Gamma(C^{\perp})$ form the blocks of a t-design over F_q.

Definition 30. Assmus-Mattson (AM) rank-metric code
Let t be a positive integer. We're interested in the existence of a code C with the following properties.

1. C is an F_{q^m}-$[n,k,d]$ code, $d \geq t$ is the **minimum rank distance** of C.
2. $|\{i \in \{1, \ldots, n-t\} : W_i(C^{\perp}) \neq 0\}| \leq d - t$

We will call a code that satisfies the above an **Assmus-Mattson (AM) rank-metric code**.
If we can find such a code, then we can construct a $t\text{-}(n,d,\lambda)_q$ **subspace design**.

Example 31. Example 23.3 gives an F_{2^4} - $[4,2,2]$ rank metric code and the weight distribution of $C^{\perp}$ is

$$[1, 0, 75, 0, 180]$$

We can see that $C^{\perp}$ has F_{16}- ranks $\{0,2,4\}$. So if we take $t = 1$, the desired **Assmus-mattson property** is hold. i.e. $t = 1 \leq d = 2$ and $|\{i \in \{1, 2, n-t = 4-1 = 3\} : W_i(C^{\perp}) \neq 0\}| = 1 \leq d - t = 2 - 1 = 1$. Thus this code is a Assmus-mattson rank-metric code.

Example 32. The F_{2^4}-$[4,2,2]$ rank metric code showed in example 31 has the desired Assmus-mattson property with $t = 1$. The supports of the codewords of C of rank 2 form a 1-design over F_q and the words of rank $s = 2$ in $C^{\perp}$ form a $1\text{-}(m = 4, s = 2, 1)$ design.

Example 33. Example 23.3, 31 and 32 above is actually a special case where $F_{q^m} = F_{2^4}$. Generally, let s be a positive integer and let $m = 2s$. Let $\{\alpha_1, \ldots \alpha_m\}$ be a basis of F_{q^m} over F_q. Let C be the F_{q^m}-$[m, m-2, 2]$ rank metric code with parity check matrix

$$H = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \alpha_1^{q^s} & \alpha_2^{q^s} & \cdots & \alpha_m^{q^s} \end{bmatrix}.$$

then

$$W_0(C^\perp) = 1, \quad W_s(C^\perp) = \frac{(q^{2s}-1)^2}{q^s-1}, \quad W_{2s}(C^\perp) = q^{4s} - \frac{(q^{2s}-1)^2}{q^s-1} - 1,$$

and that $W_i(C^\perp) = 0$ otherwise. In particular, $C^\perp$ has F_q-ranks $\{0, s, 2s\}$ and so C has the desired Assmus-mattson property with $t = 1$. That is, $C^\perp$ has exactly one weight s in $\{1, \dots, 2s-1\}$. The supports of the codewords of C of rank 2 form a 1-design over F_q and the words of rank s in $C^\perp$ form a 1-$(m, s, 1)$ subspace design.

Definition 34. The **q-binomial** or **Gaussian coefficient** counts the number of r-dimensional subspaces of an n-dimensional subspace over F_q and is given by:

$$\begin{bmatrix} n \\ r \end{bmatrix}_q := \prod_{i=0}^{r-1} \frac{q^n - q^i}{q^r - q^i}.$$

Example 35.

1. $\begin{bmatrix} 3 \\ 2 \end{bmatrix}_2 := \prod_{i=0}^{2-1} \frac{2^3-2^i}{2^2-2^i} = \frac{2^3-2^0}{2^2-2^0} \cdot \frac{2^3-2}{2^2-2} = \frac{7}{3} \cdot \frac{6}{2} = 7$ i.e.the number of 2-dimensional subspaces of an 3-dimensional subspace over F_2 is 7.
2. $\begin{bmatrix} 5 \\ 3 \end{bmatrix}_2 := \prod_{i=0}^{3-1} \frac{2^5-2^i}{2^3-2^i} = \frac{2^5-2^0}{2^3-2^0} \cdot \frac{2^5-2^1}{2^3-2^1} \cdot \frac{2^5-2^2}{2^3-2^2} = \frac{31}{7} \cdot \frac{30}{6} \cdot \frac{28}{4} = 155$ which means there are 155 3-dimensional subspaces of an 5-dimensional subspace over F_2.

Tools

Here are some useful tools we will use in the process of determining the non-existence of possible codes. Some of the tools can help us rule out millions of cases such that our python codes can run much faster than before. More detail can also be found in [2].

Definition 36. The **rank-metric Singleton bound** says that for any F_{q^m}-$[n,k,d]$ rank metric code we have:

$$k \le \begin{cases} n - d + 1 & \text{if } m \ge n, \\ \frac{n}{m}(m - d + 1) & \text{if } m \le n. \end{cases}$$

Definition 37. Codes that meet the rank-metric Singleton bound are called **maximum rank distance (MRD)** codes.

Lemma 38. Let C be an F_q^m-$[n,k,d]$ code, and let Γ be any F_q-basis of F_q^m. Assume $m \ge n$. Then the following are equivalent.

1. C is MRD,
2. The words of rank d in $\Gamma(C)$ hold a trivial design over F_q.

Moreover, for $m < n$, C is MRD $\Rightarrow$ The words of rank d in $\Gamma(C)$ hold a trivial design over F_q.

Example 39. Let $n \le m$ and let $\{\alpha_1, \dots, \alpha_n\} \subset F_{q^m}$ be a set of n F_q-linearly independent elements. Let C be the F_{q^m}-$[n, k, n-k+1]$ rank metric code with generator matrix

$$G = \begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^q & \alpha_2^q & \cdots & \alpha_n^q \\ \alpha_1^{q^2} & \alpha_2^{q^2} & \cdots & \alpha_n^{q^2} \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{q^{k-1}} & \alpha_2^{q^{k-1}} & \cdots & \alpha_n^{q^{k-1}} \end{bmatrix}.$$

Then C is called a **Delsarte-Gabidulin code**[12] and is optimal, in fact MRD, since it satisfies the **rank-metric Singleton bound**. This is a very famous code and its weight distribution is known and uniquely determined by its parameters q, n, m, k. In particular, writing $d = n - k + 1$, its weight distribution has the form:

$$[1, 0, \cdots, \begin{bmatrix} n \\ d \end{bmatrix}_q (q^m - 1), W_{d+1}(C), \ldots, W_n(C)].$$

Its dual code $C^\perp$ is also MRD, with parameters $[n, n-k, k+1]$ and has weight distribution of the form:

$$[1, 0, \cdots, \begin{bmatrix} n \\ d^\perp \end{bmatrix}_q (q^m - 1), W_{d^\perp+1}(C^\perp), \ldots, W_n(C^\perp)],$$

where $d^\perp = k + 1$. $C^\perp$ has $n - t - d^\perp + 1 = n - t - k$ non-zero weights in $\{1, \ldots, n-t\}$, which is less than $d - t = n - k + 1 - t$ and so C is an Assmus-Mattson code for any $1 \le t \le n - k + 1$. Similarly, so is $C^\perp$ an AM-code.

Example 33 gives 1-designs and we would like to find designs for $t \ge 2$. Example 39 gives trivel designs, since every d- dimensional space is a support of the code C in this case which we are not interested in. At the moment, example 33 and 39 are pretty much the only examples of AM codes known. It would be great to find more of them.

Remarks 40.
Part of the difficulty in finding new designs by Rank-Metric Assmus-Mattson Theorem is that very few classes of rank-metric codes are known, and all the major classes are MRD codes which don't return anything useful (Lemma 38 shows that MRD codes hold trivial designs and we are not intereted in this case). And we want to find more families of rank-metric codes.

One step towards solving the problem would be to look at the parameters of realizable and possible subspace designs and see whether there exists some code which would induce these designs.

Note that if D is a t-$(n, d, \lambda)_q$ design, then by double-counting the size of the set

$$\{(T, B) : T \le B, \dim T = t\},$$

we see that $\begin{bmatrix} n \\ t \end{bmatrix}_q \lambda = \begin{bmatrix} d \\ t \end{bmatrix}_q |D|$, so $|D| = \begin{bmatrix} n \\ t \end{bmatrix}_q \begin{bmatrix} d \\ t \end{bmatrix}_q^{-1} \lambda$. So if an AM code C holds this design in its minimum weight codewords, it must have

$$W_d(C) = (q^m - 1)|D| = (q^m - 1)\begin{bmatrix} n \\ t \end{bmatrix}_q \begin{bmatrix} d \\ t \end{bmatrix}_q^{-1} \lambda.$$

In other words, by the above property, we can contruct the weight distribution of an possible AM code C and then we can compute the weight distribution of the dual code $C^\perp$ by the following theorem.

Theorem 41. The MacWilliams identities [22, Theorem 31]
Let C be an F_{q^m}-$[n, k, d]$ code. Then for each $\ell \in \{0, \ldots, n\}$ we have:

$$\sum_{i=0}^{n-\ell} W_i(C^\perp) \begin{bmatrix} n-i \\ \ell \end{bmatrix}_q = q^{m(n-k-\ell)} \sum_{i=0}^{\ell} W_i(C) \begin{bmatrix} n-i \\ \ell - i \end{bmatrix}_q.$$

In fact, letting $W_i^\perp = W_i(C^\perp)$ and $W_i = W_i(C)$ we have:

$$\begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 1 \\ \begin{bmatrix} n \\ 1 \end{bmatrix}_q & \begin{bmatrix} n-1 \\ 1 \end{bmatrix}_q & \cdots & \begin{bmatrix} 2 \\ 1 \end{bmatrix}_q & 1 & 0 \\ \begin{bmatrix} n \\ 2 \end{bmatrix}_q & \begin{bmatrix} n-1 \\ 2 \end{bmatrix}_q & \cdots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ W_1^\perp \\ W_2^\perp \\ \vdots \\ W_n^\perp \end{bmatrix} = \begin{bmatrix} q^{m(n-k)} & 0 & 0 & 0 & \cdots & 0 \\ q^{m(n-k-1)} \begin{bmatrix} n \\ 1 \end{bmatrix}_q & q^{m(n-k-1)} & 0 & 0 & \cdots & 0 \\ q^{m(n-k-2)} \begin{bmatrix} n \\ 2 \end{bmatrix}_q & q^{m(n-k-2)} \begin{bmatrix} n-1 \\ 1 \end{bmatrix}_q & q^{m(n-k-2)} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ q^{-mk} & q^{-mk} & q^{-mk} & q^{-mk} & q^{-mk} & q^{-mk} \end{bmatrix} \begin{bmatrix} 1 \\ W_1 \\ W_2 \\ \vdots \\ W_n \end{bmatrix}$$

In particular, we can compute the $W_i(C^\perp)$ from the $W_i(C)$ via:

$$\begin{bmatrix} 1 \\ W_1^\perp \\ W_2^\perp \\ \vdots \\ W_n^\perp \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 & 1 \\ \begin{bmatrix} n \\ 1 \end{bmatrix}_q & \begin{bmatrix} n-1 \\ 1 \end{bmatrix}_q & \cdots & \begin{bmatrix} 2 \\ 1 \end{bmatrix}_q & 1 & 0 \\ \begin{bmatrix} n \\ 2 \end{bmatrix}_q & \begin{bmatrix} n-1 \\ 2 \end{bmatrix}_q & \cdots & 1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} q^{m(n-k)} & 0 & 0 & 0 & \cdots & 0 \\ q^{m(n-k-1)} \begin{bmatrix} n \\ 1 \end{bmatrix}_q & q^{m(n-k-1)} & 0 & 0 & \cdots & 0 \\ q^{m(n-k-2)} \begin{bmatrix} n \\ 2 \end{bmatrix}_q & q^{m(n-k-2)} \begin{bmatrix} n-1 \\ 1 \end{bmatrix}_q & q^{m(n-k-2)} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ q^{-mk} & q^{-mk} & q^{-mk} & q^{-mk} & q^{-mk} & q^{-mk} \end{bmatrix} \begin{bmatrix} 1 \\ W_1 \\ W_2 \\ \vdots \\ W_n \end{bmatrix}$$

By checking several properties of the elements of the weight distribution of $C^\perp$, we can determine the non-existence of the possible AM codes we constructed. The following section will give more information about this method.

3. Examples of process and codes

We are going to apply our method to an example to show the process of what we are doing in our project. Moreover, by some analysis, we found that there will be a large amount of cases needing to be checked. And it is obvious that it's unwise to check and compute everything by hand, so we chose to use python to help our checking process.

Example 42. We take $2\text{-}(6,3,3)_2$ design as our example. We know that there exists a $2\text{-}(6,3,3)_2$ design. If the blocks of this design were held by a code C, it would have to be an $F_{2^m}\text{-}[6,k,3]$ code such that every 2-dimensional subspace of F_2^6 is contained in exactly $\lambda = 3$ of the 3-supports of C. If C is an AM code for $t = 2$, then the criterion of the theorem says $C^\perp$ must have at most one non-zero weight $i \in \{1,2,3,4\}$.

Since the rank of any $n \times m$ matrix is at most $\min\{m,n\}$, in order for C to have words of weight 3 we'd need $m \geq 3$. We'd also have

$$W_3(C) = (2^m - 1)(3) \begin{bmatrix} 6 \\ 2 \end{bmatrix}_2 \begin{bmatrix} 3 \\ 2 \end{bmatrix}_2^{-1} = (2^m - 1)(279).$$

Since C has 2^{mk} codewords, we'd need $2^{mk} \geq (2^m - 1)(279) + 1$. If $m = 3$, then $W_3(C) = 1953$ and in fact C must have weight distribution $[1,0,0,1953,0,0,0]$, which is impossible, since $1 + 1953 = 1954$ is not a power of 2. If $m = 4$, then $W_3(C) = 4185$ and C has weight distribution $[1,0,0,4185,W_4(C),0,0]$, so we see that $W_4(C) = 2^{4k} - 4185 - 1$. Then the possible weight distributions of C are: $[1,0,0,4185,61350,0,0], [1,0,0,4185,1044390,0,0]$.
The first case occurs when $k = 4$ and the 2nd when $k = 5$. If $k = 6$ then C is the whole space $F_{q^m}^6$, which is not interesting. If we plug these weight distribution values into the equations given by the Macwilliams identities, we get:

$$[1,-1395/128,5565/128,-585/64,3705/16,0,0], \quad [1,-30195/2048,139985/2048,-115785/1024,19065/256,0,0],$$

which do not correspond to any actual codes, so neither of theses cases is possible. We could then consider $m = 5,6,7,\ldots$ cases and check if there are any possible code parameters that might work. As m increasing, there are more possibilities for the weight distribution of C.

We first tried to use python code to simulate the process of example 42 and the codes showed in the next page.

In [4]:

```
# 2-(6,3,3)2 design for m = 4
import numpy as np
from numpy.linalg import inv
from fractions import Fraction

# define q-binomial
def getqb(n,r,q):
    a=1
    for i in range(r):
        a = a*((q**n-q**i)/(q**r-q**i))
    return a

# define the second matrix
def getmtx2(n,k,m,q):
    mtx2 = np.zeros((n+1,n+1)).astype(float)
    for l in range(n+1):
        for i in range(l+1):
            mtx2[l][i]=q**(m*(n-k-l))*getqb(n-i,l-i,q)
    return mtx2

# define the first matrix
def getmtx1(n,q):
    mtx1 = np.zeros((n+1,n+1)).astype(float)
    for l in range(n+1):
        for i in range(n-l+1):
            mtx1[l][i] = getqb(n-i,l,q)
    return mtx1

# get the inverse of the first matrix
def getinv(mtx):
    mtxinv = inv(mtx)
    return mtxinv

# define the process of getting the weight distribution of C perp
def getWCp(mtx1inv,mtx2,WC):
    WCp = np.matmul(mtx1inv,np.matmul(mtx2,WC))
    return WCp

# define the d-weight enumerator of C
def getWdC(n,d,t,q,l,m):
    WdC = (q**m-1)*getqb(n,t,q)*l/getqb(d,t,q)
    return WdC

# define the weight distribution of C
WC1 = np.zeros((7,1)).astype(float)
WC1[0][0]=1
WC1[3][0]=4185
WC1[4][0]=61350
print('The weight distribution of C when k = 4 is')
print(WC1)
WC2 = np.zeros((7,1)).astype(float)
WC2[0][0]=1
WC2[3][0]=4185
WC2[4][0]=1044390
print('The weight distribution of C when k = 5 is')
print(WC2)

# calculate the weight ditribution of C perp
mtx2_1=getmtx2(6,4,4,2)
```

```
mtx2_2=getmtx2(6,5,4,2)
mtx1 = getmtx1(6,2)
mtx1inv = getinv(mtx1)
print('The weight distribution of C perp when k = 4 is')
print(getWCp(mtx1inv,mtx2_1,WC1))
print('The weight distribution of C perp when k = 5 is')
print(getWCp(mtx1inv,mtx2_2,WC2))
```

```
The weight distribution of C when k = 4 is
[[  1.00000000e+00]
 [  0.00000000e+00]
 [  0.00000000e+00]
 [  4.18500000e+03]
 [  6.13500000e+04]
 [  0.00000000e+00]
 [  0.00000000e+00]]
The weight distribution of C when k = 5 is
[[  1.00000000e+00]
 [  0.00000000e+00]
 [  0.00000000e+00]
 [  4.18500000e+03]
 [  1.04439000e+06]
 [  0.00000000e+00]
 [  0.00000000e+00]]
The weight distribution of C perp when k = 4 is
[[  1.00000000e+00]
 [ -1.08984375e+01]
 [  4.34765625e+01]
 [ -9.14062500e+00]
 [  2.31562500e+02]
 [ -7.27595761e-12]
 [  3.63797881e-12]]
The weight distribution of C perp when k = 5 is
[[  1.00000000e+00]
 [ -1.47436523e+01]
 [  6.83422852e+01]
 [ -1.13071289e+02]
 [  7.44726562e+01]
 [ -7.27595761e-12]
 [  7.27595761e-12]]
```

Codes end here. (Everything printed below the codes are the outputs.)

The codes above is a sample code for $2\text{-}(6,3,3)_2$ design when $m = 4$. We call the $(n+1)\times(n+1)$ matrix in the left hand side of equation from theorem 41 as 'mtx1' and the $(n+1)\times(n+1)$ matrix on the right hand side is called 'mtx2'. The results these codes get is actually the same as example 42 where we calculate everything by magma (which is a progamme normally used in coding theory) if we transfer fraction to float and view the numbers with 10^{-12} as zero. The reason that the results show $7.27595761e-12$ here instead of 0 may be the property of the package we use when calculating the inverse of a matrix, but this won't be a problem in our codes since we have found a way which will be talked about later to deal with this problem.

Lemma 43. If C is an F_{q^m}-$[n,k,d]$ rank metric code and $X \in C$, then

1. $\beta X \in C$ for all $\beta \in F_{q^m}$,
2. $rk(\beta X) = rk(X)$ for all non-zero $\beta \in F_{q^m}$.

In particular $W_i(C)$ is a non-negative integer multiple of $q^m - 1$ for each $i \geq 1$.

Example 44. Example 42 comtinued. We know that m should be no less than 3 and we have

$$W_3(C) = (2^m - 1)(3)\begin{bmatrix}6\\2\end{bmatrix}_2\begin{bmatrix}3\\2\end{bmatrix}_2^{-1} = (2^m - 1)(279).$$

So for $m = d + 2$ i.e. $m = 3 + 2 = 5$, we have $W_3(C) = 8649$ and $W_4(C) + W_5(C) = 2^{5k} - 8649 - 1$ and we are not interested in the case $k = n$ since C will be the whole space $F_{2^5}^6$ in this case. So we have $k \leq 5$. Of course we also have the condition where $W_4(C) + W_5(C) = 2^{5k} - 8649 - 1 \geq 0$, thus $kmin$ should be 3. Then for cases $k = 3, k = 4, k = 5$, we can contruct the possible weight distribution for C. For example, when $k = 3$, the possible weight distributions of C are:

$$[1, 0, 0, 8649, (2^{15} - 8649 - 1 - x), x, 0]$$

and then get the weight distribution of $C^{\perp}$ by the equation given by **the Macwilliams identities** to check whether it is possible that this weight distribution correspond to any actual codes.

We set two simple properties to determine this:

1. The elements of the weight distribution of $C^{\perp}$ should be integers.
2. The elements of the weight distribution of $C^{\perp}$ should not be negative.

Though everything above is easy to check but we still have a lot of work to do since x can be any number between 0 and $(2^{5k} - 8649 - 1)$ and it will take more and more work as k and m increase. Thus, we are going to use **Lemma 43** to improve our mathod above.

Lemma 43 tells that $W_i(C)$ is a non-negative integer multiple of $q^m - 1$ for each $i \geq 1$ if C is an F_{q^m}-$[n, k, d]$ rank metric code and $X \in C$. In other words, we don't need to check every number in $[0, (2^{5k} - 8649 - 1)]$ and we just need to take the numbers which are multiples of $q^m - 1$ in this interval. i.e. the possible weight distribution of C can be modified as:

$$[1, 0, 0, 8649, (2^{15} - 8649 - 1 - x \cdot (q^m - 1)), x \cdot (q^m - 1), 0]$$

where $q = 2, m = 5$. And before this, we also need to check whether $(2^{5k} - 8649 - 1)$ is the multiple of $2^5 - 1$ but it is obvious that it is, because $(2^{5k} - 1)$ and $W_3(C)$ (example 42) are both multiple of $(2^m - 1)$. Thus we can take x to be numbers in range $[0, \frac{2^{5k}-8649-1}{2^5-1}]$ where $k = 3$ in this example.

Even though the improved method help us rule out a lot of cases but there are still a large amount of cases left. So asking computer to help our checking will be a nice choice. The following codes are the simulation of the process of example 44.

```
In [5]:
```

```python
n = 6
d = 3
t = 2
q = 2
l = 3
m = 5
W3C5 = getWdC(n,d,t,q,l,m)
print('W3(C) is ',W3C5,'when m = 5')
mtx1_5 = getmtx1(n,q)
mtx1_5inv = getinv(mtx1_5)
for k in range(3,5):
    if q**(m*k)>W3C5+1:
        if float((q**(m*k)-W3C5-1)/(q**m-1)).is_integer()==True:
            mtx2_5 = getmtx2(n,k,m,q)
            for x in range(int((q**(m*k)-W3C5-1)/(q**m-1))+1):
                WC5 = np.zeros((n+1,1)).astype(np.longfloat)
                WC5[0][0]=1
                WC5[3][0]=W3C5
                WC5[4][0]=q**(m*k)-W3C5-1-x*(q**m-1)
                WC5[5][0]=x*(q**m-1)
                flag = True
                WCp_5=getWCp(mtx1_5inv,mtx2_5,WC5)
                for i in range(1,n+1):
                    if round(WCp_5[i][0])==0:
                        continue
                    elif -10**(-5) < (round(WCp_5[i][0])-WCp_5[i][0]) < 10**(-5):
                        continue
                    else:
                        flag = False
                        break
                if flag == True:
                    print('Codes may exist for k =',k,'and x = ',x)
    elif q**(m*k)==W3C5+1:
        print('Here is the case q**(m*k)==W3C5+1 and k =',k)
print('Programme ends here')
```

```
W3(C) is  8649.0 when m = 5
Programme ends here
```

Codes end here

The python codes above will return sentences like "Codes may exist for k=, and x=" if there is one case satisfying the conditions we have set but It seems that these python codes failed to return what we would like to see, so we can conclude that no such code holding the $2\text{-}(6,3,3)_2$ design.

As what we talked about in remarks 40, we are also not interested in the k which meets and exceeds the **MRD** (Definition 37), so we can calculate the maximun value kmax by **rank-metric Singleton bound**. Here we have $m=5\leq n=6$, thus $k\leq\frac{n}{m}(m-d+1)=\frac{6}{5}(5-3+1)=\frac{18}{5}=3.6$ and $kmax$ should be 3. So we only need to check k from 1 to $kmax=3$ which makes the process much faster than before (we check every $k\leq 6$ before) i.e. for the design $2\text{-}(6,3,3)_2$, we only need to check $k=3$ case since the minimum of k is also 3.

Up to now, we have just talked about the cases $m=d, m=d+1$ and $m=d+2$. And for the cases such that m getting larger, the process will be similar with the $m=d+2$ case. We just need to add one more variable to the possible weight distribution of C each time as $m-d$ increasing by 1 until $m=n$. For example when $m=d+3$, what we need to do is just add one more variable which corresponds to the $W_{d+3}(C)$ in the weight distribution of the possible codes. i.e. for example, the $2\text{-}(6,3,3)_2$ design, when $m = d + 3 = 6$, the weight distribution of C will become:

$$[1, 0, 0, 17577 \quad , \quad (2^{mk} - 17577 - 1 - x \cdot (q^m - 1) - y \cdot (q^m - 1)) \quad , x \cdot (q^m - 1) \quad , y \cdot (q^m - 1)]$$

and then do the same process as before (checking weight distribution of dual codes) for every possible x, y, k.

Of course the python codes above are not beautiful since we need to modify several places everytime we move to a new design or a new λ. And as n and m increasing, the times of checking will also increase exponentially such that it may run for several years in my computer. So we chose to run our python codes on **school cluster** which is much faster than our own computers and can run jobs simultaneously. To make everything we modify as simple as possible eveytime when we want to move to a new design and prevent making codes in mess, we make some new python codes (which are then our main codes in this project) and put them in the next several pages:

In [ ]:

```python
import numpy as np
from numpy.linalg import inv
from fractions import Fraction

# define q-binomial
def getqb(n,r,q):
    a=1
    for i in range(r):
        a = a*((q**n-q**i)/(q**r-q**i))
    return a

# define the d-weight enumerator of C
def getWdC(n,d,t,q,l,m):
    WdC = (q**m-1)*getqb(n,t,q)*l/getqb(d,t,q)
    return WdC

# define the second matrix
def getmtx2(n,k,m,q):
    mtx2 = np.zeros((n+1,n+1)).astype(float)
    for l in range(n+1):
        for i in range(l+1):
            mtx2[l][i]=q**(m*(n-k-l))*getqb(n-i,l-i,q)
    return mtx2

# define the first matrix
def getmtx1(n,q):
    mtx1 = np.zeros((n+1,n+1)).astype(float)
    for l in range(n+1):
        for i in range(n-l+1):
            mtx1[l][i] = getqb(n-i,l,q)
    return mtx1

# get the inverse of the first matrix
def getinv(mtx):
    mtxinv = inv(mtx)
    return mtxinv

# define the process of getting the weight distribution of C perp
def getWCp(mtx1inv,mtx2,WC):
    WCp = np.dot(mtx1inv,np.dot(mtx2,WC))
    return WCp

# For convenience, I am going to define codes for getting maximun of k by MRD
def getkmax(n,d,m):
    if m >= n:
        km = int(np.floor(n-d+1))
        if float(km) == float(n-d+1):
            kmax = km-1
        else:
            kmax = km
    elif m < n:
        km = int(np.floor((n*(m-d+1))/m))
        if float(km) == float((n*(m-d+1))/m):
            kmax = km-1
        else:
            kmax = km
    return kmax

# Count the number of k which are available
```

```
def chk_k(t,n,d,l,q,m):
    WdC = getWdC(n,d,t,q,l,m)
    kmax = getkmax(n,d,m)
    count = 0
    for i in range(1,kmax+1):
        if q**(i*m)-WdC-1 >= 0:
            count += 1
    return count

# Since we have maximum k value (by MRD), then Lambda also have maximum value as below
def getlmax(t,n,d,q,m,kmax):
    lmax = int(np.floor((2**(m*kmax)-1)*getqb(d,t,q)/((q**m-1)*getqb(n,t,q))))
    return lmax

def ChkExs(t,n,d,l,q,m):
    mtx1 = getmtx1(n,q)
    mtx1inv = getinv(mtx1)
    if m < d:
        f = open("output.txt","a+")
        # Set case m<d as type 1 non-exsitence
        f.write("\n"+"No such code exists, type 1")
        f.close()
    else:
        # get the maximun k by MRD
        kmax = getkmax(n,d,m)

        # No variable Case
        if m == d:
            WdC = getWdC(n,d,t,q,l,m)
            # Check whether WdC is integer
            if -10**(-5) < (round(WdC)-WdC) < 10**(-5):
                # Check whether WdC+1 is power of q
                if -10**(-5) < (round(np.log(WdC+1)/np.log(q))-(np.log(WdC+1)/np.log(q)))
                    < 10**(-5):
                    f = open("output.txt","a+")
                    f.write("\n"+"Code may exist for m ="+str(m))
                    f.close()
                else:
                    f = open("output.txt","a+")
                    # Set case float(np.log(WdC+1)/np.log(q)).is_integer() == False as
                    # type 3 non-exsitence
                    f.write("\n"+"No such code exists, type 3")
                    f.close()
            else:
                f = open("output.txt","a+")
                # Set case float(WdC).is_integer() == False as type 2 non-exsitence
                f.write("\n"+"No such code exists, type 2")
                f.close()

        # One variable Case
        if m == d+1:
            WdC = getWdC(n,d,t,q,l,m)
            if -10**(-5) < (round(WdC)-WdC) < 10**(-5):
                for i in range(1,kmax+1):
                    if q**(i*m)-WdC-1 == 0:
                        if -10**(-5) < (round(np.log(WdC+1)/np.log(q))-(np.log(WdC+1)
                        /np.log(q))) < 10**(-5):
                            f = open("output.txt","a+")
                            f.write("\n"+"Code may exist for m ="+str(m)+"and k ="+str(i))
                            f.close()
                        else:
```

```
                        f = open("output.txt","a+")
                        # Set case float(np.log(WdC+1)/np.log(q)).is_integer() ==
                        #False as type 3 non-exsitence
                        f.write("\n"+"No such code exists, type 3")
                        f.close()
                elif q**(i*m)-WdC-1 > 0:
                    WC = np.zeros((n+1,1)).astype(float)
                    WC[0][0] = 1
                    WC[d][0] = WdC
                    WC[d+1][0] = q**(i*m)-WdC-1
                    mtx2 = getmtx2(n,i,m,q)
                    # Get the weight distribution of C perp
                    WCp = getWCp(mtx1inv,mtx2,WC)
                    flag = True
                    for j in range(1,n+1):
                        if round(WCp[j][0]) == 0:
                            continue
                        # Consider float as integer when error is small enough
                        elif -10**(-5) < (round(WCp[j][0])-WCp[j][0]) < 10**(-5):
                            if round(WCp[j][0]) < 0:
                                flag = False
                                break
                            else:
                                continue
                        else:
                            flag = False
                            break
                    if flag == True:
                        f = open("output.txt","a+")
                        f.write("\n"+"Code may exist for m ="+str(m)+"and k ="+str(i))
                        f.close()
            f = open("output.txt","a+")
            # Set case q**(i*m)-WdC-1 > 0 as type 4 ending
            f.write("\n"+"Program ends here, type 4")
            f.close()
        else:
            f = open("output.txt","a+")
            # Set case float(WdC).is_integer() == False as type 2 non-exsitence
            f.write("\n"+"No such code exists, type 2")
            f.close()

    # Two variables case
    if m == d+2:
        WdC = getWdC(n,d,t,q,l,m)
        if -10**(-5) < (round(WdC)-WdC) < 10**(-5):
            for i in range(1,kmax+1):
                if q**(i*m)-WdC-1 == 0:
                    #print("k =",i)
                    if -10**(-5) < (round(np.log(WdC+1)/np.log(q))-
                        (np.log(WdC+1)/np.log(q)))< 10**(-5):
                        f = open("output.txt","a+")
                        f.write("\n"+"Code may exist for m ="+str(m)+"and k ="+str(i))
                        f.close()
                    else:
                        f = open("output.txt","a+")
                        # Set case float(np.log(WdC+1)/np.log(q)).is_integer() ==
                        # False as type 3 non-exsitence
                        f.write("\n"+"No such code exists, type 3")
                        f.close()
                elif q**(i*m)-WdC-1 > 0:
                    #print("k =",i)
                    mtx2 = getmtx2(n,i,m,q)
                    for x in range(int((q**(m*i)-WdC-1)/((q**m)-1))+1):
                        WC = np.zeros((n+1,1)).astype(float)
                        WC[0][0] = 1
                        WC[d][0] = WdC
                        WC[d+1][0] = q**(i*m)-WdC-x*((q**m)-1)-1
                        WC[d+2][0] = x*((q**m)-1)
                        flag = True
                        WCp = getWCp(mtx1inv,mtx2,WC)
                        for j in range(1,n+1):
                            if round(WCp[j][0]) == 0:
                                continue
                            elif -10**(-5) < (round(WCp[j][0])-WCp[j][0]) < 10**(-5):
                                if round(WCp[j][0]) < 0:
                                    flag = False
                                    break
                                else:
                                    continue
                            else:
                                flag = False
                                break
                        if flag == True:
                            f = open("output.txt","a+")
                            f.write("\n"+"Code may exist for m ="+str(m)+"and k ="+
                            str(i)+"x = "+str(x))
                            f.close()
            f = open("output.txt","a+")
            # Set case q**(i*m)-WdC-1 > 0 as type 4 ending
            f.write("\n"+"Program ends here, type 4")
            f.close()
        else:
            f = open("output.txt","a+")
            # Set case float(WdC).is_integer() == False as type 2 non-exsitence
            f.write("\n"+"No such code exists, type 2")
            f.close()

    # Three variables case
    if m == d+3:
        WdC = getWdC(n,d,t,q,l,m)
        if -10**(-5) < (round(WdC)-WdC) < 10**(-5):
            for i in range(1,kmax+1):
                if q**(i*m)-WdC-1 == 0:
                    #print("k =",i)
                    if -10**(-5) < (round(np.log(WdC+1)/np.log(q))-
                        (np.log(WdC+1)/np.log(q))) < 10**(-5):
                        f = open("output.txt","a+")
                        f.write("\n"+"Code may exist for m ="+str(m)+"and k ="
                        +str(i))
                        f.close()
                    else:
                        f = open("output.txt","a+")
                        # Set case float(np.log(WdC+1)/np.log(q)).is_integer() ==
                        # False as type 3 non-exsitence
                        f.write("\n"+"No such code exists, type 3")
                        f.close()
                elif q**(i*m)-WdC-1 > 0:
                    #print("k =",i)
                    mtx2 = getmtx2(n,i,m,q)
                    for x in range(int((q**(m*i)-WdC-1)/((q**m)-1))+1):
                        for y in range(int((q**(m*i)-WdC-1)/((q**m)-1))-x+1):
                            WC = np.zeros((n+1,1)).astype(float)
                        WC[0][0] = 1
                        WC[d][0] = WdC
                        WC[d+1][0] = q**(i*m)-WdC-x*((q**m)-1)-y*((q**m)-1)-1
                        WC[d+2][0] = x*((q**m)-1)
                        WC[d+3][0] = y*((q**m)-1)
                        flag = True
                        WCp = getWCp(mtx1inv,mtx2,WC)
                        for j in range(1,n+1):
                            if round(WCp[j][0]) == 0:
                                continue
                            elif -10**(-5) < (round(WCp[j][0])-WCp[j][0])
                            < 10**(-5):
                                if round(WCp[j][0]) < 0:
                                    flag = False
                                    break
                                else:
                                    continue
                            else:
                                flag = False
                                break
                        if flag == True:
                            f = open("output.txt","a+")
                            f.write("\n"+"Code may exist for m ="+str(m)+"and k ="
                            +str(i)+"x = "+str(x))
                            f.close()
            f = open("output.txt","a+")
            # Set case q**(i*m)-WdC-1 > 0 as type 4 ending
            f.write("\n"+"Program ends here, type 4")
            f.close()
        else:
            f = open("output.txt","a+")
            # Set case float(WdC).is_integer() == False as type 2 non-exsitence
            f.write("\n"+"No such code exists, type 2")
            f.close()

    # Four variables case
    if m == d+4:
        WdC = getWdC(n,d,t,q,l,m)
        if -10**(-5) < (round(WdC)-WdC) < 10**(-5):
            for i in range(1,kmax+1):
                if q**(i*m)-WdC-1 == 0:
                    #print("k =",i)
                    if -10**(-5) < (round(np.log(WdC+1)/np.log(q))-
                    (np.log(WdC+1)/np.log(q))) < 10**(-5):
                        f = open("output.txt","a+")
                        f.write("\n"+"Code may exist for m ="+str(m)+"and k ="+str(i))
                        f.close()
                    else:
                        f = open("output.txt","a+")
                        # Set case float(np.log(WdC+1)/np.log(q)).is_integer() ==
                        # False as type 3 non-exsitence
                        f.write("\n"+"No such code exists, type 3")
                        f.close()
                elif q**(i*m)-WdC-1 > 0:
                    #print("k =",i)
                    mtx2 = getmtx2(n,i,m,q)
                    for x in range(int((q**(m*i)-WdC-1)/((q**m)-1))+1):
                        for y in range(int((q**(m*i)-WdC-1)/((q**m)-1))-x+1):
                            for z in range(int((q**(m*i)-WdC-1)/((q**m)-1))-x-y+1):
                                WC = np.zeros((n+1,1)).astype(float)
                                WC[0][0] = 1
```

```
                                        WC[d][0] = WdC
                                        WC[d+1][0] = q**(i*m)-WdC-x*((q**m)-1)-y*((q**m)-1)
                                        -z*((q**m)-1)-1
                                        WC[d+2][0] = x*((q**m)-1)
                                        WC[d+3][0] = y*((q**m)-1)
                                        WC[d+4][0] = z*((q**m)-1)
                                        flag = True
                                        WCp = getWCp(mtx1inv,mtx2,WC)
                                        for j in range(1,n+1):
                                            if round(WCp[j][0]) == 0:
                                                continue
                                            elif -10**(-5) < (round(WCp[j][0])-WCp[j][0])
                                            < 10**(-5):
                                                if round(WCp[j][0]) < 0:
                                                    flag = False
                                                    break
                                                else:
                                                    continue
                                            else:
                                                flag = False
                                                break
                                        if flag == True:
                                            f = open("output.txt","a+")
                                            f.write("\n"+"Code may exist for m ="+str(m)+
                                            "and k ="+str(i)+"x = "+str(x))
                                            f.close()
                    f = open("output.txt","a+")
                    # Set case q**(i*m)-WdC-1 > 0 as type 4 ending
                    f.write("\n"+"Program ends here, type 4")
                    f.close()
                else:
                    f = open("output.txt","a+")
                    # Set case float(WdC).is_integer() == False as type 2 non-exsitence
                    f.write("\n"+"No such code exists, type 2")
                    f.close()


# For convenience, define codes for flexible lambda and check the lambda maximum by codes
def ChkExs_l(t,n,d,q,m,lmin,lmax):
    kmax = getkmax(n,d,m)
    l_end = getlmax(t,n,d,q,m,kmax)
    f = open("output.txt","a+")
    f.write("\n\n"+"Here starts t = "+str(t)+" n = "+str(n)+" d = "+str(d)+" q = "+str(q)
    +" m = "+str(m)+" lmin = "+str(lmin)+" lmax = "+str(lmax)+" designs.")
    f.write("\n"+"Where kmax = "+str(kmax)+" and lambda_end = "+str(l_end)+" by MRD: ")
    f.close()
    print("maximun k =",kmax)
    print("maximun lambda can be",l_end)
    if l_end < lmin:
        f = open("output.txt","a+")
        # Set case l_end < lmin as type 5 non-exsitence
        f.write("\n"+"For all lambda here, No such code exists, type 5")
        f.close()
    elif lmin <= l_end <= lmax:
        for i in range(lmin,l_end+1):
            #print("lambda = ",i)
            #print("# of avaiable k :",chk_k(t,n,d,i,q,m))
            f = open("output.txt","a+")
            f.write("\n"+"For lambda = "+str(i))
            f.close()
            ChkExs(t,n,d,i,q,m)
    elif lmax < l_end:
        for i in range(lmin,lmax+1):
            #print("Lambda = ",i)
            #print("# of avaiable k :",chk_k(t,n,d,i,q,m))
            f = open("output.txt","a+")
            f.write("\n"+"For lambda = "+str(i))
            f.close()
            ChkExs(t,n,d,i,q,m)
```

Sample application codes:

In [11]:

```
# 2-(6,3,Lambda)2 design
t = 2
n = 6
d = 3
q = 2
m = 5
lmin = 3
lmax = 15
ChkExs_l(t,n,d,q,m,lmin,lmax)
```

```
maximun k = 3
maximun lambda can be 11
```

Codes end here

We have updated the codes such that it can check for $m = d, m = d + 1, m = d + 2, m = d + 3$ and $m = d + 4$. What we need to do everytime we move to a new design are just setting parameters t, n, d, q, m and the interval of λ we want to check. The codes will write the results into a file called "output.txt" in the same file where python codes' file is.

The codes will also directly print out the maximum k it can reach by **MRD** and the maximun λ according to the total number of elements can a code have when we set parameters unchanged. For example, we have $2\text{-}(6, 3, \lambda)$ design with $kmax = 3$ as showed above and this design is known realizable then

$$W_3(C) = (2^m - 1)\begin{bmatrix}6\\2\end{bmatrix}_2\begin{bmatrix}3\\2\end{bmatrix}_2^{-1}\lambda$$

and if we set $m = 5$ unchanged then we require that $2^{m\cdot kmax} - (2^m - 1)\begin{bmatrix}6\\2\end{bmatrix}_2\begin{bmatrix}3\\2\end{bmatrix}_2^{-1}\lambda - 1 \geq 0$ which means that λ will have a bound above in this case. That is why the codes showed above as "maximun lambda can be 11".

The codes will only return successful cases or something (non-existence) with type 1,2,3,4,5 which have been explained in the line starting with # in the codes. Actually we met some problems when making the codes and applying them in the school cluster. In the codes, we set an interval $[-10^{-5}, 10^{-5}]$ and check whether the error between a number and its rounding number is in this interval to determine whether it is integer. Before this, we first use $'float(number).is\_integer() == True'$ command to check integers but we found that python will transfer fraction to float first which is no longer exactly that fraction anymore when dealing with fraction numbers, so the result number we get will sometimes have a very small error compared with the exact results and this will make the command not work well. An example showed below will make everything much clearer.

Example 45. We want to calculate q-binomial: $\begin{bmatrix}5\\3\end{bmatrix}_2 := \prod_{i=0}^{3-1}\frac{2^5-2^i}{2^3-2^i} = \frac{2^5-2^0}{2^3-2^0}\cdot\frac{2^5-2^1}{2^3-2^1}\cdot\frac{2^5-2^2}{2^3-2^2} = \frac{31}{7}\cdot\frac{30}{6}\cdot\frac{28}{4} = 155$ and the python codes will return:

In [6]:

```
(31/7)*(30/6)*(28/4)
```

Out[6]:

```
155.00000000000003
```

which is a number very close to 155 and that's exactly the problem we have just talked about. Also as we said after the outputs of the codes for example 42, the package we use for getting inverse of a matrix will have the same problem. Thus we introduce a way to solve this problem as what we have said before: Calculate the error between the number the codes return and the rounding number of it. If the error is small enough, then we view this number the codes return as integer and continue the process. We view those errors which are in the interval $[-10^{-5}, 10^{-5}]$ as small enough.

And here are some tables we focus on for checking existence of possible codes.

Tables of parameters of design known to be realizable [7]

Table 1 Parameters of simple t-$(n, d, \lambda)_2$ designs known to be realizable

t-$(n, d, \lambda)_q$	λ_{min}	λ_{max}	λ
2-$(6, 3, \lambda)_2$	3	15	$3, 6$. **Open**:$-$
2-$(7, 3, \lambda)_2$	1	31	$3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 14, 15$. **Open**:$-$
2-$(8, 3, \lambda)_2$	21	63	21. **Open**:$-$
2-$(8, 4, \lambda)_2$	7	651	$7, 14, 21, 35, 49, 56, 63, 70, 84, 91, 98, 105, 112, 126, 133, 140, 147, 154, 161, 168, 175, 189, 196, 203, 210, 217, 231, 245, 252, 259, 266, 273, 280, 294, 301, 308, 315$. **Open**:$28, 42, 77, 119, 182, 224, 268, 287, 322$
3-$(8, 4, \lambda)_2$	1	31	$11, 15$. **Open**: $1, \dots, 10, 12, 13, 14$
2-$(9, 3, \lambda)_2$	1	127	$7, 12, 19, 21, 22, 24, 31, 36, 42, 43, 48, 49, 55, 60, 63$
2-$(9, 4, \lambda)_2$	7	2667	$21, 63, 84, 126, 147, 1889, 210, 252, 273, 315, 336, 378, 399, 441, 462, 504, 525, 567, 588, 630, 651, 693, 714, 756, 777, 819, 840, 882, 889, 903, 945, 966, 1008, 1029, 1071, 1092, 1134, 1155, 1197, 1218, 1260, 1281, 1323$
3-$(9, 4, \lambda)_2$	21	63	**Open**: 21
2-$(10, 3, \lambda)_2$	3	255	$15, 30, 45, 60, 75, 90, 105, 120$
2-$(10, 4, \lambda)_2$	5	10795	$595, 1020, 1615, 1785, 1870, 2040, 2635, 3060, 3570, 3655, 4080, 4165, 4675, 5100, 5355$
3-$(10, 4, \lambda)_2$	1	127	$-$
2-$(10, 5, \lambda)_2$	15	97155	$765, 4590, 5355, 6885, 7650, 9180, 9945, 2295, 3060, 11475, 12240, 13770, 14535, 16065, 16830, 18360, 191125, 20655, 21420, 22950, 23715, 25245, 26010, 27540, 28305, 29835, 30600, 32130, 32385, 32895, 34425, 35190, 36720, 37485, 39015, 39780, 41310, 42075, 43605, 44370, 45900, 46665, 48195$
3-$(10, 5, \lambda)_2$	21	63	**Open**: 21
2-$(11, 3, \lambda)_2$	7	511	$7, 245, 252$
2-$(11, 4, \lambda)_2$	35	43435	$-$
2-$(11, 5, \lambda)_2$	5	788035	$43435, 74460, 117895, 130305, 136510, 148920, 192355, 223380, 260610, 266815, 297840, 304045, 341275, 372300, 390915$
2-$(12, 3, \lambda)_2$	3	1023	$-$

-			
$2\text{-}(12,4,\lambda)_2$	7	174251	–
$2\text{-}(12,5,\lambda)_2$	465	6347715	–
$2\text{-}(12,6,\lambda)_2$	31	53743987	$2962267, 5078172, 8040439, 8886801, 9309982, 10156344, 13115611, 15234516, 17773602, 18196783, 20312688, 20735869, 23274955, 25390860, 26660403$
$2\text{-}(12,k,\lambda)_2$	–	–	–
$2\text{-}(13,3,\lambda)_2$	1	2047	$1,2,\ldots,6,7,8,\ldots,1023$. **Open**:–

Table 2 Parameters of simple $t\text{-}(n,d,\lambda)_3$ designs known to be realizable

$t\text{-}(n,d,\lambda)_q$	λ_{min}	λ_{max}	λ
$2\text{-}(6,3,\lambda)_3$	4	40	$8,12,16,20$. **Open**:–
$2\text{-}(7,3,\lambda)_3$	1	121	$5,6,\ldots,12,13,14,\ldots,40,41,42,\ldots,60$.**Open**:–
$2\text{-}(8,3,\lambda)_3$	52	364	$52,104,156$. **Open**:–
$2\text{-}(8,4,\lambda)_3$	13	11011	$91\cdot 5, 91\cdot 6,\ldots,91\cdot 60$
$2\text{-}(10,3,\lambda)_3$	4	3280	1640
$2\text{-}(11,3,\lambda)_3$	13	9841	13
$2\text{-}(13,4,\lambda)_2$	1	88573	13

Table 3 Parameters of simple $t\text{-}(n,d,\lambda)_4$ designs known to be realizable

$t\text{-}(n,d,\lambda)_q$	λ_{min}	λ_{max}	λ
$2\text{-}(6,3,\lambda)_4$	5	85	$10,15,25,30,35$. **Open**: $5,40$
$2\text{-}(7,3,\lambda)_4$	1	341	21
$2\text{-}(8,4,\lambda)_4$	21	93093	5733

Table 4 Parameters of simple $t\text{-}(n,d,\lambda)_5$ designs known to be realizable

$t\text{-}(n,d,\lambda)_q$	λ_{min}	λ_{max}	λ
$2\text{-}(6,3,\lambda)_5$	6	78	78. **Open**: $6,12,\ldots,72$
$2\text{-}(7,3,\lambda)_5$	1	781	31
$2\text{-}(8,4,\lambda)_5$	31	508431	20181
$2\text{-}(10,4,\lambda)_5$	6	97656	48828

The tables above are designs which are known to be realizable and we will focusing but not only on the parameters showed in these tables. We will check every design and every λ between the λ_{min} and λ_{max} above but not just those specific λs showed in the tables which have been proved to be realizable or are open for proof.

If our codes return something exciting i.e. if it returns that some design with m we set may be held by some codes by our method, then we will use magma to check the weight distribution of $C^{\perp}$ again to prevent that error interval we set is a bit loose (magma codes don't have error problem). Setting a loose error interval will just let codes check more cases (i.e. viewing more cases as something we want to find but actually they may not) to prevent the checking integer problem may make some problem (Even the probability of this happening is very small, almost impossible if we set a tight and appropriate interval, but we still think that setting a loose interval will be better). In other words, we won't miss any successful cases. Just prevent anything unpredictable happening, we choose to double check by both magma and python and set a slightly loose error interval.

Thanks to my supervisor's help, she has finished the magma codes already before. After her consent, the codes will be put below. These codes are for checking weight distrution of dual codes in magma.

Magma codes:

In [ ]:

```
/////computes the `q-product', as shown below,
/////which is used to compute the q-binomial coefficient
qprod:=function(q,t,r)
if r in [1..t] then
return &*{*q^t-q^j:j in [0..r-1]*};
elif
r eq 0 then
return 1;
else
return 0;
end if;
end function;

/////computes the q-binomial coefficient
qbin:=function(q,n,r)
if r le -1 then
ans:=0;
elif
q eq 1 then
ans:=Binomial(n,r);
else
ans:=qprod(q,n,r)/qprod(q,r,r);
end if;
return ans;
end function;

/////computes the (n+1) x (n+1) matrix that appears
/////in the LHS of Th 6 of project.pdf notes
qpascal1:=function(q,n)
L:=[[qbin(q,n-i,j): i in [0..n]]: j in [0..n]];
M:=Matrix(Rationals(),n+1,n+1,L);
return M;
end function;

/////computes the (n+1) x (n+1) matrix that appears
/////in the RHS of Th 6 of project.pdf notes
qpascal2:=function(q,n,m,k)
L:=[[q^(m*(n-k-j))*qbin(q,n-i,j-i): i in [0..n]]: j in [0..n]];
M:=Matrix(Rationals(),n+1,n+1,L);
return M;
end function;
```

Sample application codes:

```
In [ ]:
```

```
/////checking w.e. of dual code for putative code
/////holding a 2-(7,3,1) design
/////checked m=4,m=5,k=4,5

m:=7;
k:=4;
f:=Floor((2^(m*k)-(2^m-1)*381-1)/(2^m-1));
for x in [1..f] do
W:=Matrix(Rationals(),8,1,[1,0,0,(2^m-1)*381,0,0,2^(m*k)-(2^m-1)*(381+x)-1,x*(2^m-1)]);
wenu:=qpascal1(2,7)^(-1)*qpascal2(2,7,7,k)*W;
flag:=true;
for i in [2..8] do
if IsIntegral(wenu[i][1]) eq false then
flag:=false;
break;
end if;
end for;
if flag eq true then
print wenu;
end if;
end for;
```

Codes end here

And if there is one case successfully returned by both python codes and magma codes, then we will check whether this possible code is AM code or not. If it's not, we will move to another design or another m and continue our checking. And it will be exciting if the codes can return some case.

4. Results and Conclusions

Here are some tables which show the results of what we have checked based on the previous tables.

Table 1 Parameters of simple t-$(n,d,\lambda)_2$ designs known to be realizable

t-$(n,d,\lambda)_q$	λ_{min}	λ_{max}	$m=d$	$m=d+1$	$m=d+2$	$m=d+3$	$m=d+4$
2-$(6,3,\lambda)_2$	3	15	×	×	×	−	−
2-$(7,3,\lambda)_2$	1	31	×	×	×	−	−
2-$(8,3,\lambda)_2$	21	63	×	×	×	−	−
2-$(8,4,\lambda)_2$	7	651	×	×	×	−	−
3-$(8,4,\lambda)_2$	1	31	×	×	×	−	−
2-$(9,3,\lambda)_2$	1	127	×	×	×	−	−
2-$(9,4,\lambda)_2$	7	2667	×	×	×	−	−
3-$(9,4,\lambda)_2$	21	63	×	×	×	−	−
2-$(10,3,\lambda)_2$	3	255	×	×	×	−	−

-							
2-$(10,4,\lambda)_2$	5	10795	×	×	×	−	−
3-$(10,4,\lambda)_2$	1	127	×	×	×	−	−
2-$(10,5,\lambda)_2$	15	97155	×	×	×	−	−
3-$(10,5,\lambda)_2$	21	63	×	×	×	−	−
2-$(11,3,\lambda)_2$	7	511	×	×	×	−	−
2-$(11,4,\lambda)_2$	35	43435	×	×	×	−	−
2-$(11,5,\lambda)_2$	5	788035	×	×	×	−	−
2-$(12,3,\lambda)_2$	3	1023	×	×	−	−	−
2-$(12,4,\lambda)_2$	7	174251	×	×	×	−	−
2-$(12,5,\lambda)_2$	465	6347715	×	×	×	−	−
2-$(12,6,\lambda)_2$	31	53743987	×	×	×	−	−
2-$(12,k,\lambda)_2$	−	−	−	−	−	−	−
2-$(13,3,\lambda)_2$	1	2047	×	×	×	−	−

Table 2 Parameters of simple t-$(n,d,\lambda)_3$ designs known to be realizable

t-$(n,d,\lambda)_q$	λ_{min}	λ_{max}	$m=d$	$m=d+1$	$m=d+2$	$m=d+3$	$m=d+4$
2-$(6,3,\lambda)_3$	4	40	×	×	×	−	−
2-$(7,3,\lambda)_3$	1	121	×	×	×	−	−
2-$(8,3,\lambda)_3$	52	364	×	×	×	×	−
2-$(8,4,\lambda)_3$	13	11011	×	×	×	−	−
2-$(10,3,\lambda)_3$	4	3280	×	×	×	−	−
2-$(11,3,\lambda)_3$	13	9841	×	×	×	−	−
2-$(13,4,\lambda)_2$	1	88573	×	×	×	−	−

Table 3 Parameters of simple t-$(n,d,\lambda)_4$ designs known to be realizable

t-$(n,d,\lambda)_q$	λ_{min}	λ_{max}	$m=d$	$m=d+1$	$m=d+2$	$m=d+3$	$m=d+4$
2-$(6,3,\lambda)_4$	5	85	×	×	×	×	×
2-$(7,3,\lambda)_4$	1	341	×	×	×	×	−
2-$(8,4,\lambda)_4$	21	93093	×	×	×	×	−

Table 4 Parameters of simple t-$(n,d,\lambda)_5$ designs known to be realizable

t-$(n,d,\lambda)_q$	λ_{min}	λ_{max}	$m=d$	$m=d+1$	$m=d+2$	$m=d+3$	$m=d+4$
2-$(6,3,\lambda)_5$	6	78	×	×	×	×	×
2-$(7,3,\lambda)_5$	1	781	×	×	×	×	−

-							
2-$(8,4,\lambda)_5$	31	508431	×	×	×	×	×
2-$(10,4,\lambda)_5$	6	97656	×	×	×	×	×

The "×" in the tables above means we have checked this design and no code hold that design.
The "-" means that we haven't checked up to such m for such design or no interval of λ of such design are known to be realizable.

As we said before, we used 'school cluster' to run our codes. We first use a server called 'magnet' but we heart from administrator that it is not that fast as another one called 'orr2'. But when we tried to apply our codes on 'orr2', we found a problem. When you try dividing an interger by an integer, the python will always return an integer. For example, if we want the python in 'orr2' help us calculate 4 dividng by 2 it will of course return 2 but if we let python calculate what is 5 over 3, it will return 1. After some checking and experiments, we found the reason why that happened. It's because of the version of python. The python the 'orr2' use is version 2.7 but my codes is for version 3.0 and above and that's why the codes do work in my computer but return wrong results in the server. With the big help from administrator, we finally can run our codes well in the server.

Up to now, we have checked (as the tables showed above) all the designs for $m \leq d+2$ except the design 2-$(12,3,\lambda)_2$ with $m = d+2$ case. The codes of this case are still running and it may take more than 1 year to finish (since there are really too many cases needing to be checked). So we can't conclude anything about this designs whith $m = d+2$, but for this design up to $m = d+1$ and all other designs up to $m = d+2$ case, we can conclude that no code hold any of these designs. Yes, even we set a relatively large checking-integer error interval $[-10^{-5}, 10^{-5}]$, the codes still did't return anything we'd like to see.

In the future, we will first move to $m = d+3$ and $m = d+4$ cases based on the tables above (since we have finished the python codes for these two cases). It will take a lot more time than $m = d+2$ case to finish running the codes for those designs. We will also continue to code for $m = d+x$ cases as x going larger and larger if the codes still return nothing after finishing the $d+3$ and $d+4$ cases.

Hope that our codes will return something exciting in the future.

5. References

[1] C. Bachoc, *On Harmonic Weight Enumerators of Binary Codes, Designs, Codes and Cryptography*, 18 (1-3) pp. 11–28, 1999.
[2] E. Byrne, A. Ravagnani, *An assmus-mattson theorem for rank metric codes*, pp. 1,2, 9-11.
[3] E. Byrne, notes from a lecture course given in *Introduction to Error-Correcting Codes*,UCD, pp. 1-18, 46-49, 2018.
[4] M. Braun, *Systematic Construction of q-Analogs of t(v, k, λ)-Designs*, Designs, Codes and Cryptography, 34 (1) pp. 55–70, 2005.
[5] M. Braun, T. Etzion, P. R. Ostergard, A. Vardy, A. Wassermann, *Existence of q-Analogs of Steiner Systems*, Forum Math. Pi 4, 2016.
[6] M. Braun, M. Kiermaier, A. Kohnert, R. Laue, *Large Sets of Subspace Designs*, Jour. Comb. Thy(A), 147, pp. 155-185, 2017.
[7] M. Braun, M. Kiermaier, A. Wassermann, *q-Analogs of Designs: Subspace Designs*, in *Network Coding and Subspace Designs*, Eds. M. Greferath, M. Pavcevic, A. Vazquez-Castro, N. Silberstein, Springer-Verlag Berlin, pp. 193-194, 2018.
[8] M. Braun, M. Kiermaier, A. Wassermann, *Computational Methods in Subspace Designs*, in *Network Coding and Subspace Designs*, Eds. M. Greferath, M. Pavcevic, A. Vazquez-Castro, N. Silberstein, Springer-Verlag

Berlin, 2018.
[9] P. Cameron, J. H. van Lint, *Designs, Graphs, Codes and their Links*, London Mathematical Society Student Texts, vol. 22, 1991.
[10] W. C. Huffman, V. Pless, *Fundamentals of Error-Correcting Codes*, Cambridge University Press, 2003.
[11] P. Delsarte, *Bilinear Forms over a Finite Field with Applications to Coding Theory*, Journal of Combinatorial Theory, Series A, 25, 226–241, 1978.
[12] P. Delsarte, *Bilinear Forms over a Finite Field with Applications to Coding Theory*, Journal of Combinatorial Theory, Series A, 25, 226–241, 1978.
[13] E. F. Assmus, Jr., H. F. Mattson, Jr., *New 5-Designs*, Jour. Combinatorial Theory, 6, 122-151, 1969.
[14] E. F. Assmus, J. Key, *Designs and their Codes*, Cambridge University Press, 1992.
[15] J. F. MacWillams, N. J. A. Sloane, *The Theory of Error Correcting Codes*, North-Holland Mathematical Library, 1977.
[16] A. Fazeli, S. Lovett, A. Vardy, *Nontrivial t-Designs Over Finite Fields Exist For All t*, J. Combin. Theory Ser. A 127, 149160, 2014.
[17] T. Itoh, *A New Family of 2-designs over $GF(q)$ Admitting $SL_m(q^\ell)$*, Geom. Dedicata 69(3), 261286, 1998.
[18] D. J. Shin, P. V. Kumar, T. Helleseth, *An Assmus-Mattson-Type Approach for Identifying 3-designs from Linear Codes over Z_4*, Designs, Codes and Cryptography, 31, No. 1, pp. 75–92, 2004.
[19] P. J. Cameron, *Generalisation of Fisher's inequality to fields with more than one element*, Combinatorics, Proceedings of the British Combinatorial Conference 1973. Eds. T. P. McDonough, V. C. Mavron. LMS Lecture Note Series 13, Cambridge University Press, pp. 9–13, 1974.
[20] R. Lidl, H. Niederreiter, *Introduction to finite fields and their applications*, chapter 1,2, 1986.
[21] A. R. Calderbank, P. Delsarte, N. J. A. Sloane, *A Strengthening of the Assmus-Mattson Theorem*, IEEE Transactions on Information Theory, 37, No. 5, pp.1261–1268, 1991.
[22] A. Ravagnani, *Rank-Metric Codes and their Duality Theory*. Designs, Codes and Cryptography, 80, No. 1, pp. 197–216, 2016. [23] H. Suzuki, *2-Designs over $GF(q)$* Graphs Comb. 8 (4), pp. 381–389.
[24] S. Thomas. *Designs over Finite Fields*, Geom. Dedicata 24, 2, pp. 237-242, 1987.
[25] J. V. S. Morales, H. Tanaka, *An Assmus-Mattson Theorem for Codes Over Commutative Association Schemes*, Designs Codes and Cryptography, 86, No. 5, pp. 1039–1062, 2018.