Weight-Enumerators of all Cyclic Linear Codes $[n, k]$ over $GF(q)$ up to a Given Size

Himanshu Chandra, Nkhalo Malawo, Lucy Martinez

Spring 2024

1 Introduction

Error-correcting codes are a fundamental tool for protecting data from noise. Applications for error-correcting range from complexity theory, to pseudorandomness, and cryptography.

We say that a (n, M, d)-code is a code of length n containing M codewords, and minimum distance d. One of the main problems in coding theory is to find M as large as possible so that we get a large good code with n small and large d. In this way, we have fast transmission of messages (n) from a wide variety (M) of messages and are able to correct as many errors (d).

In this project, we aim to construct a database for weight enumerators of cyclic linear codes $[n, k]$ over $GF(q)$ up to a given size. For each generated cyclic linear code and its weight enumerator polynomial, we find the average and standard deviation of each weight.

The outline of this paper is as follows. We first begin with some background for the reader to familiarize with some key terms and introduce the problem in more detail. We then proceed by introducing Maple procedures that were written to generate all necessary data. Following this section, we discuss the results from our analysis and end with a conclusion.

2 Background

In this section, we aim to introduce key terms in coding theory. All definitions are cited from [2], unless stated otherwise. We refer the reader to [2] for more necessary background.

Throughout this paper, we assume that the alphabet F, is the Galois field $GF(q)$, where q is a prime power, and we regard $(F_q)^n$ as the vector space $V(n, q)$.

Definition 1. *A **q-ary code** C is a set of sequences where each symbol is from a set $F_q = \{\lambda_1, \cdots, \lambda_q\}$ of q distinct elements. The set F_q is called the alphabet and it is usually taken to be the set $\mathbb{Z}_q = \{0, 1, 2, \cdots, q-1\}$.*

In this paper we focus on linear codes, specifically cyclic codes.

Definition 2. *A linear code over $GF(q)$ is just a subspace of $V(n, q)$, for some positive integer n. Thus a subset C of $V(n, q)$ is a linear code if and only if*

1. *$\boldsymbol{u} + \boldsymbol{v} \in C$ for all $\boldsymbol{u}$ and $\boldsymbol{v}$ in C, and*
2. *$a\boldsymbol{u} \in C$, for all $\boldsymbol{u} \in C$, $a \in GF(q)$.*

If C is a k-dimensional subspace of $V(n,q)$, then the linear code C is called an [n,k]-code, or sometimes, if we wish to specify also the minimum distance d of C, an [n, k, d]-code.

Linear codes are useful for a few reasons:

1. Finding the minimum distance of a linear code requires less comparisons than that of a general code.
2. We can specify a linear *[n, k]-code* by simply giving a basis of k codewords. To find a specific non-linear code, we may have to list all the codewords.
3. There are established procedures for encoding and decoding linear codes, which are discussed in Chapter 6 of [2]

Cyclic codes are a specific type of linear code. They possess a rich algebraic structure, while practically they can be efficiently implemented by means of simple devices known as shift registers. Additionally, many important codes, such as binary Hamming codes and Golay codes are equivalent to cyclic codes.

Definition 3. *A code C is **cyclic** if (i) C is a linear code and (ii) any cyclic shift of a codeword is also a code, i.e. whenever $a_0a_1...a_{n-1}$ is in C, then so is $a_{n-1}a_0a_1...a_{n-2}$*

Take the binary code $C = \{000, 101, 011, 110\}$. It is is linear and any shift of it's codewords is still a member of C.

In this project, we need to generate cyclic *[n,n-a]*-codes over $GF(q)$ for various n, a, and q prime. There is a powerful fact about cyclic codes that we used to generate all possible cyclic codes within our parameters.

Fact: Any cyclic code can be generated by a polynomial. Before we explain how, let's discuss some necessary information about polynomials.

We start with the set of all polynomials in x with coefficients in F_q, we denote this set $F[x]$. Polynomials in $F[x]$ can be added, subtracted and multiplied just as one expects. We can even divide polynomials, using the division algorithm. $F[x]$ is an example of an algebraic structure known as a ring. This means $F[x]$ shares qualities with other rings, like the set of all integers (See Chapter 3 of [2]).

3 Maple Procedures

Accompanying this paper is the Maple package `proj1.txt`, authored by Himanshu Chandra, Nkhalo Malawo, and Lucy Martinez. Additionally, certain procedures written by Doron Zeilberger were implemented during class. We thank Joseph Koutsoutis who provided one of the procedures, as described below.

`HD(u,v)`, where `u` is a codeword, `v` is another codeword of the same length. This procedure is used to calculate the Hamming distance between two words of the same length.

`LtoC(q,M)`, where `q` is number of symbols used for coding i.e. size of the alphabet, `M` is a list of basis vectors for our linear code over $GF(q)$. This procedure outputs all the codewords (the actual subset of $GF(q)^n$) with q^k elements).

`CtoL(n,x,g)`, where `n` is length of the codewords, `x` is a variable,`g` is a generator polynomial $g(x)$ over R^n(mod (x^n-1)). This procedure outputs the generator matrix for the cyclic code $< g(x) >$ over R^n(mod (x^n-1)).

`CtoDual(n,q,x,g)`, where `n` is length of the codewords, `q` is a prime number of symbols, `x` is a variable, `g` is a polynomial g (a generator of a cyclic code) mod q. This procedure is used to create the generator matrix for the dual code of CtoL(n,x,g).

`AllFactors(n,q,x)` inputs the length of the codewords denoted by `n`, the number of symbols `q` where q is prime, and a variable `x`. It outputs all the factors of $x^n - 1 \mod q$ except for 1 and $x^n - 1$. This procedure was provided by Joseph Koutsoutis.

`DualCodes(n,q,a)`, where `n` is length of the codewords, `q` is a prime number of symbols, `a` is the dimension of the subspace i.e number of codewords in the code. This procedure is used to generate all cyclic linear $[n, n-a]$ dual codes over $GF(q)$ of length n.

`WtEnumerator(q,C,t)`, where `q` is a prime number of symbols, `C` is a linear code C over $GF(q)$, `t` is a variable. This procedure is used to generate the weight-enumerator, in t, of the linear code C.

`MacWm(n,q,a,Cdual,z)`, where `n` is length of codewords, `q` is a prime number of symbols, `a` is is the dimension of the subspace i.e number of codewords in the code, `Cdual` is a Dual Code C over $GF(q)$, `z` is a variable. This procedure is used to compute its weight enumerator polynomial for the code C using the MacWilliams identity.

`AveAndMoms(f,z,N)`, where `f` is a probability generating function $f(z)$ (a polynomial, rational function and possibly beyond), `z` is a variable, `N` is the required moment (about the mean). This procedure is used to return a list whose first entry is the average (alias expectation, alias mean) followed by the variance, the third moment (about the mean) and so on, until the N-th moment (about the mean). If $f(1)$ is not 1, than it first normalizes it by dividing by $f(1)$ (if it is not 0). This procedure was provided in [1].

`Di(a,b)`, where `a` is numerator of probability, `b` is the denominator. This procedure is used to output true with prob. a/b. Try: Di(1,3);

`Rqnk(q,n,k)`, where `q` is a prime number of symbols, `n` is the length of codewords, `k` is the dimension of the subspace. This procedure is used to create (uniformly at) random n row-echelon matrices over $GF(q)^n$ representing k-dim subspaces. Try Rqnk(2,10,5);

`Span1(M,n,q))`, where `M` is a base for a vector subspace over $GF(q)^n$, `n` is the length of the codewords, `q` is a prime number of symbols. This procedure is find all the vectors for a base M for a vector subspace over $GF(q)^n$. Try: M:=Rqnk(2,10,3); Span1(M,10,2);

`Wt1(v)`, where `v` is a vector. This procedure is used to compute the weight i.e. the number of non-zero entries in the vector v.

`WtEnu(M,q,z)`, where `M` is a base for a vector subspace over $GF(q)^n$, `q` is number of symbols, `z` is a variable. This procedure is used to generate the he weight-enumerator according to the variable z, of the subspace of $GF(q)^n$ generated by M (where n:=nops(M[1]). Try: M:=Rqnk(2,10,3); WtEnu(M,2,z);

`Stat(n,q,a,z)`, where `n` is length of codewords, `q` is a prime number of symbols, `a` is the dimension of the subspace i.e number of codewords in the code, `z` is a variable. This procedure is used to generate all cyclic linear $[n, n-a]$ dual codes over $GF(q)$ of length n, C=DualCodes(n,q,a), and outputs a list of lists with the following information generating polynomial ($g(x)$) of the code, weight enumerator of the code, average and standard deviation for each code in C=DualCodes(n,q,a). For example, `Stat(7,2,3,z);` returns

$$[[x^3+x+1, z^7+7z^4+7z^3+1, 7/2, \sqrt{7}/2], [x^3+x^2+1, z^7+7z^4+7z^3+1, 7/2, \sqrt{7}/2]].$$

The above example is

`CreateTable(n,q,a,z)`, where `n` is length of codewords, `q` is number of symbols, `a` is is the dimension of the subspace i.e number of codewords in the code, `z` is a variable. This procedure is used to tabulate the results.

4 Results

In this section, we discuss some of the observations taken from generating some data. The data discussed in this section is attached as a separate file as some of the files are large.

The goal of this project is to generate as much as data as possible using `CreateTable(n,q,a,z)`. This procedure will output four columns as follows: The first column prints $[n, q, k]$ which corresponds to the cyclic code of length n over $GF(q)$, the second columns outputs the generator polynomial of the code, $g(x)$, of degree $n-k$, the third column prints the weight enumerator in variable z, and the last column outputs a list of length two where the first value is the average weight and the second value is the standard deviation of the weight of the code.

Running `CreateTable(120,2,3,z);` outputs a table with 107 rows for $3 \leq n \leq 120, 0 \leq k \leq 117$ and $q = 2$. In this table, all generator polynomials will be of degree 3 since `Stat(n,q,a,z)` looks for codes with generator polynomial of degree $a = n - k$. The time that it takes to generate this table in Maple is 8.406 seconds. In this table, one of the examples is the cyclic linear code with parameters $[4, 2, 1]$ with generator polynomial $x^3 + x^2 + x + 1$, and weight enumerator $z^4 + 1$. The average weight for this cyclic code is $\frac{3}{2}$ and its standard deviation is $\frac{\sqrt{3}}{7}$.

Running `CreateTable(120,2,4,z);` outputs a table with 144 rows for $4 \leq n \leq 120, 0 \leq k \leq 116$ and $q = 2$. In this table, all generator polynomials will be of degree 4 since `Stat(n,q,a,z)` looks for codes with generator polynomial of degree $a = n - k$. The time that it takes to generate this table in Maple is 7.281 seconds. One of the examples is the cyclic linear code with parameters $[5, 2, 1]$ with generator polynomial $x^4 + x^3 + x^2 + 1$, and weight enumerator $z^5 + 1$. The average weight for this cyclic code is $\frac{5}{2}$ and its standard deviation is $\frac{5}{2}$.

Generating data for $4 \leq n \leq 120, q = 2$ for generator polynomials of degree $n - k$ is possible for up to $n - k = 12$. The running time in Maple for `CreateTable(120,2,12,z)` is 56.453 seconds.

We generated data for the following:

- `CreateTable(120,2,a,z)` for $3 \leq a \leq 12$.
- `CreateTable(50,3,3,z)` and `CreateTable(50,3,4,z)`.
- `CreateTable(50,5,3,z)`.
- `CreateTable(30,7,3,z)`.

Running `CreateTable(50,3,3,z)` takes 1.218 seconds in Maple, while `CreateTable(70,3,3,z)` takes 3.140 seconds. However, once you run `CreateTable(50,3,3,z)`, it takes a while. We do not have an estimate on how long it takes.

One of the observations on generating data is the following.

Observation 1. *The average weight for a cyclic linear code $[n, k]$ over $GF(q)$ is*

$$n\left(1 - \frac{1}{q}\right).$$

Running through the tables, we notice this pattern in the average weight. We do not know whether this is a known fact. We invite the reader to provide a reference if this is known.

References

[1] S. B. Ekhad, and D. Zeilberger. *Implementing and Experimenting with the Calabi-Wilf algorithm for random selection of a subspace over a finite field.* Personal Journal of Shalosh B. Ekhad and Doron Zeilberger (2023).

[2] R. Hill. *A First Course in Coding Theory*. Oxford University Press Inc. (1986).