CODING THEORY PACKAGE FOR MACAULAY2

TAYLOR BALL, EDUARDO CAMPS, HENRY CHIMAL-DZUL, DELIO JARAMILLO-VELEZ, HIRAM H. LÓPEZ, NATHAN NICHOLS, MATTHEW PERKINS, IVAN SOPRUNOV, GERMAN VERA-MARTÍNEZ, AND GWYN WHIELDON

Abstract. In this Macaulay2 [5] package we define an object called *linear code*. We implement functions that compute basic parameters and objects associated with a linear code, such as generator and parity check matrices, the dual code, length, dimension, and minimum distance, among others. We define an object *evaluation code*, a construction which allows to study linear codes using tools of algebraic geometry and commutative algebra. We implement functions to generate important families of linear codes such as Hamming codes, cyclic codes, Reed–Solomon codes, Reed–Muller codes, Cartesian codes, monomial–Cartesian codes, and toric codes. In addition, we define functions for the syndrome decoding algorithm and locally recoverable code construction, which are important tools in applications of linear codes. The package *CodingTheory.m2* is available at `https://github.com/Macaulay2/Workshop-2020-Cleveland/tree/CodingTheory/CodingTheory`

1. Introduction

Coding theory has been extensively studied since 1949, when Claude Shannon proved in his seminal paper [15] that linear codes can be used to reliably transmit information from a single source to a single receiver through a noisy channel. Since then, coding theory has found many important engineering applications. For example, coding theory has been used in designing reliable data storage systems, radio communication protocols, and in the emerging field of quantum computers. Coding theory has close ties with many areas in mathematics including linear algebra, commutative algebra, algebraic geometry, and combinatorics.

In this note we introduce a new package written for Macaulay2 [5] called *CodingTheory*. The goal of this package is to provide a range of functions for constructing linear and evaluation codes over finite fields, and for computing some of their main properties. To this aim, we define two objects, namely *linear code* and *evaluation code*. The package also includes implementation of functions for generating important families of linear codes like Hamming codes, cyclic codes, Reed-Solomon codes, Reed-Muller codes, Cartesian codes, monomial-Cartesian codes and toric codes. It also has functions for the syndrome decoding algorithm and locally recoverable codes.

The organization of this note is as follows. In Section 2 we describe different ways to define a linear code over a finite field using the *CodingTheory* package. In Section 3 we show how to compute the main parameters of a linear code: length, dimension, and minimum distance. We also illustrate how to compute some of the main algebraic objects associated with linear codes like generator and parity check matrices, dual codes, etc. In

2010 *Mathematics Subject Classification.* Primary 94B05; Secondary 13P25, 14G50, 11T71.

Key words and phrases. Linear codes, locally recoverable codes, Cartesian codes, evaluation codes, Hamming codes.

Section 4 we give a brief introduction to evaluation codes and describe some functions implemented to study these objects. In Section 5 we explain how to create some of the most studied families of linear codes, including Hamming codes, cyclic codes, Reed-Solomon codes, and Reed-Muller codes. Finally, we give instructions on how to create locally recoverable codes.

In this paper we do not attempt to fully explain every function distributed in this package. For a detailed explanation of all functions in the package, we refer to the Macaulay2 help page which can be accessed by running

```
viewHelp CodingTheory
```

More information about basics of coding theory can be found in [7, 10, 20]. Constructions of codes using commutative algebra as evaluation codes can be seen in [1, 4, 6, 8, 11, 12, 13, 19, 14, 16, 17]. Excellent references for theory of vanishing ideals and their properties are [3, 21].

2. Defining linear codes

Let $\mathbb{F}_q$ be a finite field with q elements. Mathematically, a *linear code* is defined as a vector subspace $C \subseteq \mathbb{F}_q^n$. For Macaulay2 (M2), a linear code is a submodule of $\mathbb{F}_q^n$. Assume $q = p^r$, where p is a prime number and r a positive integer. By definition, the dual code $C^\perp$ is the orthogonal complement of C in $\mathbb{F}_q^n$ with respect to the standard inner product. One can define C by specifying a list L of elements of $\mathbb{F}_q^n$ that span C or by giving a a *generator matrix* G whose rows form a basis of C. Alternatively, one can specify a list L_H of elements of $\mathbb{F}_q^n$ that span the dual code $C^\perp$ or a *parity check matrix* H whose columns form a basis of the dual code $C^\perp$. Below are the commands for the constructor linearCode to construct these equivalent instances of the LinearCode type:

- linearCode($\mathbb{F}_q$,L)
- linearCode($\mathbb{F}_q$,n,L)
- linearCode(G)
- linearCode($\mathbb{F}_q$,L_H, ParityCheck => true)
- linearCode(H, ParityCheck => true)
- linearCode(p,r,n,L)
- linearCode(p,r,n,L_H, ParityCheck => true)

Now, here is a more specific example of how to construct a simple linear code:

Example 2.1.

```
i2 : F = GF 4;
i3 : L = {{1,1,0,0},{0,0,1,1}};
i4 : C = linearCode(F,L)
o4 = Code with Generator Matrix: | 1 1 0 0 |
                                 | 0 0 1 1 |
o4 : LinearCode
```

One way to refer to a primitive element of a finite field is by specifying a symbol using the Variable option of the constructor GF.

Example 2.2.

```
i2 : F = GF(9,Variable => a);
i3 : LH = {{1,0,a,0,0},{0,a,a+1,1,0},{1,1,1,a,0}};
```

```
i4 : C = linearCode(F,LH,ParityCheck => true)
o4 = Code with Generator Matrix: | a-1 0 a+1 1 0 |
                                 | 0   0 0   0 1 |
o4 : LinearCode
```

To construct a linear code from a matrix, it is necessary to correctly specify the underlying field. This can be done by passing a field to the matrix constructor.

Example 2.3.

```
i2 : F = GF 4;

i3 : M = matrix(F, {{1,0,1,0},{0,1,1,1}});

             2       4
o3 : Matrix F  <--- F

i4 : C = linearCode(M)

o4 = Code with Generator Matrix: | 1 0 1 0 |
                                 | 0 1 1 1 |

o4 : LinearCode
```

3. Basic parameters linear codes

The *dimension* and the *length* are two of the basic parameters of a code $C \subseteq \mathbb{F}_q^n$. They are defined as the subspace dimension $\dim_{\mathbb{F}_q} C$ and the ambient space dimension n, respectively. A third basic parameter is the *minimum weight*, which is given by

$$\min\{\|c\| \colon c \in C, c \neq 0\},$$

where $\|c\|$ is the number of non-zero entries of c. The *rate* of C is defined as the rational number k/n. Some of the functions that can be used in M2 to compute basic parameters and algebraic objects associated with linear codes are the following:

- C.GeneratorMatrix
- C.Generators
- C.ParityCheckMatrix
- C.AmbientModule
- alphabet C
- field C
- informationRate C
- ambientSpace C
- length C
- dim C
- minimumWeight C
- codewords C
- dualCode C
- shorten(C, List)
- ==

Example 3.1.

```
i2: F = GF 4;
i3 : L = {{1,1,0,0},{0,0,1,1}};
i4 : C = linearCode(F,L)
o4 = Code with Generator Matrix: | 1 1 0 0 |
                                 | 0 0 1 1 |
o4 : LinearCode
i5 : length C
o5 = 4
```

```
i6 : dim C
o6 = 2
i7 : informationRate C
     1
o7 = -
     2
o7 : QQ
i8 : ambientSpace C
      4
o8 = F
o8 : F-module, free
i9 : alphabet C
o9 = {0, a, a + 1, 1}
o9 : List
i10 : minimumWeight C
o10 = 2
i11 : dualCode C
o11 = Code with Generator Matrix: | 1 1 0 0 |
                                  | 0 0 1 1 |

o11 : LinearCode
```

4. Evaluation codes

Let $\mathcal{X} = \{\mathbf{a}_1, \ldots, \mathbf{a}_n\}$ be a subset of an m-dimensional space $\mathbb{F}_q^m$. Consider a finite dimensional subspace $S \subset \mathbb{F}_q[X_1, \ldots, X_m]$ of the m-variate polynomial ring over $\mathbb{F}_q$.

The *evaluation map*

$$\mathrm{ev}_S \colon S \longrightarrow {\mathbb{F}_q}^{|\mathcal{X}|}, \qquad f \mapsto (f(\mathbf{a}_1), \ldots, f(\mathbf{a}_n)),$$

defines a linear map of $\mathbb{F}_q$-vector spaces. The image of ev_S in ${\mathbb{F}_q}^{|\mathcal{X}|}$, denoted by $C_{\mathcal{X}}(S)$, is the *evaluation code* on the set $\mathcal{X}$ corresponding to S. The *vanishing ideal* of $\mathcal{X}$, denoted by $I(\mathcal{X})$, is the ideal in S of all polynomials that vanish on $\mathcal{X}$. A key observation that allows the use of commutative algebra in studying evaluation codes is that the kernel of the evaluation map ev_S is precisely $S \cap I(\mathcal{X})$.

An evaluation code $C_{\mathcal{X}}(S)$ is defined in M2 as a separate type because there are more objects associated with it than with a linear code. For instance, the vanishing ideal associated to the set $\mathcal{X}$ plays an important role when finding and estimating parameters of the code, so it is convenient to be able to access it. Given an evaluation code C in M2, the object C.linearCode is a linear code in M2. The command evaluationCode$(\mathbb{F}_q, \mathcal{X}, L)$ defines an evaluation code where $\mathcal{X}$ is a list of elements in $\mathbb{F}_q^m$ and L is a list of polynomials that span S. In the case when polynomials in L are monomials, one may give the matrix of exponent vectors instead of L.

There are many construction of evaluation codes for specific choices of the set $\mathcal{X}$ and the subspace S. These include Reed-Muller codes, Cartesian and monomial Cartesian codes, toric codes, and evaluation codes from graphs. We refer to [1, 6, 8, 9, 11, 13, 14, 16] for details on how these codes are defined and what properties they have from coding theory, commutative algebra, and algebraic geometry perspectives. Some functions defined

in this package for various constructions of evaluation codes and associated algebraic objects are the following:

- evaluationCode($\mathbb{F}_q$,List,List)
- toricCode($\mathbb{F}_q$,Integer matrix)
- cartesianCode($\mathbb{F}_q$,List,List)
- orderCode($\mathbb{F}_q$,List,List,ZZ)
- evCodeGraph($\mathbb{F}_q$,Incident Matrix,integer)
- vNumber(Ideal)
- footPrint(Integer,Integer,Ideal)
- hYpFunction(Integer,Integer,Ideal)
- gMdFunction(Integer,Integer,Ideal)
- vasFunction(Integer,Integer,Ideal)

The mathematical definitions of the vNumber, the footprint function, the hyp function, the generalized footprint function and the Vasconcelos function can be found in [2]. The following example shows how to construct an evaluation code.

Example 4.1.

```
i2 : F=GF(4); R=F[x,y,z];
i4 : P={{0,0,0},{1,0,0},{0,1,0},{0,0,1},{1,1,1},{a,a,a}};
i5 : S={x+y+z,a+y*z^2,z^2,x+y+z+z^2};
i6 : C=evaluationCode(F,P,S)
o6 = Code with Generator Matrix: | 0 1 1 1 1   a   |
                                 | a a a a a+1 a+1 |
                                 | 0 0 0 1 1   a+1 |
                                 | 0 1 1 0 0   1   |
o6 : EvaluationCode
i7 : length C.LinearCode
o7 = 6
i8 : dim C.LinearCode
o8 = 3
i9 : C.Points
o9 = {{0, 0, 0}, {1, 0, 0}, {0, 1, 0}, {0, 0, 1}, {1, 1, 1}, {a, a, a}}
o9 : List
i10 : C.VanishingIdeal;
o10 = Ideal of R
```

5. Families of linear codes

We continue with the same notation: n represents the length of the code, k the dimension and q the size of the field. Some families of linear codes that have been implemented in this package are the following:

- HammingCode(q,integer)
- randLDPC(n, k, integer, integer)
- cyclicCode($\mathbb{F}_q$,polynomial, n)
- quasiCyclicCode($\mathbb{F}_q$,list)
- RSCode($\mathbb{F}_q$,List,integer)
- RMCode(q,integer,integer)
- zeroCode($\mathbb{F}_q$,n)
- universeCode($\mathbb{F}_q$,n)
- repetitionCode($\mathbb{F}_q$,n)
- zeroSumCode($\mathbb{F}_q$,n)
- random($\mathbb{F}_q$, n, k)

Mathematical definitions of the above families can be found in [7, 10, 20].

Example 5.1.

```
i2 : C = HammingCode(2,3)
o2 = Code with Generator Matrix: | 1 1 1 1 0 0 0 |
```

```
                                    | 0 1 1 0 1 0 0 |
                                    | 1 0 1 0 0 1 0 |
                                    | 1 1 0 0 0 0 1 |
o2 : LinearCode
i3 : F=GF(5); R=F[x]; g=x-1; C=cyclicCode(F,g,6)
Cyclic Code
o6 = Code with Generator Matrix: |-1  1  0  0  0  0 |
                                 | 0 -1  1  0  0  0 |
                                 | 0  0 -1  1  0  0 |
                                 | 0  0  0 -1  1  0 |
                                 | 0  0  0  0 -1  1 |
o6 : LinearCode
i7 : F = GF(5);
i8 : L = apply(toList(1..2),j-> apply(toList(1..4),i-> random(F)))
o8 = {{0, -2, 2, -1}, {-2, 1, 0, -1}}
o8 : List
i9 : C=quasiCyclicCode(L)
o9 = Code with Generator Matrix: |  0 -2  2 -1 |
                                 | -1  0 -2  2 |
                                 |  2 -1  0 -2 |
                                 | -2  2 -1  0 |
                                 | -2  1  0 -1 |
                                 | -1 -2  1  0 |
                                 |  0 -1 -2  1 |
                                 |  1  0 -1 -2 |
o9 : LinearCode
i7 : C=RSCode(GF 5,{1,2,3},3)
o7 = Code with Generator Matrix: | 1  1  1 |
                                 | 1  2 -2 |
                                 | 1 -1 -1 |
o7 : EvaluationCode
```

6. Applications of linear codes

A basic application of a linear code is *decoding*, which is used for reliable transmission of information trough a noisy channel. In a few words the idea is the following. Take a vector $c \in C$. Change the value of some of the entries of c to obtain a new vector v. Decoding the vector v means to recover the vector c when only v and C are given. Detailed treatment of decoding algorithms can be found in [7]. Another, more recent application of linear codes for found in distributed and cloud storage systems. The idea is to use *locally recoverable codes*, which are linear codes with the property that every entry can be recovered from a few other entries. For more information on locally recoverable codes see [18].

Some of the most important functions from this package that can be used for applications of coding theory are the following:

- syndromeDecode(C,v,minimumWeight(C))
- LocallyRecoverableCode(List,List,a polynomial)

Here is a small example.

Example 6.1.

```
i2 : C = HammingCode(2,3);
i3 : msg = matrix {{1,0,1,0}};
i4 : v = msg*(C.GeneratorMatrix)
o4 = | 0 1 0 1 0 1 0 |
i5 : err = matrix take(random entries basis source v, 1)
o5 = | 0 0 0 0 1 0 0 |
i6 : received = transpose(transpose (v+err))
o6 = | 0 1 0 1 1 1 0 |
i7 : transpose syndromeDecode(C, transpose recieved, 3)
o7 = | 0 1 0 1 0 1 0 |
```

Acknowledgments

We thank Federico Galetto, Courtney Gibbons, Hiram López, and Branden Stone for organizing the Macualay2 workshop at Cleveland State University, where this collaboration started. We want to give a special thanks to Branden Stone for helping us to develop the package during the workshop.

References

[1] C. Carvalho, V. G. Neumann and H. H. López, Projective Nested Cartesian Codes, Bull Braz Math Soc, New Series (2016).

[2] S. M. Cooper, A. Seceleanu, S. O. Tohăneanu, M. Vaz Pinto and R. H. Villarreal, Generalized minimum distance functions and algebraic invariants of Geramita ideals, Adv. in Appl. Math. **112** (2020), 101940.

[3] D. Cox, J. Little and D. O-Shea, *Ideals, Varieties, and Algorithms*, Springer-Verlag, 1992.

[4] L. Gold, J. Little, H. Schenck, Cayley-Bacharach and evaluation codes on complete intersections, J. Pure Appl. Algebra **196** (1) (2005) 91–99.

[5] D. R. Grayson and M. E. Stillman, Macaulay2, a software system for research in algebraic geometry. Available at `http://www.math.uiuc.edu/Macaulay2/`.

[6] J. Hansen, Toric Surfaces and Error–correcting Codes in Coding Theory, Cryptography, and Related Areas, Springer (2000), pp. 132-142.

[7] W. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*, Cambridge University Press, Cambridge, 2003.

[8] J. Little, H. Schenck, *Toric surface codes and Minkowski sums*, SIAM J. Discrete Math. **20** (2006), no. 4, 999–1014 (electronic).

[9] H. H. López, C. Rentería-Márquez and R. H. Villarreal, Affine Cartesian codes, Des. Codes Cryptogr. **71**(1) (2014) 5–19.

[10] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-correcting Codes*, North-Holland, 1977.

[11] J. Martínez-Bernal, Y. Pitones and R. H. Villarreal, Minimum distance functions of graded ideals and Reed-Muller-type codes, J. Pure Appl. Algebra 221 (2017), 251–275.

[12] J. Martínez-Bernal, Y. Pitones and R. H. Villarreal, Minimum distance functions of complete intersections, J. Algebra Appl. 17 (2018), no. 11, 1850204 (22 pages).

[13] C. Rentería, A. Simis and R. Villarreal, Algebraic methods for parameterized codes and invariants of vanishing ideals over finite fields, Finite Fields Appl. **17** (2011), no. 1, 81–104.

[14] D. Ruano, On the parameters of r-dimensional toric codes, Finite Fields and Their Applications **13** (2007), pp. 962–976.

[15] C. Shannon, A Mathematical Theory of Communication, The Bell System Technical Journal, **27** (1948), no. 3, 379–423.

[16] I. Soprunov, J. Soprunova, Toric surface codes and Minkowski length of polygons, SIAM J. Discrete Math. **23**, Issue 1, (2009) 384-400
[17] I. Soprunov, Toric complete intersection codes. Journal of Symbolic Computation. **50**, (2013) 374–385.
[18] I. Tamo and A. Barg, A Family of Optimal Locally Recoverable Codes, IEEE Transactions on Information Theory **60** (2014), no. 8, 4661–4676.
[19] C. Rentería and H. Tapia-Recillas, Reed-Muller codes: an ideal theory approach, Comm. Algebra **25** (1997), no. 2, 401–413.
[20] J. H. Van Lint, Introduction to coding theory, Third edition, Graduate Texts in Mathematics **86**, Springer-Verlag, Berlin, 1999.
[21] R. H. Villarreal, *Monomial Algebras*, Second edition, Monographs and Research Notes in Mathematics, Chapman and Hall/CRC, Boca Raton, FL, 2015.

(Taylor Ball) Department of Mathematics, University of Notre Dame, Notre Dame, IN USA
Email address: trball13@gmail.com

(Eduardo Camps) Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional, Mexico City, Mexico
Email address: camps@esfm.ipn.mx

(Henry Chimal-Dzul) Department of Mathematics and Center of Ring Theory and its Applications, Ohio University, Athens, OH USA
Email address: hc118813@ohio.edu

(Delio Jaramillo-Velez) Departamento de Matemáticas, Centro de Investigación y de Estudios Avanzados del IPN, Apartado Postal 14–740, 07000 Mexico City, D.F.
Email address: djaramillo@math.cinvestav.mx

(Hiram H. López) Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH USA
Email address: h.lopezvaldez@csuohio.edu

(Nathan Nichols) School of Mathematics, University of Minnesota Twin Cities, Minneapolis, MN USA
Email address: nathannichols454@gmail.com

(Matthew Perkins) Department of Mathematics, Cleveland State University, Cleveland, OH USA
Email address: m.r.perkins73@vikes.csuohio.edu

(Ivan Soprunov) Department of Mathematics, Cleveland State University, Cleveland, OH USA
Email address: i.soprunov@csuohio.edu

(German Vera-Martínez) Escuela Superior de Física y Matemáticas, Instituto Politécnico Nacional, Mexico City, Mexico
Email address: gveram1100@alumno.ipn.mx

(Gwyn Whieldon)
Email address: gwyn.whieldon@gmail.com