Journal of Software for Algebra and Geometry

Coding theory package for Macaulay2

Taylor Ball, Eduardo Camps, Henry Chimal-Dzul, Delio Jaramillo-Velez, Hiram López, Nathan Nichols, Matthew Perkins, Ivan Soprunov, German Vera-Martínez and Gwyn Whieldon

vol 11 2021

JSAG **11** (2021), 113–122
https://doi.org/10.2140/jsag.2021.11.113

The Journal of Software for
Algebra and Geometry

Coding theory package for Macaulay2

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

ABSTRACT: In this *Macaulay*2 package we implement a type of object called a `LinearCode`. 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 implement a type of object called an `EvaluationCode`, a construction which allows users 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 implement functions for the syndrome decoding algorithm and locally recoverable code construction, which are important tools in applications of linear codes.

1. INTRODUCTION. Coding theory has been extensively studied since 1948, when Claude Shannon [1948] proved in his seminal paper 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 the new [Macaulay2] package 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 implement two types of objects, `LinearCode` and `EvaluationCode`. The package also includes implementations 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 various ways to construct 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

MSC2020: primary 13P25, 94B05; secondary 11T71, 14G50.
Keywords: linear codes, locally recoverable codes, Cartesian codes, evaluation codes, Hamming codes.
CodingTheory version 1.0

compute some of the main algebraic objects associated with linear codes, such as generator and parity check matrices, dual codes, etc. In 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 *Macaulay*2 help page which can be accessed by running

```
i1: viewHelp CodingTheory
```

More information about basics of coding theory can be found in [Huffman and Pless 2003; MacWilliams and Sloane 1977; van Lint 1999]. Constructions of codes using commutative algebra as evaluation codes can be seen in [Carvalho et al. 2017; Gold et al. 2005; Hansen 2000; Little and Schenck 2006; Martínez-Bernal et al. 2017; 2018; Rentería-Márquez et al. 2011; Rentería and Tapia-Recillas 1997; Ruano 2007; Soprunov and Soprunova 2009; Soprunov 2013]. Excellent references for the theory of vanishing ideals and their properties are [Cox et al. 1992; Villarreal 2015].

2. CONSTRUCTING LINEAR CODES. Let $\mathbb{F}_q$ be a finite field with q elements. Mathematically, a *linear code* is defined as a vector subspace $C \subseteq \mathbb{F}_q^n$ and it is often specified by a *generator matrix*, which is a $k \times n$ matrix G with entries in $\mathbb{F}_q$ whose k rows form a basis for C. In *Macaulay*2, a linear code is defined as an $\mathbb{F}_q$-submodule of $\mathbb{F}_q^n$ using the constructor `linearCode`. This constructor is an instance of the `LinearCode` type. There are various ways to use the command `linearCode`. For example, one can use this command to construct a linear code $C \subseteq \mathbb{F}_q^n$ by specifying a generator matrix G of C (Example 2.1). Alternatively, one can use the command `linearCode` to construct a linear code $C \subseteq \mathbb{F}_q^n$ by indicating the finite field $\mathbb{F}_q$ and a list L of elements of $\mathbb{F}_q^n$ that span C (Example 2.2). More details and equivalent ways to use the constructor `linearCode` are given next.

Inputs:	Usage:
• `F = GF(q)`, a finite field with `q` elements	• `linearCode(G)`
• `n, r, p`, positive integers with `p` prime	• `linearCode(F,L)`
• `G`, a matrix with entries in `GF(q)`	• `linearCode(F,n,L)`
• `L`, a list of elements of $\texttt{GF(q)}^{\texttt{n}}$	• `linearCode(p,r,n,L)`

In the next examples we construct a simple binary linear code $C \subseteq \mathbb{F}_2^4$ with generator matrix $G = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ using equivalent versions of the command `linearCode`.

Example 2.1.

```
i1 : F = GF(2);
i2 : L = {{1,1,0,0},{0,0,1,1}};
i3 : G = matrix apply(L,codeword->apply(codeword,entry->sub(entry,F)));
```

```
i4 : C = linearCode(G);

i5 : C.GeneratorMatrix
o5 = | 1 1 0 0 |
     | 0 0 1 1 |
```

Note that in Example 2.1 it was necessary to coerce the entries of each vector in the list `L` into elements of `F = GF(2)`. An equivalent way to do this is to pass the field `GF(q)` to the `matrix` constructor or to use the constructor `linearCode(GF(q),L)`.

Example 2.2.

```
i1 : L = {{1,1,0,0},{0,0,1,1}};

i2 : C = linearCode(GF(2),L);

i3 : C.GeneratorMatrix
o3 = | 1 1 0 0 |
     | 0 0 1 1 |
```

The set $\mathbb{F}_q^*$ of nonzero elements of a finite field $\mathbb{F}_q$ is a multiplicative cyclic group ([Huffman and Pless 2003, Theorem 3.3.1]). A generator of $\mathbb{F}_q^*$ is called a *primitive element* of $\mathbb{F}_q$. 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`. In the next example we illustrate how to define a linear code $C \subseteq \mathbb{F}_{11}^{10}$ with generator matrix

$$G = \begin{pmatrix} 1 & a^1 & a^2 & \cdots & a^9 \\ 1 & a^2 & a^4 & \cdots & (a^2)^9 \\ 1 & a^3 & (a^3)^2 & \cdots & (a^3)^9 \\ 1 & a^4 & (a^4)^2 & \cdots & (a^4)^9 \end{pmatrix},$$

where a is a primitive element of $\mathbb{F}_{11}$. In *Macaulay*2, $a = 2$.

Example 2.3.

```
i1 : F = GF(11,Variable => a);

i2 : G = matrix table({1,2,3,4},
                      {1,a,a^2,a^3,a^4,a^5,a^6,a^7,a^8,a^9},(i,j)->j^i);

i3 : C = linearCode(G);

i4 : C.GeneratorMatrix
o4 = | 1 2  4  -3 5  -1 -2 -4 3  -5 |
     | 1 4  5  -2 3  1  4  5  -2 3  |
     | 1 -3 -2 -5 4  -1 3  2  5  -4 |
     | 1 5  3  4  -2 1  5  3  4  -2 |
```

In $\mathbb{F}_q^n$ there is a standard inner product defined for all $x = (x_1, \ldots, x_n)$ and $y = (y_1, \ldots, y_n)$ in $\mathbb{F}_q^n$ as $x \cdot y = x_1y_1 + \cdots + x_ny_n$. Given a linear code $C \subseteq \mathbb{F}_q^n$, the *dual* or *orthogonal code* of C with respect to this inner product is the linear code $C^\perp$ defined as

$$C^\perp = \{x \in \mathbb{F}_q^n \,:\, x \cdot c = 0 \text{ for all } c \in C\}.$$

A generator matrix of $C^\perp$ is called a *parity check matrix* for C. Since $(C^\perp)^\perp = C$, another common way to mathematically define C is by specifying one of its parity check matrices. By using the command `linearCode` and the `ParityCheck` option of this command, we can define a linear code $C \subseteq \mathbb{F}_q^n$ by either specifying a parity check matrix H of C or a list L of elements of $\mathbb{F}_q^n$ that span $C^\perp$.

In the next example, we define a linear code $C \subseteq \mathbb{F}_9^5$ whose dual code $C^\perp$ is generated by the set $\{(1, 0, a, 0, 0), (0, 1, a+1, 1, 0), (1, 1, 1, a, 0)\} \subseteq \mathbb{F}_9^5$, where a is a primitive element of $\mathbb{F}_9$. In *Macaulay*2, $a \in \mathbb{F}_9$ satisfies $a^2 = a + 1$.

Example 2.4.

```
i1 : F = GF(9,Variable => a);

i2 : L = {{1,0,a,0,0},{0,a,a+1,1,0},{1,1,1,a,0}};

i3 : C = linearCode(F,L,ParityCheck => true);

i4 : C.GeneratorMatrix
o4 = | a-1 0 a+1 1 0 |
     | 0   0 0   0 1 |

i5 : C.ParityCheckMatrix
o5 = | 1 0 a   0 0 |
     | 0 a a+1 1 0 |
     | 1 1 1   a 0 |
```

Although the dual code of a linear code can be constructed using the command `dualCode` implemented in the *CodingTheory* package, the `ParityCheck` option of the command `linearCode` also allows us to construct the dual of a linear code. In the next example we construct the dual of the linear code in Example 2.3.

Example 2.5.

```
i1 : F = GF(11,Variable => a);

i2 : G = matrix table({1,2,3,4},
                      {1,a,a^2,a^3,a^4,a^5,a^6,a^7,a^8,a^9},(i,j)->j^i);

i3 : D = linearCode(G,ParityCheck => true);

i4 : D.GeneratorMatrix
o4 = | 1  -3 5  3  1 0 0 0 0 0 |
     | -3 -1 4  -4 0 1 0 0 0 0 |
     | 4  -4 -3 5  0 0 1 0 0 0 |
     | -5 -3 4  4  0 0 0 1 0 0 |
     | -4 -4 -1 3  0 0 0 0 1 0 |
     | -3 5  3  1  0 0 0 0 0 1 |
```

3. BASIC PARAMETERS OF LINEAR CODES. The dimension k and length n are basic parameters of a linear code $C \subseteq \mathbb{F}_q^n$. The *information rate* of a linear code is defined as k/n. Another parameter of a linear code $C \subseteq \mathbb{F}_q^n$ is the *minimum distance*, which is defined as

$$w_H(C) := \min\{\|c\| : c \in C,\ c \neq 0\},$$

where $\|c\|$ denotes the Hamming weight of $c \in \mathbb{F}_q^n$, that is, $\|c\|$ is the number of nonzero entries in c. This parameter is important in determining the error-correcting capability of C; the higher the minimum distance, the more errors the code can detect and correct (see [Huffman and Pless 2003]). While the dimension and the length of linear code are computationally easy to determine, it is known that computing the minimum distance of an arbitrary linear code is an NP-hard problem [Vardy 1997]. For this task, the function `minimumWeight` is provided.

In *Macaulay*2, the space $\mathbb{F}_q^n$ has been called the *ambient module* or *ambient space* of C. The field $\mathbb{F}_q$ is called the *alphabet* or *field* of C. The elements of C are referred to as *codewords*. The input in all the

following commands implemented in the `CodingTheory` package is always a linear code `C`:

- `dim C`
- `field C`
- `C.AmbientModule`
- `length C`
- `codewords C`
- `minimumWeight C`
- `alphabet C`
- `C.Generators`
- `informationRate C`
- `ambientSpace C`
- `C.GeneratorMatrix`
- `C.ParityCheckMatrix`

Example 3.1.

```
i1 : L = {{1,1,0,0},{0,0,1,1}};

i2 : C = linearCode(GF(4),L);

i3 : dim C
o3 : 2

i4 : length C
o4 = 4

i5 : alphabet C
o5 = {0, a, a + 1, 1}

i6 : ambientSpace C
          4
o6 = (GF 4)

i7 : field C
o7 = GF 4

i8 : minimumWeight C
o8 = 2

i9 : codewords C
o9 = {{1, 1, 1, 1}, {1, 1, a, a}, {a, a, 1, 1}, {a, a, a, a},
     ------------------------------------------------------------------
      {a + 1, a + 1, a, a}, {a + 1, a + 1, 1, 1}, {1, 1, a + 1, a + 1},
     ------------------------------------------------------------------
      {a, a, a + 1, a + 1}, {a + 1, a + 1, a + 1, a + 1}, {1, 1, 0, 0},
     ------------------------------------------------------------------
      {0, 0, 1, 1}, {0, 0, a, a}, {a, a, 0, 0}, {a + 1, a + 1, 0, 0},
     ------------------------------------------------------------------
      {0, 0, a + 1, a + 1}, {0, 0, 0, 0}}
```

4. EVALUATION CODES. Let $P = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n\}$ be a subset of $\mathbb{F}_q^m$. Consider a finite-dimensional subspace $\mathcal{S} \subset \mathbb{F}_q[X_1, \ldots, X_m]$ of the ring of polynomials over $\mathbb{F}_q$ in m variables. The *evaluation map*

$$\mathrm{ev}_{\mathcal{S}} : \mathcal{S} \to {\mathbb{F}_q}^{|P|}, \quad f \mapsto (f(\boldsymbol{a}_1), \ldots, f(\boldsymbol{a}_n)),$$

defines a linear map of $\mathbb{F}_q$-vector spaces. The image of $\mathrm{ev}_{\mathcal{S}}$ in ${\mathbb{F}_q}^{|P|}$, denoted by $C_P(\mathcal{S})$, is the *evaluation code* on the set P corresponding to $\mathcal{S}$. The *vanishing ideal* of P, denoted by $I(P)$, is the ideal in $\mathbb{F}_q[X_1, \ldots, X_n]$ of all polynomials that vanish on P. 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 $\mathcal{S} \cap I(P)$.

An evaluation code $C_P(\mathcal{S})$ is implemented in *Macaulay*2 as an object of type `EvaluationCode`. However, the object `C.LinearCode` is a linear code in *Macaulay*2. This has been done in this way because there are more objects associated with an evaluation code than with a linear code. For instance, the vanishing ideal associated to the set P plays an important role when finding and estimating parameters of the code, so it is convenient to be able to access it.

There are many constructions of evaluation codes for specific choices of the set P and the subspace $\mathcal{S}$. These include Reed–Muller codes, Cartesian, monomial Cartesian codes, toric codes, and evaluation codes from graphs. We refer to [Carvalho et al. 2017; Hansen 2000; Little and Schenck 2006; López et al. 2014; Martínez-Bernal et al. 2017; Rentería-Márquez et al. 2011; Ruano 2007; Soprunov and Soprunova 2009] for details on how these codes are defined and what properties they have from coding theory, commutative algebra, and algebraic geometry perspectives.

Some functions implemented in the *CodingTheory* package for various constructions of evaluation codes and associated algebraic objects are the following:

Inputs:

- `I`, an ideal
- `d,r`, positive integers
- `F=GF(q)`, a finite field with `q` elements
- `P`, a list of points in $\mathtt{F}^{\mathtt{m}}$
- `S`, a list of polynomials in `m` variables
- `M`, an integer matrix
- `L`, a list of subsets of `F`
- `v`, a list of `m` positive integers
- `MI`, an incident matrix of a graph

Usage:

- `evaluationCode(F,P,S)`
- `toricCode(F,M)`
- `cartesianCode(F,L,d)`
- `orderCode(F,P,v,d)`
- `evCodeGraph(F,MI,S)`
- `vNumber(I)`
- `footPrint(d,r,I)`
- `hYp(d,r,I)`
- `genMinDisIdeal(d,r,I)`
- `vasconcelosDegree(d,r,I)`

The input `S` above is a list of polynomials that span the subspace

$$\mathcal{S} \subset \mathbb{F}_q[X_1, \ldots, X_m].$$

The mathematical definitions of the functions in the second column of the previous list can be found in [Cooper et al. 2020]. The following example shows how to construct an evaluation code in *Macaulay*2 using the *CodingTheory* package.

Example 4.1.

```
i1 : F=GF(4,Variable=>a); R=F[x,y,z];

i3 : P={{0,0,0},{1,0,0},{0,1,0},{0,0,1},{1,1,1},{a,a,a}};

i4 : S={x+y+z,a+y*z^2,z^2,x+y+z+z^2};

i5 : C=evaluationCode(F,P,S);

i6 : (C.LinearCode).GeneratorMatrix
o6 = | 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   |
```

```
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}}

i10 : C.VanishingIdeal
                          2    2                      2    2
o10 = ideal (x*z + y*z, y  + z  + y + z, x*y + y*z, x  + z  + x + z,
      ------------------------------------------------------------
       3          2
     z  + (a +1)z  + a*z)
```

5. FAMILIES OF LINEAR CODES. Various important families of linear codes have been defined through the development of coding theory. These include Hamming codes, cyclic codes, Reed–Solomon codes, Reed–Muller codes, etc. The mathematical definitions of these and many other families of codes can be found in [Huffman and Pless 2003; MacWilliams and Sloane 1977; van Lint 1999].

The following functions have been implemented in the *CodingTheory* package to construct some of these important families of codes.

Inputs:

- `n, k, d, m, s`, positive integers
- `F = GF(q)`, a finite field with `q` elements
- `g(x)`, a polynomial in `F[x]`
- `E`, a list of elements of `F`
- `L`, a list of vectors in $\mathtt{F}^{\mathtt{m}}$

Usage:

- `hammingCode(q,s)`
- `cyclicCode(F,g(x),n)`
- `zeroCode(F,n)`
- `reedSolomonCode(F,E,s)`
- `repetitionCode(F,n)`
- `universeCode(F,n)`
- `reedMullerCode(q,m,d)`
- `randomCode(F,n,k)`
- `zeroSumCode(F,n)`

Example 5.1.

```
i1 : C = hammingCode(2,3);

i2 : C.GeneratorMatrix
o2 = | 1 1 1 1 0 0 0 |
     | 1 1 0 0 1 0 0 |
     | 0 1 1 0 0 1 0 |
     | 1 0 1 0 0 0 1 |

i3 : F = GF(5); R = F[x]; g = x-1; C = cyclicCode(F,g,6);

i7 : C.GeneratorMatrix
o7 = | -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 |

i8 : C = reedSolomonCode(GF(5),{1,2,3},3);

i9 : (C.LinearCode).GeneratorMatrix
o9 = | 1 1  1  |
     | 1 2  -2 |
     | 1 -1 -1 |
```

6. APPLICATIONS OF LINEAR CODES. An important aspect in coding theory is *decoding*, which is used when information is transmitted 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 [Huffman and Pless 2003]. Another, more recent application of coding theory is 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 [Tamo and Barg 2014].

Some of the most important functions implemented in the *CodingTheory* package that can be used for applications of coding theory are the following:

Inputs:

- `C`, a linear code over `GF(q)`
- `v`, a vector in the ambient space of `C`
- `{q,n,k,r}`, where `q` is a prime power, and `n`, `k`, and `r` are positive integers
- `L`, a list of pairwise disjoint subsets of `GF(q)`

Usage:

- `syndromeDecode(C,v,minimumWeight(C))`
- `LocallyRecoverableCode({q,n,k,r},L,a polynomial)`

Example 6.1.

```
i1 : C = hammingCode(2,3);

i2 : msg = matrix {{1,0,1,0}};

i3 : v = msg*(C.GeneratorMatrix)
o3 = | 0 1 0 1 0 1 0 |

i4 : err = matrix take(random entries basis source v, 1)
o4 = | 0 0 0 0 1 0 0 |

i5 : received = transpose(transpose (v+err))
o5 = | 0 1 0 1 1 1 0 |

i6 : transpose syndromeDecode(C, transpose recieved, 3)
o6 = | 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 special thanks to Branden Stone for helping us to develop the package during the workshop. Finally, we are grateful to the anonymous referees for their valuable comments and suggestions. This work was partially supported by the NSF grant DMS-2003883.

SUPPLEMENT. The online supplement contains version 1.0 of *CodingTheory*.

REFERENCES.

[Carvalho et al. 2017] C. Carvalho, V. G. L. Neumann, and H. H. López, "Projective nested cartesian codes", *Bull. Braz. Math. Soc. (N.S.)* **48**:2 (2017), 283–302. MR Zbl

[Cooper et al. 2020] S. M. Cooper, A. Seceleanu, c. O. Tohăneanu, M. V. Pinto, and R. H. Villarreal, "Generalized minimum distance functions and algebraic invariants of Geramita ideals", *Adv. in Appl. Math.* **112** (2020), art. id. 101940. MR

[Cox et al. 1992] D. Cox, J. Little, and D. O'Shea, *Ideals, varieties, and algorithms*, Springer, 1992. MR Zbl

[Gold et al. 2005] L. Gold, J. Little, and H. Schenck, "Cayley–Bacharach and evaluation codes on complete intersections", *J. Pure Appl. Algebra* **196**:1 (2005), 91–99. MR Zbl

[Hansen 2000] J. P. Hansen, "Toric surfaces and error-correcting codes", pp. 132–142 in *Coding theory, cryptography and related areas* (Guanajuato (1998)), edited by J. Buchmann et al., Springer, 2000. MR Zbl

[Huffman and Pless 2003] W. C. Huffman and V. Pless, *Fundamentals of error-correcting codes*, Cambridge University Press, 2003. MR Zbl

[van Lint 1999] J. H. van Lint, *Introduction to coding theory*, 3rd ed., Graduate Texts in Mathematics **86**, Springer, 1999. MR Zbl

[Little and Schenck 2006] J. Little and H. Schenck, "Toric surface codes and Minkowski sums", *SIAM J. Discrete Math.* **20**:4 (2006), 999–1014. MR Zbl

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

[Macaulay2] D. R. Grayson and M. E. Stillman, "Macaulay2, a software system for research in algebraic geometry", available at https://math.uiuc.edu/Macaulay2/.

[MacWilliams and Sloane 1977] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*, *I*, vol. 16, North-Holland Mathematical Library, North-Holland Publishing Co., Amsterdam, 1977. MR

[Martínez-Bernal et al. 2017] 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**:2 (2017), 251–275. MR Zbl

[Martínez-Bernal et al. 2018] J. Martínez-Bernal, Y. Pitones, and R. H. Villarreal, "Minimum distance functions of complete intersections", *J. Algebra Appl.* **17**:11 (2018), art. id. 1850204. MR Zbl

[Rentería and Tapia-Recillas 1997] C. Rentería and H. Tapia-Recillas, "Reed–Muller codes: an ideal theory approach", *Comm. Algebra* **25**:2 (1997), 401–413. MR Zbl

[Rentería-Márquez et al. 2011] C. Rentería-Márquez, A. Simis, and R. H. Villarreal, "Algebraic methods for parameterized codes and invariants of vanishing ideals over finite fields", *Finite Fields Appl.* **17**:1 (2011), 81–104. MR Zbl

[Ruano 2007] D. Ruano, "On the parameters of r-dimensional toric codes", *Finite Fields Appl.* **13**:4 (2007), 962–976. MR Zbl

[Shannon 1948] C. E. Shannon, "A mathematical theory of communication", *Bell System Tech. J.* **27** (1948), 379–423. MR Zbl

[Soprunov 2013] I. Soprunov, "Toric complete intersection codes", *J. Symbolic Comput.* **50** (2013), 374–385. MR Zbl

[Soprunov and Soprunova 2009] I. Soprunov and J. Soprunova, "Toric surface codes and Minkowski length of polygons", *SIAM J. Discrete Math.* **23**:1 (2009), 384–400. MR Zbl

[Tamo and Barg 2014] I. Tamo and A. Barg, "A family of optimal locally recoverable codes", *IEEE Trans. Inform. Theory* **60**:8 (2014), 4661–4676. MR

[Vardy 1997] A. Vardy, "The intractability of computing the minimum distance of a code", *IEEE Trans. Inform. Theory* **43**:6 (1997), 1757–1766. MR Zbl

[Villarreal 2015] R. H. Villarreal, *Monomial algebras*, 2nd ed., CRC Press, Boca Raton, FL, 2015. MR

RECEIVED: 13 Jul 2020 REVISED: 16 Jul 2021 ACCEPTED: 10 Aug 2021

TAYLOR BALL:

trball13@gmail.com
University of Notre Dame, Notre Dame, IN, United States

EDUARDO CAMPS:

camps@esfm.ipn.mx
Escuela Superior de Física y Matemáticas, Instituto Politecnico Nacional, Zacatenco, Mexico City, Mexico

HENRY CHIMAL-DZUL:

hc118813@ohio.edu
Department of Mathematics and Center of Ring Theory and its Applications, Ohio University, Athens, OH, United States

DELIO JARAMILLO-VELEZ:

djaramillo@math.cinvestav.mx
Departamento de Matemáticas, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Mexico City, Mexico

HIRAM LÓPEZ:

h.lopezvaldez@csuohio.edu
Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH, United States

NATHAN NICHOLS:

nathannichols454@gmail.com
School of Mathematics, University of Minnesota Twin Cities, Minneapolis, MN, United States

MATTHEW PERKINS:

m.r.perkins73@vikes.csuohio.edu
Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH, United States

IVAN SOPRUNOV:

i.soprunov@csuohio.edu
Department of Mathematics and Statistics, Cleveland State University, Cleveland, OH, United States

GERMAN VERA-MARTÍNEZ:

gveram1100@alumno.ipn.mx
Escuela Superior de Física y Matemáticas, Instituto Politecnico Nacional, Zacatenco, Mexico City, Mexico

GWYN WHIELDON:

gwyn.whieldon@gmail.com
Frederick, MD, United States

Journal of Software for Algebra and Geometry vol 11, no 1, 2021