$\mathbb{Z}_2\mathbb{Z}_4$-ADDITIVE CODES

A MAGMA Package*

by

Joaquim Borges, Cristina Fernández, Bernat Gastón,
Jaume Pujol, Josep Rifà and Mercè Villanueva

Combinatoric, Coding and Security Group (CCSG)

Universitat Autònoma de Barcelona

Version 4.0

Barcelona
December, 2017

*This work has been partially supported by the Spanish MICINN under Grants PCI2006-A7-0616, MTM2006-03250, MTM2009-08435, TIN2010-17358 and TIN2013-40524-P; by the Catalan AGAUR under Grants 2009SGR1224 and 2014SGR-691; and by the UAB under Grant PNL2006-13.

Contents

Chapter 1

Preface

The *Combinatoric, Coding and Security Group* (CCSG) is a research group in the Department of Information and Communications Engineering (DEIC) at the Universitat Autònoma de Barcelona (UAB).

The research group CCSG has been uninterruptedly working since 1987 in several projects and research activities on Information Theory, Communications, Coding Theory, Source Coding, Cryptography, Electronic Voting, Network Coding, etc. The members of the group have been producing mainly results on optimal coding. Specifically, the research has been focused on uniformly-packed codes; perfect codes in the Hamming space; perfect codes in distance-regular graphs; the classification of optimal codes of a given length; and codes which are close to optimal codes by some properties, for example, Reed-Muller codes, Preparata codes, Kerdock codes and Hadamard codes.

Part of the research developed by CCSG deals with $\mathbb{Z}_2\mathbb{Z}_4$-linear codes. There are no symbolic software to work with these codes, so the members of CCSG have been developing this new package that supports the basic facilities for $\mathbb{Z}_2\mathbb{Z}_4$-additive codes. Specifically, this MAGMA package generalizes most of the known functions for codes over the ring $\mathbb{Z}_4$, which are subgroups of $\mathbb{Z}_4^n$, to $\mathbb{Z}_2\mathbb{Z}_4$-additive codes, which are subgroups of $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^\delta$, maintaining all the functionality for codes over $\mathbb{Z}_4$ and adding new functions which, not only generalize the previous ones, but introduce new variants when it is needed. A beta version of this new package for $\mathbb{Z}_2\mathbb{Z}_4$-additive codes and this manual with the description of all functions can be downloaded from the web page `http://ccsg.uab.cat`. For any comment or further information about this package, you can send an e-mail to *support-ccsg@deic.uab.cat*.

The authors would like to thank Lorena Ronquillo, Jaume Pernas, Roger Ten-Valls, and Cristina Diéguez for their contributions developing some parts of this MAGMA package.

Chapter 2

$\mathbb{Z}_2\mathbb{Z}_4$-additive codes

2.1 Introduction

MAGMA currently supports the basic facilities for linear codes over integer residue rings and galois rings [7, Chapter 119], including cyclic codes and the complete weight enumerator calculation. Moreover, some functions are available for the special case of linear codes over $\mathbb{Z}_4$ (also called $\mathbb{Z}_4$-codes or quaternary linear codes), which are subgroups of $\mathbb{Z}_4^n$.

Linear codes over $\mathbb{Z}_4$ have been studied and became significant since, after applying the Gray map from $\mathbb{Z}_4$ to binary pairs, we obtain binary nonlinear codes better than any known binary linear code with the same parameters. More specifically, Hammons et. al. [11] show how to construct well known binary nonlinear codes like Kerdock codes and Delsarte-Goethals codes by applying the Gray map to linear codes over $\mathbb{Z}_4$. Further, they solve an old open problem on coding theory about that the weight enumerators of the nonlinear Kerdock and Preparata codes satisfy the MacWilliams identities. Later, several other binary nonlinear codes constructed using the Gray map on linear codes over $\mathbb{Z}_4$, and with the same parameters as some well known families of binary linear codes (for example, extended Hamming codes, Hadamard codes, and Reed-Muller codes) have been studied and classified [5, 13, 15, 18, 19, 22].

MAGMA also supports functions for additive codes over a finite field, which are a generalization of the linear codes over a finite field [7, Chapter 120] in a Hamming scheme. According to a more general definition of additive codes given by Delsarte in 1973 [8, 9], the additive codes are subgroups of the underlying abelian group in a translation association scheme. In the special case of a binary Hamming scheme, that is, when the underlying abelian group is of size $2^{n_{bin}}$, the only structures for the abelian group are those of the form

$\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, with $\alpha + 2\beta = n_{bin}$. Therefore, the codes that are subgroups of $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ are also called additive codes. In order to distinguish them from the additive codes over a finite field, we call them $\mathbb{Z}_2\mathbb{Z}_4$-*additive codes*.

$\mathbb{Z}_2\mathbb{Z}_4$-additive codes are subgroups of $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, so they can be seen as a generalization of binary and quaternary linear codes. Note that when $\alpha = 0$, these codes are the linear codes over $\mathbb{Z}_4$, and when $\beta = 0$, they are the binary linear codes. As for linear codes over $\mathbb{Z}_4$, after applying the Gray map to the $\mathbb{Z}_4$ coordinates of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, we obtain binary nonlinear codes. The binary image of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code under the extended Gray map is called $\mathbb{Z}_2\mathbb{Z}_4$-linear code. There are $\mathbb{Z}_2\mathbb{Z}_4$-linear codes in several important classes of binary codes. For example, $\mathbb{Z}_2\mathbb{Z}_4$-linear perfect single error-correcting codes (or 1-perfect codes) are found in [21] and fully characterized in [4]. Also, in subsequent papers [5, 17, 16], $\mathbb{Z}_2\mathbb{Z}_4$-linear extended 1-perfect and Hadamard codes are studied and classified. Therefore, $\mathbb{Z}_2\mathbb{Z}_4$-additive codes have allowed to classify more binary nonlinear codes, giving them an structure as linear codes over $\mathbb{Z}_4$ or $\mathbb{Z}_2\mathbb{Z}_4$-additive codes.

Part of the research developed by the Combinatorics, Coding and Security Group (CCSG) deals with linear codes over $\mathbb{Z}_4$, as well as $\mathbb{Z}_2\mathbb{Z}_4$-additive codes. Since there are no symbolic software to work with $\mathbb{Z}_2\mathbb{Z}_4$-additive codes, the members of CCSG have been developing this new package that supports the basic facilities for $\mathbb{Z}_2\mathbb{Z}_4$-additive codes. Specifically, this MAGMA package generalizes most of the known functions for codes over the ring $\mathbb{Z}_4$ to $\mathbb{Z}_2\mathbb{Z}_4$-additive codes, maintaining all the functionality for codes over $\mathbb{Z}_4$ and adding new functions which, not only generalize the previous ones, but introduce new variants when it is needed.

Let C be an $\mathbb{Z}_2\mathbb{Z}_4$-additive code. Since C is a subgroup of $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, it is isomorphic to an abelian structure $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^\delta$. Therefore, we have that $|C| = 2^\gamma 4^\delta$ and the number of codewords of order two in C is $2^{\gamma+\delta}$. Let X (respectively Y) be the set of $\mathbb{Z}_2$ (respectively $\mathbb{Z}_4$) coordinate positions, so $|X| = \alpha$ and $|Y| = \beta$. Unless otherwise stated, the set X corresponds to the first α coordinates and Y corresponds to the last β coordinates. Call C_X (respectively C_Y) the punctured code of C by deleting the coordinates outside X (respectively Y). Let C_b be the $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode of C which contains all order two codewords and let κ be the dimension of $(C_b)_X$, which is a binary linear code. For the case $\alpha = 0$, we will write $\kappa = 0$. Considering all these parameters, we will say that C is of type $(\alpha, \beta; \gamma, \delta; \kappa)$. Note that the quaternary linear code C_Y is of type $(0, \beta; \gamma_Y, \delta; 0)$, where $0 \leq \gamma_Y \leq \gamma$, and the binary linear code C_X is a code of length α and dimension γ_X, or equivalently of type $(\alpha, 0; \gamma_X, 0; \gamma_X)$, where $\kappa \leq \gamma_X \leq \gamma$.

Moreover, since the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes are subgroups of $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ and have an abelian structure $\mathbb{Z}_2^{\gamma} \times \mathbb{Z}_4^{\delta}$, every codeword is uniquely expressible in the form

$$\sum_{i=1}^{\gamma} \lambda_i u_i + \sum_{j=\gamma+1}^{\gamma+\delta} \mu_j v_j,$$

where $\lambda_i \in \mathbb{Z}_2$ for $1 \leq i \leq \gamma$, $\mu_j \in \mathbb{Z}_4$ for $\gamma + 1 \leq j \leq \gamma + \delta$ and u_i, v_j are vectors in $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ of order two and order four, respectively. The vectors u_i, v_j give us a generator matrix of size $(\gamma + \delta) \times (\alpha + \beta)$ of the form

$$\left(\begin{array}{c|c} B_1 & 2B_3 \\ \hline B_2 & Q \end{array}\right), \tag{2.1}$$

where B_1, B_2, B_3 are matrices over $\mathbb{Z}_2$ of size $\gamma \times \alpha$, $\delta \times \alpha$ and $\gamma \times \beta$, respectively; and Q is a matrix over $\mathbb{Z}_4$ of size $\delta \times \beta$ with quaternary row vectors of order four.

In this chapter, the term "code" will refer to a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, unless otherwise specified. In order to be able to use the functions for quaternary linear codes, from now on the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes will be represented as quaternary linear codes changing the ones by twos in the first α binary coordinates, so they will be subgroups of $\mathbb{Z}_4^{\alpha+\beta}$. However, these codes are not equivalent to the quaternary linear codes, since the inner product defined in $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ gives us that the dual code of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code is not equivalent to the dual code of the corresponding quaternary linear code.

Finally, as a general reference on $\mathbb{Z}_2\mathbb{Z}_4$-additive codes, the reader is referred to [6], where most of the concepts on $\mathbb{Z}_2\mathbb{Z}_4$-additive codes implemented with the following functions are described in detail.

Note: All the names of the functions in this MAGMA package contain Z2Z4 to distinguish them from the ones that have the same name and are for linear codes over finite rings. In this manual, the functions that do not contain Z2Z4 are not implemented here because they are functions that already exist for codes over finite rings and can be also applied to $\mathbb{Z}_2\mathbb{Z}_4$-additive codes.

Note: Since the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes will be represented as quaternary linear codes, it is necessary to remember that for modules defined over rings with zero divisors, it is not possible to talk about the concept of dimension (the modules are not free) [7, p. 5521]. However, in MAGMA each code over such a ring has a unique generator matrix corresponding to the Howell form [12, 23]. The number of rows k in this generator matrix will be called the *pseudo-dimension* of the code. It should be noted that this pseudo-dimension

is not invariant between equivalent codes, and so does not provide structural information like the dimension of a code over a finite field.

Note: In order to use the functions in Subsection 2.5.4, it is necessary to use MAGMA version 2.22 or later. It is also possible installing the package "Codes over Z4", which can be downloaded from the web page `http://ccsg.uab.cat`. The functions in this package expand the current functionality for codes over $\mathbb{Z}_4$ in MAGMA. Specifically, there are functions to construct some families of codes over $\mathbb{Z}_4$, efficient functions for computing the rank and dimension of the kernel of any code over $\mathbb{Z}_4$, as well as general functions for computing coset representatives for a subcode in a code over $\mathbb{Z}_4$. There are also functions for computing the permutation automorphism group for Hadamard and extended perfect codes over $\mathbb{Z}_4$, and their orders. Functions related to the information space, information sets, syndrome space and coset leaders for codes over $\mathbb{Z}_4$ can also be found. Finally, functions to decode codes over $\mathbb{Z}_4$ by using different methods are also provided.

2.2 Construction of $\mathbb{Z}_2\mathbb{Z}_4$-Additive Codes

2.2.1 General $\mathbb{Z}_2\mathbb{Z}_4$-Additive Codes

`Z2Z4AdditiveCode(L : parameters)`

`Alpha`	RNGINTELT	*Default*: 0
`OverZ2`	BOOLELT	*Default*: `false`

Create a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C given as a record with two fields:

- `Alpha`: The length of the binary part of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code.
- `Code`: The quaternary linear code equal to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code, where the ones in the first `Alpha` coordinates are represented by twos.

The corresponding quaternary linear code of C is obtained using the function `LinearCode()` as a subspace of $V = \mathbb{Z}_4^n$ generated by L, where:

1. L is a sequence of elements of V,
2. or, L is a subspace of V,
3. or, L is a $m \times n$ matrix A over the ring $\mathbb{Z}_4$,
4. or, L is a quaternary linear code,
5. or, L is a binary linear code.

The parameter `Alpha` specifies the length of the binary part of the code. When L is a binary linear code, the default value is the length of the code, otherwise the default value is 0.

The parameter `OverZ2` specifies whether the first `Alpha` coordinates of each element in L are represented as elements in $\mathbb{Z}_2 = \{0, 1\}$ or, otherwise, they are represented as elements in $\{0, 2\} \subset \mathbb{Z}_4$. The default value is `false`. If `OverZ2` is `true` and `Alpha` > 0, then the first `Alpha` coordinates of each element in L must be 0 or 1. Otherwise, if `OverZ2` is `false` and `Alpha` > 0, then the first `Alpha` coordinates of each element in L must be 0 or 2.

Example H2E1

We define a $\mathbb{Z}_2\mathbb{Z}_4$-additive code by giving a sequence of elements of V, or equivalently a $m \times n$ matrix over $\mathbb{Z}_4$, and the length of the binary part of the code.

```
> Z4 := IntegerRing(4);

> V  := RSpace(Z4, 5);
> C1 := Z2Z4AdditiveCode([V![2,2,1,1,3],V![0,2,1,2,1],
                          V![2,2,2,2,2],V![2,0,1,1,1]] : Alpha:=2);
> C1;
rec<Z2Z4Code |
    Code := ((5, 4^2 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 0 2]
    [0 2 0 0 2]
    [0 0 1 0 3]
    [0 0 0 1 0],
    Alpha := 2
    >

> A  := Matrix(Z4, [[2,2,1,1,3],[0,2,1,2,1],[2,2,2,2,2],[2,0,1,1,1]]);
> C2 := Z2Z4AdditiveCode(A : Alpha:=2);
> C2;
rec<Z2Z4Code |
    Code := ((5, 4^2 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 0 2]
    [0 2 0 0 2]
    [0 0 1 0 3]
    [0 0 0 1 0],
    Alpha := 2
    >

> Z2Z4Equal(C1, C2);
true
```

Alternatively, we also define the same $\mathbb{Z}_2\mathbb{Z}_4$-additive codes by giving the first `Alpha` coordinates of each vector as elements in $\{0,1\}$ instead of in $\{0,2\}$, changing the twos by ones and adding the parameter `OverZ2`.

```
> C1b := Z2Z4AdditiveCode([V![1,1,1,1,3],V![0,1,1,2,1],V![1,1,2,2,2],
                           V![1,0,1,1,1]] : Alpha:=2, OverZ2:=true);

> Ab  := Matrix(Z4, [[1,1,1,1,3],[0,1,1,2,1],[1,1,2,2,2],[1,0,1,1,1]]);
> C2b := Z2Z4AdditiveCode(Ab : Alpha:=2, OverZ2:=true);

> Z2Z4Equal(C1, C1b) and Z2Z4Equal(C2, C2b);
true
```

Example H2E2

Any quaternary linear code and any binary linear code is also a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, since the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes can be seen as a generalization of the quaternary linear codes, when $\alpha = 0$, and a generalization of the binary linear codes, when $\beta = 0$.

```
> B := RandomLinearCode(GF(2), 7, 3);
> B;
[7, 3, 1] Linear Code over GF(2)
Generator matrix:
[1 0 0 1 0 0 1]
[0 1 1 1 0 1 1]
[0 0 0 0 1 0 0]
> B_add := Z2Z4AdditiveCode(B);
> B_add;
rec<Z2Z4Code |
    Code := ((7, 4^0 2^3)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2 0 0 2]
    [0 2 2 2 0 2 2]
    [0 0 0 0 2 0 0],
    Alpha := 7
    >

> Q := RandomLinearCode(IntegerRing(4), 5, 2);
> Q;
((5, 4^1 2^1)) Linear Code over IntegerRing(4)
Generator matrix:
[2 0 0 2 0]
[0 1 0 1 1]
> Q_add := Z2Z4AdditiveCode(Q);
> Q_add;
rec<Z2Z4Code |
    Code := ((5, 4^1 2^1)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2 0]
```

```
[0 1 0 1 1],
Alpha := 0
>
```

2.2.2 Some Trivial $\mathbb{Z}_2\mathbb{Z}_4$-Additive Codes

`Z2Z4AdditiveZeroCode(`α`,`β`)`

Given two non-negative integers α and β, return the $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; 0, 0; 0)$ consisting of only the zero codeword.

`Z2Z4AdditiveRepetitionCode(`α`,`β`)`

Given two non-negative integers α and β, return the $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; 1, 0; 1)$ generated by the vector $(1, \ldots, 1|2, \ldots, 2)$.

`Z2Z4AdditiveUniverseCode(`α`,`β`)`

Given two non-negative integers α and β, return the generic $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \alpha, \beta; \alpha)$ consisting of all possible codewords.

`RandomZ2Z4AdditiveCode(`α`,`β`,`γ`,`δ`,`κ`)`

Given two, three, four or five non-negative integers α, β, γ, δ, κ, return a random $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \gamma, \delta; \kappa)$. Note that there exists a $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ if and only if $\alpha, \beta, \gamma, \delta, \kappa \geq 0$, $\alpha + \beta > 0$, $\kappa \leq min(\alpha, \gamma)$ and $0 < \delta + \gamma \leq \beta + \kappa$. When there are less than five integers, they correspond to the first ones in the above order and the rest are computed randomly.

Example H2E3

The zero $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; 0, 0; 0)$ is contained in every $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \gamma, \delta; \kappa)$, and similarly every $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ is contained in the universe $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \alpha, \beta; \alpha)$.

```
> a := 2; b := 3;
> U := Z2Z4AdditiveUniverseCode(a, b);
> Z := Z2Z4AdditiveZeroCode(a, b);
> R := RandomZ2Z4AdditiveCode(a, b);
> Z2Z4Subset(Z, R) and Z2Z4Subset(R, U);
true
```

2.2.3 Families of $\mathbb{Z}_2\mathbb{Z}_4$-Additive Codes

`Z2Z4HadamardCode(`δ`, m :` *parameters*`)`

`OverZ4` BoolElt *Default*: `false`

The function returns a $\mathbb{Z}_2\mathbb{Z}_4$-additive Hadamard code of type $(\alpha, \beta; \gamma, \delta; \kappa)$. The parameter `OverZ4` specifies whether the code is over $\mathbb{Z}_4$, that is $\alpha = 0$, or, otherwise, $\alpha \neq 0$. The default value is `false`. When `OverZ4` is `true`, given an integer $m \geq 1$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, return a $\mathbb{Z}_2\mathbb{Z}_4$-additive Hadamard code of type $(0, \beta; \gamma, \delta; 0)$, where $\beta = 2^{m-1}$ and $\gamma = m+1-2\delta$. When `OverZ4` is `false`, given an integer $m \geq 1$ and an integer δ such that $0 \leq \delta \leq \lfloor m/2 \rfloor$, return a $\mathbb{Z}_2\mathbb{Z}_4$-additive Hadamard code of type $(\alpha, \beta; \gamma, \delta; \gamma)$, where $\alpha = 2^{m-\delta}$, $\beta = 2^{m-1} - 2^{m-\delta-1}$ and $\gamma = m + 1 - 2\delta$. Moreover, return a generator matrix with $\gamma + \delta$ rows constructed in a recursive way, where the ones in the first α coordinates are represented by twos.

A $\mathbb{Z}_2\mathbb{Z}_4$-additive Hadamard code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ is a $\mathbb{Z}_2\mathbb{Z}_4$-additive code such that, after the Gray map, give a binary (not necessarily linear) code with the same parameters as the binary Hadamard code of length 2^m.

`Z2Z4ExtendedPerfectCode(`δ`, m :` *parameters*`)`

`OverZ4` BoolElt *Default*: `false`

The function returns a $\mathbb{Z}_2\mathbb{Z}_4$-additive extended perfect code of type $(\alpha, \beta; \gamma, \delta; \kappa)$. The parameter `OverZ4` specifies whether the code is over $\mathbb{Z}_4$, that is $\alpha = 0$, or, otherwise, $\alpha \neq 0$. The default value is `false`. When `OverZ4` is `true`, given an integer $m \geq 1$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, return a $\mathbb{Z}_2\mathbb{Z}_4$-additive extended perfect code, such that its additive dual code is of type $(0, \beta; \gamma, \delta; 0)$, where $\beta = 2^{m-1}$ and $\gamma = m + 1 - 2\delta$. When `OverZ4` is `false`, given an integer $m \geq 1$ and an integer δ such that $0 \leq \delta \leq \lfloor m/2 \rfloor$, return a $\mathbb{Z}_2\mathbb{Z}_4$-additive extended perfect code, such that its additive dual code is of type $(\alpha, \beta; \gamma, \delta; \gamma)$, where $\alpha = 2^{m-\delta}$, $\beta = 2^{m-1} - 2^{m-\delta-1}$ and $\gamma = m + 1 - 2\delta$. Moreover, return a parity check matrix with $\gamma + \delta$ rows constructed in a recursive way, where the ones in the first α coordinates are represented by twos.

A $\mathbb{Z}_2\mathbb{Z}_4$-additive extended perfect code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ is a $\mathbb{Z}_2\mathbb{Z}_4$-additive code such that, after the Gray map, give a binary (not necessarily linear) code with the same parameters as the binary extended perfect code of length 2^m.

Example H2E4

Some $\mathbb{Z}_2\mathbb{Z}_4$-additive codes whose images under the Gray map are binary codes having the same parameters as some well-known families of binary linear codes are explored.

First, a $\mathbb{Z}_2\mathbb{Z}_4$-additive Hadamard code C with $\alpha = 0$ and another one with $\alpha \neq 0$ are defined. The matrix Gc is the quaternary matrix used to generate C where the ones in the first α coordinates are represented by twos.

```
> C, Gc := Z2Z4HadamardCode(3, 5 : OverZ4 := true);
> C`Alpha;
0
> (C`Code eq HadamardCodeZ4(3, 5)) and (C`Code eq LinearCode(Gc));
true
> HasZ2Z4LinearGrayMapImage(C);
false
> n := Z2Z4BinaryLength(C);
32
> (Z2Z4MinimumLeeDistance(C) eq n/2) and (Z2Z4Cardinal(C) eq 2*n);
true

> C, Gc := Z2Z4HadamardCode(2, 5);
> C`Alpha;
8
> Z2Z4Equal(C, Z2Z4AdditiveCode(LinearCode(Gc) : Alpha := C`Alpha));
true
> HasZ2Z4LinearGrayMapImage(C);
false
> n := Z2Z4BinaryLength(C);
32
> (Z2Z4MinimumLeeDistance(C) eq n/2) and (Z2Z4Cardinal(C) eq 2*n);
true
```

Then, a $\mathbb{Z}_2\mathbb{Z}_4$-additive extended perfect code D with $\alpha \neq 0$ is defined such that its additive dual code is of type $(2^{5-2}, 2^{5-1} - 2^{5-2-1}; 5 + 1 - 2 \cdot 2, 3; 2^{5-2}) = (8, 12; 2, 2; 2)$. The matrix Gd is the quaternary matrix which is used to generate the additive dual code of D where the ones in the first α coordinates are represented by twos.

```
> C, Gc := Z2Z4HadamardCode(2, 5);
> D, Gd := Z2Z4ExtendedPerfectCode(2, 5);
> Z2Z4Equal(D, Z2Z4Dual(C));
true
> Gc eq Gd;
true
> n := Z2Z4BinaryLength(D);
> Z2Z4Cardinal(D) eq 2^(n-1-Log(2,n));
true
> Z2Z4MinimumLeeDistance(D) eq 4;
true
```

2.3 Invariants of a $\mathbb{Z}_2\mathbb{Z}_4$-Additive Code

2.3.1 Basic Numerical Invariants

`Z2Z4Length(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the length $n = \alpha + \beta$, and the sequence $[\alpha, \beta]$ with the number of coordinates over $\mathbb{Z}_2$ and the number of coordinates over $\mathbb{Z}_4$, respectively.

`Z2Z4BinaryLength(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the binary length $n_{bin} = \alpha + 2\beta$ of C, which corresponds to the length of the binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map defined in Subsection 2.5.1.

`Z2Z4PseudoDimension(C)`
`Z2Z4NumberOfGenerators(C)`
`Z2Z4Ngens(C)`

The number of generators (which equals the pseudo-dimension k) of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C as a quaternary linear code.

`Z2Z4Type(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return a sequence with the parameters $[\alpha, \beta, \gamma, \delta, \kappa]$, that is, the type of the code.

`Z2Z4Cardinal(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the number of codewords belonging to C, that is $2^\gamma 4^\delta$.

`Z2Z4InformationRate(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the information rate of C, that is the ratio $(\gamma + 2\delta)/(\alpha + 2\beta)$.

Example H2E5

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, we compute its type, the number of codewords and its information rate.

```
> V := RSpace(IntegerRing(4), 6);
> C := Z2Z4AdditiveCode([V![2,2,1,1,3,1],V![2,2,2,2,2,2],V![0,0,1,1,1,3]] :
                        Alpha:=2);
> Z2Z4Type(C);
[ 2, 4, 2, 1, 1 ]
> Z2Z4Cardinal(C);
16
> Z2Z4InformationRate(C);
0.40000000000000000000000000000000
```

2.3.2 The Code Space

`Z2Z4Generic(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the generic $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \alpha, \beta; \alpha)$ in which C is contained.

`Z2Z4Name(C,i)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ and a positive integer i, return the i-th generator of C as a quaternary linear code, where the ones in the first α coordinates are represented by twos.

`Z2Z4Set(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the set containing all its codewords, where the ones in the first α coordinates are represented by twos.

`Z2Z4Generators(C)`

The generators for the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ as a quaternary linear code, returned as a set of elements of C`$Code$, where the ones in the first α coordinates are represented by twos.

`Z2Z4GeneratorMatrix(C)`

The unique generator matrix for the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ as a quaternary linear code, corresponding to the Howell form (see [12, 23]). The ones in the first α coordinates are represented by twos.

`Z2Z4OrderTwoGenerators(C)`

The γ generators of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, returned as a set of elements of C`$Code$, where the ones in the first α coordinates are represented by twos.

`Z2Z4OrderFourGenerators(C)`

The δ generators of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, returned as a set of elements of C`$Code$, where the ones in the first α coordinates are represented by twos.

`Z2Z4OrderTwoSubcodeGenerators(C)`

The $\gamma + \delta$ generators of the $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode C_b which contains all order two codewords of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, returned as a set of elements of C_b`$Code$, where the ones in the first α coordinates are represented by twos.

`Z2Z4MinRowsGeneratorMatrix(C)`

A generator matrix of the form (2.1) for the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, with the minimum number of rows, that is with $\gamma + \delta$ rows: γ rows of order two and δ rows of order four. It also returns the parameters γ and δ. The ones in the first α coordinates are represented by twos.

2.3.3 Conversion Functions

The $\mathbb{Z}_2\mathbb{Z}_4$-additive codes are internally represented as quaternary linear codes, with their first α coordinates represented as elements in $\{0, 2\}$ over $\mathbb{Z}_4$ instead of elements in $\{0, 1\}$ over $\mathbb{Z}_2$. This section provides functions to convert the outputs given as a sequence, set or matrix of vectors over $\mathbb{Z}_4^{\alpha+\beta}$ to a sequence or a set of tuples in the cartesian product set $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$, changing the twos over $\mathbb{Z}_4$ in the first α coordinates to ones over $\mathbb{Z}_2$.

`FromZ4toZ2Z4(L,`α`)`

Given a sequence L of vectors over $\mathbb{Z}_4^{\alpha+\beta}$ and an integer α, return the conversion of these vectors to a sequence of tuples in the cartesian product set $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$, changing the twos over $\mathbb{Z}_4$ in the first α coordinates to ones over $\mathbb{Z}_2$. It is checked whether the elements in the first α coordinates are in $\{0, 2\}$.

`FromZ4toZ2Z4(S,`α`)`

Given a set S of vectors over $\mathbb{Z}_4^{\alpha+\beta}$ and an integer α, return the conversion of these vectors to a set of tuples in the cartesian product set $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$, changing the twos over $\mathbb{Z}_4$ in the first α coordinates to ones over $\mathbb{Z}_2$. It is checked whether the elements in the first α coordinates are in $\{0, 2\}$.

`FromZ4toZ2Z4(M,`α`)`

Given a matrix M over $\mathbb{Z}_4$ with $\alpha + \beta$ columns and an integer α, return the conversion from M to a sequence of tuples in the cartesian product set $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$, changing the twos over $\mathbb{Z}_4$ in the first α coordinates to ones over $\mathbb{Z}_2$. It is checked whether the elements in the first α coordinates are in $\{0, 2\}$.

2.3.4 The Dual Space

The inner product in $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ is defined uniquely by

$$\langle u, v \rangle = 2(\sum_{i=1}^{\alpha} u_i v_i) + \sum_{j=\alpha+1}^{\alpha+\beta} u_j v_j \in \mathbb{Z}_4, \tag{2.2}$$

where $u, v \in \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$. Note that when $\alpha = 0$ the inner product is the usual one for vectors over $\mathbb{Z}_4$, and when $\beta = 0$ it is twice the usual one for vectors over $\mathbb{Z}_2$.

The *additive orthogonal code* of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, denoted by $C^\perp$, is defined in the standard way

$$C^\perp = \{v \in \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta \mid \langle u, v\rangle = 0 \text{ for all } u \in C\}.$$

We will also call $C^\perp$ the *additive dual code* of C. Note that the additive dual code $C^\perp$ is also a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, that is a subgroup of $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$.

`Z2Z4Dual(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the additive dual code of C.

`Z2Z4DualType(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return a sequence with the parameters $[\alpha, \beta, \gamma', \delta', \kappa']$, that is the type of the additive dual code of C. If C is of type $(\alpha, \beta; \gamma, \delta; \kappa)$, then its additive dual code is of type $(\alpha, \beta; \alpha + \gamma - 2\kappa, \beta - \gamma - \delta + \kappa; \alpha - \kappa)$.

`Z2Z4ParityCheckMatrix(C)`

The unique parity-check matrix for the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, that is the unique generator matrix for the additive dual code of C as a quaternary linear code, corresponding to the Howell form (see [12, 23]). The ones in the first α coordinates are represented by twos.

`Z2Z4MinRowsParityCheckMatrix(C)`

A parity-check matrix of the form (2.1) for the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, that is a generator matrix for the additive dual code of C, with the minimum number of rows, that is with $\gamma + \delta$ rows: γ rows of order two and δ rows of order four. The ones in the first α coordinates are represented by twos.

2.4 The Standard Form

A $\mathbb{Z}_2\mathbb{Z}_4$-additive code is in *standard form* if its generator matrix is of the form:

$$\left(\begin{array}{cc|ccc} I_\kappa & T_b & 2T_2 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & 2T_1 & 2I_{\gamma-\kappa} & \mathbf{0} \\ \hline \mathbf{0} & S_b & S_q & R & I_\delta \end{array}\right),$$

where T_b, T_1, T_2, R, S_b are matrices over $\mathbb{Z}_2$ and S_q is a matrix over $\mathbb{Z}_4$. Any $\mathbb{Z}_2\mathbb{Z}_4$-additive code C is permutation-equivalent to a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C_{SF}

which is in standard form (see [6]). Two $\mathbb{Z}_2\mathbb{Z}_4$-additive codes which differ only by a coordinate permutation are said to be *permutation-equivalent*.

`Z2Z4StandardForm(C)`

Given any $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return a permutation-equivalent $\mathbb{Z}_2\mathbb{Z}_4$-additive code C_{SF} in standard form, together with the corresponding isomorphism from C to C_{SF}, the generator matrix in standard form, and the coordinate permutation used to define the isomorphism.

`IsZ2Z4StandardFormMatrix(M,[`α`,`β`,`γ`,`δ`,`κ`])`

Return `true` if and only if the matrix M over $\mathbb{Z}_4$, where the ones in the first α coordinates are represented by twos, is a generator matrix of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ in standard form.

Example H2E6

We compute the standard form of a certain $\mathbb{Z}_2\mathbb{Z}_4$-additive code. Note that the number of rows in the general matrix of the standard code may be less than the number of rows in the matrix of the original code.

```
> V := RSpace(IntegerRing(4), 8);
> C := Z2Z4AdditiveCode([V![0,0,2,1,1,1,2,1],V![0,0,0,2,0,1,3,1],
                         V![0,0,0,0,2,1,3,1],V![0,2,2,0,0,2,2,0],
                         V![2,2,0,0,1,2,0,2]] : Alpha:=3);
> C;
rec<recformat<Code: CodeLinRng, Alpha: RngIntElt> |
    Code := ((8, 4^3 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 1 0 0 1 2]
    [0 2 0 1 1 0 1 2]
    [0 0 2 1 1 0 1 0]
    [0 0 0 2 0 0 0 0]
    [0 0 0 0 2 0 0 0]
    [0 0 0 0 0 1 1 1]
    [0 0 0 0 0 0 2 0],
    Alpha := 3>

> C_SF, f, G_SF, p := Z2Z4StandardForm(C);
> G_SF;
[2 0 2 2 0 0 0 0]
[0 0 0 0 2 0 0 0]
[0 2 2 2 0 1 0 0]
[0 2 0 2 1 0 1 0]
[0 2 0 3 1 0 0 1]
> IsZ2Z4StandardFormMatrix(G_SF, Z2Z4Type(C));
true
> Z2Z4Equal(C, C_SF);
false
```

```
> Z2Z4Equal(C_SF, Z2Z4AdditiveCode(G_SF : Alpha := C`Alpha));
true
> C`Code^p eq C_SF`Code;
true
> {f(c) : c in C`Code} eq Set(C_SF`Code);
true
> {c^p : c in C`Code} eq Set(C_SF`Code);
true
```

2.5 Derived Binary and Quaternary Codes

2.5.1 The Gray Map

The $\mathbb{Z}_2\mathbb{Z}_4$-additive codes are subgroups C of $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ and the corresponding binary codes are $C_{bin} = \Phi(C)$, where Φ is the following extension of the usual Gray map: $\Phi : \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta \longrightarrow \mathbb{Z}_2^{n_{bin}}$, where $n_{bin} = \alpha + 2\beta$, given by

$$\Phi(x, y) = (x, \phi(y_1), \ldots, \phi(y_\beta)) \quad \forall x \in \mathbb{Z}_2^\alpha,\ \forall y = (y_1, \ldots, y_\beta) \in \mathbb{Z}_4^\beta;$$

where $\phi : \mathbb{Z}_4 \longrightarrow \mathbb{Z}_2^2$ is the usual Gray map, that is,

$$\phi(0) = (0, 0),\ \phi(1) = (0, 1),\ \phi(2) = (1, 1),\ \phi(3) = (1, 0).$$

This Gray map is an isometry which transforms Lee distances defined in codes C over $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ to Hamming distances defined in the binary codes $C_{bin} = \Phi(C)$ (see Subsection 2.6.4). Note that the length of the binary code C_{bin} is $n_{bin} = \alpha + 2\beta$.

`Z2Z4GrayMap(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the Gray map for C. This is the map Φ from C to $\Phi(C)$, as defined above.

`Z2Z4GrayMapImage(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the image of C under the Gray map as a sequence of vectors in $\mathbb{Z}_2^{\alpha+2\beta}$. As the resulting image may not be a binary linear code, a sequence of vectors is returned rather than a code.

`HasZ2Z4LinearGrayMapImage(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return `true` if and only if the image of C under the Gray map is a binary linear code. If so, the function also returns the image B as a binary linear code, together with the bijection $\Phi : C \to B$.

Example H2E7

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, we compute its image under the Gray map. This image is not always a binary linear code.

```
> V := RSpace(IntegerRing(4), 4);
> C := Z2Z4AdditiveCode([V![2,0,0,2],V![0,1,0,3],V![0,0,1,3]] : Alpha:=1);
> C;
rec<Z2Z4Code |
    Code := ((4, 4^2 2^1)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2]
    [0 1 0 3]
    [0 0 1 3],
    Alpha := 1
    >
> HasZ2Z4LinearGrayMapImage(C);
false
> Cb := Z2Z4GrayMapImage(C);
> #Cb;
32

> V := RSpace(IntegerRing(4), 6);
> D := Z2Z4AdditiveCode([V![2,2,1,1,3,1],V![2,2,2,2,2,2],V![0,0,1,1,1,3]] :
                          Alpha:=2);
> D;
rec<Z2Z4Code |
    Code := ((6, 4^1 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 2 0 0 0 0]
    [0 0 1 1 1 3]
    [0 0 0 0 2 2],
    Alpha := 2
    >
> f := Z2Z4GrayMap(D);
> D`Code.1;
(2 2 0 0 0 0)
> f(D`Code.1);
(1 1 0 0 0 0 0 0 0 0)
> D`Code.2;
(0 0 1 1 1 3)
> f(D`Code.2);
(0 0 0 1 0 1 0 1 1 0)
> D`Code.3;
(0 0 0 0 2 2)
> f(D`Code.3);
(0 0 0 0 0 0 1 1 1 1)
> l, B, f := HasZ2Z4LinearGrayMapImage(D);
> l;
true
```

```
> B;
(10, 16, 2) Linear Code over IntegerRing(2)
Generator matrix:
[1 1 0 0 0 0 0 0 0 0]
[0 0 1 0 1 0 0 1 1 0]
[0 0 0 1 0 1 0 1 1 0]
[0 0 0 0 0 0 1 1 1 1]
> f(D`Code.1) in B;
true
> f(D`Code.2) in B;
true
> f(D`Code.3) in B;
true
> Length(B) eq Z2Z4BinaryLength(D);
true
```

2.5.2 From Linear to $\mathbb{Z}_2\mathbb{Z}_4$-Additive Codes

`Z2Z4AdditiveFromBinaryLinearCode(C)`

Given a binary linear code C of length n, return the same code as a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, so with $\alpha = n$ and $\beta = 0$.

`Z2Z4AdditiveFromQuaternaryLinearCode(C)`

Given a quaternary linear code C of length n, return the same code as a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, so with $\alpha = 0$ and $\beta = n$.

Example H2E8

We convert a binary linear code and a quaternary linear code to a $\mathbb{Z}_2\mathbb{Z}_4$-additive code.

```
> B := RandomLinearCode(GF(2), 7, 3);
> Q := RandomLinearCode(IntegerRing(4), 5, 2);

> B_add := Z2Z4AdditiveFromBinaryLinearCode(B);
> B_add;
rec<Z2Z4Code |
    Code := ((7, 4^0 2^3)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 2 0 0 2 2]
    [0 2 0 0 0 0 0]
    [0 0 0 2 0 2 0],
    Alpha := 7
    >

> Q_add := Z2Z4AdditiveFromQuaternaryLinearCode(Q);
> Q_add;
rec<Z2Z4Code |
```

```
    Code := ((5, 4^2 2^0)) Linear Code over IntegerRing(4)
    Generator matrix:
    [1 0 0 3 3]
    [0 1 0 3 3],
    Alpha := 0
    >

> IsZ2Z4AdditiveCode(B_add) and IsZ2Z4AdditiveCode(Q_add);
true
```

2.5.3 Subcodes C_X and C_Y

`Z2Z4LinearBinaryCode(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the binary linear code C_X of length α which is the punctured code of C by deleting the coordinates outside X, where X is the set of the first α coordinates.

`Z2Z4LinearQuaternaryCode(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the quaternary linear code C_Y of length β which is the punctured code of C by deleting the coordinates outside Y, where Y is the set of the last β coordinates.

Example H2E9

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, we compute the corresponding binary linear code C_X and the corresponding quaternary linear code C_Y.

```
> V  := RSpace(IntegerRing(4), 4);
> C  := Z2Z4AdditiveCode([V![2,0,0,2],V![0,1,0,3],V![0,0,1,3]] : Alpha:=1);
> CX := Z2Z4LinearBinaryCode(C);
> CX;
(1, 2, 1) Cyclic Linear Code over IntegerRing(2)
Generator matrix:
[1]
> CY := Z2Z4LinearQuaternaryCode(C);
> CY;
((3, 4^2 2^1)) Cyclic Linear Code over IntegerRing(4)
Generator matrix:
[1 0 1]
[0 1 1]
[0 0 2]
> _, n := Z2Z4Length(C);
> alpha := n[1]; beta := n[2];
> alpha; beta;
1
3
> Length(CX) eq alpha and Length(CY) eq beta;
true
```

2.5.4 Span and Kernel Codes

`Z2Z4SpanZ2Code(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return $S_C = \Phi^{-1}(S_{bin})$ as a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, and $S_{bin} = \langle C_{bin} \rangle$, that is the linear span of C_{bin}, as a binary linear code of length $\alpha + 2\beta$, where $C_{bin} = \Phi(C)$ and Φ is the Gray map.

`Z2Z4KernelZ2Code(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return its kernel K_C as a $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode of C, and $K_{bin} = \Phi(K_C)$ as a binary linear subcode of C_{bin} of length $\alpha + 2\beta$, where $C_{bin} = \Phi(C)$ and Φ is the Gray map.

The kernel K_C contains the codewords v such that $2v * u \in C$ for all $u \in C$, where $*$ denotes the component-wise product. Equivalently, the kernel $K_{bin} = \Phi(K_C)$ contains the codewords $c \in C_{bin}$ such that $c + C_{bin} = C_{bin}$, where $C_{bin} = \Phi(C)$ and Φ is the Gray map.

`Z2Z4KernelCosetRepresentatives(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the coset representatives $[c_1, \ldots, c_t]$ as a sequence of codewords of C, such that $C = K_C \cup \bigcup_{i=1}^{t} \big(K_C + c_i\big)$, where K_C is the kernel of C as a $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode. It also returns the coset representatives of the corresponding binary code $C_{bin} = \Phi(C)$ as a sequence of binary codewords $[\Phi(c_1), \ldots, \Phi(c_t)]$, such that $C_{bin} = K_{bin} \cup \bigcup_{i=1}^{t} \big(K_{bin} + \Phi(c_i)\big)$, where $K_{bin} = \Phi(K_C)$ and Φ is the Gray map.

`Z2Z4DimensionOfSpanZ2(C)`

`Z2Z4RankZ2(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the dimension of the linear span of C_{bin}, that is, the dimension of $\langle C_{bin} \rangle$, where $C_{bin} = \Phi(C)$ and Φ is the Gray map.

`Z2Z4DimensionOfKernelZ2(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the dimension of the Gray map image of its $\mathbb{Z}_2\mathbb{Z}_4$-additive kernel K_C, that is the dimension of $K_{bin} = \Phi(K_C)$, where Φ is the Gray map. Note that K_{bin} is always a binary linear code.

Example H2E10

```
> M1 := Matrix(Integers(4), [[0,2,2,1,3,1],[0,0,0,2,2,2]]);
> C1 := Z2Z4AdditiveCode(M1 : Alpha:=3);
```

```
> HasZ2Z4LinearGrayMapImage(C1);
true [9, 2, 5] Linear Code over GF(2)
Generator matrix:
[0 1 1 0 1 1 0 0 1]
[0 0 0 1 1 1 1 1 1]
Mapping from: ((6, 4^1 2^0)) Linear Code over IntegerRing(4) to [9, 2, 5] Linear
Code over GF(2) given by a rule
> Z2Z4DimensionOfSpanZ2(C1) eq Z2Z4DimensionOfKernelZ2(C1);
true
> S, Sb := Z2Z4SpanZ2Code(C1);
> K, Kb := Z2Z4KernelZ2Code(C1);
> Z2Z4Equal(K, C1) and Z2Z4Equal(C1, S);
true
> Kb eq Sb;
true

> M2 := Matrix(Integers(4),[[2,0,0,0,0,0,0,0,0,0,0,2],
                            [0,2,0,0,0,0,0,0,0,0,0,2],
                            [0,0,2,0,0,0,0,0,0,0,0,2],
                            [0,0,0,2,0,0,0,0,0,0,0,2],
                            [0,0,0,0,2,0,0,0,0,0,0,0],
                            [0,0,0,0,0,2,0,0,0,0,0,2],
                            [0,0,0,0,0,0,1,0,0,0,0,1],
                            [0,0,0,0,0,0,0,1,0,1,0,0],
                            [0,0,0,0,0,0,0,0,1,0,0,1],
                            [0,0,0,0,0,0,0,0,0,2,0,2],
                            [0,0,0,0,0,0,0,0,0,0,1,2]]);
> C2 := Z2Z4AdditiveCode(M2 : Alpha:=6);
> HasZ2Z4LinearGrayMapImage(C2);
false
> Z2Z4DimensionOfSpanZ2(C2) eq Z2Z4DimensionOfKernelZ2(C2);
false
> S, Sb := Z2Z4SpanZ2Code(C2);
> K, Kb := Z2Z4KernelZ2Code(C2);
> Z2Z4Subset(K, C2) and Z2Z4Subset(C2, S);
true
> Kb subset Sb;
true
```

2.6 Operations on Codewords

A $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ is represented in MAGMA as a record with two fields: `Alpha`, the length of the binary part of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code; and `Code`, the quaternary linear code (or equivalently, the subspace of $V = \mathbb{Z}_4^{\alpha+\beta}$) equal to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code, where the ones in the first α coordinates are represented by twos (see Subsection 2.2.1).

In this section, in order to use the same functions as for quaternary linear codes, notice that we will write `C‘Code` instead of `C`. Also notice that all codewords, which are elements in $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, are represented as elements in $V = \mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos.

2.6.1 Construction of a Codeword

`C‘Code ! [` $a_1, \ldots, a_n$ `]`

`elt< C‘Code |` $a_1, \ldots, a_n$ `>`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, which is represented in MAGMA as a subspace of $V = \mathbb{Z}_4^n$ ($n = \alpha + \beta$), and elements $a_1, \ldots, a_n$ belonging to $\mathbb{Z}_4$, construct the codeword $(a_1, \ldots, a_n)$ of C. It is checked whether the vector $(a_1, \ldots, a_n)$ is an element of $C‘Code$.

`C‘Code ! u`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, which is represented in MAGMA as a subspace of $V = \mathbb{Z}_4^{\alpha+\beta}$, and an element u belonging to V, create the codeword of C corresponding to u. The function will fail if u does not belong to C.

`C‘Code ! 0`

The zero word of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C.

`Z2Z4Random(C)`

A random codeword of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, which is a vector in $V = \mathbb{Z}_4^{\alpha+\beta}$ where the ones in the first α coordinates are represented by twos.

Example H2E11

We create some elements of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code.

```
> V := RSpace(IntegerRing(4), 4);
> C := Z2Z4AdditiveCode([V![2,0,0,2],V![0,1,0,3],V![0,0,1,3]] : Alpha:=1);
> C‘Code![2,0,0,2];
(2 0 0 2)
> elt<C‘Code | 2,1,1,0>;
(2 1 1 0)
> Z2Z4Random(C);
(2 0 1 1)
```

If the given vector does not lie in the given $\mathbb{Z}_2\mathbb{Z}_4$-additive code, then an error will result.

```
> C‘Code![2,1,0,0];

>> C‘Code![2,1,0,0];
          ^
Runtime error in '!': Result is not in the given structure
```

2.6.2 Operations on Codewords and Vectors

`u + v`

Sum of the codewords u and v, where u and v belong to the same $\mathbb{Z}_2\mathbb{Z}_4$-additive code C.

`- u`

Additive inverse of the codeword u belonging to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C.

`u - v`

Difference of the codewords u and v, where u and v belong to the same $\mathbb{Z}_2\mathbb{Z}_4$-additive code C.

`a * u`

Given an element a belonging to $\mathbb{Z}_4$, and a codeword u belonging to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the codeword $a \cdot u$.

`Normalize(u)`

Given an element u belonging to a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, which is represented in MAGMA as a subspace of $V = \mathbb{Z}_4^{\alpha+\beta}$, return the normalization of u, which is the unique vector v such that $v = a \cdot u$ for some scalar a in $\mathbb{Z}_4$ such that the first non-zero entry of v is the canonical associate in $\mathbb{Z}_4$ of the first non-zero entry of u (v is zero if u is zero).

`Z2Z4InnerProduct(u, v, α)`

The inner product $\langle u, v \rangle$ in $\mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ defined in Subsection 2.3.4. The vectors u and v are represented as vectors in $V = \mathbb{Z}_4^{\alpha+\beta}$, that is the parent vector space of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C.

`Support(w)`

Given a codeword w belonging to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, which is represented in MAGMA as a subspace of $V = \mathbb{Z}_4^{\alpha+\beta}$, return its support as a subset of the integer set $\{1, \ldots, \alpha + \beta\}$. The support of w consists of the coordinates at which w has non-zero entries.

`Z2Z4Coordinates(C, u)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C and a codeword u of C, return the coordinates of u with respect to C. The coordinates of u are returned as a sequence $Q = [a_1, \ldots, a_k]$ of elements from $\mathbb{Z}_4$ so that $u = a_1 \cdot C$ `` `Code.1 + \cdots + a_k \cdot C `` `Code.k$.

`Rotate(u, k)`

Given a vector u, return the vector obtained from u by cyclically shifting its components to the right by k coordinate positions.

`Rotate(~u, k)`

Given a vector u, destructively rotate u by k coordinate positions.

`Parent(w)`

Given a codeword w belonging to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the ambient space $V = \mathbb{Z}_4^{\alpha+\beta}$ of C.

Example H2E12

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, we explore various operations on its codewords.

```
> V := RSpace(IntegerRing(4), 4);
> C := Z2Z4AdditiveCode([V![2,0,0,2],V![0,1,0,3],V![0,0,1,3]] : Alpha:=1);
> u := C‘Code.1;
> v := C‘Code.2;
> u; v;
(2 0 0 2)
(0 1 0 3)
> u+v;
(2 1 0 1)
> 2*v;
(0 2 0 2)
> u+v in C‘Code;
true
> Z2Z4InnerProduct(u, v, 1);
2
> Support(u);
{ 1, 4 }
> Z2Z4Coordinates(C, u+2*v);
[ 1, 2, 0 ]
> Parent(u);
Full RSpace of degree 4 over IntegerRing(4)
```

2.6.3 Operations on Vectors and Matrices

`Z2Z4Mult(u, A, α)`

Given a vector u belonging to $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ represented as an element in $\mathbb{Z}_4^{\alpha+\beta}$, where the ones in the first α coordinates are represented by twos; and a matrix A over $\mathbb{Z}_4$ having $\alpha + \beta$ rows and the entries in the first α rows in $\{0, 2\}$; return the vector $u * A$.

`Z2Z4Mult(A, B, α)`

Given a $m \times (\alpha+\beta)$ matrix A where the rows belong to $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ represented as $\mathbb{Z}_4^{\alpha+\beta}$, that is, having the entries in the first α columns in $\{0, 2\}$; and a $(\alpha + \beta) \times p$ matrix B over $\mathbb{Z}_4$ having the entries in the first α rows in $\{0, 2\}$, return the $m \times p$ matrix $A * B$ having in the i-th row the vector `Z2Z4Mult(`u_i`, B)`, where u_i is the i-th row of A.

Example H2E13

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, an information vector is encoded by using the generator matrix, it is checked that the syndrome of the obtained codeword is zero, then some errors are introduced to give the vector u, and finally the syndrome of u is computed by using the parity check matrix.

```
> V := RSpace(IntegerRing(4), 6);
> C := Z2Z4AdditiveCode([V![2,0,0,2,0,2],
                         V![0,2,0,3,1,1],
                         V![0,0,1,3,0,2]] : Alpha:=2);

> G := Z2Z4MinRowsGeneratorMatrix(C);
> i := RSpace(IntegerRing(4),3)![2,1,3];
> c := Z2Z4Mult(i, G, Z2Z4Type(C)[3]);
> c;
(2 0 3 1 2 2)
> c in C`Code;
true
> Z2Z4Mult(i, G, Z2Z4Type(C)[3]) eq i*G;
false

> H := Z2Z4MinRowsParityCheckMatrix(C);
> Z2Z4Mult(c, Transpose(H), C`Alpha);
(0 0 0)
> Z2Z4Mult(c, Transpose(H), C`Alpha) eq c*Transpose(H);
false
> u := c;
> u[2] := u[2] + 2;
> u[4] := u[4] + 3;
> Z2Z4Mult(u, Transpose(H), C`Alpha);
(2 2 1)
> Z2Z4Mult(u, Transpose(H), C`Alpha) eq u*Transpose(H);
false
```

2.6.4 Distance and Weight

`Weight(v)`

The Hamming weight of the codeword v, i.e., the number of non-zero components of v.

`Distance(u, v)`

The Hamming distance between the codewords u and v, where u and v belong to the same $\mathbb{Z}_2\mathbb{Z}_4$-additive code C. This is defined to be the Hamming weight of $u - v$.

`Z2Z4LeeWeight(v, α)`

The Lee weight of the codeword v, i.e., the Hamming weight of the binary part (that is, the first α coordinates) of v plus the Lee weight of the quaternary part (that is, the rest of the coordinates) of v (see [6]). Equivalently, it corresponds to the number of non-zero components of $\Phi(v)$, where Φ is the Gray map defined in Subsection 2.5.1.

`Z2Z4LeeDistance(u, v, α)`

The Lee distance between the codewords u and v, where u and v belong to the same $\mathbb{Z}_2\mathbb{Z}_4$-additive code C. This is defined to be the Lee weight of $u - v$.

Example H2E14

We calculate the Hamming weight and distance of some vectors in $V = \mathbb{Z}_4^{\alpha+\beta}$, as well as the Lee weight and distance of these vectors. Note that when $\alpha = 0$, the functions for the Lee weight and distance return the same as that the corresponding functions for vectors over $\mathbb{Z}_4$.

```
> V := RSpace(IntegerRing(4),4);
> u := V![2,1,2,3];
> v := V![0,0,2,1];
> Distance(u, v);
3
> Distance(u, v) eq Weight(u-v);
true
> Z2Z4LeeDistance(u, v, 1);
4
> Z2Z4LeeDistance(u, v, 1) eq Z2Z4LeeWeight(u-v, 1);
true
> Z2Z4LeeDistance(u, v, 0) eq LeeDistance(u, v);
true
> Z2Z4LeeWeight(u, 0) eq LeeWeight(u);
true
```

2.6.5 Accessing Components of a Codeword

`u[i]`

Given a codeword u belonging to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, which is represented in MAGMA as a subspace of $V = \mathbb{Z}_4^{\alpha+\beta}$, return the i-th component of u (as an element of $\mathbb{Z}_4$).

`u[i] := x;`

Given an element u belonging to a $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode C of the full $\mathbb{Z}_4$-space $V = \mathbb{Z}_4^{\alpha+\beta}$, a positive integer i, $1 \leq i \leq \alpha + \beta$, and an element x of $\mathbb{Z}_4$, this function returns a vector in V which is u with its i-th component redefined to be x. It is not checked whether the elements in the first α coordinates are in $\{0, 2\}$.

2.7 Boolean Predicates

Again notice that sometimes in order to use the same functions as for quaternary linear codes, we will write `C‘Code` instead of `C`. Also, all codewords, which are elements in $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, are represented as elements in $V = \mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos.

2.7.1 Membership and Equality

`u in C‘Code`

Return `true` if and only if the vector u of $V = \mathbb{Z}_4^{\alpha+\beta}$ belongs to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$.

`u notin C‘Code`

Return `true` if and only if the vector u of $V = \mathbb{Z}_4^{\alpha+\beta}$ does not belong to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$.

`Z2Z4Subset(C, D)`

Return `true` if and only if the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C is a subcode of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code D.

`Z2Z4NotSubset(C, D)`

Return `true` if and only if the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C is not a subcode of the $\mathbb{Z}_2\mathbb{Z}_4$-additive code D.

`Z2Z4Equal(C, D)`

Return `true` if and only if the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes C and D are equal.

`Z2Z4NotEqual(C, D)`

Return `true` if and only if the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes C and D are not equal.

`IsZero(u)`

Return `true` if and only if the codeword u is the zero vector.

2.7.2 Properties

`IsZ2Z4AdditiveCode(C)`

Return `true` if and only if C is a $\mathbb{Z}_2\mathbb{Z}_4$-additive code.

`IsZ2Z4SelfDual(C)`

Return `true` if and only if the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C is additive self-dual, i.e., C equals the additive dual of C.

`IsZ2Z4SelfOrthogonal(C)`

Return `true` if and only if the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C is additive self-orthogonal, that is, return whether C is contained in the additive dual of C.

Example H2E15

We consider a $\mathbb{Z}_2\mathbb{Z}_4$-additive code and examine some of its properties.

```
> V := RSpace(IntegerRing(4), 5);
> C := Z2Z4AdditiveCode([V![2,2,2,0,0],V![0,0,0,2,0],V![0,2,1,0,1]] : Alpha:=2);
> C;
rec<Z2Z4Code |
    Code := ((5, 4^1 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 1 0 3]
    [0 2 1 0 1]
    [0 0 2 0 2]
    [0 0 0 2 0],
    Alpha := 2
    >
> IsZ2Z4SelfOrthogonal(C);
true
> IsZ2Z4SelfDual(C);
true
> Z2Z4Equal(C, Z2Z4Dual(C));
true
```

2.8 The Weight Distribution

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, we recall that the Lee weight of a codeword v is the Hamming weight of the binary part of v (that is, the first α coordinates) plus the Lee weight of the quaternary part of v (that is, the rest of the coordinates) (see [6]). Equivalently, it corresponds to the Hamming weight of $\Phi(v)$, where Φ is the Gray map defined in Subsection 2.5.1.

In the case of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, Lee weight and Lee distance distributions are equivalent (in particular, minimum Lee weight and minimum

Lee distance are equivalent). There are two different methods available to compute the minimum Lee weight. One of them is by using brute force, which is equivalent to compute the whole Lee weight distribution of the code. The other one is by using the representation of the code as the union of cosets of the kernel defined in Subsection 2.5.4, and the known Brouwer-Zimmermann's algorithm applied to linear subcodes given by the cosets of the kernel.

2.8.1 The Minimum Weight

`Z2Z4MinimumLeeWeight(C : parameters)`

`Z2Z4MinimumLeeDistance(C : parameters)`

`Method` MonStgElt *Default*: "Auto"

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the minimum Lee weight of the codewords belonging to the code C, which is also the minimum Lee distance between any two codewords.

Sometimes a brute force calculation of the entire Lee weight distribution can be a faster way to get the minimum Lee weight for small codes. When the parameter `Method` is set to the default `"Auto"`, then the method is internally chosen. The user can specify which method they want using setting it to either `"Distribution"` or `"KernelCoset"`.

`Z2Z4MinimumWord(C)`

`Method` MonStgElt *Default*: "Auto"

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return one codeword of the code C having minimum Lee weight.

Sometimes a brute force calculation of the entire Lee weight distribution can be a faster way to get one codeword of minimum Lee weight for small codes. When the parameter `Method` is set to the default `"Auto"`, then the method is internally chosen. The user can specify which method they want using setting it to either `"Distribution"` or `"KernelCoset"`.

`Z2Z4MinimumWords(C : parameters)`

`NumWords` RngIntElt *Default*:

`Method` MonStgElt *Default*: "Auto"

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, return the set of all codewords of C having minimum Lee weight. If `NumWords` is set to a non-negative integer, then the algorithm will terminate after at least that total of codewords have been found.

In some cases (such as small codes) a brute force enumeration may be a faster way to collect the words of minimum Lee weight. When the parameter `Method` is set to the default `"Auto"`, then the method is internally chosen. The user can specify which method they want using setting it to either `"Distribution"` or `"KernelCoset"`.

Example H2E16

We compute the minimum Lee weight of two $\mathbb{Z}_2\mathbb{Z}_4$-additive codes by using different methods. We show that if the code is small, a brute force calculation can be faster than using the method based on the computation of the cosets of the kernel.

```
> C := Z2Z4ExtendedPerfectCode(2, 5);
> Z2Z4Cardinal(C);
67108864
> time Z2Z4MinimumLeeWeight(C : Method := "Distribution");
4
Time: 1585.100
> time Z2Z4MinimumLeeWeight(C : Method := "KernelCosets");
4
Time: 0.140

> C := Z2Z4HadamardCode(3, 11);
> Z2Z4Cardinal(C);
4096
> time Z2Z4MinimumLeeWeight(C : Method := "Distribution");
1024
Time: 2.270
> time Z2Z4MinimumLeeWeight(C : Method := "KernelCosets");
1024
Time: 6.060
```

Then, we check some relations given by the functions related to the minimum Lee weight.

```
> C := Z2Z4HadamardCode(2, 5);
> Z2Z4MinimumLeeWeight(C) eq Z2Z4LeeWeight(Z2Z4MinimumWord(C), C`Alpha);
true
> Z2Z4MinimumWord(C) in Z2Z4MinimumWords(C);
true
> Z2Z4LeeWeightDistribution(C);
[ <0, 1>, <16, 62>, <32, 1> ]
> Z2Z4MinimumWords(C) eq Z2Z4MinimumWords(C : NumWords := 62);
true
```

2.8.2 The Weight Distribution

`Z2Z4LeeWeightDistribution(C)`

`Method` MonStgElt *Default*: "Auto"

Determine the Lee weight distribution for the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C. The distribution is returned in the form of a sequence of tuples, where the i-th tuple contains the i-th weight, w_i say, and the number of codewords having Lee weight w_i.

When the parameter `Method` is set to the default `"Auto"`, then the method is internally chosen. The user can specify which method they want using setting it to either `"Distribution"` or `"KernelCoset"`. In the first case, a brute force enumeration is used. In the second case, the representation of the code as the union of cosets of the kernel and the function `WeightDistribution` to compute the weight distribution of a coset of a binary linear code are used.

`Z2Z4DualLeeWeightDistribution(C)`

`Method` MonStgElt *Default*: "Auto"

The Lee weight distribution of the additive dual code of C (see `Z2Z4LeeWeightDistribution`).

`Z2Z4ExternalDistance(C)`

`Method` MonStgElt *Default*: "Auto"

Determine the external distance for the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C. The external distance of a code is the number of different nonzero Lee weights of the dual code of C.

When the parameter `Method` is set to the default `"Auto"`, then the method is internally chosen. The user can specify which method they want using setting it to either `"Distribution"` or `"KernelCoset"`. In the first case, to compute the Lee weight distribution a brute force enumeration is used. In the second case, the representation of the code as the union of cosets of the kernel and the function `WeightDistribution` to compute the weight distribution of a coset of a binary linear code are used.

Example H2E17

```
> C := Z2Z4HadamardCode(2, 5);
> Z2Z4LeeWeightDistribution(C);
[ <0, 1>, <16, 62>, <32, 1> ]
> Z2Z4MinimumLeeWeight(C) eq Z2Z4LeeWeightDistribution(C)[2][1];
true
> Z2Z4DualLeeWeightDistribution(C) eq Z2Z4LeeWeightDistribution(Z2Z4Dual(C));
```

```
true
> Z2Z4ExternalDistance(C) eq #Z2Z4DualLeeWeightDistribution(C)-1;
true
```

2.9 Constructing New Codes from Old

2.9.1 Construction of Subcodes

`Z2Z4OrderTwoSubcode(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode C_b which contains all order two codewords of C.

`Z2Z4Subcode(C, t1, t2)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ and two integers, $t1$ and $t2$, such that $1 \leq t1 \leq \gamma$ and $1 \leq t2 \leq \delta$, return a $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode of C of type $(\alpha, \beta; t1, t2; \kappa')$, where $\kappa' \leq \kappa$. This $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode is generated by the first $t1$ rows of order two and the first $t2$ rows of order four in the generator matrix given by the function `Z2Z4MinRowsGeneratorMatrix(C)`.

`Z2Z4Subcode(C, S1, S2)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ and two sets of integers, $S1$ and $S2$, such that each of their elements lies in the range $[1, \gamma]$ and $[1, \delta]$, respectively, return a $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode of C of type $(\alpha, \beta; |S1|, |S2|; \kappa')$, where $\kappa' \leq \kappa$. This $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode is generated by the rows of order two and four whose positions appear in $S1$ and $S2$, respectively, in the generator matrix given by the function `Z2Z4MinRowsGeneratorMatrix(C)`.

Example H2E18

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, we compute some subcodes of this code.

```
> C := RandomZ2Z4AdditiveCode(2, 4, 2, 3);
> C;
rec<Z2Z4Code |
    Code := ((6, 4^3 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 0 0 0]
    [0 2 0 0 0 1]
    [0 0 1 0 1 0]
    [0 0 0 1 0 0]
    [0 0 0 0 2 0]
    [0 0 0 0 0 2],
```

```
    Alpha := 2
    >
> Z2Z4Type(C);
[ 2, 4, 2, 3, 1 ]
> Z2Z4MinRowsGeneratorMatrix(C);
[0 0 0 0 2 0]
[2 0 0 0 0 0]
[0 0 1 0 1 0]
[0 0 0 1 0 0]
[0 2 0 0 0 1]
2 3

> C1 := Z2Z4Subcode(C,2,1);
> C1;
rec<Z2Z4Code |
    Code := ((6, 4^1 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 0 0 0]
    [0 0 1 0 1 0]
    [0 0 0 0 2 0],
    Alpha := 2
    >
> Z2Z4Subset(C1, C);
true
> Z2Z4Type(C1);
[ 2, 4, 2, 1, 1 ]
> Z2Z4MinRowsGeneratorMatrix(C1);
[0 0 2 0 0 0]
[2 0 0 0 0 0]
[0 0 1 0 1 0]
2 1

> C2 := Z2Z4Subcode(C,{2},{1,3});
> C2;
rec<Z2Z4Code |
    Code := ((6, 4^2 2^1)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 0 0 0]
    [0 2 0 0 0 1]
    [0 0 1 0 1 0]
    [0 0 0 0 0 2],
    Alpha := 2
    >
> Z2Z4MinRowsGeneratorMatrix(C2);
[2 0 0 0 0 0]
[0 0 1 0 1 0]
[0 2 0 0 0 1]
1 2
```

2.9.2 Sum and Intersection

`Z2Z4Sum(C, D)`

The (vector space) sum of the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes C and D, where C and D have the same parameters α and β, hence also the same length.

`Z2Z4Intersection(C, D)`

The intersection of the $\mathbb{Z}_2\mathbb{Z}_4$-additive codes C and D, where C and D have the same parameters α and β, hence also the same length.

Example H2E19

We verify some simple results from the sum and intersection of $\mathbb{Z}_2\mathbb{Z}_4$-additive subcodes.

```
> V  := RSpace(IntegerRing(4), 4);
> C  := Z2Z4AdditiveCode([V![2,0,0,2],V![0,1,0,3],V![0,0,1,3]] : Alpha:=1);

> C1 := Z2Z4Subcode(C,1,0);
> C2 := Z2Z4Subcode(C,0,1);
> C3 := Z2Z4Subcode(C,1,1);

> Z2Z4Equal(Z2Z4Sum(C1,C2), C3);
true
> Z2Z4Equal(Z2Z4Intersection(C1,C3), C1);
true
```

2.9.3 Standard Constructions

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C, which is represented in MAGMA as a subspace of $V = \mathbb{Z}_4^{\alpha+\beta}$, any codeword u belonging to C can be written as $u = (u_1|u_2)$, where $u_1 \in \mathbb{Z}_4^{\alpha}$ and $u_2 \in \mathbb{Z}_4^{\beta}$.

`Z2Z4DirectSum(C, D)`

Given $\mathbb{Z}_2\mathbb{Z}_4$-additive codes C and D, construct the direct sum of C and D. The direct sum is a $\mathbb{Z}_2\mathbb{Z}_4$-additive code that consists of all vectors of the form $(u_1, v_1|u_2, v_2)$, where $(u_1|u_2) \in C$ and $(v_1|v_2) \in D$.

`Z2Z4Concatenation(C, D)`

Given $\mathbb{Z}_2\mathbb{Z}_4$-additive codes C and D, return the concatenation of C and D. If $G_c = (A_\alpha|A_\beta)$ and $G_d = (B_\alpha|B_\beta)$ are the generator matrices of C and D, respectively, the concatenation of C and D is the $\mathbb{Z}_2\mathbb{Z}_4$-additive code with generator matrix whose rows consist of each row $(a_\alpha|a_\beta)$ of G_c concatenated with each row $(b_\alpha|b_\beta)$ of G_d in the following way: $(a_\alpha, b_\alpha|a_\beta, b_\beta)$.

`Z2Z4PlotkinSum(C, D)`

Given $\mathbb{Z}_2\mathbb{Z}_4$-additive codes C and D both with the same parameters α and β, hence also the same length, construct the Plotkin sum of C and D. The Plotkin sum is a $\mathbb{Z}_2\mathbb{Z}_4$-additive code that consists of all vectors of the form $(u_\alpha, u_\alpha + v_\alpha | u_\beta, u_\beta + v_\beta)$, where $(u_\alpha | u_\beta) \in C$ and $(v_\alpha | v_\beta) \in D$.

`Z2Z4PunctureCode(C, i)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C and an integer i, $1 \leq i \leq \alpha + \beta$, construct a new $\mathbb{Z}_2\mathbb{Z}_4$-additive code C' by deleting the i-th coordinate from each codeword of C.

`Z2Z4PunctureCode(C, S)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C and a set S of distinct integers $\{i_1, \ldots, i_r\}$ each of which lies in the range $[1, \alpha + \beta]$, construct a new $\mathbb{Z}_2\mathbb{Z}_4$-additive code C' by deleting the components $i_1, \ldots, i_r$ from each codeword of C.

`Z2Z4ShortenCode(C, i)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C and an integer i, $1 \leq i \leq \alpha + \beta$, construct a new $\mathbb{Z}_2\mathbb{Z}_4$-additive code from C by selecting only those codewords of C having a zero as their i-th component and deleting the i-th component from these codewords.

`Z2Z4ShortenCode(C, S)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C and a set S of distinct integers $\{i_1, \ldots, i_r\}$ each of which lies in the range $[1, \alpha + \beta]$, construct a new $\mathbb{Z}_2\mathbb{Z}_4$-additive code from C by selecting only those codewords of C having zeros in each of the coordinate positions $i_1, \ldots, i_r$, and deleting these components.

`Z2Z4ExtendCode(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C form a new $\mathbb{Z}_2\mathbb{Z}_4$-additive code C' from C by adding the appropriate extra binary coordinate to each codeword v of C such that $\Phi(v)$ has even Hamming weight, where Φ is the Gray map defined in Subsection 2.5.1.

Example H2E20

We combine $\mathbb{Z}_2\mathbb{Z}_4$-additive codes in diferent ways and look at the parameters α and β of the new $\mathbb{Z}_2\mathbb{Z}_4$-additive codes.

```
> C1 := RandomZ2Z4AdditiveCode(2, 3);
> C2 := RandomZ2Z4AdditiveCode(1, 4);
> Z2Z4Length(C1);
5 [ 2, 3 ]
> Z2Z4Length(C2);
5 [ 1, 4 ]
```

```
> C3 := Z2Z4DirectSum(C1,C2);
> Z2Z4Length(C3);
10 [ 3, 7 ]
> C4 := Z2Z4Concatenation(C1,C2);
> Z2Z4Length(C4);
10 [ 3, 7 ]
> Z2Z4Subset(C4,C3);
true

> C5 := Z2Z4PunctureCode(C1,3);
> Z2Z4Length(C5);
4 [ 2, 2 ]
```

2.10 Coset Representatives

`Z2Z4CosetRepresentatives(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, with ambient space $V = \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, return a set of coset representatives (not necessarily of minimal weight in their cosets) for C in V as an indexed set of vectors from V. The elements in V are represented as elements in $\mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos. The set of coset representatives $\{c_0, c_1, \ldots, c_t\}$ satisfies the conditions that c_0 is the zero codeword, and $V = \bigcup_{i=0}^{t} \big(C + c_i\big)$. Note that this function is only applicable when V and C are small.

`Z2Z4CosetRepresentatives(C, S)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, and a $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode S of C, return a set of coset representatives (not necessarily of minimal weight in their cosets) for S in C as an indexed set of codewords from C. The codewords in $C \subseteq \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ are represented as elements in $\mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos. The set of coset representatives $\{c_0, c_1, \ldots, c_t\}$ satisfies the conditions that c_0 is the zero codeword, and $C = \bigcup_{i=0}^{t} \big(S + c_i\big)$. Note that this function is only applicable when S and C are small.

Example H2E21

```
> V := RSpace(Integers(4), 5);
> C := Z2Z4AdditiveCode([V![2,0,0,2,0],
                         V![2,0,1,2,3],
                         V![0,0,2,1,3]] : Alpha := 1);
> L := Z2Z4CosetRepresentatives(C);
```

```
> {x : x in Set(V) | x[1] in {0,2}} eq {v+ci : v in Z2Z4Set(C), ci in L};
true

> K := Z2Z4KernelZ2Code(C);
> L := Z2Z4CosetRepresentatives(C, K);
> {C`Code!0} join Set(Z2Z4KernelCosetRepresentatives(C)) eq L;
true
> Z2Z4Set(C) eq {v+ci : v in Z2Z4Set(K), ci in L};
true
```

2.11 Cyclic codes

A $\mathbb{Z}_2\mathbb{Z}_4$-additive code C is cyclic if for any codeword

$$c = (a_0, \ldots, a_{\alpha-1} \mid b_0, \ldots, b_{\beta-1}) \in C \subseteq \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta,$$

its double right cyclic shift $(a_{\alpha-1}, a_0, \ldots, a_{\alpha-2} \mid b_{\beta-1}, b_0, \ldots, b_{\beta-2})$ is also a codeword in C. An element $c = (a_0, \ldots, a_{\alpha-1} \mid b_0, \ldots, b_{\beta-1}) \in \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ can be identified with a module element consisting of two polynomials $c(x) = (a_0 + a_1 x + \cdots + a_{\alpha-1}x^{\alpha-1} \mid b_0 + b_1 x + \cdots + b_{\beta-1}x^{\beta-1}) = (a(x) \mid b(x)) \in R_{\alpha,\beta}$, where $R_{\alpha,\beta} = \mathbb{Z}_2[x]/(x^\alpha - 1) \times \mathbb{Z}_4[x]/(x^\beta - 1)$. This identification gives a one-to-one correspondence between the elements of $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ and $R_{\alpha,\beta}$. Let $C(x)$ be the set of all polynomials associated to the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C. A subset $C \subseteq \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ is a $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code if and only if the subset $C(x) \subseteq R_{\alpha,\beta}$ is a $\mathbb{Z}_4[x]$-submodule of $R_{\alpha,\beta}$. Moreover, if C is a $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ with β odd, then

$$C(x) = \langle (p(x) \mid 0), (l(x) \mid f(x)h(x) + 2f(x)) \rangle,$$

where $p(x), l(x) \in \mathbb{Z}_2[x]/(x^\alpha - 1)$, $\deg(l(x)) < \deg(p(x))$, $p(x)|(x^\alpha - 1)$, $f(x), h(x) \in \mathbb{Z}_4[x]/(x^\beta - 1)$ with $f(x)h(x)|(x^\beta - 1)$, and $p(x)$ divides $\frac{x^\beta - 1}{f(x)} l(x)$ a $\mathbb{Z}_2[x]$. Note that if β is even, then $x^\beta - 1$ does not factorize uniquely over $\mathbb{Z}_4[x]$.

For more information about $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic codes, the reader is referred to [1, 2, 3], where these codes were introduced and have been studied deeply.

`Z2Z4CyclicCode(`α`, `β`, `p`, `l`, `f`, `h`)`

Given two non-negative integers α and β, and four polynomials $p(x)$, $l(x)$, $f(x)$ and $h(x)$, such that $p(x)$, $l(x) \in \mathbb{Z}_2[x]$ and $f(x)$, $h(x) \in \mathbb{Z}_4[x]$, construct the $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ generated by $(p(x) \mid 0)$ and $(l(x) \mid f(x)h(x) + 2f(x))$.

`Z2Z4CyclicCode(`α`, `β`, `a`, `b`)`

Given two non-negative integers α and β, and two polynomials $a(x) \in \mathbb{Z}_2[x]$ and $b(x) \in \mathbb{Z}_4[x]$, construct the $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ generated by $(a(x) \mid b(x))$.

`Z2Z4CyclicCode(`α`, `β`, G)`

Given two non-negative integers α and β, and a non-empty sequence G containing r tuples of polynomials, that is $G = [< a_1(x), b_1(x) >, \ldots, < a_r(x), b_r(x) >]$, where $a_i(x) \in \mathbb{Z}_2[x]$ and $b_i(x) \in \mathbb{Z}_4[x]$, for $1 \leq i \leq r$, construct the $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ generated by $(a_1(x) \mid b_1(x)), \ldots, (a_r(x) \mid b_r(x))$.

`Z2Z4CyclicCode(`α`, u)`

Given a non-negative integer α and a vector $u = (u_\alpha | u_\beta) \in \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, represented as an element in $V = \mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos, construct the $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ generated by the double right cyclic shifts of the vector u. It is checked whether the elements in the first α coordinates are in $\{0, 2\}$.

`Z2Z4CyclicCode(`α`, G)`

Given a non-negative integer α and a non-empty sequence of r vectors $G = [u_1, u_2, \ldots, u_r]$, where, for $1 \leq i \leq r$, $u_i \in \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ is represented as an element in $V = \mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos, construct the $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ generated by the double right cyclic shifts of the vectors $u_1, u_2, \ldots, u_r$. It is checked whether the elements in the first α coordinates are in $\{0, 2\}$.

`RandomZ2Z4CyclicCode(`α`, `β`)`

Given two non-negative integers α and β, with β odd, return a random $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code of type $(\alpha, \beta; \gamma, \delta; \kappa)$ and a tuple containing the generator polynomials $< p(x), l(x), f(x), h(x) >$, where $p(x)$, $l(x) \in \mathbb{Z}_2[x]$ and $f(x)$, $h(x) \in \mathbb{Z}_4[x]$, satisfying the conditions described in Subsection 2.11.

Example H2E22

```
> PR2<x> := PolynomialRing(Integers(2));
> PR4<y> := PolynomialRing(Integers(4));
> alpha := 15;
> beta := 7;

> p := x^5+x^3+x+1;
> l := x^4+x^3+1;
```

```
> f := PR4!1;
> h := y^4+y^3+3*y^2+2*y+1;
> C1 := Z2Z4CyclicCode(alpha, beta, p, l, f, h);
> C2 := Z2Z4CyclicCode(alpha, beta, [<p, PR4!0>, <l, f*h + 2*f>]);
> Z2Z4Equal(C1, C2);
true
> IsZ2Z4Cyclic(C1);
true

> V := RSpace(Integers(4), alpha + beta);
> g1 := V!([2*(Coefficient(PR4!p, i)) : i in [0 .. alpha - 1]]
          cat [Integers(4)!0 : j in [0 .. beta - 1]]);
> g1;
(2 2 0 2 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
> g2 := V!([2*(Coefficient(PR4!l, i)) : i in [0 .. alpha - 1]]
          cat [Coefficient(f*h + 2*f, j) : j in [0 .. beta - 1]]);
> g2;
(2 0 0 2 2 0 0 0 0 0 0 0 0 0 3 2 3 1 1 0 0)
> C3 := Z2Z4CyclicCode(alpha, [g1, g2]);
> Z2Z4Equal(C1, C3);
true

> C4 := Z2Z4CyclicCode(alpha, g1);
> C5 := Z2Z4CyclicCode(alpha, g2);
> Z2Z4Subset(C4, C3) and Z2Z4Subset(C5, C3);
true

> G := Z2Z4MinRowsGeneratorMatrix(C1);
> v := Eltseq(G[1]);
> v;
[ 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 0, 0, 0, 0, 0, 0, 2 ]
> u := Rotate(v[1..alpha],3) cat Rotate(v[alpha+1..alpha+beta],3);
> u;
[ 2, 2, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 2, 0, 0, 0, 0 ]
> V ! u in C1`Code;
true

> C6 := RandomZ2Z4CyclicCode(alpha, beta);
> IsZ2Z4AdditiveCode(C6) and IsZ2Z4Cyclic(C6);
true
> Z2Z4Type(C6)[1] eq alpha  and Z2Z4Type(C6)[2] eq beta;
true
```

`IsZ2Z4Cyclic(C)`

Return `true` if and only if the $\mathbb{Z}_2\mathbb{Z}_4$-additive code C is cyclic.

`Z2Z4GeneratorPolynomials(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ with β odd, return a tuple containing the generator polynomials $< p(x), l(x), f(x), h(x) >$, where $p(x)$, $l(x) \in \mathbb{Z}_2[x]$ and $f(x)$, $h(x) \in \mathbb{Z}_4[x]$, satisfying the conditions described in Subsection 2.11 (see [3]).

`Z2Z4CheckPolynomials(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ with β odd, return a tuple containing the check polynomials $< p'(x), l'(x), f'(x), h'(x) >$, where $p'(x)$, $l'(x) \in \mathbb{Z}_2[x]$ and $f'(x)$, $h'(x) \in \mathbb{Z}_4[x]$. These check polynomials are the generator polynomials of the additive dual code of C, and satisfy the conditions described in Subsection 2.11 (see [2]).

`Z2Z4CheckPolynomials(`α`, `β`, `p`, `l`, `f`, `h`)`

Given two non-negative integers α and β, with β odd, and four polynomials, $p(x)$, $l(x)$, $f(x)$ and $h(x)$, such that $p(x)$, $l(x) \in \mathbb{Z}_2[x]$ and $f(x)$, $h(x) \in \mathbb{Z}_4[x]$, return a tuple containing the check polynomials $< p'(x), l'(x), f'(x), h'(x) >$, where $p'(x)$, $l'(x) \in \mathbb{Z}_2[x]$ and $f'(x)$, $h'(x) \in \mathbb{Z}_4[x]$, of the $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ generated by the given polynomials. These check polynomials are the generator polynomials of the additive dual code of C, and satisfy the conditions described in Subsection 2.11.

Example H2E23

```
> PR2<x> := PolynomialRing(Integers(2));
> PR4<y> := PolynomialRing(Integers(4));
> alpha := 15;
> beta := 7;

> U := Z2Z4AdditiveUniverseCode(alpha, beta);
> Z := Z2Z4AdditiveZeroCode(alpha, beta);
> IsZ2Z4Cyclic(U);
true
> IsZ2Z4Cyclic(Z);
true
> Z2Z4GeneratorPolynomials(U);
<1, 0, 1, 1>
> Z2Z4GeneratorPolynomials(Z);
<x^15 + 1, 0, y^7 + 3, 1>

> a1 := x^6+x^4+x^2+x;
> a2 := x^5+x^4+x;
> b1 := PR4!0;
```

```
> b2 := y^5+y^4+3*y^3+2*y^2+3*y;
> C1 := Z2Z4CyclicCode(alpha, beta, [<a1, b1>, <a2, b2>]);
> Z2Z4GeneratorPolynomials(C1);
<x^5 + x^3 + x + 1, x^4 + x^3 + 1, 1, y^4 + y^3 + 3*y^2 + 2*y + 1>
> p := x^5+x^3+x+1;
> l := x^4+x^3+1;
> f := PR4!1;
> h := y^4+y^3+3*y^2+2*y+1;
> C2 := Z2Z4CyclicCode(alpha, beta, [<p, PR4!0>, <l, f*h + 2*f>]);
> Z2Z4Equal(C1, C2);
true

> Z2Z4CheckPolynomials(C2) eq Z2Z4CheckPolynomials(alpha, beta, p, l, f, h);
true
> Z2Z4CheckPolynomials(C2) eq Z2Z4GeneratorPolynomials(Z2Z4Dual(C2));
true
```

2.12 Information Space and Information Sets

`Z2Z4InformationSpace(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{\gamma+\delta}$ isomorphic to $\mathbb{Z}_2^{\gamma} \times \mathbb{Z}_4^{\delta}$ such that the first γ coordinates are of order two, that is, the space of information vectors for C. The function also returns the $(\gamma + 2\delta)$-dimensional binary vector space, which is the space of information vectors for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map. Finally, for the encoding process, it also returns the corresponding isomorphisms f and f_{bin} from these spaces of information vectors onto C and C_{bin}, respectively.

Example H2E24

```
> C := Z2Z4HadamardCode(2, 4);
> C;
rec<recformat<Code: CodeLinRng, Alpha: RngIntElt> |
    Code := ((10, 4^2 2^1)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2 1 3 1 0 3 2]
    [0 2 0 2 0 2 1 1 1 1]
    [0 0 2 2 1 1 0 1 2 3]
    [0 0 0 0 2 2 0 2 0 2]
    [0 0 0 0 0 0 2 2 2 2],
    Alpha := 4>
> R, V, f, fbin := Z2Z4InformationSpace(C);
> G := Z2Z4MinRowsGeneratorMatrix(C);
```

```
> (#R eq #C‘Code) and (#V eq #C‘Code);
true
> Set([f(i) : i in R]) eq Z2Z4Set(C);
true
> Set([Z2Z4Mult(i, G, Z2Z4Type(C)[3]): i in R]) eq Z2Z4Set(C);
true
> Set([i*G : i in R]) eq Z2Z4Set(C);
false

> i := R![2,3,1];
> c := f(i);
> c;
(2 0 0 2 3 1 1 2 3 0)
> c in C‘Code;
true
> c eq Z2Z4Mult(i, G, Z2Z4Type(C)[3]);
true
> c eq i*G;
false

> ibin := V![1,1,0,0,1];
> ibin eq Z2Z4GrayMap(Z2Z4AdditiveUniverseCode(1, 2))(i);
true
> cbin := fbin(ibin);
> cbin;
(1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 0)
> cbin eq Z2Z4GrayMap(C)(c);
true
```

`Z2Z4InformationSet(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return an information set $I = [i_1, \ldots, i_{\gamma+\delta}] \subseteq \{1, \ldots, \alpha + \beta\}$ for C such that $\{i_1, \ldots, i_\kappa\} \subseteq \{1, \ldots, \alpha\}$ and the code C punctured on $\{1, \ldots, \alpha + \beta\} \backslash \{i_{\gamma+1}, \ldots, i_{\gamma+\delta}\}$ is of type 4^δ, and the corresponding information set $\Phi(I) = [i_1, \ldots, i_\kappa, 2i_{\kappa+1} - 1 - \alpha, \ldots, 2i_\gamma - 1 - \alpha, 2i_{\gamma+1} - 1 - \alpha, 2i_{\gamma+1} - \alpha, \ldots, 2i_{\gamma+\delta} - 1 - \alpha, 2i_{\gamma+\delta} - \alpha] \subseteq \{1, \ldots, \alpha + 2\beta\}$ for the binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map. The information sets I and $\Phi(I)$ are returned as a sequence of $\gamma + \delta$ and $\gamma + 2\delta$ integers, giving the coordinate positions that correspond to the information set of C and C_{bin}, respectively.

An information set I for C is an ordered set of $\gamma + \delta$ coordinate positions such that $|C^I| = 2^\gamma 4^\delta$, where $C^I = \{v^I : v \in C\}$ and v^I is the vector v restricted to the I coordinates. An information set J for C_{bin} is an ordered set of $\gamma + 2\delta$ coordinate positions such that $|C_{bin}^J| = 2^{\gamma+2\delta}$.

`IsZ2Z4InformationSet(C, I)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ and a sequence $I \subseteq \{1, \ldots, \alpha + \beta\}$ or $I \subseteq \{1, \ldots, \alpha + 2\beta\}$, return `true` if and only if $I \subseteq \{1, \ldots, \alpha + \beta\}$ is an information set for C. This function also returns another boolean, which is `true` if an only if $I \subseteq \{1, \ldots, \alpha + 2\beta\}$ is an information set for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map.

An information set I for C is an ordered set of $\gamma + \delta$ coordinate positions such that $|C^I| = 2^\gamma 4^\delta$, where $C^I = \{v^I : v \in C\}$ and v^I is the vector v restricted to the I coordinates. An information set J for C_{bin} is an ordered set of $\gamma + 2\delta$ coordinate positions such that $|C_{bin}^J| = 2^{\gamma+2\delta}$.

Example H2E25

```
> C := Z2Z4HadamardCode(2, 5);
> C;
rec<recformat<Code: CodeLinRng, Alpha: RngIntElt> |
    Code := ((20, 4^2 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2 0 2 2 0 1 3 1 0 3 2 3 1 3 2 1 0]
    [0 2 0 2 0 2 0 2 0 2 1 1 1 1 0 2 1 1 1 1]
    [0 0 2 2 0 0 2 2 1 1 0 1 2 3 1 1 0 1 2 3]
    [0 0 0 0 2 2 2 2 0 0 0 0 0 0 2 2 2 2 2 2]
    [0 0 0 0 0 0 0 0 2 2 0 2 0 2 2 2 2 0 2 0 2]
    [0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 0 0 2 2 2 2],
    Alpha := 8>
> n := Z2Z4Length(C);
> gamma := Z2Z4Type(C)[3];
> delta := Z2Z4Type(C)[4];

> I, Ibin := Z2Z4InformationSet(C);
> I;
[ 1, 5, 19, 20 ]
> Ibin;
[ 1, 5, 29, 30, 31, 32 ]
> #Z2Z4PunctureCode(C, {1..n} diff Set(I))`Code eq #C`Code;
true
> Cbin := Z2Z4GrayMapImage(C);
> V := VectorSpace(GF(2), gamma + 2*delta);
> #{V![c[i] : i in Ibin] : c in Cbin} eq #Cbin;
true

> IsZ2Z4InformationSet(C, I);
true false
> IsZ2Z4InformationSet(C, Ibin);
false true
```

```
> IsZ2Z4InformationSet(C, [1, 5, 9, 11]);
true false
> IsZ2Z4InformationSet(C, [1, 5, 9, 10, 13, 14]);
false true

> V := RSpace(IntegerRing(4), 5);
> D := Z2Z4AdditiveCode([V![2,0,0,2,0],V![0,2,0,2,2],V![0,0,2,2,0]] : Alpha:=5);
> IsZ2Z4InformationSet(D, [1, 3, 5]);
true true
```

2.13 Syndrome Space and Coset Leaders

`Z2Z4SyndromeSpace(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, return the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{\alpha+\beta-\delta-\kappa}$ isomorphic to $\mathbb{Z}_2^{\alpha+\gamma-2\kappa} \times \mathbb{Z}_4^{\beta-\gamma-\delta+\kappa}$ such that the first $\alpha + \gamma - 2\kappa$ coordinates are of order two, that is, the space of syndrome vectors for C. The function also returns the $(\alpha + 2\beta - \gamma - 2\delta)$-dimensional binary vector space, which is the space of syndrome vectors for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map. Note that these spaces are computed by using the function `Z2Z4InformationSpace(C)` applied to the dual code of C, produced by function `Z2Z4Dual(C)`.

`Z2Z4Syndrome(u, C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, and a vector u from the ambient space $V = \mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ or $V_2 = \mathbb{Z}_2^{\alpha+2\beta}$, construct the syndrome of u relative to the code C. The elements in $V = \mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ are represented as elements in $\mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos. The syndrome is an element of the syndrome space of C, considered as the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{\alpha+\beta-\delta-\kappa}$ isomorphic to $\mathbb{Z}_2^{\alpha+\gamma-2\kappa} \times \mathbb{Z}_4^{\beta-\gamma-\delta+\kappa}$ such that the first $\alpha + \gamma - 2\kappa$ coordinates are of order two.

`Z2Z4CosetLeaders(C)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, with ambient space $V = \mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$, return a set of coset leaders (vectors of minimal Lee weight in their cosets) for C in V as an indexed set of vectors from V. The elements in V are represented as elements in $\mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos. This function also returns a map from the syndrome space of C onto the coset leaders (mapping a syndrome into its corresponding coset leader). Note that this function is only applicable when V and C are small.

Example H2E26

```
> C := Z2Z4ExtendedPerfectCode(2, 5);
> C;
rec<recformat<Code: CodeLinRng, Alpha: RngIntElt> |
    Code := ((20, 4^2 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2 0 2 2 0 1 3 1 0 3 2 3 1 3 2 1 0]
    [0 2 0 2 0 2 0 2 0 2 1 1 1 1 0 2 1 1 1 1]
    [0 0 2 2 0 0 2 2 1 1 0 1 2 3 1 1 0 1 2 3]
    [0 0 0 0 2 2 2 2 0 0 0 0 0 0 2 2 2 2 2 2]
    [0 0 0 0 0 0 0 0 2 2 0 2 0 2 2 2 0 2 0 2]
    [0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 2 2 2 2],
    Alpha := 8>
> alpha := C`Alpha;
> beta := Z2Z4Length(C) - alpha;
> R, V, f, fbin := Z2Z4InformationSpace(C);
> Rs, Vs := Z2Z4SyndromeSpace(C);
> #R * #Rs eq 2^alpha * 4^beta;
true
> #V * #Vs eq 2^alpha * 4^beta;
true

> i := R![2,0,2,0,2,0,1,3,0,0,0,1,3,0,0,0];
> c := f(i);
> c;
(0 0 2 0 2 2 0 2 3 1 0 0 0 3 0 0 0 1 0 2)
> u := c;
> u[11] := u[11] + 3;
> u;
(0 0 2 0 2 2 0 2 3 1 3 0 0 3 0 0 0 1 0 2)

> s := Z2Z4Syndrome(u, C);
> s in Rs;
true
> H := Z2Z4MinRowsParityCheckMatrix(C);
> s eq Z2Z4Mult(u, Transpose(H), alpha);
true
> s eq u*Transpose(H);
false

> L, mapCosetLeaders := Z2Z4CosetLeaders(C);
> errorVector := mapCosetLeaders(s);
> errorVector;
(0 0 0 0 0 0 0 0 0 0 3 0 0 0 0 0 0 0 0 0)
> errorVector in L;
true
> u-errorVector eq c;
true
```

2.14 Decoding

This section describes functions for decoding vectors from the ambient space of a $\mathbb{Z}_2\mathbb{Z}_4$-additive code, or the corresponding space over $\mathbb{Z}_2$ under the Gray map, using two different algorithms: coset decoding and syndrome decoding. The reader is referred to [10, 24] for more information on coset decoding; and to [11, 14, 25] on syndrome decoding.

2.14.1 Coset Decoding

`Z2Z4CosetDecode(C, u : parameters)`

`MinWeightCode`	RngIntElt	*Default* : -
`MinWeightKernel`	RngIntElt	*Default* : -

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, and a vector u from the ambient space $V = \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ or $V_2 = \mathbb{Z}_2^{\alpha+2\beta}$, attempt to decode u with respect to C. The elements in $V = \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$ are represented as elements in $\mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos. If the decoding algorithm succeeds in computing a vector $u' \in C$ as the decoded version of $u \in V$, then the function returns `true`, u' and $\Phi(u')$, where Φ is the Gray map. If the decoding algorithm does not succeed in decoding u, then the function returns `false`, the zero vector in V and the zero vector in V_2.

The coset decoding algorithm considers the binary linear code $C_u = C_{bin} \cup (C_{bin} + \Phi(u))$, when $C_{bin} = \Phi(C)$ is linear. On the other hand, when C_{bin} is nonlinear, we have $C_{bin} = \bigcup_{i=0}^{t}(K_{bin} + \Phi(c_i))$, where $K_{bin} = \Phi(K_C)$, K_C is the kernel of C as a $\mathbb{Z}_2\mathbb{Z}_4$-additive subcode, $[c_0, c_1, \ldots, c_t]$ are the coset representatives of C with respect to K_C (not necessarily of minimal weight in their cosets) and c_0 is the zero codeword. In this case, the algorithm considers the binary linear codes $K_0 = K_{bin} \cup (K_{bin} + \Phi(u))$, $K_1 = K_{bin} \cup (K_{bin} + \Phi(c_1) + \Phi(u))$, ..., $K_t = K_{bin} \cup (K_{bin} + \Phi(c_t) + \Phi(u))$.

If the parameter `MinWeightCode` is not assigned, then the minimum Lee weight of C, which coincides with the minimum weight of C_{bin}, denoted by d, is computed. Note that the minimum distance of C_{bin} coincides with its minimum weight. If C_{bin} is linear and the minimum weight of C_u is less than d, then $\Phi(u') = \Phi(u) + e$, where e is a word of minimum weight of C_u; otherwise, the decoding algorithm returns `false`. On the other hand, if C_{bin} is nonlinear and the minimum weight of $\cup_{i=0}^{t} K_i$ is less than the minimum weight of K_{bin}, then $\Phi(u') = \Phi(u) + e$, where e is a word of minimum weight of $\cup_{i=0}^{t} K_i$; otherwise, the decoding algorithm returns `false`. If the parameter `MinWeightKernel` is not assigned, then the minimum Hamming weight of K_{bin} is computed.

`Z2Z4CosetDecode(C, Q : parameters)`

`MinWeightCode`	RngIntElt	*Default* : -
`MinWeightKernel`	RngIntElt	*Default* : -

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$ and a sequence Q of vectors from the ambient space $V = \mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ or $V_2 = \mathbb{Z}_2^{\alpha+2\beta}$, attempt to decode the vectors of Q with respect to C. This function is similar to the function `Z2Z4CosetDecode(C, u)` except that rather than decoding a single vector, it decodes a sequence of vectors and returns a sequence of booleans and two sequences of decoded vectors corresponding to the given sequence. The algorithm used and effect of the parameters `MinWeightCode` and `MinWeightKernel` are identical to those for the function `Z2Z4CosetDecode(C, u)`.

Example H2E27

Starting with the $\mathbb{Z}_2\mathbb{Z}_4$-additive Hadamard code C of type $(8, 12; 2, 2; 2)$, a codeword $c \in C$ is selected and then perturbed to give a vector u in the ambient space of C. The vector u is then decoded to recover c.

```
> C := Z2Z4HadamardCode(2, 5);
> C;
rec<recformat<Code: CodeLinRng, Alpha: RngIntElt> |
    Code := ((20, 4^2 2^2)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2 0 2 2 0 1 3 1 0 3 2 3 1 3 2 1 0]
    [0 2 0 2 0 2 0 2 0 2 1 1 1 1 0 2 1 1 1 1]
    [0 0 2 2 0 0 2 2 1 1 0 1 2 3 1 1 0 1 2 3]
    [0 0 0 0 2 2 2 2 0 0 0 0 0 0 2 2 2 2 2 2]
    [0 0 0 0 0 0 0 0 2 2 0 2 0 2 2 2 0 2 0 2]
    [0 0 0 0 0 0 0 0 0 0 2 2 2 2 0 0 2 2 2 2],
    Alpha := 8>
> alpha := C`Alpha;
> beta := Z2Z4Length(C) - alpha;
> d := Z2Z4MinimumLeeDistance(C);
```

```
> t := Floor((d-1)/2);
> t;
7

> c := C`Code![0,0,2,2,0,0,2,2,1,1,0,1,2,3,1,1,0,1,2,3];
> c in C`Code;
true
> u := c;
> u[5] := u[5] + 2;
> u[12] := u[12] + 1;
> u[13] := u[13] + 3;
> u[16] := u[16] + 2;
> u;
(0 0 2 2 2 0 2 2 1 1 0 2 1 3 1 3 0 1 2 3)
> grayMap := Z2Z4GrayMap(Z2Z4AdditiveUniverseCode(alpha, beta));
> grayMap(c-u);
(0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0)

> isDecoded, uDecoded := Z2Z4CosetDecode(C, u : MinWeightCode := d);
> isDecoded;
true
> uDecoded eq c;
true
```

2.14.2 Syndrome Decoding

`Z2Z4SyndromeDecode(C, u)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, and a vector u from the ambient space $V = \mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ or $V_2 = \mathbb{Z}_2^{\alpha+2\beta}$, attempt to decode u with respect to C. The elements in $V = \mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ are represented as elements in $\mathbb{Z}_4^{\alpha+\beta}$ by changing the ones in the first α coordinates by twos. The decoding algorithm always succeeds in computing a vector $u' \in C$ as the decoded version of $u \in V$, and the function returns `true`, u' and $\Phi(u')$, where Φ is the Gray map. Although the function never returns `false`, the first output parameter `true` is given to be consistent with the other decoding functions.

The syndrome decoding algorithm consists of computing a table pairing each possible syndrome s with a vector of minimum Lee weight e_s, called coset leader, in the coset of C containing all vectors having syndrome s. After receiving a vector u, its syndrome s is computed using the parity check matrix. Then, u is decoded into the codeword $c = u - e_s$.

`Z2Z4SyndromeDecode(C, Q)`

Given a $\mathbb{Z}_2\mathbb{Z}_4$-additive code C of type $(\alpha, \beta; \gamma, \delta; \kappa)$, and a sequence Q of vectors from the ambient space $V = \mathbb{Z}_2^{\alpha} \times \mathbb{Z}_4^{\beta}$ or $V_2 = \mathbb{Z}_2^{\alpha+2\beta}$, attempt to decode the vectors of Q with respect to C. This function is similar to the function `Z2Z4SyndromeDecode(C, u)` except that rather than decoding a single vector, it decodes a sequence of vectors and returns a sequence of booleans and two sequences of decoded vectors corresponding to the given sequence. The algorithm used is the same as that of function `Z2Z4SyndromeDecode(C, u)`.

Example H2E28

The $\mathbb{Z}_2\mathbb{Z}_4$-additive Hadamard code C of type $(4, 6; 1, 2; 1)$ is constructed. Next, information bits are encoded using C and three errors are introduced to give the vector u. Then u is decoded by calculating its syndrome and applying the map, given by the `Z2Z4CosetLeaders` function, to the syndrome to recover the original vector.

```
> C := Z2Z4HadamardCode(2, 4);
> C;
rec<recformat<Code: CodeLinRng, Alpha: RngIntElt> |
    Code := ((10, 4^2 2^1)) Linear Code over IntegerRing(4)
    Generator matrix:
    [2 0 0 2 1 3 1 0 3 2]
    [0 2 0 2 0 2 1 1 1 1]
    [0 0 2 2 1 1 0 1 2 3]
    [0 0 0 0 2 2 0 2 0 2]
    [0 0 0 0 0 0 2 2 2 2],
    Alpha := 4>
> alpha := C`Alpha;
> beta := Z2Z4Length(C) - alpha;
> t := Floor((Z2Z4MinimumLeeDistance(C)-1)/2);
> t;
3

> R, V, f, fbin := Z2Z4InformationSpace(C);
> i := R![2,1,0];
> c := f(i);
> c;
(2 2 0 0 1 1 0 3 2 1)
> u := c;
> u[2] := u[2] + 2;
> u[7] := u[7] + 3;
> u[9] := u[9] + 1;
> u;
(2 0 0 0 1 1 3 3 3 1)
> grayMap := Z2Z4GrayMap(Z2Z4AdditiveUniverseCode(alpha, beta));
> grayMap(c-u);
(0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0)
```

```
> isDecoded, uDecoded := Z2Z4SyndromeDecode(C, u);
> isDecoded;
true
> uDecoded eq c;
true

> L, mapCosetLeaders := Z2Z4CosetLeaders(C);
> errorVector := mapCosetLeaders(Z2Z4Syndrome(u, C));
> errorVector;
(0 2 0 0 0 0 3 0 1 0)
> u-errorVector eq c;
true
```

Bibliography

[1] T. Abualrub, I. Siap, H. Aydin, "$\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic codes," *IEEE Trans. on Information Theory*, vol. 60(3), pp. 1508-1514, 2014.

[2] J. Borges, C. Fernández-Córdoba, R. Ten-Valls, "$\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic codes, generator polynomials and dual codes," *IEEE Trans. on Information Theory*, vol. 62(11), pp. 6348-6354, 2016.

[3] J. Borges, C. Fernández-Córdoba, R. Ten-Valls, "Computing the generator polynomials of $\mathbb{Z}_2\mathbb{Z}_4$-additive cyclic codes," arXiv:1606.01745, 2016.

[4] J. Borges, J. Rifà, "A characterization of 1-perfect additive codes," *IEEE Trans. on Information Theory*, vol. 45, pp. 1688-1697, 1999.

[5] J. Borges, K.T. Phelps, J. Rifà, "The Rank and Kernel of Extended 1-perfect $\mathbb{Z}_4$-Linear and Additive Non-$\mathbb{Z}_4$-Linear Codes," *IEEE Trans. on Information Theory*, vol. 49(8), pp. 2028-2034, 2003.

[6] J. Borges, C. Fernández-Córdoba, J. Pujol, J. Rifà and M. Villanueva, "$\mathbb{Z}_2\mathbb{Z}_4$-linear codes: generator matrices and duality," *Designs, Codes and Cryptography*, vol 54, no. 2, pp. 167-179, 2010.

[7] J. J. Cannon, W. Bosma, C. Fieker and A. Steel (Eds.) *Handbook of* Magma *Functions*, Edition 2.22, 5669 pages, 2016.

[8] P. Delsarte, "An algebraic approach to the association schemes of coding theory," *Philips Res. Rep. Suppl.*, vol. 10, no. 2, pp. 1-97, 1973.

[9] P. Delsarte, V. Levenshtein, "Association schemes and coding theory," *IEEE Trans. on Information Theory*, vol. 44(6), pp. 2477–2504, 1998.

[10] C. Fernández-Córdoba, J. Pujol, and M. Villanueva, "$\mathbb{Z}_2\mathbb{Z}_4$-linear codes: rank and kernel," *Designs, Codes and Cryptography*, vol 56, no. 1, pp. 43-59, 2010.

[11] A.R. Hammons, P.V. Kumar, A.R. Calderbank, N.J.A. Sloane and P. Solé, "The $\mathbb{Z}_4$-linearity of kerdock, preparata, goethals and related codes," *IEEE Trans. on Information Theory*, vol. 40, no. 2, pp. 301-319, 1994.

[12] J. A. Howell, "Spans in the module $\mathbb{Z}_m^s$," *Linear and Multilinear Algebra*, 19, pp. 67-77, 1986.

[13] D. S. Krotov, "$\mathbb{Z}_4$-linear Hadamard and extended perfect codes," *Proc. of the International Workshop on Coding and Cryptography*, Paris (France), Jan. 8-12, pp. 329–334, 2001.

[14] F. I. MacWilliams and N. J. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, New York, 1977.

[15] J. Pernas, J. Pujol, M. Villanueva, "Classification of Some Families of Quaternary Reed-Muller Codes," *IEEE Trans. on Information Theory*, vol. 57(9), pp. 6043-6051, 2011.

[16] K. T. Phelps, J. Rifà, "On binary 1-perfect additive codes: some structural properties," *IEEE Trans. on Inform. Theory*, vol. 48, pp. 2587-2592, 2002.

[17] K.T. Phelps, J. Rifà, M. Villanueva, "On the Additive ($\mathbb{Z}_4$-Linear and Non-$\mathbb{Z}_4$-Linear) Hadamard Codes: Rank and Kernel," *IEEE Trans. on Information Theory*, vol. 52(1), pp. 316-319, 2006.

[18] J. Pujol, J. Rifà, F. I. Solov'eva, "Quaternary Plotkin constructions and Quaternary Reed-Muller codes," *Lecture Notes in Computer Science* n. 4851, pp. 148-157, 2007.

[19] J. Pujol, J. Rifà and F. I. Solov'eva "Construction of $\mathbb{Z}_4$-linear Reed-Muller codes," *IEEE Trans. on Information Theory*, vol. 55(1), pp. 99-104, 2009.

[20] J. Rifà, J.M. Basart, L. Huguet, "On completely regular propelinear codes," in *Proc. 6th Int. Conference, AAECC-6*, no. 357 LNCS, pp. 341-355, 1989.

[21] J. Rifà, J. Pujol, "Translation invariant propelinear codes," *IEEE Trans. on Information Theory*, vol. 43, pp. 590-598, 1997.

[22] F. I. Solov'eva "On $\mathbb{Z}_4$-linear codes with parameters of Reed-Muller codes," *Problems of Information Transmission*, vol. 43, no. 1, pp. 26-32, 2007.

[23] A. Storjohann and T. Mulders, "Fast algorithms for linear algebra modulo N," *Lecture Notes In Computer Science*, vol. 1461, pp. 139-150, 1998.

[24] M. Villanueva, F. Zeng, J. Pujol, "Efficient representation of binary nonlinear codes: constructions and minimum distance computation," *Designs, Codes and Cryptography*, vol. 76(1), pp. 3-21, 2015.

[25] Z.-X. Wan, *Quaternary Codes*, World Scientific, 1997.