LINEAR CODES OVER THE INTEGER RESIDUE RING $\mathbb{Z}_{p^s}$

A MAGMA Package[1]

by

Cristina Fernández-Córdoba, Adrián Torres-Martín and Mercè Villanueva

Combinatoric, Coding and Security Group (CCSG)

Universitat Autònoma de Barcelona

Version 1.0

Barcelona
February 6, 2024

[1]This work has been partially supported by the Spanish Ministerio de Ciencia e Innovación under grants PID2019-104664GB-I00 and PID2022-137924NB-I00 (AEI / 10.13039/501100011033), and by the Catalan AGAUR grant 2021SGR 00643.

Contents

Chapter 1

Linear Codes over $\mathbb{Z}_{p^s}$

1.1 Introduction

Magma currently supports the basic facilities for linear codes over integer residue rings and Galois rings [8, Chapter 164], including cyclic codes and the complete weight enumerator calculation. Moreover, specific functions 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 $\mathbb{Z}_4^n$ are also supported [8, Chapter 165].

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 may obtain binary nonlinear codes better than any known binary linear code with the same parameters. More specifically, Hammons et. al. [21] 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 establish the relation between both families of nonlinear Kerdock and Preparata codes, by showing that their weight enumerators satisfy the MacWilliams identities, because they are dual as codes over $\mathbb{Z}_4$. 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 have been studied and classified (for example, extended Hamming codes, Hadamard codes, and Reed-Muller codes) [6, 25, 28, 32, 35].

Magma also supports functions for additive codes over a finite field, which are a generalization of the linear codes over a finite field [8, Chapter 166] in a Hamming scheme. According to a more general definition of additive codes given by Delsarte in 1973 [11, 12], 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_2}, the only structures for the abelian group are

those of the form $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, with $\alpha + 2\beta = n_2$. 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, they are called $\mathbb{Z}_2\mathbb{Z}_4$-*additive codes*.

The $\mathbb{Z}_2\mathbb{Z}_4$-additive codes can be seen as a generalization of linear codes over $\mathbb{Z}_2$ and $\mathbb{Z}_4$, since they are subgroups of $\mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$. Specifically, when $\alpha = 0$, these codes are linear codes over $\mathbb{Z}_4$, and when $\beta = 0$, they are 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 a binary code, which is not necessarily linear in general. 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 described in [33] and fully characterized in [7]. Also, in subsequent papers [6, 30, 31], $\mathbb{Z}_2\mathbb{Z}_4$-linear extended 1-perfect and Hadamard codes are studied and classified. Since there was not any symbolic software to work with $\mathbb{Z}_2\mathbb{Z}_4$-additive codes, the members of Combinatorics, Coding and Security Group (CCSG) have been developing a new package [5] in MAGMA that supports the basic facilities for these codes. The last version of this package and the manual with the description of all functions can be downloaded from the CCSG web page (`http://ccsg.uab.cat`) and GitHub (`https://github.com/merce-github/ZpAdditiveCodes`).

Linear codes over $\mathbb{Z}_{p^s}$, with p prime, were studied by Blake [3] and Shankar [34] in 1975 and 1979, respectively. Nevertheless, the study of codes over rings in general increased significantly after the publication of [21] on linear codes over $\mathbb{Z}_4$. Note that linear codes over $\mathbb{Z}_{p^s}$, also called $\mathbb{Z}_{p^s}$-additive codes, can be seen as a generalization of linear codes over $\mathbb{Z}_4$, when $p = 2$ and $s = 2$; and binary linear codes, when $p = 2$ and $s = 1$. Since a linear code C over $\mathbb{Z}_{p^s}$ of length n is a subgroup of $\mathbb{Z}_{p^s}^n$, it is isomorphic to an abelian structure $\mathbb{Z}_{p^s}^{t_1} \times \mathbb{Z}_{p^{s-1}}^{t_2} \times \cdots \times \mathbb{Z}_p^{t_s}$, so we say that C is of type $(n; t_1, \ldots, t_s)$. Note that $|C| = p^{st_1}p^{(s-1)t_2} \cdots p^{t_s}$. In MAGMA, each linear code over a ring has associated a unique generator matrix corresponding to the Howell form. The number of rows k in this unique generator matrix is called the pseudo-dimension of the code. It is not an invariant between equivalent codes, so does not provide structural information like the dimension of a linear code over a finite field. For linear codes over rings $\mathbb{Z}_{p^s}$, there also exists a generator matrix having minimum number of rows, that is, $t_1 + \cdots + t_s$ rows.

Again, since there is not any symbolic software to work with linear codes over $\mathbb{Z}_{p^s}$, the members of CCSG have been developing this new MAGMA package that supports the basic facilities for these codes. Specifically, it gen-

eralizes most of the known functions for linear codes over the ring $\mathbb{Z}_4$ to linear codes over $\mathbb{Z}_{p^s}$, 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. This package has been developed by Mercè Villanueva with the collaboration of Abdullah Irfan Basheer, Javier Esmoris, Guillermo Mosse, Noam von Rotberg, and Adrián Torres-Martín. The last version of this package and the manual with the description of all functions can be downloaded from the CCSG web page (`http://ccsg.uab.cat`) and GitHub (`https://github.com/merce-github/MagmaZpAdditiveCodes`).

1.2 The Generalized Gray Maps

Let p be a prime number. We denote by $\mathbb{F}_p$ or $\mathrm{GF}(p)$ the finite field with p elements, and by $\mathbb{Z}_{p^s}$ the ring of integer modulo p^s with $s \geq 1$.

The usual Gray map from $\mathbb{Z}_4$ to $\mathbb{F}_2^2$ is defined as $\phi(0) = (0,0)$, $\phi(1) = (0,1)$, $\phi(2) = (1,1)$ and $\phi(3) = (1,0)$ [21, 27]. There exist different generalizations of this Gray map, which go from $\mathbb{Z}_{p^s}$ to $\mathbb{F}_p^{p^{s-1}}$, with p prime [9, 13, 22, 24]. The one given in [9], by Carlet, is the map $\phi_s : \mathbb{Z}_{p^s} \to \mathbb{F}_p^{p^{s-1}}$ defined as follows:

$$\phi_s(u) = (u_{s-1}, \ldots, u_{s-1}) + (u_0, \ldots, u_{s-2})Y_{s-1}, \tag{1.1}$$

where $u \in \mathbb{Z}_{p^s}$, $[u_0, u_1, \ldots, u_{s-1}]_p$ is the p-ary expansion of u, that is, $u = \sum_{i=0}^{s-1} p^i u_i$ with $u_i \in \{0, 1, \ldots, p-1\}$, and Y_{s-1} is a matrix of size $(s-1) \times p^{s-1}$ whose columns are all the vectors in $\mathbb{F}_p^{s-1}$. Without loss of generality, we assume that the columns of Y_{s-1} are ordered in ascending order, by considering the elements of $\mathbb{F}_p^{s-1}$ as the p-ary expansions of the elements of $\mathbb{Z}_{p^{s-1}}$. Note that ϕ_1 is the identity map, and $(u_{s-1}, \ldots, u_{s-1})$ and $(u_0, \ldots, u_{s-2})Y_{s-1}$ are vectors over $\mathbb{F}_p$ of length p^{s-1}. The map ϕ_s can also be defined as follows:

$$\phi_s(u) = (u_0, \ldots, u_{s-2}, u_{s-1})M_{s-1},$$

where

$$M_{s-1} = \begin{pmatrix} Y_{s-1} \\ \mathbf{1} \end{pmatrix}. \tag{1.2}$$

Then, the image of ϕ_s can be seen as a linear code over $\mathbb{F}_p$ generated by M_{s-1}. Moreover, it is easy to check that it is a linear two-weight code of size p^s with nonzero weights $(p-1)p^{s-2}$ and p^{s-1}. Indeed, it is a linear generalized Hadamard code of length p^{s-1} over $\mathbb{F}_p$, which comes from a generalized Hadamard (GH) matrix $H(p, p^{s-2})$, known as the Sylvester Hadamard matrix [36]. This linear code is also known as the p-ary first order Reed-Muller code, as mentioned in [19].

A more general definition of the Gray map was given in [24] in terms of the elements of a generalized Hadamard matrix. A generalized Hadamard (GH) matrix $H(p,\lambda) = (h_{ij})$ of order $n = p\lambda$ over $\mathbb{F}_p$ is a $p\lambda \times p\lambda$ matrix with entries from $\mathbb{F}_p$ with the property that for every i, j, $1 \le i < j \le p\lambda$, each of the multisets $\{h_{is} - h_{js} : 1 \le s \le p\lambda\}$ contains every element of $\mathbb{F}_p$ exactly λ times [23]. An ordinary Hadamard matrix of order 4μ corresponds to a GH matrix $H(2,\lambda)$ over $\mathbb{F}_2$, where $\lambda = 2\mu$ [1]. Two GH matrices H_1 and H_2 of order n are said to be equivalent if one can be obtained from the other by a permutation of the rows and columns and adding the same element of $\mathbb{F}_p$ to all the coordinates in a row or in a column. We can always change the first row of a GH matrix into zeros and we obtain an equivalent GH matrix. Let H be a GH matrix with zeros in the first row, and let $\mathbf{h}_i$ be the ith row of H, for all $0 \le i \le p^{s-1} - 1$. Note that $\mathbf{h}_0 = \mathbf{0}$. Then, the Gray map $\phi_s : \mathbb{Z}_{p^s} \to \mathbb{F}_p^{p^{s-1}}$ given by H is defined as

$$\phi_s(u) = \begin{cases} \mathbf{h}_u, & \text{if } 0 \le u \le p^{s-1} - 1 \\ \mathbf{h}_{u - jp^{s-1}} + (j, \ldots, j), & \text{if } p^{s-1} \le u \le p^s - 1. \end{cases} \tag{1.3}$$

Note that the above Carlet's Gray map (1.1) is a particular case of this one satisfying $\sum \lambda_i \phi_s(2^i) = \phi_s(\sum \lambda_i 2^i)$ [17].

These generalized Gray maps ϕ_s are extended to a map $\Phi_s : \mathbb{Z}_{p^s}^n \to \mathbb{F}_p^{np^{s-1}}$ in the obvious way, that is, as the component-wise Gray map Φ_s.

`CarletGrayMap(p, s)`

Given a prime p and an integer $s \ge 1$, this function returns Carlet's generalized Gray map ϕ_s from $\mathbb{Z}_{p^s}$ to $\mathbb{F}_p^{p^{s-1}}$ as defined in (1.1).

`CarletGrayMap(p, [`$t_1, t_2, \ldots, t_s$`])`

Given a prime p and a sequence $[t_1, \ldots, t_s]$ of $s \ge 1$ nonnegative integers, this function returns a map from the $\mathbb{Z}_{p^s}$-submodule of $\mathbb{Z}_{p^s}^{t_1 + \cdots + t_s}$ isomorphic to $\mathbb{Z}_{p^s}^{t_1} \times \cdots \times \mathbb{Z}_p^{t_s}$ to the space $\mathbb{F}_p^k$, where $k = t_1 p^{s-1} + t_2 p^{s-2} + \cdots + t_{s-1} p + t_s$. In the first t_1 coordinates, Carlet's generalized Gray map ϕ_s from $\mathbb{Z}_{p^s}$ to $\mathbb{F}_p^{p^{s-1}}$ is considered; in the next t_2, Carlet's generalized Gray map ϕ_{s-1} from $\mathbb{Z}_{p^{s-1}}$ to $\mathbb{F}_p^{p^{s-2}}$; and so on, until the last t_s coordinates, where the identity map ϕ_1 from $\mathbb{Z}_p$ to $\mathbb{F}_p$ is considered. The map is also provided with an inverse function, since it is bijective.

Note that this map coincides with Carlet's generalized Gray map from the information space of a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ (as a $\mathbb{Z}_{p^s}$-submodule) to the information space of $\Phi_s(C)$.

`CarletGrayMap(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n, this function returns Carlet's generalized Gray map for C. That is, the map Φ_s from C to $\mathbb{F}_p^{np^{s-1}}$.

If the linear code C is over $\mathbb{Z}_4$, function `CarletGrayMap(C)` coincides with function `GrayMap(C)`, which works only for linear codes over $\mathbb{Z}_4$.

`CarletGrayMapImage(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n, this function returns the image of C under Carlet's generalized Gray map as a sequence of vectors in $\mathbb{F}_p^{np^{s-1}}$. As the resulting image may not be a linear code over $\mathbb{F}_p$, a sequence of vectors is returned rather than a code.

If the linear code C is over $\mathbb{Z}_4$, function `CarletGrayMapImage(C)` coincides with `GrayMapImage(C)`, which works only for linear codes over $\mathbb{Z}_4$.

`HasLinearCarletGrayMapImage(C)`

`AlgMethod` MonStgElt *Default*: `"Auto"`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n, this function returns `true` if and only if the image of C under Carlet's generalized Gray map is a linear code over $\mathbb{F}_p$. If so, the function also returns the image C_p as a linear code over $\mathbb{F}_p$, together with the bijection $\Phi_s : C \to C_p$.

The user can specify the method to be used by setting the parameter `AlgMethod` to `"BruteForce"`, `"StarProduct"` or `"StarProductMemory"`. The first one is based on computing the span of the Gray map image of C, and the other two on Theorem 4.13 given in [37], without considering some of the codewords of order p. `"StarProductMemory"` method does more computations than `"StarProduct"`, but it does not need to store any set of codewords. By default, `AlgMethod` is set to `"StarProduct"`. However, sometimes the brute force method can be faster, for example, when the image of C under Carlet's Gray map gives a linear code over $\mathbb{F}_p$, that is, when almost all pairs of codewords need to be checked in the default method. In cases where there is not enough memory to perform the default method, the option `"StarProductMemory"` can be used.

If the linear code C is over $\mathbb{Z}_4$, `HasLinearCarletGrayMapImage(C)` coincides with function `HasLinearGrayMapImage(C)`, which works only for linear codes over $\mathbb{Z}_4$.

Example H1E1

After defining a linear code C over $\mathbb{Z}_8$, the images of some codewords under Carlet's generalized Gray map are found. Moreover, it is shown that the image of C under this Gray map is not a linear code over GF(2).

```
> Z8 := Integers(8);
> C := LinearCode<Z8, 8 | [1,1,1,1,2,2,2,2],
                          [0,1,2,3,4,5,6,7],
                          [0,0,2,2,4,4,6,6] >;
> C;
(8, 256, 4) Linear Code over IntegerRing(8)
Generator matrix:
[1 0 1 0 2 1 2 1]
[0 1 0 1 0 1 0 1]
[0 0 2 2 4 4 6 6]
>
> V4 := VectorSpace(GF(2), 4);
> mapGray := CarletGrayMap(2, 3);
> mapGray(0);
(0 0 0 0)
> mapGray(1);
(0 1 0 1)
> mapGray(2);
(0 0 1 1)
> mapGray(3);
(0 1 1 0)
> mapGray(4); mapGray(4) eq mapGray(0) + V4![1,1,1,1];
(1 1 1 1)
true
> mapGray(5); mapGray(5) eq mapGray(1) + V4![1,1,1,1];
(1 0 1 0)
true
> mapGray(6); mapGray(6) eq mapGray(2) + V4![1,1,1,1];
(1 1 0 0)
true
> mapGray(7); mapGray(7) eq mapGray(3) + V4![1,1,1,1];
(1 0 0 1)
true

> mapGrayC := CarletGrayMap(C);
> C.1;
(1 0 1 0 2 1 2 1)
> mapGrayC(C.1);
(0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1)
> mapGrayC(C.1) eq VectorSpace(GF(2), 8*4)!&cat[Eltseq(mapGray(i)) : i in Eltseq(C.1)];
true

> HasLinearCarletGrayMapImage(C);
false 0 0
> mapGrayU := CarletGrayMap(UniverseCode(Z8, Length(C)));
> (mapGrayC(C.1) + mapGrayC(C.2)) @@ mapGrayU in C;
false
```

Now, a linear code D over $\mathbb{Z}_{27}$ of type $(10; 4, 1, 0)$, so of size $27^4 \cdot 9 = 4782969$, is defined. Due to the size of D, the default method `StarProduct` to determine Carlet's generalized

Gray map image does not work. In this case, we can use the option `StarProductMemory`.

```
> Z27 := Integers(27);
> D := LinearCode<Z27, 10 | [1, 0, 0, 0, 0, 19, 18, 14, 10, 24],
                            [0, 1, 0, 0, 1, 10, 23,  3,  8,  6],
                            [0, 0, 1, 0, 1, 14, 23, 11, 18,  4],
                            [0, 0, 0, 1, 2, 25, 20, 17,  2,  1],
                            [0, 0, 0, 0, 3, 21,  0,  9, 21,  0]>;
> #D;
4782969
> HasLinearCarletGrayMapImage(D);
System error: Out of memory.
All virtual memory has been exhausted so Magma cannot perform this statement.
> time HasLinearCarletGrayMapImage(D : AlgMethod := "StarProductMemory");
false 0 0
Time: 1077.031
```

Finally, a linear code E over $\mathbb{Z}_{27}$ is defined. In this case, Carlet's generalized Gray map image of E is a linear code over GF(3). Moreover, for this code, the brute force method is faster than the default method.

```
> E := LinearCode< Integers(27), 6 | [[1,0,3,7,18,1],
                                     [0,3,0,0,21,18],
                                     [0,0,9,0,0,0],
                                     [0,0,0,9,0,0]] >;
> time IsLinear, GrayCode := HasLinearCarletGrayMapImage(E);
Time: 3.813
> time IsLinear := HasLinearCarletGrayMapImage(E : AlgMethod := "BruteForce");
Time: 0.359
> IsLinear; Length(GrayCode); Dimension(GrayCode); Alphabet(GrayCode);
true
54
7
Finite field of size 3
```

`GrayMap(H)`

Given a generalized Hadamard matrix H over $\mathbb{F}_p$ of order p^{s-1}, for an integer $s > 1$, this function returns the generalized Gray map ϕ_s from $\mathbb{Z}_{p^s}$ to $\mathbb{F}_p^{p^{s-1}}$ given by H as defined in (1.3). The matrix must have zeros in the first row, but it does not need to be normalized. Matrix H can also be given as an ordinary Hadamard matrix with 1's and -1's. In this case, it is transformed into a binary matrix by swapping 1's for 0's and -1's for 1's.

Note that if H is the Sylvester Hadamard matrix, which is the matrix generated by all linear combinations of the rows of a matrix Y_{s-1} of size $(s-1) \times p^{s-1}$ whose columns are all the vectors in $\mathbb{F}_p^{s-1}$, then this map coincides with the one given by function `CarletGrayMap(p, s)`.

`GrayMap(C, H)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n and a generalized Hadamard matrix H over $\mathbb{F}_p$ of order p^{s-1}, for an integer $s > 1$, this function returns the generalized Gray map Φ_s from C to $\mathbb{F}_p^{np^{s-1}}$ given by H applied to each coordinate. The matrix must have zeros in the first row, but it does not need to be normalized. Matrix H can also be given as an ordinary Hadamard matrix with 1's and -1's. In this case, it is transformed into a binary matrix by swapping 1's for 0's and -1's for 1's.

Note that if H is the Sylvester Hadamard matrix, which is the matrix generated by all linear combinations of the rows of a matrix Y_{s-1} of size $(s-1) \times p^{s-1}$ whose columns are all the vectors in $\mathbb{F}_p^{s-1}$, then this map coincides with the one given by function `CarletGrayMap(C)`.

`GrayMapImage(C, H)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n and a generalized Hadamard matrix H over $\mathbb{F}_p$ of order p^{s-1}, for an integer $s > 1$, this function returns the image of C under the generalized Gray map Φ_s from C to $\mathbb{F}_p^{np^{s-1}}$ given by H applied to each coordinate. As the resulting image may not be a linear code over $\mathbb{F}_p$, a sequence of vectors in $\mathbb{F}_p^{np^{s-1}}$ is returned rather than a code. The matrix must have zeros in the first row, but it does not need to be normalized. Matrix H can also be given as an ordinary Hadamard matrix with 1's and -1's. In this case, it is transformed into a binary matrix by swapping 1's for 0's and -1's for 1's.

Note that if H is the Sylvester Hadamard matrix, which is the matrix generated by all linear combinations of the rows of a matrix Y_{s-1} of size $(s-1) \times p^{s-1}$ whose columns are all the vectors in $\mathbb{F}_p^{s-1}$, then this map coincides with the one given by function `CarletGrayMapImage(C)`.

`HasLinearGrayMapImage(C, H)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n and a generalized Hadamard matrix over $\mathbb{F}_p$ of order p^{s-1}, for an integer $s > 1$, this function returns `true` if and only if the image of C, under the generalized Gray map Φ_s from C to $\mathbb{F}_p^{np^{s-1}}$ given by H applied to each coordinate, is a linear code over $\mathbb{F}_p$. If so, the function also returns the image C_p as a linear code over $\mathbb{F}_p$, together with the bijection $\Phi_s : C \to C_p$. Matrix H can also be given as an ordinary Hadamard matrix with 1's and -1's. In this case, it is transformed into a binary matrix by swapping 1's for 0's and -1's for 1's.

Example H1E2

After defining the same linear code C over $\mathbb{Z}_8$ as in Example H1E1 and a generalized Hadamard matrix H over GF(2) of order 4, the images of some codewords under the generalized Gray map given by H are found. We check that this Gray map coincides with Carlet's Gray map. Moreover, it is shown that the image of C under this Gray map is not a linear code over GF(2).

```
> Z8 := Integers(8);
> C := LinearCode<Z8, 8 | [1,1,1,1,2,2,2,2],
                          [0,1,2,3,4,5,6,7],
                          [0,0,2,2,4,4,6,6] >;

> H := Matrix(GF(2), [[0,0,0,0],
                      [0,1,0,1],
                      [0,0,1,1],
                      [0,1,1,0]]);
> mapGrayH := GrayMap(H);
> mapGrayCarlet := CarletGrayMap(2, 3);
> HCarlet := Matrix([mapGrayCarlet(a) : a in [0..3]]);
> H eq HCarlet;
true
> a := Random(Z8);
> mapGrayH(a) eq mapGrayCarlet(a);
true

> mapGrayC := GrayMap(C, H);
> C.1;
(1 0 1 0 2 1 2 1)
> mapGrayC(C.1);
(0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 1 0 1 0 1)
> C.2;
(0 1 0 1 0 1 0 1)
> mapGrayC(C.2);
(0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 1)

> HasLinearGrayMapImage(C, H);
false 0 0
> Cbin := GrayMapImage(C, H);
> #Cbin eq #LinearCode(Matrix(Cbin));
false
```

Now, another generalized Hadamard matrix H over GF(2) of order 4 is consider. This one does not give the same map as Carlet's generalized Gray map. It is also checked that the image of C under this Gray map is not a linear code over GF(2).

```
> H := Matrix(GF(2), [[0,0,0,0],
                      [0,0,1,1],
                      [0,1,0,1],
                      [0,1,1,0]]);
```

```
> mapGrayH := GrayMap(H);
> H eq HCarlet;
false
> mapGrayH(2) eq mapGrayCarlet(2);
false
> mapGrayH(2) eq mapGrayCarlet(1);
true
> HasLinearGrayMapImage(C, H);
false 0 0
> Cbin := GrayMapImage(C, H);
> #Cbin eq #LinearCode(Matrix(Cbin));
false
> mapGrayC := GrayMap(C, H);
> mapGrayU := GrayMap(UniverseCode(Z8, Length(C)), H);
> (mapGrayC(C.1) + mapGrayC(C.2)) @@ mapGrayU in C;
false
```

Example H1E3

After defining another linear code C over $\mathbb{Z}_{2^s}$ and a generalized Hadamard matrix H of order 2^{s-1}, which is not the Sylvester Hadamard matrix, we check that the image of C under this Gray map is not a linear code over GF(2).

```
> C := ZpHadamardCode(2, [1,0,0,0,2]);
> HasLinearCarletGrayMapImage(C);
true [64, 7, 32] Linear Code over GF(2)
...
> Cbin := CarletGrayMapImage(C);
> #Cbin eq #LinearCode(Matrix(Cbin));
true

> D := HadamardDatabase();
> H := Matrix(D, 16, 2);
> HasLinearGrayMapImage(C, H);
false 0 0
> Cbin := GrayMapImage(C, H);
> #Cbin eq #LinearCode(Matrix(Cbin));
false
```

1.3 Random Codes over $\mathbb{Z}_{p^s}$

In order to generate random linear codes over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, we can use the function `RandomLinearCode(R, n, k)` available for any ring R in MAGMA [8, Chapter 164]. Indeed, when R is set to `Integers(p^s)`, this function returns a random linear code over $\mathbb{Z}_{p^s}$ of length n with pseudo-dimension k, that is, such that $t_1 + t_2 + \cdots + t_s \leq k$. However, due to its implementation, it tends to produce just codes of type $[k, 0, \ldots, 0]$. Unlike

this function, the ones described in this section allow us to generate random linear codes, only over $\mathbb{Z}_{p^s}$, of different types.

`RandomZpAdditiveCode(p, n, [`$t_1, t_2, \ldots, t_s$`])`

Given a prime p, a positive integer n, and a sequence $L = [t_1, \ldots, t_s]$ of $s \geq 2$ nonnegative integers, return a random linear code over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$. The function also returns the matrix used to generate the random code, which is in standard form, that is, of the form (1.5).

`RandomZpAdditiveCode(p, n, s, k)`

Given a prime p, and three positive integers n, s, and k such that $s \geq 2$ and $k \leq n$, return a random linear code over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ such that $t_1 + \cdots + t_s = k$. The function also returns the matrix used to generate the random code, which is in standard form, that is, of the form (1.5).

This function generates a random sequence $[t_1, \ldots, t_s]$ such that $t_1 + \cdots + t_s = k$, and then it uses `RandomZpAdditiveCode(p, n, [`$t_1, t_2, \ldots, t_s$`])`. Note that it does not use function `RandomLinearCode(Integers(p^s), n, k)` available in MAGMA [8].

Example H1E4

A random linear code C over $\mathbb{Z}_{3^3}$ of type $(16; 2, 3, 3)$ is generated. Then, it is checked that the matrix G is the generator matrix of C, and it is in standard form.

```
> C, G := RandomZpAdditiveCode(3, 16, [2,3,3]);
> IsStandardFormMatrix(G);
true
> LinearCode(G) eq C;
true
> ZpType(C);
[ 2, 3, 3 ]
```

Some random linear codes over $\mathbb{Z}_{2^3}$ are constructed to show the difference between function `RandomLinearCode(R, n, k)`, available for any ring R, and the function available only for rings $\mathbb{Z}_{p^s}$, `RandomZpAdditiveCode(p, n, s, k)`.

```
> C := RandomZpAdditiveCode(2, 16, 3, 8);
> ZpType(C);
[ 1, 3, 4 ]
> C := RandomLinearCode(Integers(2^3), 16, 8);
> ZpType(C);
[ 8, 0, 0 ]
```

1.4 Families of Codes over $\mathbb{Z}_{p^s}$

This section presents some constructions for linear codes over $\mathbb{Z}_{p^s}$ such that, after Carlet's generalized Gray map defined in Section 1.2, they are $\mathbb{Z}_{p^s}$-linear codes (not necessarily linear over $\mathbb{F}_p$) with the same parameters as some known families of linear codes over $\mathbb{F}_p$.

`ZpHadamardCode(p, [`$t_1, t_2, \ldots, t_s$`])`

Given a prime p and a sequence of nonnegative integers $[t_1, \ldots, t_s]$ with $t_1 \geq 1$ and $s \geq 2$, return a linear generalized Hadamard code over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ [2, 17], where $n = p^{m-s+1}$ and $m = (\sum_{i=1}^{s}(s-i+1)\cdot t_i) - 1$. Moreover, return a generator matrix $A^{t_1,\ldots,t_s}$ with $t_1 + \cdots + t_s$ rows constructed in a recursive way as follows. We start with $A^{1,0,\ldots,0} = (1)$. Then, if we have a matrix $A = A^{t_1,\ldots,t_s}$, for any $i \in \{1, \ldots, s\}$,

$$A^{t'_1,\ldots,t'_s} = \begin{pmatrix} A & A & \cdots & A \\ 0 \cdot \mathbf{p^{i-1}} & 1 \cdot \mathbf{p^{i-1}} & \cdots & (p^{s-i+1}-1) \cdot \mathbf{p^{i-1}} \end{pmatrix}, \qquad (1.4)$$

where $t'_j = t_j$ for $j \neq i$ and $t'_i = t_i + 1$, and $\mathbf{p^{i-1}}$ is the vector having the element p^{i-1} in all its coordinates. First, we add $t_1 - 1$ rows of order p^s, up to obtain $A^{t_1,0,\ldots,0}$; then t_2 rows of order p^{s-1} up to generate $A^{t_1,t_2,0,\ldots,0}$; and so on, until we add t_s rows of order p to achieve $A^{t_1,\ldots,t_s}$.

A generalized Hadamard code over $\mathbb{Z}_{p^s}$ of length p^{m-s+1} is a code over $\mathbb{Z}_{p^s}$ such that, after Carlet's generalized Gray map, defined in Section 1.2, gives a code over $\mathbb{F}_p$ (not necessarily linear) with the same parameters as a generalized Hadamard code over $\mathbb{F}_p$ of length p^m.

If $p = 2$ and $s = 2$, function `ZpHadamardCode(2, [`t_1, t_2`])` coincides with function `HadamardCodeZ4(`t_1`, `$t_2 + 2t_1 - 1$`)`.

Example H1E5

First, a linear Hadamard code C over $\mathbb{Z}_4$ of type $4^2 2^1$ is defined. It is checked that we can use either function `ZpHadamardCode(2, [`t_1, t_2`])` or `HadamardCodeZ4(`t_1`, `$t_2 + 2t_1 - 1$`)`. Note that its Gray map image is a binary Hadamard code, that is, a $(N, 2N, N/2)$ code, where $N = 2^m$. In this case, the obtained binary Hadamard code is linear.

```
> t1 := 2; t2 := 1;
> C := ZpHadamardCode(2, [t1, t2]);
> C eq HadamardCodeZ4(t1, t2 + 2*t1 - 1);
true

> Cbin := CarletGrayMapImage(C);
> N := OverDimension(Cbin[1]);
```

```
> minDistance := MinimumHomogeneousWeight(C);
> (#Cbin eq 2*N) and (minDistance eq N/2);
true
> HasLinearCarletGrayMapImage(C);
true [16, 5, 8] Linear Code over GF(2)
Generator matrix:
[1 0 0 0 0 1 1 1 0 1 1 1 1 0 0 0]
[0 1 0 0 1 0 1 1 0 1 0 0 1 0 1 1]
[0 0 1 0 1 1 0 1 0 0 1 0 1 1 0 1]
[0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0]
[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]
Mapping from: CodeLinRng: C to [16, 5, 8] Linear Code over GF(2) given by a rule
```

Now, a linear Hadamard code C over $\mathbb{Z}_{27}$ of type $(27; 1, 1, 1)$ is defined. The matrix G over $\mathbb{Z}_{27}$, used to generate C and obtained recursively, is also given. It is checked that its Gray map image is a generalized Hadamard code over $GF(3)$, which is a $(N, pN, (p-1)N/p)$ code, where $N = p^m$. In this case, the Hadamard code over $GF(3)$ is nonlinear.

```
> p := 3; t1 := 1; t2 := 1; t3 := 1;
> C, Gc := ZpHadamardCode(p, [t1, t2, t3]);
> Gc;
[ 0  0  0  0  0  0  0  0  0  9  9  9  9  9  9  9  9 18 18 18 18 18 18 18 18 18]
[ 0  3  6  9 12 15 18 21 24  0  3  6  9 12 15 18 21 24  0  3  6  9 12 15 18 21 24]
[ 1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]
> C eq LinearCode(Gc);
true
> ZpType(C) eq [t1,t2,t3];
true

> Cp := CarletGrayMapImage(C);
> N := OverDimension(Cp[1]);
> minDistance := MinimumHomogeneousWeight(C);
> (#Cp eq p*N) and (minDistance eq (p-1)*N/p);
true
> HasLinearCarletGrayMapImage(C);
false 0 0
```

`ZpSimplexAlphaCode(p, s, t)`

Given a prime p and two integers, $s \geq 2$ and $t \geq 1$, return the simplex alpha code S_t^α over $\mathbb{Z}_{p^s}$ of type $(n; t, 0, \ldots, 0)$ [18, 20], where $n = p^{st}$. Moreover, return a generator matrix G_t^α of S_t^α, which is constructed recursively as follows. We start with $G_1^\alpha = \begin{pmatrix} 0 & 1 & 2 & \cdots & p^s - 1 \end{pmatrix}$. Then,

$$G_t^\alpha = \begin{pmatrix} G_{t-1}^\alpha & G_{t-1}^\alpha & G_{t-1}^\alpha & \cdots & G_{t-1}^\alpha \\ \mathbf{0} & \mathbf{1} & \mathbf{2} & \cdots & \mathbf{p^s - 1} \end{pmatrix}$$

for $t \geq 2$, where $\mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots, \mathbf{p^s - 1}$ are the vectors having the elements $0, 1, 2, \ldots, p^s - 1$ from $\mathbb{Z}_{p^s}$ in all its coordinates, respectively.

After Carlet's generalized Gray map, defined in Section 1.2, $\Phi_s(S_t^\alpha)$ is a simplex code over $\mathbb{F}_p$ of type α, that is, a $(p^{s(t+1)-1}, p^{st}, p^{s(t+1)-2}(p-1))$ code over $\mathbb{F}_p$ having all codewords of the same Hamming weight equal to $p^{s(t+1)-2}(p-1)$, except the all-zero codeword.

If $p = 2$ and $s = 2$, function `ZpSimplexAlphaCode(2, 2, t)` coincides with function `SimplexAlphaCodeZ4(t)`.

`ZpSimplexBetaCode(p, s, t)`

Given a prime p and two integers, $s \geq 2$ and $t \geq 1$, return the simplex beta code S_t^β over $\mathbb{Z}_{p^s}$ of type $(n; t, 0, \ldots, 0)$ [18, 20], where $n = p^{(t-1)(s-1)}(p^t - 1)/(p-1)$. Moreover, return a generator matrix G_t^β of S_t^β, which is constructed recursively as follows. We start with $G_1^\beta = (1)$. Then,

$$G_t^\beta = \begin{pmatrix} G_{t-1}^\alpha & G_{t-1}^\beta & G_{t-1}^\beta & G_{t-1}^\beta & \cdots & G_{t-1}^\beta \\ \mathbf{1} & \mathbf{0} & \mathbf{p} & 2\mathbf{p} & \cdots & (p^{s-1}-1)\mathbf{p} \end{pmatrix}$$

for $t \geq 2$, where $\mathbf{1}$, $\mathbf{0}$, and $\mathbf{p}$ are the vectors having the elements $1, 0$, and p in all its coordinates, respectively.

After Carlet's generalized Gray map, defined in Section 1.2, $\Phi_s(S_t^\beta)$ is a simplex code over $\mathbb{F}_p$ of type β, that is, a $(p^{st-t}(p^t-1)/(p-1), p^{st}, p^{st-t-1}(p^t-1))$ code over $\mathbb{F}_p$ with $p^t(p^{(s-1)t}-1)$ codewords of Hamming weight $p^{st-t-1}(p^t-1)$, p^t-1 codewords of Hamming weight p^{st-1}, and the all-zero codeword.

If $p = 2$ and $s = 2$, function `ZpSimplexBetaCode(2, 2, t)` may not coincide with function `SimplexBetaCodeZ4(t)`, but the obtained codes are monomially equivalent (one can be obtained from the other by permuting coordinates and, if necessary, changing the sign of certain coordinates).

Example H1E6

A simplex alpha code S_2^α over $\mathbb{Z}_8$ of type $(64; 2, 0, 0)$ is defined. The relation of S_2^α with a linear Hadamard code over the same ring is shown. It is also checked that its Gray map image has the correct parameters: length, size, and minimum distance.

```
> p := 2; s := 3; t := 2;
> Zps := Integers(p^s);
> H, GH := ZpHadamardCode(p, [t+1] cat [0^^(s-1)]);

> Salpha, GSalpha := ZpSimplexAlphaCode(p, s, t);
> Salpha eq LinearCode(GSalpha);
true
> GSalpha eq RemoveRow(GH, 1);
true
> _, A := ZpSimplexAlphaCode(p, s, t-1);
```

```
> R1 := HorizontalJoin([A^^(p^s)]);
> R2 := Matrix(Zps, [&cat[[i^^Ncols(A)] : i in [0..p^s-1]]]);
> GSalpha eq VerticalJoin(R1, R2);
true
> Length(Salpha) eq p^(s*t);
true
> ZpType(Salpha) eq [t] cat [0^^(s-1)];
true

> Salpha_p := CarletGrayMapImage(Salpha);
> N := OverDimension(Salpha_p[1]);
> minDistance := MinimumHomogeneousWeight(Salpha);
> N eq Length(Salpha)*p^(s-1);
true
> #Salpha eq p^(s*t);
true
> minDistance eq (p-1)*N/p;
true
```

A simplex beta code S_2^{β} over $\mathbb{Z}_8$ of type $(12; 2, 0, 0)$ is defined. It is also checked that its Gray map image has the correct parameters: length, size, and minimum distance.

```
> Sbeta, GSbeta := ZpSimplexBetaCode(p, s, t);
> Sbeta eq LinearCode(GSbeta);
true
> _, A := ZpSimplexAlphaCode(p, s, t-1);
> _, B := ZpSimplexBetaCode(p, s, t-1);
> R1 := HorizontalJoin(A, HorizontalJoin([B^^(p^(s-1))]));
> R2 := HorizontalJoin(Matrix(Zps, [[1^^Ncols(A)]]),
                       Matrix(Zps, [[p*i : i in [0..p^(s-1)-1]]]));
> GSbeta eq VerticalJoin(R1, R2);
true
> Length(Sbeta) eq p^((s-1)*(t-1))*(p^t-1)/(p-1);
true
> ZpType(Sbeta) eq ZpType(Salpha);
true

> Sbeta_p := CarletGrayMapImage(Sbeta);
> N := OverDimension(Sbeta_p[1]);
> minDistance := MinimumHomogeneousWeight(Sbeta);
> N eq Length(Sbeta)*p^(s-1);
true
> #Sbeta eq p^(s*t);
true
> minDistance eq (p-1)*N/p;
true

> weightDistribution := {* Weight(c) : c in Sbeta_p *};
> Multiplicity(weightDistribution, p^(s*t-t-1)*(p^t-1)) eq p^t*(p^((s-1)*t)-1);
```

```
true
> Multiplicity(weightDistribution, p^(s*t-1)) eq p^t-1;
true
> Multiplicity(weightDistribution, 0) eq 1;
true
```

Example H1E7

A simplex alpha and a simplex beta code over $\mathbb{Z}_4$ are constructed by using the above functions. Then, they are compared with the ones obtained by using `SimplexAlphaCodeZ4(t)` and `SimplexBetaCodeZ4(t)`, which work only for linear codes over $\mathbb{Z}_4$.

```
> ZpSimplexAlphaCode(2, 2, 3) eq SimplexAlphaCodeZ4(3);
true

> ZpSimplexBetaCode(2, 2, 2) eq SimplexBetaCodeZ4(2);
true
> S3, G3 := ZpSimplexBetaCode(2, 2, 3);
> S3Z4 := SimplexBetaCodeZ4(3);
> S3 eq S3Z4;
false
> perm := Sym(28)!(17,21)(19,22)(23,27)(25,28);
> G3b := G3^perm;
> MultiplyColumn(~G3b, -1, 20);
> MultiplyColumn(~G3b, -1, 26);
> S3Z4 eq LinearCode(G3b);
true
```

`ZpMacDonaldAlphaCode(p, s, t, u)`

Given a prime p and three integers $s \geq 2$, $t \geq 1$, and $1 \leq u < t$, return the MacDonald alpha code $M^{\alpha}_{t,u}$ over $\mathbb{Z}_{p^s}$ of type $(n; t, 0, \ldots, 0)$ [18]. Moreover, return a generator matrix $G^{\alpha}_{t,u}$ of $M^{\alpha}_{t,u}$, constructed from a generator matrix G^{α}_{t} of the simplex alpha code over $\mathbb{Z}_{p^s}$ as follows:

$$G^{\alpha}_{t,u} = \left(\begin{array}{cc} G^{\alpha}_{t} & \backslash \quad \frac{G^{\alpha}_{u}}{\mathbf{0}} \end{array} \right),$$

where $(A \backslash B)$ denotes the matrix formed by all the columns of matrix A which are not in matrix B, and $\mathbf{0}$ is the $(t-u) \times p^{su}$ zero matrix.

After Carlet's generalized Gray map, defined in Section 1.2, $\Phi_s(M^{\alpha}_{t,u})$ is a $(p^{s(t+1)-1} - p^{s(u+1)-1}, p^{st})$ MacDonald code over $\mathbb{F}_p$ of type α.

`ZpMacDonaldBetaCode(p, s, t, u)`

Given a primer p and three integers $s \geq 2$, $t \geq 1$, and $1 \leq u < t$, return the MacDonald beta code $M^{\beta}_{t,u}$ over $\mathbb{Z}_{p^s}$ of type $(n; t, 0, \ldots, 0)$ [18]. Moreover, return a generator matrix $G^{\beta}_{t,u}$ of $M^{\beta}_{t,u}$, constructed from a generator matrix G^{β}_{t} of the simplex beta code over $\mathbb{Z}_{p^s}$ as follows:

$$G^{\beta}_{t,u} = \left(\begin{array}{ccc} G^{\beta}_{t} & \setminus & \begin{matrix} G^{\beta}_{u} \\ \mathbf{0} \end{matrix} \end{array} \right),$$

where $(A \setminus B)$ denotes the matrix obtained from the matrix A by deleting the columns of the matrix B, and $\mathbf{0}$ is a zero matrix.

After Carlet's generalized Gray map, defined in Section 1.2, $\Phi_s(M^{\beta}_{t,u})$ is a $(p^{st-t}(p^t-1)/(p-1) - p^{su-u}(p^u-1)/(p-1), p^{st})$ MacDonald code over $\mathbb{F}_p$ of type β.

Example H1E8

A MacDonald alpha code $M^{\alpha}_{2,1}$ over $\mathbb{Z}_8$ of type $(56; 2, 0, 0)$ is defined. It is checked that its Gray map image has the correct length and number of codewords.

```
> p := 2; s := 3; t := 2; u := 1;
> Malpha, GMalpha := ZpMacDonaldAlphaCode(p, s, t, u);
> Malpha eq LinearCode(GMalpha);
true
> _, A := ZpSimplexAlphaCode(p, s, t);
> GMalpha eq ColumnSubmatrixRange(A, (p^(s*u))+1, (p^s)^t);
true
> ZpType(Malpha) eq [t] cat [0^^(s-1)];
true

> N := OverDimension(CarletGrayMap(Malpha)(Random(Malpha)));
> N eq Length(Malpha)*p^(s-1);
true
> N eq p^(s*(t+1)-1)-p^(s*(u+1)-1);
true
> #Malpha eq p^(s*t);
true
```

A MacDonald beta code $M^{\beta}_{2,1}$ over $\mathbb{Z}_8$ of type $(11; 2, 0, 0)$ is defined. It is checked that its Gray map image has the correct length and number of codewords.

```
> Mbeta, GMbeta := ZpMacDonaldBetaCode(p, s, t, u);
> Mbeta eq LinearCode(GMbeta);
true
> ZpType(Mbeta) eq ZpType(Malpha);
true
```

```
> N := OverDimension(CarletGrayMap(Mbeta)(Random(Mbeta)));
> N eq Length(Mbeta)*p^(s-1);
true
> N eq p^(s*t-t)*(p^t-1)/(p-1)-p^(s*u-u)*(p^u-1)/(p-1);
true
> #Mbeta eq p^(s*t);
true
```

1.5 Derived Codes over $\mathbb{F}_p$

As well as the image of a linear code C over $\mathbb{Z}_{p^s}$ under the Gray map (see section The Generalized Gray Maps), there are also other associated canonical codes over $\mathbb{F}_p$. They are known as the residue code and the ith torsion code, for $i \in \{1, \ldots, s\}$. The residue code coincides with the 1st torsion code, and the ith torsion code is a subcode of the jth torsion code for all $j \geq i$.

`ZpResidueCode(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, return the code over $\mathbb{F}_p$ formed by taking each codeword in C modulo p. This is known as the residue code of C. If C is in standard form, that is, it is generated by a matrix of the form (1.5), then the residue code is the linear code over $\mathbb{F}_p$ generated by the following matrix:

$$\left(\begin{array}{ccccccc} Id_{t_1} & \bar{A}_{0,1} & \bar{A}_{0,2} & \bar{A}_{0,3} & \cdots & \bar{A}_{0,s} \end{array} \right),$$

where $\bar{A}_{0,j}$ is $A_{0,j}$ after applying modulo p to each entry, $j \in \{1, \ldots, s\}$.

If the linear code C is over $\mathbb{Z}_4$, function `ZpResidueCode(C)` coincides with function `BinaryResidueCode(C)`.

`ZpTorsionCode(C, i)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and an integer $i \in \{1, \ldots, s\}$, return the code over $\mathbb{F}_p$ formed by the vectors in $\{\bar{v} : p^{i-1}v \in C\}$, where $\bar{v}$ is the vector v modulo p. This is a code of length n and dimension $t_1 + \ldots + t_i$, which is known as the ith torsion code of C. If C is in standard form, that is, it is generated by a matrix of the form (1.5), then the ith torsion code is the linear code over $\mathbb{F}_p$ generated by the following matrix:

$$\begin{pmatrix} Id_{t_1} & \bar{A}_{0,1} & \bar{A}_{0,2} & \cdots & \bar{A}_{0,i-1} & \cdots & \bar{A}_{0,s} \\ \mathbf{0} & Id_{t_2} & \bar{A}_{1,2} & \cdots & \bar{A}_{1,i-1} & \cdots & \bar{A}_{1,s} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & Id_{t_i} & \cdots & \bar{A}_{i-1,s} \end{pmatrix},$$

where $\bar{A}_{k,j}$ is matrix $A_{k,j}$ after applying modulo p to each entry, for $k \in \{0, \ldots, i-1\}$ and $j \in \{1, \ldots, s\}$.

If the linear code C is over $\mathbb{Z}_4$, function `ZpTorsionCode(C, 2)` coincides with function `BinaryTorsionCode(C)`.

Example H1E9

First, a random linear code over $\mathbb{Z}_4$ is defined, and the residue and torsion codes given by functions `ZpResidueCode(C)` and `ZpTorsionCode(C, i)` are compared with the ones obtained by using functions `BinaryResidueCode(C)` and `BinaryTorsionCode(C)`, which works only for linear codes over $\mathbb{Z}_4$.

```
> C := RandomLinearCode(Integers(4), 10, 8);
> ZpResidueCode(C) eq BinaryResidueCode(C);
true
> ZpTorsionCode(C, 1) eq ZpResidueCode(C);
true
> ZpTorsionCode(C, 2) eq BinaryTorsionCode(C);
true
```

A linear code over $\mathbb{Z}_8$ of type $(8; 2, 1, 1)$ is defined, and the residue and torsion codes of this code are also constructed. Then, some of the properties of these codes are checked.

```
> G := Matrix(Integers(8), [[1,0,7,6,5,4,3,2],
                            [0,1,2,3,4,5,6,7],
                            [0,0,2,2,4,4,6,6],
                            [0,0,0,4,0,4,0,4]]);
> C := LinearCode(G);
> Gdiv := Matrix(Integers(8), [[1,0,7,6,5,4,3,2],
                               [0,1,2,3,4,5,6,7],
                               [0,0,1,1,2,2,3,3],
                               [0,0,0,1,0,1,0,1]]);
> G eq DiagonalMatrix(Integers(8), [1,1,2,4])*Gdiv;
true
> Gbin := Matrix(GF(2), Gdiv);

> CRes := ZpResidueCode(C);
> CRes eq LinearCode(Matrix(GF(2), GeneratorMatrix(C)));
true

> CTor1 := ZpTorsionCode(C, 1);
> CTor2 := ZpTorsionCode(C, 2);
> CTor3 := ZpTorsionCode(C, 3);
```

```
> CTor1 eq CRes;
true
> CTor1 eq LinearCode(RowSubmatrix(Gbin, 1, ZpType(C)[1]));
true
> CTor2 eq LinearCode(RowSubmatrix(Gbin, 1, &+ZpType(C)[1..2]));
true
> CTor3 eq LinearCode(RowSubmatrix(Gbin, 1, &+ZpType(C)[1..3]));
true
> (CTor1 subset CTor2) and (CTor2 subset CTor3);
true
```

1.6 The Code Space and Dual Space

`ZpMinRowsGeneratorMatrix(C)`

A generator matrix for the linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, with the minimum number of rows, that is with $t_1 + \cdots + t_s$ rows: t_1 rows of order p^s, t_2 of order p^{s-1}, and so on until t_s rows of order p. It also returns the sequence $[t_1, \ldots, t_s]$ and a permutation transforming C into a permutation-equivalent code with generator matrix in standard form.

If C is a linear code over $\mathbb{Z}_4$ of type $(n; t_1, t_2)$, to obtain a generator matrix with minimum number of rows, function `MinRowsGeneratorMatrix(C)` can also be used. However, instead of returning the sequence $[t_1, t_2]$, it returns t_2, t_1, and the generator matrix may be different.

`ZpDual(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, return the dual code D of C. The dual code consists of all codewords in the $\mathbb{Z}_{p^s}$-space $V = \mathbb{Z}_{p^s}^n$ which are orthogonal to all codewords of C. In particular, the dual code D is of type $(n; n - t_1 - t_2 - \cdots - t_s, t_s, \ldots, t_2)$.

This function creates the generator matrix of D using a specific known structure based on the generator matrix of C. This construction improves the computation time with respect to the generic function `Dual(C)` for codes over finite rings.

If C is over $\mathbb{Z}_4$, function `ZpDual(C)` coincides with function `DualZ4(C)`, but the former may perform less efficiently in general.

`ZpMinRowsParityCheckMatrix(C)`

A parity check matrix for the linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, with the minimum number of rows, that is, with $n - t_1$ rows. It also returns the sequence $[n - t_1 - t_2 - \cdots - t_s, t_s, \ldots, t_2]$ and a permutation transforming $C^\perp$ into a permutation-equivalent code with generator matrix in standard form.

This function should be faster for most codes over $\mathbb{Z}_{p^s}$ than the general function `ParityCheckMatrix(C)` for codes over finite rings. Another parity check matrix for the code C can be obtained as the generator matrix of the dual of C with the minimum number of rows, that is, as `ZpMinRowsGeneratorMatrix(ZpDual(C))`.

If C is a linear code over $\mathbb{Z}_4$ of type $(n; t_1, t_2)$, then `MinRowsParityCheck` Matrix(C) can also be used to obtain a parity check matrix with minimum number of rows. However, only the matrix is returned, which may not coincide with the one given by `ZpMinRowsGeneratorMatrix(C)`.

Example H1E10

For a random linear code C over $\mathbb{Z}_{3^{10}}$, the dual code is computed by using function `Dual(C)` and the new function `ZpDual(C)`, which is only applicable for linear codes over $\mathbb{Z}_{p^s}$ with p prime, and is usually faster than the former one. It is also checked the relation between a generator matrix of C and of $C^\perp$, and how to obtain the type of the dual code.

```
> C := RandomZpAdditiveCode(3, 1000, [2^^10]);

> time D := Dual(C);
Time: 12.875
> time Dp := ZpDual(C);
Time: 0.234
> D eq Dp;
true

> G := GeneratorMatrix(C);
> H := GeneratorMatrix(D);
> IsZero(G * Transpose(H));
true

> ZpTypeDual(C) eq ZpType(ZpDual(C));
true
```

Example H1E11

Again, for a random linear code C over $\mathbb{Z}_{3^{10}}$, the relation between a generator matrix of C and of $C^\perp$, both having minimum number of rows, is checked. Different ways of computing a parity check matrix for C are also shown.

```
> C := RandomZpAdditiveCode(3, 1000, [2^^10]);

> G := ZpMinRowsGeneratorMatrix(C);
> H := ZpMinRowsParityCheckMatrix(C);
> IsZero(G * Transpose(H));
true
> LinearCode(G) eq C;
true
> LinearCode(H) eq ZpDual(C);
true

> Nrows(G) eq &+ZpType(C);
true
> Nrows(H) eq &+ZpTypeDual(C);
true

> time H := ZpMinRowsParityCheckMatrix(C);
Time: 0.016
> time H2 := ZpMinRowsGeneratorMatrix(ZpDual(C));
Time: 6.297
> time H3 := ParityCheckMatrix(C);
Time: 31.422
> H eq H2;
false
> H2 eq H3;
false
> H eq H3;
false
> LinearCode(H2) eq ZpDual(C);
true
> LinearCode(H3) eq ZpDual(C);
true
```

Example H1E12

A linear Hadamard code over $\mathbb{Z}_4$ is constructed, and the outputs of functions `ZpDual(C)`, `ZpMinRowsGeneratorMatrix(C)` and `ZpMinRowsParityCheckMatrix(C)` are compared with the ones obtained by using `DualZ4(C)`, `MinRowsGeneratorMatrix(C)` and `MinRowsParity CheckMatrix(C)`, respectively, which work only for linear codes over $\mathbb{Z}_4$.

```
> C := ZpHadamardCode(2, [3,1]);

> G_Zp, type := ZpMinRowsGeneratorMatrix(C);
> G_Z4, t2, t1 := MinRowsGeneratorMatrix(C);
> G_Zp eq G_Z4;
false
> LinearCode(G_Zp) eq LinearCode(G_Z4);
true
> type eq [t1, t2];
```

```
true

> H_Zp := ZpMinRowsParityCheckMatrix(C);
> H_Z4 := MinRowsParityCheckMatrix(C);
> H_Zp eq H_Z4;
false
> LinearCode(H_Zp) eq LinearCode(H_Z4);
true

> ZpDual(C) eq DualZ4(C);
true
```

1.7 The Standard Form

A linear code over $\mathbb{Z}_{p^s}$ is in standard form if it has a generator matrix G of the form

$$\begin{pmatrix} Id_{t_1} & A_{0,1} & A_{0,2} & A_{0,3} & \cdots & \cdots & A_{0,s} \\ \mathbf{0} & pId_{t_2} & pA_{1,2} & pA_{1,3} & \cdots & \cdots & pA_{1,s} \\ \mathbf{0} & \mathbf{0} & p^2Id_{t_3} & p^2A_{2,3} & \cdots & \cdots & p^2A_{2,s} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \ddots & \ddots & & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} & p^{s-1}Id_{t_s} & p^{s-1}A_{s-1,s} \end{pmatrix}, \tag{1.5}$$

where Id_k is the identity matrix of size $k \times k$, and $A_{i,j}$ are matrices over $\mathbb{Z}_{p^s}$. We also say that a matrix of the form (1.5) is in standard form. Any linear code over $\mathbb{Z}_{p^s}$ is permutation-equivalent to a code S which is in standard form. Furthermore, the integers $t_1, t_2, \ldots, t_s$, defined above, are unique [13].

Note that given a matrix in standard form, by performing linear combinations of their rows, we can obtain another matrix G', which is also in standard form, such that the submatrices $p^iA_{i,j}$, $i < j$, have all entries in $\{0, 1, \ldots, p^j - 1\} \subseteq \mathbb{Z}_{p^s}$. We say that a matrix of this form, or a linear code with a generator matrix of this form, is in reduced standard form.

From a generator matrix of S in standard form, it is possible to compute more efficiently a parity check matrix of S, that is, a generator matrix of its dual code, as shown in [16]. This method is used in function `ZpStandardFormDual(C)` to obtain a parity check matrix of S. It is also used in function `ZpMinRowsParityCheckMatrix(C)` to compute a parity check matrix for any linear code over $\mathbb{Z}_{p^s}$, not necessarily in standard form.

`ZpStandardForm(C)`

`IsReducedStandardForm` BoolElt *Default*: `false`

Given a linear code C over $\mathbb{Z}_{p^s}$, return a permutation-equivalent code S in standard form, together with the corresponding isomorphism from C onto S. It also returns the generator matrix in standard form used to generate the code S and the permutation x such that $C\hat{}x = S$. MAGMA returns one of the many codes in standard form which is isomorphic to C (the same code is returned each time).

The parameter `IsReducedStandardForm` specifies whether the generator matrix is given as a matrix in reduced standard form. The default value is `false`. If it is set to `true`, the function returns a generator matrix which is in reduced standard form.

If C is a linear code over $\mathbb{Z}_4$, the first two output parameters coincide with the ones given by the function `StandardForm(C)`, and the last two parameters with the first and forth ones given by `StandardFormInfo(C)`.

`IsStandardFormMatrix(G)`

`IsReducedStandardForm` BOOLELT *Default*: `false`

Given a matrix G over $\mathbb{Z}_{p^s}$, return `true` if and only if G is in standard form, that is, it is of the form (1.5).

The parameter `IsReducedStandardForm` is set to `false` by default. If it is set to `true`, the function returns `true` if and only if G is in reduced standard form.

`ZpStandardFormDual(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$, return the dual of a permutation-equivalent code S in standard form, together with the corresponding isomorphism from the dual of C onto the dual of S. It also returns the parity check matrix used to generate the dual code of S and the permutation x such that $(C^\perp)\hat{}x = S^\perp$. MAGMA returns one of the many codes which is isomorphic to $C^\perp$ (the same code is returned each time).

If C is a linear code over $\mathbb{Z}_4$, the first two output parameters coincide with the ones given by the function `StandardFormDual(C)`.

Example H1E13

```
> C := ZpHadamardCode(3, [1,0,2]);
> S, f, Gs, perm := ZpStandardForm(C);
> S;
(9, 243, 6) Linear Code over IntegerRing(27)
Generator matrix:
[ 1  1  1  1  1  1  1  1  1]
[ 0  9  0 18  9 18  0  9 18]
[ 0  0  9  0  9  9 18 18 18]
```

```
> (Domain(f) eq C) and (Codomain(f) eq S);
true
> f(Random(C)) in S;
true
> Gs eq GeneratorMatrix(C)^perm;
true
> Gs eq ZpMinRowsGeneratorMatrix(C)^perm;
true
> Gs eq GeneratorMatrix(S);
true
> IsStandardFormMatrix(GeneratorMatrix(C));
false
> IsStandardFormMatrix(Gs);
true

> Ds, fDual, Hs, permDual := ZpStandardFormDual(C);
> (Domain(fDual) eq ZpDual(C)) and (Codomain(fDual) eq Ds);
true
> fDual(Random(ZpDual(C))) in Ds;
true
> Hs eq ZpMinRowsParityCheckMatrix(C)^permDual;
true
> perm eq permDual;
true
> (Ds eq LinearCode(Hs)) and (Ds eq ZpDual(S));
true
> IsStandardFormMatrix(Hs);
false
> IsZero(Hs * Transpose(Gs));
true
```

1.8 Invariants

`ZpPseudoDimension(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, t_2, \ldots, t_s)$, return the value $st_1 + (s-1)t_2 + \cdots + t_s$. Note that $|C| = p^{st_1+(s-1)t_2+\cdots+t_s}$. Function `PseudoDimension(C)`, for linear codes over rings in general, return the number of generators of the linear code C, that is, $t_1 + t_2 + \cdots + t_s$.

`ZpInformationRate(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, t_2, \ldots, t_s)$, return the information rate of the code, that is, the ratio $(st_1 + (s-1)t_2 + \cdots + t_s)/sn$.

ZpType(C)

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n, return the type of the code, that is, the unique sequence $[t_1, \ldots, t_s]$ such that the code, as a subgroup of $\mathbb{Z}_{p^s}^n$, is isomorphic to $\mathbb{Z}_{p^s}^{t_1} \times \mathbb{Z}_{p^{s-1}}^{t_2} \times \cdots \times \mathbb{Z}_{p^2}^{t_{s-1}} \times \mathbb{Z}_p^{t_s}$.

ZpTypeDual(C)

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, return the type of the dual code of C, that is, the sequence $[n - t_1 - t_2 - \cdots - t_s, t_s, \ldots, t_2]$.

Example H1E14

```
> type := [1,2,3,1]; n := 20;
> C := RandomZpAdditiveCode(3, n, type);
> PseudoDimension(C) eq &+type;
true
> ZpPseudoDimension(C) eq Log(3, #C);
true
> ZpInformationRate(C) eq ZpPseudoDimension(C)/(n*#type);
true
> ZpType(C) eq type;
true
> ZpTypeDual(C) eq [n-&+type] cat Reverse(type[2..#type]);
true
> ZpTypeDual(C) eq ZpType(ZpDual(C));
true

> type := [1,0,1,0,2];
> C := ZpHadamardCode(2, [1,0,1,0,2]);
> PseudoDimension(C) eq &+type;
true
> ZpPseudoDimension(C) eq Log(2, #C);
true
> ZpInformationRate(C) eq ZpPseudoDimension(C)/(Length(C)*#type);
true
> ZpType(C) eq type;
true
> ZpTypeDual(C) eq [Length(C)-&+type] cat Reverse(type[2..#type]);
true
> ZpTypeDual(C) eq ZpType(ZpDual(C));
true
```

1.9 Coset Representatives

`CosetRepresentatives(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n, with ambient space $V = \mathbb{Z}_{p^s}^n$, 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 set of coset representatives $\{c_0, c_1, \ldots, c_t\}$ satisfies that c_0 is the zero codeword and $V = \bigcup_{i=0}^{t} \left(C + c_i\right)$. Note that this function is only applicable when V and C are small.

`CosetRepresentatives(C, S)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of length n, and a subcode S over $\mathbb{Z}_{p^s}$ 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 set of coset representatives $\{c_0, c_1, \ldots, c_t\}$ satisfies that c_0 is the zero codeword and $C = \bigcup_{i=0}^{t} \left(S + c_i\right)$. The function also returns a second set containing the images of the coset representatives $\{c_0, c_1, \ldots, c_t\}$ by Carlet's generalized Gray map. Note that this function is only applicable when S and C are small.

Example H1E15

```
> C := LinearCode<Integers(27), 4 | [[1,1,2,20],
>                                   [0,3,0,12],
>                                   [0,0,9,18]]>;
> L := CosetRepresentatives(C);
> Set(RSpace(Integers(27),4)) eq {v+ci : v in Set(C), ci in L};
true
>
> S := LinearCode<Integers(27), 4 | [[1,1,2,20],
>                                   [0,3,0,12]]>;
> S subset C;
true
> L := CosetRepresentatives(C, S);
> Set(C) eq {v+ci : v in Set(S), ci in L};
true
```

1.10 The Homogeneous Weight Distribution

For elements of $\mathbb{Z}_{p^s}$ with p prime and $s \geq 2$, we consider the following metric defined in [10], and also used in [19, 41], known as the homogeneous weight:

$$wt^*(x) = \begin{cases} 0 & \text{if } x = 0, \\ p^{s-1} & \text{if } x \in p^{s-1}\mathbb{Z}_{p^s} \setminus \{0\}, \\ (p-1)p^{s-2} & \text{otherwise.} \end{cases} \tag{1.6}$$

Note that it corresponds to the Hamming weight of $\phi_s(x)$, where ϕ_s is the generalized Gray map defined in Section 1.2. The homogeneous weight of a vector $u = (u_1, u_2, \ldots, u_n) \in \mathbb{Z}_{p^s}^n$ is defined as $wt^*(u) = \sum_{j=1}^n wt^*(u_j) \in \mathbb{Z}_{p^s}$; and the homogeneous distance between two vectors $u, v \in \mathbb{Z}_{p^s}^n$ is defined as $d^*(u, v) = wt^*(u - v)$.

Given a linear code over $\mathbb{Z}_{p^s}$, the minimum homogeneous distance of C is $d^*(C) = \min\{d^*(u, v) : u, v \in C, u \neq v\}$. For linear codes over $\mathbb{Z}_{p^s}$, homogeneous weight and homogeneous distance distributions coincide (in particular, minimum homogeneous weight and minimum homogeneous distance coincide). Thus, $d^*(C) = wt^*(C) = \min\{wt^*(u) : u \in C, u \neq \mathbf{0}\}$.

`HomogeneousWeight(a)`

The homogeneous weight of the element $a \in \mathbb{Z}_{p^s}$ with p prime and $s \geq 2$.

If C is over $\mathbb{Z}_4$, this function coincides with function `LeeWeight(a)`.

`HomogeneousWeight(v)`

The homogeneous weight of the vector $v \in \mathbb{Z}_{p^s}^n$ with p prime and $s \geq 2$.

If C is over $\mathbb{Z}_4$, this function coincides with function `LeeWeight(v)`.

`HomogeneousDistance(u, v)`

The homogeneous distance between the vectors u and v, where $u, v \in \mathbb{Z}_{p^s}^n$ with p prime and $s \geq 2$. This is defined to be the homogeneous weight of $u - v$.

If C is over $\mathbb{Z}_4$, this function coincides with function `LeeDistance(u, v)`.

`MinimumHomogeneousWeight(C)`

`MinimumHomogeneousDistance(C)`

The minimum homogeneous weight of the linear code C over $\mathbb{Z}_{p^s}$ with p prime and $s \geq 2$.

If C is over $\mathbb{Z}_4$, these functions coincide with functions `MinumumLeeWeight(C)` and `MinumumLeeDistance(C)`, but the former may perform less efficiently because they are implemented just by using a brute force approach.

`HomogeneousWeightDistribution(C)`

The homogeneous weight distribution of the linear code C over $\mathbb{Z}_{p^s}$ with p prime and $s \geq 2$. The distribution is returned in the form of a sequence of tuples, where the ith tuple contains the ith homogeneous weight, w_i say, and the number of codewords having weight w_i.

This function coincides with function `LeeWeightDistribution(C)` for linear codes C over $\mathbb{Z}_4$.

`DualHomogeneousWeightDistribution(C)`

The homogeneous weight distribution of the dual of the linear code C over $\mathbb{Z}_{p^s}$ with p prime and $s \geq 2$ (see `HomogeneousWeightDistribution`). The distribution is returned in the form of a sequence of tuples, where the ith tuple contains the ith homogeneous weight, w_i say, and the number of codewords having weight w_i.

This function coincides with function `DualLeeWeightDistribution(C)` for linear codes C over $\mathbb{Z}_4$.

Example H1E16

The minimum homogeneous weight and weight distribution are computed for some linear codes over $\mathbb{Z}_{27}$: the linear generalized Hadamard code of type $(81; 2, 0, 1)$, the simplex alpha code S_2^α, the simplex beta code S_2^β, the MacDonald alpha code $M_{2,1}^\alpha$, and the MacDonald beta code $M_{2,1}^\beta$, described in Section 1.4.

```
> p := 3; s := 3; t := 2; u := 1;

> C := ZpHadamardCode(p, [2,0,1]);
> N := OverDimension(CarletGrayMap(C)(Random(C)));
> d := MinimumHomogeneousWeight(C);
> d eq (p-1)*N/p;
true
> d eq Min([HomogeneousWeight(c) : c in C | c ne C!0]);
true
> HomogeneousWeightDistribution(C);
[ <0, 1>, <486, 2184>, <729, 2> ]

> C := ZpSimplexAlphaCode(p, s, t);
> N := OverDimension(CarletGrayMap(C)(Random(C)));
> d := MinimumHomogeneousWeight(C);
> d eq (p-1)*N/p;
true
> HomogeneousWeightDistribution(C);
[ <0, 1>, <4374, 728> ]

> C := ZpSimplexBetaCode(p, s, t);
> N := OverDimension(CarletGrayMap(C)(Random(C)));
```

```
> d := MinimumHomogeneousWeight(C);
> d eq (p-1)*N/p;
true
> HomogeneousWeightDistribution(C);
[ <0, 1>, <216, 720>, <243, 8> ]

> C := ZpMacDonaldAlphaCode(p, s, t, u);
> MinimumHomogeneousWeight(C);
4212
> HomogeneousWeightDistribution(C);
[ <0, 1>, <4212, 702>, <4374, 26> ]

> C := ZpMacDonaldBetaCode(p, s, t, u);
> MinimumHomogeneousWeight(C);
207
> HomogeneousWeightDistribution(C);
[ <0, 1>, <207, 48>, <210, 648>, <216, 24>, <234, 6>, <243, 2> ]
```

Example H1E17

A linear Hadamard code over $\mathbb{Z}_4$ is constructed, and the outputs of some of the functions described in this section are compared with the ones obtained by using `LeeWeight(v)`, `MinimumLeeWeight(C)`, `LeeWeightDistribution(C)`, and `DualLeeWeightDistribution(C)`, which work only for linear codes over $\mathbb{Z}_4$. Note that the homogeneous weight coincides with the Lee weight for vectors over $\mathbb{Z}_4$.

```
> C := ZpHadamardCode(2, [2,1]);
> v := Random(C);
> HomogeneousWeight(v) eq LeeWeight(v);
true
> MinimumHomogeneousWeight(C) eq MinimumLeeWeight(C);
true
> HomogeneousWeightDistribution(C) eq LeeWeightDistribution(C);
true
> DualHomogeneousWeightDistribution(C) eq DualLeeWeightDistribution(C);
true
```

Another linear Hadamard code is constructed to show that function `MinimumLeeWeight(C)` is usually faster than function `MinimumHomogeneousWeight(C)`, since the latter has been implemented in the first version of the package just by using a brute force approach.

```
> C := ZpHadamardCode(2, [5,2]);
> time MinimumHomogeneousWeight(C);
1024
Time: 9.609
> time MinimumLeeWeight(C);
1024
Time: 3.188
```

1.11 Information Space and Information Sets

Let C be a linear code over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$. Let $C_I = \{v_I : v \in C\}$ and v_I is the vector v restricted to the subset $I \subseteq \{1, \ldots, n\}$ of coordinate positions. An ordered set $I = \{i_1, \ldots, i_{t_1+\cdots+t_s}\} \subseteq \{1, \ldots, n\}$ of $t_1 + \cdots + t_s$ coordinate positions is said to be an information set for C if $|C_I| = |C| = (p^s)^{t_1}(p^{s-1})^{t_2} \cdots p^{t_s}$. If the elements of an information set I are ordered in such a way that, for any $j \in \{1, \ldots, s\}$, we have that $|C_{\{i_1,\ldots,i_{t_1+\cdots+t_j}\}}| = (p^s)^{t_1}(p^{s-1})^{t_2} \cdots (p^{s-j+1})^{t_j}$, then it can be seen that the set $\Phi(I)$, defined as

$$\begin{aligned}\Phi(I) =& \Phi^{(1)}(\{i_1, \ldots, i_{t_1}\}) \cup \Phi^{(2)}(\{i_{t_1+1}, \ldots, i_{t_1+t_2}\}) \cup \\ & \cdots \cup \Phi^{(s)}(\{i_{t_1+\cdots+t_{s-1}+1}, \ldots, i_{t_1+\cdots+t_s}\}),\end{aligned} \tag{1.7}$$

where

$$\begin{aligned}\Phi^{(j)}(I) = \cup_{i \in I}\{& p^{s-1}(i-1)+1, \\ & p^{s-1}(i-1)+p^{j-1}+1, \\ & p^{s-1}(i-1)+p^{j-1+1}+1, \\ & p^{s-1}(i-1)+p^{j-1+2}+1, \\ & \ldots, \\ & p^{s-1}(i-1)+p^{s-2}+1\},\end{aligned}$$

is an information set for the corresponding code $C_p = \Phi_s(C)$ [38], where Φ_s is Carlet's generalized Gray map defined in Section 1.2. Note that $s - 2 - (j-1) = s - j - 1$, hence $\Phi^{(j)}(I)$ has $s - j + 1$ coordinate positions for each element in I.

The space of information vectors for C is $V = \mathbb{Z}_{p^s}^{t_1} \times p\mathbb{Z}_{p^s}^{t_2} \times \cdots \times p^{s-1}\mathbb{Z}_{p^s}^{t_s}$, that is, a $\mathbb{Z}_{p^s}$-submodule of $\mathbb{Z}_{p^s}^{t_1+\cdots+t_s}$ whose first t_1 coordinates are of order p^s, the next t_2 coordinates of order p^{s-1}, and so on, until the last t_s coordinates of order p. Since $p^{j-1}\mathbb{Z}_{p^s}$ is isomorphic to $\mathbb{Z}_{p^{s-j+1}}$ for all $j \in \{1, \ldots, s\}$, V is isomorphic to $\overline{V} = \mathbb{Z}_{p^s}^{t_1} \times \mathbb{Z}_{p^{s-1}}^{t_2} \times \cdots \times \mathbb{Z}_p^{t_s}$. Indeed, $(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_s) \in V$ if and only if $(\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2, \ldots, \bar{\mathbf{b}}_s) \in \overline{V}$ with $\bar{\mathbf{b}}_i = \mathbf{b}_i/p^{i-1}$. Let G be a generator matrix of C in standard form, that is, as shown in (1.5). We can define an encoding map from V to C as follows:

$$\begin{aligned} f : V &\longrightarrow C \\ (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_s) &\mapsto (\bar{\mathbf{b}}_1, \bar{\mathbf{b}}_2, \ldots, \bar{\mathbf{b}}_s)G. \end{aligned} \tag{1.8}$$

Note that since G is in standard form, we have that $J = \{1, \ldots, t_1 + \cdots + t_s\} \subseteq \{1, \ldots, n\}$ is an information set for C, and $J_p = \Phi(J) \subseteq \{1, \ldots, np^{s-1}\}$, as

defined in (1.7), is an information set for $C_p = \Phi_s(C)$. Let $\Phi_s|_{J_p}$ be the projection of the image of Φ_s onto the coordinates from the set J_p. Note that if Φ_s is applied to V, then J_p is seen as a subset of $\{1, \ldots, (t_1 + \cdots + t_s)p^{s-1}\}$. Then, we denote by f_p the encoding map for C_p such that diagram (1.9) commutes. This encoding f_p is defined over the space of information vectors for C_p, which is the k-dimensional vector space over $\mathbb{F}_p$, denoted by $V_p = \mathbb{F}_p^k$, where $k = st_1 + (s-1)t_2 + \ldots, t_s$. Note that neither the encoding f is necessarily systematic with respect to J, nor f_p is with respect to $J_p = \Phi(J)$. Recall that a map f (or f_p) is systematic if $f(v)_J = v$ for all $v \in V$ (resp. $f_p(v_p)_{J_p} = v$ for all $v_p \in V_p$).

$$\begin{array}{ccc} V & \xrightarrow{f} & C \\ \Big\downarrow{\scriptstyle \Phi_s|_{J_p}} & & \Big\downarrow{\scriptstyle \Phi_s} \\ V_p & \xrightarrow{f_p} & C_p \end{array} \tag{1.9}$$

Let s, t be two integers such that $s \geq t \geq 1$. We define the function $\psi_{s,t} : \mathbb{Z}_{p^s} \longrightarrow \mathbb{Z}_{p^t}$ as follows. Let $[u_0, \ldots, u_{s-1}]_p$ be the p-ary expansion of $u \in \mathbb{Z}_{p^s}$, that is, $u = \sum_{i=0}^{s-1} u_i p^i$. Then, $\psi_{s,t}(u) = \sum_{j=0}^{t-1} u_{s-t+j} p^j \in \mathbb{Z}_{p^t}$, that is, the element of $\mathbb{Z}_{p^t}$ such that its p-ary expansion is $[u_{s-t}, \ldots, u_{s-1}]_p$. Note that if $t = s$, then $\psi_{s,s}(u) = u$. We denote by $\Psi_{s,t}$ its extension coordinate-wise. Note that we can also see $\psi_{s,t}(u)$ as an element of $\mathbb{Z}_{p^s}$, since $s \geq t$, by considering $\psi_{s,t}(u)$ as one of the elements of $\{0, \ldots, p^t - 1\} \subseteq \mathbb{Z}_{p^s}$. Now, let us define the function $\sigma : \overline{V} \longrightarrow \overline{V}$ as $\sigma(\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_s) = (\mathbf{b}'_1, \mathbf{b}'_2, \ldots, \mathbf{b}'_s)$, where the vectors $\mathbf{b}'_j \in \mathbb{Z}_{p^{s-j+1}}^{t_j}$, for $j \in \{1, \ldots, s\}$, are given by

$$\begin{aligned}
\mathbf{b}'_1 &= \mathbf{b}_1, \\
\mathbf{b}'_2 &= \mathbf{b}_2 - \Psi_{s,s-1}(\mathbf{b}'_1 A_{0,1}), \\
\mathbf{b}'_3 &= \mathbf{b}_3 - \Psi_{s,s-2}(\mathbf{b}'_1 A_{0,2} + p\mathbf{b}'_2 A_{1,2}), \\
&\vdots \\
\mathbf{b}'_j &= \mathbf{b}_j - \Psi_{s,s-j+1}(\mathbf{b}'_1 A_{0,j-1} + p\mathbf{b}'_2 A_{1,j-1} + \cdots + p^{j-2}\mathbf{b}'_{j-1} A_{j-2,j-1}), \\
&\vdots \\
\mathbf{b}'_s &= \mathbf{b}_s - \Psi_{s,1}(\mathbf{b}'_1 A_{0,s-1} + p\mathbf{b}'_2 A_{1,s-1} + \cdots + p^{s-2}\mathbf{b}'_{s-1} A_{s-2,s-1}),
\end{aligned}$$

and $A_{i,j}$ are the submatrices of matrix G in standard form (1.5). Now, by using the map σ, we can define another encoding map from V to C as follows:

$$\begin{aligned} f : V &\longrightarrow C \\ (\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_s) &\mapsto \sigma(\frac{\mathbf{b}_1}{p^0}, \frac{\mathbf{b}_2}{p}, \ldots, \frac{\mathbf{b}_s}{p^{s-1}})G. \end{aligned} \tag{1.10}$$

By using this map, the map f_p such that diagram (1.9) commutes is indeed systematic with respect to $J_p = \Phi(J)$, as it is proven in [38].

`ZpInformationSpace(C)`

`IsSystematicEncoding` BOOLELT *Default*: `true`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, return $V = \mathbb{Z}_{p^s}^{t_1} \times p\mathbb{Z}_{p^s}^{t_2} \times \cdots \times p^{s-1}\mathbb{Z}_p^{t_s}$, that is, the space of information vectors for C. Note that V is a $\mathbb{Z}_{p^s}$-submodule of $\mathbb{Z}_{p^s}^{t_1+\cdots+t_s}$ whose first t_1 coordinates of V are of order p^s, the next t_2 coordinates of order p^{s-1}, and so on, until the last t_s coordinates of order p. The function also returns the space of information vectors for the corresponding code $C_p = \Phi_s(C)$ over $\mathbb{F}_p$, where Φ_s is Carlet's generalized Gray map; that is, the k-dimensional vector space over $\mathbb{F}_p$, $V_p = \mathbb{F}_p^k$, where $k = st_1 + (s-1)t_2 + \cdots + t_s$. Finally, for the encoding process, it returns two isomorphisms f and f_p from these spaces of information vectors, V and V_p, onto C and C_p, respectively.

The map f is given as a bijective map from V to C. Nevertheless, f_p is given as an injective map from V_p to $\mathbb{F}_p^{np^{s-1}}$, where np^{s-1} is the length of C_p, having the inverse only defined for the elements in $C_p \subseteq \mathbb{F}_p^{np^{s-1}}$. These two maps are related to each other in the sense that $\Phi_s \circ f = f_p \circ \Phi_s|_{J_p}$, where $\Phi_s|_{J_p}$ denotes the projection to the set of k coordinates $J_p = \Phi(J) \subseteq \{1, \ldots, (t_1 + \cdots + t_s)p^{s-1}\}$ as defined in (1.7), for the set $J = \{1, \ldots, t_1 + \cdots + t_s\}$. That is, diagram (1.9) commutes.

The parameter `IsSystematicEncoding` specifies whether the map f_p corresponds to a systematic encoding for the code C_p or not. It is set to `true` by default. In this case, it returns a systematic encoding f_p with respect to the information set I_p given by function `ZpInformationSet(C)`. Indeed, f_p is such that diagram (1.9) commutes for the encoding f given in (1.10). Otherwise, it returns an encoding f_p, which may not be systematic. In particular, f_p is such that diagram (1.9) commutes for the encoding f which corresponds to multiplying by the generator matrix given by function `ZpMinRowsGeneratorMatrix(C)` as shown in (1.8).

If C is a linear code over $\mathbb{Z}_4$ of type $(n; t_1, t_2)$, then `InformationSpace(C)` can also be used. The second output parameter coincides in both functions. However, `InformationSpace(C)` instead of returning $\mathbb{Z}_4^{t_1} \times p\mathbb{Z}_2^{t_2}$, it returns $p\mathbb{Z}_2^{t_2} \times \mathbb{Z}_4^{t_1}$, and both isomorphisms f and f_p from the spaces of information vectors, V and V_p, onto C and C_p, are also different.

Example H1E18

The function is applied to a linear code C over $\mathbb{Z}_{27}$ of type $(5; 1, 1, 1)$. Two information vectors $(17, 21, 18)$ and $\Phi_3(17, 21, 18) = (1, 0, 0, 2, 0, 2)$ for C and $C_p = \Phi_3(C)$, respectively,

are encoded by using the given encodings f and f_p. It is also checked that $\Phi_3 \circ f = f_p \circ \Phi_3|_{J_p}$, where $J_p = \Phi_3(\{1,2,3\}) = \{1,2,4,10,13,19\}$.

```
> C := LinearCode<Integers(27), 5 | [[1,2,5,8,6],
>                                   [0,3,6,12,21],
>                                   [0,0,9,9,18]]>;
> V, Vp, f, fp := ZpInformationSpace(C);

> (#V eq #C) and (#Vp eq #C);
true
> Set([f(i) : i in V]) eq Set(C);
true
> Set([fp(i) : i in Vp]) eq Set(CarletGrayMapImage(C));
true

> i := V![17,21,18];
> c := f(i);
> c;
(17 22 25 25  0)
> c in C;
true

> ip := Vp![1,0,0,2,0,2];
> cp := fp(ip);
> cp;
(1 0 2 0 2 1 2 1 0 2 0 1 0 1 2 1 2 0 2 0 1 1 2 0 0 1 2 2 0
 1 1 2 0 0 1 2 0 0 0 0 0 0 0 0 0)
> cp in CarletGrayMapImage(C);
true

> mapGrayC := CarletGrayMap(C);
> mapGrayV := CarletGrayMap(3, ZpType(C));
> Set(Vp) eq {mapGrayV(v) : v in V};
true
> ip eq mapGrayV(i);
true
> cp eq mapGrayC(c);
true
> [mapGrayC(f(v)) : v in V] eq [fp(mapGrayV(v)) : v in V];
true
```

`ZpInformationSet(C)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, return an information set $I \subseteq \{1, \ldots, n\}$ for C. Moreover, it also returns an information set I_p for the corresponding code $C_p = \Phi_s(C)$ over $\mathbb{F}_p$, where Φ_s is Carlet's generalized Gray map. These information sets I and I_p are returned as a sequence of $t_1 + t_2 + \cdots + t_s$ and $st_1 + (s-1)t_2 + \cdots + t_s$ coordinate positions, respectively. The information set I_p coincides with $\Phi(I)$ as defined in (1.7), and the encoding map f_p given by function `ZpInformationSpace(C)` is systematic with respect to this information set I_p.

An information set I for C is an ordered set of $t_1 + t_2 + \cdots + t_s$ coordinate positions such that $|C_I| = |C|$, where $C_I = \{v_I : v \in C\}$ and v_I is the vector v restricted to the I coordinates. An information set I_p for C_p is an ordered set of $st_1 + (s-1)t_2 + \cdots + t_s$ coordinate positions such that $|(C_p)_{I_p}| = |C_p| = |C|$.

If C is a linear code over $\mathbb{Z}_4$, then function `InformationSet(C)` can also be used even though both output parameters may be different.

`IsZpInformationSet(C, I)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and a sequence $I \subseteq \{1, \ldots, n\}$ or $I \subseteq \{1, \ldots, np^{s-1}\}$, return `true` if and only if $I \subseteq \{1, \ldots, n\}$ is an information set for C. This function also returns another boolean, which is `true` if an only if $I \subseteq \{1, \ldots, np^{s-1}\}$ is an information set for the corresponding code $C_p = \Phi_s(C)$ over $\mathbb{F}_p$, where Φ_s is Carlet's generalized Gray map.

An information set I for C is an ordered set of $t_1 + t_2 + \cdots + t_s$ coordinate positions such that $|C_I| = |C|$, where $C_I = \{v_I : v \in C\}$ and v_I is the vector v restricted to the I coordinates. An information set I_p for C_p is an ordered set of $st_1 + (s-1)t_2 + \cdots + t_s$ coordinate positions such that $|(C_p)_{I_p}| = |C_p| = |C|$.

If C is over $\mathbb{Z}_4$, function `IsZpInformationSet(C, I)` coincides with function `IsInformationSet(C, I)`, which works only for linear codes over $\mathbb{Z}_4$, but the former may perform less efficiently when $I \subseteq \{1, \ldots, 2n\}$.

Example H1E19

The functions are applied to the same linear code over $\mathbb{Z}_{27}$ of type $(5; 1, 1, 1)$ as in Example H1E18. It is checked that the given sets are information sets for C and $C_p = \Phi_s(C)$, and that the encoding f_p given by function `ZpInformationSpace(C)` is systematic with respect to the information set I_p given by function `ZpInformationSet(C)`.

```
> C := LinearCode<Integers(27), 5 | [[1,2,5,8,6],
```

```
>                                     [0,3,6,12,21],
>                                     [0,0,9,9,18]]>;
> V, Vp, f, fp := ZpInformationSpace(C);

> I, Ip := ZpInformationSet(C);
> I;
[ 1, 2, 3 ]
> Ip;
[ 1, 2, 4, 10, 13, 19 ]

> #PunctureCode(C, {1..5} diff Set(I)) eq #C;
true
> Cp := CarletGrayMapImage(C);
> #{Eltseq(cp)[Ip] : cp in Cp} eq #Cp;
true

> IsZpInformationSet(C, I);
true false
> IsZpInformationSet(C, Ip);
false true
> IsZpInformationSet(C, [1,2,3,4,5,6]);
false false

> ip := Vp![1,0,0,2,0,2];
> cp := fp(ip);
> Vp![cp[i] : i in Ip] eq ip;
true
> #[ip : ip in Vp | Vp![fp(ip)[i] : i in Ip] eq ip ] eq #Vp;
true
```

1.12 Encoding and Systematic Encoding

The reader is referred to the introduction of Section 1.11 for the description of the encoding functions considered in this section, since they coincide with the ones provided by function `ZpInformationSpace(C)`.

`Encoding(C, v)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and an element v from the space $V_p = \mathbb{F}_p^k$, where $k = st_1 + (s-1)t_2 + \cdots + t_s$, of information vectors for $C_p = \Phi_s(C)$, where Φ_s is Carlet's generalized Gray map (or an element v from the space $V = \mathbb{Z}_{p^s}^{t_1} \times p\mathbb{Z}_{p^s}^{t_2} \times \cdots \times p^{s-1}\mathbb{Z}_p^{t_s}$ of information vectors for C, given as a vector of length $t_1 + \cdots + t_s$ over $\mathbb{Z}_{p^s}$), return the codewords $c \in C$ and $c_p = \Phi_s(c) \in C_p$ corresponding to an encoding of $(\Phi_s|_{J_p})^{-1}(v) \in V$ and $v \in V_p$, respectively (or $v \in V$ and $(\Phi_s|_{J_p})(v) \in V_p$ if v is given from V), where $\Phi_s|_{J_p}$ is the projection of the image of Φ_s onto the coordinates from $J_p = \Phi(\{1, \ldots, t_1 + \cdots + t_s\})$.

This encoding for C, denoted by f, corresponds to multiplying an element in V by the generator matrix given by function `ZpMinRowsGenerator Matrix(C)`, and the encoding for C_p corresponds to the map f_p that makes diagram (1.9) commute for the encoding f. Note that $f((\Phi_s|_{J_p})^{-1}(v)) = c$ and $f_p(v) = c_p$ (or $f(v) = c$ and $f_p((\Phi_s|_{J_p})(v)) = c_p$ if v is given from V). The encodings f and f_p coincide with the ones provided by function `ZpInformationSpace(C : IsSystematicEncoding := false)`.

`Encoding(C, L)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and a sequence L of elements from $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, return a sequence of codewords from C, and also the corresponding sequence of codewords from $C_p = \Phi_s(C)$, given by an injective map, which corresponds to an encoding of the elements of L. The encodings for C and C_p coincide with the ones provided by function `ZpInformationSpace(C : IsSystematicEncoding := false)`.

Depending on the elements of L, the function automatically selects the appropriate encoding over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, by using function `Encoding(C, v)`. If it detects that L contains more than one information vector, then it computes the encoding for each one of them and returns a sequence of codewords. If it is necessary, zeros are added at the end of the sequence L to complete the last information vector before encoding.

`Encoding(C, M)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and a matrix M over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, returns a sequence of codewords from C, and also the corresponding sequence of codewords from $C_p = \Phi_s(C)$, given by an injective map, which corresponds to an encoding of the rows of M. The encodings for C and C_p coincide with the ones provided by function `ZpInformationSpace(C : IsSystematicEncoding := false)`.

The matrix can contain either information vectors over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$. Depending on the length of the rows of M and its entries, the function automatically selects the appropriate encoding over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, by using function `Encoding(C, v)`.

Example H1E20

The functions are applied to the same linear code over $\mathbb{Z}_{27}$ of type $(5; 1, 1, 1)$ as in Example H1E18. Some information vectors for C and $C_p = \Phi_3(C)$ are encoded by using the above functions. It is also checked that the enccoding for C corresponds to multiplying by the generator matrix given by function `ZpMinRowsGeneratorMatrix(C)`, as shown in (1.8).

```
> C := LinearCode<Integers(27), 5 | [[1,2,5,8,6],
>                                   [0,3,6,12,21],
>                                   [0,0,9,9,18]]>;
> V, Vp, f, fp := ZpInformationSpace(C : IsSystematicEncoding := false);
> mapGrayC := CarletGrayMap(C);
> mapGrayV := CarletGrayMap(3, ZpType(C));

> i := V![21,12,18];
> ip := Vp![2,2,0,1,2,2];
> ip eq mapGrayV(i);
true
> Encoding(C, ip) eq Encoding(C, i);
true
> c, cp := Encoding(C, i);
> (c eq f(i)) and (cp eq fp(ip));
true
> cp eq mapGrayC(c);
true

> ibar := [i[1], i[2] div 3, i[3] div 9];
> c eq Vector(ibar) * ZpMinRowsGeneratorMatrix(C);
true

> L := Eltseq(i) cat Eltseq(V![13,3,9]);
> Encoding(C, L);
[
    (21  0 12 18  3),
    (13  2 26 17  9)
]
[
    (2 2 2 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 1 1 1 2 2 2 0 0 0 2 2 2 2 2 2 2 2 2 0 0
        0 1 1 1 2 2 2),
    (1 2 0 2 0 1 0 1 2 0 2 1 0 2 1 0 2 1 2 1 0 1 0 2 0 2 1 1 0 2 0 2 1 2 1 0 1 1
        1 1 1 1 1 1 1)
]
> Encoding(C,L)[1] eq Encoding(C, i);
true
```

```
> M := Matrix(Integers(27), 3, L);
> Encoding(C, M) eq Encoding(C, L);
true
```

`SystematicEncoding(C, v)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and an element v from the space $V_p = \mathbb{F}_p^k$, where $k = st_1 + (s-1)t_2 + \cdots + t_s$, of information vectors for $C_p = \Phi_s(C)$, where Φ_s is Carlet's generalized Gray map (or an element v from the space $V = \mathbb{Z}_{p^s}^{t_1} \times p\mathbb{Z}_{p^s}^{t_2} \times \cdots \times p^{s-1}\mathbb{Z}_p^{t_s}$ of information vectors for C, given as a vector of length $t_1 + \cdots + t_s$ over $\mathbb{Z}_{p^s}$), return the codewords $c \in C$ and $c_p = \Phi_s(c) \in C_p$ corresponding to an encoding of $(\Phi_s|_{J_p})^{-1}(v) \in V$ and $v \in V_p$, respectively (or $v \in V$ and $(\Phi_s|_{J_p})(v) \in V_p$ if v is given from V), where $\Phi_s|_{J_p}$ is the projection of the image of Φ_s onto the coordinates from $J_p = \Phi(\{1, \ldots, t_1 + \cdots + t_s\})$. Unlike function `Encoding(C, v)`, in this case, the given encoding for C_p is systematic with respect to the information set I_p given by function `ZpInformationSet(C)`.

This encoding for C_p, denoted by f_p, corresponds to the systematic encoding with respect to the information set I_p given by function `ZpInformationSet(C)`, and the encoding for C corresponds to the map f that makes diagram (1.9) commute for the encoding f_p. Note that $f((\Phi_s|_{J_p})^{-1}(v)) = c$ and $f_p(v) = c_p$ (or $f(v) = c$ and $f_p((\Phi_s|_{J_p})(v)) = c_p$ if v is given from V). Moreover, since f_p is systematic, $(c_p)_{I_p} = v$ (or $(c_p)_{I_p} = (\Phi_s|_{J_p})(v)$ if v is given from V). The encodings f and f_p coincide with the ones provided by function `ZpInformationSpace(C)`.

`SystematicEncoding(C, L)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and a sequence L of elements from $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, return a sequence of codewords from C, and also the corresponding sequence of codewords from $C_p = \Phi_s(C)$. The encodings for C and C_p coincide with the ones provided by function `ZpInformationSpace(C)`. Unlike function `Encoding(C, L)`, in this case, the given encoding for C_p is systematic with respect to the information set I_p given by function `ZpInformationSet(C)`.

Depending on the elements of L, the function automatically selects the appropriate encoding over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, by using function `SystematicEncoding(C, v)`. If it detects that L contains more than one information vector, then it computes the encoding for each one of them and returns a sequence of codewords. If it is necessary, zeros are added at the end of the sequence L to complete the last information vector before encoding.

`SystematicEncoding(C, M)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and a matrix M over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, returns a sequence of codewords from C, and also the corresponding sequence of codewords from $C_p = \Phi_s(C)$. The encodings for C and C_p coincide with the ones provided by function `ZpInformationSpace(C)`. Unlike function `Encoding(C, M)`, in this case, the given encoding for C_p is systematic with respect to the information set I_p given by function `ZpInformationSet(C)`.

The matrix can contain either information vectors over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$. Depending on the length of the rows of M and its entries, the function automatically selects the appropriate encoding over $\mathbb{Z}_{p^s}$ or $\mathbb{F}_p$, by using function `SystematicEncoding(C, v)`.

Example H1E21

The functions are applied to the same linear code over $\mathbb{Z}_{27}$ of type $(5; 1, 1, 1)$ as in Example H1E18. Some information vectors for C and $C_p = \Phi_3(C)$ are encoded by using the above functions. It is also checked that the enccoding for C_p is systematic with respect to the information set given by function `ZpInformationSet(C)`.

```
> C := LinearCode<Integers(27), 5 | [[1,2,5,8,6],
>                                   [0,3,6,12,21],
>                                   [0,0,9,9,18]]>;
> V, Vp, f, fp := ZpInformationSpace(C);

> I, Ip := ZpInformationSet(C);
> I;
[ 1, 2, 3 ]
> Ip;
[ 1, 2, 4, 10, 13, 19 ]
> mapGrayC := CarletGrayMap(C);
> mapGrayV := CarletGrayMap(3, ZpType(C));
> [fp(vp) : vp in Vp] eq [mapGrayC(f(vp @@ mapGrayV)) : vp in Vp ];
true
> [vp : vp in Vp] eq [ Vector([fp(vp)[i] : i in Ip ]) : vp in Vp ];
true

> i := V![21,12,18];
> ip := Vp![2,2,0,1,2,2];
> ip eq mapGrayV(i);
true
> SystematicEncoding(C, ip) eq SystematicEncoding(C, i);
true
> c, cp := SystematicEncoding(C, i);
> (c eq f(i)) and (cp eq fp(ip));
true
> cp eq mapGrayC(c);
```

```
true
> ip eq Vp![cp[j] : j in Ip];
true

> L := Eltseq(i) cat Eltseq(V![13,3,9]);
> SystematicEncoding(C, L);
[
    (21 12 18 21 24),
    (13  5 14 11 21)
]
[
    (2 2 2 0 0 0 1 1 1 1 1 1 2 2 2 0 0 0 2 2 2 2 2 2 2 2 2 2 2 0 0 0
        1 1 1 2 2 2 1 1 1 0 0 0),
    (1 2 0 2 0 1 0 1 2 0 2 1 1 0 2 2 1 0 1 0 2 2 1 0 0 2 1 1 0 2 1 0 2
        1 0 2 2 2 2 0 0 0 1 1 1)
]
> SystematicEncoding(C, L)[1] eq SystematicEncoding(C, i);
true

> M := Matrix(Integers(27), 3, L);
> SystematicEncoding(C, M) eq SystematicEncoding(C, L);
true
```

1.13 Decoding

1.13.1 Permutation Decoding

Permutation decoding is a technique, introduced by Prange and developed by MacWilliams, that involves finding a subset of the permutation automorphism group of a code in order to assist in decoding. This method can also be used for any nonlinear code, as long as it is systematic. Since all codes over $\mathbb{F}_p$ that are Carlet's generalized Gray map image of a linear code over $\mathbb{Z}_{p^s}$ are systematic [39], we can use this method in order to decode.

Let C be a linear code over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$ and $C_p = \Phi_s(C)$ be the correspoding code over $\mathbb{F}_p$, where Φ_s is Carlet's generalized Gray map given in Section 1.2. Let t be the error-correcting capability of C_p and $r \in \{1, \ldots, t\}$. A subset $S_p \subseteq \mathrm{PAut}(C_p)$ is an r-PD-set for C_p with respect to an information set $I_p \subseteq \{1, \ldots, np^{s-1}\}$ if every r-set of coordinate positions in $\{1, \ldots, np^{s-1}\}$ is moved out of I_p by at least one element of S_p.

Let $\Phi : \mathrm{Sym}(n) \to \mathrm{Sym}(np^{s-1})$ be the map defined as

$$\Phi(\tau)(i) = p^{s-1}\tau((i + \chi(i))/p^{s-1}) - \chi(i),$$

where $\chi(i) = p^{s-1} - (i \bmod p^{s-1})$, for all $\tau \in \mathrm{Sym}(n)$ and $i \in \{1, \ldots, np^{s-1}\}$. Given a subset $S \subseteq \mathrm{Sym}(n)$, we define

$$\Phi(S) = \{\Phi(\tau) : \tau \in S\} \subseteq \mathrm{Sym}(np^{s-1}). \tag{1.11}$$

It is easy to see that if $S \subseteq \mathrm{PAut}(C) \subseteq \mathrm{Sym}(n)$, then $\Phi(S) \subseteq \mathrm{PAut}(\Phi(C)) \subseteq \mathrm{Sym}(np^{s-1})$.

If every r-set of coordinate positions in $\{1, \ldots, n\}$ can be moved out of an information set $I \subseteq \{1, \ldots, n\}$ for C by an element of $S \subseteq \mathrm{PAut}(C)$, then $S_p = \Phi(S)$ is an r-PD-set for $C_p = \Phi_s(C)$ with respect to the information set $I_p = \Phi(I)$, where $\Phi(I)$ is defined as in (1.7) and $\Phi(S)$ as in (1.11) [39]. The converse is also true.

`IsZpPermutationDecodeSet(C, I, S, r)`

Given a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, a sequence $I \subseteq \{1, \ldots, np^{s-1}\}$, a sequence S of elements in the symmetric group $\mathrm{Sym}(np^{s-1})$ of permutations on the set $\{1, \ldots, np^{s-1}\}$, and an integer $r \geq 1$, return `true` if and only if S is an r-PD-set for $C_p = \Phi_s(C)$, where Φ_s is Carlet's generalized Gray map, with respect to the information set I.

The arguments I and S can also be given as a sequence $I \subseteq \{1, \ldots, n\}$ and a sequence S of elements in the symmetric group $\mathrm{Sym}(n)$ of permutations on the set $\{1, \ldots, n\}$, respectively. In this case, the function returns `true` if and only if $\Phi(S)$ is an r-PD-set for $C_p = \Phi_s(C)$ with respect to the information set $\Phi(I)$, where $\Phi(I)$ is defined as in (1.7) and $\Phi(S)$ as in (1.11).

Depending on the length of the code C, its type, and the integer r, this function could take some time to compute whether S or $\Phi(S)$ is an r-PD-set for C_p with respect to I or $\Phi(I)$, respectively. Specifically, if the function returns `true`, it is necessary to check $\sum_{i=1}^{r} \binom{|I|}{i} \cdot \binom{N-|I|}{r-i}$ r-sets, where $N = n$ and $|I| = t_1 + \cdots + t_s$ when I is given as an information set for C, or $N = np^{s-1}$ and $|I| = st_1 + (s-1)t_2 + \cdots t_1$ when I is given as an information set for $C_p = \Phi_s(C)$.

The verbose flag `IsPDsetFlag` is set to level 0 by default. If it is set to level 1, the total time used to check the condition is shown. Moreover, the reason why the function returns `false` is also shown, that is, whether I is not an information set, S is not a subset of the permutation automorphism group or S is not an r-PD-set. If it is set to level 2, the percentage of the computation process performed is also printed.

If C is over $\mathbb{Z}_4$, function `IsZpPermutationDecodeSet(C, I, S, r)` coincides with function `IsPermutationDecodeSet(C, I, S, r)`, which works only for linear codes over $\mathbb{Z}_4$, but the former may perform less efficiently when $I \subseteq \{1, \ldots, 2n\}$ because it calls function `IsZpInformationSet(C, I)` instead of `IsInformationSet(C, I)`.

Example H1E22

The Hadamard code C over $\mathbb{Z}_8$ of type $(8; 2, 0, 0)$ is defined, and two sets of permutations $S \subseteq \mathrm{Sym}(8)$ and $S_p \subseteq \mathrm{Sym}(32)$ are considered. It is checked that $\Phi(S)$ and S_p are both 3-PD-sets for $C_p = \Phi_3(C)$, with respect to the information set I_p given by function `ZpInformationSet(C)`. The verbose flag `IsPDsetFlag` is set to level 2 to show the whole process of checking.

```
> C := ZpHadamardCode(2, [2,0,0]);

> p1 := Sym(8)!(1,3,5,7)(2,4,6,8);
> p2 := Sym(8)!(1,5)(2,6)(3,7)(4,8);
> p3 := Sym(8)!(1,7,5,3)(2,8,6,4);
> S := [Sym(8)!1, p1, p2, p3];

> p1 := Sym(32)!(1,9,17,25)(2,10,18,26)(3,11,19,27)(4,12,20,28)
                (5,13,21,29)(6,14,22,30)(7,15,23,31)(8,16,24,32);
> p2 := Sym(32)!(1,17)(2,18)(3,19)(4,20)(5,21)(6,22)(7,23)(8,24)
                (9,25)(10,26)(11,27)(12,28)(13,29)(14,30)(15,31)(16,32);
> p3 := Sym(32)!(1,25,17,9)(2,26,18,10)(3,27,19,11)(4,28,20,12)
                (5,29,21,13)(6,30,22,14)(7,31,23,15)(8,32,24,16);
> Sp := [Sym(32)!1, p1, p2, p3];

> I, Ip := ZpInformationSet(C);

> SetVerbose("IsPDSetFlag", 2);

> IsZpPermutationDecodeSet(C, I, S, 3);
Checking whether I is an information set...
Checking whether S is in the permutation automorphism group...
Checking whether S is an r-PD-set...
10 %
20 %
30 %
40 %
50 %
60 %
70 %
80 %
90 %
Took 0.010 seconds (CPU time)
true

> IsZpPermutationDecodeSet(C, Ip, Sp, 3);
```

```
Checking whether I is an information set...
Checking whether S is in the permutation automorphism group...
Checking whether S is an r-PD-set...
10 %
20 %
30 %
40 %
50 %
60 %
70 %
80 %
90 %
Took 0.040 seconds (CPU time)
true
```

`ZpPermutationDecode(C, Ip, S, r, u)`

Given

- a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$,
- an information set $I_p \subseteq \{1, \ldots, np^{s-1}\}$ for $C_p = \Phi_s(C)$ as a sequence of coordinate positions,
- a sequence S such that either S or $\Phi(S)$ is an r-PD-set for C_p with respect to the information set I_p,
- an integer $r \in \{1, \ldots, t\}$, where t is the error-correcting capability of C_p, and
- a vector u which can be defined from the ambient space $U = \mathbb{Z}_{p^s}^n$ or from the ambient space $U_p = \mathbb{F}_p^{np^{s-1}}$,

the function attempts to decode $u \in U_p$ (or $\Phi_s(u) \in U_p$ if $u \in U$) with respect to the code C_p (or C if $u \in U$), assuming a systematic encoding with respect to the information set I_p. If the decoding algorithm succeeds in computing a codeword $u' \in C_p$ as the decoded version of $u \in U_p$ (or $\Phi_s(u) \in U_p$ if $u \in U$), then the function returns `true`, the preimage of u' by Carlet's generalized Gray map Φ_s and finally u'. If the decoding algorithm does not succeed in decoding u, then the function returns `false`, the zero codeword in C and the zero codeword in C_p.

The permutation decoding algorithm consists in moving all errors in a received vector $u = c + e$, where $u \in U_p$, $c \in C_p$ and $e \in U_p$ is an error vector with at most t errors, out of the information positions, that is, moving the nonzero coordinates of e out of the information set I_p for C_p, by using a permutation in $S \subseteq \mathrm{PAut}(C_p)$ (or $\Phi(S) \subseteq \mathrm{PAut}(C_p)$ if $S \subseteq \mathrm{PAut}(C)$). If $S \subseteq \mathrm{PAut}(C) \subseteq \mathrm{Sym}(n)$, then $\Phi(S) \subseteq \mathrm{PAut}(C_p) \subseteq \mathrm{Sym}(np^{s-1})$ is computed by using the map Φ defined in (1.11). The function does not check whether I_p is an information set for C_p, whether S or $\Phi(S)$ is an r-PD-set for C_p with respect to I_p, or whether $r \leq t$.

If C is over $\mathbb{Z}_4$, function `ZpPermutationDecode(C, Ip, S, r, u)` coincides with function `PermutationDecode(C, I, S, r, u)`, which works only for linear codes over $\mathbb{Z}_4$. However, the former only accepts an information set $I_p \subseteq \{1, \ldots, 2n\}$ for C_2, while the latter also accepts an information set $I \subseteq \{1, \ldots, n\}$ for C as long as the sequence $S \subseteq \mathrm{PAut}(C)$.

`ZpPermutationDecode(C, Ip, S, r, Q)`

Given

- a linear code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$,
- an information set $I_p \subseteq \{1, \ldots, np^{s-1}\}$ for $C_p = \Phi_s(C)$ as a sequence of coordinate positions,
- a sequence S such that either S or $\Phi(S)$ is an r-PD-set for C_p with respect to the information set I_p,
- an integer $r \in \{1, \ldots, t\}$, where t is the error-correcting capability of C_p, and
- a sequence Q of vectors which can be defined from the ambient space $U = \mathbb{Z}_{p^s}^n$ or from the ambient space $U_p = \mathbb{F}_p^{np^{s-1}}$,

the function attempts to decode all vectors $u \in Q$ if $Q \subseteq U_p$ (or $\Phi_s(u)$ if $Q \subseteq U$) with respect to the code C_p (or C if $Q \subseteq U$), assuming a systematic encoding with respect to the information set I_p. The function returns three values: a sequence of booleans representing whether the decoding process have been successful for each $u \in Q$, a sequence of codewords of C and a sequence of codewords of C_p. For each $u \in Q$, if the decoding algorithm succeeds in computing a codeword $u' \in C_p$ as the decoded version of $u \in U_p$ (or $\Phi_s(u) \in U_p$ if $u \in U$), then it returns the preimage of u' by Carlet's generalized Gray map Φ_s in the second sequence, and u' in the third sequence. If the decoding algorithm does not succeed in decoding u, then the function returns the zero codeword in C and the zero codeword in C_p in the corresponding positions of the second and third sequences, respectively.

The permutation decoding algorithm consists in moving all errors in a received vector $u = c + e$, where $u \in U_p$, $c \in C_p$ and $e \in U_p$ is an error vector with at most t errors, out of the information positions, that is, moving the nonzero coordinates of e out of the information set I_p for C_p, by using a permutation in $S \subseteq \mathrm{PAut}(C_p)$ (or $\Phi(S) \subseteq \mathrm{PAut}(C_p)$ if $S \subseteq \mathrm{PAut}(C)$). If $S \subseteq \mathrm{PAut}(C) \subseteq \mathrm{Sym}(n)$, then $\Phi(S) \subseteq \mathrm{PAut}(C_p) \subseteq \mathrm{Sym}(np^{s-1})$ is computed by using the map Φ defined in (1.11). The function does not check whether I_p is an information set for C_p or not, nor whether S or $\Phi(S)$ is an r-PD-set for C_p with respect to I_p.

If C is over $\mathbb{Z}_4$, function `ZpPermutationDecode(C, Ip, S, r, Q)` coincides with function `PermutationDecode(C, I, S, r, Q)`, which works only for linear codes over $\mathbb{Z}_4$. However, the former only accepts an information set $I_p \subseteq \{1, \ldots, 2n\}$ for C_2, while the latter also accepts an information set $I \subseteq \{1, \ldots, n\}$ for C as long as the sequence $S \subseteq \mathrm{PAut}(C)$.

Example H1E23

First, the generalized Hadamard code C over $\mathbb{Z}_9$ of type $(9; 2, 0)$ is constructed. Then, it is checked that the set S_p, formed by 4 permutations in $\mathrm{Sym}(9)$, is a 3-PD-set for $C_p = \Phi_2(C)$, with respect to the information set I_p given by function `ZpInformationSet(C)`.

A codeword $c \in C$ is considered and then perturbed by an error vector e of weight 3 (considering the homogeneous weight). The resulting vector $u = c + e$ is mapped to a vector $u_p = \Phi_2(u)$ over $\mathbb{F}_3$, which is not a codeword of C_p. Both vectors u and u_p are decoded using function `ZpPermutationDecode`, which is successful in obtaining the correct codewords c and $c_p = \Phi_2(c)$, respectively.

```
> C := ZpHadamardCode(3, [2,0]);
> n := Length(C);
> np := n*3;

> U := RSpace(Integers(9), n);
```

```
> Up := VectorSpace(GF(3), np);
> grayMap := CarletGrayMap(UniverseCode(Integers(9), n));

> I, Ip := ZpInformationSet(C);

> p1 := Sym(np)!(1,22,16,10,4,25,19,13,7)(2,23,17,11,5, 26,20,14,8)
>               (3,24,18,12,6,27,21,15,9);
> p2 := Sym(np)!(1,16,4,19,7,22,10,25,13)(2,17,5,20,8,23,11,26,14)
>               (3,18,6,21,9,24,12,27,15);
> p3 := Sym(np)!(1,10,19)(2,11,20)(3,12,21)(4,13,22)(5,14,23)
>               (6,15,24)(7,16,25)(8,17,26)(9,18,27);
> Sp := [Sym(np)!1, p1, p2, p3];

> IsZpPermutationDecodeSet(C, Ip, Sp, 3);
true

> c := C![0,1,2,3,4,5,6,7,8];
> e := U![0,1,0,0,0,0,0,0,0];
> u := c+e;
> cp := grayMap(c);
> up := grayMap(u);
> (u in U) and (up in Up);
true

> isDecoded, uDecoded, upDecoded := ZpPermutationDecode(C, Ip, Sp, 3, u);
> isDecoded;
true
> uDecoded eq c;
true
> upDecoded eq cp;
true

> isDecoded, uDecoded, upDecoded := ZpPermutationDecode(C, Ip, Sp, 3, up);
> isDecoded;
true
> uDecoded eq c;
true
> upDecoded eq cp;
true
```

`PDSetHadamardCodeZps(p, [$t_1, t_2, \ldots, t_s$])`

`AlgMethod` MonStgElt *Default*: `"Deterministic"`

Given a prime p and a sequence of nonnegative integers $[t_1, \ldots, t_s]$ with $t_1 > 0$ and $s > 1$, the generalized Hadamard code C over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, given by function `ZpHadamardCode(p, [$t_1, \ldots, t_s$])`, is considered. The function returns an information set $I = \{i_1, \ldots, i_{t_1+\cdots+ts}\} \subseteq \{1, \ldots, n\}$ for C together with a subset S of the permutation automorphism group of C such that $\Phi(S)$ is an r-PD-set for $C_p = \Phi_s(C)$ with respect to $\Phi(I)$, where Φ_s is Carlet's generalized Gray map as defined in (1.1), $\Phi(I)$ is as in (1.7) and $\Phi(S)$ as in (1.11). The function also returns the information set $\Phi(I)$ and the r-PD-set $\Phi(S)$.

Note that for $p = 2$ and $[t_1, \ldots, t_s] = [1, 0, \ldots, 0, t_s]$ or $[t_1, \ldots, t_s] = [1, 0, \ldots, 0, 1, t_s]$, we have that C_p is linear [17], so it is possible to find an r-PD-set of size $r+1$ for C_p, for any $r \leq \lfloor 2^m/(m+1) \rfloor$, by using function `PDSetHadamardCode(m)` with $m = t_s + s$ if $[t_1, \ldots, t_s] = [1, 0, \ldots, 0, t_s]$ and $m = t_s + s + 2$ if $[t_1, \ldots, t_s] = [1, 0, \ldots, 0, 1, t_s]$.

The information sets I and $\Phi(I)$ are returned as sequences of $t_1 + \cdots + t_s$ and $s(t_1 - 1) + (s-1)t_2 + \cdots + t_s$ integers, giving the coordinate positions that correspond to the information sets for C and C_p, respectively. The sets S and $\Phi(S)$ are also returned as sequences of elements in the symmetric groups $\mathrm{Sym}(n)$ and $\mathrm{Sym}(np^{s-1})$ of permutations on the set $\{1, \ldots, n\}$ and $\{1, \ldots, np^{s-1}\}$, respectively.

A deterministic algorithm is used by default. In this case, when $t_1 \geq 2$, the function first computes r as the maximum of $g_p^{t_1,\ldots,t_s}$ and $\tilde{f}_p^{t_1,\ldots,t_s}$, where $g_p^{t_1,\ldots,t_s}$ is given by the general construction described in [40] and

$$\tilde{f}_p^{t_1,\ldots,t_s} = \max\{f_p^{t_1,0,\ldots,0}, f_p^{1,t_2,0,\ldots,0}, \ldots, f_p^{1,0,\ldots,0,t_s}\} \leq f_p^{t_1,\ldots,t_s},$$

where $f_p^{t_1,\ldots,t_s}$ is the theoretical upper bound for the value of r such that there exists an r-PD-set of size $r+1$ for a Hadamard code over $\mathbb{Z}_{p^s}$ of type $(n; t_1, \ldots, t_s)$, given in [39]. Let i be the first index $i \geq 1$ such that the maximum $\tilde{f}_p^{t_1,\ldots,t_s}$ is achieved. Then, if $\tilde{f}_p^{t_1,\ldots,t_s} > g_p^{t_1,\ldots,t_s}$, it constructs an r-PD-set of size $r+1$ for the generalized Hadamard code over $\mathbb{Z}_{p^s}$ of type $(n; t_1, 0, \ldots, 0)$ if $i = 1$ or $(n; 1, 0, \ldots, 0, t_i, 0, \ldots, 0)$ otherwise, which is transformed into an r-PD-set for C, by using the recursive construction defined in [39]. The value of r remains unchanged after the recursive construction, so $r = \tilde{f}_p^{t_1,\ldots,t_s}$. On the other hand, if $g_p^{t_1,\ldots,t_s} \geq \tilde{f}_p^{t_1,\ldots,t_s}$, the general construction is applied and an r-PD-set of size $r+1$ with $r = g_p^{t_1,\ldots,t_s}$ is obtained. When $t_1 = 1$ and there exists an index $i \geq 2$ such that $t_i \neq 0$ and $t_2 = \cdots = t_{i-1} = 0$, it first constructs an r-PD-set of size $r+1$ for the generalized Hadamard code over $\mathbb{Z}_{p^s}$ of type $(n; 1+t_i, t_{i+1}, \ldots, t_s)$, which satisfies that $1+t_i \geq 2$, by following the same process as described above when $t_1 \geq 2$. Then, the obtained r-PD-set is transformed into an r-PD-set for C as described in [39, 40].

If the parameter `AlgMethod` is assigned the value `"Nondeterministic"`, the function tries to improve the previous results by finding an r-PD-set of size $r+1$ such that $\tilde{f}_p^{t_1,\ldots,t_s} \leq r \leq f_p^{t_1,\ldots,t_s}$. In this case, the function starts from the maximum value of $r = f_p^{t_1,\ldots,t_s}$ and attempts to find an r-PD-set within a time limit of 5.0 seconds of "user time". This is performed 5 times, each time starting from an empty set and trying to construct the r-PD set incrementally, by adding elements randomly. If such an r-PD-set is not found, the value of r decreases by one and the same process takes place with the new value of r. The value of r keeps decreasing until an r-PD-set is found or $r = \tilde{f}_p^{t_1,\ldots,t_s}$.

The verbose flag `PDsetHadamardFlag` is set to level 0 by default. If it is set to level 1, some information about the process of constructing the r-PD-set of size $r+1$ is shown. Moreover, the value of the theoretical upper bound $f_p^{t_1,\ldots,t_s}$ given in [39] is also shown.

If $p = 2$ and $s = 2$, then function `PDSetHadamardCodeZ4(`t_1`,` $2t_1 + t_2 - 1$`)` can also be used. Both function only coincide when $t_2 = 0$. When $t_2 > 0$, the output parameters I and $\Phi(I)$ coincide as sets and `PDSetHadamardCodeZps(2,` $[t_1, t_2]$`)` may give a larger r-PD-set.

Example H1E24

First, the generalized Hadamard code C over $\mathbb{Z}_{27}$ of type $(729; 3, 0, 0)$ is constructed. Then, by using function `PDSetHadamardCodeZps(p, [`$t_1, t_2, \ldots, t_s$`])`, we obtain an information set I for C and a set $S \subseteq \mathrm{PAut}(C)$ such that $I_p = \Phi(I)$ is an information set for $C_p = \Phi_3(C)$ and $S_p = \Phi(S)$ is an r-PD-set of size $r+1$ for C_p with $r = 242$. In this case, the maximum value of r is obtained since $r \leq f_3^{3,0,0} = 242$.

A codeword $c \in C$ is considered and the corresponding $c_p = \Phi_3(c) \in C_p$ is perturbed by an error vector e_p of Hamming weight 242. The resulting vector $u_p = c_p + e_p$ is decoded using function `ZpPermutationDecode`, which is successful in obtaining the correct codewords c and c_p, respectively.

```
> p := 3;
> type := [3,0,0];
> s := #type;
> C := ZpHadamardCode(p, type);
> n := Length(C);
> np := n*p^(s-1);

> U := RSpace(Integers(p^s), n);
> Up := VectorSpace(GF(p), np);
> grayMap := CarletGrayMap(UniverseCode(Integers(p^s), n));

> I, S, Ip, Sp := PDSetHadamardCodeZps(p, type);
> r := #Sp-1; r;
242
> t1 := type[1];
> r eq Floor((p^(s*(t1-1))-t1)/t1);
true
> c := C![6^^n];
> cp := grayMap(c);
> ep := Up![1^^r,0^^(np-r)];
> up := cp + ep;

> isDecoded, uDecoded, upDecoded := ZpPermutationDecode(C, Ip, Sp, r, up);
> isDecoded;
true
> uDecoded eq c;
true
> upDecoded eq cp;
true
```

If we consider a generalized Hadamard code over $\mathbb{Z}_{p^s}$ that is not of type $(n; t_1, 0, \ldots, 0)$ or $(n; 1, 0, \ldots, 0, t_i, 0, \ldots, 0)$ for $i \in \{2, \ldots, s\}$, then the deterministic method may not be able to give an r-PD-set of size $r+1$ for the maximum value of $r \leq f_p^{t_1,\ldots,t_s}$. However, by selecting the `Nondeterministic` method, the function tries to increase the value of r. The flag `PDsetHadamardFlag` is set to level 1 to see the process of trying to obtain the r-PD-set.

```
> p := 3;
```

```
> type := [1,0,1,1];
> s := #type;
> C := ZpHadamardCode(p, type);
> n := Length(C);
> np := n*p^(s-1);

> SetVerbose("PDSetHadamardFlag", 1);
> I, S, Ip, Sp := PDSetHadamardCodeZps(p, type);
The upper bound for an r-PD-set of size r+1 is 8.
A 5-PD-set of size 6 has been obtained by applying the general construction
  for a Hadamard code of type [ 2, 1 ] and then adapted for a Hadamard code of
type [ 1, 0, 1, 1 ].
> r := #Sp-1; r;
5
> IsZpPermutationDecodeSet(C, I, S, r);
true

> I, S, Ip, Sp := PDSetHadamardCodeZps(p, type : AlgMethod := "Nondeterministic");
The upper bound for an r-PD-set of size r+1 is 8.
Trying to find an 8-PD-set of size 9 randomly...
Trying to find an 7-PD-set of size 8 randomly...
A 7-PD-set has been found in iteration 2 of 5!!
> r := #Sp-1; r;
7
> IsZpPermutationDecodeSet(C, I, S, r);
true
```

When $p = 2$ and $s = 2$, function `PDSetHadamardCodeZps(p, [t1,0])` coincides with function `PDSetHadamardCodeZ4(t1, 2*t1-1)`.

```
> I, S, Ibin, Sbin := PDSetHadamardCodeZps(2, [3,0]);
The upper bound for an r-PD-set of size r+1 is 4.
A 4-PD-set of size 5 has been obtained.
> I2, S2, I2bin, S2bin := PDSetHadamardCodeZ4(3, 5);

> I eq I2; I;
true
[ 1, 2, 5 ]
> S eq S2; S;
true
[
    Id($),
    (1, 12, 13, 8, 9, 4, 5, 16)(2, 15, 14, 11, 10, 7, 6, 3),
    (1, 13, 11, 7)(2, 8, 12, 14)(3, 15, 9, 5)(4, 6, 10, 16),
    (1, 8, 5, 10)(2, 9, 16, 13)(3, 14, 7, 4)(6, 15, 12, 11),
    (1, 11)(2, 12)(3, 9)(4, 10)(5, 15)(6, 16)(7, 13)(8, 14)
]
> Ibin eq I2bin; Ibin;
true
```

```
[ 1, 2, 3, 4, 9, 10 ]
> Sbin eq S2bin; Sbin;
true
[
    Id($),
    (1, 23, 25, 15, 17, 7, 9, 31)(2, 24, 26, 16, 18, 8, 10, 32)
    (3, 29, 27, 21, 19, 13, 11, 5)(4, 30, 28, 22, 20, 14, 12, 6),
    (1, 25, 21, 13)(2, 26, 22, 14)(3, 15, 23, 27)(4, 16, 24, 28)
    (5, 29, 17, 9)(6, 30, 18, 10)(7, 11, 19, 31)(8, 12, 20, 32),
    (1, 15, 9, 19)(2, 16, 10, 20)(3, 17, 31, 25)(4, 18, 32, 26)
    (5, 27, 13, 7)(6, 28, 14, 8)(11, 29, 23, 21)(12, 30, 24, 22),
    (1, 21)(2, 22)(3, 23)(4, 24)(5, 17)(6, 18)(7, 19)(8, 20)(9, 29)
    (10, 30)(11, 31)(12, 32)(13, 25)(14, 26)(15, 27)(16, 28)
]
```

Bibliography

[1] E. F. Assmus and J. D. Key, *Designs Their Codes*, Cambridge University Press, 1992.

[2] D. K. Bhunia, C. Fernández-Córdoba, and M. Villanueva, "On the linearity and classification of $\mathbb{Z}_{p^s}$-linear generalized Hadamard codes," *Designs, Codes and Cryptography*, vol. 90, pp. 1037-1058, 2022.

[3] I. F. Blake, "Codes over integer residue rings," *Information and Control*, vol. 29, no. 4, pp. 295–300, 1975.

[4] 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.

[5] J. Borges, C. Fernández-Córdoba, J. Pujol, J. Rifà, and M. Villanueva, "$\mathbb{Z}_2\mathbb{Z}_4$-additive codes. A MAGMA package", version 5.0, Universitat Autònoma de Barcelona, 2022. DOI: 10.34810/data601. Available at http://ccsg.uab.cat and GitHub.

[6] J. Borges, K. T. Phelps, and 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, no. 8, pp. 2028-2034, 2003.

[7] J. Borges and J. Rifà, "A characterization of 1-perfect additive codes," *IEEE Trans. on Information Theory*, vol. 45, no. 5, pp. 1688-1697, 1999.

[8] J. J. Cannon, W. Bosma, C. Fieker, and A. Steel (Eds.) *Handbook of* Magma *Functions*, Edition 2.26-4, 6347 pages, 2021.

[9] C. Carlet, "$\mathbb{Z}_{2^k}$-linear codes," *IEEE Trans. on Information Theory*, vol. 44, no. 4, pp. 1543-1547, 1998.

[10] I. Constantinescu and W. Heise, "A metric for codes over residue class rings," *Problemy Peredachi Informatsii*, vol. 33, no. 3, pp. 22-28, 1997.

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

[12] P. Delsarte and V. Levenshtein, "Association schemes and coding theory," *IEEE Trans. on Information Theory*, vol. 44, no. 6, pp. 2477–2504, 1998.

[13] S. T. Dougherty and C. Fernández-Córdoba, "Codes over $\mathbb{Z}_{2^k}$, Gray map and self-dual codes," *Advances in Mathematics of Communications*, vol. 5, no. 4, pp. 571-588, 2011.

[14] C. Fernández-Córdoba, J. Pujol, and M. Villanueva, "On rank and kernel of $\mathbb{Z}_4$-linear codes," *Lecture Notes in Computer Science*, no. 5228, pp. 46-55, 2008.

[15] 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.

[16] C. Fernández-Córdoba, A. Torres-Martín, C. Vela, and M. Villanueva, "Computing efficiently a parity-check matrix for $\mathbb{Z}_{p^s}$-additive codes," submitted to *IEEE Trans. on Information Theory*, 2023.

[17] C. Fernández-Córdoba, C. Vela, and M. Villanueva, "On $\mathbb{Z}_{2^s}$-linear Hadamard codes: kernel and partial classification," *Designs, Codes and Cryptography*, vol. 87, no. 2–3, pp. 417-435, 2019.

[18] C. Fernández-Córdoba, C. Vela, and M. Villanueva, "Nonlinearity and kernel of $\mathbb{Z}_{2^s}$-linear simplex and MacDonald Codes," *IEEE Trans. on Information Theory*, vol. 68, no. 11, pp. 7174-7183, 2022.

[19] M. Greferath and S. E. Schmidt, "Gray isometries for finite chain rings and a nonlinear ternary $(36, 3^{12}, 15)$ code," *IEEE Trans. on Information Theory*, vol. 45, no. 7, pp. 2522-2524, 1999.

[20] M. K. Gupta, M. C. Bhandari, and A. K. Lal, "On some linear codes over $\mathbb{Z}_{2^s}$," *Designs, Codes and Cryptography*, vol. 36, no. 3, pp. 227-244, 2005.

[21] 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.

[22] T. Honold, and A. A. Nechaev, "Weighted modules and representations of codes," *Problems of Information Transmission*, vol. 35, no. 3, pp. 18–39, 1999.

[23] D. Jungnickel, "On difference matrices, resolvable designs and generalized Hadamard matrices," *Mathematische Zeitschrift*, vol. 167, pp. 49-60, 1979.

[24] D. S. Krotov, "On $\mathbb{Z}_{2^k}$-dual binary codes," *IEEE Trans. on Information Theory*, vol. 53, no. 4, pp. 1532-1537, 2007.

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

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

[27] A. A. Nechaev, "The Kerdock code in a cyclic form," *Discrete Mathematics and Applications*, vol. 1, pp. 365-384, 1991.

[28] J. Pernas, J. Pujol, and M. Villanueva, "Classification of some families of quaternary Reed-Muller codes," *IEEE Trans. on Information Theory*, vol. 57, no. 9, pp. 6043-6051, 2011.

[29] J. Pernas, J. Pujol, and M. Villanueva, "Characterization of the automorphism group of quaternary linear Hadamard codes," *Designs, Codes and Cryptography*, vol. 70, no. 1-2, pp. 105-115, 2014.

[30] K. T. Phelps and J. Rifà, "On binary 1-perfect additive codes: some structural properties," *IEEE Trans. on Information Theory*, vol. 48, no. 9, pp. 2587-2592, 2002.

[31] K. T. Phelps, J. Rifà, and 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, no. 1, pp. 316-319, 2006.

[32] 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, no. 1, pp. 99-104, 2009.

[33] J. Rifà and J. Pujol, "Translation invariant propelinear codes," *IEEE Trans. on Information Theory*, vol. 43, no. 2, pp. 590-598, 1997.

[34] P. Shankar, "On BCH codes over arbitrary integer rings," *IEEE Trans. on Information Theory*, vol. 25, no. 4, pp. 480–483, 1979.

[35] 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.

[36] J. J. Sylvester, "Thoughts on orthogonal matrices, simultaneous sign-successions, and tessellated pavements in two or more colours, with applications to Newton's rule, ornamental tile-work, and the theory of numbers," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 34, pp. 461-475, 1867.

[37] H. Tapia-Recillas and G. Vega, "On $\mathbb{Z}_{2^k}$-linear and quaternary codes," *SIAM Journal on Discrete Mathematics*, vol 17, no. 1, pp. 103-113, 2003.

[38] A. Torres-Martín and M. Villanueva, "Systematic encoding and permutation decoding for $\mathbb{Z}_{p^s}$-linear codes," *IEEE Trans. on Information Theory*, vol. 68, no. 7, pp. 4435-4443, 2022.

[39] A. Torres-Martín and M. Villanueva, "Partial permutation decoding and PD-sets for $\mathbb{Z}_{p^s}$-linear generalized Hadamard codes," *Finite Fields and Their Applications*, vol. 93, 102316, 2024. DOI: 10.1016/j.ffa.2023.102316.

[40] J. Rifà, A. Torres-Martín and M. Villanueva, "Explicit construction of r-PD-sets for non-free $\mathbb{Z}_{p^s}$-linear generalized Hadamard codes," submitted to *Finite Fields and Their Applications*, 2023.

[41] M. Shi, R. Wu, and D. S. Krotov, "On $Z_pZ_{p^k}$-additive codes and their duality," *IEEE Trans. on Information Theory*, vol. 65, no. 6, pp. 3841-3847, 2019.

[42] M. Villanueva, F. Zeng, and J. Pujol, "Efficient representation of binary nonlinear codes: constructions and minimum distance computation," *Designs, Codes and Cryptography*, vol. 76, no. 1, pp. 3-21, 2015.

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