CODES OVER $\mathbb{Z}_4$

AND

PERMUTATION DECODING OF LINEAR CODES

A MAGMA Package[1]

by

Roland D. Barrolleta, Jaume Pernas, Jaume Pujol, and Mercè Villanueva

Combinatoric, Coding and Security Group (CCSG)

Universitat Autònoma de Barcelona

Version 2.1

Barcelona
November 2, 2017

[1]This work has been partially supported by the Spanish MICINN under Grants PCI2006-A7-0616, MTM2006-03250, MTM2009-08435, TIN2010-17358, TIN2013-40524-P and TIN2016-77918-P; and by the Catalan AGAUR under Grants 2009SGR1224 and 2014SGR691.

Contents

Chapter 1

Preface

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

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

Part of the research developed by CCSG deals with codes over $\mathbb{Z}_4$. Some members of CCSG have been developing this new package that expands the current functionality for codes over $\mathbb{Z}_4$ in Magma. Magma is a software package designed to solve computationally hard problems in algebra, number theory, geometry and combinatorics. The latest version of this package for codes over $\mathbb{Z}_4$ and this manual with the description of all developed functions can be downloaded from the web page `http://ccsg.uab.cat`. For any comment or further information about this package, you can send an e-mail to *support-ccsg@deic.uab.cat*.

The authors would like to thank Lorena Ronquillo and Bernat Gastón for their contributions developing part of this Magma package. Functions described in Sections 2.2, 2.3, 2.4, 2.5 and 2.9 have been developed mainly by the PhD student Jaume Pernas, and the ones in Sections 2.6, 2.7, 2.8 and 3.1 by the PhD student Roland D. Barrolleta, both under the supervision of Jaume Pujol and Mercè Villanueva.

Chapter 2

Codes over $\mathbb{Z}_4$

2.1 Introduction

Magma currently supports the basic facilities for linear codes over integer residue rings and galois rings (see [7, Chapter 130]), including additional functionality for the special case of codes over $\mathbb{Z}_4$ (or equivalently, quaternary linear codes). A code over $\mathbb{Z}_4$ is a subgroup of $\mathbb{Z}_4^n$, so it is isomorphic to an abelian structure $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^\delta$ and we will say that it is of type $2^\gamma 4^\delta$, or simply that it has $2^{\gamma+2\delta}$ codewords. As general references on the available functions in Magma for codes over $\mathbb{Z}_4$, the reader is referred to [13, 24].

The functions described in this chapter expand the current functionality for codes over $\mathbb{Z}_4$ in Magma. Specifically, there are functions to construct some families of codes over $\mathbb{Z}_4$ and new codes over $\mathbb{Z}_4$ from given codes over $\mathbb{Z}_4$ (Sections 2.2 and 2.3). Moreover, efficient functions for computing the rank and dimension of the kernel of any code over $\mathbb{Z}_4$ are also included (Section 2.4), as well as general functions for computing coset representatives for a subcode in a code over $\mathbb{Z}_4$ (Section 2.5). There are also functions for computing the permutation automorphism group for Hadamard and extended perfect codes over $\mathbb{Z}_4$, and their orders (Section 2.9). Functions related to the information space, information sets, syndrome space and coset leaders for codes over $\mathbb{Z}_4$ can also be found (Sections 2.6 and 2.7). Finally, functions to decode codes over $\mathbb{Z}_4$ by using different methods are also provided (Section 2.8). As general references on these new functions, the reader is referred to the bibliography included at the end of this document.

In this chapter the term "code" will refer to a code over $\mathbb{Z}_4$, unless otherwise specified.

2.2 Families of Codes over $\mathbb{Z}_4$

These functions give some constructions for some families of codes over $\mathbb{Z}_4$.

`HadamardCodeZ4(δ, m)`

Given an integer $m \geq 1$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, return a Hadamard code over $\mathbb{Z}_4$ of length 2^{m-1} and type $2^\gamma 4^\delta$, where $\gamma = m + 1 - 2\delta$. Moreover, return a generator matrix with $\gamma + \delta$ rows constructed in a recursive way from the Plotkin and BQPlotkin constructions defined in Section 2.3.

A Hadamard code over $\mathbb{Z}_4$ of length 2^{m-1} is a code over $\mathbb{Z}_4$ such that, after the Gray map, give a binary (not necessarily linear) code with the same parameters as the binary Hadamard code of length 2^m.

`ExtendedPerfectCodeZ4(δ, m)`

Given an integer $m \geq 2$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, return an extended perfect code over $\mathbb{Z}_4$ of length 2^{m-1}, such that its dual code is of type $2^\gamma 4^\delta$, where $\gamma = m + 1 - 2\delta$. Moreover, return a generator matrix constructed in a recursive way from the Plotkin and BQPlotkin constructions defined in Section 2.3.

An extended perfect code over $\mathbb{Z}_4$ of length 2^{m-1} is a code over $\mathbb{Z}_4$ such that, after the Gray map, give a binary (not necessarily linear) code with the same parameters as the binary extended perfect code of length 2^m.

Example H2E1

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

First, a Hadamard code C over $\mathbb{Z}_4$ of length 8 and type $2^1 4^2$ is defined. The matrix Gc is the quaternary matrix used to generate C and obtained by a recursive method from `Plotkin` and `BQPlotkin` constructions.

```
> C, Gc := HadamardCodeZ4(2,4);
> C;
((8, 4^2 2^1)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 3 2 1 0 3 2]
[0 1 2 3 0 1 2 3]
[0 0 0 0 2 2 2 2]
> Gc;
[1 1 1 1 1 1 1 1]
[0 1 2 3 0 1 2 3]
[0 0 0 0 2 2 2 2]
> HasLinearGrayMapImage(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
```

Then, an extended perfect code D over $\mathbb{Z}_4$ of length 8 is defined, such that its dual code is of type $2^1 4^2$. The matrix Gd is the quaternary matrix which is used to generate D and obtained by a recursive method from `Plotkin` and `BQPlotkin` constructions. Note that the code D is the Kronecker dual code of C.

```
> D, Gd := ExtendedPerfectCodeZ4(2,4);
> D;
((8, 4^5 2^1)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 0 1 0 0 1 3]
[0 1 0 1 0 0 2 2]
[0 0 1 1 0 0 1 1]
[0 0 0 2 0 0 0 2]
[0 0 0 0 1 0 3 2]
[0 0 0 0 0 1 2 3]
> Gd;
[1 1 1 1 1 1 1 1]
[0 1 2 3 0 1 2 3]
[0 0 1 1 0 0 1 1]
[0 0 0 2 0 0 0 2]
[0 0 0 0 1 1 1 1]
[0 0 0 0 0 1 2 3]

> DualKroneckerZ4(C) eq D;
true
```

`ReedMullerCodeZ4(r, m)`

`ReedMullerCodeQRMZ4(r, m)`

Given an integer $m \geq 2$ and an integer r such that $0 \leq r \leq m$, return the r-th order Reed-Muller code over $\mathbb{Z}_4$ of length 2^m.

The binary image under the modulo 2 map is the binary linear r-th order Reed-Muller code of length 2^m. For $r = 1$ and $r = m - 2$, the function returns the quaternary linear Kerdock and Preparata code, respectively.

`ReedMullerCodesLRMZ4(r, m)`

Given an integer $m \geq 1$ and an integer r such that $0 \leq r \leq m$, return a set of r-th order Reed-Muller codes over $\mathbb{Z}_4$ of length 2^{m-1}.

The binary image under the Gray map of any of these codes is a binary (not necessarily linear) code with the same parameters as the binary linear r-th order Reed-Muller code of length 2^m. Note that for these codes neither the usual inclusion nor duality properties of the binary linear Reed-Muller family are satisfied.

`ReedMullerCodeRMZ4(s, r, m)`

Given an integer $m \geq 1$, an integer r such that $0 \leq r \leq m$, and an integer s such that $0 \leq s \leq \lfloor (m-1)/2 \rfloor$, return a r-th order Reed-Muller code over $\mathbb{Z}_4$ of length 2^{m-1}, denoted by $RM_s(r,m)$, as well as the generator matrix used in the recursive construction.

The binary image under the Gray map is a binary (not necessarily linear) code with the same parameters as the binary linear r-th order Reed-Muller code of length 2^m. Note that the inclusion and duality properties are also satisfied, that is, the code $RM_s(r-1,m)$ is a subcode of $RM_s(r,m)$, $r > 0$, and the code $RM_s(r,m)$ is the Kronecker dual code of $RM_s(m-r-1,m)$, $r < m$.

Example H2E2

Taking the Reed-Muller codes $RM_1(1,4)$ and $RM_1(2,4)$, it can be seen that the former is a subcode of the latter. Note that $RM_1(1,4)$ and $RM_1(2,4)$ are the same as the ones given in Example H2E1 by `HadamardCodeZ4(2,4)` and `ExtendedPerfectCodeZ4(2,4)`, respectively.

```
> C1, G1 := ReedMullerCodeRMZ4(1,1,4);
> C2, G2 := ReedMullerCodeRMZ4(1,2,4);

> C1;
((8, 4^2 2^1)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 3 2 1 0 3 2]
[0 1 2 3 0 1 2 3]
[0 0 0 0 2 2 2 2]
> C2;
((8, 4^5 2^1)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 0 1 0 0 1 3]
[0 1 0 1 0 0 2 2]
[0 0 1 1 0 0 1 1]
[0 0 0 2 0 0 0 2]
[0 0 0 0 1 0 3 2]
```

```
[0 0 0 0 0 1 2 3]

> C1 subset C2;
true
> DualKroneckerZ4(C2) eq C1;
true
```

`ReedMullerCodesRMZ4(s, m)`

Given an integer $m \geq 1$, and an integer s such that $0 \leq s \leq \lfloor (m-1)/2 \rfloor$, return a sequence containing the family of Reed-Muller codes over $\mathbb{Z}_4$ of length 2^{m-1}, that is, the codes $RM_s(r, m)$, for all $0 \leq r \leq m$.

The binary image of these codes under the Gray map gives a family of binary (not necessarily linear) codes with the same parameters as the binary linear Reed-Muller family of codes of length 2^m. Note that

$$RM_s(0, m) \subset RM_s(1, m) \subset \cdots \subset RM_s(m, m).$$

Example H2E3

The family of Reed-Muller codes over $\mathbb{Z}_4$ of length 2^2 given by $s = 0$ is constructed.

```
> F := ReedMullerCodesRMZ4(0,3);
> F;
[((4, 4^0 2^1)) Cyclic Linear Code over IntegerRing(4)
Generator matrix:
[2 2 2 2],
((4, 4^1 2^2)) Cyclic Linear Code over IntegerRing(4)
Generator matrix:
[1 1 1 1]
[0 2 0 2]
[0 0 2 2],
((4, 4^3 2^1)) Cyclic Linear Code over IntegerRing(4)
Generator matrix:
[1 0 0 1]
[0 1 0 1]
[0 0 1 1]
[0 0 0 2],
((4, 4^4 2^0)) Cyclic Linear Code over IntegerRing(4)
Generator matrix:
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]]

> F[1] subset F[2] and F[2] subset F[3] and F[3] subset F[4];
true
```

2.3 New Codes from Old

The functions described in this section produce a new code over $\mathbb{Z}_4$ by modifying in some way the codewords of some given codes over $\mathbb{Z}_4$.

`PlotkinSum(A, B)`

Given matrices A and B both over the same ring and with the same number of columns, return the P_{AB} matrix over the same ring of A and B, where

$$P_{AB} = \begin{pmatrix} A & A \\ 0 & B \end{pmatrix}.$$

`PlotkinSum(C, D)`

Given codes C and D both over the same ring and of the same length, construct the Plotkin sum of C and D. The Plotkin sum consists of all vectors of the form $(u|u+v)$, where $u \in C$ and $v \in D$.

Note that the Plotkin sum is computed using generator matrices for C and D and the `PlotkinSum` function for matrices. Thus, this function returns the code over $\mathbb{Z}_4$ generated by the matrix P_{AB} defined above, where A and B are generators matrices for C and D, respectively.

`QuaternaryPlotkinSum(A, B)`

Given two matrices A and B over $\mathbb{Z}_4$, both with the same number of columns, return the QP_{AB} matrix over $\mathbb{Z}_4$, where

$$QP_{AB} = \begin{pmatrix} A & A & A & A \\ 0 & B & 2B & 3B \end{pmatrix}.$$

`QuaternaryPlotkinSum(C, D)`

Given two codes C and D over $\mathbb{Z}_4$, both of the same length, construct the Quaternary Plotkin sum of C and D. The Quaternary Plotkin sum is a code over $\mathbb{Z}_4$ that consists of all vectors of the form $(u, u+v, u+2v, u+3v)$, where $u \in C$ and $v \in D$.

Note that the Quaternary Plotkin sum is computed using generator matrices for C and D and the `QuaternaryPlotkinSum` function for matrices. Thus, this function returns the code over $\mathbb{Z}_4$ generated by the matrix QP_{AB} defined above, where A and B are generators matrices for C and D, respectively.

`BQPlotkinSum(A, B, C)`

Given three matrices A, B, and C over $\mathbb{Z}_4$, all with the same number of columns, return the BQP_{ABC} matrix over $\mathbb{Z}_4$, where

$$BQP_{ABC} = \begin{pmatrix} A & A & A & A \\ 0 & B' & 2B' & 3B' \\ 0 & 0 & \hat{B} & \hat{B} \\ 0 & 0 & 0 & C \end{pmatrix},$$

B' is obtained from B replacing the twos with ones in the rows of order two, and $\hat{B}$ is obtained from B removing the rows of order two.

`BQPlotkinSum(D, E, F)`

Given three codes D, E and F over $\mathbb{Z}_4$, all of the same length, construct the BQ Plotkin sum of D, E and F. Let Ge be a generator matrix for E of type $2^\gamma 4^\delta$. The code E' over $\mathbb{Z}_4$ is obtained from E by replacing the twos with ones in the γ rows of order two of Ge, and the code $\hat{E}$ over $\mathbb{Z}_4$ is obtained from E removing the γ rows of order two of Ge.
The BQ Plotkin sum is a code over $\mathbb{Z}_4$ that consists of all vectors of the form $(u, u+v', u+2v'+\hat{v}, u+3v'+\hat{v}+z)$, where $u \in Gd$, $v' \in Ge'$ $\hat{v} \in \hat{Ge}$, and $z \in Gf$, where Gd, Ge', $\hat{Ge}$ and Gf are generators matrices for D, E', $\hat{E}$ and F, respectively.

Note that the BQPlotkin sum is computed using generator matrices for D, E and F and the `BQPlotkinSum` function for matrices. However, this function does not necessarily return the same code over $\mathbb{Z}_4$ as that generated by the matrix BQP_{ABC} defined above, where A, B and C are generators matrices for D, E and F, respectively, as shown in Example H2E4.

`DoublePlotkinSum(A, B, C, D)`

Given four matrices A, B, C, and D over $\mathbb{Z}_4$, all with the same number of columns, return the DP_{ABC} matrix over $\mathbb{Z}_4$, where

$$DP_{ABCD} = \begin{pmatrix} A & A & A & A \\ 0 & B & 2B & 3B \\ 0 & 0 & C & C \\ 0 & 0 & 0 & D \end{pmatrix}.$$

`DoublePlotkinSum(E, F, G, H)`

Given four codes E, F, G and H over $\mathbb{Z}_4$, all of the same length, construct the Double Plotkin sum of E, F, G and H. The Double Plotkin sum is a code over $\mathbb{Z}_4$ that consists of all vectors of the form $(u, u+v, u+2v+z, u+3v+z+t)$, where $u \in E$, $v \in F$, $z \in G$ and $t \in H$.

Note that the Double Plotkin sum is computed using generator matrices of E, F, G and H and the `DoublePlotkinSum` function for matrices, that is, this function returns the code over $\mathbb{Z}_4$ generated by the matrix DP_{ABCD} defined above, where A, B, C and D are generators matrices for E, F, G and H, respectively.

`DualKroneckerZ4(C)`

Given a code C over $\mathbb{Z}_4$ of length 2^m, return its Kronecker dual code. The Kronecker dual code of C is $C_{\otimes}^{\perp} = \{x \in \mathbb{Z}_4^{2^m} : x \cdot K_{2^m} \cdot y^t = 0, \forall y \in C\}$, where $K_{2^m} = \otimes_{j=1}^{m} K_2$, $K_2 = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix}$ and $\otimes$ denotes the Kronecker product of matrices. Equivalently, K_{2^m} is a quaternary matrix of length 2^m with the vector $(1,3,3,1,3,1,1,3,\ldots)$ in the main diagonal and zeros elsewhere.

Example H2E4

The purpose of this example is to show that the codes over $\mathbb{Z}_4$ constructed from the `BQPlotkinSum` function for matrices are not necessarily the same as the ones constructed from the `BQPlotkinSum` function for codes.

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

> Ga := Matrix(Z4,1,2,[1,1]);
> Gb := Matrix(Z4,2,2,[1,2,0,2]);
> Gc := Matrix(Z4,1,2,[2,2]);

> Ca := LinearCode(Ga);
> Cb := LinearCode(Gb);
> Cc := LinearCode(Gc);

> C := LinearCode(BQPlotkinSum(Ga, Gb, Gc));
> D := BQPlotkinSum(Ca, Cb, Cc);
> C eq D;
false
```

Example H2E5

```
> Ga := GeneratorMatrix(ReedMullerCodeRMZ4(1,2,3));
> Gb := GeneratorMatrix(ReedMullerCodeRMZ4(1,1,3));
> Gc := GeneratorMatrix(ReedMullerCodeRMZ4(1,0,3));

> C := ReedMullerCodeRMZ4(1,2,4);
> Cp := LinearCode(PlotkinSum(Ga, Gb));
> C eq Cp;
true

> D := ReedMullerCodeRMZ4(2,2,5);
> Dp := LinearCode(BQPlotkinSum(Ga, Gb, Gc));
> D eq Dp;
true
```

2.4 Invariants

`StandardFormDual(C)`

Given a code C over $\mathbb{Z}_4$ of length n, 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. Since S is generated by a matrix of the form

$$\begin{pmatrix} I_{k_1} & A & B \\ 0 & 2I_{k_2} & 2C \end{pmatrix},$$

the dual of S is generated by the matrix

$$\begin{pmatrix} -(AC+B)^t & C^t & I_{n-k_1-k_2} \\ 2A^t & 2I_{k_2} & 0 \end{pmatrix},$$

where I_{k_1} and I_{k_2} are the $k_1 \times k_1$ and $k_2 \times k_2$ identity matrices, respectively, A and C are $\mathbb{Z}_2$-matrices, and B is a $\mathbb{Z}_4$-matrix.

`MinRowsGeneratorMatrix(C)`

A generator matrix for the code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, with the minimum number of rows, that is with $\gamma + \delta$ rows: γ rows of order two and δ rows of order four. It also returns the parameters γ and δ.

`MinRowsParityCheckMatrix(C)`

A parity check matrix for the code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, with the minimum number of rows, that is, with γ rows of order two and $n-\gamma-\delta$ rows of order four. This function should be faster for most codes over $\mathbb{Z}_4$ 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 `MinRowsGeneratorMatrix(DualZ4(C))`.

`DualZ4(C)`

The dual D of the code C over $\mathbb{Z}_4$ of length n. The dual consists of all codewords in the $\mathbb{Z}_4$-space $V = \mathbb{Z}_4^n$ which are orthogonal to all codewords of C. This function should be faster for most codes over $\mathbb{Z}_4$ than the general function `Dual(C)` for codes over finite rings.

Example H2E6

```
> C := HadamardCodeZ4(3, 11);
> G, gamma, delta := MinRowsGeneratorMatrix(C);
> Nrows(G) eq gamma + delta;
true
> deltaH := Length(C) - gamma - delta;
> H1 := MinRowsParityCheckMatrix(C);
> Nrows(H1) eq gamma + deltaH;
true
> H2 := MinRowsGeneratorMatrix(DualZ4(C));
> Nrows(H2) eq gamma + deltaH;
true

> time D := Dual(C);
Time: 24.660
> time D4 := DualZ4(C);
Time: 0.340
> D eq D4, D4 eq LinearCode(H1), D4 eq LinearCode(H2);
true true true

> DualS, f := StandardFormDual(C);
> DualS eq LinearCode(Matrix([f(H1[i]) : i in [1..Nrows(H1)]]));
true
```

`SpanZ2CodeZ4(C)`

Given a code C over $\mathbb{Z}_4$ of length n, return $S_C = \Phi^{-1}(S_{bin})$ as a code over $\mathbb{Z}_4$, and the linear span of C_{bin}, $S_{bin} = \langle C_{bin} \rangle$, as a binary linear code of length $2n$, where $C_{bin} = \Phi(C)$ and Φ is the Gray map.

`KernelZ2CodeZ4(C)`

Given a code C over $\mathbb{Z}_4$ of length n, return its kernel K_C as a subcode over $\mathbb{Z}_4$ of C, and $K_{bin} = \Phi(K_C)$ as a binary linear subcode of C_{bin} of length $2n$, where $C_{bin} = \Phi(C)$ and Φ is the Gray map.

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

`KernelCosetRepresentatives(C)`

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

`DimensionOfSpanZ2(C)`

`RankZ2(C)`

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

`DimensionOfKernelZ2(C)`

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

Example H2E7

```
> C := ReedMullerCodeRMZ4(0,3,5);

> DimensionOfKernelZ2(C);
20
> DimensionOfSpanZ2(C);
27

> K, Kb := KernelZ2CodeZ4(C);
> S, Sb := SpanZ2CodeZ4(C);

> K subset C;
true
> C subset S;
true

> Dimension(Kb) eq DimensionOfKernelZ2(C);
true
> Dimension(Sb) eq DimensionOfSpanZ2(C);
true
```

2.5 Coset Representatives

`CosetRepresentatives(C)`

Given a code C over $\mathbb{Z}_4$ of length n, with ambient space $V = \mathbb{Z}_4^n$, 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 the two conditions that c_0 is the zero codeword, and $V = \bigcup_{i=0}^{t} \big(C + c_i\big)$. Note that this function is only applicable when V and C are small.

`CosetRepresentatives(C, S)`

Given a code C over $\mathbb{Z}_4$ of length n, and a subcode S over $\mathbb{Z}_4$ 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 the two conditions that c_0 is the zero codeword, and $C = \bigcup_{i=0}^{t} \big(S + c_i\big)$. Note that this function is only applicable when S and C are small.

Example H2E8

```
> C := LinearCode<Integers(4), 4 | [[1,0,0,3],[0,1,1,3]]>;
> L := CosetRepresentatives(C);
> Set(RSpace(Integers(4),4)) eq {v+ci : v in Set(C), ci in L};
true

> K := KernelZ2CodeZ4(C);
> L := CosetRepresentatives(C, K);
> {C!0} join Set(KernelCosetRepresentatives(C)) eq L;
true
> Set(C) eq {v+ci : v in Set(K), ci in L};
true
```

2.6 Information Space and Information Sets

`InformationSpace(C)`

Given a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, return the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{\gamma+\delta}$ isomorphic to $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^\delta$ such that the first γ coordinates are of order two, that is, the space of information vectors for C. The function also returns the $(\gamma + 2\delta)$-dimensional binary vector space, which is the space of information vectors for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map. Finally, for the encoding process, it also returns the corresponding isomorphisms f and f_{bin} from these spaces of information vectors onto C and C_{bin}, respectively.

Example H2E9

```
> C := LinearCode<Integers(4), 4 | [[2,0,0,2],[0,1,1,3]]>;
> R, V, f, fbin := InformationSpace(C);
> G := MinRowsGeneratorMatrix(C);

> (#R eq #C) and (#V eq #C);
true
> Set([f(i) : i in R]) eq Set(C);
true
> Set([i*G : i in R]) eq Set(C);
false

> i := R![2,3];
> c := f(i);
> c;
(2 3 3 3)
> c in C;
true
> i*G eq c;
false

> ibin := V![1,1,0];
> cbin := fbin(ibin);
> cbin;
(1 1 1 0 1 0 1 0)
> cbin in GrayMapImage(C);
true
> cbin eq GrayMap(C)(c);
true
```

`InformationSet(C)`

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

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

`IsInformationSet(C, I)`

Given a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$ and a sequence $I \subseteq \{1, \ldots, n\}$ or $I \subseteq \{1, \ldots, 2n\}$, 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 $I \subseteq \{1, \ldots, 2n\}$ is an information set for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map.

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

Example H2E10

```
> C := HadamardCodeZ4(3,6);
> C;
((32, 4^3 2^1)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 3 2 0 3 2 1 3 2 1 0 2 1 0 3 1 0 3 2 0 3 2 1 3 2 1 0 2 1 0 3]
[0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3]
[0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]

> I, Ibin := InformationSet(C);
> I;
[ 16, 28, 31, 32 ]
> Ibin;
[ 31, 55, 56, 61, 62, 63, 64 ]
> #PunctureCode(C, {1..32} diff Set(I)) eq #C;
true
> Cbin := GrayMapImage(C);
> V := VectorSpace(GF(2), 7);
> #{V![c[i] : i in Ibin] : c in Cbin} eq #Cbin;
true

> IsInformationSet(C, I);
true false
> IsInformationSet(C, Ibin);
false true

> IsInformationSet(C, [1, 2, 5, 17]);
true false
> IsInformationSet(C, [1, 2, 3, 4, 9, 10, 33]);
false true

> D := LinearCode<Integers(4), 5 | [[2,0,0,2,0],[0,2,0,2,2],[0,0,2,2,0]]>;
> IsInformationSet(D, [1,3,5]);
true true
```

2.7 Syndrome Space and Coset Leaders

`SyndromeSpace(C)`

Given a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, return the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{n-\delta}$ isomorphic to $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^{n-\gamma-\delta}$ such that the first γ coordinates are of order two, that is, the space of syndrome vectors for C. The function also returns the $(2n-2\delta-\gamma)$-dimensional binary vector space, which is the space of syndrome vectors for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map. Note that these spaces are computed by using the function `InformationSpace(C)` applied to the dual code of C, produced by function `DualZ4(C)`.

`Syndrome(u, C)`

Given a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, and a vector u from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$, construct the syndrome of u relative to the code C. This will be an element of the syndrome space of C, considered as the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{n-\delta}$ isomorphic to $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^{n-\gamma-\delta}$ such that the first γ coordinates are of order two.

`CosetLeaders(C)`

Given a code C over $\mathbb{Z}_4$ of length n, with ambient space $V = \mathbb{Z}_4^n$, return a set of coset leaders (vectors of minimal Lee weight in their cosets) for C in V as an indexed set of vectors from V. This function also returns a map from the syndrome space of C onto the coset leaders (mapping a syndrome into its corresponding coset leader). Note that this function is only applicable when V and C are small.

Example H2E11

```
> C := LinearCode<Integers(4), 4 | [[2,0,0,2],[0,1,1,3]]>;
> R, V, f, fbin := InformationSpace(C);
> Rs, Vs := SyndromeSpace(C);

> #R * #Rs eq 4^Length(C);
true
> #V * #Vs eq 4^Length(C);
true

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

> s := Syndrome(u, C);
> s in Rs;
true
> H := Transpose(MinRowsParityCheckMatrix(C));
> s eq u*H;
true

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

2.8 Decoding

This section describes functions for decoding vectors from the ambient space of a code over $\mathbb{Z}_4$, or the corresponding space over $\mathbb{Z}_2$ under the Gray map, using four different algorithms: coset decoding, syndrome decoding, lifted decoding and permutation decoding. The reader is referred to [9, 10, 23] for more information on coset decoding; to [13, 16, 24] on syndrome decoding; to [1, 12] on lifted decoding; and to [2, 3, 4] on permutation decoding.

2.8.1 Coset Decoding

`CosetDecode(C, u : parameters)`

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

Given a code C over $\mathbb{Z}_4$ of length n, and a vector u from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$, attempt to decode u with respect to C. If the decoding algorithm succeeds in computing a vector $u' \in C$ as the decoded version of $u \in V$, then the function returns `true`, u' and $\Phi(u')$, where Φ is the Gray map. If the decoding algorithm does not succeed in decoding u, then the function returns `false`, the zero vector in V and the zero vector in V_2.

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

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

`CosetDecode(C, Q : parameters)`

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

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

Example H2E12

Starting with the Hadamard code C over $\mathbb{Z}_4$ of length 16 and type $2^0 4^3$, a codeword $c \in C$ is selected and then perturbed to give a vector u in the ambient space of C. The vector u is then decoded to recover c.

```
> C := HadamardCodeZ4(3, 5);
> C;
((16, 4^3 2^0)) Linear Code over IntegerRing(4)
Generator matrix:
```

```
[1 0 3 2 0 3 2 1 3 2 1 0 2 1 0 3]
[0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3]
[0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3]
> d := MinimumLeeDistance(C);
> t := Floor((d-1)/2);
> t;
7

> c := C ! [1,1,1,1,2,2,2,2,3,3,3,3,0,0,0,0];
> c in C;
true
> u := c;
> u[5] := u[5] + 2;
> u[12] := u[12] + 1;
> u[13] := u[13] + 3;
> u[16] := u[16] + 2;
> c;
(1 1 1 1 2 2 2 2 3 3 3 3 0 0 0 0)
> u;
(1 1 1 1 0 2 2 2 3 3 3 0 3 0 0 2)
> grayMap := GrayMap(UniverseCode(Integers(4), Length(C)));
> grayMap(c-u);
(0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 1)

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

2.8.2 Syndrome Decoding

`SyndromeDecode(C, u)`

Given a code C over $\mathbb{Z}_4$ of length n, and a vector u from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$, attempt to decode u with respect to C. The decoding algorithm always succeeds in computing a vector $u' \in C$ as the decoded version of $u \in V$, and the function returns `true`, u' and $\Phi(u')$, where Φ is the Gray map. Although the function never returns `false`, the first output parameter `true` is given to be consistent with the other decoding functions.

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

`SyndromeDecode(C, Q)`

Given a code C over $\mathbb{Z}_4$ of length n, and a sequence Q of vectors from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$, attempt to decode the vectors of Q with respect to C. This function is similar to the function `SyndromeDecode(C, u)` except that rather than decoding a single vector, it decodes a sequence of vectors and returns a sequence of booleans and two sequences of decoded vectors corresponding to the given sequence. The algorithm used is the same as that of function `SyndromeDecode(C, u)`.

Example H2E13

The Hadamard code C over $\mathbb{Z}_4$ of length 8 and type $2^1 4^2$ is constructed. Next, information bits are encoded using C and three errors are introduced to give the vector u. Then u is decoded by calculating its syndrome and applying the map, given by the `CosetLeaders` function, to the syndrome to recover the original vector.

```
> C := HadamardCodeZ4(2, 4);
> C;
((8, 4^2 2^1, 8)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 3 2 1 0 3 2]
[0 1 2 3 0 1 2 3]
[0 0 0 0 2 2 2 2]
> t := Floor((MinimumLeeDistance(C)-1)/2);
> t;
3

> R, V, f, fbin := InformationSpace(C);
> i := R![2,1,0];
> c := f(i);
> c;
(1 0 3 2 3 2 1 0)
> u := c;
> u[5] := u[5] + 3;
> u[7] := u[7] + 2;
> c;
(1 0 3 2 3 2 1 0)
> u;
(1 0 3 2 2 2 3 0)
> grayMap := GrayMap(UniverseCode(Integers(4), Length(C)));
> grayMap(c-u);
(0 0 0 0 0 0 0 0 1 0 0 1 1 0 0)

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

```
true

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

2.8.3 Lifted Decoding

`LiftedDecode(C, u : parameters)`

`AlgMethod` MonStgElt *Default* : "Euclidean"

Given a code C over $\mathbb{Z}_4$ of length n, and a vector u from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$, attempt to decode u with respect to C. If the decoding algorithm succeeds in computing a vector $u' \in C$ as the decoded version of $u \in V$, then the function returns `true`, u' and $\Phi(u')$, where Φ is the Gray map. If the decoding algorithm does not succeed in decoding u, then the function returns `false`, the zero vector in V and the zero vector in V_2 (in the Euclidean case it may happen that u' is not in C because there are too many errors in u to correct).

The lifted decoding algorithm comprises lifting decoding algorithms for two binary linear codes C_0 and C_1, being the residue and torsion codes of C. Let t_0 and t_1 be the error-correcting capability of C_0 and C_1, respectively. Assume the received vector $u = c + e$, where $c \in C$ and $e \in V$ is the error vector. Then, the lifted decoding algorithm can correct all error vectors e such that $\tau_1 + \tau_3 \leq t_0$ and $\tau_2 + \tau_3 \leq t_1$, where τ_i is the number of occurrences of i in e.

In the decoding process, the function `Decode(C, u)` for linear codes is used. The accessible algorithms for linear codes are: syndrome decoding and a Euclidean algorithm, which operates on alternant codes (BCH, Goppa, and Reed–Solomon codes, etc.). If C_0 or C_1 is alternant, the Euclidean algorithm is used by default, but the syndrome algorithm will be used if the parameter `AlgMethod` is assigned the value `"Syndrome"`. For non-alternant codes C_0 and C_1, only syndrome decoding is possible, so the parameter `AlgMethod` is not relevant.

`LiftedDecode(C, Q : parameters)`

`AlgMethod` MonStgElt *Default* : "Euclidean"

Given a code C over $\mathbb{Z}_4$ of length n, and a sequence Q of vectors from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$, attempt to decode the vectors of Q with respect to C. This function is similar to the function `LiftedDecode(C, u)` except that rather than decoding a single vector, it decodes a sequence of vectors and returns a sequence of booleans and two sequences of decoded vectors corresponding to the given sequence. The algorithm used and effect of the parameter `AlgMethod` are the same as for `LiftedDecode(C, u)`.

Example H2E14

The Hadamard code C over $\mathbb{Z}_4$ of length 8 and type $2^1 4^2$ is constructed. Then an information word is encoded using C, three errors are introduced into the codeword c, and then c is recovered by using the lifted decoding algorithm.

```
> C := HadamardCodeZ4(2, 4);
> C;
((8, 4^2 2^1, 8)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 3 2 1 0 3 2]
[0 1 2 3 0 1 2 3]
[0 0 0 0 2 2 2 2]
> d := MinimumLeeDistance(C);
> t := Floor((d-1)/2);
> t;
3
> C0 := BinaryResidueCode(C);
> C1 := BinaryTorsionCode(C);
> t0 := Floor((MinimumDistance(C0)-1)/2);
> t1 := Floor((MinimumDistance(C1)-1)/2);
> t0, t1;
1 1
```

Using the lifted decoding, it is possible to correct all error vectors e such that $\tau_1 + \tau_3 \leq t_0 = 1$ and $\tau_2 + \tau_3 \leq t_1 = 1$, where τ_i is the number of occurrences of i in e. The following statements show that it is not possible to correct the error vector $e = (00003020)$ since $\tau_2 + \tau_3 = 2 > 1$, but it is possible to correct the error vector $e = (00001020)$ since $\tau_1 + \tau_3 = 1 \leq 1$ and $\tau_2 + \tau_3 = 1 \leq 1$.

```
> R, V, f, fbin := InformationSpace(C);
> i := R![2,1,0];
> c := f(i);
> c;
(1 0 3 2 3 2 1 0)
```

```
> u := c;
> u[5] := u[5] + 3;
> u[7] := u[7] + 2;
> c;
(1 0 3 2 3 2 1 0)
> u;
(1 0 3 2 2 2 3 0)
> e := u - c;
> e;
(0 0 0 0 3 0 2 0)

> isDecoded, uDecoded := LiftedDecode(C, u);
> isDecoded;
true
> uDecoded eq c;
false

> u := c;
> u[5] := u[5] + 1;
> u[7] := u[7] + 2;
> c;
(1 0 3 2 3 2 1 0)
> u;
(1 0 3 2 0 2 3 0)
> e := u - c;
> e;
(0 0 0 0 1 0 2 0)

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

2.8.4 Permutation Decoding

Let C be a code over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$ and $C_{bin} = \Phi(C)$, where Φ is the Gray map. A subset $S \subseteq \mathrm{Sym}(2n)$ is an s-PD-set for C_{bin} with respect to a subset of coordinate positions $I \subseteq \{1, \ldots, 2n\}$ if S is a subset of the permutation automorphism group of C_{bin}, I is an information set for C_{bin}, and every s-set of coordinate positions in $\{1, \ldots, 2n\}$ is moved out of the information set I by at least one element of S, where $1 \leq s \leq t$ and t is the error-correcting capability of C_{bin}.

If $I = [i_1, \ldots, i_{\gamma+\delta}] \subseteq \{1, \ldots, n\}$ is an information set for C such that the code obtained by puncturing C at positions $\{1, \ldots, n\} \backslash \{i_{\gamma+1}, \ldots, i_{\gamma+\delta}\}$ is of type 4^δ, then $\Phi(I) = [2i_1 - 1, \ldots, 2i_\gamma - 1, 2i_{\gamma+1} - 1, 2i_{\gamma+1}, \ldots, 2i_{\gamma+\delta} - 1, 2i_{\gamma+\delta}]$

is an information set for C_{bin}. It is also easy to see that if S is a subset of the permutation automorphism group of C, that is, $S \subseteq \mathrm{PAut}(C) \subseteq \mathrm{Sym}(n)$, then $\Phi(S) = [\Phi(\tau) : \tau \in S] \subseteq \mathrm{PAut}(C_{bin}) \subseteq \mathrm{Sym}(2n)$, where

$$\Phi(\tau)(i) = \begin{cases} 2\tau(i/2), & \text{if } i \text{ is even,} \\ 2\tau((i+1)/2) - 1 & \text{if } i \text{ is odd.} \end{cases} \tag{2.1}$$

Given a subset of coordinate positions $I \subseteq \{1, \ldots, n\}$ and a subset $S \subseteq \mathrm{Sym}(n)$, in order to check that $\Phi(S)$ is an s-PD-set for C_{bin} with respect to $\Phi(I)$, it is enough to check that S is a subset of the permutation automorphism group of C, I is an information set for C, and every s-set of coordinate positions in $\{1, \ldots, n\}$ is moved out of the information set I by at least one element of S [2, 3].

`IsPermutationDecodeSet(C, I, S, s)`

Given a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, a sequence $I \subseteq \{1, \ldots, 2n\}$, a sequence S of elements in the symmetric group $\mathrm{Sym}(2n)$ of permutations on the set $\{1, \ldots, 2n\}$, and an integer $s \geq 1$, return `true` if and only if S is an s-PD-set for $C_{bin} = \Phi(C)$, where Φ is the 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 s-PD-set for $C_{bin} = \Phi(C)$ with respect to the information set $\Phi(I)$, where $\Phi(I)$ and $\Phi(S)$ are the sequences defined as above.

Depending on the length of the code C, its type, and the integer s, this function could take some time to compute whether S or $\Phi(S)$ is an s-PD-set for C_{bin} with respect to I or $\Phi(I)$, respectively. Specifically, if the function returns `true`, it is necessary to check $\sum_{i=1}^{s} \binom{|I|}{i} \cdot \binom{N-|I|}{s-i}$ s-sets, where $N = n$ and $|I| = \gamma + \delta$ when I is given as an information set for C, or $N = 2n$ and $|I| = \gamma + 2\delta$ when I is given as an information set for C_{bin}.

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 return `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 s-PD-set. If it is set to level 2, the percentage of the computation process performed is also printed.

`PermutationDecode(C, I, S, s, u)`

The arguments for the intrinsic are as follows:

- C is a code over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$;

- $I = [i_1, \ldots, i_{\gamma+\delta}] \subseteq \{1, \ldots, n\}$ is an information set for C given as a sequence of coordinate positions such that the code C obtained by puncturing C at coordinate positions $\{1, \ldots, n\} \backslash \{i_{\gamma+1}, \ldots, i_{\gamma+\delta}\}$ is of type 4^δ;

- S is a sequence S such that either S or $\Phi(S)$ is an s-PD-set for $C_{bin} = \Phi(C)$, where Φ is the Gray map, with respect to $\Phi(I)$;

- s is an integer such that $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C_{bin};

- u is a vector from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$,

Given the above assumptions, the function attempts to decode u with respect to C. If the decoding algorithm succeeds in computing a vector $u' \in C$ as the decoded version of $u \in V$, then the function returns the values `true`, u' and $\Phi(u')$. If the decoding algorithm does not succeed in decoding u, then the function returns the values `false`, the zero vector in V and the zero vector in V_2.

Assume that the received vector $\Phi(u) = c + e$, where $u \in V$, $c \in C_{bin}$ and $e \in V_2$ is the error vector with at most t errors. The permutation decoding algorithm proceeds by moving all errors in a received vector $\Phi(u)$ out of the information positions. That is, the nonzero coordinates of e are moved out of the information set $\Phi(I)$ for C_{bin}, by using an automorphism of C_{bin}.

Note that $\Phi(I)$ and $\Phi(S)$ are the sequences defined as above. Moreover, the function does not check whether I is an information set for C, nor whether S or $\Phi(S)$ is an s-PD-set for C_{bin} with respect to $\Phi(I)$, nor that $s \leq t$.

`PermutationDecode(C, I, S, s, Q)`

Given

- a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$;
- an information set $I = [i_1, \ldots, i_{\gamma+\delta}] \subseteq \{1, \ldots, n\}$ for C as a sequence of coordinate positions, such that the code C punctured on $\{1, \ldots, n\} \backslash \{i_{\gamma+1}, \ldots, i_{\gamma+\delta}\}$ is of type 4^δ;
- a sequence S such that either S or $\Phi(S)$ is an s-PD-set for $C_{bin} = \Phi(C)$, where Φ is the Gray map, with respect to $\Phi(I)$;
- an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C_{bin};
- and a sequence Q of vectors from the ambient space $V = \mathbb{Z}_4^n$ or $V_2 = \mathbb{Z}_2^{2n}$,

attempt to decode the vectors of Q with respect to C. This function is similar to the version of `PermutationDecode` that decodes a single vector except that it decodes a sequence of vectors and returns a sequence of booleans and two sequences of decoded vectors corresponding to the given sequence. The algorithm used is the same as that used by the single vector version of `PermutationDecode`.

Example H2E15

First the Hadamard code C over $\mathbb{Z}_4$ of length 32 and type $2^1 4^3$ is constructed. It is known that $I = [17, 1, 2, 5]$ is an information set for C and $S = \{\pi^i : 1 \leq i \leq 8\}$, where $\pi = (1, 24, 26, 15, 3, 22, 28, 13)$ $(2, 23, 27, 14, 4, 21, 25, 16)$ $(5, 11, 32, 20, 7, 9, 30, 18)$ $(6, 10, 29, 19, 8, 12, 31, 17)$, is a subset of the permutation automorphism group of C such that $\Phi(S)$ is a 7-PD-set for $C_{bin} = \Phi(C)$ with respect to $\Phi(I)$. Then, choosing a codeword c of C, c is perturbed by the addition of an error vector to give a new vector u, and finally permutation decoding is applied to u to recover c.

```
> C := HadamardCodeZ4(3, 6);
> C;
((32, 4^3 2^1)) Linear Code over IntegerRing(4)
Generator matrix:
[1 0 3 2 0 3 2 1 3 2 1 0 2 1 0 3 1 0 3 2 0 3 2 1 3 2 1 0 2 1 0 3]
[0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3]
[0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
> t := Floor((MinimumLeeDistance(C) - 1)/2);
> t;
15
```

```
> I := [17, 1, 2, 5];
> p := Sym(32)!(1, 24, 26, 15, 3, 22, 28, 13)(2, 23, 27, 14, 4, 21, 25, 16)
               (5, 11, 32, 20, 7, 9, 30, 18)(6, 10, 29, 19, 8, 12, 31,17);
> S := [ p^i : i in [1..8] ];
> SetVerbose("IsPDSetFlag", 2);
> IsPermutationDecodeSet(C, I, S, 7);
Checking whether I is an information set...
Checking whether S is in the permutation automorphism group...
Checking whether S is an s-PD-set...
10 %
20 %
30 %
40 %
50 %
60 %
70 %
80 %
90 %
Took 136.430 seconds (CPU time)
true
> SetVerbose("IsPDSetFlag", 0);

> c := C ! [1,2,3,0,0,1,2,3,3,0,1,2,2,3,0,1,3,0,1,2,2,3,0,1,1,2,3,0,0,1,2,3];
> c in C;
true
> u := c;
> u[1] := c[1] + 2;
> u[2] := c[2] + 2;
> u[3] := c[3] + 1;
> u[16] := c[16] + 3;
> u[27] := c[27] + 1;
> u in C;
false
> LeeDistance(u, c);
7

> grayMap := GrayMap(UniverseCode(Integers(4), Length(C)));
> cbin := grayMap(c);
> ubin := grayMap(u);
> Distance(ubin, cbin);
7

> isDecoded, uDecoded, ubinDecoded := PermutationDecode(C, I, S, 7, u);
> isDecoded;
true
> uDecoded eq c;
true
> ubinDecoded eq cbin;
true
```

```
> isDecoded, uDecoded, ubinDecoded := PermutationDecode(C, I, S, 7, ubin);
> isDecoded;
true
> uDecoded eq c;
true
> ubinDecoded eq cbin;
true
```

`PDSetHadamardCodeZ4(δ, m)`

`AlgMethod` MonStgElt *Default* : "Deterministic"

Given an integer $m \geq 5$, and an integer δ such that $3 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, the Hadamard code C over $\mathbb{Z}_4$ of length $n = 2^{m-1}$ and type $2^\gamma 4^\delta$, where $\gamma = m + 1 - 2\delta$, given by the function `HadamardCodeZ4(δ, m)`, is considered. The function returns an information set $I = [i_1, \ldots, i_{\gamma+\delta}] \subseteq \{1, \ldots, n\}$ for C together with a subset S of the permutation automorphism group of C such that $\Phi(S)$ is an s-PD-set for $C_{bin} = \Phi(C)$ with respect to $\Phi(I)$, where Φ is the Gray map and $\Phi(I)$ and $\Phi(S)$ are defined above. The function also returns the information set $\Phi(I)$ and the s-PD-set $\Phi(S)$. For $m \geq 1$ and $1 \leq \delta \leq 2$, the Gray map image of C is linear and it is possible to find an s-PD-set for $C_{bin} = \Phi(C)$, for any $s \leq \lfloor 2^m/(m+1) \rfloor - 1$, by using the function `PDSetHadamardCode(m)`.

The information sets I and $\Phi(I)$ are returned as sequences of $\gamma + \delta$ and $\gamma + 2\delta$ integers, giving the coordinate positions that correspond to the information sets for C and C_{bin}, respectively. The sets S and $\Phi(S)$ are also returned as sequences of elements in the symmetric groups $\mathrm{Sym}(n)$ and $\mathrm{Sym}(2n)$ of permutations on the set $\{1, \ldots, n\}$ and $\{1, \ldots, 2n\}$, respectively.

A deterministic algorithm is used by default. In this case, the function returns the s-PD-set of size $s+1$ with $s = \lfloor (2^{2\delta-2} - \delta)/\delta \rfloor$, which is the maximum value of s when $\gamma = 0$, as described in [2]. If the parameter `AlgMethod` is assigned the value `"Nondeterministic"`, the function tries to improve the previous result by finding an s-PD-set of size $s+1$ such that $\lfloor (2^{2\delta-2} - \delta)/\delta \rfloor \leq s \leq \lfloor (2^{m-1} + \delta - m - 1)/(m + 1 - \delta) \rfloor$. In this case, the function starts from the maximum value of s and decreases it if the s-PD-set is not found after a specified time.

`PDSetKerdockCodeZ4(m)`

Given an integer $m \geq 4$ such that $2^m - 1$ is not a prime number, the Kerdock code C over $\mathbb{Z}_4$ of length $n = 2^m$ and type 4^{m+1}, given by the function `KerdockCodeZ4(m)` is considered. The function returns the information set $I = [1, \ldots, m+1]$ for C together with a subset S of the permutation automorphism group of C such that $\Phi(S)$ is an s-PD-set for $C_{bin} = \Phi(C)$ with respect to $\Phi(I)$, where Φ is the Gray map and $\Phi(I)$ and $\Phi(S)$ are defined above. The function also returns the information set $\Phi(I) = [1, \ldots, 2m+2]$ and the s-PD-set $\Phi(S)$. The size of the s-PD-set S is always $\lambda = s + 1$, where λ is the greatest divisor of $2^m - 1$ such that $\lambda \leq 2^m/(m+1)$.

The information sets I and $\Phi(I)$ are returned as sequences of $m+1$ and $2m+2$ integers, giving the coordinate positions that correspond to the information sets for C and C_{bin}, respectively. The sets S and $\Phi(S)$ are also returned as sequences of elements in the symmetric groups Sym(n) and Sym$(2n)$ of permutations on the sets $\{1, \ldots, n\}$ and $\{1, \ldots, 2n\}$, respectively. The s-PD-set S contains the $s+1$ permutations described in [3].

Example H2E16

A 4-PD-set S of size 5 for the Hadamard code C over $\mathbb{Z}_4$ of length 16 and type $2^0 4^3$ is constructed. A check that it really is a 4-PD-set for C is then made. Note that $\lfloor (2^{2\delta-2} - \delta)/\delta \rfloor = 4$. Finally, a codeword c of C is selected, perturbed by an error vector e to give a vector u, to which permutation decoding is applied to recover c.

```
> C := HadamardCodeZ4(3, 5);

> I, S, Ibin, Sbin := PDSetHadamardCodeZ4(3, 5);
> s := #Sbin-1; s;
4
> s eq Floor((2^(2*3-2)-3)/3);
true
> IsPermutationDecodeSet(C, I, S, s);
true
> IsPermutationDecodeSet(C, Ibin, Sbin, s);
true

> c := C ! [3,2,1,0,1,0,3,2,3,2,1,0,1,0,3,2];
> R := UniverseCode(Integers(4), Length(C));
> u := R ! [2,3,2,0,1,0,3,2,3,2,1,0,1,0,3,3];
> u in C;
false
> LeeDistance(u, c);
4
```

```
> grayMap := GrayMap(R);
> cbin := grayMap(c);

> isDecoded, uDecoded, ubinDecoded := PermutationDecode(C, I, S, 4, u);
> isDecoded;
true
> uDecoded eq c;
true
> ubinDecoded eq cbin;
true
```

For the Hadamard code C over $\mathbb{Z}_4$ of length 32 and type $2^1 4^3$, a 4-PD-set of size 5 can be constructed either by using the deterministic method (by default), or by using a nondeterministic method to obtain an s-PD-set of size $s+1$ with $4 \leq s \leq 7$. In both cases, the given sets are checked for really being s-PD-sets for C.

```
> C := HadamardCodeZ4(3, 6);

> I, S, Ibin, Sbin := PDSetHadamardCodeZ4(3, 6);
> s := #Sbin-1; s;
4
> IsPermutationDecodeSet(C, I, S, s);
true

> I, S, Ibin, Sbin := PDSetHadamardCodeZ4(3, 6 : AlgMethod := "Nondeterministic");
> s := #Sbin-1; s;
6
> IsPermutationDecodeSet(C, I, S, s);
true
```

Finally, a 2-PD-set of size 3 is constructed for the Kerdock code of length 16 and type $2^0 4^5$, and formally checked for being a 2-PD-set for this code.

```
> C := KerdockCode(4);

> I, S, Ibin, Sbin := PDSetKerdockCodeZ4(4);
> IsPermutationDecodeSet(C, I, S, 2);
true
> IsPermutationDecodeSet(C, Ibin, Sbin, 2);
true
```

2.9 Automorphism Groups

`PermutationGroupHadamardCodeZ4(`δ`, m)`

`PAutHadamardCodeZ4(`δ`, m)`

Given an integer $m \geq 1$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, this function returns the permutation group G of a Hadamard code over $\mathbb{Z}_4$ of length 2^{m-1} and type $2^\gamma 4^\delta$, where $\gamma = m + 1 - 2\delta$. The group G contains all permutations of the coordinates which preserve the code. Thus only permutation of coordinates is allowed, and the degree of G is always 2^{m-1}. Moreover, the generator matrix with $\gamma + \delta$ rows used to generate the code is returned. This matrix is constructed in a recursive way using the `Plotkin` and `BQPlotkin` constructions defined in Section 2.3.

`PermutationGroupHadamardCodeZ4Order(`δ`, m)`

`PAutHadamardCodeZ4Order(`δ`, m)`

Given an integer $m \geq 1$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, return the order of the permutation group G of a Hadamard code over $\mathbb{Z}_4$ of length 2^{m-1} and type $2^\gamma 4^\delta$, where $\gamma = m + 1 - 2\delta$. The group G contains all permutations of the coordinates which preserve the code.

`PermutationGroupExtendedPerfectCodeZ4(`δ`, m)`

`PAutExtendedPerfectCodeZ4(`δ`, m)`

Given an integer $m \geq 2$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, return the permutation group G of an extended perfect code over $\mathbb{Z}_4$ of length 2^{m-1}, such that its dual code is of type $2^\gamma 4^\delta$, where $\gamma = m+1-2\delta$. The group G contains all permutations of the coordinates which preserve the code. Thus only permutation of coordinates is allowed, and the degree of G is always 2^{m-1}. Moreover, the generator matrix with $\gamma+\delta$ rows used to generate the code is returned. This matrix is constructed in a recursive way using the `Plotkin` and `BQPlotkin` constructions defined in Section 2.3.

`PermutationGroupExtendedPerfectCodeZ4Order(`δ`, m)`

`PAutExtendedPerfectCodeZ4Order(`δ`, m)`

Given an integer $m \geq 2$ and an integer δ such that $1 \leq \delta \leq \lfloor (m+1)/2 \rfloor$, return the order of the permutation group G of an extended perfect code over $\mathbb{Z}_4$ of length 2^{m-1}, such that its dual code is of type $2^\gamma 4^\delta$, where $\gamma = m+1-2\delta$. The group G contains all permutations of the coordinates which preserve the code.

Example H2E17

```
> C := HadamardCodeZ4(2,4);
> PAut := PAutHadamardCodeZ4(2,4);
> PAut;
Permutation group PAut acting on a set of cardinality 8
    (1, 2)(3, 4)(5, 6)(7, 8)
    (2, 4)(6, 8)
    (5, 7)(6, 8)
    (1, 5)(3, 7)
> {p : p in Sym(8) | C^p eq C} eq Set(PAut);
true
> #PAut eq CardinalPAutHadamardCodeZ4(2,4);
true
> d := 2; m := 4; g := m+1-2*d;
> CardinalPAutHadamardCodeZ4(d, m) eq
  #GL(d-1,Integers(4))*#GL(g,Integers(2))*2^g*4^((g+1)*(d-1));
true

> d := 4; m := 8; g := m+1-2*d;
> CardinalPAutHadamardCodeZ4(d, m) eq
  #GL(d-1,Integers(4))*#GL(g,Integers(2))*2^g*4^((g+1)*(d-1));
true

> PAutHadamardCodeZ4(2,4) eq PAutExtendedPerfectCodeZ4(2,4);
true
```

`PermutationGroup(C)`

The permutation group G of the linear code C of length n over the ring R, where G is the group of all permutation-action permutations which preserve the code. Thus only permutation of coordinates is allowed, and the degree of G is always n.

`PermutationGroupGrayMapImage(C)`

Given a code C over $\mathbb{Z}_4$ of length n, return the permutation group G_{bin} of $C_{bin} = \Phi(C)$, where G_{bin} is the group of all permutation-action permutations which preserve the binary code C_{bin} of length $2n$ and Φ is the Gray map. Thus only permutation of coordinates is allowed, and the degree of G_{bin} is always $2n$.

Example H2E18

```
> C := HadamardCodeZ4(2,4);
> PAutC := PermutationGroup(C);
> PAutC eq PAutHadamardCodeZ4(2,4);
true
```

```
> #PAutC eq PAutHadamardCodeZ4Order(2,4);
true
> {p : p in Sym(8) | C^p eq C} eq Set(PAutC);
true

> Cbin := GrayMapImage(C);
> PAutCbin := PermutationGroupGrayMapImage(C);
> {p : p in PAutCbin | Set(Cbin)^p eq Set(Cbin)} eq Set(PAutCbin);
true
```

Chapter 3

Linear Codes over Finite Fields

3.1 Permutation Decoding

This section supplies functions for decoding vectors from the ambient space of a linear code C over a finite field by using the permutation decoding method. The reader is referred to [2, 11, 16] for more information.

`IsPermutationDecodeSet(C, I, S, s)`

Given

- an $[n, k]$ linear code C over a finite field K;
- a sequence $I \subseteq \{1, \ldots, n\}$;
- a sequence S of elements in the group of monomial matrices of degree n over K, or a sequence of elements in the symmetric group $\mathrm{Sym}(n)$ acting on the set $\{1, \ldots, n\}$;
- and an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;

this intrinsic returns `true` if and only if S is an s-PD-set for C with respect to the information set I.

Depending on the length of the code C, its dimension k, and the integer s, this function could take some time to compute whether S is an s-PD-set for C with respect to I. Specifically, if the function returns `true`, it is necessary to check $\sum_{i=1}^{s} \binom{k}{i} \cdot \binom{n-k}{s-i}$ s-sets.

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 the function returns `false` is also shown, that is, whether I is not an information set, S is not a subset of the monomial automorphism group of C or S is not an s-PD-set. If it is set to level 2, the percentage of the computation process performed is also printed.

```
PermutationDecode(C, I, S, s, u)
```

Given

- an $[n, k]$ linear code C over a finite field K;
- an information set $I \subseteq \{1, \ldots, n\}$ for C as a sequence of coordinate positions;
- a sequence S of elements in the group of monomial matrices of degree n over K, or a sequence of elements in the symmetric group $\mathrm{Sym}(n)$ acting on the set $\{1, \ldots, n\}$. In either case S must be an s-PD-set for C with respect to I;
- an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;
- and a vector u from the ambient space V of C,

the intrinsic attempts to decode u with respect to C. If the decoding algorithm succeeds in computing a vector $u' \in C$ as the decoded version of $u \in V$, then the function returns `true` and the codeword u'. If the decoding algorithm does not succeed in decoding u, then the function returns `false` and the zero vector in V.

The permutation decoding algorithm works by moving all errors in the received vector $u = c + e$, where $c \in C$ and $e \in V$ is the 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 for C, by using an automorphism of C. Note that the function does not check any of the conditions that I is an information set for C, S is an s-PD-set for C with respect to I, or $s \leq t$.

`PermutationDecode(C, I, S, s, Q)`

Given

- an $[n, k]$ linear code C over a finite field K;
- an information set $I \subseteq \{1, \ldots, n\}$ for C as a sequence of coordinate positions;
- a sequence S of elements in the group of monomial matrices of degree n over K, or a sequence of elements in the symmetric group $\mathrm{Sym}(n)$ acting on the set $\{1, \ldots, n\}$. In either case S must be an s-PD-set for C with respect to I;
- an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;
- and a sequence Q of vectors from the ambient space V of C,

the intrinsic attempts to decode the vectors of Q with respect to C. This function is similar to the function `PermutationDecode(C, I, S, s, u)` except that rather than decoding a single vector, it decodes a sequence of vectors and returns a sequence of booleans and a sequence of decoded vectors corresponding to the given sequence. The algorithm used is as for the function `PermutationDecode(C, I, S, s, u)`.

`PDSetSimplexCode(K, m)`

Given a finite field K of cardinality q, and a positive integer m, the intrinsic constructs the $[n = (q^m - 1)/(q - 1), m, q^{m-1}]$ linear simplex code C over K, as `Dual(HammingCode(K, m))`, and then searches for an s-PD-set for C. The function returns an information set I for C together with a subset S of the monomial automorphism group of C such that S is an s-PD-set for C with respect to I, where $s = \lfloor (q^m - 1)/(m(q - 1)) \rfloor - 1$.

The information set I is returned as a sequence of m integers, giving the coordinate positions that correspond to the information set for C. The set S is also returned as a sequence, which contains the $s + 1$ elements in the group of monomial matrices of degree n over K described in [11]. When K is $GF(2)$, the function also returns the elements of S represented as elements in the symmetric group $\mathrm{Sym}(n)$ of permutations on the set $\{1, \ldots, n\}$.

`PDSetHadamardCode(m)`

Given a positive integer m, the intrinsic constructs the $[2^m, m + 1, 2^{m-1}]$ binary linear Hadamard code C, as `Dual(ExtendCode(HammingCode(GF(2), m)))`, and then searches for an s-PD-set for C. The function returns an information set $I \subseteq \{1, \ldots, 2^m\}$ for C together with a subset S of the permutation automorphism group of C such that S is an s-PD-set for C with respect to I, where $s = \lfloor 2^m/(m+1) \rfloor - 1$.

The information set I is returned as a sequence of $m+1$ integers, giving the coordinate positions that correspond to the information set for C. The set S is also returned as a sequence, which contains the $s+1$ elements in the group of permutation matrices of degree 2^m over $GF(2)$ described in [2]. The function also returns the elements of S represented as elements in the symmetric group $\mathrm{Sym}(2^m)$ of permutations on the set $\{1, \ldots, 2^m\}$.

Example H3E1

```
> C := Dual(ExtendCode(HammingCode(GF(2), 5)));
> C;
[32, 6, 16] Linear Code over GF(2)
Generator matrix:
[1 0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 1]
[0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0]
[0 0 1 0 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 1 0 0 0 0 0 1 1 1 1]
[0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0 0]
[0 0 0 0 1 0 0 1 0 1 1 0 0 1 1 1 1 1 0 0 0 1 1 0 1 1 1 0 1 0 1 0]
[0 0 0 0 0 1 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1 1 0 1 1 0 1 0 0 1 1 1]

> I, SMAut, SPAut := PDSetHadamardCode(5);
> I in AllInformationSets(C);
true
> s := #SMAut-1;  s;
4
> [ LinearCode(GeneratorMatrix(C)*SMAut[i]) eq C : i in [1..s+1] ];
[true, true, true, true, true];
> [ LinearCode(GeneratorMatrix(C)^SPAut[i]) eq C : i in [1..s+1] ];
[true, true, true, true, true];
> IsPermutationDecodeSet(C, I, SMAut, s);
true
> IsPermutationDecodeSet(C, I, SPAut, s);
true

> c := C ! [1^^32];
> c in C;
true
> u := c;
> u[1] := c[1] + 1;
> u[2] := c[2] + 1;
> u[4] := c[4] + 1;
> u[32] := c[32] + 1;
> u in C;
false
> isDecoded, uDecoded := PermutationDecode(C, I, SMAut, s, u);
> isDecoded;
true
> uDecoded eq c;
true
> isDecoded, uDecoded := PermutationDecode(C, I, SPAut, s, u);
> isDecoded;
true
> uDecoded eq c;
true
```

Example H3E2

```
> K<a> := GF(4);
> C := Dual(HammingCode(K, 3));
> C;
[21, 3, 16] Linear Code over GF(2^2)
Generator matrix:
[1 0 a^2 a 1 0 a^2 a 1 a^2 0 1 a a 1 0 a^2 1 a a^2 0]
[0 1 1 1 1 0 0 0 0 a^2 a^2 a^2 a^2 a a a a 1 1 1 1]
[0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

> I, SMAut := PDSetSimplexCode(K, 3);
> I in AllInformationSets(C);
true
> s := #SMAut-1;  s;
6
> [ LinearCode(GeneratorMatrix(C)*SMAut[i]) eq C : i in [1..s+1] ];
[true, true, true, true, true, true, true];
> IsPermutationDecodeSet(C, I, SMAut, s);
true

> c := C ! [0,1,1,1,1,0,0,0,0,a^2,a^2,a^2,a^2,a,a,a,a,1,1,1,1];
> c in C;
true
> u := c;
> u[1] := c[1] + a;
> u[2] := c[2] + a^2;
> u[3] := c[3] + a;
> u[4] := c[4] + a^2;
> u[5] := c[5] + a;
> u[6] := c[6] + a^2;
> u in C;
```

```
false
> isDecoded, uDecoded := PermutationDecode(C, I, SMAut, s, u);
> isDecoded;
true
> uDecoded eq c;
true
```

Appendix A

Updated and new functions of version 2.1

A.1 Updated functions

`SyndromeSpace(C)      v2.0`

Given a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, return the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{n-\delta}$ isomorphic to $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^{n-\gamma-\delta}$ such that the first γ coordinates are of order two, that is, the space of syndrome vectors for C. The function also returns the $(2n-2\delta-\gamma)$-dimensional binary vector space, which is the space of syndrome vectors for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map. Note that these spaces are computed by using the function `InformationSpace(C)` applied to the dual code of C, given by function `Dual(C)`.

`SyndromeSpace(C)      v2.1`

Given a code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, return the $\mathbb{Z}_4$-submodule of $\mathbb{Z}_4^{n-\delta}$ isomorphic to $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^{n-\gamma-\delta}$ such that the first γ coordinates are of order two, that is, the space of syndrome vectors for C. The function also returns the $(2n-2\delta-\gamma)$-dimensional binary vector space, which is the space of syndrome vectors for the corresponding binary code $C_{bin} = \Phi(C)$, where Φ is the Gray map. Note that these spaces are computed by using the function `InformationSpace(C)` applied to the dual code of C, given by function `DualZ4(C)`.

`IsPermutationDecodeSet(C, I, S, s)     v2.0`

Given

- an $[n, k]$ linear code C over a finite field K;
- a sequence $I \subseteq \{1, \ldots, n\}$;
- a sequence S of elements in the group of monomial matrices of degree n over K, OR if C is a binary code, a sequence of elements in the symmetric group $\mathrm{Sym}(n)$ acting on the set $\{1, \ldots, n\}$;
- and an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;

this intrinsic returns **`true`** if and only if S is an s-PD-set for C with respect to the information set I.

`IsPermutationDecodeSet(C, I, S, s)     v2.1`

Given

- an $[n, k]$ linear code C over a finite field K;
- a sequence $I \subseteq \{1, \ldots, n\}$;
- a sequence S of elements in the group of monomial matrices of degree n over K, or ~~if C is a binary code,~~ a sequence of elements in the symmetric group $\mathrm{Sym}(n)$ acting on the set $\{1, \ldots, n\}$;
- and an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;

this intrinsic returns **`true`** if and only if S is an s-PD-set for C with respect to the information set I.

`PermutationDecode(C, I, S, s, u)      v2.0`

Given

- an $[n, k]$ linear code C over a finite field K;
- an information set $I \subseteq \{1, \ldots, n\}$ for C as a sequence of coordinate positions;
- a sequence S of elements in the group of monomial matrices of degree n over K, OR if C is a binary code, a sequence of elements in the symmetric group Sym(n) acting on the set $\{1, \ldots, n\}$. In either case S must be an s-PD-set for C with respect to I;
- an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;
- and a vector u from the ambient space V of C,

the intrinsic attempts to decode...

`PermutationDecode(C, I, S, s, u)      v2.1`

Given

- an $[n, k]$ linear code C over a finite field K;
- an information set $I \subseteq \{1, \ldots, n\}$ for C as a sequence of coordinate positions;
- a sequence S of elements in the group of monomial matrices of degree n over K, or ~~if C is a binary code,~~ a sequence of elements in the symmetric group Sym(n) acting on the set $\{1, \ldots, n\}$. In either case S must be an s-PD-set for C with respect to I;
- an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;
- and a vector u from the ambient space V of C,

the intrinsic attempts to decode...

`PermutationDecode(C, I, S, s, Q)      v2.0`

Given

- an $[n, k]$ linear code C over a finite field K;
- an information set $I \subseteq \{1, \ldots, n\}$ for C as a sequence of coordinate positions;
- a sequence S of elements in the group of monomial matrices of degree n over K, OR if C is a binary code, a sequence of elements in the symmetric group $\mathrm{Sym}(n)$ acting on the set $\{1, \ldots, n\}$. In either case S must be an s-PD-set for C with respect to I;
- an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;
- and a sequence Q of vectors from the ambient space V of C,

the intrinsic attempts to decode ...

`PermutationDecode(C, I, S, s, Q)      v2.1`

Given

- an $[n, k]$ linear code C over a finite field K;
- an information set $I \subseteq \{1, \ldots, n\}$ for C as a sequence of coordinate positions;
- a sequence S of elements in the group of monomial matrices of degree n over K, or ~~if C is a binary code,~~ a sequence of elements in the symmetric group $\mathrm{Sym}(n)$ acting on the set $\{1, \ldots, n\}$. In either case S must be an s-PD-set for C with respect to I;
- an integer $s \in \{1, \ldots, t\}$, where t is the error-correcting capability of C;
- and a sequence Q of vectors from the ambient space V of C,

the intrinsic attempts to decode...

A.2 New functions

`StandardFormDual(C)`

Given a code C over $\mathbb{Z}_4$ of length n, 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. Since S is generated by a matrix of the form

$$\begin{pmatrix} I_{k_1} & A & B \\ 0 & 2I_{k_2} & 2C \end{pmatrix},$$

the dual of S is generated by the matrix

$$\begin{pmatrix} -(AC+B)^t & C^t & I_{n-k_1-k_2} \\ 2A^t & 2I_{k_2} & 0 \end{pmatrix},$$

where I_{k_1} and I_{k_2} are the $k_1 \times k_1$ and $k_2 \times k_2$ identity matrices, respectively, A and C are $\mathbb{Z}_2$-matrices, and B is a $\mathbb{Z}_4$-matrix.

`MinRowsParityCheckMatrix(C)`

A parity check matrix for the code C over $\mathbb{Z}_4$ of length n and type $2^\gamma 4^\delta$, with the minimum number of rows, that is, with γ rows of order two and $n-\gamma-\delta$ rows of order four. This function should be faster for most codes over $\mathbb{Z}_4$ 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 `MinRowsGeneratorMatrix(DualZ4(C))`.

`DualZ4(C)`

The dual D of the code C over $\mathbb{Z}_4$ of length n. The dual consists of all codewords in the $\mathbb{Z}_4$-space $V = \mathbb{Z}_4^n$ which are orthogonal to all codewords of C. This function should be faster for most codes over $\mathbb{Z}_4$ than the general function `Dual(C)` for codes over finite rings.

Example HAE3

```
> C := HadamardCodeZ4(3, 11);
> G, gamma, delta := MinRowsGeneratorMatrix(C);
> Nrows(G) eq gamma + delta;
true
> deltaH := Length(C) - gamma - delta;
> H1 := MinRowsParityCheckMatrix(C);
> Nrows(H1) eq gamma + deltaH;
true
> H2 := MinRowsGeneratorMatrix(DualZ4(C));
> Nrows(H2) eq gamma + deltaH;
```

```
true

> time D := Dual(C);
Time: 24.660
> time D4 := DualZ4(C);
Time: 0.340
> D eq D4, D4 eq LinearCode(H1), D4 eq LinearCode(H2);
true true true

> DualS, f := StandardFormDual(C);
> DualS eq LinearCode(Matrix([f(H1[i]) : i in [1..Nrows(H1)]]));
true
```

`PermutationGroup(C)`

The permutation group G of the linear code C of length n over the ring R, where G is the group of all permutation-action permutations which preserve the code. Thus only permutation of coordinates is allowed, and the degree of G is always n.

`PermutationGroupGrayMapImage(C)`

Given a code C over $\mathbb{Z}_4$ of length n, return the permutation group G_{bin} of $C_{bin} = \Phi(C)$, where G_{bin} is the group of all permutation-action permutations which preserve the binary code C_{bin} of length $2n$ and Φ is the Gray map. Thus only permutation of coordinates is allowed, and the degree of G_{bin} is always $2n$.

Example HAE4

```
> C := HadamardCodeZ4(2,4);
> PAutC := PermutationGroup(C);
> PAutC eq PAutHadamardCodeZ4(2,4);
true
> #PAutC eq PAutHadamardCodeZ4Order(2,4);
true
> {p : p in Sym(8) | C^p eq C} eq Set(PAutC);
true

> Cbin := GrayMapImage(C);
> PAutCbin := PermutationGroupGrayMapImage(C);
> {p : p in PAutCbin | Set(Cbin)^p eq Set(Cbin)} eq Set(PAutCbin);
true
```

A.3 Undocumented functions

`StandardFormInfo(C)`

Internal Magma function.
Signatures: `(C::CodeLin)` $\rightarrow$ `Mtrx, CodeLin, Mtrx, GrpPermElt`
This function is called from `StandardForm(C)` function.

`Z4Type(C)`

Magma function.
Signatures: `(C::CodeLinRng)` $\rightarrow$ `RngIntElt, RngIntElt`
Return the "type" of the code C over $\mathbb{Z}_4$. Which is m_1, m_2 such that $|C| = 4^{m_1} \cdot 2^{m_2}$.

`StandardFormInfoInverted(C)`

Given a code C over $\mathbb{Z}_4$ of length n, return a generator matrix of a permutation-equivalent code S in standard form, together with the permutation of coordinates used to define the corresponding isomorphism from C onto S.
This function is called from `StandardFormInverted(C)` function. The generator matrix is of the form

$$\begin{pmatrix} 2C' & 2I_{k_2} & 0 \\ B' & A' & I_{k_1} \end{pmatrix},$$

where I_{k_1} and I_{k_2} are the $k_1 \times k_1$ and $k_2 \times k_2$ identity matrices, respectively, A' and C' are $\mathbb{Z}_2$-matrices, and B' is a $\mathbb{Z}_4$-matrix, so it has the rows and columns in inverted order compared to the matrix given by the function `StandardFormInfo(C)`.

`StandardFormInverted(C)`

Given a code C over $\mathbb{Z}_4$ of length n, return a permutation-equivalent code S in standard form, together with the corresponding isomorphism from C onto S. The code S is generated by a matrix in standard form having the rows and columns in inverted order compared to the one given by the function `StandardFormInfo(C)`, that is, the matrix

$$\begin{pmatrix} 2C' & 2I_{k_2} & 0 \\ B' & A' & I_{k_1} \end{pmatrix},$$

instead of the matrix

$$\begin{pmatrix} I_{k_1} & A & B \\ 0 & 2I_{k_2} & 2C \end{pmatrix},$$

where I_{k_1} and I_{k_2} are the $k_1 \times k_1$ and $k_2 \times k_2$ identity matrices, respectively, A, A', C and C' are $\mathbb{Z}_2$-matrices, and B and B' are $\mathbb{Z}_4$-matrices.

`StandardFormDualInfo(C)`

Given a code C over $\mathbb{Z}_4$ of length n, return a generator matrix of the dual of a permutation-equivalent code S in standard form, together with the permutation of coordinates used to define the corresponding isomorphism from the dual of C onto the dual of S.
This function is called from `StandardFormDual(C)` function. The matrix is of the form

$$\begin{pmatrix} -(AC+B)^t & C^t & I_{n-k_1-k_2} \\ 2A^t & 2I_{k_2} & 0 \end{pmatrix},$$

where I_{k_1} and I_{k_2} are the $k_1 \times k_1$ and $k_2 \times k_2$ identity matrices, respectively, A and C are $\mathbb{Z}_2$-matrices, and B is a $\mathbb{Z}_4$-matrix.

Bibliography

[1] N.S. Babu and K.H. Zimmermann, "Decoding of linear codes over Galois rings," *IEEE Trans. on Information Theory*, vol. 47, no. 4, pp. 1599–1603, 2001.

[2] R. Barrolleta and M. Villanueva, "Partial permutation decoding for binary linear and $\mathbb{Z}_4$-linear Hadamard codes," *Designs, Codes and Cryptography*, 2017. DOI 10.1007/s10623-017-0342-8

[3] R. Barrolleta and M. Villanueva, "PD-sets for $\mathbb{Z}_4$-linear codes: Hadamard and Kerdock codes," in *Proceedings of the IEEE International Symposium on Information Theory*, pp. 1317-1321, 2016.

[4] J. J. Bernal, J. Borges, C. Fernández-Córboda, and M. Villanueva, "Permutation decoding of $\mathbb{Z}_2\mathbb{Z}_4$-linear codes," *Des. Codes and Cryptogr.*, vol. 76, no. 2, pp. 269–277, 2015.

[5] J. Borges, C. Fernández, J. Pujol, J. Rifà and M. Villanueva, "On $\mathbb{Z}_2\mathbb{Z}_4$-linear codes and duality," *V Jornadas de Matemática Discreta y Algorítmica*, Soria (Spain), July 11-14, pp. 171–177, 2006.

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

[7] J. J. Cannon and W. Bosma (Eds.) *Handbook of* MAGMA *Functions*, Edition 2.13, 4350 pages, 2006.

[8] P. Delsarte, "An algebraic approach to the association schemes of coding theory," *Philips Research Rep. Suppl.*, vol. 10, 1973.

[9] 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.

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

[11] W. Fish, J. Key, and E. Mwambene, "Partial permutation decoding for simplex codes," *Advances in Mathematics of Communications*, vol. 6, no. 4, pp. 505–516, 2012.

[12] M. Greferath and U. Velbinger, "Efficient decoding of $\mathbb{Z}_{p^k}$-linear codes," *IEEE Trans. on Information Theory*, vol. 44, pp. 1288–1291, 1998.

[13] 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, pp. 301–319, 1994.

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

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

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

[17] J. Pernas, J. Pujol, and M. Villanueva, "On the Permutation Automorphism Group of Quaternary Linear Hadamard Codes," *Proceedings of 3rd International Castle Meeting on Coding Theory and Applications*, Cardona (Spain), September 11-15, pp. 213–218, 2011.

[18] 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.

[19] J. Pujol, J. Rifà, F. I. Solov'eva, "Quaternary Plotkin constructions and Quaternary Reed-Muller codes," *Lecture Notes in Computer Science* no. 4851, pp. 148–157, 2007.

[20] 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.

[21] 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.

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

[23] M. Villanueva, F. Zeng, 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.

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