Q-ARY CODES

MAGMA Packages

by

Jaume Pujol and Mercè Villanueva

Combinatorics, Coding and Security Group (CCSG)

Autonomous University of Barcelona

Version 1.0

Barcelona
July, 2017

Contents

Preface

The research group CCSG (Combinatorics, Coding and Security Group) is one of the research groups in the dEIC (Department of Information and Communications Engineering) at the UAB (Universitat Autònoma de Barcelona) in Spain.

From 1987 the team CCSG has been uninterruptedly working in several projects and research activities on Information Theory, Communications, Coding Theory, Source Coding, Teledetection, Cryptography, Electronic Voting, e-Auctions, Mobile Agents, etc.

The more important know-how of CCSG is about algorithms for forward error correction (FEC), such as Golay codes, Hamming product codes, Reed-Solomon codes, Preparata and Preparata-like codes, (extended) nonlinear 1-perfect codes, $\mathbb{Z}_4$-linear codes, $\mathbb{Z}_2\mathbb{Z}_4$-linear codes, etc.; computations of the rank and the dimension of the kernel for nonlinear codes as binary 1-perfect codes, q-ary 1-perfect codes, Preparata codes, Hadamard codes, Kerdock codes, quaternary Reed-Muller codes, etc.; the existence and structural properties for 1-perfect codes, uniformly packed codes, completely regular codes, completely transitive codes, etc.

- Chapter 1 has been developed by Jaume Pujol and Mercè Villanueva with the collaboration of Joan Cuadros, Adrià Figuerola, Miriam Gutiérrez, Víctor Ovalle, Erik Vicent, Laura Vidal, and Fanxuan Zeng.

Chapter 1

Qary Codes

1.1 Introduction

MAGMA currently supports the basic facilities for codes over finite fields and codes over integer residue rings and galois rings, all that are linear codes [6, Chapters 127-130]. This chapter describes functions which are applicable to codes over finite fields not necessarily linear.

Let $\mathbb{F}_q^n$ be the n-dimensional vector space over the finite field $\mathbb{F}_q = GF(q)$. An (n, M, d) *q-ary code* C is a subset of $\mathbb{F}_q^n$ with cardinality M and minimum Hamming distance d. Two q-ary codes C_1 and C_2 of length n are said to be *equivalent* if there is a vector $a \in \mathbb{F}_q^n$, a monomial matrix $\mathcal{M}$ and an automorphism Γ of the field $\mathbb{F}_q$ such that $C_2 = \{\mathcal{M}\Gamma(c) + a \mid c \in C_1\}$. Note that two equivalent q-ary codes have the same minimum Hamming distance. Moreover, if C is linear, the zero word belongs to C, but if C is nonlinear, the zero word does not need to belong to C. In any case, for every $u \in C$, we can consider the q-ary code, called *zero-translation code*, $C^u = C - u = \{c - u \mid c \in C\}$. Note that C^u always contains the zero word and is equivalent to C. Moreover, if C is linear, then $C^u = C$ for all $u \in C$. In this chapter, the term "code" will refer to a q-ary code not necessarily linear, which may not contain the zero word, unless otherwise specified.

Given a q-ary code C, we can define two q-ary linear codes related to C: the *linear span* of C, which is $\langle C \rangle$; and the *kernel* of C, defined as $K(C) = \{x \in \mathbb{F}_q^n \mid \lambda x + C = C,\ \forall \lambda \in \mathbb{F}_q\}$ [1, 18]. Note that $K(C)$ is a linear subspace of $\mathbb{F}_q^n$. Moreover, if the zero word, denoted by $\mathbf{0}$, is in C, then $K(C)$ is also a subset of C, so $K(C) \subseteq C \subseteq \langle C \rangle$. Otherwise, if $\mathbf{0} \notin C$, then it is easy to see that $K(C) \cap C = \emptyset$. The dimension of the linear span of C, called *rank*, is denoted by $r = \dim(\langle C \rangle)$; and the dimension of the kernel of C is denoted by $k = \dim(K(C))$. These two parameters can be used to

distinguish between nonequivalent q-ary codes, since equivalent ones have the same parameters r and k. Therefore, they provide a sufficient condition which is not necessary, since two nonequivalent q-ary codes could have the same parameters r and k.

Let C be a q-ary code of length n and cardinality M with kernel $K(C)$ of dimension k. Then, C can be written as the union of cosets of $K(C)$:

$$C = \bigcup_{i=0}^{t} \Big(K(C) + c_i \Big),$$

where $t + 1 = M/q^k$ [1, 18]. The elements $c_0, c_1, \ldots, c_t$ are called *coset representatives* of C. If $\mathbf{0} \in C$, since $K(C) \subseteq C$, then exactly one of the coset representatives has to be in the kernel. In this case, we always consider that $\mathbf{0} \in K(C)$ is a coset representative and $c_0 = \mathbf{0}$.

The *parity check system* of the q-ary code C is an $(n-k) \times (n+t+1)$ matrix over $\mathbb{F}_q$, $(H|S)$, where H is a parity check matrix of $K(C)$ and $S = (H \cdot c_0 \; H \cdot c_1 \; \ldots \; H \cdot c_t)$. The *super dual* of the q-ary code C is the linear code generated by the parity check system $(H|S)$. Note that if C is linear, then $c_0 = \mathbf{0}$, S is a zero matrix of size $(n-k) \times 1$, and the super dual coincides with the dual code of C after removing the last zero coordinate. From these definitions, we can establish the following properties [11, 12]:

(i) Let $\mathrm{col}(S)$ denote the set of columns of S. Then, $c \in C$ if and only if $H \cdot c \in \mathrm{col}(S)$.

(ii) Let $r = \dim(\langle C \rangle)$ and $k = \dim(K(C))$. Then, $r = n - \dim(H) + \dim(S)$ and $k = n - \dim(H)$.

(iii) The super dual of a q-ary code C is unique, up to a permutation of the columns of S.

(iv) Let π be any permutation of the set of coordinate positions. Then, $(\pi(H)|S)$ is a parity check system for the q-ary code $\pi(C)$.

(v) Let $C^{c_i} = C - c_i$, for $i \in \{0, 1, \ldots, t\}$. Then, $(H|S^{c_i})$ is a parity check system for C^{c_i}, where S^{c_i} is obtained from S by replacing the column vector $H \cdot c_i$ with $-H \cdot c_i$ and adding $-H \cdot c_i$ to all other columns of S.

Let C be a q-ary code with kernel $K(C)$. For every $u \in C$, we can consider the zero-translation code $C^u = C - u$. Note that $C^u = C$ for all $u \in K(C)$. Moreover, it is also easy to see that $K(C) = K(C^u)$ for all $u \in C$. However, note that, if $C^u \neq C^v$, then the coset representatives of C^u are not a suitable set of coset representatives for C^v, where $u \neq v \in C$.

1.2 Construction of Qary Codes

1.2.1 Construction of General Qary Codes

`QaryCode(L)`

Create a q-ary code C determined by the kernel $K(C)$ and a set of coset representatives $c_0, c_1, \ldots, c_t$ of C. If $\mathbf{0} \in C$, then $c_0 = \mathbf{0}$. The q-ary code C is generated by the elements specified by L, where

1. L is a sequence of elements of $V = \mathbb{F}_q^n$,
2. or, L is a subspace of $V = \mathbb{F}_q^n$,
3. or, L is an $m \times n$ matrix A over a finite field $\mathbb{F}_q$,
4. or, L is an $m \times n$ matrix A over a ring $\mathbb{Z}_p$, p prime,
5. or, L is an $m \times n$ matrix A over a ring $\mathbb{Z}_{2^s}$, $s \geq 2$,
6. or, L is a linear code over a finite field $\mathbb{F}_q$,
7. or, L is a linear code over a ring $\mathbb{Z}_p$, p prime,
8. or, L is a linear code over a ring $\mathbb{Z}_{2^s}$, $s \geq 2$.

The verbose flag `KernelFlag` is set to level 0 by default. If it is set to level 1, the total time used to compute the kernel and coset representatives is shown. If it is set to level 2, the percentage of the computation process performed is also printed.

Note that, depending on the cardinality M of the q-ary code C, this function could take some time to compute $K(C)$ and $c_0, c_1, \ldots, c_t$, in order to return the q-ary code. The following table shows the time (in seconds) to compute the kernel and coset representatives for several q-ary codes, using this function (QC column) and the brute force method (BF column). All these computations have been performed in MAGMA version V2.22-1, running on a server with an Intel Xeon processor (clock speed 2.40GHz) and 32GB of memory.

n	$M = q^k(t+1)$	Base field	k	t	Time BF(s.)	Time QC(s.)
16	64	$GF(4)$	2	3	0.02	0.01
16	2048	$GF(2)$	9	3	2.62	0.04
13	59049	$GF(3)$	9	2	7381.87	1.86
13	118098	$GF(3)$	10	1	47922.29	4.25
20	163840	$GF(8)$	5	4	128656.85	7.92

If L is an $m \times n$ matrix A over the ring $\mathbb{Z}_{2^s}$, with $s \geq 2$, then the q-ary code has as codewords the image under the generalized Gray map of the rows of L [5, 10]. Similarly, if L is a linear code over the ring $\mathbb{Z}_{2^s}$, with $s \geq 2$, then the q-ary code has as codewords the image under the generalized Gray map of the codewords of L. In both cases, $q = 2$, and the binary code is not necessarily linear. Note that the coset representatives $c_0, c_1, \ldots, c_t$ have $c_0 = \mathbf{0}$, since the binary code always contains the zero word. However, note that if $s = 2$, then the function `KernelCosetRepresentatives` for codes over $\mathbb{Z}_4$ [6, Chapter 162] returns the coset representatives $c_1, \ldots, c_t$ without including the zero word.

If L is an $m \times n$ matrix A over $\mathbb{F}_q$ or $\mathbb{Z}_p$, then the q-ary code has as codewords the rows of L.

This constructor defines a new user-defined type, `CodeFld`, and appends seven attributes to this new category:

- `BaseField`: The base finite field $\mathbb{F}_q$ of the q-ary code C.
- `Kernel`: The kernel $K(C)$ of the code C as a linear code over $\mathbb{F}_q$.
- `CosetRepresentatives`: A sequence of coset representatives $[c_0, c_1, \ldots, c_t]$ of the q-ary code C.
- `ParityCheckSystem`: An $(n-k) \times (n+t+1)$ matrix over $\mathbb{F}_q$ which represents the parity check system of the q-ary code C.
- `Length`: The length n of the q-ary code C.
- `IsLinear`: It is `true` if and only if C is a linear code over $\mathbb{F}_q$.
- `MinimumDistance`: The minimum (Hamming) distance d of C.

If L is a subspace of $V = \mathbb{F}_q^n$ or a linear code over $\mathbb{F}_q$ or $\mathbb{Z}_p$, this function returns the q-ary code, where the `BaseField` attribute is $\mathbb{F}_q$ or $\mathbb{F}_p$, the `Kernel` attribute is the corresponding linear code L defined by the function `LinearCode`, the `CosetRepresentatives` attribute is the sequence containing only the zero word, the `ParityCheckSystem` attribute is the $(n-k) \times (n+1)$ matrix having the generator matrix of the dual code of L in the first n columns and zeroes in the last column, the `Length` attribute is set to n, the `IsLinear` attribute is `true`, and the `MinimumDistance` attribute is computed by using the function `MinimumDistance` for linear codes. Note that, in this case, the type `CodeLinFld` for linear codes over $\mathbb{F}_q$ and the new type `CodeFld` are not compatible.

Example H1E1

We define a q-ary code by giving a sequence of elements of a vector space $V = \mathbb{F}_q^n$, or a matrix over $\mathbb{F}_q$. Note that the function `QaryCode` returns a q-ary code C determined by the kernel and a set of coset representatives of C.

```
> V  := VectorSpace(GF(2), 4);
> L := [V!0, V![1,0,0,0], V![0,1,1,1],
        V![1,0,1,0], V![0,1,0,1], V![1,1,1,1]];
> C1 := QaryCode(L);
> C1;
(4, 6) code over a finite field of size 2
with a kernel of dimension 1 and 3 coset representatives
> C1:Maximal;
(4, 6) code over a finite field of size 2
with a kernel of dimension 1 and 3 coset representatives
Kernel generator matrix:
[1 1 1 1]
Coset representatives sequence:
[
(0 0 0 0),
(1 0 0 0),
(1 0 1 0)
]
> C1‘Kernel;
[4, 1, 4] Cyclic Linear Code over GF(2)
Generator matrix:
[1 1 1 1]
> C1‘CosetRepresentatives;
[
    (0 0 0 0),
    (1 0 0 0),
    (1 0 1 0)
]
> C1‘IsLinear;
false

> A := Matrix(L);
> C2 := QaryCode(A);
> C2;
(4, 6) code over a finite field of size 2
with a kernel of dimension 1 and 3 coset representatives
> C1 eq C2;
true

> C3 := QaryCode([x + V![0,1,0,0] : x in L]);
> C3;
(4, 6) code over a finite field of size 2
with a kernel of dimension 1 and 3 coset representatives
> C3‘CosetRepresentatives;
```

```
[
    (0 1 0 0),
    (1 1 0 0),
    (1 1 1 0)
]
> C1 eq C3;
false
> C1`Kernel eq C3`Kernel;
true
```

Example H1E2

From a linear code L over $\mathbb{Z}_4$, we can define a binary code (that is, a q-ary code with $q = 2$) having as codewords the Gray map image of the codewords of L. In this case, note that the functions `KernelZ2CodeZ4` and `KernelCosetRepresentatives` for codes over $\mathbb{Z}_4$ also return the kernel and a set of coset representatives without the zero word, respectively.

```
> L := HadamardCodeZ4(3, 5);
> L;
((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]
> C4 := QaryCode(L);
> C4;
(32, 64) code over a finite field of size 2
with a kernel of dimension 4 and 4 coset representatives
> C4`Kernel;
[32, 4, 16] Quasicyclic of degree 4 Linear Code over GF(2)
Generator matrix:
[1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0]
[0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1]
[0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1]
[0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]
> C4`CosetRepresentatives;
[
    (0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0),
    (0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 0),
    (0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0),
    (0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0)
]
> _, kernelZ2 := KernelZ2CodeZ4(L);
> C4`Kernel eq kernelZ2;
true
> _, cosetRepZ2 := KernelCosetRepresentatives(L);
> cosetRepZ2;
[
    (0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 0),
```

```
    (0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0),
    (0 1 0 0 1 0 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 0 0 1 0)
]
```

Example H1E3

A linear code C over $\mathbb{F}_q$ can also be generated as a q-ary code by using this constructor. In this cas, note that the function `QaryCode` returns a `CodeFld` type code and not a `CodeLinFld` type code.

```
> V := VectorSpace(GF(2), 4);
> C := LinearCode(sub<V|[[0,0,1,1],[1,0,1,1]]>);
> D := QaryCode(sub<V|[[0,0,1,1],[1,0,1,1]]>);
> D;
[4, 2, 1] Linear Code over GF(2)
Generator matrix:
[0 1 0 0]
[0 0 1 1]

> C eq D;

>> C eq D;
     ^
Runtime error in 'eq': Bad argument types
Argument types given: CodeLinFld, CodeFld

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

`QaryCode(L, K)`

Create a q-ary code C determined by the kernel $K(C)$ and a set of coset representatives $c_0, c_1, \ldots, c_t$ of C. The q-ary code C is generated from a sequence L of elements of $V = \mathbb{F}_q^n$, and a partial kernel K. A partial kernel K is a subspace of $K(C)$, that is, a subspace of V such that $\lambda x + L \subseteq L$ for all $\lambda \in \mathbb{F}_q$ and $x \in K$.

The partial kernel K must be given as a linear code over $\mathbb{F}_q$. Moreover, if $\mathbf{0}$ belongs to L, then K must be a subset of L; otherwise, K must be a subset of $L^u = L - u$ for all $u \in L$. Finally, K must always be a subcode of $K(C)$. These last two conditions are not checked by the function.

The verbose flag `KernelFlag` is set to level 0 by default. If it is set to level 1, the total time used to compute the kernel and coset representatives is shown. If it is set to level 2, the percentage of the computation process performed is also printed.

`QaryCode(K, Rep)`

Create a q-ary code C determined by the kernel $K(C)$ and a set of coset representatives $c_0, c_1, \ldots, c_t$ of C. The q-ary code C is generated from a partial kernel K, and a sequence with the corresponding partial coset representatives Rep of C, so such that the codeword of C are the elements of $L = \bigcup_{c \in Rep}(K + c)$. A partial kernel K is a subspace of $K(C)$, that is, a subspace of V such that $\lambda x + L \subseteq L$ for all $\lambda \in \mathbb{F}_q$ and $x \in K$.

The partial kernel K must be given as a linear code over $\mathbb{F}_q$. If more than one element in Rep belongs to K, only one of them is considered. Moreover, if Rep contains different coset representatives for the same coset of K, only one of them for each coset is considered. Finally, note that if Rep contains an element from K, then $\mathbf{0} \in C$; otherwise, $\mathbf{0} \notin C$.

The verbose flag `KernelFlag` is set to level 0 by default. If it is set to level 1, the total time used to compute the kernel and coset representatives is shown. If it is set to level 2, the percentage of the computation process performed is also printed.

Example H1E4

We can define a q-ary code directly by giving the kernel and a sequence with the corresponding coset representatives. Next, we show the construction of two q-ary codes, one without containing the zero word and another one containing it.

```
> F4<w> := GF(4);
> V := VectorSpace(F4, 5);
> K := LinearCode<F4, 5 | [[1,0,w,1,0], [0,1,w^2,0,1]]>;
> Rep1 := [V![w,w^2,0,w,0], V![1,w^2,1,0,w]];
> Rep2 := [V!0, V![w,w^2,0,w,0], V![1,w^2,1,0,w]];

> C1 := QaryCode(K, Rep1);
> C1;
(5, 32) code over a finite field of size 4
with a kernel of dimension 2 and 2 coset representatives
> C1‘Kernel eq K;
true
> Set(C1) eq {k + x : k in K, x in Rep1};
true
> Set(C1) eq {k + x : k in K, x in C1‘CosetRepresentatives};
true

> C2 := QaryCode(K, Rep2);
> C2;
(5, 48) code over a finite field of size 4
with a kernel of dimension 2 and 3 coset representatives
> C2‘Kernel eq K;
```

```
true
> Set(C2) eq {k + x : k in K, x in Rep2};
true
> Set(C2) eq {k + x : k in K, x in C2‘CosetRepresentatives};
true
```

We can also define the same q-ary code `C2` by giving a partial kernel, that is, a subspace of the kernel and a sequence with the corresponding partial coset representatives.

```
> partialK := LinearCode<F4, 5 | [[1,0,w,1,0]]>;
> partialRep := [V!0, V![w,w,w^2,w,1], V![w,1,1,w,w], V![w,0,w,w,w^2],
                 V![w,w^2,0,w,0], V![1,w,w,0,w^2], V![1,1,0,0,0],
                 V![1,0,w^2,0,1], V![1,w^2,1,0,w], V![0,1,w^2,0,1],
                 V![0,w,1,0,w], V![0,w^2,w,0,w^2]];
> C3 := QaryCode(partialK, partialRep);
> C3 eq C2;
true
```

Note that we can speed up the construction of the q-ary code `C2` from a sequence of elements of a vector space $V = \mathbb{F}_q^n$, when we know the kernel or just a partial kernel, by using the function `QaryCode(L, K)` instead of the function `QaryCode(L)`.

```
> L := [k + rep : k in C2‘Kernel, rep in C2‘CosetRepresentatives];
> time C4 := QaryCode(L);
Time: 0.020
> time C5 := QaryCode(L, partialK);
Time: 0.000
> (C2 eq C4) and (C2 eq C5);
true
```

Example H1E5

In this example, we can see more clear how, when we know the kernel or just a partial kernel, the function `QaryCode(L, K)` can speed up the construction of a q-ary code from a sequence of elements of a vector space $V = \mathbb{F}_q^n$.

```
> CZ4 := ReedMullerCodeRMZ4(1, 2, 5);
> L := GrayMapImage(CZ4);
> #L;
65536

> time C := QaryCode(L);
Time: 1.890
> B := Basis(KernelCode(C));
> partialKernel1 := LinearCode<GF(2), 32 | B[1..2]>;
> time C1 := QaryCode(L, partialKernel1);
Time: 1.600
> partialKernel2 := LinearCode<GF(2), 32 | B[1..5]>;
> time C2 := QaryCode(L, partialKernel2);
```

```
Time: 1.080
> partialKernel3 := LinearCode<GF(2), 32 | B[1..7]>;
> time C3 := QaryCode(L, partialKernel3);
Time: 0.840
> partialKernel4 := LinearCode<GF(2), 32 | B[1..9]>;
> time C4 := QaryCode(L, partialKernel4);
Time: 0.580
> kernel := KernelCode(C);
> time C5 := QaryCode(L, kernel);
Time: 0.330
> (C1 eq C2) and (C2 eq C3) and (C3 eq C4) and (C4 eq C5) and (C5 eq C);
true
```

1.2.2 Some Trivial Qary Codes

`ZeroQaryCode(F, n)`

Given a finite field $F = \mathbb{F}_q$ and positive integer n, return the $(n, 0, n)$ q-ary code consisting of only the zero word, (where the minimum weight is by convention equal to n).

`RepetitionQaryCode(F, n)`

Given a finite field $F = \mathbb{F}_q$ and positive integer n, return the (n, q, n) q-ary code over F generated by the all-ones word.

`UniverseQaryCode(F, n)`

Given a finite field $F = \mathbb{F}_q$ and positive integer n, return the $(n, q^n, 1)$ q-ary code consisting of all possible codewords.

`RandomQaryCode(F, n, M : parameters)`

`ZeroCodeword` BoolElt *Default*: `false`

Given a finite field $F = \mathbb{F}_q$ and positive integers n and M, such that $0 < M \leq q^n$, the function returns a random q-ary code C of length n and cardinality M over the finite field $\mathbb{F}_q$. The method employed is first to find an integer $k \geq 0$ such that $q^k | M$; then a random linear code K of length n and dimension k over $\mathbb{F}_q$; and finally to successively choose random vectors from $\mathbb{F}_q^n$ until M/q^k different cosets of K have been found.

The parameter `ZeroCodeword` specifies whether the random q-ary code C has to contain the zero codeword or not. The default value is `false`. In this case, the random q-ary code C may not contain the zero codeword. If it is set to `true`, then $\mathbf{0} \in C$.

`RandomQaryCode(F, n, M, k : parameters)`

`ZeroCodeword` BoolElt *Default*: `false`

Given a finite field $F = \mathbb{F}_q$ and positive integers n, M, and k, such that $0 < M \leq q^n$ and $q^k | M$, the function returns a random q-ary code C of length n, cardinality M, and with a kernel of dimension at least k over the finite field $\mathbb{F}_q$. The method employed is first to find a random linear code K of length n and dimension k over $\mathbb{F}_q$; and then to successively choose random vectors from $\mathbb{F}_q^n$ until M/q^k different cosets of K have been found.

The parameter `ZeroCodeword` specifies whether the random q-ary code C has to contain the zero codeword or not. The default value is `false`. In this case, the random q-ary code C may not contain the zero codeword. If it is set to `true`, then $\mathbf{0} \in C$.

Example H1E6

A first random q-ary code over $\mathbb{F}_8 = GF(8)$ is constructed having a given length and number of codewords. A second random q-ary code over $\mathbb{F}_8 = GF(8)$ is constructed having also a given minimum dimension of the kernel.

```
> C1 := RandomQaryCode(GF(8), 8, 56);
> (Length(C1) eq 8) and (#C1 eq 56);
true
> C1zero := RandomQaryCode(GF(8), 8, 56 : ZeroCodeword := true);
> (Length(C1zero) eq 8) and (#C1zero eq 56) and (C1zero!0 in C1zero);
true

> C2 := RandomQaryCode(GF(8), 8, 448, 2);
> (Length(C2) eq 8) and (#C2 eq 448);
true
> DimensionOfKernel(C2) ge 2;
true
> C2zero := RandomQaryCode(GF(8), 8, 448, 2 : ZeroCodeword :=true);
> (Length(C2zero) eq 8) and (#C2zero eq 448) and (C2zero!0 in C2zero);
true
> DimensionOfKernel(C2zero) ge 2;
true
```

Over any specific finite field $\mathbb{F}_q$, the q-ary zero code of length n is contained in every q-ary code of length n, and similarly every q-ary code of length n is contained in the q-ary universe code of length n. This is illustrated over $\mathbb{F}_3 = GF(3)$ for codes of length 10 with an arbitrary code of length 10 and cardinality 50.

```
> F := GF(3);
> U := UniverseQaryCode(F, 10);
```

```
> Z := ZeroQaryCode(F, 10);
> R := RandomQaryCode(F, 10, 50);
> (Z subset R) and (R subset U);
true
```

1.3 Invariants of a Qary Code

1.3.1 Basic Numerical Invariants

`Length(C)`

Given a q-ary code C, return the length of C.

`#C`

Given a q-ary code C, return the number of codewords belonging to C.

`Parameters(C)`

Given a q-ary code C, return a sequence with the parameters n, M, and d; where n is the length of C, M the number of codewords, and d the minimum (Hamming) distance.

`InformationRate(C)`

The information rate of the (n, M, d) q-ary code C. This is the ratio $(\log_q M)/n$.

Example H1E7

Given a binary code C, we compute its length, number of codewords, minimum distance, and information rate.

```
> V21 := VectorSpace(GF(2), 21);
> partialK := LinearCode(sub< V21 |
    [1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,0,0],
    [1,1,1,1,1,0,0,0,0,0,1,1,1,1,1,0,0,0,0,1,0],
    [1,1,1,0,0,1,1,0,0,0,1,1,0,0,0,1,1,1,0,0,1]
>);
> partialRep := [
    V21!0,
    V21![0,0,0,1,0,1,1,0,1,1,0,1,0,1,0,1,1,0,0,0,0],
    V21![0,0,1,0,1,0,1,1,1,0,0,0,0,0,1,1,1,1,0,0,0],
    V21![0,1,0,1,0,1,1,0,1,0,1,0,1,0,1,0,0,1,0,0,0],
    V21![0,1,1,0,0,0,0,1,1,1,1,1,1,0,0,1,0,0,0,0,0],
    V21![1,0,0,0,0,1,0,0,0,1,1,0,1,1,1,1,0,0,0,0,1],
    V21![1,0,1,0,0,1,0,1,0,1,0,1,0,0,1,0,1,0,0,1,0],
    V21![1,1,0,1,0,0,1,1,0,0,0,0,0,1,1,0,1,0,0,0,1]
];
```

```
> C := QaryCode(partialK, partialRep);

> Length(C);
21
> #C;
64
> Parameters(C);
[ 21, 64 ]
> MinimumDistance(C);
9
> Parameters(C);
[ 21, 64, 9 ]
> InformationRate(C) eq Log(2, #C)/Length(C);
true
```

1.3.2 The Ambient Space and Alphabet

`AmbientSpace(C)`

The ambient space of the q-ary code C, i.e. the generic vector space V over $\mathbb{F}_q$ in which C is contained.

`Generic(C)`

Given a q-ary code C of length n, return the generic $(n, q^n, 1)$ q-ary code in which C is contained.

`Alphabet(C)`
`Field(C)`

The underlying finite field (or alphabet) $\mathbb{F}_q$ of the q-ary code C.

Example H1E8

Given the same binary code C as in Example H1E7, we compute its ambient space, generic $(n, q^n, 1)$ q-ary code and underlying finite field.

```
> AmbientSpace(C);
Full Vector space of degree 21 over GF(2)
> Generic(C);
[21, 21, 1] Cyclic Linear Code over GF(2)
> Alphabet(C);
Finite field of size 2
```

1.3.3 The Code Space

`SpanCode(C)`

Given a q-ary code C of length n, return the linear span of C, that is, the linear code over $\mathbb{F}_q$ generated by the codewords of C.

Note that if the q-ary code C is linear, the span of C coincides with C.

`DimensionOfSpan(C)`

`Rank(C)`

Given a q-ary code C of length n, return its rank. The rank of a q-ary code C is the dimension of the linear span of C over $\mathbb{F}_q$.

`KernelCode(C)`

Given a q-ary code C of length n, return its kernel as a linear code over $\mathbb{F}_q$. The kernel of a q-ary code C is the set of codewords x such that $\lambda x + C = C$ for all $\lambda \in \mathbb{F}_q$.

Note that if the q-ary code C is linear, the kernel of C coincides with C.

`DimensionOfKernel(C)`

Given a q-ary code C of length n, return the dimension of its kernel. The kernel of a q-ary code C is the set of codewords x such that $\lambda x + C = C$ for all $\lambda \in \mathbb{F}_q$.

`PseudoLinearityRate(C)`

Given a q-ary code C of length n, return the pseudo linearity rate of C. The pseudo linearity rate of a q-ary code C is the ratio $1 - (r - k)/n$, where r and k are the dimension of the linear span (rank) and dimension of the kernel, respectively.

Note that $0 \leq 1 - (r - k)/n \leq 1$. Moreover, if the q-ary code C is linear, then this ratio is equal to 1.

Example H1E9

Given the same binary code C as in Example H1E7, we compute the dimension of its linear span (rank) and dimension of its kernel, as long as its linear span and kernel.

```
> r := DimensionOfSpan(C);
> r;
9
> span := SpanCode(C);
> Dimension(span) eq r;
true

> k := DimensionOfKernel(C);
```

```
> k;
3
> kernel := KernelCode(C);
> Dimension(kernel) eq k;
true

> (kernel subset C) and (C subset span);
true
> k le Ilog(#Field(C), #C);
true
> Ilog(#Field(C), #C) le r;
true
> PseudoLinearityRate(C);
0.714285714285714285714285714285
> PseudoLinearityRate(C) eq 1-(r-k)/Length(C);
true
```

For any linear code, the linear span and kernel coincide with the code itself, so the dimension of the linear span and kernel is the same as the dimension of the code. Moreover, the pseudo linearity rate is always 1.

```
> C2 := RandomLinearCode(GF(9), 10, 4);
> Cq2 := QaryCode(C2);
> (KernelCode(Cq2) eq SpanCode(Cq2)) and (SpanCode(Cq2) eq C2);
true
> (DimensionOfKernel(Cq2) eq Rank(Cq2)) and (Rank(Cq2) eq Dimension(C2));
true
> PseudoLinearityRate(Cq2) eq 1;
true
```

1.3.4 The Dual and Superdual Space

`Dual(C)`

The dual D of the q-ary code C of length n. The dual consists of all elements in the vector space $V = \mathbb{F}_q^n$ which are orthogonal to all codewords of C, so it also coincides with the dual of the linear span of C.

`ParityCheckSystem(C)`

Given a q-ary code C of length n, with kernel $K(C)$ of dimension k and $t+1$ coset representatives $c_0, c_1, \ldots, c_t$, return the $(n-k) \times (n+t+1)$ matrix over $\mathbb{F}_q$, $(H|S)$, where H is a parity check matrix of $K(C)$ and $S = (H \cdot c_0 \ H \cdot c_1 \ \ldots \ H \cdot c_t)$.

Note that if the q-ary code C is linear, then $c_0 = \mathbf{0}$, S is a zero matrix of size $(n-k) \times 1$, and $(H|S)$ is a parity check matrix of $K(C)$ after removing the last zero column.

`SuperDualCode(C)`

Given a q-ary code C of length n, with kernel $K(C)$ of dimension k and $t+1$ coset representatives $c_0, c_1, \ldots, c_t$, return the super dual of C, that is, the linear code over $\mathbb{F}_q$ of length $n+t+1$ generated by the parity check system $(H|S)$.

Note that if the q-ary code C is linear, then the super dual is the dual code of C after removing the last zero coordinate.

Example H1E10

Given a random q-ary code over $\mathbb{F}_5 = GF(5)$, we check that its dual corresponds to the dual of the linear span of the code. We also check that its super dual is the linear code generated by a parity check system $(H|S)$ of the code.

```
> C1 := RandomQaryCode(GF(5), 10, 60);
> Dual(C1) eq Dual(SpanCode(C1));
true
> SuperDualCode(C1) eq LinearCode(ParityCheckSystem(C1));
true
```

For any linear code, the parity check system coincides with a parity check matrix of C after removing the last zero column, and the super dual coincides with the dual code of C after removing the last zero coordinate.

```
> C2 := RandomLinearCode(GF(8), 6, 3);
> Cq2 := QaryCode(C2);
> SuperDualCode(Cq2) eq LinearCode(ParityCheckSystem(Cq2));
true
> RemoveColumn(ParityCheckSystem(Cq2), 7) eq ParityCheckMatrix(C2);
true
> PunctureCode(SuperDualCode(Cq2), 7) eq Dual(C2);
true
```

1.4 Operations on Codewords

1.4.1 Construction of a Codeword

`C ! [a1,...,an]`

Given a q-ary code C which is defined as a subset of the vector space $V = \mathbb{F}_q^n$, and elements $a_1, \ldots, a_n$ belonging to $\mathbb{F}_q$, construct the codeword $(a_1, \ldots, a_n)$ of C. It is checked that the vector $(a_1, \ldots, a_n)$ is an element of C.

`C ! u`

Given a q-ary code C which is defined as a subset of the vector space $V = \mathbb{F}_q^n$, and an element u belonging to V, create the codeword of C corresponding to u. The function will fail if u does not belong to C.

`C ! 0`

The zero word of the q-ary code C.

1.4.2 Codewords

`Random(C)`

Return a random codeword of the q-ary code C.

`CosetRepresentatives(C)`

Given a q-ary code C of length n, return the coset representatives of C as a sequence of vectors of length n over $\mathbb{F}_q$.

`Set(C)`

Given a q-ary code C, return all its codewords as a set. It can be useful to iterate over the codewords of C.

Example H1E11

We first construct a binary code as the Gray map image of a linear code over the ring $\mathbb{Z}_4$. Then, we check that the set of codewords of this binary code coincides with the return of the function `GrayMapImage` applied to the linear code over $\mathbb{Z}_4$.

```
> CZ4 := LinearCode(Matrix(Integers(4), [[1,0,3,0,3], [0,1,0,2,3]]));
> C := QaryCode(CZ4);
> C;
(10, 16) code over a finite field of size 2
with a kernel of dimension 2 and 4 coset representatives

> kernel := KernelCode(C);
> kernel;
[10, 2, 4] Quasicyclic of degree 5 Linear Code over GF(2)
Generator matrix:
[1 1 0 0 1 1 0 0 1 1]
[0 0 1 1 0 0 0 0 1 1]
> cosetRep := CosetRepresentatives(C);
> cosetRep;
[
    (0 0 0 0 0 0 0 0 0 0),
    (0 1 0 0 1 0 0 0 1 0),
    (0 0 0 1 0 0 1 1 1 0),
    (0 1 0 1 1 0 1 1 1 1)
]
```

```
> Set(C) eq Set(GrayMapImage(CZ4));
true
> Set(C) eq {k + c : k in kernel, c in cosetRep};
true
> Random(C) in C;
true
```

1.5 Subcodes of Qary Codes

1.5.1 Derived Codes

`ZeroTranslationCode(C)`

Given a q-ary code C of length n, return a zero-translation code $C^w = C - w$ equivalent to C, together with w. The vector w is always chosen as the first element of the sequence of coset representatives of C. Note that, since $w \in C$, then C^w always contains the zero word.

`TranslationCode(C, w)`

Given a q-ary code C of length n and a vector $w \in V = \mathbb{F}_q^n$, return the q-ary code $C^w = C - w$ equivalent to C. Note that if $w \in K(C)$, then $C^w = C$. Moreover, C^w contains the zero word, so it is a zero-translation code of C, as long as $w \in C$.

Example H1E12

```
> V := VectorSpace(GF(5), 6);
> partialK := LinearCode(sub< V |
    [1,2,3,4,1,1],
    [0,1,0,2,2,0]
>);
> partialRep := [
    V![4,0,2,1,4,1],
    V![3,0,2,4,2,4],
    V![2,1,2,3,4,4]
];
> C := QaryCode(partialK, partialRep);
> C;
(6, 75) code over a finite field of size 5
with a kernel of dimension 2 and 3 coset representatives
> V!0 in C;
false

> Cz := ZeroTranslationCode(C);
```

```
> V!0 in Cz;
true
> MinimumDistance(C) eq MinimumDistance(Cz);
true
> Parameters(C) eq Parameters(Cz);
true
> Cz eq TranslationCode(C, C‘CosetRepresentatives[1]);
true

> C eq TranslationCode(C, Random(C‘Kernel));
true
> Parameters(C) eq Parameters(TranslationCode(C, Random(V)));
true
> KernelCode(C) eq KernelCode(TranslationCode(C, Random(V)));
true
> V!0 in TranslationCode(C, Random(C));
true
> w := Random(Set(V) diff Set(C));
> w in C;
false
> V!0 in TranslationCode(C, w);
false
```

1.5.2 Membership and Equality

`u in C`

Return `true` if and only if the vector u of $V = \mathbb{F}_q^n$ belongs to the q-ary code C of length n.

`u notin C`

Return `true` if and only if the vector u of $V = \mathbb{F}_q^n$ does not belong to the q-ary code C of length n.

`C eq D`

Return `true` if and only if the q-ary codes C and D are equal.

`C ne D`

Return `true` if and only if the q-ary codes C and D are not equal.

Example H1E13

```
> V  := VectorSpace(GF(2), 5);
> C1 := QaryCode([V![1,1,1,1,1],V![0,0,0,0,0],
                  V![1,1,0,0,0],V![1,0,1,1,1]]);
> C2 := QaryCode(Matrix([V![1,1,1,1,1],V![0,0,0,0,0],
                         V![1,1,0,0,0],V![1,0,1,1,1]]));
```

```
> C1 eq C2;
true

> C3 := RandomQaryCode(GF(3), 8, 108, 2);
> C4 := RandomQaryCode(GF(3), 8, 27);
> C3 eq C4;
false

> C5 := QaryCode(C4`Kernel);
> C5 subset C4;
true
```

1.6 Properties of Qary Codes

`IsQary(C)`

Return `true` if and only if C is a q-ary code.

`IsLinear(C)`

Return `true` if and only if C is a linear q-ary code.

1.7 The Weight and Distance Distribution

`QaryMinimumWeight(C)`

`AlgMethod` MonStgElt *Default*: "Auto"

Given a q-ary code C, determine the minimum weight of the codewords belonging to the code C.

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

`MinimumDistance(C)`

`AlgMethod` MonStgElt *Default*: "Auto"

Given a q-ary code C, determine the minimum distance of the codewords belonging to the code C.

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

Note that for distance invariant codes, the minimum weight and distance coincide.

`WeightDistribution(C)`

Determine the weight distribution of the q-ary code C. The distribution is returned in the form of a sequence of tuples, where the i-th tuple contains the i-th weight, w_i say, and the number of codewords having weight w_i.

Example H1E14

```
> V := VectorSpace(GF(2), 31);
> C_kernel := SimplexCode(5);
> C_representatives := [
    V!0,
    V![ 0,0,1,0,0,0,1,1,1,0,0,1,1,0,1,0,0,1,1,1,1,0,0,0,1,0,1,1,1,1,0],
    V![ 0,1,0,1,1,0,1,0,1,0,1,1,1,1,0,0,1,0,1,1,1,0,1,0,0,1,1,1,1,0,1],
    V![ 0,0,0,0,0,1,1,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,1,1,1,1,0,1,0,1,1]
];
> C := QaryCode(C_kernel, C_representatives);
> QaryMinimumWeight(C);
10;
> MinimumDistance(C);
8

> weightDistribution := WeightDistribution(C);
> weightDistribution;
[ <0, 1>, <10, 3>, <11, 4>, <12, 9>, <13, 6>, <14, 15>, <15, 8>, <16, 50>,
<17, 8>, <18, 13>, <19, 4>, <20, 3>, <21, 2>, <22, 1>, <24, 1> ]
> &+[wD[2] : wD in weightDistribution ] eq #C;
true
```

Bibliography

[1] H. Bauer, B. Ganter, and F. Hergert, "Algebraic techniques for nonlinear codes," *Combinatorica*, vol. 3, pp. 21-33, 1983.

[2] 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.

[3] J. Borges, C. Fernández, 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.

[4] A. Brouwer, "Table of general binary codes," http://www.win.tue.nl/ aeb/codes/binary.html#toc1

[5] C. Carlet. "$\mathbb{Z}_{2^k}$-linear codes," *IEEE Trans. on Information Theory*, vol. 44, pp. 1543-1547, 1998.

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

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

[8] C. Fernández-Córdoba, J. Pujol, and M. Villanueva, "On rank and kernel of $\mathbb{Z}_4$-linear codes," *Lecture Notes in Computer Science*, n. 5228, 2008.

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

[10] 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.

[11] O. Heden, "On perfect p-ary codes of length $p+1$", *Designs, Codes and Cryptography*, vol. 46, no. 1, pp. 45-56, 2008.

[12] O. Heden, "Perfect codes from the dual point of view I", *Discrete Mathematics*, vol. 308, no. 24, pp. 6141-6156, 2008.

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

[14] D. S. Krotov, "$\mathbb{Z}_4$-linear Hadamard and extended perfect codes," in *WCC2001, International Workshop on Coding and Cryptography*, ser. Electron. Notes Discrete Math., 2001, vol. 6, pp. 107–112.

[15] S. Litsyn, E.M. Rains and N.J.A Sloane, "Table of nonlinear binary codes," `http://www.eng.tau.ac.il/ litsyn/tableand/`

[16] P. R. J. Östergård and O. Pottonen, "The perfect binary one-error-correcting codes of length 15: part I-classification," *IEEE Trans. on Information Theory*, vol. 55, no. 10, pp. 4657-4660, 2009.

[17] P. R. J. Östergård, O. Pottonen, and K. T. Phelps, "The perfect binary one-error-correcting codes of length 15: Part IIproperties," *IEEE Trans. on Information Theory*, vol. 56, no. 6, pp. 2571-2582, 2010.

[18] K. T. Phelps, J. Rifà, and M. Villanueva, "Kernels and p-kernels of p^r-ary 1-perfect codes," *Designs, Codes and Cryptography*, vol. 37, no. 2, pp. 243-261, 2001.

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

[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] J. Pujol, F. Zeng, and M. Villanueva, "Representation, constructions and minimum distance computation of binary nonlinear codes," in Proceedings of 19th International Conference on Aplications of Computer Algebra, Málaga, Spain, July 2-6, pp. 142-143, 2013.

[22] F. I. Solov'eva "On $\mathbb{Z}_4$-linear codes with parameters of Reed-Muller codes," *Problems of Information Transmission*, vol. 43(1), pp. 26-32, 2007.

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

[24] M. Villanueva, F. Zeng, and J. Pujol, "Nonlinear Q-ary codes: Constructions and minimum distance computation," in Proceedings of Encuentros de Álgebra Computacional y Aplicaciones, Barcelona, Spain, June 18-20, pp. 171-174, 2014.

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

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