Lecture 2: Coding Theory with GUAVA

Michel Lavrauw
University of Primorska

GAP Days Spring 2025, VUB

Overview

- Introduction to Coding Theory
- Using the GUAVA package
- Available code constructions in GUAVA
- Code manipulations
- Bounds on codes

Basic Concepts

- A **code** is a subset of A^n where A is some non-empty set (the **alphabet** A).
- A **linear code** is a subspace of $\mathbb{F}_q^n$ (alphabet $A = \mathbb{F}_q$).
- The parameters of a code are its **length** n, **dimension** k, and **minimum distance** d.
- Used for all kinds of data transmission: detect and correct errors.

The GUAVA package

```
gap> LoadPackage("guava");

  ___________________________________________________________________________
 /        ___
||       /   \                 /\    Version 2.9.6
||      ||   ||  |\    |      /  \               /\       Erhard Aichinger
 \___   ||   ||  |\\   |     /____\_____________/__\      Franz Binder
     \  ||   ||  | \\  |    /      \     ||    /    \     Juergen Ecker
     ||  \___/   |  \\ |   /        \    ||   /      \    Peter Mayr
     ||          |   \\|  /          \   ||               Christof Noebauer
  \___/          |    \|                 ||

 System    Of   Nearrings     And      Their Applications
 Info: https://gap-packages.github.io/sonata/


   ____                             |
  /                \          /   --+--  Version 3.19
 /      |     | |\ \        / /|    |
|    __  |     | | \ \      / / |          the GUAVA Group
|      | |     | |--\ \    / /--|
 \     | |     | |   \ \  / /   |
  \___/   \___/  |    \ \/ /    |

 Homepage: https://gap-packages.github.io/guava
 Report issues at https://github.com/gap-packages/guava/issues
true
```

The GUAVA package

The functions in GUAVA can be divided into three subcategories:

1. Construction of codes: GUAVA can construct three types of codes: unrestricted, linear and cyclic codes. Information about the code, is stored in a record-like data structure.
2. Manipulations of codes: to construct a new code from (a) given code(s).
3. Computations of information about codes.

Creating a Code

The first code one might think of is the "repetition code".

The alphabet is

```
A:=GF(2);
```

and the lenght is

```
n:=3;
```

The list of codewords is

```
list:=["000","111"];
```

The code is defined from this list using ElementsCode

```
gap> C:=ElementsCode(list,A);
a (3,2,1..3)1 user defined unrestricted code over GF(2)
```

GUAVA calls this an "unrestricted code" (the other types of codes are "linear code" and "cyclic code")

Codewords, distance, weight

Note !

```
gap> IsCodeword("000");
false
gap> IsCodeword(Codeword("000"));
true
```

Distance between codewords is the Hamming distance.

```
c1:=Codeword("000"); c2:=Codeword("111");
DistanceCodeword(c1,c2); MinimumDistance(C);
```

The number of nonzero symbols in a codeword (the alphabet A always contain some "zero") is called the "Hamming weight" of the codeword:

```
Weight(c1);Weight(c2);
```

We can also immediatlely ask for all the weights in the code C

```
WeightDistribution(C);
CodeWeightEnumerator(C);
```

Linear codes: generator matrix, check matrix, dual code

Parameters of a code C

```
Length(C);
Dimension(C);
MinimumDistance(C);
CoveringRadius(C);
```

Generator matrix: rows are a basis C

```
G:=GeneratorMat(C);
```

and **partity check matrix** of C

```
H:=CheckMat(C);
```

whose rows are a basis for the $C^{\perp}$, the **dual code**

```
DualCode(C);
```

Example Code, distance, weight

```
gap> G:=List([1..4],i->Random(GF(2)^7));;
gap> Display(G);
 . 1 . . 1 1 1
 1 1 . . . . .
 1 . 1 . . . .
 1 1 . 1 1 1 1
gap> C:=GeneratorMatCode(G,GF(2));
a linear [7,4,1..3]1..3 code defined by generator matrix over GF(2)
gap> CodeWeightEnumerator(C);
4*x_1^6+5*x_1^4+6*x_1^2+1
gap> WeightDistribution(C);
[ 1, 0, 6, 0, 5, 0, 4, 0 ]
gap> Display(CheckMat(C));
 1 1 1 1 1 . .
 1 1 1 1 . 1 .
 1 1 1 1 . . 1
gap> DualCode(C)=GeneratorMatCode(CheckMat(C),GF(2));
true
```

Intermezzo: Coding theory and Galois geometry

Reed–Solomon Code from a Normal Rational Curve

A NRC $\mathcal{A}$ is the image of $\nu_{k-1} : \mathrm{PG}(1, q) \to \mathrm{PG}(k-1, q)$:

$$[s : t] \mapsto [s^{k-1} : s^{k-2}t : \cdots : t^{k-1}]$$

It consists of $q+1$ points in $\mathrm{PG}(k-1, q)$, no k contained in a hyperplane (arc).

Reed–Solomon Code from a Normal Rational Curve

A NRC $\mathcal{A}$ is the image of $\nu_{k-1} : \mathrm{PG}(1, q) \to \mathrm{PG}(k-1, q)$:

$$[s : t] \mapsto [s^{k-1} : s^{k-2}t : \cdots : t^{k-1}]$$

It consists of $q+1$ points in $\mathrm{PG}(k-1, q)$, no k contained in a hyperplane (arc).

Matrix G with columns: vectors representing the points of $\mathcal{A}$.

Reed–Solomon Code from a Normal Rational Curve

A NRC $\mathcal{A}$ is the image of $\nu_{k-1} : \mathrm{PG}(1, q) \to \mathrm{PG}(k-1, q)$:

$$[s : t] \mapsto [s^{k-1} : s^{k-2}t : \cdots : t^{k-1}]$$

It consists of $q + 1$ points in $\mathrm{PG}(k-1, q)$, no k contained in a hyperplane (arc).

Matrix G with columns: vectors representing the points of $\mathcal{A}$.

The code generated by the rows of G is a Reed-Solomon code

$$C = \{\mathrm{m}G \mid \mathrm{m} \in \mathrm{GF}(q)^k\} \quad \Rightarrow \quad [n, k, n-k+1]$$

Reed–Solomon Code from a Normal Rational Curve

A NRC $\mathcal{A}$ is the image of $\nu_{k-1} : \mathrm{PG}(1, q) \to \mathrm{PG}(k-1, q)$:

$$[s : t] \mapsto [s^{k-1} : s^{k-2}t : \cdots : t^{k-1}]$$

It consists of $q+1$ points in $\mathrm{PG}(k-1, q)$, no k contained in a hyperplane (arc).

Matrix G with columns: vectors representing the points of $\mathcal{A}$.

The code generated by the rows of G is a Reed-Solomon code

$$C = \{\mathrm{m}G \mid \mathrm{m} \in \mathrm{GF}(q)^k\} \quad \Rightarrow \quad [n, k, n-k+1]$$

Reed Solomon codes reach the Singleton bound!

They are Maximum Distance Separable Codes (MDS).

Geometry Behind the Code: Twisted Cubic over $\mathrm{PG}(3,5)$.

Twisted Cubic in $\mathbb{P}^3(\mathrm{GF}(5))$:

$$[s:t] \mapsto [s^3 : s^2 t : st^2 : t^3] \quad \text{for } [s:t] \in \mathbb{P}^1(\mathrm{GF}(5))$$

Points on the curve:

$[1:0] \mapsto [1:0:0:0]$

$[1:1] \mapsto [1:1:1:1]$

$[1:2] \mapsto [1:2:4:3]$

$[1:3] \mapsto [1:3:4:2]$

$[1:4] \mapsto [1:4:1:4]$

$[0:1] \mapsto [0:0:0:1]$

Geometry Behind the Code: Twisted Cubic over $\mathrm{PG}(3,5)$.

Twisted Cubic in $\mathbb{P}^3(\mathrm{GF}(5))$:

$$[s:t] \mapsto [s^3 : s^2t : st^2 : t^3] \quad \text{for } [s:t] \in \mathbb{P}^1(\mathrm{GF}(5))$$

Points on the curve:

$[1:0] \mapsto [1:0:0:0]$

$[1:1] \mapsto [1:1:1:1]$

$[1:2] \mapsto [1:2:4:3]$

$[1:3] \mapsto [1:3:4:2]$

$[1:4] \mapsto [1:4:1:4]$

$[0:1] \mapsto [0:0:0:1]$

Generator matrix G:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 4 & 4 & 1 & 0 \\ 0 & 1 & 3 & 2 & 4 & 0 \\ 0 & 1 & 2 & 2 & 4 & 1 \end{bmatrix}$$

Geometry Behind the Code: Twisted Cubic over $\mathrm{PG}(3,5)$.

Twisted Cubic in $\mathbb{P}^3(\mathrm{GF}(5))$:

$$[s:t] \mapsto [s^3 : s^2t : st^2 : t^3] \quad \text{for } [s:t] \in \mathbb{P}^1(\mathrm{GF}(5))$$

Points on the curve:

$$[1:0] \mapsto [1:0:0:0]$$
$$[1:1] \mapsto [1:1:1:1]$$
$$[1:2] \mapsto [1:2:4:3]$$
$$[1:3] \mapsto [1:3:4:2]$$
$$[1:4] \mapsto [1:4:1:4]$$
$$[0:1] \mapsto [0:0:0:1]$$

Generator matrix G:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 4 & 4 & 1 & 0 \\ 0 & 1 & 3 & 2 & 4 & 0 \\ 0 & 1 & 2 & 2 & 4 & 1 \end{bmatrix}$$

Encoding a message:

$$\mathrm{m} = (1,0,3,2) \quad \Rightarrow \quad \mathrm{c} = \mathrm{m} \cdot G$$

Geometry Behind the Code: Twisted Cubic over $\mathrm{PG}(3,5)$.

Twisted Cubic in $\mathbb{P}^3(\mathrm{GF}(5))$:

$$[s:t] \mapsto [s^3 : s^2 t : st^2 : t^3] \quad \text{for } [s:t] \in \mathbb{P}^1(\mathrm{GF}(5))$$

Points on the curve:

$$[1:0] \mapsto [1:0:0:0]$$
$$[1:1] \mapsto [1:1:1:1]$$
$$[1:2] \mapsto [1:2:4:3]$$
$$[1:3] \mapsto [1:3:4:2]$$
$$[1:4] \mapsto [1:4:1:4]$$
$$[0:1] \mapsto [0:0:0:1]$$

Generator matrix G:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 4 & 4 & 1 & 0 \\ 0 & 1 & 3 & 2 & 4 & 0 \\ 0 & 1 & 2 & 2 & 4 & 1 \end{bmatrix}$$

Encoding a message:

$$\mathrm{m} = (1,0,3,2) \quad \Rightarrow \quad \mathrm{c} = \mathrm{m} \cdot G$$

This is a $[6,4,3]$-Reed–Solomon code
from a classical algebraic curve!

Reed–Solomon Codes in the CD Player

CDs use a two-level RS code: For the CD player:

$$[n, k]_q = [28, 24]_{256}, \quad d = 5$$

(i.e., 24 data bytes, 4 parity bytes, $q = 256$)

- **CIRC** = Cross-Interleaved Reed–Solomon Code
- Combines two RS codes with interleaving
- Spreads burst errors (e.g., scratches) across multiple codewords
- Allows accurate reconstruction even when parts are unreadable

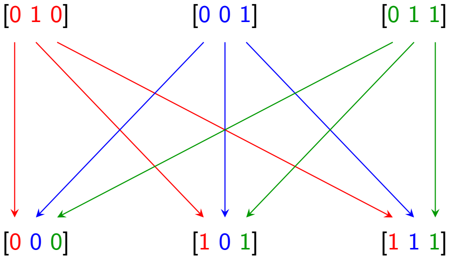
Result: **Up to 3500 erroneous bits per second can be corrected!**

The Interleaving Technique in CD Error Correction

Correcting errors caused by a scratch (*burst error*) on a CD.

Idea:
Interleaving spreads each codeword across multiple positions. A burst error (e.g. scratch) affects only parts of each codeword, allowing **error-correction**.

Original Codewords (before interleaving)



Interleaved Codewords (spread out)

Reed–Solomon Codes in QR Codes

A QR code stores information (text, URL, etc.) as a 2D array of black and white squares.

Reed–Solomon Codes in QR Codes

A QR code stores information (text, URL, etc.) as a 2D array of black and white squares.

The data is converted to byte sequences, and then:

- Divided into blocks (number depends on QR version and error level)
- Each block is encoded with a **RS code in** $\mathrm{PG}(222, 2^8)$
- The encoded bytes are **interleaved** and placed in the QR grid following a specific pattern.

Reed–Solomon Codes in QR Codes

A QR code stores information (text, URL, etc.) as a 2D array of black and white squares.

The data is converted to byte sequences, and then:

- Divided into blocks (number depends on QR version and error level)
- Each block is encoded with a **RS code in** $\mathrm{PG}(222, 2^8)$
- The encoded bytes are **interleaved** and placed in the QR grid following a specific pattern.

Each QR code block secretly carries the geometry of a NRC in a 222-dimensional projective space over $\mathrm{GF}(2^8)$

Encoding and Decoding

A **message**

```
m := [1,0,1];
```

the encoded message: a **codeword**

```
codeword := EncodeWord(C, m); (or codeword := m*C;)
```

a(received) **word**

```
received := [1,1,1,1,0];
DecodeWord(C, received);
```

the decoded received word

```
DecodeWord(C, received);
```

Decoding

```
gap> m:="1010";
gap> c := m*C;  # encoding
[ 1 0 1 1 0 1 0 ]
gap> Decode(C, c);  # decoding
[ 1 0 1 0 ]
gap> w:=c+"1000000";  # introduce one error
[ 0 0 0 1 0 1 0 ]
gap> Decode(C,w);  # the correct message
[ 1 0 1 0 ]
gap> Decodeword(C,w); # the correct codeword
[ 1 0 1 1 0 1 0 ]
```

NOTE: If the code record has a field "SpecialDecoder", this special algorithm is used to decode the vector, otherwise: **Syndrome Decoding**.

Syndrome Decoding

```
sa:=StandardArray(C);;
sa[1]; # these are all the codewords in C
sa[2]; # this is the first coset of C,
# where l1=sa[2][1] is the coset leader
```

SyndromeTable returns a syndrome table of a linear code C, consisting of two columns. The first column consists of the **coset leaders** that correspond to the **syndrome vectors** in the second column.

```
gap> st:=SyndromeTable(C);
```

```
 [ [ [ 0 0 0 0 0 0 0 ], [ 0 0 0 ] ], [ [ 1 0 0 0 0 0 0 ], [ 0 0 1 ] ],
[ [ 0 1 0 0 0 0 0 ], [ 0 1 0 ] ], [ [ 0 0 1 0 0 0 0 ], [ 0 1 1 ] ],
 [ [ 0 0 0 1 0 0 0 ], [ 1 0 0 ] ], [ [ 0 0 0 0 1 0 0 ], [ 1 0 1 ] ],
[ [ 0 0 0 0 0 1 0 ], [ 1 1 0 ] ], [ [ 0 0 0 0 0 0 1 ], [ 1 1 1 ] ] ]
```

Syndrome Decoding Example

```
gap> m:="1010";
"1010"
gap> c := m*C;                      # encoding
[ 1 0 1 1 0 1 0 ]
gap> w:=c+"1000000";
[ 0 0 1 1 0 1 0 ]
gap> H:=CheckMat(C);;
gap> s:=H*w;          # compute the syndrome w
[ 0 0 1 ]
gap>
gap> coset:=First(st,r->r[2]=s); # # according to the syndrome table, tl
[ [ 1 0 0 0 0 0 0 ], [ 0 0 1 ] ]
gap> ev:=coset[1];  # the coset leader, which is the error vector
[ 1 0 0 0 0 0 0 ]
gap> c:=w-ev;  # the corrected codeword
[ 1 0 1 1 0 1 0 ]
gap> c=Decodeword(C,w);
true
```

Available code constructions in GUAVA

```
EC := ElementsCode( ["1000", "1101", "0011" ], GF(2) );
C := HammingCode(r,GF(q));
RS := ReedSolomonCode(q-1, n-k+1);  # ExtendedReedSolomonCode
GRS := GeneralizedReedSolomonCode( [a_1,..,a_n] , k , GF(q)[X] );
RM := ReedMullerCode( r, k ); # r-th order binary of length 2^k
GRM := GeneralizedReedMullerCode( pts, r, GF(q) );
  # evaluate poly's in GF(q)[X_1,.., X_d] of degree \leq r at pts
x := Indeterminate(GF(q),"x");
GC := GoppaCode(x^2+x+1,Elements(GF(q)));
BG := BinaryGolayCode(); # ExtendedBinaryGolayCode();)
TG := TernaryGolayCode(); # ExtendedTernaryGolayCode();
EVC := EvaluationCode(elems,pol_list,pol_ring);
```

Cyclic codes in GUAVA

Cyclic codes are linear codes satisfying $c \in C \Rightarrow \sigma(c) \in C$ where $\sigma = (1, .., n) \in \mathrm{Sym}(n)$. They correspond to ideals in $\mathrm{GF}(q)[X]/(X^n - 1)$.

```
gap> x:= Indeterminate( GF(2), "x" );; P:= x^2+1;
x^2+Z(2)^0
gap> C1 := GeneratorPolCode(P, 7, GF(2));
a cyclic [7,6,1..2]1 code defined by generator polynomial over GF(2)
gap> GeneratorPol( C1 );
x+Z(2)^0
gap>  x := Indeterminate( GF(3), "x" );; P:= x^2+2;
x^2-Z(3)^0
gap> H := CheckPolCode(P, 7, GF(3));
a cyclic [7,1,7]4 code defined by check polynomial over GF(3)
gap> CheckPol(H);
x-Z(3)^0
gap> Gcd(P, X(GF(3))^7-1);
x-Z(3)^0
```

Cyclic codes defined by its "roots"

```
gap> C2;
a cyclic [7,1,7]4 code defined by check polynomial over GF(3)
gap> f:=GeneratorPol(C2);
x^6+x^5+x^4+x^3+x^2+x+Z(3)^0
gap> roots:=RootsOfCode(C2);
[ Z(3^6)^104, Z(3^6)^208, Z(3^6)^312, Z(3^6)^416, Z(3^6)^520, Z(3^6)^624 ]
gap> Set(roots,a->f(a));
[ 0*Z(3) ]
```

```
gap> a := PrimitiveUnityRoot( 3, 14 );
Z(3^6)^52
gap> C1 := RootsCode( 14, [ a^0, a, a^3 ] );
a cyclic [14,7,2..6]3..7 code defined by roots over GF(3)
gap> GeneratorPol(C1);
x^7+x^6-x^5+x^4-x^3+x^2-x-Z(3)^0
gap> RootsOfCode(C1);
[ Z(3)^0, Z(3^6)^52, Z(3^6)^156, Z(3^6)^260, Z(3^6)^468, Z(3^6)^572,
  Z(3^6)^676 ]
gap> ForAll([a^0,a,a^3],y->y in RootsOfCode(C1));
true
```

BCHCode (Bose-Chaudhuri-Hockenghem)

Let $\min(\alpha, \mathbb{F}_q) \in \mathbb{F}_q[X]$ denote the minimal polynomial of an element $\alpha \in \mathbb{F}_{q^m}$. A BCH code over $\mathbb{F}_q$ of length n and **designed minimal distance** δ is a cyclic code with generator polynomial

$$g(X) = lcm\{\min(\beta^i, \mathbb{F}_q) \ : \ a \leq i \leq a + \delta - 2\}$$

where $\beta \in \mathbb{F}_{q^m}$ is a primitive n-th root of unity and a is some integer such that

$$\beta^a, \ldots, \beta^{a+\delta-2}$$

are $\delta - 1$ distinct elements of $\mathbb{F}_{q^m}$.

Example of BCHCode

Aim: Construct a $[15, k, d \geq 7]$-code over $\mathbb{F}_2$.

```
gap> a:=Z(16); # primitive 15-th root of unity
Z(2^4)
gap> CyclotomicCosets(2,15);
[ [ 0 ], [ 1, 2, 4, 8 ], [ 3, 6, 12, 9 ], [ 5, 10 ], [ 7, 14, 13, 11 ] ]
gap> Union(List([2,3,4],i->last[i])); # contains 6 consecutive integers
[ 1, 2, 3, 4, 5, 6, 8, 9, 10, 12 ]
gap> roots:=List([1..6],i->a^i);
[ Z(2^4), Z(2^4)^2, Z(2^4)^3, Z(2^4)^4, Z(2^2), Z(2^4)^6 ]
gap> C:=RootsCode(15,roots);
a cyclic [15,5,2..7]5 code defined by roots over GF(2)
gap> MinimumDistance(C); # by construction at least 7
7
gap> BCHCode(15,7,GF(2));
a cyclic [15,5,7]5 BCH code, delta=7, b=1 over GF(2)
gap> C=last;
true
```

Code manipulations in GUAVA

- ExtendedCode(C[, i])
- EvenWeightSubcode(C)
- PuncturedCode(C) or PuncturedCode(C , L)
- ExpurgatedCode(C, L)
- AugmentedCode(C, L)
- ShortenedCode(C[, L])
- LengthenedCode(C[, i])
- SubCode(C[, s])
- ResidueCode(C[, c])
- ConversionFieldCode(C)

Example 1 of code manipulations

```
gap> C1 := HammingCode( 3, GF(2) );
a linear [7,4,3]1 Hamming (3,2) code over GF(2)
gap> C2 := ExtendedCode( C1 );
a linear [8,4,4]2 extended code
gap> CodeWeightEnumerator(C2);
x_1^8+14*x_1^4+1
gap> C3 := EvenWeightSubcode( C1 );
a linear [7,3,4]2..3 even weight subcode
gap> CodeWeightEnumerator(C3);
7*x_1^4+1
gap> PuncturedCode(C2);
a linear [7,4,3]1 punctured code
gap> PuncturedCode(C2,[1,2]); # the minimum distance might decrease
a linear [6,4,2]1 punctured code
```

Example 2 of code manipulations

```
gap> G:=[ [ Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ],
>   [ 0*Z(2), 0*Z(2), 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, Z(2)^0, Z(2)^0 ],
>   [ 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0, 0*Z(2), 0*Z(2), Z(2)^0, Z(2)^0 ],
>   [ 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0, 0*Z(2), Z(2)^0 ] ];;
gap> Display(G);
 1 1 1 1 1 1 1 1
 . . . . 1 1 1 1
 . . 1 1 . . 1 1
 . 1 . 1 . 1 . 1
gap> C1:=GeneratorMatCode(G,GF(2));
a linear [8,4,1..4]2 code defined by generator matrix over GF(2)
gap> MinimumDistance(C1);
4
gap> C2:=AugmentedCode(C1,["00000011","00000101","00010001"]);
a linear [8,7,1..2]1 code, augmented with 3 word(s)
gap> MinimumDistance(C2);
2
```

Example 3 of code manipulations

ShortenedCode(C): this is done by removing all codewords that start with a non-zero entry, after which the first column is cut off. If C was a linear $[n, k, d]$ code, the shortened code is a $[n-1, \leq k, \geq d]$ code.

```
gap> C3 := ElementsCode( ["1000", "1101", "0011" ], GF(2) );
a (4,3,1..4)2 user defined unrestricted code over GF(2)
gap> C4 := ShortenedCode( C3 );
a (3,2,1..3)1..2 shortened code
gap> AsSSortedList( C4 );
[ [ 0 0 0 ], [ 1 0 1 ] ]
gap> C5 := HammingCode( 5, GF(2) );
a linear [31,26,3]1 Hamming (5,2) code over GF(2)
gap> C6 := ShortenedCode( C5, [ 1, 2, 3 ] );
a linear [28,23,3]2 shortened code
```

Bounds on codes in GUAVA

Upper bounds on the size (dimension) of a code of length n and minimum distance d over GF(q)

- UpperBoundSingleton(n, d, q)
- UpperBoundHamming(n, d, q) (sphere-packing bound)
- UpperBoundJohnson(n, d)
- UpperBoundPlotkin(n, d, q)
- UpperBoundElias
- UpperBoundGriesmer(n, d, q)
- UpperBound(n, d, q) (best known upper bound $A(n, d)$)

Lower bounds on the size

- LowerBoundGilbertVarshamov(n, d, q)
- LowerBoundSpherePacking(n, d, q)

Upper and lower bounds on the minimum distance and the covering radius

- BoundsMinimumDistance(n, k, F)
- BoundsCoveringRadius(C)

Example of bounds on codes in GUAVA

```
gap> UpperBoundSingleton(4, 3, 5);
25
gap> C := ReedSolomonCode(4,3);; Size(C);
25
gap> IsMDSCode(C);
true
gap> UpperBoundHamming( 15, 3, 2 );
2048
gap> C := HammingCode( 4, GF(2) );
a linear [15,11,3]1 Hamming (4,2) code over GF(2)
gap> Size( C );
2048
gap> IsPerfectCode(C);
true
gap> Filtered([1..10],i->IsGriesmerCode( HammingCode( i, GF(2) ) ));
[ 2, 3 ]
```

Summary

- GUAVA provides tools for constructing and analysing codes.
- Basic operations: define codes, encode/decode, compute parameters.
- Advanced constructions: Reed–Solomon codes, cyclic codes, etc.
- Tools for manipulating codes: puncture, shorten, etc.
- Testing existence of codes with given parameters: bounds on codes.