

The HERmitian Package

Divisors and Riemann-Roch Spaces of Algebraic Function Fields of Hermitian Curves

Version 0.1

14 March 2019

Gábor P. Nagy
Sabira El Khalfaoui

Gábor P. Nagy Email: nagy@math.u-szeged.hu
Homepage: <http://www.math.u-szeged.hu/~nagy/>

Sabira El Khalfaoui Email: sabira@math.u-szeged.hu

Copyright

© 2019 by Gábor P. Nagy

HERmitian package is free software; you can redistribute it and/or modify it under the terms of the [GNU General Public License](#) as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

Acknowledgements

We appreciate very much all past and future comments, suggestions and contributions to this package and its documentation provided by **GAP** users and developers.

Contents

1	Introduction	4
1.1	Unpacking the HERmitian Package	4
1.2	Loading the HERmitian Package	5
1.3	Testing the HERmitian Package	5
2	Mathematical background	6
2.1	Blabla	6
3	How to use the package	7
3.1	Hermitian curves	7
3.2	Hermitian divisors	9
3.3	Hermitian Riemann-Roch spaces	12
3.4	Hermitian AG-codes	12
3.5	Utilities for Hermitian AG-codes	15
4	An example: BCH codes as Hermitian AG-codes	16
	References	18
	Index	19

Chapter 1

Introduction

This chapter describes the GAP package HERmitian. This package implements functionalities for divisors and Riemann-Roch spaces of an algebraic function field of Hermitian.

If you are viewing this with on-line help, type:

Example

```
gap> ?HERmitian package
```

to see the functions provided by the HERmitian package.

1.1 Unpacking the HERmitian Package

If the HERmitian package was obtained as a part of the GAP distribution from the “Download” section of the GAP website, you may proceed to Section ???. Alternatively, the HERmitian package may be installed using a separate archive, for example, for an update or an installation in a non-default location (see (**Reference: GAP Root Directories**)).

Below we describe the installation procedure for the .tar.gz archive format. Installation using other archive formats is performed in a similar way.

To install the HERmitian package, unpack the archive file, which should have a name of form HERmitian-XXX.tar.gz for some version number XXX, by typing

```
gzip -dc HERmitian-XXX.tar.gz | tar xpv
```

It may be unpacked in one of the following locations:

- in the pkg directory of your GAP 4 installation;
- or in a directory named .gap/pkg in your home directory (to be added to the GAP root directory unless GAP is started with -r option);
- or in a directory named pkg in another directory of your choice (e.g. in the directory mygap in your home directory).

In the latter case one must start GAP with the -l option, e.g. if your private pkg directory is a subdirectory of mygap in your home directory you might type:

```
gap -l ";myhomedir/mygap"
```

where myhomedir is the path to your home directory, which (since GAP 4.3) may be replaced by a tilde (the empty path before the semicolon is filled in by the default path of the GAP 4 home directory).

1.2 Loading the HERmitian Package

To use the HERmitian Package you have to request it explicitly. This is done by calling `LoadPackage` (**Reference: `LoadPackage`**):

```
gap> LoadPackage("HERmitian");
-----
Loading  HERmitian 0.1
by Gábor P. Nagy (http://www.math.u-szeged.hu/~nagyg)
For help, type: ?HERmitian package
-----
true
```

If GAP cannot find a working binary, the call to `LoadPackage` will still succeed but a warning is issued informing that the `HelloWorld()` function will be unavailable.

If you want to load the HERmitian package by default, you can put the `LoadPackage` command into your `gaprc` file (see Section **(Reference: The `gap.ini` and `gaprc` files)**).

1.3 Testing the HERmitian Package

You can run tests for the package by

```
gap> Test(Filename(DirectoriesPackageLibrary("HERmitian"), "../tst/testall.tst"));
```

Chapter 2

Mathematical background

2.1 Blabla

Blabla. [[Sti09](#)] [[HKT08](#)] [[GAP17](#)]

Chapter 3

How to use the package

3.1 Hermitian curves

The following functions are available:

3.1.1 IsHermitian_Curve

▷ `IsHermitian_Curve(obj)` (Category)

Hermitian curve $H(q)$ is an algebraic curve over an algebraically closed field, having an affine equation $X^{q+1} = Y^q + Y$. The base field of $H(q)$ is $GF(q^2)$.

3.1.2 Hermitian_Curve

▷ `Hermitian_Curve(K, hratfn)` (operation)

returns the corresponding Hermitian curve $H(q)$ over the algebraic closure of the field K . The indeterminates X, Y of $hratfn$ generate the corresponding Hermitian function field $K(X, Y)$ such that $X^{q+1} = Y^q + Y$. K must be a finite field of square order. The points of $H(q)$ are either affine $P(a, b)$ satisfying $a^{q+1} = b^q + b$, or the infinite point `[infinity]`. One can use the `in` operation to test if a point lies on the Hermitian curve.

3.1.3 IndeterminatesOfHermitian_Curve

▷ `IndeterminatesOfHermitian_Curve(Hq)` (function)

returns the indeterminates of the function field of the Hermitian curve \mathcal{C} .

3.1.4 UnderlyingField

▷ `UnderlyingField(Hq)` (attribute)

The underlying field of a Hermitian curve is the field of coefficients of the corresponding algebraic function field, it is a finite field of square order.

3.1.5 RandomPlaceOfGivenDegreeOfHermitian_Curve

▷ `RandomPlaceOfGivenDegreeOfHermitian_Curve(Hq, d)` (operation)

returns a random place of degree d of the Hermitian curve Hq , that is, a place defined over the field $GF(q^{2d})$. Notice that the place at infinity has degree 1.

Example

```
gap> Y:=Indeterminate(GF(9),"Y");
Y
gap> C:=Hermitian_Curve(GF(9),Y);
<GZ curve over GF(9) with indeterminate Y>
gap> aut:=AutomorphismGroup(C);
<group of GZ curve automorphisms of size 720>
gap> Random(aut);
Hermitian_CurveAut([ [ Z(3)^0, Z(3^2)^3 ], [ Z(3^2)^5, Z(3) ] ])
```

3.1.6 FrobeniusAutomorphismOfHermitian_Curve

▷ `FrobeniusAutomorphismOfHermitian_Curve(Hq)` (attribute)

returns the Frobenius automorphism of the underlying field of the Hermitian curve Hq . More precisely, the output is an AC-Frobenius automorphism in the sense of the package `OnAlgClosure`, acting on the algebraic closure of the underlying finite field.

3.1.7 IsHermitian_CurveAutomorphism

▷ `IsHermitian_CurveAutomorphism(obj)` (Category)

With automorphisms of an algebraic curve C one means the automorphisms of the corresponding algebraic function field $K(C)$. For Hermitian curves over finite fields, the algebraic function field is the field $K(t)$ of rational functions in one indeterminate. $Aut(K(t))$ consists of fractional linear mappings $t \mapsto \frac{a+bt}{c+dt}$, where $ad - bc \neq 0$. Hence, $Aut(K(t)) \cong PGL(2, K)$.

With fixed Frobenius automorphism $\Phi : x \mapsto x^q$, we can speak of $GF(q)$ -rational automorphisms, or, automorphisms defined over $GF(q)$. These form a subgroup isomorphic to $PGL(2, q)$, having a faithful permutation representation of the set $GF(q) \cup \{\infty\}$ of $GF(q)$ -rational places.

3.1.8 Hermitian_CurveAutomorphism

▷ `Hermitian_CurveAutomorphism(mat)` (operation)

Returns: the automorphism $t \mapsto \frac{a+bt}{c+dt}$ of the Hermitian curve, where M is the nonsingular 2×2 matrix $\begin{pmatrix} a & c \\ b & d \end{pmatrix}$.

3.1.9 AutomorphismGroup

▷ `MatrixGroupToHermitian_CurveAutGroup(matgr, C)` (function)

Returns: the GZ curve automorphism group $\$G\$$ corresponding to the matrix group $matgr$.

The permutation action of *matgr* on the set of rational places of \mathcal{C} is stored as a nice monomorphism of $\$G\$$. \triangleright `AutomorphismGroup(\mathcal{C})` (operation)

Returns: the automorphism group of the Hermitian curve \mathcal{C} . The elements are Hermitian automorphisms. The group is isomorphic to $PGL(2, q)$, where $GF(q)$ is the underlying field of \mathcal{C} .

3.2 Hermitian divisors

The following functions are available:

3.2.1 IsHermitian_Divisor

\triangleright `IsHermitian_Divisor(obj)` (Category)

A Hermitian divisor is a divisor of an algebraic function field of the Hermitian curve $H(q) : X^{q+1} = Y^q + Y$. Hermitian divisors form an additive commutative group.

3.2.2 Hermitian_DivisorConstruct

\triangleright `Hermitian_DivisorConstruct(Hq, pts, ords)` (function)

returns the Hermitian divisor over Hq with points from *pts* and corresponding orders from *ords*. It checks the input.

3.2.3 Hermitian_Divisor

\triangleright `Hermitian_Divisor(Hq, pts, ords)` (operation)

\triangleright `Hermitian_Divisor(Hq, pairs)` (operation)

returns the corresponding Hermitian divisor over the Hermitian curve Hq . The list *pts* must be points of Hq ; the infinite point is [*infinity*]. The list *ords* contains the respective orders. The elements of the list *pairs* are the point-order pairs.

3.2.4 Hermitian_Place

\triangleright `Hermitian_Place(Hq, pt)` (operation)

returns the corresponding place of the Hermitian curve Hq , where *pt* is either an affine point Hq , or the infinite point is [*infinity*].

3.2.5 ZeroHermitian_Divisor

\triangleright `ZeroHermitian_Divisor(Hq)` (operation)

returns the zero divisor over the Hermitian curve Hq .

3.2.6 IsRationalHermitian_Divisor

▷ `IsRationalHermitian_Divisor(D)` (attribute)

Returns true if D is invariant under the Frobenius automorphism of the underlying Hermitian curve.

3.2.7 UnderlyingField

▷ `UnderlyingField(D)` (attribute)

The underlying field of a Hermitian divisor is the field of coefficients of the corresponding Hermitian curve.

3.2.8 Support

▷ `Support(D)` (attribute)

The support of a Hermitian divisor is the set of points with nonzero orders.

3.2.9 Valuation

▷ `Valuation(D , pt)` (operation)

The valuation of a Hermitian divisor D at the point or place pt is its corresponding order.

3.2.10 PrincipalHermitian_Divisor

▷ `PrincipalHermitian_Divisor(Hq , f)` (operation)

returns the principal divisor of the rational function f of the Hermitian curve Hq .

3.2.11 SupremumHermitian_Divisor

▷ `SupremumHermitian_Divisor($D1$, $D2$)` (function)

returns the place-wise maximum of the orders of $D1$ and $D2$.

3.2.12 InfimumHermitian_Divisor

▷ `InfimumHermitian_Divisor($D1$, $D2$)` (function)

returns the place-wise minimum of the orders of $D1$ and $D2$.

3.2.13 PositivePartOfHermitian_Divisor

▷ `PositivePartOfHermitian_Divisor(D)` (function)

returns the positive part of the divisor D .

3.2.14 NegativePartOfHermitian_Divisor

▷ NegativePartOfHermitian_Divisor(D)

(function)

returns the negative part of the divisor D .

Example

```
gap> p1:=Hermitian_Place(C,infinity);
<GZ divisor with support of length 1 over indeterminate Y>
gap> p2:=Hermitian_Place(C,Z(3));
<GZ divisor with support of length 1 over indeterminate Y>
gap> d:=3*p1-4*p2;
<GZ divisor with support of length 2 over indeterminate Y>
gap> Support(d);
[ infinity, Z(3) ]
gap> UnderlyingField(d);
GF(3^2)
gap> Zero(d);
<GZ divisor with support of length 0 over indeterminate Y>
gap> Characteristic(d);
3
gap>
gap> d:=Hermitian_Divisor(C,[Z(27)^2,Z(3),infinity],[5,-1,2]);
<GZ divisor with support of length 3 over indeterminate Y>
gap> Valuation(Z(3),d);
-1
gap> Valuation(Z(3)^2,d);
0
gap>
gap> fr:=AC_FrobeniusAutomorphism(9);
AC_FrobeniusAutomorphism(3^2)
gap> d^fr;
<GZ divisor with support of length 3 over indeterminate Y>
gap> Support(d^fr);
[ infinity, Z(3), Z(3^3)^18 ]
gap> Support(d);
[ infinity, Z(3), Z(3^3)^2 ]
gap>
gap> rf:=Y^8-1;
Y^8-Z(3)^0
gap> List(GF(9),u->Valuation(u,rf));
[ 0, 1, 1, 1, 1, 1, 1, 1, 1 ]
gap> List(GF(9),u->Valuation(u,One(Y)));
[ 0, 0, 0, 0, 0, 0, 0, 0, 0 ]
gap> List(GF(9),u->Valuation(u,Zero(Y)));
[ -infinity, -infinity, -infinity, -infinity, -infinity, -infinity,
  -infinity, -infinity, -infinity ]
gap>
gap>
gap> List(GF(3),u->Valuation(u,One(Y)));
[ 0, 0, 0 ]
gap> List(GF(3),u->Valuation(u,Zero(Y)));
[ -infinity, -infinity, -infinity ]
```

3.3 Hermitian Riemann-Roch spaces

3.3.1 Hermitian_RiemannRochSpaceBasis

▷ `Hermitian_RiemannRochSpaceBasis(D)` (function)

returns a BASIS of the Riemann-Roch space of the Hermitian divisor D , which is defined by $\{f \in K[Y] \mid \text{Div}(f) \geq -D\}$.

Example

```
gap> a:=RandomPlaceOfHermitian_Curve(C,4);
<GZ divisor with support of length 1 over indeterminate Y>
gap> fr:=FrobeniusAutomorphismOfHermitian_Curve(C);
AC_FrobeniusAutomorphism(3^2)
gap> d:=Sum(AC_FrobeniusAutomorphismOrbit(fr,a));
<GZ divisor with support of length 4 over indeterminate Y>
gap> IsRationalHermitian_Divisor(d);
true
gap>
gap> Hermitian_RiemannRochSpaceBasis(3*d);
[ Z(3)^0/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^2/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^3/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^4/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^5/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^6/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^7/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^8/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^9/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^10/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^11/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2),
  Y^12/(Y^12+Y^9+Z(3^2)^2*Y^6+Z(3^2)^3*Y^3+Z(3^2)^2) ]
gap> ForAll(last,x->x=x^fr);
true
```

3.4 Hermitian AG-codes

The following functions are available:

3.4.1 IsHermitian_Code

▷ `IsHermitian_Code(obj)` (Category)
 ▷ `IsHermitian_FunctionalCode(obj)` (Category)
 ▷ `IsHermitian_DifferentialCode(obj)` (Category)

A Hermitian code is an algebraic-geometric (AG) code defined on the Hermitian curve of equation $X^{q+1} = Y^q + Y$. AG-codes are either of functional or of differential type.

3.4.2 GeneratorMatrixOfFunctionalHermitian_CodeNC

▷ `GeneratorMatrixOfFunctionalHermitian_CodeNC(G , pls)` (function)

returns the generator matrix of the functional AG code $C_L(D, G)$, where D is the sum of the degree one places in the list pls . The support of G must be disjoint from pls .

3.4.3 Hermitian_FunctionalCode

▷ `Hermitian_FunctionalCode(G , D)` (operation)

▷ `Hermitian_FunctionalCode(G)` (operation)

returns the functional AG code $C_L(D, G) = \{(f(P_1), \dots, f(P_n)) \mid f \in L(G)\}$. D and G are rational divisors of the Hermitian curve C . $D = P_1 + \dots + D_n$, where P_1, \dots, P_n are degree one places of C . The supports of D and G are disjoint. If D is not given then it is the sum of affine rational places of C . By the Riemann-Roch theorem, functional codes have dimension $\deg(G) + 1 - g$.

3.4.4 Hermitian_DifferentialCode

▷ `Hermitian_DifferentialCode(G , D)` (operation)

▷ `Hermitian_DifferentialCode(G)` (operation)

returns the differential AG code $C_\Omega(D, G) = \{res_{P_1}(\omega), \dots, res_{P_n}(\omega) \mid \omega \in \Omega(G - D)\}$. D and G are rational divisors of the Hermitian curve C . $D = P_1 + \dots + D_n$, where P_1, \dots, P_n are degree one places of C . The supports of D and G are disjoint. If D is not given then it is the sum of affine rational places of C . The differential code is the dual of the corresponding functional code. By the Riemann-Roch theorem, differential codes have dimension $n - \deg(G) - 1 + g$.

3.4.5 Length

▷ `Length(C)` (attribute)

returns the length of the AG code C .

3.4.6 GeneratorMatrixOfHermitian_Code

▷ `GeneratorMatrixOfHermitian_Code(C)` (attribute)

returns the generator matrix of the AG code C in CVEC matrix format.

3.4.7 DesignedMinimumDistance

▷ `DesignedMinimumDistance(C)` (attribute)

returns the designed minimum distance δ of the Hermitian AG code C . When $\deg(G) \geq 2g - 2$, then the general formulas for δ are as follows. For the functional code $C_L(D, G)$, $\delta = n - \deg(G)$, and for the differential code $C_\Omega(D, G)$, $\delta = \deg(G) - (2g - 2)$.

Example

```

gap> code:=Hermitian_FunctionalCode(d);
<[9,5] Hermitian AG-code over GF(3^2)>
gap> Print(code);
Hermitian_FunctionalCode(Hermitian_Divisor(Hermitian_Curve(GF(9),Y),
[ Z(3^8)^302, Z(3^8)^2718, Z(3^8)^3678, Z(3^8)^4782 ],
[ 1, 1, 1, 1 ]),Hermitian_Divisor(Hermitian_Curve(GF(9),Y),
[ 0*Z(3), Z(3)^0, Z(3), Z(3^2), Z(3^2)^2, Z(3^2)^3, Z(3^2)^5,
  Z(3^2)^6, Z(3^2)^7 ],[ 1, 1, 1, 1, 1, 1, 1, 1, 1 ]))
gap> DesignedMinimumDistance(code);
5

```

3.4.8 Hermitian_DecodeToCodeword

▷ Hermitian_DecodeToCodeword(C , w)

(operation)

Let δ be the designed minimum distance of C , and define $t = \lceil (\delta - 1 - g)/2 \rceil$. If there is a codeword $c \in C$ with $d(c, w) \leq t$ then c is returned. Otherwise, the output is fail.

The decoding algorithm is from [Hoholdt-Pellikaan 1995]. The function Hermitian_DECODER_DATA precomputes two matrices which are stored as attributes of the AG code. The decoding consists of solving linear equations.

Example

```

gap> q:=5^3;
125
gap> # construct the curve and the divisors
gap> Y:=Indeterminate(GF(q),"Y");
Y
gap> C:=Hermitian_Curve(GF(q),Y);
<GZ curve over GF(125) with indeterminate Y>
gap> P_infty:=Hermitian_1PointDivisor(C,infinity);
<GZ divisor with support of length 1 over indeterminate Y>
gap>
gap> fr:=FrobeniusAutomorphismOfHermitian_Curve(C);
AC_FrobeniusAutomorphism(5^3)
gap> P4:=Sum(AC_FrobeniusAutomorphismOrbit(fr,RandomPlaceOfHermitian_Curve(C,4)));
<GZ divisor with support of length 4 over indeterminate Y>
gap> G:=5*P4+7*P_infty;
<GZ divisor with support of length 5 over indeterminate Y>
gap> Degree(G);
27
gap>
gap> len:=90;
90
gap> D:=Sum([1..len],i->Hermitian_1PointDivisor(C,Elements(GF(q))[i]));
<GZ divisor with support of length 90 over indeterminate Y>
gap>
gap> # construct the AG differential code
gap> agcode:=Hermitian_DifferentialCode(G,D);
<[90,62] Hermitian AG-code over GF(5^3)>
gap> DesignedMinimumDistance(agcode);
29

```

```

gap> Length(agcode)-Degree(G)-1;
62
gap>
gap> # test codeword generation
gap> t:=Int((DesignedMinimumDistance(agcode)-1)/2);
14
gap> sent:=Random(agcode);;
gap> err:=RandomVectorOfGivenWeight(GF(q),Length(agcode),t);;
gap> received:=sent+err;;
gap>
gap> # decoding
gap> sent_decoded:=Hermitian_DecomposeToCodeword(agcode,received);
<cvec over GF(5,3) of length 90>
gap> sent=sent_decoded;
true

```

3.5 Utilities for Hermitian AG-codes

3.5.1 RestrictVectorSpace

▷ `RestrictVectorSpace(V, F)` (function)

Let K be a field and V a linear subspace of K^n . The restriction of V to the field F is the intersection $V \cap F^n$.

3.5.2 UPolCoeffsToSmallFieldNC

▷ `UPolCoeffsToSmallFieldNC(f, q)` (function)

This non-checking function returns the same polynomial as f , making sure that the coefficients are in $GF(q)$.

3.5.3 RandomVectorOfGivenWeight

▷ `RandomVectorOfGivenWeight(F, n, k)` (function)

returns a random vector of F^n of Hamming weight k . ▷ `RandomVectorOfGivenDensity(F, n, delta)` (function)

returns a random vector of F^n in which the density of nonzero elements is approximatively δ .
▷ `RandomBinaryVectorOfGivenWeight(n, k)` (function)

returns a random vector of $GF(2)^n$ of Hamming weight k .
▷ `RandomBinaryVectorOfGivenDensity(n, delta)` (function)

returns a random vector of $GF(2)^n$ in which the density of nonzero elements is approximatively δ .

Chapter 4

An example: BCH codes as Hermitian AG-codes

The following example constructs BCH codes as Hermitian AG-codes.

Example

```
gap> my_BCH:=function(n,l,delta,F)
>   local q,m,r,s,beta,Y,C,D_beta,P_0,P_infty,agcode;
>   #
>   q:=Size(F);
>   m:=OrderMod(q,n);
>   beta:=Z(q^m)^((q^m-1)/n);
>   #
>   Y:=Indeterminate(F,"Y");
>   C:=Hermitian_Curve(GF(q^m),Y);
>   D_beta:=Sum([0..n-1],i->Hermitian_1PointDivisor(C,beta^i));
>   P_0:=Hermitian_1PointDivisor(C,0);
>   P_infty:=Hermitian_1PointDivisor(C,infinity);
>   #
>   r:=l-1;
>   s:=n+1-delta-1;
>   agcode:=Hermitian_FunctionalCode(r*P_0+s*P_infty,D_beta);
>   #
>   return RestrictVectorSpace(agcode,F);
> end;
function( n, l, delta, F ) ... end
gap>
gap> ###
gap>
gap> q:=2;
2
gap> n:=35;
35
gap> l:=1;
1
gap> delta:=5;
5
gap>
gap>
gap> C0:=BCHCode(n,l,delta,GF(q)); time;
```



```

a cyclic [35,11,5]8..13 BCH code, delta=5, b=1 over GF(2)
24
gap> C1:=my_BCH(n,l,delta,GF(q)); time;
<vector space over GF(2), with 11 generators>
364
gap>
gap> Collected(List(C0,x->Number(x,y->IsOne(y))));
[ [ 0, 1 ], [ 5, 7 ], [ 7, 5 ], [ 10, 56 ], [ 13, 105 ], [ 14, 10 ],
  [ 15, 105 ], [ 16, 385 ], [ 17, 350 ], [ 18, 350 ], [ 19, 385 ],
  [ 20, 105 ], [ 21, 10 ], [ 22, 105 ], [ 25, 56 ], [ 28, 5 ],
  [ 30, 7 ], [ 35, 1 ] ]
gap> Collected(List(C1,x->Number(x,y->IsOne(y))));
[ [ 0, 1 ], [ 5, 7 ], [ 7, 5 ], [ 10, 56 ], [ 13, 105 ], [ 14, 10 ],
  [ 15, 105 ], [ 16, 385 ], [ 17, 350 ], [ 18, 350 ], [ 19, 385 ],
  [ 20, 105 ], [ 21, 10 ], [ 22, 105 ], [ 25, 56 ], [ 28, 5 ],
  [ 30, 7 ], [ 35, 1 ] ]
gap>
gap> SetDesignedMinimumDistance(C1,delta);
gap> DesignedMinimumDistance(C1);
5

```

References

- [GAP17] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.8.8*, 2017. [6](#)
- [HKT08] J. W. P. Hirschfeld, G. Korchmáros, and F. Torres. *Algebraic curves over a finite field*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ, 2008. [6](#)
- [Sti09] Henning Stichtenoth. *Algebraic function fields and codes*, volume 254 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, second edition, 2009. [6](#)

Index

AutomorphismGroup, 9
DesignedMinimumDistance, 13
FrobeniusAutomorphismOfHermitian_
Curve, 8
GeneratorMatrixOfFunctionalHermitian_
CodeNC, 13
GeneratorMatrixOfHermitian_Code, 13
HERmitian package, 4
Hermitian_Curve, 7
Hermitian_CurveAutomorphism, 8
Hermitian_DecomposeToCodeword, 14
Hermitian_DifferentialCode, 13
Hermitian_Divisor, 9
Hermitian_DivisorConstruct, 9
Hermitian_FunctionalCode, 13
Hermitian_Place, 9
Hermitian_RiemannRochSpaceBasis, 12
IndeterminatesOfHermitian_Curve, 7
InfimumHermitian_Divisor, 10
IsHermitian_Code, 12
IsHermitian_Curve, 7
IsHermitian_CurveAutomorphism, 8
IsHermitian_DifferentialCode, 12
IsHermitian_Divisor, 9
IsHermitian_FunctionalCode, 12
IsRationalHermitian_Divisor, 10
Length, 13
License, 2
MatrixGroupToHermitian_CurveAutGroup, 8
NegativePartOfHermitian_Divisor, 11
PositivePartOfHermitian_Divisor, 10
PrincipalHermitian_Divisor, 10
RandomBinaryVectorOfGivenDensity, 15
RandomBinaryVectorOfGivenWeight, 15
RandomPlaceOfGivenDegreeOfHermitian_
Curve, 8
RandomVectorOfGivenDensity, 15
RandomVectorOfGivenWeight, 15
RestrictVectorSpace, 15
Support, 10
SupremumHermitian_Divisor, 10
UnderlyingField, 7, 10
UPolCoeffsToSmallFieldNC, 15
Valuation, 10
ZeroHermitian_Divisor, 9