ALNUTH

ALgebraic NUmber THeory

and an interface to the

KANT System

Gap code is written by:

Björn Assmann, Andreas Distler, Bettina Eick

Institut Computational Mathematics

Fachbereich Mathematik und Informatik

Technische Universität Braunschweig

Pockelsstr. 14, D-38106 Braunschweig

`b.assmann@tu-bs.de, a.distler@tu-bs.de, b.eick@tu-bs.de`

Kant part

This package incorporates an interface to some functions of the computer algebra system Kant. Kant is developed by Michael Pohst and his group at the Technische Universität Berlin. The Kant system itself is not part of this interface. It can be obtained at

`www.math.tu-berlin.de/~kant/kash.html`

Contents

1 Introduction **3**

2 Methods for number fields **4**

2.1 Creation of number fields . . . 4

2.2 Methods for number fields . . . 4

2.3 Presentations of multiplicative subgroups . . . 5

2.4 Methods to compute with subgroups of the unit group . . . 6

2.5 Factorisation of polynomials over a number field . . . 6

2.6 Examples . . . 6

3 An example application **7**

3.1 Number fields defined by matrices . . . 7

3.2 Number fields defined by a polynomial . . . 8

4 Installation **10**

4.1 Getting and installing KASH . . . 10

4.2 Installing this package . . . 10

4.3 Adjust the path of the executable for KASH . . . 11

4.4 Loading and testing the package . . . 11

Bibliography **12**

Index **13**

1 Introduction

A number field is a finite extension of the field of rational numbers. This package provides various methods to compute with number fields which are given by a defining polynomial or by generators. For background on number fields we refer to [ST79].

Some of the methods provided in this package are written in Gap code. The other part of the methods is imported from the Computer Algebra System KANT [DCK+97]. Hence this package contains some Gap functions and an interface to some functions in the computer algebra system KANT. Therefore one has to have installed KANT to use the full functionality of this package. Furthermore the interface only runs with the Linux version of Gap.

We note that not all available functions of KANT are linked to Gap and the KANT system provides much more methods for computations in number fields.

The main methods included in this package are: creating a number field, computing its maximal order (using KANT), computing its unit group (using KANT) and a presentation of this unit group, computing the elements of a given norm of the number field (using KANT) and determining a presentation for a finitely generated multiplicative subgroup (using KANT). For background on algorithms for number fields we refer to [Poh93], [PZ89] and [Coh93].

The functions provided by this package are introduced in the following chapter. Then an example application is outlined. In the final chapter of this manual the installation of the package is described. We note that the computer algebra system KANT itself is not included in the package.

2 Methods for number fields

An algebraic number field is a finite-dimensional extension of the rational numbers $\mathbb{Q}$. Such a number field has a primitive element and it can be defined by the minimal polynomial of this primitive element. Another important way to define an algebraic number field is by a set of rational matrices which generate a number field.

2.1 Creation of number fields

We provide functions to create number fields defined by rational matrices or by rational polynomials.

1 ► `FieldByMatricesNC(` *matrices* `)`
► `FieldByMatrices(` *matrices* `)`

Creates a field generated by the rational matrices *matrices*. In the faster NC version, the function assumes that the input generates a field and there are no checks on this performed.

2 ► `FieldByMatrixBasisNC(` *matrices* `)`
► `FieldByMatrixBasis(` *matrices* `)`

Creates a field with basis *matrices*. The list *matrices* must consist of rational matrices which form a basis for a number field. In the faster NC version, the function assumes that the input is a matrix basis for a field and no checks are performed.

3 ► `FieldByPolynomialNC(` *polynomial* `)`
► `FieldByPolynomial(` *polynomial* `)`

Creates a field defined by *polynomial*. The polynomial *polynomial* must be an irreducible rational polynomial. In the faster NC version, no checks on the input are performed.

2.2 Methods for number fields

We outline a number of functions for number fields.

1 ► `PrimitiveElement(` F `)`
► `DefiningPolynomial(` F `)`

Computes a primitive element and a defining polynomial for the given number field. The defining polynomial is the minimal polynomial of the primitive element. Since F contains various primitive elements, `PrimitiveElement` tries to find a primitive element which has a minimal polynomial with small coefficients. Via the global variable *PRIM_TEST* the user can decide how many primitive elements will be compared. The default value is 20.

2 ► `IsPrimitiveElement(` F`,` a `)`

Checks if the given element generates the field.

3 ► `DegreeOverPrimeField(` F `)`

Returns the degree of F over the rationals.

4 ► `EquationOrderBasis(` *F* `)`
► `MaximalOrderBasis(` *F* `)`
► `IsIntegerOfNumberField(` *F*`,` *k* `)`

These functions return bases for the equation order or the maximal order of the number field F. Also, they allow to check if a given element is an integer in the given number field.

5 ► `UnitGroup(` *F* `)`
► `IsomorphismPcpGroup(` *U* `)`
► `IsUnitOfNumberField(` *F*`,` *k* `)`

These functions determine the unit group of F and an isomorphism to a pcp group. (Recall that the unit group of F is a finitely generated abelian group.) The isomorphism can be used for various computations with the unit group. Also, the last function allows to check whether a given element is a unit in F.

6 ► `ExponentsOfUnits(` *F*`,` *elms* `)`

This function determines the exponent vectors of the elements in *elms* with respect to the generators of the unit group of F. If the unit group of F is not known, then the function computes this unit group also.

7 ► `IsCyclotomicField(` *F* `)`

Check whether F is cyclotomic.

8 ► `NormCosetsOfNumberField(` *F*`,` *norm* `)`

Returns a description for the set of all elements of norm *norm* in F. These elements can be written as a finite union of cosets of the unit group of F. The function returns coset representatives for these cosets.

2.3 Presentations of multiplicative subgroups

Suppose that a finite number of invertible elements of a number field are given. Then these elements generate a finitely generated abelian group. However, it is a non-trivial task to provide a presentation for this abelian group. The most useful representation for such groups is as pcp group.

1 ► `PcpPresentationOfMultiplicativeSubgroup(` *F*`,` *elms* `)`
► `IsomorphismPcpGroup(` *F*`,` *elms* `)`

Determine a pcp presentation for the multiplicative group of $F\backslash\{0\}$ generated by *elms* and an isomorphism on this presentation. Note, that the method `IsomorphismPcpGroup` is defined in the Polycyclic package [EN00]. We refer to the manual of this package for further background.

2 ► `Kernel(` *map* `)`
► `ImagesSet(` *map*`,` *fieldelms* `)`
► `ImageElm(` *map*`,` *fieldelm* `)`
► `PreImagesRepresentative(` *map*`,` *pcpelm* `)`

These functions can be used to compute with an isomorphism to a pcp presented image. If *fieldelm* is not contained in the source of *map*, then the function `ImageElm` returns fail.

In the determination of the Pcp-presentation of a multiplicative subgroup generated by *elms* the relations between the elements in *elms* play an important role. Let $elms = \{e_1, \ldots, e_l\}$ be a finite subset of a field F. The relation lattice for *elms* is

$$rl(elms) := \left\{(h_1, \ldots, h_l) \in \mathbb{Z}^l | e_1^{h_1} \cdots e_l^{h_l} = 1\right\}.$$

3 ► `RelationLattice(` *F*`,` *elms* `)`

Determines a generating set for the relation lattice of the field elements *elms*.

2.4 Methods to compute with subgroups of the unit group

1 ▶ `RelationLatticeOfUnits(` F `,` *elms* `)`

Determines a basis for the relation lattice of the units *elms* in triangularized form. Note that this method is more efficient than the method `RelationLattice`.

2 ▶ `IntersectionOfUnitSubgroups(` F `,` *gen1* `,` *gen2* `)`

The lists *gen1* and *gen2* are supposed to generate two subgroups U_1 and U_2 of the unit group of F. This function determines the intersection of U_1 with U_2. The result is returned as a list of vectors generating the lattice $\{e \in \mathbb{Z}^n \mid g_1^{e_1} \cdots g_n^{e_n} \in U_2\}$ for $gen1 = [g_1, \ldots, g_n]$.

This function does not check the input for efficiency reasons and it may return wrong results if the input generators do not fulfil the requirements.

2.5 Factorisation of polynomials over a number field

1 ▶ `FactorsPolynomialKant(` *pol* `,` F `)`

Factors *pol*, a polynomial with coefficients in the rationals, over the number field F, which have to be constructed by `FieldByPolynomial` or may be the rationals itself. In the coefficients of the factors, a denotes the primitive Element one can obtain from `PrimitiveElement(`F`)`, i. e. a root of the defining Polynomial of F.

```
gap> x := Indeterminate( Rationals );
x_1
gap> pol := 2*x^7+2*x^5+8*x^4+8*x^2;
2*x_1^7+2*x_1^5+8*x_1^4+8*x_1^2
gap> L := FieldByPolynomial(  x^3-4 );
<field in characteristic 0>
gap> FactorsPolynomialKant( pol, L );
[ !2*x_1, x_1, x_1+(a), x_1^2+(-1*a)*x_1+(a^2), x_1^2+!1 ]
```

2.6 Examples

1 ▶ `ExampleMatField(` l `)`

This function returns some examples of fields generated by matrices. There are 9 such example fields provided and they can be obtained by assigning the input l to an integer between 1 and 9. Some of the properties of the examples are summarized in the following table.

	degree over Q	number of generators	dimension of generators
`ExampleMatField(1)`	4	4	4
`ExampleMatField(2)`	4	4	4
`ExampleMatField(3)`	4	4	4
`ExampleMatField(4)`	4	13	4
`ExampleMatField(5)`	4	13	4
`ExampleMatField(6)`	4	7	4
`ExampleMatField(7)`	4	18	4
`ExampleMatField(8)`	4	13	4
`ExampleMatField(9)`	4	7	4

3 An example application

In this section we outline two example computations with the functions of the previous chapter. The first example uses number fields defined by matrices and the second example considers number fields defined by a polynomial.

3.1 Number fields defined by matrices

```
gap> m1 := [ [ 1, 0, 0, -7 ],
             [ 7, 1, 0, -7 ],
             [ 0, 7, 1, -7 ],
             [ 0, 0, 7, -6 ] ];;

gap> m2 := [ [ 0, 0, -13, 14 ],
             [ -1, 0, -13, 1 ],
             [ 13, -1, -13, 1 ],
             [ 0, 13, -14, 1 ] ];;

gap> F := FieldByMatricesNC( [m1, m2] );
<field in characteristic 0>

gap> DegreeOverPrimeField(F);
4
gap> PrimitiveElement(F);
[ [ 1, 0, 0, -7 ], [ 7, 1, 0, -7 ], [ 0, 7, 1, -7 ], [ 0, 0, 7, -6 ] ]

gap> Basis(F);
Basis( <field in characteristic 0>,
[ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 0, 1, 0, 0 ], [ -1, 1, 1, 0 ], [ -1, 0, 1, 1 ], [ -1, 0, 0, 1 ] ],
  [ [ 0, 0, 1, 0 ], [ -1, 0, 1, 1 ], [ -1, -1, 1, 1 ], [ 0, -1, 0, 1 ] ],
  [ [ 0, 0, 0, 1 ], [ -1, 0, 0, 1 ], [ 0, -1, 0, 1 ], [ 0, 0, -1, 1 ] ] ] )

gap> MaximalOrderBasis(F);
Basis( <field in characteristic 0>,
[ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ 1, 0, 0, -1 ], [ 1, 1, 0, -1 ], [ 0, 1, 1, -1 ], [ 0, 0, 1, 0 ] ],
  [ [ 1, 0, -1, 0 ], [ 1, 1, -1, -1 ], [ 1, 1, 0, -1 ], [ 0, 1, 0, 0 ] ],
  [ [ 1, -1, 0, 0 ], [ 1, 0, -1, 0 ], [ 1, 0, 0, -1 ], [ 1, 0, 0, 0 ] ] ] )

gap> U := UnitGroup(F);
<matrix group with 2 generators>

gap> u := GeneratorsOfGroup( U );;
```

```
gap> nat := IsomorphismPcpGroup(U);
[ [ [ 0, 1, -1, 0 ], [ 0, 1, 0, -1 ], [ 0, 1, 0, 0 ], [ -1, 1, 0, 0 ] ],
  [ [ 1, 0, -1, 1 ], [ 0, 1, -1, 0 ], [ 1, 0, 0, 0 ], [ 0, 1, -1, 1 ] ] ] ->
[ g1, g2 ]

gap> H := Image(nat);
Pcp-group with orders [ 10, 0 ]
gap> ImageElm( nat, u[1]*u[2] );
g1*g2
gap> PreImagesRepresentative(nat, GeneratorsOfGroup(H)[1] );
[ [ 0, 1, -1, 0 ], [ 0, 1, 0, -1 ], [ 0, 1, 0, 0 ], [ -1, 1, 0, 0 ] ]
```

3.2 Number fields defined by a polynomial

```
gap> x:=Indeterminate(Rationals);
x_1
gap> g:= x^4-4*x^3-28*x^2+64*x+16;
x_1^4-4*x_1^3-28*x_1^2+64*x_1+16

gap> F := FieldByPolynomialNC(g);
<field in characteristic 0>
gap> PrimitiveElement(F);
(a)
gap> MaximalOrderBasis(F);
Basis( <field in characteristic 0>,
[ !1, (1/2*a), (1/4*a^2), (5/7+1/14*a+1/14*a^2+1/56*a^3) ] )

gap> U := UnitGroup(F);
[ !-1, (-3/7+6/7*a+3/28*a^2-1/28*a^3),
  (13/7+25/14*a+1/28*a^2-3/56*a^3), (36/7-9/7*a-2/7*a^2+3/56*a^3) ]
<group with 4 generators>

gap> natU := IsomorphismPcpGroup(U);
[ !-1, (-3/7+6/7*a+3/28*a^2-1/28*a^3),
  (13/7+25/14*a+1/28*a^2-3/56*a^3), (36/7-9/7*a-2/7*a^2+3/56*a^3)
 ] -> [ g1, g2, g3, g4 ]

gap> elms := List( [1..10], x-> Random(F) );
[ (4-1/2*a-1*a^2+3/2*a^3), (4/5-2/3*a+4/3*a^3), (1+a+2*a^2-1*a^3),
  (3/4+3*a+3*a^2), (-1-1/5*a^3), (-1/4*a-5/3*a^2), (1-1*a+1/2*a^2),
  (4-3/2*a+1/2*a^2), (-2/5+a-3/2*a^2), (-1*a+a^2+2*a^3) ]

gap>  PcpPresentationOfMultiplicativeSubgroup( F, elms );
Pcp-group with orders [ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ]

gap>isom := IsomorphismPcpGroup( F, elms );
[ (4-1/2*a-1*a^2+3/2*a^3), (4/5-2/3*a+4/3*a^3),
  (1+a+2*a^2-1*a^3), (3/4+3*a+3*a^2), (-1-1/5*a^3),
  (-1/4*a-5/3*a^2), (1-1*a+1/2*a^2), (4-3/2*a+1/2*a^2),
  (-2/5+a-3/2*a^2), (-1*a+a^2+2*a^3) ]
[ (4-1/2*a-1*a^2+3/2*a^3), (4/5-2/3*a+4/3*a^3), (1+a+2*a^2-1*a^3),
```

```
    (3/4+3*a+3*a^2), (-1-1/5*a^3), (-1/4*a-5/3*a^2), (1-1*a+1/2*a^2),
    (4-3/2*a+1/2*a^2), (-2/5+a-3/2*a^2), (-1*a+a^2+2*a^3) ] ->
  [ g1, g2, g3, g4, g5, g6, g7, g8, g9, g10 ]

gap> y := RandomGroupElement( elms );
(-4757097249767070313713 25/7180632878818977576795 2976
-3795846412612995922398 25/1305569614330723195780 9632*a
-4622491885705937713775 95/2872253151527591030718 11904*a^2+
2639763613873579813685/29012658096238293239576 96*a^3)

gap> ImageElm( isom, y );
g1^-1*g3^-2*g6^2*g8^-1*g9^-1
gap> z := last;
g1^-1*g3^-2*g6^2*g8^-1*g9^-1

gap> PreImagesRepresentative( isom, z );
(-4757097249767070313713 25/7180632878818977576795 2976
-3795846412612995922398 25/1305569614330723195780 9632*a
-4622491885705937713775 95/2872253151527591030718 11904*a^2+
2639763613873579813685/29012658096238293239576 96*a^3)

gap> FactorsPolynomialKant( g, F );
[ x_1+(-40/7+31/7*a+3/7*a^2-1/7*a^3), x_1+(-2+a), x_1+(-1*a),
  x_1+(26/7-31/7*a-3/7*a^2+1/7*a^3) ]
```

4 Installation

This package provides an interface between GAP 4 and KANT respectively KASH, the shell of the computational algebraic number theory system KANT. By now the interface can only be used on a Linux system. KASH itself is not part of this package. It has to be obtained and installed independently of this package. Alnuth works with KASH version 2.4 or higher.

4.1 Getting and installing KASH

KASH is available at

`www.math.tu-berlin.de/~kant/download.html`

Note that you have to download two files for a complete installation of KASH. For the installation of version 2.5 of KASH on a Linux system you would do the following steps:

1. Download the files kash_2.5.common.tar.gz and kash_2.5.3.linux.tar.gz into the same directory on your system.
2. Unpack the files using tar. This will create a directory KASH_2.5 containing among other files the KASH executable called kash.

The place where KASH is located in your system is independent of the place where the Alnuth-package is installed.

4.2 Installing this package

This package is available at

`http://www.icm.tu-bs.de/ag_algebra/software/assmann/Alnuth`

in form of a gzipped tar-archive or as an uncompressed tar-archive.

There are two ways of installing the package. If you have permission to add files to the installation of GAP 4 on your system you may install the Alnuth-package into the pkg subdirectory of the GAP installation tree. If you do not have the permission to do that you may install the Alnuth-package in your private area. In the latter case you need to have a directory named pkg in your private area (for details see 74.1 in the reference manual).

Now move the alnuth.tar.gz of alnuth.tar file into the directory pkg and unpack it:

```
bash> tar xfz alnuth.tar.gz        # for the gzipped tar-archive
bash> tar xf alnuth.tar         # for the uncompressed tar-archive
```

4.3 Adjust the path of the executable for KASH

The package needs to know where the executable for KASH is. Again there are several possibilities. If you are able to edit the file pkg/alnuth/defs.g you can change the line

```
BindGlobal( "KANTEXEC", fail );
```

to something like

```
BindGlobal( "KANTEXEC", "mykash/kash -l mykash/lib" );
```

where `mykash` needs to be replaced with the directory where kash was installed.

Like always you can also change your personal `.gaprc` file (see 3.4) for setting the variable KANTEXEC to a proper value. To do this add the command line mentioned above to `.gaprc`.

The other alternative is to change the path to the executable within GAP using one of the following two functions. To do this you first have to load the package (see 4.4). To use

1 ► `SetKantExecutablePermanently(` *path* `)`

where the string *path* should be the filename of the executable for KASH, you need to be allowed to overwrite the file pkg/alnuth/defs.g to make this work. It changes the file pkg/alnuth/defs.g as described above.

2 ► `SetKantExecutable(` *path* `)`

changes the filename of the executable for KASH to the string *path* for the current session.

Both functions run a test wether *path* is a valid string for a filename of an executable for KASH version 2.4 or higher.

4.4 Loading and testing the package

To use this package you have to request it explicitly. This is done by calling

```
gap> LoadingPackage("alnuth");
Loading Alnuth 1.0 ...
true
gap>
```

Once the package is loaded, it is possible to check the correct installation by running the test suite of the package.

```
gap> Read( "DIR/pkg/alnuth/tst/testall.g" );
```

where DIR needs to be replaced with the directory, in which pkg is located.

Bibliography

[Coh93] Henri Cohen. *A course in computational algebraic number theory*. Springer-Verlag, New York, Heidelberg, Berlin, 1993.

[DCK+97] M. Daberkow, C.Fieker, J. Klüners, M. Pohst, K.Roegner, and K. Wildanger. Kant V4. *J. Symb. Comput.*, 24:267 – 283, 1997.

[EN00] Bettina Eick and Werner Nickel. *Polycyclic*, 2000. A GAP 4 package, see [GAP02].

[GAP02] The GAP Group. *GAP – Groups, Algorithms, and Programming, Version 4.3*, 2002.
`http://www.gap-system.org`.

[Poh93] Michael E. Pohst. *Computational Algebraic Number Theory*, volume 21 of *DMV Seminar*. Birkäuser, 1993.

[PZ89] M. Pohst and H. Zassenhaus. *Algorithmic algebraic number theory*. Cambridge University Press, 1989.

[ST79] I. N. Stuart and D. O. Tall. *Algebraic number theory*. Chapman and Hall, 1979.

Index

This index covers only this manual. A page number in *italics* refers to a whole section which is devoted to the indexed subject. Keywords are sorted with case and spaces ignored, e.g., "`PermutationCharacter`" comes before "permutation group".

A

Adjust the path of the executable for KASH, *10*

C

Creation of number fields, *4*

D

`DefiningPolynomial`, 4
`DegreeOverPrimeField`, 4

E

`EquationOrderBasis`, 4
`ExampleMatField`, 6
Examples, *6*
`ExponentsOfUnits`, 5

F

Factorisation of polynomials over a number field, *6*
`FactorsPolynomialKant`, 6
`FieldByMatrices`, 4
`FieldByMatricesNC`, 4
`FieldByMatrixBasis`, 4
`FieldByMatrixBasisNC`, 4
`FieldByPolynomial`, 4
`FieldByPolynomialNC`, 4

G

Getting and installing KASH, *10*

I

`ImageElm`, 5
`ImagesSet`, 5
Installing this package, *10*
`IntersectionOfUnitSubgroups`, 6
`IsCyclotomicField`, 5
`IsIntegerOfNumberField`, 4
`IsomorphismPcpGroup`, 5
`IsPrimitiveElement`, 4
`IsUnitOfNumberField`, 5

K

`Kernel`, 5

L

Loading and testing the package, *11*

M

`MaximalOrderBasis`, 4
Methods for number fields, *4*
Methods to compute with subgroups of the unit group, *5*

N

`NormCosetsOfNumberField`, 5
Number fields defined by a polynomial, *8*
Number fields defined by matrices, *7*

P

`PcpPresentationOfMultiplicativeSubgroup`, 5
`PreImagesRepresentative`, 5
Presentations of multiplicative subgroups, *5*
`PrimitiveElement`, 4

R

`RelationLattice`, 5
`RelationLatticeOfUnits`, 5

S

`SetKantExecutable`, 11
`SetKantExecutablePermanently`, 11

U

`UnitGroup`, 5