ALNUTH

Version 3.1.0, December 2017

ALgebraic NUmber THeory

and an interface to

PARI/GP

GAP code written by:

Björn Assmann

`http://www.dcs.st-and.ac.uk/~bjoern`

Andreas Distler

CAUL (Centro de Álgebra da Universidade de Lisboa)

Av. Prof. Gama Pinto, 2; 1649-003 Lisboa; Portugal

`a.distler@tu-bs.de`

Bettina Eick

AG Algebra und Diskrete Mathematik; Institut Computational Mathematics; TU Braunschweig

Pockelsstr. 14; D-38106 Braunschweig; Germany

`b.eick@tu-bs.de`

GP code written by:

Bill Allombert

Note: PARI/GP is **not** part of this package. It can be obtained from `http://pari.math.u-bordeaux.fr/`

Contents

1 Introduction

A number field is a finite extension of the field of rational numbers. Alnuth 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 PARI/GP [PAR11]. Hence this package contains some GAP functions and an interface to some functions in the computer algebra system PARI/GP. Therefore one has to have PARI/GP installed to use the full functionality of Alnuth.

We note that only a very small part of the functions available in PARI/GP are linked to GAP and PARI/GP provides many more methods for computations in number fields.

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

The functions provided by Alnuth are introduced in the following chapter. Then an example application is outlined. In the final chapter of this manual the installation of the package and configuration of the interface, including hints on the installation of PARI/GP, are described.

1.1 Acknowledgements

To begin with we are very grateful for all the feedback by users of former versions of Alnuth.

We thank Bill Allombert who wrote the GP code for the interface to PARI/GP and who was extremely helpful in the transition from KANT to PARI/GP.

For feedback on the development version, including a code patch, we are much obliged to Max Horn.

The second author acknowledges the financial support at CAUL within the projects PTDC/MAT/101993/2008 and ISFL-1-143, financed by FEDER and FCT, in the development of Version 3 of Alnuth.

1.2 License

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 ► `IsPrimitiveElementOfNumberField(` *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* `)`

determines the unit group of F.

Recall that the unit group of F is a finitely generated abelian group. The function `IsomorphismPcpGroup` from the Polycyclic [EHN11] package gives an isomorphism to a pcp group which can be used for various computations with the unit group.

6 ► `IsUnitOfNumberField(` *F*, *k* `)`

checks whether the element k is a unit in F.

7 ► `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.

8 ► `IsCyclotomicField(` *F* `)`

Check whether F is cyclotomic.

9 ► `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 [EHN11]. We refer to the manual of this package for further background.

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

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

2 ► `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 $gen1 = [g_1, \ldots, g_n]$.

For efficiency reasons this function does not check the input 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 ▶ `FactorsPolynomialAlgExt(` *F*, *pol* `)`

embeds the rational polynomial *pol* into the polynomial ring over the number field *F*, which has to be constructed by `FieldByPolynomial` or `AlgebraicExtension`, and returns the factorization of the embedded polynomial. By default *a* denotes the primitive element of the field one can obtain from `PrimitiveElement(`*F*`)`, that is, a root of the defining polynomial of *F*.

2 ▶ `FactorsPolynomialPari(` *pol* `)`

takes a polynomial *pol* defined over an algebraic extension of the Rationals and factors it using PARI/GP.

```
gap> x := Indeterminate( Rationals, "x" );;
gap> pol := 2*x^7+2*x^5+8*x^4+8*x^2;
2*x^7+2*x^5+8*x^4+8*x^2
gap> L := FieldByPolynomial( x^3-4 );
<algebraic extension over the Rationals of degree 3>
gap> y := Indeterminate( L, "y" );;
gap> FactorsPolynomialAlgExt( L, pol );
[ !2*y, y, y+(a), y^2+!1, y^2+((-1*a))*y+(a^2) ]
gap> FactorsPolynomialPari( last[5] );
[ y^2+((-1*a))*y+(a^2) ]
gap>
```

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	dim. 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] );
<rational matrix field of unknown degree>

gap> DegreeOverPrimeField(F);
4
gap> PrimitiveElement(F);
[ [ -1, 1, 1, 0 ], [ -2, 0, 2, 1 ], [ -2, -1, 1, 2 ], [ -1, -1, 0, 1 ] ]

gap> Basis(F);
Basis( <rational matrix field of degree 4>,
[ [ [ 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( <rational matrix field of degree 4>,
[ [ [ 1, 0, 0, 0 ], [ 0, 1, 0, 0 ], [ 0, 0, 1, 0 ], [ 0, 0, 0, 1 ] ],
  [ [ -1, 1, 1, 0 ], [ -2, 0, 2, 1 ], [ -2, -1, 1, 2 ], [ -1, -1, 0, 1 ] ],
  [ [ -3, -2, 2, 3 ], [ -3, -5, 0, 5 ], [ 0, -5, -3, 3 ], [ 2, -2, -3, 0 ] ],
  [ [ -1, -1, 0, 1 ], [ 0, -2, -1, 1 ], [ 1, -1, -2, 0 ], [ 1, 0, -1, -1 ] ]
 ] )

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

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

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

3.2 Number fields defined by a polynomial

```
gap> g := UnivariatePolynomial( Rationals, [ 16, 64, -28, -4, 1 ] );
x_1^4-4*x_1^3-28*x_1^2+64*x_1+16

gap> F := FieldByPolynomialNC(g);
<algebraic extension over the Rationals of degree 4>
gap> PrimitiveElement(F);
a
gap> MaximalOrderBasis(F);
Basis( <algebraic extension over the Rationals of degree 4>,
[ !1, 1/2*a, 1/4*a^2, 1/56*a^3+1/14*a^2+1/14*a-2/7 ] )

gap> U := UnitGroup(F);
<group with 4 generators>

gap> natU := IsomorphismPcpGroup(U);;
gap> elms := List( [1..10], x-> Random(F) );;

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

gap> isom := IsomorphismPcpGroup( F, elms );;
gap> y := RandomGroupElement( elms );;
gap> z := ImageElm( isom, y );;
gap> y = PreImagesRepresentative( isom, z );
true

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

4 Installation

This package provides an interface between GAP and PARI/GP. PARI/GP itself is not part of this package. It has to be obtained and installed independently of this package. Alnuth works with PARI/GP Version 2.5 or higher.

4.1 Installing Alnuth

The package Alnuth is part of the standard distribution of GAP so that in most cases there is no need to install it separately. To use Alnuth you need to have PARI/GP installed. See the following section for information on PARI/GP.

In case you want to update Alnuth independently of your main GAP installation or if you are interested in an old version of Alnuth interfacing to KANT/KASH you can find all released versions of Alnuth at the package's homepage

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

in the form of gzipped tar-archives or as uncompressed tar-archives, and starting with Version 3.0.0 also as zip archives for Windows. The main page always contains the latest version of Alnuth, while older versions can be found under `Archives`.

There are two ways of installing a GAP package. If you have permission to add files to the installation of GAP on your system you may install Alnuth into the `pkg` subdirectory of the GAP installation tree. Otherwise you may install Alnuth in a private `pkg` directory (for details see Subsections 76.1 and 9.2 in the GAP reference manual).

To install the latest version of Alnuth under Linux download one of the archives `alnuth.tar` and `alnuth.tar.gz`, move it to the directory `pkg` in which you want to install, and unpack the archive. If you are using the command line you can unpack the former with the command `tar xf alnuth.tar`, respectively `tar xzf alnuth.tar.gz` for the gzipped tar-archive.

For installation under Windows download the archive `alnuth-win.zip`, move it to the `pkg` directory in which you want to install, and extract the archive (If you right click on the archive there should appear a menu containing the entry `Extract here`.).

4.2 Getting PARI/GP

Using Alnuth requires an installation of PARI/GP in Version 2.5 or higher. The software PARI/GP is freely available at

`http://pari.math.u-bordeaux.fr/`

Note that the place where PARI/GP is located in your system is independent of the place where Alnuth is installed.

a. Installing under Linux

In many Linux distributions PARI/GP can be installed via the software package manager, but this might sometimes be an older version which cannot be used together with Alnuth. (Starting GP from the command line with the option `--version-short` will show you the version number.)

If you install PARI/GP from source make sure you install with GMP support for better performance and complete the installation with `make install` so that you can start GP by just calling gp from the command line.

b. Installing under Windows

For Windows it is sufficient to get the basic GP binary which can be found at

`http://pari.math.u-bordeaux.fr/download.html`

4.3 Adjust the path of the executable for GP

This package needs to know where the executable for GP is. In the default setting Alnuth looks for an executable program named gp in the search paths of the system. More precisely, for a file gp inside one of the directories in the list returned by `DirectoriesSystemPrograms()` (called in a GAP session).

Under Linux the default setting should work with a standard installation of PARI/GP.

For the default setting to work under Windows the downloaded executable file, for example `gp-2-5-0.exe` has to be renamed to `gp.exe` and moved to one of the directories listed by `DirectoriesSystemPrograms()` (Ignore the leading `cygdrive` in each path name and note that the single letter specifies the drive, for example `/cygdrive/c/Windows/` denotes the folder `Windows` on drive `C:`).

To check whether an executable of GP in Version 2.5 or higher is available with the default setting, you can use the function

1 ► `PariVersion( )`

which prints the verion number, if the global variable `AL_EXECUTABLE` is bound to the file name of a GP executable.

If you cannot use the default setting for you purpose, you have two options to specify a new place and name for the GP executable.

If you are able to edit the file `defs.g` in the main directory of Alnuth, then you can change the line

```
BindGlobal("AL_EXECUTABLE", Filename(DirectoriesSystemPrograms(), "gp"));
```

to something like

```
BindGlobal("AL_EXECUTABLE", "/home/my/pari-2.5.0/gp");
```

under Linux or

```
BindGlobal("AL_EXECUTABLE", "/cygdrive/c/Users/my/Downloads/gp-2-5-0");
```

under Windows, where the second argument gives the path to the GP executable you want to use (Change the strings given above to reflect the actual path on your system.). Please note that in case of a new installation of Alnuth you will have to change `defs.g` again.

Alternatively you can also change your personal `gaprc` file (see the Subsection 3.2.2 in the GAP reference manual) for setting the variable AL_EXECUTABLE to a proper value. To do this add the appropriate line from above with the `BindGlobal` command to `gaprc`.

The third possibility is to change the path to the GP executable from within GAP using one of the following two functions. To do this you first have to load the package (see Section 4.4).

2 ► `SetAlnuthExternalExecutable(` *path* `)`

adjusts the global variable `AL_EXECUTABLE` for the current GAP session. Depending on your installation of PARI/GP and your operating system the string *path* has to be either the command to start GP in a terminal (for example gp) or the complete path to the executable of GP (for example `/cygwindrive/c/windows/gp`). The function returns `fail` if *path* does not execute GP in Version 2.5.0 or higher.

To make the change take effect for all of your GAP sessions, you can add `SetKantExecutable` with the path to the GP executable as argument to your personal `gaprc` file (see the Subsection 3.2.2 in the GAP reference manual).

If you want the change to affect all GAP session of all users of the GAP installation you can use the following function:

3 ► `SetAlnuthExternalExecutablePermanently(` *path* `)`

does the same as `SetAlnuthExternalExecutable` and in addition trys to change the file `defs.g` accordingly in the currently loaded version of Alnuth. A warning is issued if there is no write access to `defs.g` from the current GAP session.

4 ► `RestoreAlnuthExternalExecutable( )`

trys to restore the original content of the file `defs.g` in the directory corresponding to the loaded version of Alnuth. The function returns `fail` if there is no write access to `defs.g` from the current GAP session.

4.4 Loading and testing the package

If Alnuth is not loaded when GAP is started you have to request it explicitly to use it. This is done by calling `LoadPackage("Alnuth");` in a GAP session. If Alnuth had not been loaded already a short banner will be displayed.

```
gap> LoadingPackage("alnuth");
Loading Alnuth 3.0.0 ...
true
gap>
```

To load a certain version of Alnuth you can specify the version number as second argument in the call to `LoadPackage`. (See `77.2 Loading a GAP package` in the reference manual or type `?LoadPackage` within a GAP session).

Once the package is loaded, it is possible to check the correct installation running a short test by calling `ReadPackage("Alnuth", "tst/testinstall.tst");`.

```
gap> ReadPackage("Alnuth", "tst/testinstall.g");
Line 5 :
+ Test availability and version of PARI/GP
Line 5 :
+ GAP4stones: 25000
Line 85 :
+ Installation test of Alnuth package
Line 85 :
+ GAP4stones: 609
true
gap>
```

The number of GAP4stones will usually differ; other discrepancies in the output indicate some problem.

If the test suite runs into an error in the first part, which verifies the availability of PARI/GP, check your installation of PARI/GP and consult the last chapter of the documentation of Alnuth for more information.

If you find any bugs or have any suggestions or comments, we would very much appreciate it if you would let us know by writing to `a.distler(at)tu-bs.de`.

Bibliography

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

[EHN11] Bettina Eick, Max Horn, and Werner Nickel. *Polycyclic - a GAP package*, 2011.
`http://www.icm.tu-bs.de/ag_algebra/software/polycyclic/`.

[PAR11] The PARI Group, Bordeaux. *PARI/GP, version* `2.5.0`, 2011. available from
`http://pari.math.u-bordeaux.fr/`.

[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".