

Alnuth

ALgebraic NUmber THeory and an interface to PARI/GP and OSCAR

4.0.2

21 June 2026

GAP code by Björn Assmann, Andreas Distler and Bettina Eick

PARI/GP code by Bill Allombert

OSCAR code by Claus Fieker and Max Horn

Note: PARI/GP is *not* part of this package. It can be obtained from <https://pari.math.u-bordeaux.fr/>. If you use GAP via OSCAR, then OSCAR will automatically be used instead of PARI/GP.

Copyright

This program 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.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <https://www.gnu.org/licenses/>

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.

Support for using OSCAR instead of PARI/GP was added by Claus Fieker and Max Horn.

Contents

1	Introduction	4
2	Working with number fields	5
2.1	Creation of number fields	5
2.2	Methods for number fields	6
2.3	Presentations of multiplicative subgroups	7
2.4	Methods to compute with subgroups of the unit group	8
2.5	Factorisation of polynomials over a number field	8
2.6	Examples	9
3	An example application	10
3.1	Number fields defined by matrices	10
3.2	Number fields defined by a polynomial	11
4	Installation	13
4.1	Installing Alnuth	13
4.2	Getting PARI/GP	13
4.3	Adjust the path of the executable for GP	14
4.4	Loading and testing the package	16
	References	17
	Index	18

Chapter 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 Systems PARI/GP [PAR11] respectively OSCAR [DEF⁺25], [OSC24]. Hence this package contains some **GAP** functions and an interface to some functions to these computer algebra systems. Therefore one has to have PARI/GP or OSCAR installed to use the full functionality of **Alnuth**.

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

The main methods included in **Alnuth** are: creating a number field, computing its maximal order, computing its unit group and a presentation of this unit group, computing the elements of a given norm of the number field, determining a presentation for a finitely generated multiplicative subgroup, and factoring polynomials defined over number fields. 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 respectively OSCAR, are described.

Chapter 2

Working with 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.

2.1.1 FieldByMatricesNC

- ▷ `FieldByMatricesNC(matrices)` (function)
- ▷ `FieldByMatrices(matrices)` (function)

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.

2.1.2 FieldByMatrixBasisNC

- ▷ `FieldByMatrixBasisNC(matrices)` (function)
- ▷ `FieldByMatrixBasis(matrices)` (function)

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.

2.1.3 FieldByPolynomialNC

- ▷ `FieldByPolynomialNC(polynomial)` (function)
- ▷ `FieldByPolynomial(polynomial)` (function)

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.

2.2.1 PrimitiveElement

- ▷ `PrimitiveElement(F)` (function)
- ▷ `DefiningPolynomial(F)` (function)

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 user preference `PrimitiveElementTrials` the user can decide how many primitive elements will be compared. The default value is 20. (See `SetUserPreference` (**Reference: SetUserPreference**) for more information about user preferences.)

2.2.2 IsPrimitiveElementOfNumberField

- ▷ `IsPrimitiveElementOfNumberField(F , a)` (function)

Checks if the given element generates the field.

2.2.3 DegreeOverPrimeField

- ▷ `DegreeOverPrimeField(F)` (function)

Returns the degree of F over the rationals.

2.2.4 EquationOrderBasis

- ▷ `EquationOrderBasis(F)` (function)
- ▷ `MaximalOrderBasis(F)` (function)
- ▷ `IsIntegerOfNumberField(F , k)` (function)

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.

2.2.5 UnitGroup

- ▷ `UnitGroup(F)` (function)

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.

2.2.6 IsUnitOfNumberField

▷ `IsUnitOfNumberField(F , k)` (function)

checks whether the element k is a unit in F .

2.2.7 ExponentsOfUnits

▷ `ExponentsOfUnits(F , $elems$)` (function)

This function determines the exponent vectors of the elements in $elems$ 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.

2.2.8 IsCyclotomicField

▷ `IsCyclotomicField(F)` (function)

Check whether F is cyclotomic.

2.2.9 NormCosetsOfNumberField

▷ `NormCosetsOfNumberField(F , $norm$)` (function)

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 pcg group.

2.3.1 PcpPresentationOfMultiplicativeSubgroup

▷ `PcpPresentationOfMultiplicativeSubgroup(F , $elems$)` (function)

▷ `IsomorphismPcpGroup(F , $elems$)` (function)

Determine a pcg presentation for the multiplicative group of $F \setminus \{0\}$ generated by $elems$ 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 $elems$ the relations between the elements in $elems$ play an important role. Let $elems = \{e_1, \dots, e_l\}$ be a finite subset of a field F . The relation lattice for $elems$ is

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

2.3.2 RelationLattice

▷ `RelationLattice(F, elms)` (function)

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

2.4 Methods to compute with subgroups of the unit group

2.4.1 RelationLatticeOfUnits

▷ `RelationLatticeOfUnits(F, elms)` (function)

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`.

2.4.2 IntersectionOfUnitSubgroups

▷ `IntersectionOfUnitSubgroups(F, gen1, gen2)` (function)

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 $gen1 = [g_1, \dots, g_n]$.

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

2.5.1 FactorsPolynomialAlgExt

▷ `FactorsPolynomialAlgExt(F, pol)` (function)

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 .

2.5.2 FactorsPolynomialAlnuth

▷ `FactorsPolynomialAlnuth(pol)` (function)

takes a polynomial *pol* defined over an algebraic extension of the Rationals and factors it using the external CAS (PARI/GP or OSCAR).

The alias `FactorsPolynomialPari` is provided for backwards compatibility.

Example

```
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+(-a)*y+a^2 ]
gap> FactorsPolynomialAlnuth( last[5] );
[ y^2+(-a)*y+a^2 ]

```

2.6 Examples

2.6.1 ExampleMatField

▷ ExampleMatField(1)

(function)

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 1 to an integer between 1 and 9. Some of the properties of the examples are summarized in the following table.

	Example		
	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

Chapter 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

Example

```
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);
[ [ 0, -1, 1, 0 ], [ 0, -1, 0, 1 ], [ 0, -1, 0, 0 ], [ 1, -1, 0, 0 ] ]

#
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 ] ],
    [ [ 0, -1, 1, 0 ], [ 0, -1, 0, 1 ], [ 0, -1, 0, 0 ], [ 1, -1, 0, 0 ] ],
    [ [ 0, 0, 0, -1 ], [ 1, 0, 0, -1 ], [ 0, 1, 0, -1 ], [ 0, 0, 1, -1 ] ],
    [ [ -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);;
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

Example

```

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

Chapter 4

Installation

This package provides an interface between **GAP** and either **PARI/GP** or **OSCAR**. When **Alnuth** is loaded in **GAP** running inside **OSCAR**, it automatically uses **OSCAR**. Otherwise, **Alnuth** uses **PARI/GP**, which 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. If you are using **GAP** inside **OSCAR**, then **Alnuth** is already bundled and automatically uses **OSCAR**, so no further installation is required. Otherwise, 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** in the form of gzipped tar-archives at <https://github.com/gap-packages/alnuth/releases>

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 **(Reference: Installing a GAP Package)** and **(Reference: GAP Root Directories)** in the **GAP** reference manual).

To install the latest version of **Alnuth** download one of the archives `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 with the command `tar xzf alnuth.tar.gz`.

4.2 Getting PARI/GP

Using **Alnuth** outside **OSCAR** requires an installation of **PARI/GP** in Version 2.5 or higher. The software **PARI/GP** is freely available at <https://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.

1. *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.

2. Installing under Windows

For Windows it is sufficient to get the basic GP binary which can be found at <https://pari.math.u-bordeaux.fr/download.html>

4.3 Adjust the path of the executable for GP

When Alnuth is used outside OSCAR, it 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

4.3.1 PariVersion

▷ `PariVersion()` (function)

which prints the version number, if the user preference `PariGpPath` refers to a usable GP executable.

If you cannot use the default setting for your purpose, you have several ways to configure Alnuth. The recommended approach is to use the GAP user preference system:

Example

```
gap> SetUserPreference("alnuth", "PariGpPath", "/home/my/pari-2.5.0/gp");
gap> SetUserPreference("alnuth", "PariStackSize", 256);
gap> WriteGapIniFile();
```

This changes the current session immediately. `WriteGapIniFile()` writes the current non-default settings to your personal `gap.ini`, so they are used again in future GAP sessions. See (**Reference: Configuring User preferences**) for background on the user preference system.

The user preferences relevant for Alnuth are shown by `ShowUserPreferences("alnuth");`. The most important ones are `PariGpPath`, `PariGpOptions`, `PariStackSize`, and `PrimitiveElementTrials`.

In older versions of **Alnuth** configuration was done via the global variables `AL_EXECUTABLE`, `AL_PATH`, `AL_OPTIONS`, `AL_STACKSIZE`, and `PRIM_TEST`. These globals are deprecated, and they trigger a warning when used. The globals except for `AL_PATH` still serve as a fallback if the corresponding user preference is unusable. Their replacements are as follows:

`AL_EXECUTABLE`

use the user preference `PariGpPath`

`AL_OPTIONS`

use the user preference `PariGpOptions`

`AL_STACKSIZE`

use the user preference `PariStackSize`

`PRIM_TEST`

use the user preference `PrimitiveElementTrials`

`AL_PATH`

has been removed; **Alnuth** now always uses the bundled GP helper files.

For backwards compatibility, the following functions still exist after loading the package (see Section 4.4), but they are deprecated wrappers around the user preference system.

4.3.2 SetAlnuthExternalExecutable

▷ `SetAlnuthExternalExecutable(path)` (function)

sets the user preference `PariGpPath` for the current **GAP** session. Depending on your installation of **PARI/GP** and your operating system the string `path` can be either the command to start GP in a terminal (for example `gp`) or the complete path to the executable of GP. The function is deprecated; new code should call `SetUserPreference("alnuth", "PariGpPath", path);` directly.

The function returns the stored preference value.

4.3.3 SetAlnuthExternalExecutablePermanently

▷ `SetAlnuthExternalExecutablePermanently(path)` (function)

does the same as `SetAlnuthExternalExecutable` and then calls `WriteGapIniFile()` to persist the current user preference settings in your personal `gap.ini`. The function is deprecated; new code should call `SetUserPreference("alnuth", "PariGpPath", path); WriteGapIniFile();` instead.

4.3.4 RestoreAlnuthExternalExecutablePermanently

▷ `RestoreAlnuthExternalExecutablePermanently()` (function)

restores the default value of `PariGpPath` and then calls `WriteGapIniFile()`. The function is deprecated; new code should set `PariGpPath` to the desired default command or path and then call `WriteGapIniFile()` directly.

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.

Example

```
gap> LoadPackage("alnuth");
Loading Alnuth 4.0.0 (ALgebraic NUmber THEory and an interface to PARI/GP and OSCAR)
by Björn Assmann,
   Andreas Distler (a.distler@tu-bs.de), and
   Bettina Eick (http://www.iaa.tu-bs.de/beick).
maintained by:
   Max Horn (https://www.quendi.de/math) and
   The GAP Team (support@gap-system.org).
Homepage: https://gap-packages.github.io/alnuth
Report issues at https://github.com/gap-packages/alnuth/issues
true
gap>
```

To load a certain version of Alnuth you can specify the version number as second argument in the call to `LoadPackage`. (See `LoadPackage` (**Reference:** `LoadPackage`) in the GAP 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");`.

Example

```
gap> ReadPackage("Alnuth", "tst/testinstall.g");
Architecture: aarch64-apple-darwin21.4.0-default64-kv8

testing: GAPROOT/pkg/alnuth/tst/ALNUTH.tst
        66 ms (33 ms GC) and 11.0MB allocated for GAPROOT/pkg/alnuth/tst/ALNUTH.tst
testing: GAPROOT/pkg/alnuth/tst/version.tst
        21 ms (21 ms GC) and 29.6KB allocated for GAPROOT/pkg/alnuth/tst/version.tst
-----
total          87 ms (54 ms GC) and 11.0MB allocated
              0 failures in 2 files

#I No errors detected while testing
gap>
```

The architecture, timings and memory usage will usually differ; other discrepancies in the output indicate some problem.

If the test suite runs into an error in the first part, which when running outside OSCAR 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 submitting an issue at the Alnuth issue tracker on GitHub <https://github.com/gap-packages/alnuth/issues> or by writing an mail to support@gap-system.org.

References

- [Coh93] Henri Cohen. *A course in computational algebraic number theory*. Springer-Verlag, New York, Heidelberg, Berlin, 1993. [4](#)
- [DEF⁺25] Wolfram Decker, Christian Eder, Claus Fieker, Max Horn, and Michael Joswig, editors. *The Computer Algebra System OSCAR: Algorithms and Examples*, volume 32 of *Algorithms and Computation in Mathematics*. Springer, 1 edition, 2025. [4](#)
- [EHN11] Bettina Eick, Max Horn, and Werner Nickel. *Polycyclic - a GAP package*, 2011. [6](#), [7](#)
- [OSC24] Oscar – open source computer algebra research system, version 1.0.0, 2024. [4](#)
- [PAR11] The PARI Group, Bordeaux. *PARI/GP, version 2.5.0*, 2011. [4](#)
- [Poh93] Michael E. Pohst. *Computational Algebraic Number Theory*, volume 21 of *DMV Seminar*. Birkhäuser, 1993. [4](#)
- [PZ89] M. Pohst and H. Zassenhaus. *Algorithmic algebraic number theory*. Cambridge University Press, 1989. [4](#)
- [ST79] I. N. Stuart and D. O. Tall. *Algebraic number theory*. Chapman and Hall, 1979. [4](#)

Index

DefiningPolynomial, 6
DegreeOverPrimeField, 6

EquationOrderBasis, 6
ExampleMatField, 9
ExponentsOfUnits, 7

FactorsPolynomialAlgExt, 8
FactorsPolynomialAlnuth, 8
FieldByMatrices, 5
FieldByMatricesNC, 5
FieldByMatrixBasis, 5
FieldByMatrixBasisNC, 5
FieldByPolynomial, 5
FieldByPolynomialNC, 5

IntersectionOfUnitSubgroups, 8
IsCyclotomicField, 7
IsIntegerOfNumberField, 6
IsomorphismPcpGroup, 7
IsPrimitiveElementOfNumberField, 6
IsUnitOfNumberField, 7

MaximalOrderBasis, 6

NormCosetsOfNumberField, 7

PariVersion, 14
PcpPresentationOfMultiplicative-
 Subgroup, 7
PrimitiveElement, 6

RelationLattice, 8
RelationLatticeOfUnits, 8
RestoreAlnuthExternalExecutable-
 Permanently, 15

SetAlnuthExternalExecutable, 15
SetAlnuthExternalExecutable-
 Permanently, 15

UnitGroup, 6