ARITHMETIC IN BIG NUMBER FIELDS: 
THE '+IDEALS' PACKAGE

JORDI GUÀRDIA, JESÚS MONTES, AND ENRIC NART

Abstract. We introduce our package +Ideals for Magma, designed to perform the basic tasks related to ideals in number fields without pre-computing integral bases. It is based on Montes algorithm and a number of local techniques that we have developed in a series of papers in the last years.

Introduction

A commonplace in Number Theory problems is the need of effective computations. The last decade has seen an exponential growth of mathematical software to satisfy this necessity. Striking ideas have expanded our computational limits far beyond our present requirements in many fields, specially those related to cryptographic applications. However, there remain some parts of Computational Number Theory which have not evolved that much. Algebraic Number Theory appears to be one of these.

While the available algorithms for working over number fields are bright and efficient, the computational requirements of real life problems in Number Theory frequently exceed their capability. As soon as we have to work in a number field defined by a polynomial with large discriminant, the determination of its ring of integers, which is the cornerstone of all the classic algorithms, becomes unfeasible, thus making impossible further computations.

We have developed a number of techniques, based on higher order Newton polygons, to deal with the prime ideals in number fields ([GMN08], [GMN09a], [GMN09b], [GMN10]). The application of these ideas leads to extremely fast and efficient algorithms for the basic tasks in Computational Number Theory. These new algorithms can work over number fields of degree 1000 or with a 300-digit discriminant with a home computer in a few seconds. A new philosophy is beyond this project: the centre of attention in Computational Algebraic Number Theory may be not the ring of integers, but the prime ideals.

We have implemented these algorithms in the +Ideals package for Magma. Besides factorization of ideals, this package gives facilities to compute valuations at prime ideals, reduction modulo prime ideals and Chinese Remainder Theorem problems. The algorithms have been presented in [GMN10]. The package can also compute integral bases of number fields, following the algorithm introduced in [GMN09a]. The goal of this paper is to introduce the package, presenting some examples of use and describing its main functions.

2010 Mathematics Subject Classification. Primary 11Y40; Secondary 11Y05, 11R04, 11R27.
Key words and phrases. number field, fractional ideal, Montes algorithm, ideal arithmetic.
Partially supported by MTM2009-13060-C02-02 and MTM2009-10359 from the Spanish MEC.
The paper is structured in three sections: after giving a general overview of the package in the first section, we illustrate its use and capabilities by means of some concrete examples in Section 2. Section 3 gives a more exhaustive description of routines of the package.

A primitive version of the package was published about two years ago, under the name Newton. It included essentially the decomposition of primes in number fields and the computation of integral bases. The functions in the Newton package remain in the +Ideals package, to ensure compatibility for all users.

The +Ideals package can be obtained on demand from the authors, or downloaded from its web page \(\text{http://www-ma4.upc.edu/~guardia/+Ideals.html}\). The package requires version v2.15 of Magma or newer.

1. General overview of the package

The +Ideals package is concerned with the arithmetic of fractional ideals in number fields. The keystone is Montes algorithm, which provides local information for the decomposition of rational primes in number fields. The great advantage of the algorithm is to avoid the use of integral bases. This shortcut makes possible to perform fast computations in number fields of huge degree and index.

The underlying philosophy of the package is that prime ideals are the basic tools for many computations in number fields, and hence all the functions in the package perform tasks related to them. The main task, of course, is factorization of fractional ideals, but computation of valuations and reduction maps are also important. As soon as a prime ideal is detected along a computation, the information which can be useful for other computations is stored in some new attributes for number fields, so that it can be easily retrieved if needed.

The package introduces two new data types, \texttt{PrimeIdealRecord} and \texttt{IdealRecord}, designed to represent fractional ideals in number fields, and some attributes for number fields to store information about these ideals. Most of the functions included in the package are extended versions of the usual Magma routines for ideal arithmetic, designed to deal with these new data types, so that the experienced user of Magma should find no difficulty in the use of the package.

Let us finish this overview with a self-explanatory small piece of code illustrating how to perform the most common tasks in ideal arithmetic with our package:

```
> Z<x>:=PolynomialRing(Integers());
> K:=NumberField(x^100-x^75+x^50+2^500);
> I:=ideal(K,2);
> time Factorization(~I);
Time: 1.300
> I`FactorizationString;
P(2,1)^2*P(2,2)^2*P(2,3)^2*P(2,4)*P(2,5)*P(2,6)*P(2,7)*P(2,8)
> K`LocalIndex[2];
12250
```
2. SOME HARD EXAMPLES

We illustrate the use of the package by means of some examples picked from different topics in Computational Number Theory. The computations have been made with Magma v2.15-11 in a Linux server, with two Intel Quad Core processors, running at 3.0 Ghz, with 32Gb of RAM memory. All the running times mentioned in these examples are expressed in seconds.

2.1. A real life problem. A very common problem in Arithmetic Geometry is the determination of good models for curves over number fields. Usually, a curve with potentially good reduction acquires good reduction over a number field generated by certain torsion points in the jacobian of the curve. The number fields encountered in this process are usually huge, and it is a difficult computational problem to find the places over the rational primes we are interested in. We show in this example how to use our package to solve such a problem. We will determine the reduction type of a rational elliptic curve over a huge number field, applying Tate’s algorithm. The decomposition of primes in the number field is the minimal information required, but no integral basis is necessary to run the algorithm, so that the problem fits very well the philosophy of our package.

Consider the elliptic curve

\[ E : y^2 = x^3 + 3x^2 + 3x, \]

and let \( K = \mathbb{Q}(\theta) \) be the number field determined by a root \( \theta \in \overline{\mathbb{Q}} \) of the 17-th division polynomial. It is a number field of degree 144. We can determine it with the following Magma code:

\[
> E := EllipticCurve([0,3,0,3,0]);
> f := DivisionPolynomial(E,17);
> K<w> := NumberField(f);
\]

The discriminant of the polynomial \( f \) defining \( K \) is huge, but easy to factor, due to its geometric origin:

\[
> d := Discriminant(f);
> Factorization(d);
[ <2, 13728>, <3, 10296>, <17, 143> ]
\]

We cannot expect to find the ring of integers \( \mathbb{Z}_K \) of such a big number field with the standard Magma routines: the function \texttt{MaximalOrder} exhausts the computer’s memory after some hours of work. But we do can compute in this ring! We must redefine \( K \) through a monic polynomial with integer coefficients; since the leading coefficient of \( f \) is 17, we simply type:

\[
> Qx<x> := PolynomialRing(Rationals());
> g := 17^143*Evaluate(f,x/17);
> K<w> := NumberField(PolynomialRing(Integers())!g);
\]

Note that a root of \( g \) is \( w = 17\theta \). Let us now attach our package:\footnote{Replace \texttt{dir} with the name of the directory where you saved the package.}

\[
> Attach("dir/+ideals.m");
\]
Let us find the decomposition of 2 in \( \mathbb{Z}_K \). We first define the ideal \( 2\mathbb{Z}_K \) using the new function `ideal`, and then we factor it into prime ideals:

\[
> \text{i:=ideal(K,2)};
> \text{time Factorization(~i)};
\text{Time: 0.010}
> \text{i\textquotesingle\textbackslash FactorizationString};
\text{P(2,1)^3*P(2,2)^3*P(2,3)^3*P(2,4)^3*P(2,5)^3*P(2,6)^3}
\]

The result means that \( 2\mathbb{Z}_K \) is the cube of the product of six different prime ideals: \( 2\mathbb{Z}_K = (p_1 \cdots p_6)^3 \). The use of \( \sim \) allows the factorization routine to append information to the record corresponding to the given ideal. A shorter way to find the decomposition of a prime in \( \mathbb{Z}_K \) is:

\[
> \text{time Factorization(ideal(K,3))};
\text{[ [ 3, 1, 2 ], [ 3, 2, 2 ], [ 3, 3, 2 ], [ 3, 4, 2 ], [ 3, 5, 2 ] ]}
\text{Time: 0.010}
\]

which indicates that there are 5 different prime ideals in \( \mathbb{Z}_K \) over 3, all of them with ramification index 2 (but now we can not access the assigned attributes of the ideal, since they are not saved) We can access these prime ideals through the new attribute `PrimeIdeals` established for number fields. The command:

\[
> \text{K\textquotesingle\textbackslash PrimeIdeals[2]};
\]

will generate a long output with all the data already computed for these ideals. For instance:

\[
> \text{K\textquotesingle\textbackslash PrimeIdeals[2,1]\textquotesingle f};
\text{8}
\]

gives the residual degree of the first of the prime ideals dividing 2. The residual degrees of the prime ideals above 3 are:

\[
> \text{[P\textquotesingle f:P in K\textquotesingle PrimeIdeals[3]]};
\text{[ 8, 16, 16, 16, 16 ]}
\]

We already have generators for the ideals at hand; typing:

\[
> \text{alpha:=K\textquotesingle\textbackslash PrimeIdeals[3,1]\textbackslash Generator;alpha};
1/3433683820292512484657849089281*(w^129+2451*w^128+...}
\]

we obtain an element \( \alpha \in \mathbb{Z}_K \) such that \( p = (3, \alpha) \) is the prime over 3 with residual degree 8.

Along the factorization routine of the ideal \( p\mathbb{Z}_K \), the program attaches to each prime ideal over \( p \) a local `type`, which carries on some canonical invariants ([GMN09b]). These types are stored as an attribute of the prime ideals (K\textquotesingle\textbackslash PrimeIdeals[p,k]\textquotesingle Type) and contain essential information. As a by-product of the computation of these types we obtain the \( p \)-index of the polynomial \( g \). It is saved in an associative array of Magma called `LocalIndex`, with one entry for each prime number already factored in \( \mathbb{Z}_K \):
> K`LocalIndex[2];
6816

Using this array, we can easily find the discriminant of $K$; we first factor completely
the discriminant $d$ of $g$ and then get the global index:

```plaintext
> time Factorization(ideal(K,17));
...
Time: 0.380
> ind:=(2^(K`LocalIndex[2])*3^(K`LocalIndex[3])*17^(K`LocalIndex[17]));
> disc:=d div ind^2;
> Factorization(disc);
> [ <2, 96>, <3, 72>, <17, 143> ]
```

We can check that the point $P = (\frac{w}{17}, \sqrt{\frac{w^3}{17^3} + 3\frac{w^2}{17^2} + 3\frac{w}{17}}) \in E(K)$
is not defined over $K$. The computation of the square root in the $y$-coordinate may
be quite expensive, so that we simply reduce its second coordinate modulo a prime
ideal:

```plaintext
> Factorization(ideal(K,5));
> yred:=(w^3/17^3+3*w^2/17^2+3*w/17) mod K`PrimeIdeals[5,1];
> IsSquare(yred);
false
```

Let us apply now Tate's algorithm to determine the reduction type of $E$ over the
prime $p = (3, \alpha)$ defined above. We have $a_1 = a_3 = a_6 = 0$, $a_2 = a_4 = 3$, $b_2 = 12$, $b_4 = 6$, $b_6 = 0$, $b_8 = -9$. It’s a simple matter to check the cumulative hypothesis of
Tate's algorithm:

```plaintext
> P:=K`PrimeIdeals[3,1];
> dE:=Discriminant(E);
> PValuation(dE,P);
6
> [PValuation(ai,P): ai in aInvariants(E)];
[ Infinity, 2, Infinity, 2, Infinity ]
> [PValuation(bi,P): bi in bInvariants(E)];
[ 2, 2, Infinity, 4 ]
```

Thus, we have

- $v_p(\Delta) > 0$
- $v_p(a_3) > 0, v_p(a_4) > 0, v_p(a_6) > 0$,
- $v_p(b_2) > 0$,
- $v_p(a_6) \geq 2$,
- $v_p(b_3) \geq 3$,
- $v_p(b_6) \geq 3$,
- $v_p(a_1) > 0, v_p(a_2) > 0, v_p(a_3) \geq 2, v_p(a_4) \geq 2, v_p(a_6) \geq 3$.

We must now consider the polynomial

$$P(T) = T^3 + a_{2,1}T^2 + a_{4,2}T + a_{6,3},$$

where $a_{i,m} = a_i/\Pi^m$, and $\Pi$ is a local uniformizer for $p$. We get one by typing:
Since \( a_{21} \equiv a_{6.3} \equiv 0 \pmod{p} \) and \( v_p(a_{4.2}) = 0 \), the polynomial \( P(T)(\text{mod}p) \) has three different roots in an algebraic closure of the residue field \( k(p) \) of \( p \), so we already know that the Kodaira symbol of \( E \) is \( I^*_0 \). The local conductor is \( p^{e_p(\Delta) - 4} = p^2 \). To determine the local index \( c_p = [E(K_p)/E_0(K_p)] \), we need the number of roots of \( P(T) \) in \( k(p) \). We may proceed as follows:

\[
> \text{time m:=3/PI^2 mod P;}
> \text{Time: 13.570}
> \text{F:=ResidueField(P);}
> \text{FT<T>:=PolynomialRing(F);}
> \text{Factorization(T^3+m*T);}
> \text{<T, 1>,}
> \text{<T^2 + 2*z1^7 + 2*z1^6 + 2*z1^4 + 2*z1^3 + z1 + 2, 1> ]}
\]

Hence, the local index is \( c_p = 2 \). It is easy to check that for the remaining prime ideals in \( Z_K \) above 3, the algorithm proceeds in the same way (all the ramification indices are 2) up to this point, where we find that the corresponding polynomial \( P(T) \) has three different roots in the residue field, and hence the local indices at these primes are equal to 4.

Finally, our package has its own routine \text{IntegralBasis} to compute a \( Z \)-basis of \( Z_K \):

\[
> \text{time basis:=IntegralBasis(K);}
> \text{Time: 18.620}
\]

It has to be mentioned, however, that this routine is based on a conjecture that we have not been able to prove yet (\cite{GMN09a}). In any case, the routine always returns a correct result (it performs a check of its output) and is prepared to warn the user in the improbable case that the algorithm fails (after millions of tests, we have not encountered any counterexamples).

22. \text{A tall number field}. In this second example we will force the computational capabilities of the package to show its power. The concrete computations are not significant on its own; they have been chosen to illustrate the different routines of the package.

Consider the number field \( K = \mathbb{Q}(\theta) \) given by a root \( \theta \in \overline{\mathbb{Q}} \) of the polynomial \( f(x) = x^{1000} + 2^{50}x^{50} + 2^{60} \). The factorization of the discriminant of \( f \) is

\[
\text{Disc}(f) = 2^{53940}3^{50}5^{2000}127^{50}313^{50}743^{50}4886229527^{50}p^{50},
\]

with \( p = 33757269855122049488232352840456326947916489629537 \). The large degree of \( f \) makes impossible to work in this number field using the standard functions of Magma, even after factoring the discriminant, since the computation of the integral basis is compulsory for these functions. But our functions avoid this

\[\text{\footnote{The computacional representation of the residue field of a prime ideal is computed as a successive extension of finite fields:}}\] \[F_0 \subseteq F_1 \subseteq \cdots \subseteq F_{r+1} = F_p = F_0[z_0, z_1, \ldots, z_r],\]

where \( F_0 \) is the prime field and \( F_i := F_{i-1}(z_{i-1}) \), for \( 1 \leq i \leq r + 1 \). The parameter \( r \) is the order of the type attached to \( p \).
computation, so that we can work with ideals in $K$. Let us begin by determining the discriminant of $K$:

```latex
> Attach("/dir/+Ideals.m");
> Z:=Integers();
> Zx<x>:=PolynomialRing(Integers());
> f := x^1000 + 2^50* x^50 + 2^60;
> d:=Discriminant(f);
> Factorization(d);
[ <2, 53940>, <3, 50>, <5, 2000>, <127, 50>,
  <313, 50>, <743, 50>, <4886229527, 50>,
  <337572698551220494882323528404563236947916489629537, 50> ]
> badprimes:=PrimeDivisors(d);
> K<w>:=NumberField(f);
> for p in badprimes do time Factorization(ideal(K,p)); end for;
...

> [K`LocalIndex[p]: p in K`FactorizedPrimes];
[ 26235, 0, 20, 0, 0, 0, 0, 0 ]
```

The last command uses the new attribute `FactorizedPrimes` for number fields: it is a list containing the rational prime numbers that have already been decomposed in $\mathbb{Z}_K$ already. The running time of the factorizations above and the local indexes are gathered in the following table:

| Ideal    | Index | Time  |
|----------|-------|-------|
| $2\mathbb{Z}_K$ | 26235 | 0.36s |
| $3\mathbb{Z}_K$ | 0     | 0.61s |
| $5\mathbb{Z}_K$ | 20    | 0.63s |
| $127\mathbb{Z}_K$ | 0     | 1.29s |
| $313\mathbb{Z}_K$ | 0     | 3.69s |
| $743\mathbb{Z}_K$ | 0     | 6.47s |
| $4886229527\mathbb{Z}_K$ | 0     | 6.96s |
| $p\mathbb{Z}_K$ | 0     | 60s   |

Thus, we need less than 90 seconds to see that the discriminant of $K$ is

$$\text{Disc}(K) = 2^{1470}3^{50}5^{1960}127^{50}313^{50}743^{50}4886229527^{50}p^{50}.$$ 

While we cannot expect to factor the ideals $I = (w^3 + 50)\mathbb{Z}_K$, $J = (w + 10)\mathbb{Z}_K$ in a reasonable time, because this implies the factorization of their norm, we can factor their sum in a moment:

```latex
> I:=ideal(K,w^3+50);
> J:=ideal(K,w+10);
> N:=I+J;
> time Factorization(~N);
Time: 0.030
> N`FactorizationString;
P(2,1)^2*P(2,2)^2*P(2,3)^2*P(2,4)^2*P(2,5)^2
```
This output means that all the primes $P(2,1),\ldots, P(2,5)$ appearing in the list $K`PrimeIdeals[2]$ divide $I + J$ with exponent 2. Note that, besides defining them with the new function `ideal`, we manipulate the ideals with the standard `Magma` notations. We can also multiply and divide fractional ideals in $K$.

```plaintext
> R:=ideal(K,2);
> Factorization(~R);
> time S:=R/N;
Time: 0.280
> S`FactorizationString;
P(2,1)^8*P(2,2)^8*P(2,3)^36*P(2,4)^36*P(2,5)^36
```

Generators of integral ideals are efficiently computed, but the computation of generators of non-integral ideals could be quite time-costing. In the above example, it would take much time to determine generators for $N^{-1}$, but:

```plaintext
> time gens:=TwoElement(S);
Time: 0.000
```

Our last computation in this field in this number field will be a $p$-adic square root. We take the first two prime ideals $p_1, p_2$ in the list $K`PrimeIdeals[5]$, and check whether $w/2$ is a square in their residue fields;

```plaintext
> P1:=K`PrimeIdeals[5,1];
> P2:=K`PrimeIdeals[5,2];
> IsSquare(w/2 mod P1);
false
> t,x0:=IsSquare(w/2 mod P2);t;x0;
true
2*z1+4
```

After Hensel’s lemma, we now that $w/2$ has a square root in $K_{p_2}$, but not in $K_{p_1}$. We can compute this root with Newton’s method, but we will need to lift elements from the residue field of $p_1$ to $K$. This task is performed by the routine `Lift`. For instance, to lift $x_0$ we make:

```plaintext
> X:=Lift(x0,P2);
```

The following code computes $\sqrt{w/2} \in K_{p_2}(\text{mod } p_2^{10})$:

```plaintext
for n in [1..10] do
  lambda:=Lift((2*X mod P2)^(-1),P2);
  X:=X-(X^2-w/2)*lambda;
end for;
print PValuation(X^2-w/2,P2);
```

The whole computation takes about 60 seconds.

2.3. **A fat number field coming from the modular world.** The degree of a number field has a strong influence in the cost of the computations on it, but it is not the only handicap, nor the principal one. The discriminant turns out to be the
highest obstacle in many cases, since it may be a very large integer, impossible to factor. The following example illustrates this situation.

Let $f \in \Gamma_0(1)$ be the weight 76 modular form given by the Magma command

```magma
> f := Newforms(CuspidalSubspace(ModularForms(1,76)))[1,1];
```

It is defined over the number field $K = \mathbb{Q}(\theta)$, where $\theta \in \mathbb{Q}$ is a root of the polynomial

$$F(x) = x^6 + 57080822040x^5 - 198007918566571424544768x^4 - 11405115067164354385292006554337280x^3 + 975762845413169144212884501304149583877426380x^2 + 290013995562379500498435975003716024800114593761580810240x - 92217203874207784163935379997152082331434364841943058919508374716416.$$ 

We can access this number field in Magma with the command:

```magma
> K := Parent(Coefficient(f,1));
```

The discriminant of $F(x)$ is

$$\text{disc}(F) = 2^{264}3^{72}5^{16}7^{16}11^213^417^419^243^2293^2391987^24759427^2137679681521^2M,$$

where $M$ is a composite integer of 135 decimal figures which we have not been able to factorize. J. Rasmussen asked us (\cite{Rasmussen10}) for a test to check certain divisibility conditions on the ring of integers of the number field $K = \mathbb{Q}(\theta)$, related to his work on congruences satisfied by the coefficients of certain modular forms.

Since we cannot factor completely the discriminant, the Magma function `MaximalOrder` is useless. But we can do many computations in $K$ with our package. Let us perform a Chinese Remainder computation. We first define the number field and find the decomposition of 3 in the ring of integers:

```magma
> Attach("dir/+ideals.m");
> I:=ideal(K,3);
> Factorization(~I);
> I`FactorizationString;
```

The factorization routine automatically finds generators for the prime ideals it encounters. They are stored in the attributes `IntegralGenerator` and `Generator` of the `IdealRecord` record subjacent to the ideals, but they can be accessed with the `TwoElement` function:

```magma
> TwoElement(K`PrimeIdeals[3,1]);
[ 3, 1/177147*(2*w^-5 + 1815*w^-4 + 586980*w^-3 + 732159*w^-2 + 658287*w + 1535274) ]
```

Consider the following Chinese Remainder problem: find an element $\alpha \in K$ satisfying

$$\alpha \equiv w(\mod p_1), \quad \alpha \equiv w^2(\mod p_2^3), \quad \alpha \equiv w^3(\mod p_3^4),$$

$$\alpha \equiv w^4(\mod p_4^5), \quad \alpha \equiv 1(\mod p_5).$$
where \( p_1, \ldots, p_5 \) are the prime ideals above 3 in \( \mathbb{Z}_K \), ordered according to the output of our factorization function. We can solve the problem using our extended version of the standard CRT function of Magma. This extended version takes a list of numbers in a number field and a list of ideals in its ring of integers, and solves the corresponding congruences. The following piece of code finds a solution to our problem:

\[
\begin{align*}
> \text{id} & := \text{K'PrimeIdeals}[3,1], \text{K'PrimeIdeals}[3,2]^2, \\
& \quad \text{K'PrimeIdeals}[3,3]^3, \text{K'PrimeIdeals}[3,4]^4, \text{K'PrimeIdeals}[3,5]; \\
> \text{time } r := \text{CRT([w, w^2, w^3, w^4, 1], id); r;}
\end{align*}
\]

Time: 0.010
\[
\frac{1}{19683} (1132037 w^5 + 1583958 w^4 + 498663 w^3 + 923157 w^2 + 649539 w + 354294)
\]

We can check the result using the PValuation routine:

\[
\begin{align*}
> \text{PValuation(r-w, K'PrimeIdeals[3,1]);} \\
7 \\
> \text{PValuation(r-w^2, K'PrimeIdeals[3,2]);} \\
4 \\
> \text{PValuation(r-w^3, K'PrimeIdeals[3,3]);} \\
4 \\
> \text{PValuation(r-w^4, K'PrimeIdeals[3,4]);} \\
4 \\
> \text{PValuation(r-1, K'PrimeIdeals[3,5]);} \\
1
\end{align*}
\]

A second partial test of the result can be done reducing the element \( r \) modulo \( p_1 \) and \( p_2 \):

\[
\begin{align*}
> (r-w) \mod \text{K'PrimeIdeals[3,1];} \\
0 \\
> (r-1) \mod \text{K'PrimeIdeals[3,5];} \\
0
\end{align*}
\]

### 2.4. A two-parametric family of examples.

We now present a two-parametric family of polynomials, which constitutes a good bench mark for our package, since the parameters involved give the chance to grade the difficulty of the computations at our convenience.

Let \( \theta_n \in \overline{\mathbb{Q}} \) be a root of the polynomial \( g_n = x^n + x - 1 \), and let \( K_n = \mathbb{Q}(\theta_n) \). We assume that \( g_n \) is irreducible. For every \( r \in \mathbb{Z} \), the element \( \theta \in K_n \) is also a primitive element of \( K_n \), with minimal polynomial \( f_n^r = x^n + r^{-1} x - r^n \). The discriminants of these polynomials are:

\[
\begin{align*}
\text{Disc}(g_n) &= (-1)^{(n-1)(n-2)/2} (n^n + (n-1)^{n-1}), \\
\text{Disc}(f_n^r) &= r^{n(n-1)} \text{Disc}(g_n).
\end{align*}
\]

Suppose we are given \( f_n^r \) as defining polynomial for \( K_n \). Those primes dividing \( r \) but not \( \Delta(g_n) \) will not ramify in \( K_n \) and we can make the local index arbitrarily large for them. Thus, an (im)proper election of \( r \) can make computations in the field \( K_n \) as difficult as we want.
We will use the following function to make some tests:

```plaintext
> function test(n,r)
function> K:=NumberField(x^n+r^(n-1)*x+r^n: Check:=false);
function> a:=Factorization(ideal(K,r));
function> return [K`LocalIndex[p]: p in PrimeDivisors(r)];
function> end function;
```

The function takes the degree \( n \) and the multiplier \( r \), and determines the decomposition of the primes dividing \( r \) in the field \( K_n \) defined by means of the polynomial \( f_r^n \), and it returns the local index of the discriminant of \( f_r^n \) at them. The following table collects the running times of the test for some values of \( n \) and \( r \):

| \( r \backslash n \) | 100 | 201 | 300 | 400 | 501 |
|---------------------|-----|-----|-----|-----|-----|
| \( 2^{100} \)       | 0.02| 0.05| 0.33| 1.04| 0.69|
| \( 2^{200} \)       | 0.05| 0.1 | 0.85| 2.68| 1.77|
| \( 2^{300} \)       | 0.07| 0.28| 2.23| 7.4 | 4.73|
| \( 2^{400} \)       | 0.17| 0.76| 6.15| 20  | 13.71|
| \( 6^{100} \)       | 0.13| 0.77| 1.7 | 2.9 | 9.77 |
| \( 6^{200} \)       | 0.31| 2.04| 4.57| 7.75| 28.09|
| \( 6^{300} \)       | 0.5 | 5.71| 12.78|21.75|78.38 |
| \( 30^{100} \)      | 0.35| 1.49| 5.02| 6.98|20.910|
| \( 30^{200} \)      | 0.82| 3.91| 13.7|18.34|58.81 |
| \( 30^{300} \)      | 2.24|10.95|38.49|49.66|630.81|
| \( 210^{100} \)     | 0.64| 2.31| 9.63|10.06| 35.25|
| \( 210^{200} \)     | 1.48| 6.03|26.62|25.98| 99.41 |
| \( 210^{300} \)     | 2.63|12.25|31.72|44.33|161.55|

For \( r = m^k \) and fixed \( n \), the local indices at the primes dividing \( m \) are all equal to \( kn(n-1)/2 \). For instance, the largest local index in the table was 50100000, in the case \( k = 400, n = 501 \).

We can make some comparisons with the standard Magma routines for number fields. For instance, we have compared the running time of Magma’s command `MaximalOrder` and our routine `IntegralBasis`, with the following results:

---

These running times do not include the time spent factoring the discriminant of the polynomial \( f_r^n \); which is performed before the computation of the maximal order.
One may wonder whether Magma’s factorization routine is faster than ours if the maximal order has already been computed. The following table compares the performance of both routines for $n = 40$ and multipliers $r_1 = 210^2$ and $r_2 = 210^3$:

| $n$ | $r$    | MaximalOrder | IntegralBasis |
|-----|--------|--------------|---------------|
| 5   | $2^{200}$ | 0.06         | 0.01          |
| 5   | $2^{100}$ | 0.25         | 0.02          |
| 5   | $2^{500}$ | 1.77         | 0.12          |
| 5   | $2^{400}$ | 19.58        | 0.84          |
| 5   | $6^{200}$ | 0.55         | 0.04          |
| 5   | $6^{100}$ | 2.81         | 0.23          |
| 5   | $6^{500}$ | 23.76        | 0.01          |
| 5   | $6^{400}$ | 270          | 0.02          |
| 10  | $2^{200}$ | 1.6          | 0.18          |
| 10  | $2^{100}$ | 14.37        | 1.11          |
| 10  | $2^{50}$  | 185          | 0.1           |
| 20  | $2^{200}$ | 21.37        | 0.06          |
| 20  | $2^{100}$ | 185          | 0.1           |
| 40  | $210^2$   | 168          | 0.71          |
| 40  | $210^3$   | 1066         | 3.28          |

The local index at these primes is 2340 for $r = 210^2$ and 3900 for $r = 210^3$.

2.5. The other way round: how to construct hard examples. This last example illustrates the reverse use of the package to construct examples of number fields with prescribed decomposition of a given set of primes, determining the flux of the Montes algorithm for them. This section may be skipped for the reader not specially interested in the theory behind the package.

According to the philosophy introduced in [GMN10], a prime ideal $p$ in a number field can be canonically represented as a list

$$p = [p; \phi_1, \ldots, \phi_r, \phi_p],$$

where the $\phi_k$ are certain monic polynomials with integral coefficients, determined by Montes algorithm, when applied to the polynomial defining the number field and the rational prime $p$. These polynomials are the main data in the type corresponding to the ideal $p$; the parameter $r$ is the order of this type.

Let us explain how to build a type $t$ with the package. We fix the prime $p = 2$ and the irreducible polynomial $\phi_1 = y + 1 \in \mathbb{F}_2[y]$, and initialize our type
> Attach("/dir/+Ideals.m");
> ZX<x>:=PolynomialRing(Integers());
> FY<y>:= PolynomialRing(GF(2));
> t,Y,z:=InitializeType(2,y+1);

The variable \( t \) will contain our type, the list \( z \) will have the list of generators of the successive residue fields to be built, and the list \( Y \) will be the list of indeterminates generating the corresponding polynomial rings. At this moment \( t \) is the order one type given by the lift \( \phi_1 = x + 1 \in \mathbb{Z}[x] \) of \( y + 1 \). We proceed to enlarge it, constructing a polynomial \( \phi_2 \in \mathbb{Z}[x] \) whose \( \phi_1 \)-Newton polygon has a unique side with slope 1/2 and residual polynomial \( y^2 + y + 1 \):

> EnlargeType(1,2,y^2+y+1, t, Y, z);

The polynomial \( \phi_2 \) is

> t[2]^`Phi;
\[
x^4 + 4x^3 + 8x^2 + 8x + 7
\]

We go one step further, and enlarge the type with a polynomial \( \phi_3 \in \mathbb{Z}[x] \) having a one-sided \( \phi_2 \)-Newton polygon, with slope 2/3 and residual polynomial \( Y[2]^2 + z[2]Y[2] + z[1] \), where \( F_t = F_2(z[2]) = F_2(y)/(y^2 + y + 1) \), and \( z[1] = 1 \).

> EnlargeType(2,3,Y[2]^2+z[2]*Y[2]+z[1], t, Y, z);

The polynomial \( \phi_3 \) is

> t[3]^`Phi;
\[
x^24 + 24*x^23 + 288*x^22 + 2288*x^21 + 13482*x^20 + 62664*x^19 + 238736*x^18 + 765072*x^17 + 2100447*x^16 + 5005808*x^15 + 10456232*x^14 + 19267808*x^13 + 31474060*x^12 + 45694864*x^11 + 9023136*x^10 + 17789216*x^9 + 6069935*x^8 + 62166904*x^7 + 49095584*x^6 + 33671984*x^5 + 19743658*x^4 + 9650728*x^3 + 3776208*x^2 + 1092816*x + 208081
\]

Let us build a second type:

> tt,YY,zz:=InitializeType(2,y^2+y+1);
> EnlargeType(3,2,y+1,~tt,~YY,~zz);
> EnlargeType(3,1,YY[2]^2+YY[2]+zz[1],~tt,~YY,~zz);
> EnlargeType(1,3,YY[2]^2+zz[2],~tt,~YY,~zz);
> tt[4]^`Phi;
\[
x^24 + 12*x^23 + 102*x^22 + 616*x^21 + 3045*x^20 + \ldots
\]

We can combine any number of types to produce a polynomial whose family of attached types will be the given ones.

> pol:=CombineTypes([t,tt]);
> pol;
\[
x^48 + 36*x^47 + 678*x^46 + 8808*x^45 + \ldots
\]

You can check that the polynomial \( \text{pol} \) has the two types \( t,tt \) at the prime \( p = 2 \), and follow the flux of the algorithm using the flag \texttt{montestalk} to set the level of verbosity of the package:
In some sense, these constructions lead to number fields specially ill posed for Montes algorithm, since the order of the types dominates the complexity of the computations. The relation:

\[ \deg(K) = \sum_{t \in T(f,p)} \prod_{k=0}^{t} e_k^t f_k^t, \]

between the degree of a number field and certain entries \( e \) and \( f \) (partial ramification indices and residual degrees) of the types attached to the prime ideals lying over the prime number \( p \), shows that the growth of the order \( \sharp t \) of these types is logarithmic with respect to the degree.

3. Description of the package

3.1. New data types and attributes for ideals and number fields. The centre of the attention of the package are number fields and its fractional ideals. Since computations tend to be quite time-consuming, the important information obtained from any operation is saved. This implies the introduction of new attributes for number fields, and new data types for handling their fractional ideals.

The new attributes for number fields are:

- **FactorizedPrimes**: List containing the rational primes already decomposed in \( K \).
- **PrimeIdeals**: An associative array, with one entry per each rational prime \( p \) in the FactorizedPrimes; it contains the prime ideals in \( \mathbb{Z}_K \) dividing \( p \).
- **LocalIndex**: Associative array which contains, for every rational prime in FactorizedPrimes, the local exponent of the index of the polynomial defining the number field.
• **Discriminant**: The discriminant of the number field.
• **FactorizedDiscriminant**: The factorization of the discriminant of the number field.
• **pIntegralBasis**: An associative array, which for every rational prime $p$ in **FactorizedPrimes** stores a $p$-integral basis of the maximal order of the field.
• **IntegralBasis**: A list with a basis of the maximal order of the fields.

These attributes are assigned and expanded when the corresponding objects are computed. For instance, an integral basis of $K$ is not computed unless the function **IntegralBasis** is used, but then the last four attributes are set, and they are never recomputed.

The fractional ideals are handled by means of two different record structures: the **PrimeIdealRecord** and the **IdealRecord**. The first one is designed to contain the essential information produced by Montes algorithm for every prime ideal, while the second one is used for general ideals. Their concrete fields are:

```plaintext
PrimeIdealRecord:=recformat<
  Parent: // Number field where the prime ideal lies.
  Pol: // Polynomial defining the number field.
  Prime: // Rational prime $p$ below the prime ideal.
  e: // Ramification index of the prime ideal.
  f: // Residual degree of the prime ideal.
  Generator: // The ideal is generated by Prime and this element.
  Position: Position of the prime ideal in the list $K'PrimeIdeals[p]$.
  Type: // Type corresponding to the prime ideal.
>;
```

```plaintext
IdealRecord:=recformat<
  Parent: Number field where the ideal lies.
  Generators: // Generators given in the definition of the ideal.
  Norm: // Norm of the ideal.
  Radical: // List of rational primes dividing the norm of the ideal.
  IsIntegral: // Flag to describe the integrality of the ideal.
  IsPrime: // Flag to describe the primality of the ideal.
  Factorization: // List with the prime ideals dividing the ideal, and the corresponding exponents.
  FactorizationString: // Factorization of the ideal in pretty format.
  IntegerGenerator: // The ideal is generated by it and the element in the next field.
  Generator: // The ideal is generated by it and the element in the previous field.
>;
```

The fields in these structures are assigned only when its computation is trivial or strictly necessary. Once ideals or prime ideals are created, they can be manipulated using the standard notation of Magma for ideal arithmetic.
The trivial ideals (0) and $\mathbb{Z}_K$: an ideal is the zero ideal if its \texttt{Generators} field is assigned, and all its elements are zero; and an ideal is the full ring of integers if its \texttt{Factorization} field is assigned and is an empty list. One must check whether these fields are assigned, since the ideal may be the result of an operation between other ideals, which does not assign any of these fields. The functions \texttt{IsZero} tests whether an ideal is zero.

3.2. Main functions. The essential tasks of the package is the creation and factorization of fractional ideals in number fields; they are performed by means of the functions \texttt{ideal} and \texttt{Factorization}.

The purpose of the function \texttt{ideal} is the creation of a general fractional ideal, given a generator or a list of generators. Accordingly, it admits the two forms:

\begin{verbatim}
ideal(K, alpha1 );
ideal(K, [alpha1,...,alphar]);
\end{verbatim}

where $K$ is a number field and $\alpha_1,...,\alpha_r$ are elements of $K$. The function returns an \texttt{IdealRecord} structure, with the fields \texttt{Parent} and \texttt{Generators} assigned.

Once we have created a fractional ideal $I$, we can call the extended version of \texttt{Factorization} to find its decomposition into prime ideals. We can pass the ideal as a direct argument to the function or as a reference argument:

\begin{verbatim}
Factorization(I);
Factorization(~I);
\end{verbatim}

They do the same computations, but the procedural form assigns the fields \texttt{Factorization}, \texttt{FactorizationString} and \texttt{IsPrime} in the record given by $I$ (if they are not yet assigned) and returns nothing, while the first form returns the value of the field \texttt{Factorization}. The behaviour of the functions depend on the data known for the ideal: If the field \texttt{Factorization} is assigned, the function returns its value. Otherwise, the greatest common divisors of the norms of the generators of the ideal is computed and factored. Every prime number dividing this gcd is decomposed in the number field (if it is not already decomposed) and the valuation of the given ideal at the prime ideals above these prime numbers are computed with the \texttt{PValuation} function.

The decomposition of a rational prime in a number field is performed by the \texttt{Montes} function; the command

\begin{verbatim}
Montes(p,K);
\end{verbatim}

applies Montes algorithm to the polynomial defining the number field $K$ and the rational prime $p$, to find the prime ideal decomposition of $p$ in $K$. It appends $p$ to the list $K$'\texttt{FactorizedPrimes}$, and fills the entries corresponding to $p$ in the associative arrays $K$'\texttt{LocalIndex}$, and $K$'\texttt{PrimeIdeals}$.

A key tool for the factorization is the \texttt{PValuation} function. It accepts an element of a number field and a prime ideal in the field (given as a \texttt{PrimeIdealRecord} or an \texttt{IdealRecord}) and returns the valuation of the ideal generated by the element at the given prime.

Since the discriminant of a number field is a very important invariant, the package includes the function \texttt{TrueDiscriminant} to compute it. Note that it has to factor
the discriminant of the polynomial defining the number field, which may be quite time-consuming or even impossible.

3.3. **Ideal arithmetic.** We can add (+), multiply (*) or divide (/) two ideals and raise a non-zero ideal to any integral power (^) using the standard operators. The mantra for the design of the arithmetic of ideals is "don't waste time in unnecessary hard computations". Thus, for instance, the addition of two ideals simply joins its list of generators and only if the factorization of both ideals is known also finds the factorization of the result. The multiplication of two ideals combines the factorizations of the ideals (computing them if necessary), but does not find generators for the product.

The ideals can also be compared in the usual way: the operators eq and subset have extended versions for such task. They simply compare the factorizations of the ideals to be compared. The norm of an ideal and the rational primes dividing it are given by the functions Norm, and RationalRadical.

3.4. **Generators.** It is well known that any fractional ideal in a number field can be generated by a rational integer and a second integral element in the field. The usual Magma function TwoElement has been extended to apply to our new ideals. It admits two forms

```
TwoElement(I);
TwoElement(~I);
```

They perform the same computation, but the procedural version also assigns the fields Generators, IntegralGenerator and Generator of the given ideal. The factorization of the ideal is computed if it is not known.

3.5. **Congruences. Chinese remainder theorem.** The package provides facilities for congruential computations for prime ideals. Essentially, the package provides the reduction map \( \mathbb{Z}_K \to \mathbb{Z}_K/p \) from the maximal order of the number field \( K \) to the residue field of any prime ideal \( p \). The residue field of a prime ideal is accessed through the command:

```
ResidueField(P);
```

The reduction mod \( p \) of a \( p \)-integral element \( \alpha \in K \) is computed with

```
alpha mod P;
```

An element \( z \in \mathbb{F}_p \) in the residue field of \( p \) may be lifted to an integral element in \( \mathbb{Z}_K \) with the command

```
Lift(z,P);
```

The Magma ChineseRemainderTheorem function is extended in the package to work with the new ideal records. Its syntax is

```
CRT([alpha1,...,alphar],[id1,...,idr]);
```

where \( \alpha_1,\ldots,\alpha_r \) are integral elements of a number field and \( id_1,\ldots,\ldots, id_r \) are integral ideals (given as IdealRecord's or as PrimeIdealRecord's). The function returns an element \( \alpha \) in the same number field, such that \( \alpha \equiv \alpha_k \pmod{p} \).
is divisible by \textit{idk}. Note that the function does not test the integrality of the given elements.

3.6. **Integral Basis.** A conjectural algorithm for the computation of integral bases of number fields can be derived from Montes algorithm. It has been developed in [GMN09a]. The algorithm always returns a correct answer, since it checks its validity; in case of fail, the user is warned. Extensive tests of the algorithm have been done, without no counterexample found.

The functions included in the package for this task are:

- \texttt{pIntegralBasis(K,p)};  
- \texttt{SIntegralBasis(K,[p_1,\ldots,p_r])};  
- \texttt{IntegralBasis(K)};

The basic one is \texttt{pIntegralBasis}, which takes a number field \(K\) and a prime number \(p\) and computes a \(p\)-integral basis of the maximal order \(\mathbb{Z}_K\); the basis is saved in the corresponding entry of the associative array \(K\`p\text{IntegralBasis}\). The prime \(p\) is decomposed in \(K\) only if this has not been done yet.

The \texttt{SIntegralBasis} functions takes a list of primes as second input, and returns a local integral basis for this set of primes. It calls \texttt{pIntegralBasis} for every given prime, and glues the local bases with a Chinese Remainder computation.

Finally, the function \texttt{IntegralBasis} computes a \(\mathbb{Z}\)-basis of the maximal order of the given number field. It simply calls \texttt{SIntegralBasis} with the list of primes dividing the discriminant of the polynomial defining the number field.

3.7. **Construction of hard number fields.** In order to check the main functions of the package we have developed a number of functions to create examples of number fields with prescribed behaviour of the Montes algorithm. They allow to create and combines \textit{types} from its essential data. The basic functions for this task are:

- \texttt{InitializeType(p,\psi)};  
- \texttt{EnlargeType(h,e,\psi,t,Y,z)};

A detailed example of its use has been presented in section 2.5. Remind that we can use the function \texttt{CombineTypes} to combine different types with the same rational prime. The function

\[
\text{CombinePolynomialsWithDifferentPrimes}(f_1,p_1,f_2,p_2,k)
\]

takes two polynomials \(f_1, f_2\) of the same degree and two prime numbers \(p_1, p_2\), and finds a polynomial \(f\) of the same common degree such that \(f \equiv f (\mod p_i^k)\). It is useful for the construction of examples of number fields with prescribed types at different primes.

Random examples can be easily produced using the functions:

- \texttt{CreateRandomType(p,r)};  
- \texttt{CreateRandomMultipleTypePolynomial(p,k,r,s)};

The first one generates a random type of order \(r\) for the prime number \(p\), while the second one computes a random polynomial which has \(k\) different types of order
r for the prime p. This polynomial is obtained as the product of all the last \( \phi \)-polynomials of the randomly created types, plus an integral multiple of \( p^s \) to ensure its irreducibility.

Since the examples produced by these functions use to be enormous, some functions to write them in a format proper for Magma or TeX are included. They are ExpandTeX, ExpandPhiTeX, ExpandAllPhiTeX and ExpandMagma. Their self-explanatory name makes unnecessary any further comment.

REFERENCES

[GMN08] J. Guàrdia, J. Montes, E. Nart, Higher Newton polygons in the computation of discriminants and prime ideal decomposition in number fields, arXiv:0807.4065v3[math.NT].

[GMN09a] J. Guàrdia, J. Montes, E. Nart, Higher Newton polygons and integral bases, arXiv:0902.4428v1[math.NT].

[GMN09b] J. Guàrdia, J. Montes, E. Nart, Okutsu invariants and Newton polygons, Acta Arithmetica, to appear, arXiv:0911.0286v4[math.NT].

[GMN10] J. Guàrdia, J. Montes, E. Nart, A new computational approach to ideal theory in number fields, arXiv:1005.1156v1[math.NT].

[HN08] J. Guàrdia, J. Montes, E. Nart, Newton polygons of higher order in algebraic number theory, arXiv:0807.2620v2[math.NT].

[Rasmussen10] J. Rasmussen, Personal communication.