1 Number Fields 1
2 Elements 269
3 Morphisms 371
4 Orders, Ideals and Ideal Classes 401
5 Algebraic Numbers 601
6 Enumeration of Totally Real Fields 735
7 Indices and Tables 751
Python Module Index 753
Index 755
CHAPTER ONE

NUMBER FIELDS

1.1 Number fields

We define a quartic number field and its quadratic extension:

```
sage: x = polygen(ZZ, 'x')
sage: K.<y> = NumberField(x^4 - 420*x^2 + 40000)
sage: z = y^5/11; z
420/11*y^3 - 40000/11*y
sage: R.<y> = PolynomialRing(K)
sage: f = y^2 + y + 1
sage: L.<a> = K.extension(f); L
Number Field in a with defining polynomial y^2 + y + 1 over its base field
sage: KL.<b> = NumberField([x^4 - 420*x^2 + 40000, x^2 + x + 1]); KL
Number Field in b0 with defining polynomial x^4 - 420*x^2 + 40000 over its base field
```

We do some arithmetic in a tower of relative number fields:

```
sage: K.<cuberoot2> = NumberField(x^3 - 2)
sage: L.<cuberoot3> = K.extension(x^3 - 3)
sage: S.<sqrt2> = L.extension(x^2 - 2)
sage: S
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
sage: sqrt2 * cuberoot3
(cuberoot3 + cuberoot2)
```

(continues on next page)
sage: cuberoot2 + cuberoot3 + sqrt2
sqrt2 + cuberoot3 + cuberoot2
sage: (cuberoot2 + cuberoot3 + sqrt2)^2
(2*cuberoot3 + 2*cuberoot2)*sqrt2 + cuberoot3^2 + 2*cuberoot2*cuberoot3 + cuberoot2^2 → 2
sage: cuberoot2 + sqrt2
sqrt2 + cuberoot2
sage: a = S(cuberoot2); a
cuberoot2
sage: a.parent()
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field

>>> from sage.all import *
>>> K = NumberField(x**Integer(3) - Integer(2), names=('cuberoot2',)); (cuberoot2,) =
→ K._first_ngens(1)
>>> L = K.extension(x**Integer(3) - Integer(3), names=('cuberoot3',)); (cuberoot3,) =
→ L._first_ngens(1)
>>> S = L.extension(x**Integer(2) - Integer(2), names=('sqrt2',)); (sqrt2,) = S._
→ first_ngens(1)
>>> S
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
>>> sqrt2 * cuberoot3
cuberoot3^2*sqrt2
>>> (sqrt2 + cuberoot3)**Integer(5)
(20*cuberoot3^2 + 15*cuberoot3 + 4)*sqrt2 + 3*cuberoot3^2 + 20*cuberoot3 + 60
>>> cuberoot2 + cuberoot3
cuberoot3 + cuberoot2
>>> cuberoot2 + cuberoot3 + sqrt2
sqrt2 + cuberoot3 + cuberoot2
>>> (cuberoot2 + cuberoot3 + sqrt2)**Integer(2)
(2*cuberoot3 + 2*cuberoot2)*sqrt2 + cuberoot3^2 + 2*cuberoot2*cuberoot3 + cuberoot2^2 → 2
>>> cuberoot2 + sqrt2
sqrt2 + cuberoot2
>>> a = S(cuberoot2); a
cuberoot2
>>> a.parent()
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field

Warning: Doing arithmetic in towers of relative fields that depends on canonical coercions is currently VERY SLOW. It is much better to explicitly coerce all elements into a common field, then do arithmetic with them there (which is quite fast).

AUTHORS:

- Steven Sivek (2006-05-12): added support for relative extensions
- William Stein (2007-09-04): major rewrite and documentation
- Robert Bradshaw (2008-10): specified embeddings into ambient fields
- Simon King (2010-05): improved coercion from GAP
class sage.rings.number_field.number_field.CyclotomicFieldFactory

Bases: UniqueFactory

Return the \(n\)-th cyclotomic field, where \(n\) is a positive integer, or the universal cyclotomic field if \(n==0\).

For the documentation of the universal cyclotomic field, see \texttt{UniversalCyclotomicField}.

INPUT:

- \(n\) – a nonnegative integer, default: 0
- \(names\) – name of generator (default: \(zetan\))
- \(bracket\) – Defines the brackets in the case of \(n==0\), and is ignored otherwise. Can be any even length string, with "()" being the default.
- \(embedding\) – bool or \(n\)-th root of unity in an ambient field (default: True)

EXAMPLES:

If called without a parameter, we get the \texttt{universal cyclotomic field}:

```sage
sage: CyclotomicField()
#...
˓→needs sage.libs.gap
Universal Cyclotomic Field

>>> from sage.all import *
>>> CyclotomicField()
#...
˓→needs sage.libs.gap
Universal Cyclotomic Field
```

We create the 7th cyclotomic field \(\mathbb{Q}(\zeta_7)\) with the default generator name.

```sage
sage: k = CyclotomicField(7); k
Cyclotomic Field of order 7 and degree 6

sage: k.gen()
zeta7

>>> from sage.all import *
>>> k = CyclotomicField(Integer(7)); k
Cyclotomic Field of order 7 and degree 6

>>> k.gen()
zeta7
```

The default embedding sends the generator to the complex primitive \(n\)th root of unity of least argument.
Cyclotomic fields are of a special type.

We can specify a different generator name as follows.

The $n$ must be an integer.

The degree must be nonnegative.
The special case \( n = 1 \) does not return the rational numbers:

```
sage: CyclotomicField(1)
Cyclotomic Field of order 1 and degree 1
```

```
>>> from sage.all import *
>>> CyclotomicField(Integer(1))
Cyclotomic Field of order 1 and degree 1
```

Due to their default embedding into \( \mathbb{C} \), cyclotomic number fields are all compatible.

```
sage: cf30 = CyclotomicField(30)
sage: cf5 = CyclotomicField(5)
sage: cf3 = CyclotomicField(3)
sage: cf30.gen() + cf5.gen() + cf3.gen()
zeta30^6 + zeta30^5 + zeta30 - 1
sage: cf6 = CyclotomicField(6) ; z6 = cf6.0
sage: cf3 = CyclotomicField(3) ; z3 = cf3.0
sage: cf3(z6)
zeta3 + 1
sage: cf6(z3)
zeta6 - 1
sage: cf9 = CyclotomicField(9) ; z9 = cf9.0
sage: cf18 = CyclotomicField(18) ; z18 = cf18.0
sage: cf18(z9)
zeta18^2
sage: cf9(z18)
-zeta9^5
sage: cf18(z3)
zeta18^3 - 1
sage: cf18(z6)
zeta18^3
sage: cf18(z6)**2
zeta18^3 - 1
sage: cf9(z3)
zeta9^3
```

```
>>> from sage.all import *
>>> cf30 = CyclotomicField(Integer(30))
>>> cf5 = CyclotomicField(Integer(5))
>>> cf3 = CyclotomicField(Integer(3))
>>> cf30.gen() + cf5.gen() + cf3.gen()
zeta30^6 + zeta30^5 + zeta30 - 1
>>> cf6 = CyclotomicField(Integer(6)) ; z6 = cf6.gen(0)
>>> cf3 = CyclotomicField(Integer(3)) ; z3 = cf3.gen(0)
>>> cf3(z6)
zeta3 + 1
>>> cf6(z3)
zeta6 - 1
>>> cf9 = CyclotomicField(Integer(9)) ; z9 = cf9.gen(0)
>>> cf18 = CyclotomicField(Integer(18)) ; z18 = cf18.gen(0)
>>> cf18(z9)
zeta18^2
>>> cf9(z18)
-zeta9^5
>>> cf18(z3)
zeta18^3 - 1
```

(continues on next page)
create_key \((n=0, \text{names}=\text{None}, \text{embedding}=\text{True})\)
Create the unique key for the cyclotomic field specified by the parameters.

create_object \((\text{version}, \text{key}, **\text{extra}_\text{args})\)
Create the unique cyclotomic field defined by key.

\[
\text{sage.rings.number_field.number_field.GaussianField()}
\]
The field \(\mathbb{Q}[i]\).
\[
\text{sage.rings.number_field.number_field.NumberField(\text{polynomial}, \text{name}, \text{check}=\text{None}, \text{names}=\text{True}, \text{embedding}=\text{None}, \text{latex_name}=\text{None}, \text{assume_disc_small}=\text{None}, \text{maximize_at_primes}=\text{False}, \text{structure}=\text{None}, \text{latex_names}=\text{None}, **\text{kwds})}
\]
Return the number field (or tower of number fields) defined by the irreducible polynomial.

INPUT:

- polynomial – a polynomial over \(\mathbb{Q}\) or a number field, or a list of such polynomials.
- names (or name) – a string or a list of strings, the names of the generators
- check – a boolean (default: True); do type checking and irreducibility checking.
- embedding – None, an element, or a list of elements, the images of the generators in an ambient field (default: None)
- latex_names (or latex_name) – None, a string, or a list of strings (default: None), how the generators are printed for latex output
- assume_disc_small – a boolean (default: False); if True, assume that no square of a prime greater than PARI's primelimit (which should be 500000); only applies for absolute fields at present.
- maximize_at_primes – None or a list of primes (default: None); if not None, then the maximal order is computed by maximizing only at the primes in this list, which completely avoids having to factor the discriminant, but of course can lead to wrong results; only applies for absolute fields at present.
- structure – None, a list or an instance of structure.NumberFieldStructure (default: None), internally used to pass in additional structural information, e.g., about the field from which this field is created as a subfield.

We accept implementation and prec attributes for compatibility with \texttt{AlgebraicExtensionFunctor} but we ignore them as they are not used.

EXAMPLES:

\[
\begin{align*}
\text{sage}: & \quad z = \mathbb{Q}['z'].0 \\
\text{sage}: & \quad K = \text{NumberField}(z^2 - 2, 's'); K \\
\text{Number Field in s with defining polynomial } z^2 - 2 \\
\text{sage}: & \quad s = K.0; s \\
\end{align*}
\]
Constructing a relative number field:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 2)
sage: R.<t> = K[]
sage: L.<b> = K.extension(t^3 + t + a); L
Number Field in b with defining polynomial t^3 + t + a over its base field
sage: L.absolute_field(c)
Number Field in c with defining polynomial x^6 + 2*x^4 + x^2 - 2
sage: a*b
a*b
sage: L(a)
a
sage: L.lift_to_base(b^3 + b)
-a
```

Constructing another number field:

```python
sage: k.<i> = NumberField(x^2 + 1)
sage: R.<z> = k[]
sage: m.<j> = NumberField(z^3 + i*z + 3)
sage: m
Number Field in j with defining polynomial z^3 + i*z + 3 over its base field
```
Number fields are globally unique:

```python
sage: K.<a> = NumberField(x^3 - 5)
sage: a^3
5
sage: L.<a> = NumberField(x^3 - 5)
sage: K is L
True
```

Equality of number fields depends on the variable name of the defining polynomial:

```python
sage: x = polygen(QQ, 'x'); y = polygen(QQ, 'y')
sage: k.<a> = NumberField(x^2 + 3)
sage: m.<a> = NumberField(y^2 + 3)
sage: k == m
False
```

In case of conflict of the generator name with the name given by the preparser, the name given by the preparser takes precedence:
One can also define number fields with specified embeddings, may be used for arithmetic and deduce relations with other number fields which would not be valid for an abstract number field.

```python
sage: K.<a> = NumberField(x^3 - 2, embedding=1.2)
sage: RR.coerce_map_from(K)
Composite map:
  From: Number Field in a with defining polynomial x^3 - 2 with a = 1.259921049894873?
  To:   Real Field with 53 bits of precision
  Defn: Generic morphism:
    From: Number Field in a with defining polynomial x^3 - 2
         with a = 1.259921049894873?
    To:   Real Lazy Field
    Defn: a -> 1.259921049894873?
    then
    Conversion via _mpfr_ method map:
      From: Real Lazy Field
      To:   Real Field with 53 bits of precision

sage: RR(a)
1.25992104989487
sage: 1.1 + a
2.35992104989487
sage: b = 1/(a+1); b
1/3*a^2 - 1/3*a + 1/3
sage: RR(b)
0.442493334024442
sage: L.<b> = NumberField(x^6 - 2, embedding=1.1)
sage: L(a)
b^2
sage: a + b
b^2 + b
```

(continues on next page)
Note that the image only needs to be specified to enough precision to distinguish roots, and is exactly computed to any needed precision:

```
sage: RealField(200)(a)
1.2599210498948731647672106072782283505702514647015079800820
```

One can embed into any other field:

```
sage: K.<a> = NumberField(x^3 - 2, embedding=CC.gen() - 0.6)
sage: CC(a)
-0.629960524947436 + 1.09112363597172*I
```

```
sage: # needs sage.rings.padics
sage: L = Qp(5)
sage: f = polygen(L)^3 - 2
sage: K.<a> = NumberField(x^3 - 2, embedding=f.roots()[0][0])
sage: a + L(1)
4 + 2*5^2 + 2*5^3 + 3*5^4 + 5^5 + 4*5^6 + 2*5^8 + 3*5^9 + 4*5^12 + 4*5^14 + 4*5^15 + 3*5^16 + 5^17 + 5^18 + 2*5^19 + 0(5^20)
sage: L.<b> = NumberField(x^6 - x^2 + 1/10, embedding=1)
sage: K.<a> = NumberField(x^3 - x + 1/10, embedding=b^2)
sage: a + b
b^2 + b
sage: CC(a) == CC(b)^2
True
sage: K.coerce_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10 with a = b^2
  To:  Number Field in b with defining polynomial x^6 - x^2 + 1/10
       with b = 0.9724449978911874?
  Defn: a -> b^2
```

```bash
>>> from sage.all import *
```
```python
>>> # needs sage.rings.padics
>>> L = Qp(Integer(5))
>>> f = polygen(L)**Integer(3) - Integer(2)
>>> K = NumberField(x**Integer(3) - Integer(2), embedding=f.
˓
roots() [Integer(0)] [Integer(0)], names=('a',)); (a,) = K._first_ngens(1)
>>> a + L(Integer(1))
4 + 2*5^2 + 2*5^3 + 3*5^4 + 5^5 + 4*5^6 + 2*5^8 + 3*5^9 + 4*5^12
  + 4*5^14 + 4*5^15 + 3*5^16 + 5^17 + 5^18 + 2*5^19 + O(5^20)
>>> L = NumberField(x**Integer(6) - x**Integer(2) + Integer(1)/Integer(10),␣
˓
embedding=Integer(1), names=('b',)); (b,) = L._first_ngens(1)
>>> K = NumberField(x**Integer(3) - x + Integer(1)/Integer(10),␣
˓
embedding=b**Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> a + b
b^2 + b
>>> CC(a) == CC(b)**Integer(2)
True
>>> K.coerce_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10 with a = b^2
  To:  Number Field in b with defining polynomial x^6 - x^2 + 1/10
  with b = 0.9724449978911874?
  Defn: a -> b^2
```

The `QuadraticField` and `CyclotomicField` constructors create an embedding by default unless otherwise specified:

```python
sage: K.<zeta> = CyclotomicField(15)
sage: CC(zeta)
0.913545457642601 + 0.406736643075800*I
sage: L.<sqrtn3> = QuadraticField(-3)
sage: K(sqrtn3)
2*zeta^5 + 1
sage: sqrtn3 + zeta
2*zeta^5 + zeta + 1
```

```python
>>> from sage.all import *

```
```
>>> from sage.all import *
>>> N = NumberField(x**Integer(3) + Integer(2), embedding=Integer(1), names=('g',)); (g,) = N._first_ngens(1)
>>> Integer(1) < g
False
>>> g > Integer(1)
False
>>> RR(g)
-1.25992104989487
```

If no embedding is specified or is complex, the comparison is not returning something meaningful:

```
sage: N.<g> = NumberField(x^3 + 2)
sage: 1 < g
False
sage: g > 1
True
```

Since SageMath 6.9, number fields may be defined by polynomials that are not necessarily integral or monic. The only notable practical point is that in the PARI interface, a monic integral polynomial defining the same number field is computed and used:

```
sage: K.<a> = NumberField(2*x^3 + x + 1)
sage: K.pari_polynomial()
x^3 - x^2 - 2
```

Elements and ideals may be converted to and from PARI as follows:

```
sage: pari(a)
Mod(-1/2*y^2 + 1/2*y, y^3 - y^2 - 2)
sage: K(pari(a))
a
sage: I = K.ideal(a); I
Fractional ideal (a)
sage: I.pari_hnf()
[1, 0, 0; 0, 1, 0; 0, 0, 1/2]
sage: K.ideal(I.pari_hnf())
Fractional ideal (a)
```
Here is an example where the field has non-trivial class group:

```python
sage: L.<b> = NumberField(3*x^2 - 1/5)
sage: L.pari_polynomial()
x^2 - 15
sage: J = L.primes_above(2)[0]; J
Fractional ideal (2, 15*b + 1)
sage: J.pari_hnf()
[2, 1; 0, 1]
sage: L.ideal(J.pari_hnf())
Fractional ideal (2, 15*b + 1)
```

An example involving a variable name that defines a function in PARI:

```python
sage: theta = polygen(QQ, 'theta')
sage: M.<z> = NumberField([theta^3 + 4, theta^2 + 3]); M
Number Field in z0 with defining polynomial theta^3 + 4 over its base field
```

```
>>> from sage.all import *
>>> theta = polygen(QQ, 'theta')
>>> M = NumberField([theta**Integer(3) + Integer(4), theta**Integer(2) + Integer(3)], names=('z',)); (z,) = M._first_ngens(1); M
Number Field in z0 with defining polynomial theta^3 + 4 over its base field
```

```python
class sage.rings.number_field.number_field.NumberFieldFactory
Bases: UniqueFactory

Factory for number fields.

This should usually not be called directly, use NumberField() instead.

INPUT:

- polynomial -- a polynomial over \( \mathbb{Q} \) or a number field.
```
• name – a string (default: 'a'), the name of the generator
• check – a boolean (default: True); do type checking and irreducibility checking.
• embedding – None or an element, the images of the generator in an ambient field (default: None)
• latex_name – None or a string (default: None), how the generator is printed for latex output
• assume_disc_small – a boolean (default: False); if True, assume that no square of a prime greater than PARI’s primelimit (which should be 500000); only applies for absolute fields at present.
• maximize_at_primes – None or a list of primes (default: None); if not None, then the maximal order is computed by maximizing only at the primes in this list, which completely avoids having to factor the discriminant, but of course can lead to wrong results; only applies for absolute fields at present.
• structure – None or an instance of structure.NumberFieldStructure (default: None), internally used to pass in additional structural information, e.g., about the field from which this field is created as a subfield.

create_key_and_extra_args (polynomial, name, check, embedding, latex_name, assume_disc_small, maximize_at_primes, structure)

Create a unique key for the number fields specified by the parameters.

create_object (version, key, check)

Create the unique number field defined by key.

sage.rings.number_field.number_field.NumberFieldTower (polynomials, names, check=True, embeddings=None, latex_names=None, assume_disc_small=False, maximize_at_primes=None, structures=None)

Create the tower of number fields defined by the polynomials in the list polynomials.

INPUT:

• polynomials – a list of polynomials. Each entry must be polynomial which is irreducible over the number field generated by the roots of the following entries.
• names – a list of strings or a string, the names of the generators of the relative number fields. If a single string, then names are generated from that string.
• check – a boolean (default: True), whether to check that the polynomials are irreducible
• embeddings – a list of elements or None (default: None), embeddings of the relative number fields in an ambient field.
• latex_names – a list of strings or None (default: None), names used to print the generators for latex output.
• assume_disc_small – a boolean (default: False); if True, assume that no square of a prime greater than PARI’s primelimit (which should be 500000); only applies for absolute fields at present.
• maximize_at_primes – None or a list of primes (default: None); if not None, then the maximal order is computed by maximizing only at the primes in this list, which completely avoids having to factor the discriminant, but of course can lead to wrong results; only applies for absolute fields at present.
• structures – None or a list (default: None), internally used to provide additional information about the number field such as the field from which it was created.

OUTPUT:
The relative number field generated by a root of the first entry of polynomials over the relative number field generated by root of the second entry of polynomials ... over the number field over which the last entry of polynomials is defined.

**EXAMPLES:**

```sage
sage: x = polygen(ZZ, 'x')
sage: k.<a,b,c> = NumberField([x^2 + 1, x^2 + 3, x^2 + 5]); k  # indirect doctest
Number Field in a with defining polynomial x^2 + 1 over its base field
sage: a^2
-1
sage: b^2
-3
sage: c^2
-5
sage: (a+b+c)^2
(2*b + 2*c)*a + 2*c*b - 9
```

```sage
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField([x**Integer(2) + Integer(1), x**Integer(2) + Integer(3),
                   x**Integer(2) + Integer(5)], names=('a', 'b', 'c'),); (a, b, c,) = k._first_ngens(3); k  # indirect doctest
Number Field in a with defining polynomial x^2 + 1 over its base field
>>> a**Integer(2)
-1
>>> b**Integer(2)
-3
>>> c**Integer(2)
-5
>>> (a+b+c)**Integer(2)
(2*b + 2*c)*a + 2*c*b - 9
```

The Galois group is a product of 3 groups of order 2:

```sage
sage: k.absolute_field(names='c').galois_group()  # indirect doctest
Galois group 8T3 (2[2][2][2]) with order 8 of x^8 + 36*x^6 + 302*x^4 + 564*x^2 + 121
```

Repeatedly calling base_field allows us to descend the internally constructed tower of fields:

```sage
sage: k.base_field()  # indirect doctest
Number Field in b with defining polynomial x^2 + 3 over its base field
sage: k.base_field().base_field()  # indirect doctest
Number Field in c with defining polynomial x^2 + 5
sage: k.base_field().base_field().base_field()  # indirect doctest
Rational Field
```

(continues on next page)
Number Field in b with defining polynomial x^2 + 3 over its base field

```python
>>> k.base_field().base_field()
Number Field in c with defining polynomial x^2 + 5
>>> k.base_field().base_field().base_field()
Rational Field
```

In the following example the second polynomial is reducible over the first, so we get an error:

```python
sage: v = NumberField([x^3 - 2, x^3 - 2], names='a')
Traceback (most recent call last):
...
ValueError: defining polynomial (x^3 - 2) must be irreducible
```

We mix polynomial parent rings:

```python
sage: k.<y> = QQ[]
sage: m = NumberField([y^3 - 3, x^2 + x + 1, y^3 + 2], 'beta'); m
Number Field in beta0 with defining polynomial y^3 - 3 over its base field
sage: m.base_field ()
Number Field in beta1 with defining polynomial x^2 + x + 1 over its base field
```

A tower of quadratic fields:

```python
sage: K.<a> = NumberField([x^2 + 3, x^2 + 2, x^2 + 1]); K
Number Field in a0 with defining polynomial x^2 + 3 over its base field
sage: K.base_field()  
Number Field in a1 with defining polynomial x^2 + 2 over its base field
sage: K.base_field().base_field() 
Number Field in a2 with defining polynomial x^2 + 1
```

LaTeX versions of generator names can be specified either as:
Algebraic Numbers and Number Fields

```
sage: K = NumberField([x^3 - 2, x^3 - 3, x^3 - 5], names=['a', 'b', 'c'],
                    latex_names=[r'\alpha', r'\beta', r'\gamma'])
sage: K.inject_variables(verbos=False)
sage: latex(a + b + c)
\alpha + \beta + \gamma
```

or as:

```
sage: K = NumberField([x^3 - 2, x^3 - 3, x^3 - 5], names=a, latex_names=r'\alpha')
sage: K.inject_variables()  
Defining a0, a1, a2
sage: latex(a0 + a1 + a2)
\alpha_{0} + \alpha_{1} + \alpha_{2}
```

A bigger tower of quadratic fields:

```
sage: K.<a2,a3,a5,a7> = NumberField([x^2 + p for p in [2,3,5,7]]); K
Number Field in a2 with defining polynomial x^2 + 2 over its base field
sage: a2^2
-2
sage: a3^2
-3
sage: (a2+a3+a5+a7)^3
((6*a5 + 6*a7)*a3 + 6*a7*a5 - 47)*a2 + (6*a7*a5 - 45)*a3 - 41*a5 - 37*a7
```

```
sage: from sage.all import *
>> K = NumberField([x**Integer(3) - Integer(2), x**Integer(3) - Integer(3),
                   x**Integer(3) - Integer(5)], names='a', latex_names=r'\alpha')
>> K.inject_variables()
Defining a0, a1, a2
>> latex(a0 + a1 + a2)
\alpha_{0} + \alpha_{1} + \alpha_{2}
```

The function can also be called by name:

```
1.1. Number fields
```
class sage.rings.number_field.number_field.NumberField_absolute(polynomial, name, \nlatex_name=None, \ncheck=True, \nembedding=None, assume_disc_small=False, \nmaximize_at_primes=None, \nstructure=None)

Bases: NumberField_generic

Function to initialize an absolute number field.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K = NumberField(x^17 + 3, 'a'); K
Number Field in a with defining polynomial x^17 + 3
sage: type(K)
<class 'sage.rings.number_field.number_field.NumberField_absolute_with_category'>
sage: TestSuite(K).run()
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(17) + Integer(3), 'a'); K
Number Field in a with defining polynomial x^17 + 3
>>> type(K)
<class 'sage.rings.number_field.number_field.NumberField_absolute_with_category'>
```
```python
sage: v_fin = tuple(K.primes_above(3))[0]

sage: K.abs_val(phi_real, xi^2)
2.08008382305190

sage: K.abs_val(phi_complex, xi^2)
4.32674871092223

sage: K.abs_val(v_fin, xi^2)
0.111111111111111

>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> K = NumberField(x**Integer(3) - Integer(3), names=('xi',)); (xi,) = K._first_ngens(1)

>>> phi_real = K.places()[Integer(0)]

>>> phi_complex = K.places()[Integer(1)]

>>> v_fin = tuple(K.primes_above(Integer(3)))[Integer(0)]

>>> K.abs_val(phi_real, xi**Integer(2))
2.08008382305190

>>> K.abs_val(phi_complex, xi**Integer(2))
4.32674871092223

>>> K.abs_val(v_fin, xi**Integer(2))
0.111111111111111

Check that Issue #28345 is fixed:

```python
sage: K.abs_val(v_fin, K.zero())
0.000000000000000

```python
```
EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_different()  # Fractional ideal (2)
```

```python
>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)

>>> K.absolute_different()  # Fractional ideal (2)
```

**absolute_discriminant()**

A synonym for `discriminant()`.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_discriminant()  # -4
```

```python
>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)

>>> K.absolute_discriminant()  # -4
```

**absolute_generator()**

An alias for `sage.rings.number_field.number_field.NumberField_generic.gen()`.

This is provided for consistency with relative fields, where the element returned by `sage.rings.number_field.number_field_rel.NumberField_relative.gen()` only generates the field over its base field (not necessarily over $\mathbb{Q}$).

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 - 17)
sage: K.absolute_generator()  # a
```

```python
>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> K = NumberField(x**Integer(2) - Integer(17), names=('a',)); (a,) = K._first_ngens(1)

>>> K.absolute_generator()  # a
```

**absolute_polynomial()**

Return absolute polynomial that defines this absolute field. This is the same as `polynomial()`.

EXAMPLES:
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 + 1)
sage: K.absolute_polynomial ()
x^2 + 1

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> K.absolute_polynomial ()
x^2 + 1

absolute_vector_space(*args, **kwds)

Return vector space over $\mathbb{Q}$ corresponding to this number field, along with maps from that space to this number field and in the other direction.

For an absolute extension this is identical to vector_space().

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^3 - 5)
sage: K.absolute_vector_space()
(Vector space of dimension 3 over Rational Field,
 Isomorphism map:
  From: Vector space of dimension 3 over Rational Field
  To:   Number Field in a with defining polynomial x^3 - 5,
 Isomorphism map:
  From: Number Field in a with defining polynomial x^3 - 5
  To:   Vector space of dimension 3 over Rational Field)

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(5), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> K.absolute_vector_space()
(Vector space of dimension 3 over Rational Field,
 Isomorphism map:
  From: Vector space of dimension 3 over Rational Field
  To:   Number Field in a with defining polynomial x^3 - 5,
 Isomorphism map:
  From: Number Field in a with defining polynomial x^3 - 5
  To:   Vector space of dimension 3 over Rational Field)

automorphisms()

Compute all Galois automorphisms of self.

This uses PARI's pari:nfgaloisconj and is much faster than root finding for many fields.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 + 10000)
sage: K.automorphisms ()
[ ]

Ring endomorphism of Number Field in a with defining polynomial x^2 + 10000
  Defn: a |--> a,
Ring endomorphism of Number Field in a with defining polynomial \(x^2 + 10000\)
Defn: a |\(-->\) -a

```python
>>> from sage.all import *

>>> x = polygen(QQ, 'x')

```

```python
>>> K = NumberField(x**Integer(2) + Integer(10000), names=('a',)); (a,) = K._first_ngens(1)

```

```python
>>> K.automorphisms()
```

```
[Ring endomorphism of Number Field in a with defining polynomial x^2 + 10000
  Defn: a |\(-->\) a,
  Ring endomorphism of Number Field in a with defining polynomial x^2 + 10000
  Defn: a |\(-->\) -a]
```

Here’s a larger example, that would take some time if we found roots instead of using PARI’s specialized machinery:

```python
sage: K = NumberField(x^6 - x^4 - 2*x^2 + 1, 'a')
sage: len(K.automorphisms())
2
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(6) - x**Integer(4) - Integer(2)*x**Integer(2) + Integer(1), 'a')

```

```python
>>> len(K.automorphisms())
2
```

\(L\) is the Galois closure of \(K\):

```python
sage: L = NumberField(x^24 - 84*x^22 + 2814*x^20 - 15880*x^18 - 409563*x^16 +
     - 8543892*x^14 + 25518202*x^12 + 32831026956*x^10 -
     - 672691027218*x^8 - 4985379093428*x^6 +
     - 320854419319140*x^4 +
     - 817662865724712*x^2 + 513191437605441, 'a')
```

```python
sage: len(L.automorphisms())
24
```

```python
>>> from sage.all import *

```

```python
>>> L = NumberField(x**Integer(24) - Integer(84)*x**Integer(22) + Integer(2814)*x**Integer(20) -
     - Integer(15880)*x**Integer(18) - Integer(409563)*x**Integer(16) +
     - Integer(8543892)*x**Integer(14) + Integer(25518202)*x**Integer(12) +
     - Integer(32831026956)*x**Integer(10) -
     - Integer(672691027218)*x**Integer(8) +
     - Integer(4985379093428)*x**Integer(6) +
     - Integer(320854419319140)*x**Integer(4) +
     - Integer(817662865724712)*x**Integer(2) +
     - Integer(513191437605441), 'a')
```

```python
>>> len(L.automorphisms())
24
```

Number fields defined by non-monic and non-integral polynomials are supported (Issue #252):
```python
sage: R.<x> = QQ[]
sage: f = 7/9*x^3 + 7/3*x^2 - 56*x + 123
sage: K.<a> = NumberField(f)
sage: A = K.automorphisms(); A
[Ring endomorphism of Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
  Defn: a |--> a,
  Ring endomorphism of Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
  Defn: a |--> -7/15*a^2 - 18/5*a + 96/5,
  Ring endomorphism of Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
  Defn: a |--> 7/15*a^2 + 13/5*a - 111/5]
sage: prod(x - sigma(a) for sigma in A) == f.monic()
True
```

```plaintext
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> f = Integer(7)/Integer(9)*x**Integer(3) + Integer(7)/Integer(3)*x**Integer(2) - Integer(56)*x + Integer(123)
>>> K = NumberField(f, names=('a',)); (a,) = K._first_ngens(1)
>>> A = K.automorphisms(); A
[Ring endomorphism of Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
  Defn: a |--> a,
  Ring endomorphism of Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
  Defn: a |--> -7/15*a^2 - 18/5*a + 96/5,
  Ring endomorphism of Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
  Defn: a |--> 7/15*a^2 + 13/5*a - 111/5]
>>> prod(x - sigma(a) for sigma in A) == f.monic()
True
```

**base_field()**

Return the base field of `self`, which is always QQ.

EXAMPLES:

```python
sage: K = CyclotomicField(5)
sage: K.base_field()
Rational Field
```

```plaintext
>>> from sage.all import *
>>> K = CyclotomicField(Integer(5))
>>> K.base_field()
Rational Field
```

**change_names (names)**

Return number field isomorphic to `self` but with the given generator name.

INPUT:

- names – should be exactly one variable name.
Also, \( K.\text{structure}() \) returns \( \text{from}_K \) and \( \text{to}_K \), where \( \text{from}_K \) is an isomorphism from \( K \) to \( \text{self} \) and \( \text{to}_K \) is an isomorphism from \( \text{self} \) to \( K \).

**EXAMPLES:**

```
sage: x = polygen(QQ, 'x')
sage: K.<z> = NumberField(x^2 + 3); K
Number Field in z with defining polynomial x^2 + 3
sage: L.<ww> = K.change_names()
sage: L
Number Field in ww with defining polynomial x^2 + 3
sage: L.\text{structure}()[0]
Isomorphism given by variable name change map:
  From: Number Field in ww with defining polynomial x^2 + 3
  To:   Number Field in z with defining polynomial x^2 + 3
sage: L.\text{structure}()[0](ww + 5/3)
z + 5/3
```

```
>>> \text{from} \ \text{sage.\text{all}} \ \text{import} \ *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(3), names=('z',)); (z,) = K._
˓→\text{first}_\text{ngens}(1); K
Number Field in z with defining polynomial x^2 + 3
>>> L = K.change_names(names=('ww',)); (ww,) = L._\text{first}_\text{ngens}(1)
>>> L
Number Field in ww with defining polynomial x^2 + 3
>>> L.\text{structure}()[\text{Integer}(0)]
Isomorphism given by variable name change map:
  From: Number Field in ww with defining polynomial x^2 + 3
  To:   Number Field in z with defining polynomial x^2 + 3
>>> L.\text{structure}()[\text{Integer}(0)](ww + \text{Integer}(5)/\text{Integer}(3))
z + 5/3
```

\text{elements_of_bounded_height} (**kwds)

Return an iterator over the elements of \text{self} with relative multiplicative height at most \text{bound}.

This algorithm computes 2 lists: \( L \) containing elements \( x \) in \( K \) such that \( H_k(x) \leq B \), and a list \( L' \) containing elements \( x \) in \( K \) that, due to floating point issues, may be slightly larger than the bound. This can be controlled by lowering the tolerance.

In the current implementation, both lists \( (L, L') \) are merged and returned in form of iterator.

**ALGORITHM:**

This is an implementation of the revised algorithm (Algorithm 4) in [DK2013]. Algorithm 5 is used for imaginary quadratic fields.

**INPUT:**

\text{kwds}:

- \text{bound} -- a \text{real number}
- \text{tolerance} -- (default: 0.01) a \text{rational number in (0, 1]}
- \text{precision} -- (default: 53) a \text{positive integer}

**OUTPUT:**

an iterator of number field elements

**EXAMPLES:**
There are no elements in a number field with multiplicative height less than 1:

```python
sage: x = polygen(QQ, 'x')
sage: K.<g> = NumberField(x^5 - x + 19)
sage: list(K.elements_of_bounded_height(bound=0.9))
[]
```

The only elements in a number field of height 1 are 0 and the roots of unity:

```python
sage: K.<a> = NumberField(x^2 + x + 1)
sage: list(K.elements_of_bounded_height(bound=1))
[0, a + 1, a, -1, -a - 1, -a, 1]
```

The elements in the output iterator all have relative multiplicative height at most the input bound:

```python
sage: K.<a> = NumberField(x^6 + 2)
sage: L = K.elements_of_bounded_height(bound=5)
sage: for t in L:
    ....:     exp(6^t.global_height())
1.00000000000000
1.00000000000000
1.00000000000000
2.00000000000000
2.00000000000000
2.00000000000000
2.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000
```

(continues on next page)
```python
>>> L = K.elements_of_bounded_height(bound=Integer(5))
>>> for t in L:
...     exp(Integer(6)*t.global_height())
1.00000000000000
1.00000000000000
1.00000000000000
2.00000000000000
2.00000000000000
2.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000

sage: K.<a> = NumberField(x^2 - 71)
sage: L = K.elements_of_bounded_height(bound=20)
sage: all(exp(2*t.global_height()) <= 20 for t in L) # long time (5 s)
True

>>> from sage.all import *

>>> K.<a> = NumberField(x^2 + 17)
sage: L = K.elements_of_bounded_height(bound=120)
sage: len(list(L))
9047

>>> from sage.all import *

>>> K.<a> = NumberField(x^4 - 5)
sage: L = K.elements_of_bounded_height(bound=50)
sage: len(list(L)) # long time (2 s)
2163

>>> from sage.all import *

>>> K.<a> = CyclotomicField(13)
sage: L = K.elements_of_bounded_height(bound=2)
```

(continues on next page)
AUTHORS:

• John Doyle (2013)
• David Krumm (2013)
• Raman Raghukul (2018)

embeddings ($K$)

Compute all field embeddings of this field into the field $K$ (which need not even be a number field, e.g., it could be the complex numbers). This will return an identical result when given $K$ as input again.

If possible, the most natural embedding of this field into $K$ is put first in the list.

INPUT:

• $K$ – a field

EXAMPLES:

sage: # needs sage.groups
sage: x = polygen(QQ, 'x')
We embed a quadratic field into a cyclotomic field:

```sage
sage: L.<a> = QuadraticField(-7)
sage: K = CyclotomicField(7)
sage: L.embeddings(K)
[
Ring morphism:
  From: Number Field in a with defining polynomial x^2 + 7
  with a = 2.645751311064591?*I
  To:  Cyclotomic Field of order 7 and degree 6
  Defn: a |--> 2*zeta7^4 + 2*zeta7^2 + 2*zeta7 + 1,
Ring morphism:
  From: Number Field in a with defining polynomial x^2 + 7
  with a = 2.645751311064591?*I
  To:  Cyclotomic Field of order 7 and degree 6
  Defn: a |--> -2*zeta7^4 + 2*zeta7^2 + 2*zeta7 - 1
]
```

```python
>>> from sage.all import *

>>> L = QuadraticField(-Integer(7), names=('a',)); (a,) = L._first_ngens(1)
>>> K = CyclotomicField(Integer(7))

>>> L.embeddings(K)
[
Ring morphism:
  From: Number Field in a with defining polynomial x^2 + 7
  with a = 2.645751311064591?*I
  To:  Cyclotomic Field of order 7 and degree 6
  Defn: a |--> 2*zeta7^4 + 2*zeta7^2 + 2*zeta7 + 1,
Ring morphism:
  From: Number Field in a with defining polynomial x^2 + 7
  with a = 2.645751311064591?*I
  To:  Cyclotomic Field of order 7 and degree 6
  Defn: a |--> -2*zeta7^4 + 2*zeta7^2 + 2*zeta7 - 1
]
```
with \( a = 2.645751311064591?*I \)

To: Cyclotomic Field of order 7 and degree 6

Defn: \( a \mapsto -2\zeta_7^4 - 2\zeta_7^2 - 2\zeta_7 - 1 \)

We embed a cubic field in the complex numbers:

\[
\text{sage: } x = \text{polygen}(\text{QQ}, 'x') \\
\text{sage: } K.<a> = \text{NumberField}(x^3 - 2) \\
\text{sage: } K\text{.embeddings(CC)}
\]

\[
\begin{align*}
\text{Ring morphism:} \\
\text{From: Number Field in a with defining polynomial } x^3 - 2 \\
\text{To: Complex Field with 53 bits of precision} \\
\text{Defn: } a \mapsto -0.62996052494743... - 1.09112363597172*I,
\end{align*}
\]

\[
\begin{align*}
\text{Ring morphism:} \\
\text{From: Number Field in a with defining polynomial } x^3 - 2 \\
\text{To: Complex Field with 53 bits of precision} \\
\text{Defn: } a \mapsto -0.62996052494743... + 1.09112363597172*I,
\end{align*}
\]

\[
\begin{align*}
\text{Ring morphism:} \\
\text{From: Number Field in a with defining polynomial } x^3 - 2 \\
\text{To: Complex Field with 53 bits of precision} \\
\text{Defn: } a \mapsto 1.25992104989487
\end{align*}
\]

Test that Issue #15053 is fixed:

\[
\text{sage: } K = \text{NumberField}(x^3 - 2, 'a') \\
\text{sage: } K\text{.embeddings(GF(3))}
\]

\[
\begin{align*}
\text{Ring morphism:} \\
\text{From: Number Field in a with defining polynomial } x^3 - 2 \\
\text{To: } \text{Finite Field of size 3} \\
\text{Defn: } a \mapsto 1
\end{align*}
\]
Return a vector space $V$ and isomorphisms $\text{self} \to V$ and $V \to \text{self}$.

**INPUT:**

- `base` – a subfield (default: None); the returned vector space is over this subfield $R$, which defaults to the base field of this function field
- `basis` – a basis for this field over the base
- `maps` – boolean (default True), whether to return $R$-linear maps to and from $V$

**OUTPUT:**

- $V$ – a vector space over the rational numbers
- $\text{from}_V$ – an isomorphism from $V$ to $\text{self}$ (if requested)
- $\text{to}_V$ – an isomorphism from $\text{self}$ to $V$ (if requested)

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^3 + 2)
sage: V, from_V, to_V = k.free_module()
sage: from_V(V([1,2,3]))
3*a^2 + 2*a + 1
sage: to_V(1 + 2*a + 3*a^2)
(1, 2, 3)
sage: V
Vector space of dimension 3 over Rational Field
sage: to_V
Isomorphism map:
  From: Number Field in a with defining polynomial x^3 + 2
  To: Vector space of dimension 3 over Rational Field
sage: from_V(to_V(Integer(2)/Integer(3)*a - Integer(5)/Integer(8)))
2/3*a - 5/8
sage: to_V(from_V(V([0,-1/7,0])))
(0, -1/7, 0)
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> k = NumberField(x**Integer(3) + Integer(2), names=(a,)); (a,) = k._→first_ngens(1)
>>> V, from_V, to_V = k.free_module()
>>> from_V(V([Integer(1),Integer(2),Integer(3)]))
3*a^2 + 2*a + 1
>>> to_V(Integer(1) + Integer(2)*a + Integer(3)*a**Integer(2))
(1, 2, 3)
>>> V
Vector space of dimension 3 over Rational Field
>>> to_V
Isomorphism map:
  From: Number Field in a with defining polynomial x^3 + 2
  To: Vector space of dimension 3 over Rational Field
>>> from_V(to_V(Integer(2)/Integer(3)*a - Integer(5)/Integer(8)))
2/3*a - 5/8
>>> to_V(from_V(V([Integer(0),-Integer(1)/Integer(7),Integer(0)])))
(0, -1/7, 0)
```
**galois_closure** *(names=None, map=False)*

Return number field $K$ that is the Galois closure of $\text{self}$, i.e., is generated by all roots of the defining polynomial of $\text{self}$, and possibly an embedding of $\text{self}$ into $K$.

**INPUT:**

- `names` – variable name for Galois closure
- `map` – (default: False) also return an embedding of $\text{self}$ into $K$

**EXAMPLES:**

```python
sage: # needs sage.groups
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x**4 - 2)
sage: M = K.galois_closure('b'); M
Number Field in b with defining polynomial x^8 + 28*x^4 + 2500
sage: L.<a2> = K.galois_closure(); L
Number Field in a2 with defining polynomial x^8 + 28*x^4 + 2500
sage: K.galois_group(names=('a3')).order()  
8
```

```python
>>> from sage.all import *
>>> # needs sage.groups
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> M = K.galois_closure('b'); M
Number Field in b with defining polynomial x^8 + 28*x^4 + 2500
>>> L = K.galois_closure(names=('a2',)); (a2,) = L._first_ngens(1); L
Number Field in a2 with defining polynomial x^8 + 28*x^4 + 2500
>>> K.galois_group(names=('a3')).order()  
8
```

```python
sage: # needs sage.groups
sage: phi = K.embeddings(L)[0]
sage: phi(K.0)
1/120*a2^5 + 19/60*a2
sage: phi(K.0).minpoly()
x^4 - 2
```

```python
>>> from sage.all import *
>>> # needs sage.groups
>>> phi = K.embeddings(L)[Integer(0)]
>>> phi(K.gen(0))
1/120*a2^5 + 19/60*a2
>>> phi(K.gen(0)).minpoly()
x^4 - 2
```

(continues on next page)
A cyclotomic field is already Galois:

```
>>> # needs sage.groups
>>> K.<a> = NumberField(cyclotomic_polynomial(23))
>>> L.<z> = K.galois_closure()
>>> L
Number Field in z with defining polynomial
x^22 + x^21 + x^20 + x^19 + x^18 + x^17 + x^16 + x^15 + x^14 + x^13 + x^12
 + x^11 + x^10 + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1
```

hilbert_conductor\((a, b)\)

This is the product of all (finite) primes where the Hilbert symbol is \(-1\). What is the same, this is the (reduced) discriminant of the quaternion algebra \((a, b)\) over a number field.

INPUT:

- \(a, b\) – elements of the number field \(self\)

OUTPUT:

squarefree ideal of the ring of integers of \(self\)

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: F.<a> = NumberField(x^2 - x - 1)
sage: F.hilbert_conductor(2*a, F(-1))
Fractional ideal (2)
sage: K.<b> = NumberField(x^3 - 4*x + 2)
sage: K.hilbert_conductor(K(2), K(-2))
Fractional ideal (1)
sage: K.hilbert_conductor(K(2*b), K(-2))
Fractional ideal (b^2 + b - 2)
```

```
(continues on next page)
```
>>> F = NumberField(x**Integer(2) - x - Integer(1), names=('a',)); (a,) = F._first_ngens(1)
>>> F.hilbert_conductor(Integer(2)*a, F(-Integer(1)))
Fractional ideal (2)
>>> K = NumberField(x**Integer(3) - Integer(4)*x + Integer(2), names=('b',)); (b,) = K._first_ngens(1)
>>> K.hilbert_conductor(K(Integer(2)), K(-Integer(2)))
Fractional ideal (1)
>>> K.hilbert_conductor(K(Integer(2)*b), K(-Integer(2)))
Fractional ideal (b^2 + b - 2)

AUTHOR:
• Aly Deines

hilbert_symbol(a, b, P=None)

Return the Hilbert symbol \((a, b)_P\) for a prime \(P\) of \(self\) and non-zero elements \(a\) and \(b\) of \(self\).

If \(P\) is omitted, return the global Hilbert symbol \((a, b)\) instead.

INPUT:
• \(a, b\) – elements of \(self\)
• \(P\) – (default: None) If None, compute the global symbol. Otherwise, \(P\) should be either a prime ideal of \(self\) (which may also be given as a generator or set of generators) or a real or complex embedding.

OUTPUT:
If \(a\) or \(b\) is zero, returns 0.

If \(a\) and \(b\) are non-zero and \(P\) is specified, returns the Hilbert symbol \((a, b)_P\), which is 1 if the equation \(ax^2 + by^2 = 1\) has a solution in the completion of \(self\) at \(P\), and is \(-1\) otherwise.

If \(a\) and \(b\) are non-zero and \(P\) is unspecified, returns 1 if the equation has a solution in \(self\) and \(-1\) otherwise.

EXAMPLES:
Some global examples:

```python
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 - 23)
sage: K.hilbert_symbol(0, a + 5)
0
sage: K.hilbert_symbol(a, 0)
0
sage: K.hilbert_symbol(-a, a + 1)
1
sage: K.hilbert_symbol(-a, a + 2)
-1
sage: K.hilbert_symbol(a, a + 5)
-1
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(23), names=('a',)); (a,) = K._first_ngens(1)
>>> K.hilbert_symbol(Integer(0), a + Integer(5))
0
>>> K.hilbert_symbol(a, Integer(0))
```
That the latter two are unsolvable should be visible in local obstructions. For the first, this is a prime ideal above 19. For the second, the ramified prime above 23:

```
sage: K.hilbert_symbol(-a, a + 2, a + 2)
-1
sage: K.hilbert_symbol(a, a + 5, K.ideal(23).factor()[0][0])
-1

>>> from sage.all import *
>>> K.hilbert_symbol(-a, a + Integer(2), a + Integer(2))
-1
>>> K.hilbert_symbol(a, a + Integer(5), K.ideal(Integer(23)).factor()[Integer(0)][Integer(0)])
-1
```

More local examples:

```
sage: K.hilbert_symbol(a, 0, K.fractional_ideal(5))
0
sage: K.hilbert_symbol(a, a + 5, K.fractional_ideal(5))
1
sage: K.hilbert_symbol(a + 1, 13, (a+6)*K)
-1
sage: [emb1, emb2] = K.embeddings(AA)
sage: K.hilbert_symbol(a, -1, emb1)
-1
```

```
>>> from sage.all import *
>>> K.hilbert_symbol(a, Integer(0), K.fractional_ideal(Integer(5)))
0
>>> K.hilbert_symbol(a, a + Integer(5), K.fractional_ideal(Integer(5)))
1
>>> K.hilbert_symbol(a + Integer(1), Integer(13), (a+Integer(6))*K)
-1
>>> [emb1, emb2] = K.embeddings(AA)
>>> K.hilbert_symbol(a, -Integer(1), emb1)
-1
```

Ideals $P$ can be given by generators:

```
sage: K.<a> = NumberField(x^5 - 23)
sage: pi = 2*a^4 + 3*a^3 + 4*a^2 + 15*a + 11
sage: K.hilbert_symbol(a, a + 5, pi)
1
```
This also works for non-principal ideals:

```python
sage: K.<a> = QuadraticField(-5)
sage: P = K.ideal(3).factor()[0][0]
sage: P.gens_reduced()  # random, could be the other factor
(3, a + 1)
sage: K.hilbert_symbol(a, a + 3, P)
1
sage: K.hilbert_symbol(a, a + 3, [3, a+1])
1
```

Primes above 2:

```python
sage: K.<a> = NumberField(x^5 - 23)
sage: p = [p[0] for p in (2*K).factor() if p[0].norm() == 16][0]
sage: K.hilbert_symbol(a, a + 5, p)
1
sage: K.hilbert_symbol(a, 2, p)
1
sage: K.hilbert_symbol(-1, a - 2, p)
-1
```

(continues on next page)
Various real fields are allowed:

```python
sage: K.<a> = NumberField(x^3+x+1)
sage: K.hilbert_symbol(a/3, 1/2, K.embeddings(RDF)[0])
1
sage: K.hilbert_symbol(a/5, -1, K.embeddings(RR)[0])
-1
sage: [K.hilbert_symbol(a, -1, e) for e in K.embeddings(AA)]
[-1]
```

Real embeddings are not allowed to be disguised as complex embeddings:

```python
def K.<a> = QuadraticField(5)
sage: K.hilbert_symbol(-1, -1, K.embeddings(CC)[0])
Traceback (most recent call last):
  ... ValueError: Possibly real place (=Ring morphism: From: Number Field in a with defining polynomial x^2 - 5 with a = 2.236067977499790? To: Complex Field with 53 bits of precision Defn: a |--> -2.23606797749979?) given as complex embedding in hilbert_symbol. Is it real or complex?
sage: K.hilbert_symbol(-1, -1, K.embeddings(QQbar)[0])
Traceback (most recent call last):
  ... ValueError: Possibly real place (=Ring morphism: From: Number Field in a with defining polynomial x^2 - 5 with a = 2.236067977499790? To: Algebraic Field Defn: a |--> -2.236067977499790?) given as complex embedding in hilbert_symbol. Is it real or complex?
sage: K.<b> = QuadraticField(-5)
sage: K.hilbert_symbol(-1, -1, K.embeddings(CDF)[0])
1
sage: K.hilbert_symbol(-1, -1, K.embeddings(QQbar)[0])
1
```

```python
def from sage.all import *

>>> K = QuadraticField(Integer(5), names=('a',)); (a,) = K._first_ngens(1)

>>> K.hilbert_symbol(-Integer(1), -Integer(1), K.embeddings(CC)[Integer(0)])
(continues on next page)```
Traceback (most recent call last):
...
ValueError: Possibly real place (=Ring morphism:
  From: Number Field in a with defining polynomial x^2 - 5
    with a = 2.236067977499790?
  To:     Complex Field with 53 bits of precision
Defn: a |--> -2.23606797749979)
given as complex embedding in hilbert_symbol. Is it real or complex?
>>> K.hilbert_symbol(-Integer(1), -Integer(1), K.
˓→embeddings(QQbar)[Integer(0)])
Traceback (most recent call last):
...
ValueError: Possibly real place (=Ring morphism:
  From: Number Field in a with defining polynomial x^2 - 5
    with a = 2.236067977499790?
  To:     Algebraic Field
Defn: a |--> -2.236067977499790?)
given as complex embedding in hilbert_symbol. Is it real or complex?

\(a\) and \(b\) do not have to be integral or coprime:

\[
\text{sage: } K.<i> = QuadraticField(-1)
\text{sage: } K.hilbert_symbol(1/2, 1/6, 3*K) 1
\text{sage: } p = 1 + i
\text{sage: } K.hilbert_symbol(p, p, p) 1
\text{sage: } K.hilbert_symbol(p, 3*p, p) -1
\text{sage: } K.hilbert_symbol(3, p, p) -1
\text{sage: } K.hilbert_symbol(1/3, 1/5, 1 + i) 1
\text{sage: } L = QuadraticField(5, 'a')
\text{sage: } L.hilbert_symbol(-3, -1/2, 2) 1
\]

\[
\text{>>> from sage.all import *}
\text{>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
\text{>>> K.hilbert_symbol(Integer(1))/Integer(2), Integer(1)/Integer(5), Integer(1)/Integer(3)*K) 1
\text{>>> p = Integer(1) + i
\text{>>> K.hilbert_symbol(p, p, p) 1
\text{>>> K.hilbert_symbol(p, Integer(3)*p, p) -1
\text{>>> K.hilbert_symbol(Integer(3), p, p) -1
\text{>>> K.hilbert_symbol(Integer(1)/Integer(3), Integer(1)/Integer(5), Integer(1)...)}
\]
>>> L = QuadraticField(Integer(5), 'a')

>>> L.hilbert_symbol(-Integer(3), -Integer(1)/Integer(2), Integer(2))
1

Various other examples:

```
sage: K.<a> = NumberField(x^3 + x + 1)
sage: K.hilbert_symbol(-6912, 24, -a^2 - a - 2)
1
sage: K.<a> = NumberField(x^5 - 23)
sage: P = K.ideal(-1105*a^4 + 1541*a^3 - 795*a^2 - 2993*a + 11853)
sage: Q = K.ideal(-7*a^4 + 13*a^3 - 13*a^2 - 2*a + 50)
sage: b = -a+5
sage: K.hilbert_symbol(a, b, P)
1
sage: K.hilbert_symbol(a, b, Q)
1
sage: K.<a> = NumberField(x^3 - 4*x + 2)
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(3) + x + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> K.hilbert_symbol(-Integer(6912), Integer(24), -a**Integer(2) - a - 2
1
```

(continues on next page)
Check that the bug reported at Issue #16043 has been fixed:

```sage
sage: K.<a> = NumberField(x^2 + 5)
sage: p = K.primes_above(2)[0]; p
Fractional ideal (2, a + 1)
sage: K.hilbert_symbol(2*a, -1, p)
1
sage: K.hilbert_symbol(2*a, 2, p)
-1
sage: K.hilbert_symbol(2*a, -2, p)
-1
```

```sage
from sage.all import *
```

```sage
K = NumberField(x**Integer(2) + Integer(5), names=('a',)); (a,) = K._first_ngens(1)
p = K.primes_above(Integer(2))[Integer(0)]; p
Fractional ideal (2, a + 1)
K.hilbert_symbol(Integer(2)*a, -Integer(1), p)
1
K.hilbert_symbol(Integer(2)*a, Integer(2), p)
-1
K.hilbert_symbol(Integer(2)*a, -Integer(2), p)
-1
```

**AUTHOR:**
- Aly Deines (2010-08-19): part of the doctests
- Marco Streng (2010-12-06)

**hilbert_symbol_negative_at_S** \((S, b, check=True)\)

Return \(a\) such that the Hilbert conductor of \(a\) and \(b\) is \(S\).

**INPUT:**
- \(S\) – a list of places (or prime ideals) of even cardinality
- \(b\) – a non-zero rational number which is a non-square locally at every place in \(S\).
- \(check\) – bool (default: True) perform additional checks on the input and confirm the output

**OUTPUT:**
- an element \(a\) that has negative Hilbert symbol \((a, b)_p\) for every (finite and infinite) place \(p\) in \(S\).

**ALGORITHM:**
The implementation is following algorithm 3.4.1 in [Kir2016]. We note that class and unit groups are computed using the generalized Riemann hypothesis. If it is false, this may result in an infinite loop. Nevertheless, if the algorithm terminates the output is correct.

**EXAMPLES:**
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 + 20072)
sage: S = [K.primes_above(3)[0], K.primes_above(23)[0]]
sage: b = K.hilbert_symbol_negative_at_S(S, a + 1)
sage: [K.hilbert_symbol(b, a + 1, p) for p in S]
[-1, -1]
sage: K.<d> = CyclotomicField(11)
sage: S = [K.primes_above(2)[0], K.primes_above(11)[0]]
sage: b = d + 5
sage: a = K.hilbert_symbol_negative_at_S(S, b)
sage: [K.hilbert_symbol(a, b, p) for p in S]
[-1, -1]
sage: k.<c> = K.maximal_totally_real_subfield()[0]
sage: S = [k.primes_above(3)[0], k.primes_above(5)[0]]
sage: S += k.real_places()[:2]
sage: b = 5 + c + c^9
sage: a = k.hilbert_symbol_negative_at_S(S, b)
sage: [k.hilbert_symbol(a, b, p) for p in S]
[-1, -1, -1, -1]

Note that the closely related Hilbert conductor takes only the finite places into account:

sage: k.hilbert_conductor(a, b)
Fractional ideal (15)

AUTHORS:

• Simon Brandhorst, Anna Haensch (01-05-2018)
is_absolute()

Return True since self is an absolute field.

EXAMPLES:

```
sage: K = CyclotomicField(5)
sage: K.is_absolute()
True
```

```
>>> from sage.all import *
>>> K = CyclotomicField(Integer(5))
>>> K.is_absolute()
True
```

logarithmic_embedding(prec=53)

Return the morphism of self under the logarithmic embedding in the category Set.

The logarithmic embedding is defined as a map from the number field self to \( \mathbb{R}^n \).

It is defined under Definition 4.9.6 in [Coh1993].

INPUT:

• prec – desired floating point precision.

OUTPUT:

the morphism of self under the logarithmic embedding in the category Set.

EXAMPLES:

```
sage: CF.<a> = CyclotomicField(5)
sage: f = CF.logarithmic_embedding()
sage: f(0)
(-1, -1)
sage: f(7)
(3.89182029811063, 3.89182029811063)
```

```
>>> from sage.all import *
>>> CF = CyclotomicField(Integer(5), names=('a',)); (a,) = CF._first_ngens(1)
>>> f = CF.logarithmic_embedding()
>>> f(Integer(0))
(-1, -1)
>>> f(Integer(7))
(3.89182029811063, 3.89182029811063)
```

```
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^3 + 5)
sage: f = K.logarithmic_embedding()
sage: f(0)
(-1, -1)
sage: f(7)
(1.94591014905531, 3.89182029811063)
```

```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(3) + Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> f = K.logarithmic_embedding()
```

(continues on next page)
>> f(Integer(0))
(-1, -1)

>> f(Integer(7))
(1.94591014905531, 3.89182029811063)

sage: F.<a> = NumberField(x^4 - 8*x^2 + 3)
sage: f = F.logarithmic_embedding()
sage: f(0)
(-1, -1, -1, -1)
sage: f(7)
(1.94591014905531, 1.94591014905531, 1.94591014905531, 1.94591014905531)

minkowski_embedding(B=None, prec=None)

Return an \( n \times n \) matrix over RDF whose columns are the images of the basis \( \{1, \alpha, \ldots, \alpha^{n-1}\} \) of self over \( \mathbb{Q} \) (as vector spaces), where here \( \alpha \) is the generator of self over \( \mathbb{Q} \), i.e. self.gen(0). If \( B \) is not None, return the images of the vectors in \( B \) as the columns instead. If \( \text{prec} \) is not None, use RealField(\text{prec}) instead of RDF.

This embedding is the so-called “Minkowski embedding” of a number field in \( \mathbb{R}^n \): given the \( n \) embeddings \( \sigma_1, \ldots, \sigma_n \) of self in \( \mathbb{C} \), write \( \sigma_1, \ldots, \sigma_r \) for the real embeddings, and \( \sigma_{r+1}, \ldots, \sigma_{r+s} \) for choices of one of each pair of complex conjugate embeddings (in our case, we simply choose the one where the image of \( \alpha \) has positive real part). Here \( (r, s) \) is the signature of self. Then the Minkowski embedding is given by

\[
x \mapsto (\sigma_1(x), \ldots, \sigma_r(x), \sqrt{2} \text{Re}(\sigma_{r+1}(x)), \sqrt{2} \text{Im}(\sigma_{r+1}(x)), \ldots, \sqrt{2} \text{Re}(\sigma_{r+s}(x)), \sqrt{2} \text{Im}(\sigma_{r+s}(x)))
\]

Equivalently, this is an embedding of self in \( \mathbb{R}^n \) so that the usual norm on \( \mathbb{R}^n \) coincides with \( |x| = \sum_i |\sigma_i(x)|^2 \) on self.

Todo: This could be much improved by implementing homomorphisms over VectorSpaces.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: F.<alpha> = NumberField(x^3 + 2)
sage: F.minkowski_embedding()
[ 1.00000000000000 -1.25992104989487  1.58740105196820]
[ 1.41421356237309  0.890898718178833 -1.12246204830917]
[0.000000000000000  1.54308184421117  1.94416129723666]
sage: F.minkowski_embedding([1, alpha+2, alpha^2-alpha])
[ 1.00000000000000  0.740078950105127  2.84732210186307]
[ 1.41421356237309  3.71932584284153 -2.01336076643746]
[0.000000000000000  1.54308184421117  0.4010794530246]
sage: F.minkowski_embedding() * (alpha + 2).vector().column()
[0.740078950105127]
[ 3.71932584284153]
[ 1.54308184421117]
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> F = NumberField(x**Integer(3) + Integer(2), names=('alpha',)); (alpha,) = F._first_ngens(1)
>>> F.minkowski_embedding()
[ 1.00000000000000 -1.25992104989487 1.58740105196820]
[ 1.41421356237309 0.890898718108301 -1.12246204830914]
[0.000000000000000 1.54308184421348 1.94416129723932]

>>> F.minkowski_embedding([Integer(1), alpha+Integer(2), alpha**Integer(2)-alpha])
[ 1.00000000000000 0.740078950105127 2.84732101863078]
[ 1.41421356237309 3.71932584288432 -2.01336076643947]
[0.000000000000000 1.54308184421348 0.401079453024548]

>>> F.minkowski_embedding() * (alpha + Integer(2)).vector().column()
[0.740078950105127]
[3.71932584288432]
[1.54308184421348]

optimized_representation (name=None, both_maps=True)

Return a field isomorphic to self with a better defining polynomial if possible, along with field isomorphisms from the new field to self and from self to the new field.

EXAMPLES: We construct a compositum of 3 quadratic fields, then find an optimized representation and transform elements back and forth.

```
sage: x = polygen(QQ, 'x')
sage: K = NumberField([x^2 + p for p in [5, 3, 2]], 'a').absolute_field('b'); K
Number Field in b with defining polynomial x^8 + 40*x^6 + 352*x^4 + 960*x^2 + 576
sage: L, from_L, to_L = K.optimized_representation()
Number Field in b1 with defining polynomial x^8 + 4*x^6 + 7*x^4 + 36*x^2 + 81
sage: to_L(K.0)  # random
4/189*b1^7 + 1/63*b1^6 + 1/27*b1^5 - 2/9*b1^4 - 5/27*b1^3 - 8/9*b1^2 + 3/7*b1 - 3/7
sage: from_L(L.0)  # random
1/1152*b^7 - 1/192*b^6 + 23/576*b^5 - 17/96*b^4 + 37/72*b^3 - 5/6*b^2 + 55/24*b - 3/4
```

```
sage: x = polygen(QQ, 'x')
sage: K = NumberField([x**Integer(2) + p for p in [Integer(5), Integer(3), Integer(2)]], 'a').absolute_field('b'); K
Number Field in b with defining polynomial x^8 + 40*x^6 + 352*x^4 + 960*x^2 + 576
sage: L, from_L, to_L = K.optimized_representation()
Number Field in b1 with defining polynomial x^8 + 4*x^6 + 7*x^4 + 36*x^2 + 81
sage: to_L(K.gen(0))  # random
4/189*b1^7 + 1/63*b1^6 + 1/27*b1^5 - 2/9*b1^4 - 5/27*b1^3 - 8/9*b1^2 + 3/7*b1 - 3/7
sage: from_L(L.gen(0))  # random
1/1152*b^7 - 1/192*b^6 + 23/576*b^5 - 17/96*b^4 + 37/72*b^3 - 5/6*b^2 + 55/24*b - 3/4
```

The transformation maps are mutually inverse isomorphisms.
Number fields defined by non-monic and non-integral polynomials are supported (Issue #252):

```python
sage: K.<a> = NumberField(7/9*x^3 + 7/3*x^2 - 56*x + 123)
sage: K.optimized_representation()
(Number Field in a1 with defining polynomial x^3 - 7*x - 7,
 Ring morphism:
   From: Number Field in a1 with defining polynomial x^3 - 7*x - 7
   To:   Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x...
    ↦ 123
   Defn: a1 |--> 7/225*a^2 - 7/75*a - 42/25,
Ring morphism:
   From: Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x...
   To:   Number Field in a1 with defining polynomial x^3 - 7*x - 7
    ↦ 123
   Defn: a |--> -15/7*a1^2 + 9)
```

```python
>>> from sage.all import *

>>> K = NumberField(Integer(7)/Integer(9)*x**Integer(3) + Integer(7)/...
    →Integer(3)*x**Integer(2) - Integer(56)*x + Integer(123), names=('a',)); (a,
    ↦) = K._first_ngens(1)

>>> K.optimized_representation()
(Number Field in a1 with defining polynomial x^3 - 7*x - 7,
 Ring morphism:
   From: Number Field in a1 with defining polynomial x^3 - 7*x - 7
   To:   Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x...
    ↦ 123
   Defn: a1 |--> 7/225*a^2 - 7/75*a - 42/25,
Ring morphism:
   From: Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x...
   To:   Number Field in a1 with defining polynomial x^3 - 7*x - 7
    ↦ 123
   Defn: a |--> -15/7*a1^2 + 9)
```

`optimized_subfields (degree=0, name=None, both_maps=True)`

Return optimized representations of many (but not necessarily all!) subfields of self of the given degree, or of all possible degrees if degree is 0.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K = NumberField([x^2 + p for p in [5, 3, 2]],'a').absolute_field('b'); K
Number Field in b with defining polynomial x^8 + 40*x^6 + 352*x^4 + 960*x^2 +...
    ↦ 576

sage: L = K.optimized_subfields(name='b')
sage: L[0][0]
```

(continues on next page)
Number Field in \( b_0 \) with defining polynomial \( x \)
\texttt{sage: L[1][0]}
Number Field in \( b_1 \) with defining polynomial \( x^2 - 3x + 3 \)
\texttt{sage: [z[0] for z in L]}  # random -- since algorithm is random
[Number Field in \( b_0 \) with defining polynomial \( x - 1 \),
 Number Field in \( b_1 \) with defining polynomial \( x^2 - x + 1 \),
 Number Field in \( b_2 \) with defining polynomial \( x^4 - 5x^2 + 25 \),
 Number Field in \( b_3 \) with defining polynomial \( x^4 - 2x^2 + 4 \),
 Number Field in \( b_4 \) with defining polynomial \( x^8 + 4x^6 + 7x^4 + 36x^2 + 81 \)]

\[
\begin{align*}
\text{>>> from sage.all import *} \\
\text{>>> \( x = \text{polygen(QQ, 'x') \)}} \\
\text{>>> \( K = \text{NumberField([x**Integer(2) + p \ for p in [Integer(5), Integer(3), Integer(2)]], 'a').absolute_field('b'); K} \\
\text{Number Field in \( b_2 \) with defining polynomial \( x^8 + 40x^6 + 352x^4 + 960x^2 + 576 \)} \\
\text{>>> \( L = K.\text{optimized_subfields(name=b')} \)} \\
\text{>>> L[Integer(0)][Integer(0)]} \\
\text{Number Field in \( b_0 \) with defining polynomial \( x \)} \\
\text{>>> L[Integer(1)][Integer(0)]} \\
\text{Number Field in \( b_1 \) with defining polynomial \( x^2 - 3x + 3 \)} \\
\text{>>> [z[Integer(0)] for z in L]}  # random -- since algorithm is random
[Number Field in \( b_0 \) with defining polynomial \( x - 1 \),
 Number Field in \( b_1 \) with defining polynomial \( x^2 - x + 1 \),
 Number Field in \( b_2 \) with defining polynomial \( x^4 - 5x^2 + 25 \),
 Number Field in \( b_3 \) with defining polynomial \( x^4 - 2x^2 + 4 \),
 Number Field in \( b_4 \) with defining polynomial \( x^8 + 4x^6 + 7x^4 + 36x^2 + 81 \)]
\]

We examine one of the optimized subfields in more detail:
\[
\begin{align*}
\text{\texttt{sage: M, from_M, to_M = L[2]}} \\
\text{\texttt{sage: M} \quad \# random} \\
\text{Number Field in \( b_2 \) with defining polynomial \( x^4 - 5x^2 + 25 \)} \\
\text{\texttt{sage: from_M} \quad \# may be slightly random} \\
\text{Ring morphism:} \\
\text{\hspace{1cm} From: Number Field in \( b_2 \) with defining polynomial \( x^4 - 5x^2 + 25 \)} \\
\text{\hspace{1cm} To: Number Field in \( a_1 \) with defining polynomial} \\
\text{\hspace{1cm} \( x^8 + 40x^6 + 352x^4 + 960x^2 + 576 \)} \\
\text{\hspace{1cm} Defn: \( b_2 \mapsto -5/1152*a_1^7 + 1/96*a_1^6 - 97/576*a_1^5 + 17/48*a_1^4 - 95/72*a_1^3 + 17/12*a_1^2 - 53/24*a_1 - 1 \)}
\end{align*}
\]

\[
\begin{align*}
\text{\texttt{>>> from sage.all import *} \\
\text{\texttt{>>> M, from_M, to_M = L[2]} \\
\text{\texttt{>>> M} \quad \# random} \\
\text{Number Field in \( b_2 \) with defining polynomial \( x^4 - 5x^2 + 25 \)} \\
\text{\texttt{>>> from_M} \quad \# may be slightly random} \\
\text{Ring morphism:} \\
\text{\hspace{1cm} From: Number Field in \( b_2 \) with defining polynomial \( x^4 - 5x^2 + 25 \)} \\
\text{\hspace{1cm} To: Number Field in \( a_1 \) with defining polynomial} \\
\text{\hspace{1cm} \( x^8 + 40x^6 + 352x^4 + 960x^2 + 576 \)} \\
\text{\hspace{1cm} Defn: \( b_2 \mapsto -5/1152*a_1^7 + 1/96*a_1^6 - 97/576*a_1^5 + 17/48*a_1^4 - 95/72*a_1^3 + 17/12*a_1^2 - 53/24*a_1 - 1 \)}
\end{align*}
\]

The \texttt{to_M} map is \texttt{None}, since there is no map from \( K \) to \( M \):

\section*{1.1. Number fields}
We apply the from_M map to the generator of M, which gives a rather large element of $K$:

```python
>>> from sage.all import *
>>> to_M
```

```
sage: from M(M.0)  # random
-5/1152*a1^7 + 1/96*a1^6 - 97/576*a1^5 + 17/48*a1^4
- 95/72*a1^3 + 17/12*a1^2 - 53/24*a1 - 1
```

Nevertheless, that large-ish element lies in a degree 4 subfield:

```python
>>> from sage.all import *
>>> from M(M.gen(0))  # random
>>> to_M
```

```
sage: from M(M.0).minpoly()  # random
x^4 - 5*x^2 + 25
```

```
sage: from sage.all import *
>>> from M(M.gen(0)).minpoly()  # random
x^4 - 5*x^2 + 25
```

**order** (*args, **kwds)

Return the order with given ring generators in the maximal order of this number field.

**INPUT:**

- **gens** – list of elements in this number field; if no generators are given, just returns the cardinality of this number field ($\infty$) for consistency.
- **check_is_integral** – bool (default: True), whether to check that each generator is integral.
- **check_rank** – bool (default: True), whether to check that the ring generated by **gens** is of full rank.
- **allow_subfield** – bool (default: False), if True and the generators do not generate an order, i.e., they generate a subring of smaller rank, instead of raising an error, return an order in a smaller number field.

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: k.<i> = NumberField(x^2 + 1)
sage: k.order(2*i)
Order of conductor 2 generated by 2*i in Number Field in i with defining polynomial x^2 + 1
sage: k.order(10*i)
Order of conductor 10 generated by 10*i in Number Field in i with defining polynomial x^2 + 1
sage: k.order(3)
Traceback (most recent call last):
  ...!
ValueError: the rank of the span of gens is wrong
sage: k.order(1/2)
Traceback (most recent call last):
  ...
```
... ValueError: each generator must be integral

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> k = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = k._
˓→first_ngens(1)
>>> k.order(Integer(2)*i)
Order of conductor 2 generated by 2*i in Number Field in i with defining␣
˓→polynomial x^2 + 1
>>> k.order(Integer(10)*i)
Order of conductor 10 generated by 10*i in Number Field in i with defining␣
˓→polynomial x^2 + 1
>>> k.order(Integer(3))
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
>>> k.order(i/Integer(2))
Traceback (most recent call last):
...
ValueError: each generator must be integral
```

Alternatively, an order can be constructed by adjoining elements to \( \mathbb{Z} \):

```python
sage: K.<a> = NumberField(x^3 - 2)
sage: ZZ[a]
Order generated by a0 in Number Field in a0 with defining polynomial x^3 - 2␣
˓→with a0 = a
```

```python
>>> from sage.all import *
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> ZZ[a]
Order generated by a0 in Number Field in a0 with defining polynomial x^3 - 2␣
˓→with a0 = a
```

**places** (*all_complex=False, prec=None*)

Return the collection of all infinite places of \( \text{self} \).

By default, this returns the set of real places as homomorphisms into \( \text{RIF} \) first, followed by a choice of one of each pair of complex conjugate homomorphisms into \( \text{CIF} \).

On the other hand, if \( \text{prec} \) is not None, we simply return places into \( \text{RealField}(\text{prec}) \) and \( \text{ComplexField}(\text{prec}) \) (or \( \text{RDF}, \text{CDF} \) if \( \text{prec}=53 \)). One can also use \( \text{prec} = \infty \), which returns embeddings into the field \( \overline{\mathbb{Q}} \) of algebraic numbers (or its subfield \( \mathbb{A} \) of algebraic reals); this permits exact computation, but can be extremely slow.

There is an optional flag \( \text{all_complex} \), which defaults to False. If \( \text{all_complex} \) is True, then the real embeddings are returned as embeddings into \( \text{CIF} \) instead of \( \text{RIF} \).

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: F.<alpha> = NumberField(x^3 - 100*x + 1); F.places()
[Ring morphism:
 From: Number Field in alpha with defining polynomial x^3 - 100*x + 1
(continues on next page)```
To: Real Field with 106 bits of precision
Defn: alpha |--> -10.00499625499181184573367219280,

Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 - 100x + 1$
To: Real Field with 106 bits of precision
Defn: alpha |--> 0.01000001000003000012000055000273,

Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 - 100x + 1$
To: Real Field with 106 bits of precision
Defn: alpha |--> 9.994996244991781845613530439509

```python
>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> F = NumberField(x**Integer(3) - Integer(100)*x + Integer(1), names=('alpha',)); (alpha,) = F._first_ngens(1); F.places()

[Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 - 100x + 1$
To: Real Field with 106 bits of precision
Defn: alpha |--> -10.00499625499181184573367219280,

Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 - 100x + 1$
To: Real Field with 106 bits of precision
Defn: alpha |--> 0.01000001000003000012000055000273,

Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 - 100x + 1$
To: Real Field with 106 bits of precision
Defn: alpha |--> 9.994996244991781845613530439509]

sage: F.<alpha> = NumberField(x^3 + 7); F.places()

[Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 + 7$
To: Real Field with 106 bits of precision
Defn: alpha |--> -1.91293118277238910119916839549,

Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 + 7$
To: Complex Field with 53 bits of precision
Defn: alpha |--> 0.956465591386195 + 1.6564699997230*I]

>>> from sage.all import *

>>> F = NumberField(x**Integer(3) + Integer(7), names=('alpha',)); (alpha,) = F._first_ngens(1); F.places()

[Ring morphism:
From: Number Field in alpha with defining polynomial $x^3 + 7$
To: Real Field with 106 bits of precision
Defn: alpha |--> -1.91293118277239,

(continues on next page)
Ring morphism:
  From: Number Field in alpha with defining polynomial $x^3 + 7$
  To: Complex Field with 53 bits of precision
  Defn: alpha |--> 0.956465591386195 + 1.65664699997230*I

sage: F.places(prec=10)

[Ring morphism:
  From: Number Field in alpha with defining polynomial $x^3 + 7$
  To: Real Field with 10 bits of precision
  Defn: alpha |--> -1.9,
  Ring morphism:
  From: Number Field in alpha with defining polynomial $x^3 + 7$
  To: Complex Field with 10 bits of precision
  Defn: alpha |--> 0.96 + 1.7*I]

>>> from sage.all import *

>>> F = NumberField(x**Integer(3) + Integer(7), names=('alpha',)); (alpha,) = ˓→F._first_ngens(1); F.places(all_complex=True)

[Ring morphism:
  From: Number Field in alpha with defining polynomial $x^3 + 7$
  To: Complex Field with 53 bits of precision
  Defn: alpha |--> -1.91293118277239,
  Ring morphism:
  From: Number Field in alpha with defining polynomial $x^3 + 7$
  To: Complex Field with 53 bits of precision
  Defn: alpha |--> 0.956465591386195 + 1.65664699997230*I]

>>> F.places(prec=Integer(10))

[Ring morphism:
  From: Number Field in alpha with defining polynomial $x^3 + 7$
  To: Real Field with 10 bits of precision
  Defn: alpha |--> -1.9,
  Ring morphism:
  From: Number Field in alpha with defining polynomial $x^3 + 7$
  To: Complex Field with 10 bits of precision
  Defn: alpha |--> 0.96 + 1.7*I]

real_places(prec=None)

Return all real places of self as homomorphisms into RIF.

EXAMPLES:

sage: x = polygen(QQ, 'x')

sage: F.<alpha> = NumberField(x**4 - 7) ; F.real_places()

[Ring morphism:
  From: Number Field in alpha with defining polynomial $x^4 - 7$
  To: Real Field with 106 bits of precision
  Defn: alpha |--> -1.62657656169778574321232345494,
  Ring morphism:
  From: Number Field in alpha with defining polynomial $x^4 - 7$
  To: Real Field with 106 bits of precision
  Defn: alpha |--> 1.62657656169778574321232345494]

>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> F = NumberField(x**Integer(4) - Integer(7), names=('alpha',)); (alpha,) = ˓→F._first_ngens(1); F.real_places()

[Ring morphism:
relative_degree()
A synonym for degree().

EXAMPLES:
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_degree()
2

relative_different()
A synonym for different().

EXAMPLES:
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_different()
Fractional ideal (2)

relative_discriminant()
A synonym for discriminant().

EXAMPLES:
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_discriminant()
-4

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> K.relative_degree()
2

relative_discriminant()
A synonym for discriminant().

EXAMPLES:
relative_polynomial()

A synonym for polynomial().

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_polynomial()
x^2 + 1
```

relative_vector_space(*args, **kwds)

A synonym for vector_space().

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_vector_space()
(Vector space of dimension 2 over Rational Field, Isomorphism map:
 From: Vector space of dimension 2 over Rational Field
 To: Number Field in i with defining polynomial x^2 + 1,
 Isomorphism map:
 From: Number Field in i with defining polynomial x^2 + 1
 To: Vector space of dimension 2 over Rational Field)
```

relativize(alpha, names, structure=None)

Given an element in self or an embedding of a subfield into self, return a relative number field \( K \) isomorphic to self that is relative over the absolute field \( \mathbb{Q}(\alpha) \) or the domain of \( \alpha \), along with isomorphisms from \( K \) to self and from self to \( K \).

INPUT:
The following demonstrates distinct embeddings of a subfield into a larger field:
sage: K.<a> = NumberField(x^4 + 2*x^2 + 2)
sage: K0 = K.subfields(2)[0][0]; K0
Number Field in a0 with defining polynomial x^2 - 2*x + 2
sage: rho, tau = K0.embeddings(K)
sage: L0 = K.relativize(rho(K0.gen()), 'b'); L0
Number Field in b0 with defining polynomial x^2 - b1 + 2 over its base field
sage: L1 = K.relativize(rho, 'b'); L1
Number Field in b with defining polynomial x^2 - a0 + 2 over its base field
sage: L2 = K.relativize(tau, 'b'); L2
Number Field in b with defining polynomial x^2 + a0 over its base field
sage: L0.base_field() is K0
False
sage: L1.base_field() is K0
True
sage: L2.base_field() is K0
True

Here we see that with the different embeddings, the relative norms are different:

sage: a0 = K0.gen()
sage: L1_into_K, K_into_L1 = L1.structure()
sage: L2_into_K, K_into_L2 = L2.structure()
sage: len(K.factor(41))
4
sage: w1 = -a^2 + a + 1; P = K.ideal([w1])
sage: Pp = L1.ideal(K_into_L1(w1)).ideal_below(); Pp == K0.ideal([4*a0 + 1])
True
sage: Pp == w1.norm(rho)
True
sage: w2 = a^2 + a - 1; Q = K.ideal([w2])
sage: Qq = L2.ideal(K_into_L2(w2)).ideal_below(); Qq == K0.ideal([-4*a0 + 9])
True
sage: Qq == w2.norm(tau)
True
sage: Pp == Qq
False
```python
>>> from sage.all import *

>>> a0 = K0.gen()

>>> L1_into_K, K_into_L1 = L1.structure()

>>> L2_into_K, K_into_L2 = L2.structure()

>>> len(K.factor(Integer(41)))
4

>>> w1 = -a**Integer(2) + a + Integer(1); P = K.ideal([w1])

>>> Pp = L1.ideal(K_into_L1(w1)).ideal_below(); Pp == K0.ideal([Integer(4)*a0 + Integer(1)])
True

>>> Pp == K0.ideal((-Integer(4)*a0 + Integer(1))
True

>>> w2 = a**Integer(2) + a - Integer(1); Q = K.ideal([w2])

>>> Qq = L2.ideal(K_into_L2(w2)).ideal_below(); Qq == K0.ideal([-Integer(4)*a0 + Integer(9)])
True

>>> Qq == K0.ideal((-Integer(4)*a0 + Integer(9)])
True

>>> Pp == Qq
False
```

**subfields (degree=0, name=None)**

Return all subfields of self of the given degree, or of all possible degrees if degree is 0. The subfields are returned as absolute fields together with an embedding into self. For the case of the field itself, the reverse isomorphism is also provided.

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField([x^3 - 2, x^2 + x + 1])
sage: K = K.absolute_field('b')
sage: S = K.subfields()
sage: len(S)
6
sage: [k[0].polynomial() for k in S]
[x - 3, x^2 - 3*x + 9, x^3 - 3*x^2 + 3*x + 1, x^3 - 3*x^2 + 3*x + 1, x^3 - 3*x^2 + 3*x - 17, x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1]
sage: R.<t> = QQ[]
sage: L = NumberField(t^3 - 3*t + 1, 'c')
sage: [k[1] for k in L.subfields()]
[Ring morphism:
  From: Number Field in c0 with defining polynomial t
  To: Number Field in c with defining polynomial t^3 - 3*t + 1
  Defn: 0 |--> 0,
  Ring morphism:
  From: Number Field in c1 with defining polynomial t^3 - 3*t + 1
  To: Number Field in c with defining polynomial t^3 - 3*t + 1
  Defn: c1 |--> c]
sage: len(L.subfields(2))
0
sage: len(L.subfields(1))
1
```
```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField([x**Integer(3) - Integer(2), x**Integer(2) + x + 
... Integer(1)], names=('a',)); (a,) = K._first_ngens(1)
>>> K = K.absolute_field('b')
>>> S = K.subfields()
>>> len(S)
6
>>> [k[Integer(0)].polynomial() for k in S]
[x - 3,
x^2 - 3*x + 9,
x^3 - 3*x^2 + 3*x + 1,
x^3 - 3*x^2 + 3*x + 1,
x^3 - 3*x^2 + 3*x - 17,
x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1]
>>> R = QQ['t']; (t,) = R._first_ngens(1)
>>> L = NumberField(t**Integer(3) - Integer(3)*t + Integer(1), 'c')
>>> [k[Integer(1)] for k in L.subfields()]
[Ring morphism:
  From: Number Field in c0 with defining polynomial t
  To:   Number Field in c with defining polynomial t^3 - 3*t + 1
  Defn: 0 |--> 0,
Ring morphism:
  From: Number Field in c1 with defining polynomial t^3 - 3*t + 1
  To:   Number Field in c with defining polynomial t^3 - 3*t + 1
  Defn: c1 |--> c]
>>> len(L.subfields(Integer(2)))
0
>>> len(L.subfields(Integer(1)))
1
```

```
>> from sage.all import *
>> from sage.rings.number_field.number_field import NumberField_absolute_v1
>> sage: R.<x> = QQ[]
>> sage: NumberField_absolute_v1(x**2 + 1, 'i', 'i')
Number Field in i with defining polynomial x^2 + 1
```

```
>> from sage.all import *
>> from sage.rings.number_field.number_field import NumberField_absolute_v1
>> R = QQ['x']; (x,) = R._first_ngens(1)
>> NumberField_absolute_v1(x**Integer(2) + Integer(1), 'i', 'i')
Number Field in i with defining polynomial x^2 + 1
```

```
class sage.rings.number_field.number_field.NumberField_cyclotomic(n, names, embedding=None, assume_disc_small=False, maximize_at_primes=None)
Bases: NumberField_absolute, NumberField_cyclotomic
```

1.1. Number fields

Used for unpickling old pickles.

EXAMPLES:

```
sage: NumberField_cyclotomic(1, 'i', 'i')
Number Field in i with defining polynomial x^1
```

```
sage: NumberField_cyclotomic(2, 'i', 'i')
Number Field in i with defining polynomial x^2
```

```
sage: NumberField_cyclotomic(3, 'i', 'i')
Number Field in i with defining polynomial x^3
```

```
sage: NumberField_cyclotomic(4, 'i', 'i')
Number Field in i with defining polynomial x^4
```

```
sage: NumberField_cyclotomic(5, 'i', 'i')
Number Field in i with defining polynomial x^5
```

```
sage: NumberField_cyclotomic(6, 'i', 'i')
Number Field in i with defining polynomial x^6
```

```
sage: NumberField_cyclotomic(7, 'i', 'i')
Number Field in i with defining polynomial x^7
```

```
sage: NumberField_cyclotomic(8, 'i', 'i')
Number Field in i with defining polynomial x^8
```

```
sage: NumberField_cyclotomic(9, 'i', 'i')
Number Field in i with defining polynomial x^9
```

```
sage: NumberField_cyclotomic(10, 'i', 'i')
Number Field in i with defining polynomial x^{10}
```

```
sage: NumberField_cyclotomic(11, 'i', 'i')
Number Field in i with defining polynomial x^{11}
```

```
sage: NumberField_cyclotomic(12, 'i', 'i')
Number Field in i with defining polynomial x^{12}
```

```
sage: NumberField_cyclotomic(13, 'i', 'i')
Number Field in i with defining polynomial x^{13}
```

```
sage: NumberField_cyclotomic(14, 'i', 'i')
Number Field in i with defining polynomial x^{14}
```

```
sage: NumberField_cyclotomic(15, 'i', 'i')
Number Field in i with defining polynomial x^{15}
```

```
sage: NumberField_cyclotomic(16, 'i', 'i')
Number Field in i with defining polynomial x^{16}
```

```
sage: NumberField_cyclotomic(17, 'i', 'i')
Number Field in i with defining polynomial x^{17}
```

```
sage: NumberField_cyclotomic(18, 'i', 'i')
Number Field in i with defining polynomial x^{18}
```

```
sage: NumberField_cyclotomic(19, 'i', 'i')
Number Field in i with defining polynomial x^{19}
```

```
sage: NumberField_cyclotomic(20, 'i', 'i')
Number Field in i with defining polynomial x^{20}
```

```
sage: NumberField_cyclotomic(21, 'i', 'i')
Number Field in i with defining polynomial x^{21}
```

```
sage: NumberField_cyclotomic(22, 'i', 'i')
Number Field in i with defining polynomial x^{22}
```

```
sage: NumberField_cyclotomic(23, 'i', 'i')
Number Field in i with defining polynomial x^{23}
```

```
sage: NumberField_cyclotomic(24, 'i', 'i')
Number Field in i with defining polynomial x^{24}
```

```
sage: NumberField_cyclotomic(25, 'i', 'i')
Number Field in i with defining polynomial x^{25}
```

```
sage: NumberField_cyclotomic(26, 'i', 'i')
Number Field in i with defining polynomial x^{26}
```

```
sage: NumberField_cyclotomic(27, 'i', 'i')
Number Field in i with defining polynomial x^{27}
```

```
sage: NumberField_cyclotomic(28, 'i', 'i')
Number Field in i with defining polynomial x^{28}
```

```
sage: NumberField_cyclotomic(29, 'i', 'i')
Number Field in i with defining polynomial x^{29}
```

```
sage: NumberField_cyclotomic(30, 'i', 'i')
Number Field in i with defining polynomial x^{30}
```

```
sage: NumberField_cyclotomic(31, 'i', 'i')
Number Field in i with defining polynomial x^{31}
```

```
sage: NumberField_cyclotomic(32, 'i', 'i')
Number Field in i with defining polynomial x^{32}
```

```
sage: NumberField_cyclotomic(33, 'i', 'i')
Number Field in i with defining polynomial x^{33}
```

```
sage: NumberField_cyclotomic(34, 'i', 'i')
Number Field in i with defining polynomial x^{34}
```

```
sage: NumberField_cyclotomic(35, 'i', 'i')
Number Field in i with defining polynomial x^{35}
```

```
sage: NumberField_cyclotomic(36, 'i', 'i')
Number Field in i with defining polynomial x^{36}
```

```
sage: NumberField_cyclotomic(37, 'i', 'i')
Number Field in i with defining polynomial x^{37}
```

```
sage: NumberField_cyclotomic(38, 'i', 'i')
Number Field in i with defining polynomial x^{38}
```

```
sage: NumberField_cyclotomic(39, 'i', 'i')
Number Field in i with defining polynomial x^{39}
```

```
sage: NumberField_cyclotomic(40, 'i', 'i')
Number Field in i with defining polynomial x^{40}
```

```
sage: NumberField_cyclotomic(41, 'i', 'i')
Number Field in i with defining polynomial x^{41}
```

```
sage: NumberField_cyclotomic(42, 'i', 'i')
Number Field in i with defining polynomial x^{42}
```

```
sage: NumberField_cyclotomic(43, 'i', 'i')
Number Field in i with defining polynomial x^{43}
```

```
sage: NumberField_cyclotomic(44, 'i', 'i')
Number Field in i with defining polynomial x^{44}
```

```
sage: NumberField_cyclotomic(45, 'i', 'i')
Number Field in i with defining polynomial x^{45}
```

```
sage: NumberField_cyclotomic(46, 'i', 'i')
Number Field in i with defining polynomial x^{46}
```

```
sage: NumberField_cyclotomic(47, 'i', 'i')
Number Field in i with defining polynomial x^{47}
```

```
sage: NumberField_cyclotomic(48, 'i', 'i')
Number Field in i with defining polynomial x^{48}
```

```
sage: NumberField_cyclotomic(49, 'i', 'i')
Number Field in i with defining polynomial x^{49}
```

```
sage: NumberField_cyclotomic(50, 'i', 'i')
Number Field in i with defining polynomial x^{50}
```

```
sage: NumberField_cyclotomic(51, 'i', 'i')
Number Field in i with defining polynomial x^{51}
```

```
sage: NumberField_cyclotomic(52, 'i', 'i')
Number Field in i with defining polynomial x^{52}
```

```
sage: NumberField_cyclotomic(53, 'i', 'i')
Number Field in i with defining polynomial x^{53}
```

```
sage: NumberField_cyclotomic(54, 'i', 'i')
Number Field in i with defining polynomial x^{54}
```

```
sage: NumberField_cyclotomic(55, 'i', 'i')
Number Field in i with defining polynomial x^{55}
```
```
Create a cyclotomic extension of the rational field.

The command $\text{CyclotomicField}(n)$ creates the $n$-th cyclotomic field, obtained by adjoining an $n$-th root of unity to the rational field.

**EXAMPLES:**

```python
sage: CyclotomicField(3)
Cyclotomic Field of order 3 and degree 2
sage: CyclotomicField(18)
Cyclotomic Field of order 18 and degree 6
sage: z = CyclotomicField(6).gen(); z
zeta6
sage: z^3
-1
sage: (1+z)^3
6*zeta6 - 3
```

```python
>>> from sage.all import *
>>> CyclotomicField(Integer(3))
Cyclotomic Field of order 3 and degree 2
>>> CyclotomicField(Integer(18))
Cyclotomic Field of order 18 and degree 6
>>> z = CyclotomicField(Integer(6)).gen(); z
zeta6
>>> z**Integer(3)
-1
>>> (Integer(1)+z)**Integer(3)
6*zeta6 - 3
```

```python
sage: K = CyclotomicField(197)
sage: loads(K.dumps()) == K
True
sage: loads((z^2).dumps()) == z^2
True
```

```python
>>> from sage.all import *
>>> K = CyclotomicField(Integer(197))
>>> loads(K.dumps()) == K
True
>>> loads((z**Integer(2)).dumps()) == z**Integer(2)
True
```

```python
sage: cf12 = CyclotomicField(12)
sage: z12 = cf12.gen(0)
sage: cf6 = CyclotomicField(6)
sage: z6 = cf6.gen(0)
```

```python
sage: FF = Frac(cf12[x])
sage: x = FF.0
sage: z6*x^3/(z6 + x)
zeta12^2*x^3/(x + zeta12^2)
```

```python
>>> from sage.all import *
>>> cf12 = CyclotomicField(Integer(12))
>>> z12 = cf12.gen(0)
>>> cf6 = CyclotomicField(Integer(6))
>>> z6 = cf6.gen(0)
```

(continues on next page)
complex_embedding (prec=53)

Return the embedding of this cyclotomic field into the approximate complex field with precision \(\text{prec}\) obtained by sending the generator \(\zeta\) of \(\text{self}\) to \(\exp(2\pi i/n)\), where \(n\) is the multiplicative order of \(\zeta\).

EXAMPLES:
From: Cyclotomic Field of order 4 and degree 2
To: Complex Field with 53 bits of precision
Defn: zeta4 |--> 6.12323399573677e-17 + 1.00000000000000*I

Note in the example above that the way zeta is computed (using sine and cosine in MPFR) means that only
the \texttt{prec} bits of the number after the decimal point are valid.

```python
sage: K = CyclotomicField(3)
sage: phi = K.complex_embedding(10)
sage: phi(K.0)
-0.50 + 0.87*I
sage: phi(K.0^3)
1.0
sage: phi(K.0^3 - 1)
0.00
sage: phi(K.0^3 + 7)
8.0
```

```python
>>> from sage.all import *
>>> K = CyclotomicField(Integer(3))
>>> phi = K.complex_embedding(Integer(10))
>>> phi(K.gen(0))
-0.50 + 0.87*I
>>> phi(K.gen(0)**Integer(3))
1.0
>>> phi(K.gen(0)**Integer(3) - Integer(1))
0.00
>>> phi(K.gen(0)**Integer(3) + Integer(7))
8.0
```

\texttt{complex\_embeddings}(\texttt{prec}=53)

Return all embeddings of this cyclotomic field into the approximate complex field with precision \texttt{prec}.

If you want 53-bit double precision, which is faster but less reliable, then do \texttt{self.embeddings(CDF)}.

\textbf{EXAMPLES:}

```python
sage: CyclotomicField(5).complex_embeddings()
[
    Ring morphism:
    From: Cyclotomic Field of order 5 and degree 4
    To: Complex Field with 53 bits of precision
    Defn: zeta5 |--> 0.309016994374947 + 0.951056516295154*I,
    Ring morphism:
    From: Cyclotomic Field of order 5 and degree 4
    To: Complex Field with 53 bits of precision
    Defn: zeta5 |--> -0.809016994374947 + 0.587785252292473*I,
    Ring morphism:
    From: Cyclotomic Field of order 5 and degree 4
    To: Complex Field with 53 bits of precision
    Defn: zeta5 |--> -0.809016994374947 - 0.587785252292473*I,
    Ring morphism:
    From: Cyclotomic Field of order 5 and degree 4
    To: Complex Field with 53 bits of precision
    Defn: zeta5 |--> 0.309016994374947 - 0.951056516295154*I
]```
construction()

Return data defining a functorial construction of self.

EXAMPLES:

```python
defines F, R = CyclotomicField(5).construction()
sage: R
Rational Field
sage: F.polys
[x^4 + x^3 + x^2 + x + 1]
sage: F.names
['zeta5']
sage: F.embeddings
[0.3090169943749487 + 0.9510565162951547*I]
sage: F.structures
[None]
```
```python
sage: C20 = CyclotomicField(20)
sage: C20.different()
Fractional ideal (10, 2*zeta20^6 - 4*zeta20^4 - 4*zeta20^2 + 2)
sage: C18 = CyclotomicField(18)
sage: D = C18.different().norm()
sage: D == C18.discriminant().abs()
True

>>> from sage.all import *

>>> C20 = CyclotomicField(Integer(20))

>>> C20.different()
Fractional ideal (10, 2*zeta20^6 - 4*zeta20^4 - 4*zeta20^2 + 2)

>>> C18 = CyclotomicField(Integer(18))

>>> D = C18.different().norm()

>>> D == C18.discriminant().abs()
True
```

### discriminant ($v$=None)

Return the discriminant of the ring of integers of the cyclotomic field `self`, or if $v$ is specified, the determinant of the trace pairing on the elements of the list $v$.

Uses the formula for the discriminant of a prime power cyclotomic field and Hilbert Theorem 88 on the discriminant of composita.

**INPUT:**
- $v$ – (optional) list of elements of this number field

**OUTPUT:** Integer if $v$ is omitted, and Rational otherwise.

**EXAMPLES:**

```python
sage: CyclotomicField(20).discriminant()
4000000
sage: CyclotomicField(18).discriminant()
-19683

>>> from sage.all import *

>>> CyclotomicField(Integer(20)).discriminant()
4000000

>>> CyclotomicField(Integer(18)).discriminant()
-19683
```

### embeddings ($K$)

Compute all field embeddings of this field into the field $K$.

**INPUT:**
- $K$ – a field

**EXAMPLES:**

```python
sage: CyclotomicField(5).embeddings(ComplexField(53))[1]
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Field with 53 bits of precision
  Defn: zeta5 |--> -0.809016994374947 + 0.587785252292473*I
sage: CyclotomicField(5).embeddings(Qp(11, 4, print_mode='digits'))[1]
```

(expires on next page)
needs sage.rings.padics
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To: 11-adic Field with capped relative precision 4
  Defn: zeta5 |--> ...1525

>>> from sage.all import *
>>> CyclotomicField(Integer(5)).
embeddings(ComplexField(Integer(53)))[Integer(1)]
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To: Complex Field with 53 bits of precision
  Defn: zeta5 |--> -0.809016994374947 + 0.587785252292473*I
>>> CyclotomicField(Integer(5)).embeddings(Qp(Integer(11), Integer(4), print_)
digits))[Integer(1)]  # needs sage.rings.padics
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To: 11-adic Field with capped relative precision 4
  Defn: zeta5 |--> ...1525

is_abelian()
Return True since all cyclotomic fields are automatically abelian.

EXAMPLES:
sage: CyclotomicField(29).is_abelian()
True

>>> from sage.all import *
>>> CyclotomicField(Integer(29)).is_abelian()
True

is_galois()
Return True since all cyclotomic fields are automatically Galois.

EXAMPLES:
sage: CyclotomicField(29).is_galois()
True

>>> from sage.all import *
>>> CyclotomicField(Integer(29)).is_galois()
True

is_isomorphic(other)
Return True if the cyclotomic field self is isomorphic as a number field to other.

EXAMPLES:
sage: CyclotomicField(11).is_isomorphic(CyclotomicField(22))
True
sage: CyclotomicField(11).is_isomorphic(CyclotomicField(23))
False
sage: x = polygen(QQ, 'x')
sage: CyclotomicField(3).is_isomorphic(NumberField(x^2 + x + 1, 'a'))
True
\begin{verbatim}
sage: CyclotomicField(18).is_isomorphic(CyclotomicField(9))
True
sage: CyclotomicField(10).is_isomorphic(NumberField(x^4 - x^3 + x^2 - x + 1, '\rightarrow'\ 'b'))
True
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> CyclotomicField(Integer(11)).is_isomorphic(CyclotomicField(Integer(22)))
True
>>> CyclotomicField(Integer(11)).is_isomorphic(CyclotomicField(Integer(23)))
False
>>> x = polygen(QQ, 'x')
>>> CyclotomicField(Integer(3)).is_isomorphic(NumberField(x^2 + x + 1, 'a'))
True
>>> CyclotomicField(Integer(18)).is_isomorphic(CyclotomicField(Integer(9)))
True
>>> CyclotomicField(Integer(10)).is_isomorphic(NumberField(x^4 - x^3 + x^2 - x + 1, 'b'))
True
\end{verbatim}

Check Issue #14300:
\begin{verbatim}
sage: K = CyclotomicField(4)
sage: N = K.extension(x^2 - 5, 'z')
sage: K.is_isomorphic(N)
False
sage: K.is_isomorphic(CyclotomicField(8))
False
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> K = CyclotomicField(Integer(4))
>>> N = K.extension(x^2 - 5, 'z')
>>> K.is_isomorphic(N)
False
>>> K.is_isomorphic(CyclotomicField(Integer(8)))
False
\end{verbatim}

next_split_prime \((p=2)\)

Return the next prime integer \(p\) that splits completely in this cyclotomic field (and does not ramify).

EXAMPLES:
\begin{verbatim}
sage: K.<z> = CyclotomicField(3)
sage: K.next_split_prime(7)
13
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> K = CyclotomicField(Integer(3), names=('z',)); (z,) = K._first_ngens(1)
>>> K.next_split_prime(Integer(7))
13
\end{verbatim}

number_of_roots_of_unity \()

Return number of roots of unity in this cyclotomic field.
EXAMPLES:

```
sage: K.<a> = CyclotomicField(21)
sage: K.number_of_roots_of_unity()
42
```

```
>>> from sage.all import *
>>> K = CyclotomicField(Integer(21), names=('a',)); (a,) = K._first_ngens(1)
>>> K.number_of_roots_of_unity()
42
```

**real_embeddings**(prec=53)

Return all embeddings of this cyclotomic field into the approximate real field with precision prec.

Mostly, of course, there are no such embeddings.

EXAMPLES:

```
sage: len(CyclotomicField(4).real_embeddings())
0
sage: CyclotomicField(2).real_embeddings()
[
    Ring morphism:
    From: Cyclotomic Field of order 2 and degree 1
    To:   Real Field with 53 bits of precision
    Defn: -1 |--> -1.00000000000000
]
```

```
>>> from sage.all import *
>>> len(CyclotomicField(Integer(4)).real_embeddings())
0
>>> CyclotomicField(Integer(2)).real_embeddings()
[
    Ring morphism:
    From: Cyclotomic Field of order 2 and degree 1
    To:   Real Field with 53 bits of precision
    Defn: -1 |--> -1.00000000000000
]
```

**roots_of_unity()**

Return all the roots of unity in this cyclotomic field, primitive or not.

EXAMPLES:

```
sage: K.<a> = CyclotomicField(3)
sage: zs = K.roots_of_unity(); zs
[1, a, -a - 1, -1, -a, a + 1]
sage: [z**K.number_of_roots_of_unity() for z in zs]
[1, 1, 1, 1, 1, 1]
```

```
>>> from sage.all import *
>>> K = CyclotomicField(Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> zs = K.roots_of_unity(); zs
[1, a, -a - 1, -1, -a, a + 1]
>>> [z**K.number_of_roots_of_unity() for z in zs]
[1, 1, 1, 1, 1, 1]
```
signature()

Return \((r_1, r_2)\), where \(r_1\) and \(r_2\) are the number of real embeddings and pairs of complex embeddings of this cyclotomic field, respectively.

Trivial since, apart from \(\mathbb{Q}\), cyclotomic fields are totally complex.

EXAMPLES:

```python
sage: CyclotomicField(5).signature()
(0, 2)
sage: CyclotomicField(2).signature()
(1, 0)
```

```python
>>> from sage.all import *

>>> CyclotomicField(Integer(5)).signature()
(0, 2)
>>> CyclotomicField(Integer(2)).signature()
(1, 0)
```

```
signature
```

zeta \((n=None, all=False)\)

Return an element of multiplicative order \(n\) in this cyclotomic field.

If there is no such element, raise a ValueError.

INPUT:

- \(n\) – integer (default: None, returns element of maximal order)
- \(all\) – bool (default: False); whether to return a list of all primitive \(n\)-th roots of unity.

OUTPUT: root of unity or list

EXAMPLES:

```python
sage: k = CyclotomicField(4)
sage: k.zeta()
zeta4
sage: k.zeta(2)
-1
sage: k.zeta().multiplicative_order()
4
```

```python
>>> from sage.all import *

>>> k = CyclotomicField(Integer(4))

>>> k.zeta()
zeta4
>>> k.zeta(2)
-1
>>> k.zeta().multiplicative_order()
4
```

```python
sage: k = CyclotomicField(21)
sage: k.zeta().multiplicative_order()
42
sage: k.zeta(21).multiplicative_order()
21
sage: k.zeta(7).multiplicative_order()
7
sage: k.zeta(6).multiplicative_order()
```

(continues on next page)
sage: k.zeta(84)
Traceback (most recent call last):
...  
ValueError: 84 does not divide order of generator (42)

>>> from sage.all import *
>>> k = CyclotomicField(Integer(21))
>>> k.zeta().multiplicative_order()
42
>>> k.zeta(Integer(21)).multiplicative_order()
21
>>> k.zeta(Integer(7)).multiplicative_order()
7
>>> k.zeta(Integer(6)).multiplicative_order()
6
>>> k.zeta(Integer(84))
Traceback (most recent call last):
...  
ValueError: 84 does not divide order of generator (42)

sage: K.<a> = CyclotomicField(7)

sage: K.zeta(all=True)
[-a^4, -a^5, a^5 + a^4 + a^3 + a^2 + a + 1, -a, -a^2, -a^3]

sage: K.zeta(14, all=True)
[-a^4, -a^5, a^5 + a^4 + a^3 + a^2 + a + 1, -a, -a^2, -a^3]

sage: K.zeta(2, all=True)
[-1]

sage: K.<a> = CyclotomicField(10)

sage: K.zeta(20, all=True)
Traceback (most recent call last):
...  
ValueError: 20 does not divide order of generator (10)

>>> from sage.all import *

>>> K = CyclotomicField(Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> K.zeta(all=True)
[-a^4, -a^5, a^5 + a^4 + a^3 + a^2 + a + 1, -a, -a^2, -a^3]

>>> K.zeta(Integer(14), all=True)
[-a^4, -a^5, a^5 + a^4 + a^3 + a^2 + a + 1, -a, -a^2, -a^3]

>>> K.zeta(Integer(2), all=True)
[-1]

>>> K = CyclotomicField(Integer(10), names=('a',)); (a,) = K._first_ngens(1)
>>> K.zeta(Integer(20), all=True)
Traceback (most recent call last):
...  
ValueError: 20 does not divide order of generator (10)

sage: K.<a> = CyclotomicField(5)

sage: K.zeta(4)
Traceback (most recent call last):
...  
ValueError: 4 does not divide order of generator (10)

sage: v = K.zeta(5, all=True);  v
[a, a^2, a^3, -a^3 - a^2 - a - 1]

(continues on next page)
sage: [b^5 for b in v]
[1, 1, 1, 1]

>>> from sage.all import *

>>> K = CyclotomicField(Integer(5), names=('a',)); (a,) = K._first_ngens(1)

>>> K.zeta(Integer(4))
Traceback (most recent call last):
...
ValueError: 4 does not divide order of generator (10)

>>> v = K.zeta(Integer(5), all=True); v
[a, a^2, a^3, -a^3 - a^2 - a - 1]

>>> [b**Integer(5) for b in v]
[1, 1, 1, 1]

zeta_order()

Return the order of the maximal root of unity contained in this cyclotomic field.

EXAMPLES:

sage: CyclotomicField(1).zeta_order()
2
sage: CyclotomicField(4).zeta_order()
4
sage: CyclotomicField(5).zeta_order()
10
sage: CyclotomicField(5)._n()
5
sage: CyclotomicField(389).zeta_order()
778

>>> from sage.all import *

>>> CyclotomicField(Integer(1)).zeta_order()
2
>>> CyclotomicField(Integer(4)).zeta_order()
4
>>> CyclotomicField(Integer(5)).zeta_order()
10
>>> CyclotomicField(Integer(5))._n()
5
>>> CyclotomicField(Integer(389)).zeta_order()
778

sage.rings.number_field.number_field.NumberField_cyclotomic_v1(zeta_order, name, canonical_embedding=None)

Used for unpickling old pickles.

EXAMPLES:

sage: from sage.rings.number_field.number_field import NumberField_cyclotomic_v1
sage: NumberField_cyclotomic_v1(5,'a')
Cyclotomic Field of order 5 and degree 4
sage: NumberField_cyclotomic_v1(5,'a').variable_name()
'a'
class sage.rings.number_field.number_field.NumberField_generic(polynomial, name, latex_name, check=True, embedding=None, category=None, assume_disc_small=False, maximize_at_primes=None, structure=None)

Bases: WithEqualityById, NumberField

Generic class for number fields defined by an irreducible polynomial over \( \mathbb{Q} \).

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 2); K
Number Field in a with defining polynomial x^3 - 2
sage: TestSuite(K).run()

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x^Integer(3) - Integer(2), names=('a',)); (a,) = K._first_ngens(1); K
Number Field in a with defining polynomial x^3 - 2
>>> TestSuite(K).run()
```

\textbf{S\_class\_group} \((S, \text{proof}=\text{None}, \text{names}=\text{c}')\)

Return the \( S \)-class group of this number field over its base field.

INPUT:

- \( S \) – a set of primes of the base field
- \( \text{proof} \) – if False, assume the GRH in computing the class group. Default is True. Call number_field_proof to change this default globally.
- \( \text{names} \) – names of the generators of this class group.

OUTPUT: The \( S \)-class group of this number field.

EXAMPLES:

A well known example:

```python
sage: K.<a> = QuadraticField(-5)
sage: K.S_class_group({})
S-class group of order 2 with structure C2 of Number Field in a with defining polynomial x^2 + 5 with a = 2.236067977499790?I
```
```python
>>> from sage.all import *

>>> K = QuadraticField(-Integer(5), names=('a',)); (a,) = K._first_ngens(1)
K.S_class_group([])
S-class group of order 2 with structure C2 of Number Field in a
with defining polynomial x^2 + 5 with a = 2.236067977499790*I

When we include the prime (2, a + 1), the S-class group becomes trivial:

```python
sage: K.S_class_group([K.ideal(2, a + 1)])
S-class group of order 1 of Number Field in a
with defining polynomial x^2 + 5 with a = 2.236067977499790*I
```

### S_unit_group

(proof=None, S=None)

Return the $S$-unit group (including torsion) of this number field.

**ALGORITHM:** Uses PARI's pari:bnfsunit command.

**INPUT:**

- `proof` – bool (default: True); flag passed to PARI
- `S` – list or tuple of prime ideals, or an ideal, or a single ideal or element from which an ideal can be constructed, in which case the support is used. If None, the global unit group is constructed; otherwise, the $S$-unit group is constructed.

**Note:** The group is cached.

**EXAMPLES:**

```python
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 - 10*x^3 + 20*5*x^2 - 15*5^2*x + 11*5^3)
sage: U = K.S_unit_group(S=a); U
S-unit group with structure C10 x Z x Z x Z of
Number Field in a with defining polynomial x^4 - 10*x^3 + 100*x^2 - 375*x +...
with S = (Fractional ideal (5, 1/275*a^3 + 4/55*a^2 - 5/11*a + 5),
Fractional ideal (11, 1/275*a^3 + 4/55*a^2 - 5/11*a + 9))
sage: U.gens()
(u0, u1, u2, u3)
sage: U.gens_values() # random
[-1/275*a^3 + 7/55*a^2 - 6/11*a + 4, 1/275*a^3 + 4/55*a^2 - 5/11*a + 3,
1/275*a^3 + 4/55*a^2 - 5/11*a + 5, -14/275*a^3 + 21/55*a^2 - 29/11*a + 6]
sage: U.invariants()
(10, 0, 0, 0)
sage: [u.multiplicative_order() for u in U.gens()]
[10, +Infinity, +Infinity, +Infinity]
sage: U.primes()
(Fractional ideal (5, 1/275*a^3 + 4/55*a^2 - 5/11*a + 5),
Fractional ideal (11, 1/275*a^3 + 4/55*a^2 - 5/11*a + 9))
```
Algebraic Numbers and Number Fields, Release 10.4

>>> from sage.all import *

>>> x = polygen(QQ)

>>> K = NumberField(x**Integer(4) - Integer(10)*x**Integer(3) + ...

→Integer(20)*x**Integer(2) - Integer(15)*x**Integer(2) +x...

→+ Integer(11)*x**Integer(2) + Integer(3), names=('a',)); (a,) = K._first_ngens(1)

>>> U = K.S_unit_group(S=a); U

S-unit group with structure C10 x Z x Z x Z of

Number Field in a with defining polynomial x^4 - 10*x^3 + 100*x^2 - 375*x +...

→1375

with S = (Fractional ideal (5, 1/275*a^3 + 4/55*a^2 - 5/11*a + 5),

Fractional ideal (11, 1/275*a^3 + 4/55*a^2 - 5/11*a + 9))

>>> U.gens()

(u0, u1, u2, u3)

>>> U.gens_values()  # random

[-1/275*a^3 + 7/55*a^2 - 6/11*a + 4, 1/275*a^3 + 4/55*a^2 - 5/11*a + 3,

1/275*a^3 + 4/55*a^2 - 5/11*a + 5, -14/275*a^3 + 21/55*a^2 - 29/11*a + 6]

>>> U.invariants()

(10, 0, 0, 0)

>>> [u.multiplicative_order() for u in U.gens()]

[10, +Infinity, +Infinity, +Infinity]

>>> U.primes()

(Fractional ideal (5, 1/275*a^3 + 4/55*a^2 - 5/11*a + 5),

Fractional ideal (11, 1/275*a^3 + 4/55*a^2 - 5/11*a + 9))

With the default value of S, the S-unit group is the same as the global unit group:

sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.unit_group(proof=False)
sage: U.is_isomorphic(K.S_unit_group(proof=False))

True

>>> from sage.all import *

>>> x = polygen(QQ)

>>> K = NumberField(x**Integer(3) + Integer(3), names=('a',)); (a,) = K._first_ngens(1)

>>> U = K.S_unit_group(proof=False)

>>> U.is_isomorphic(K.S_unit_group(proof=False))

True

The value of S may be specified as a list of prime ideals, or an ideal, or an element of the field:

sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=K.ideal(6).prime_factors()); U

S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial x^3 + 3

with S = (Fractional ideal (-a^2 + a - 1),

Fractional ideal (a + 1),

Fractional ideal (a))

sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=K.ideal(6)); U

S-unit group with structure C2 x Z x Z x Z of Number Field in a with defining polynomial x^3 + 3

with S = (Fractional ideal (-a^2 + a - 1),

Fractional ideal (a + 1),

Fractional ideal (a))

(continues on next page)
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=6); U
S-unit group with structure C2 x Z x Z x Z x Z
of Number Field in a with defining polynomial x^3 + 3
with S = (Fractional ideal (-a^2 + a - 1),
Fractional ideal (a + 1),
Fractional ideal (a))
sage: U.primes()
(Fractional ideal (-a^2 + a - 1),
Fractional ideal (a + 1),
Fractional ideal (a))
sage: U.gens()
(u0, u1, u2, u3, u4)
sage: U.gens_values()
[-1, a^2 - 2, -a^2 + a - 1, a + 1, a]

The expandlog method can be used to create $S$-units from sequences of exponents, and recover the exponents:

sage: U.gens_orders()
(2, 0, 0, 0, 0)
sage: u = U.exp((3,1,4,1,5)); u
(continues on next page)
\[-6a^2 + 18a - 54\]

```
sage: u.norm().factor()
sage: u.log(u)
```

```
>>> from sage.all import *
>>> U.gens_orders()
```

```
>>> u = U.exp((Integer(3),Integer(1),Integer(4),Integer(1),Integer(5))); u
```

```
>>>
```

\textbf{S\_unit\_solutions} \((S=[], \text{prec}=106, \text{include\_exponents}=\text{False}, \text{include\_bound}=\text{False}, \text{proof}=\text{None})\)

Return all solutions to the \(S\)-unit equation \(x + y = 1\) over \(self\).

\textbf{INPUT:}

\begin{itemize}
  \item \(S\) – a list of finite primes in this number field
  \item \(\text{prec}\) – precision used for computations in real, complex, and \(p\)-adic fields (default: 106)
  \item \(\text{include\_exponents}\) – whether to include the exponent vectors in the returned value (default: True).
  \item \(\text{include\_bound}\) – whether to return the final computed bound (default: False)
  \item \(\text{proof}\) – if False, assume the GRH in computing the class group. Default is True.
\end{itemize}

\textbf{OUTPUT:}

A list \([(A_1, B_1, x_1, y_1), (A_2, B_2, x_2, y_2), \ldots, (A_n, B_n, x_n, y_n)]\) of tuples such that:

1. The first two entries are tuples \(A_i = (a_0, a_1, \ldots, a_t)\) and \(B_i = (b_0, b_1, \ldots, b_t)\) of exponents. These will be omitted if \(\text{include\_exponents}\) is False.

2. The last two entries are \(S\)-units \(x_i\) and \(y_i\) in \(self\) with \(x_i + y_i = 1\).

3. If the default generators for the \(S\)-units of \(self\) are \((\rho_0, \rho_1, \ldots, \rho_t)\), then these satisfy \(x_i = \prod (\rho_i)^{(a_i)}\) and \(y_i = \prod (\rho_i)^{(b_i)}\).

If \(\text{include\_bound}\) is True, will return a pair \((\text{sols}, \text{bound})\) where \(\text{sols}\) is as above and \(\text{bound}\) is the bound used for the entries in the exponent vectors.

\textbf{EXAMPLES:}

```
sage: # needs sage.rings.padics
sage: x = polygen(QQ, 'x')
sage: K.<xi> = NumberField(x^2 + x + 1)
sage: S = K.primes_above(3)
sage: K.S_unit_solutions(S)  # random, due to ordering
[(xi + 2, -xi - 1), (1/3*xi + 2/3, -1/3*xi + 1/3), (-xi, xi + 1), (-xi + 1, xi), (-xi + 1, -xi)]
```
K = NumberField(x**Integer(2) + x + Integer(1), names=('xi',)); (xi,) = K.
˓
→_first_ngens(1)

S = K.primes_above(Integer(3))

K.S_unit_solutions(S)  # random, due to ordering
[(xi + 2, -xi - 1), (1/3*xi + 2/3, -1/3*xi + 1/3), (-xi, xi + 1), (-xi + 1,␣
˓
→xi)]

You can get the exponent vectors:

sage: K.S_unit_solutions(S, include_exponents=True)  # random, due to ordering
[((2, 1), (4, 0), xi + 2, -xi - 1),
 ((5, -1), (4, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
 ((5, 0), (1, 0), -xi, xi + 1),
 ((1, 1), (2, 0), -xi + 1, xi)]

And the computed bound:

sage: solutions, bound = K.S_unit_solutions(S, prec=100, include_bound=True)
sage: bound
7

And:

sage: from sage.all import *
˓
→# needs sage.rings.padics

sage: K.S_unit_solutions(S, include_exponents=True)  # random, due to ordering
[((2, 1), (4, 0), xi + 2, -xi - 1),
 ((5, -1), (4, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
 ((5, 0), (1, 0), -xi, xi + 1),
 ((1, 1), (2, 0), -xi + 1, xi)]

S_units(S, proof=True)

Return a list of generators of the S-units.

INPUT:

• S – a set of primes of the base field
• proof – if False, assume the GRH in computing the class group

OUTPUT: A list of generators of the unit group.

Note: For more functionality see the function S_unit_group().

EXAMPLES:

sage: K.<a> = QuadraticField(-3)
sage: K.unit_group()
Algebraic Numbers and Number Fields, Release 10.4

Unit group with structure C6 of Number Field in a
with defining polynomial x^2 + 3 with a = 1.732050807568878*I
sage: K.S_units([])  # random
[1/2*a + 1/2]
sage: K.S_units([])[0].multiplicative_order()
6

>>> from sage.all import *
>>> K = QuadraticField(-Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> K.unit_group()
Unit group with structure C6 of Number Field in a
with defining polynomial x^2 + 3 with a = 1.732050807568878*I
>>> K.S_units([])  # random
[1/2*a + 1/2]
>>> K.S_units([])[Integer(0)].multiplicative_order()
6

An example in a relative extension (see Issue #8722):

sage: x = polygen(QQ, 'x')
sage: L.<a,b> = NumberField([x^2 + 1, x^2 - 5])
sage: p = L.ideal((-1/2*b - 1/2)*a + 1/2*b - 1/2)
sage: W = L.S_units([p]); [x.norm() for x in W]
[9, 1, 1]

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> L.<a,b> = NumberField([x^2 + 1, x^2 - 5])
>>> p = L.ideal((-1/2*b - 1/2)*a + 1/2*b - 1/2)
>>> W = L.S_units([p]); [x.norm() for x in W]
[9, 1, 1]

Our generators should have the correct parent (Issue #9367):

sage: _.<x> = QQ[]
sage: L.<alpha> = NumberField(x^3 + x + 1)
sage: p = L.S_units([ L.ideal(7) ])
sage: p[Integer(0)].parent()
Number Field in alpha with defining polynomial x^3 + x + 1

>>> from sage.all import *
>>> _.<x> = QQ['x']; (x,) = _.first_ngens(1)
>>> L = NumberField(x^3 + x + Integer(1), names=('alpha',)); (alpha, →) = L._first_ngens(1)
>>> p = L.S_units([ L.ideal(Integer(7)) ])
>>> p[Integer(0)].parent()
Number Field in alpha with defining polynomial x^3 + x + 1

absolute_degree()

Return the degree of self over Q.

EXAMPLES:
Algebraic Numbers and Number Fields, Release 10.4

```python
sage: x = polygen(QQ, 'x')
sage: NumberField(x^3 + x^2 + 997*x + 1, 'a').absolute_degree()
3
sage: NumberField(x + 1, 'a').absolute_degree()
1
sage: NumberField(x^997 + 17*x + 3, 'a', check=False).absolute_degree()
997
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> NumberField(x**Integer(3) + x**Integer(2) + Integer(997)*x + Integer(1), 'a').absolute_degree()
3
>>> NumberField(x + Integer(1), 'a').absolute_degree()
1
>>> NumberField(x**Integer(997) + Integer(17)*x + Integer(3), 'a', check=False).absolute_degree()
997
```

**absolute_field**(names)

Return self as an absolute number field.

**INPUT:**

- names – string; name of generator of the absolute field

**OUTPUT:**

- K – this number field (since it is already absolute)

Also, K.structure() returns from_K and to_K, where from_K is an isomorphism from K to self and to_K is an isomorphism from self to K.

**EXAMPLES:**

```python
sage: K = CyclotomicField(5)
sage: K.absolute_field(a)
Number Field in a with defining polynomial x^4 + x^3 + x^2 + x + 1
```

**absolute_polynomial_ntl()**

Alias for polynomial_ntl(). Mostly for internal use.

**EXAMPLES:**

```python
sage: K = CyclotomicField(5)
sage: K.absolute_field('a')
Number Field in a with defining polynomial x^4 + x^3 + x^2 + x + 1
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> NumberField(x**Integer(2) + (Integer(2)/Integer(3))*x - Integer(9)/Integer(17), 'a').absolute_polynomial_ntl()
([-27 34 51], 51)
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> NumberField(x**Integer(2) + (Integer(2)/Integer(3))*x - Integer(9)/Integer(17), 'a').absolute_polynomial_ntl()
([-27 34 51], 51)
```
algebraic_closure() 
Return the algebraic closure of self (which is QQbar).

EXAMPLES:
```python
sage: K.<i> = QuadraticField(-1)
sage: K.algebraic_closure()
Algebraic Field
sage: i = polygen(QQ, 'i')
sage: K.<i> = NumberField(i^2 - 2)
sage: K.algebraic_closure()
Algebraic Field
sage: K = CyclotomicField(23)
sage: K.algebraic_closure()
Algebraic Field
```

change_generator(alpha, name=None, names=None)
Given the number field self, construct another isomorphic number field \( K \) generated by the element \( \alpha \) of self, along with isomorphisms from \( K \) to self and from self to \( K \).

EXAMPLES:
```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> L.<i> = NumberField(x^2 + 1); L
Number Field in i with defining polynomial x^2 + 1
sage: K, from_K, to_K = L.change_generator(i/2 + 3)
sage: K
Number Field in i0 with defining polynomial x^2 - 6*x + 37/4 with i0 = 1/2*i0 + 3
sage: from_K
Ring morphism:
  From: Number Field in i0 with defining polynomial x^2 - 6*x + 37/4 with i0 = 1/2*i0 + 3
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i0 |--> 2*i0 - 6
sage: to_K
Ring morphism:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Number Field in i0 with defining polynomial x^2 - 6*x + 37/4 with i0 = 1/2*i0 + 3
  Defn: i |--> 2*i0 - 6
```

>> from sage.all import *
>> x = polygen(ZZ, 'x')
>>> L = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = L._first_ngens(1); L
Number Field in i with defining polynomial x^2 + 1
>>> K, from_K, to_K = L.change_generator(i/Integer(2) + Integer(3))
>>> K
Number Field in i0 with defining polynomial x^2 - 6*x + 37/4 with i0 = 1/2*i + 3
>>> from_K
Ring morphism:
  From: Number Field in i0 with defining polynomial x^2 - 6*x + 37/4 with i0 = 1/2*i + 3
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i0 |--> 1/2*i + 3
>>> to_K
Ring morphism:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Number Field in i0 with defining polynomial x^2 - 6*x + 37/4 with i0 = 1/2*i + 3
  Defn: i |--> 2*i0 - 6

We can also do

sage: K.<c>, from_K, to_K = L.change_generator(i/2 + 3); K
Number Field in c with defining polynomial x^2 - 6*x + 37/4 with c = 1/2*i + 3

>>> from sage.all import *

>>> K, from_K, to_K = L.change_generator(i/Integer(2) + Integer(3), names=('c',)); (c,) = K._first_ngens(1); K
Number Field in c with defining polynomial x^2 - 6*x + 37/4 with c = 1/2*i + 3

We compute the image of the generator $\sqrt{-1}$ of $L$.

sage: to_K(i)
2*c - 6

>>> from sage.all import *

>>> to_K(i)
2*c - 6

Note that the image is indeed a square root of $-1$.

sage: to_K(i)**Integer(2)
-1
sage: from_K(to_K(i))
i
sage: to_K(from_K(c))
c

>>> from sage.all import *

>>> to_K(i)**Integer(2)
-1
>>> from_K(to_K(i))
i
>>> to_K(from_K(c))
c
characteristic()

Return the characteristic of this number field, which is of course 0.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^99 + 2); k
Number Field in a with defining polynomial x^99 + 2
sage: k.characteristic()
0
```}

```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> k = NumberField(x**Integer(99) + Integer(2), names=('a',)); (a,) = k._first_ngens(1)
Number Field in a with defining polynomial x^99 + 2
>>> k.characteristic()
0
```}

class_group(proof=None, names='c')

Return the class group of the ring of integers of this number field.

INPUT:

- `proof` – if True then compute the class group provably correctly. Default is True. Call `number_field_proof()` to change this default globally.
- `names` – names of the generators of this class group.

OUTPUT: The class group of this number field.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 + 23)
sage: G = K.class_group(); G
Class group of order 3 with structure C3 of Number Field in a with defining polynomial x^2 + 23
sage: G.0
Fractional ideal class (2, 1/2*a - 1/2)
sage: G.gens()
(Fractional ideal class (2, 1/2*a - 1/2),)
```}

```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(23), names=('a',)); (a,) = K._first_ngens(1)
>>> G = K.class_group(); G
Class group of order 3 with structure C3 of Number Field in a with defining polynomial x^2 + 23
>>> G.gen(0)
Fractional ideal class (2, 1/2*a - 1/2)
>>> G.gens()
(Fractional ideal class (2, 1/2*a - 1/2),)
```}

```
sage: G.number_field()
Number Field in a with defining polynomial x^2 + 23
sage: G is K.class_group()
```
True

```python
sage: G is K.class_group(proof=False)
False
```

```python
sage: G.gens()
(Fractional ideal class (2, 1/2*a - 1/2),)
```

```python
>>> from sage.all import *
```

```python
>>> G.number_field()
Number Field in a with defining polynomial x^2 + 23
```

```python
>>> G.is_K.class_group()  # random
True
```

```python
>>> G.is_K.class_group(proof=False)  # random
False
```

```python
>>> G.gens()
(Fractional ideal class (2, 1/2*a - 1/2),)
```

There can be multiple generators:

```python
sage: k.<a> = NumberField(x^2 + 20072)
sage: G = k.class_group(); G
Class group of order 76 with structure C38 x C2 of
Number Field in a with defining polynomial x^2 + 20072
```

```python
sage: G.0  # random
Fractional ideal class (41, a + 10)
sage: G.0^38
Trivial principal fractional ideal class
```

```python
sage: G.1  # random
Fractional ideal class (2, -1/2*a)
sage: G.1^2
Trivial principal fractional ideal class
```

```python
>>> from sage.all import *
```

```python
>>> k = NumberField(x**Integer(2) + Integer(20072), names=(a,)); (a,) = k._first_ngens(1)
```

```python
>>> G = k.class_group(); G
Class group of order 76 with structure C38 x C2 of
Number Field in a with defining polynomial x^2 + 20072
```

```python
>>> G.gen(0)  # random
Fractional ideal class (41, a + 10)
```

```python
>>> G.gen(0)**Integer(38)
Trivial principal fractional ideal class
```

```python
>>> G.gen(1)  # random
Fractional ideal class (2, -1/2*a)
```

```python
>>> G.gen(1)**Integer(2)
Trivial principal fractional ideal class
```

Class groups of Hecke polynomials tend to be very small:

```python
# needs sage.modular
sage: f = ModularForms(97, 2).T(2).charpoly()
sage: f.factor()
(x - 3) * (x^3 + 4*x^2 + 3*x - 1) * (x^4 - 3*x^3 - x^2 + 6*x - 1)
sage: [NumberField(g,'a').class_group().order() for g, _ in f.factor()]
[1, 1, 1]
```
>>> from sage.all import *
>>> # needs sage.modular
>>> f = ModularForms(Integer(97), Integer(2)).T(Integer(2)).charpoly()
>>> f.factor()
(x - 3) * (x^3 + 4*x^2 + 3*x - 1) * (x^4 - 3*x^3 - x^2 + 6*x - 1)
>>> [NumberField(g,'a').class_group().order() for g, _ in f.factor()]
[1, 1, 1]

Note: Unlike in PARI/GP, class group computations in Sage do not by default assume the Generalized Riemann Hypothesis. To do class groups computations not provably correctly you must often pass the flag proof=False to functions or call the function proof.number_field(False). It can easily take 1000's of times longer to do computations with proof=True (the default).

class_number(proof=None)
Return the class number of this number field, as an integer.

INPUT:
- proof - bool (default: True unless you called number_field_proof)

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + 23, 'a').class_number()
3
sage: NumberField(x^2 + 163, 'a').class_number()
1
sage: NumberField(x^3 + x^2 + 997*x + 1, 'a').class_number(proof=False)
1539
```

```
from sage.all import *

>>> x = polygen(QQ, 'x')

>>> NumberField(x**Integer(2) + Integer(23), 'a').class_number()
3
>>> NumberField(x**Integer(2) + Integer(163), 'a').class_number()
1
>>> NumberField(x**Integer(3) + x**Integer(2) + Integer(997)*x + Integer(1), 'a').class_number(proof=False)
1539
```

completely_split_primes(B=200)
Return a list of rational primes which split completely in the number field \( K \).

INPUT:
- \( B \) - a positive integer bound (default: 200)

OUTPUT:
A list of all primes \( p < B \) which split completely in \( K \).

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: K.<xi> = NumberField(x^3 - 3*x + 1)
sage: K.completely_split_primes(100)
[17, 19, 37, 53, 71, 73, 89]
```

1.1. Number fields 79
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3)*x + Integer(1), names=('xi',));
→ (xi,) = K._first_ngens(1)
>>> K.completely_split_primes(Integer(100))
[17, 19, 37, 53, 71, 73, 89]

complex_conjugation()  
Return the complex conjugation of self.

    This is only well-defined for fields contained in CM fields (i.e. for totally real fields and CM fields). Recall that
    a CM field is a totally imaginary quadratic extension of a totally real field. For other fields, a \texttt{ValueError}
    is raised.

    EXAMPLES:

    \begin{verbatim}
    sage: QuadraticField(-1, 'I').complex_conjugation()  
    Ring endomorphism of  
        Number Field in I with defining polynomial \(x^2 + 1\) with I = 1*I  
            Defn: I |--> -I
    sage: CyclotomicField(8).complex_conjugation()  
    Ring endomorphism of Cyclotomic Field of order 8 and degree 4  
            Defn: zeta8 |--> -zeta8^3
    sage: QuadraticField(5, 'a').complex_conjugation()  
    Identity endomorphism of Number Field in a with defining  
        polynomial \(x^2 - 5\) with a = 2.2360679774997990?
    \end{verbatim}
sage: F.complex_conjugation()
Identity endomorphism of Number Field in a with defining polynomial x^4 + x^3 - 3*x^2 - x + 1
sage: F.<b> = NumberField(x^2 - 2)

sage: F.extension(x^2 + 1, 'a').complex_conjugation()
Relative number field endomorphism of Number Field in a with defining polynomial x^2 + 1 over its base field
   Defn: a |--> -a
   b |--> b
sage: F2.<b> = NumberField(x^2 + 2)

sage: K2.<a> = F2.extension(x^2 + 1)

sage: cc = K2.complex_conjugation()
sage: cc(a)
-a
sage: cc(b)
-b

>>> from sage.all import *

>>> QuadraticField(-Integer(1), 'I').complex_conjugation()
Ring endomorphism of Number Field in I with defining polynomial x^2 + 1 with I = 1*I
   Defn: I |--> -I

>>> CyclotomicField(Integer(8)).complex_conjugation()
Ring endomorphism of Cyclotomic Field of order 8 and degree 4
   Defn: zeta8 |--> -zeta8^3

>>> QuadraticField(Integer(5), 'a').complex_conjugation()
Identity endomorphism of Number Field in a with defining polynomial x^2 - 5 with a = 2.236067977499790?

>>> x = polygen(QQ, 'x')

>>> F = NumberField(x**Integer(4) + x**Integer(3) - Integer(3)*x**Integer(2) - x + Integer(1), a)

>>> F.is_totally_real()
True

>>> F.complex_conjugation()
Identity endomorphism of Number Field in a with defining polynomial x^4 + x^3 - 3*x^2 - x + 1

>>> F = NumberField(x**Integer(2) - Integer(2), names=(b,)); (b,) = F._first_ngens(1)

>>> F.extension(x**Integer(2) + Integer(1), 'a').complex_conjugation()
Relative number field endomorphism of Number Field in a with defining polynomial x^2 + 1 over its base field
   Defn: a |--> -a
   b |--> b

>>> F2 = NumberField(x**Integer(2) + Integer(2), names=(b,)); (b,) = F2._first_ngens(1)

>>> K2 = F2.extension(x**Integer(2) + Integer(1), names=('a',)); (a,) = K2._first_ngens(1)

>>> cc = K2.complex_conjugation()

complex_embeddings (prec=53)

Return all homomorphisms of this number field into the approximate complex field with precision prec.

This always embeds into an MPFR based complex field. If you want embeddings into the 53-bit double
precision, which is faster, use `self.embeddings(CDF)`.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^5 + x + 17)
sage: v = k.complex_embeddings()
sage: ls = [phi(k.0^2) for phi in v]; ls # random order
[2.97572074038..., -2.40889943716 + 1.90254105304*I, -2.40889943716 - 1.90254105304*I, 0.921039066973 + 3.07553311885*I, 0.921039066973 - 3.07553311885*I]
sage: K.<a> = NumberField(x^3 + 2)
sage: ls = K.complex_embeddings(); ls # random order
[Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> -1.25992104989...,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> 0.629960524947 - 1.09112363597*I,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> 0.629960524947 + 1.09112363597*I]
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> k = NumberField(x**Integer(5) + x + Integer(17), names=('a',)); (a,) = k._first_ngens(1)
>>> v = k.complex_embeddings()
>>> ls = [phi(k.gen(0)**Integer(2)) for phi in v]; ls # random order
[2.97572074038..., -2.40889943716 + 1.90254105304*I, -2.40889943716 - 1.90254105304*I, 0.921039066973 + 3.07553311885*I, 0.921039066973 - 3.07553311885*I]
```

```python
>>> K = NumberField(x**Integer(3) + Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> ls = K.complex_embeddings(); ls # random order
[Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> -1.25992104989...,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> 0.629960524947 - 1.09112363597*I,
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Complex Double Field
  Defn: a |--> 0.629960524947 + 1.09112363597*I]
```
```
composite_fields(other, names=None, both_maps=False, preserve_embedding=True)
```

Return the possible composite number fields formed from `self` and `other`.

**INPUT:**
- `other` – number field
- `names` – generator name for composite fields
- `both_maps` – boolean (default: False)
- `preserve_embedding` – boolean (default: True)

**OUTPUT:**
A list of the composite fields, possibly with maps.

If `both_maps` is `True`, the list consists of quadruples `(F, self_into_F, other_into_F, k)` such that `self_into_F` is an embedding of `self` in `F`, `other_into_F` is an embedding of `other` in `F`, and `k` is one of the following:
- an integer such that `F.gen()` equals `other_into_F(other.gen()) + k*self_into_F(self.gen())`;
- `Infinity`, in which case `F.gen()` equals `self_into_F(self.gen())`;
- `None` (when `other` is a relative number field).

If both `self` and `other` have embeddings into an ambient field, then each `F` will have an embedding with respect to which both `self_into_F` and `other_into_F` will be compatible with the ambient embeddings.

If `preserve_embedding` is `True` and if `self` and `other` both have embeddings into the same ambient field, or into fields which are contained in a common field, only the compositum respecting both embeddings is returned. In all other cases, all possible composite number fields are returned.

**EXAMPLES:**
```
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^4 - 2)
sage: K.composite_fields(K)
[Number Field in a with defining polynomial x^4 - 2,
 Number Field in a0 with defining polynomial x^8 + 28*x^4 + 2500]
```
```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> K.composite_fields(K)
[Number Field in a with defining polynomial x^4 - 2,
 Number Field in a0 with defining polynomial x^8 + 28*x^4 + 2500]
```

A particular compositum is selected, together with compatible maps into the compositum, if the fields are endowed with a real or complex embedding:
```
sage: # needs sage.symbolic
sage: K1 = NumberField(x^4 - 2, 'a', embedding=RR(2^(1/4)))
sage: K2 = NumberField(x^4 - 2, 'a', embedding=RR(-2^(1/4)))
sage: K1.composite_fields(K2)
[Number Field in a with defining polynomial x^4 - 2 with a = 1.8920711500272272?]
sage: [F, f, g, k], = K1.composite_fields(K2, both_maps=True); F
```

(continues on next page)
Number Field in a with defining polynomial \( x^4 - 2 \) with \( a = 1.189207115002722 \)?

```
sage: f(K1.0), g(K2.0)
(a, -a)
```

```python
>>> from sage.all import *
# needs sage.symbolic
>>>
K1 = NumberField(x**Integer(4) - Integer(2), 'a',
                   embedding=RR(Integer(2)**(Integer(1)/Integer(4))))
>>> K2 = NumberField(x**Integer(4) - Integer(2), 'a', embedding=RR(-
                   Integer(2)**(Integer(1)/Integer(4))))
>>> K1.composite_fields(K2)
[Number Field in a with defining polynomial \( x^4 - 2 \) with \( a = 1.189207115002722 \)?]
>>> [F, f, g, k], = K1.composite_fields(K2, both_maps=True); F
Number Field in a with defining polynomial \( x^4 - 2 \) with \( a = 1.189207115002722 \)?
>>> f(K1.gen(0)), g(K2.gen(0))
(a, -a)
```

With `preserve_embedding` set to `False`, the embeddings are ignored:

```
sage: K1.composite_fields(K2, preserve_embedding=False)  # needs sage.symbolic
[Number Field in a with defining polynomial \( x^4 - 2 \) with \( a = 1.189207115002722 \)?,
 Number Field in a0 with defining polynomial \( x^8 + 28*x^4 + 2500 \)]
```

Changing the embedding selects a different compositum:

```
sage: K3 = NumberField(x**4 - 2, 'a', embedding=CC(2**(1/4)*I))  # needs sage.symbolic
sage: [F, f, g, k], = K1.composite_fields(K3, both_maps=True); F  # needs sage.symbolic
Number Field in a with defining polynomial \( x^8 + 28*x^4 + 2500 \) with \( a0 = -2.378414230005443? + 1.189207115002722?*I \)
>>> f(K1.0), g(K3.0)  # needs sage.symbolic
(1/240*a0^5 - 41/120*a0, 1/120*a0^5 + 19/60*a0)
```

```python
>>> from sage.all import *
# needs sage.symbolic
>>>
K3 = NumberField(x**Integer(4) - Integer(2), 'a',
                   embedding=CC(Integer(2)**(Integer(1)/Integer(4))*I))
# needs sage.symbolic
>>> [F, f, g, k], = K1.composite_fields(K3, both_maps=True); F
# needs sage.symbolic
Number Field in a0 with defining polynomial \( x^8 + 28*x^4 + 2500 \) with \( a0 = -2.378414230005443? + 1.189207115002722?*I \)
>>> f(K1.gen(0)), g(K3.gen(0))
# needs sage.symbolic
(1/240*a0^5 - 41/120*a0, 1/120*a0^5 + 19/60*a0)
```
If no embeddings are specified, the maps into the compositum are chosen arbitrarily:

```
sage: Q1.<a> = NumberField(x^4 + 10*x^2 + 1)
sage: Q2.<b> = NumberField(x^4 + 16*x^2 + 4)
sage: Q1.composite_fields(Q2, c)
[Number Field in c with defining polynomial
 x^8 + 64*x^6 + 904*x^4 + 3840*x^2 + 3600]
sage: F, Q1_into_F, Q2_into_F, k = Q1.composite_fields(Q2, c, both_maps= True)[0]
sage: Q1_into_F
Ring morphism:
  From: Number Field in a with defining polynomial x^4 + 10*x^2 + 1
  To:   Number Field in c with defining polynomial
         x^8 + 64*x^6 + 904*x^4 + 3840*x^2 + 3600
  Defn: a |--> 19/14400*c^7 + 137/1800*c^5 + 2599/3600*c^3 + 8/15*c
>>> from sage.all import *
>>> Q1 = NumberField(x**Integer(4) + Integer(10)*x**Integer(2) + Integer(1),␣
    →    names=('a',)); (a,) = Q1._first_ngens(1)
>>> Q2 = NumberField(x**Integer(4) + Integer(16)*x**Integer(2) + Integer(4),␣
    →    names=('b',)); (b,) = Q2._first_ngens(1)
>>> Q1.composite_fields(Q2, 'c')
[Number Field in c with defining polynomial
 x^8 + 64*x^6 + 904*x^4 + 3840*x^2 + 3600]
>>> F, Q1_into_F, Q2_into_F, k = Q1.composite_fields(Q2, 'c', both_ maps= True)[Integer(0)]
>>> Q1_into_F
Ring morphism:
  From: Number Field in a with defining polynomial x^4 + 10*x^2 + 1
  To:   Number Field in c with defining polynomial
         x^8 + 64*x^6 + 904*x^4 + 3840*x^2 + 3600
  Defn: a |--> 19/14400*c^7 + 137/1800*c^5 + 2599/3600*c^3 + 8/15*c
```

This is just one of four embeddings of Q1 into F:

```
sage: Hom(Q1, F).order()
4
```

Note that even with preserve_embedding=True, this method may fail to recognize that the two number fields have compatible embeddings, and hence return several composite number fields:

```
sage: x = polygen(ZZ)
sage: A.<a> = NumberField(x^3 - 7, embedding=CC(-0.95+1.65*I))
sage: x = QQbar.polynomial_root(x^9 - 7, RIF(1.2, 1.3))
sage: B.<a> = NumberField(x^9 - 7, embedding=r)
sage: len(A.composite_fields(B, preserve_embedding= True))
2
```

```
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> A = NumberField(x**Integer(3) - Integer(7), embedding=CC(-RealNumber('0.95
    →    ')+RealNumber('1.65')*I), names=('a',)); (a,) = A._first_ngens(1)
(continues on next page)
Algebraic Numbers and Number Fields, Release 10.4

(continued from previous page)

```python
>>> r = QQbar.polynomial_root(x**Integer(9) - Integer(7), RIF(RealNumber('1.2 ˓→1.3')))  
>>> B = NumberField(x**Integer(9) - Integer(7), embedding=r, names=('a',)); ...
˓→(a,) = B._first_ngens(1)  
>>> len(A.composite_fields(B, preserve_embedding=True))
2
```

**conductor** *(check_abelian=True)*

Computes the conductor of the abelian field $K$. If *check_abelian* is set to *False* and the field is not an abelian extension of $Q$, the output is not meaningful.

**INPUT:**

- *check_abelian* – a boolean (default: True); check to see that this is an abelian extension of $Q$

**OUTPUT:**

Integer which is the conductor of the field.

**EXAMPLES:**

```python
sage: # needs sage.groups
sage: K = CyclotomicField(27)
sage: k = K.subfields(9)[0][0]
sage: k.conductor()
27
sage: x = polygen(QQ, 'x')
sage: K.<t> = NumberField(x^3 + x^2 - 2*x - 1)
sage: K.conductor()
7
sage: K.<t> = NumberField(x^3 + x^2 - 36*x - 4)
sage: K.conductor()
109
sage: K = CyclotomicField(48)
sage: k = K.subfields(16)[0][0]
sage: k.conductor()
48
sage: NumberField(x, 'a').conductor()
1
sage: NumberField(x^8 - 8*x^6 + 19*x^4 - 12*x^2 + 1, 'a').conductor()
40
sage: NumberField(x^8 + 7*x^4 + 1, 'a').conductor()
40
sage: NumberField(x^8 - 40*x^6 + 500*x^4 - 2000*x^2 + 50, 'a').conductor()
160
```

```python
>>> from sage.all import *

>>> K = CyclotomicField(Integer(27))
>>> k = K.subfields(Integer(9))[Integer(0)][Integer(0)]
>>> k.conductor()
27
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(3) + x**Integer(2) - Integer(2)*x - Integer(1), ...
˓→names=('t',)); (t,) = K._first_ngens(1)
>>> K.conductor()
7
>>> K = NumberField(x**Integer(3) + x**Integer(2) - Integer(36)*x -...
```

(continues on next page)
ALGORITHM:

For odd primes, it is easy to compute from the ramification index because the $p$-Sylow subgroup is cyclic. For $p = 2$, there are two choices for a given ramification index. They can be distinguished by the parity of the exponent in the discriminant of a $2$-adic completion.

construction()

Construction of self.

EXAMPLES:

```sage
x = polygen(ZZ, 'x')
K.<a> = NumberField(x^3 + x^2 + 1, embedding=CC.gen())
F, R = K.construction()
F
AlgebraicExtensionFunctor
R
Rational Field```

```sage
def from sage.all import *
x = polygen(ZZ, 'x')
K = NumberField(x**Integer(3) + x**Integer(2) + Integer(1), embedding=CC.
    gen(), names=('a',)); (a,) = K._first_ngens(1)
F, R = K.construction()
F
AlgebraicExtensionFunctor
R
Rational Field```

The construction functor respects distinguished embeddings:

```sage
F(R) is K
True
F.embeddings
[0.2327856159383841? + 0.7925519925154479?*I]```
decomposition_type$(p)$

Return how the given prime of the base fields splits in this number field.

**INPUT:**

- $p$ – a prime element or ideal of the base field.

**OUTPUT:**

A list of triples $(e, f, g)$ where

- $e$ is the ramification index,
- $f$ is the residue class degree,
- $g$ is the number of primes above $p$ with given $e$ and $f$

**EXAMPLES:**

\begin{verbatim}
>>> from sage.all import *
>>> F(R)  is K
True
>>> F.embeddings
[[0.2327856159383841? + 0.7925519925154479?*I]

sage: R.<x> = ZZ[]
sage: K.<a> = NumberField(x^20 + 3*x^18 + 15*x^16 + 28*x^14 + 237*x^12 + 579*x^10 + 1114*x^8 + 1470*x^6 + 2304*x^4 + 1296*x^2 + 729)
sage: K.is_galois()  # needs sage.groups
True
sage: K.discriminant().factor()
2^20 * 3^10 * 53^10
sage: K.decomposition_type(2)
[(2, 5, 2)]
sage: K.decomposition_type(3)
[(2, 1, 10)]
sage: K.decomposition_type(53)
[(2, 2, 5)]

(continues on next page)
\end{verbatim}
This example is only ramified at 11:

```python
sage: K.<a> = NumberField(x^24 + 11^2*(90*x^12 - 640*x^8 + 2280*x^6
....: - 512*x^4 + 2432/11*x^2 - 11))
```

```python
sage: K.discriminant().factor()
-1 * 11^43
```

```python
sage: K.decomposition_type(11)
[(1, 1, 2), (22, 1, 1)]
```

Computing the decomposition type is feasible even in large degree:

```python
sage: K.<a> = NumberField(x^144 + 123*x^72 + 321*x^36 + 13*x^18 + 11)
```

```python
sage: K.discriminant().factor(limit=100000)
2^144 * 3^288 * 7^18 * 11^17 * 157^18 * 193^18 * 13907^18 * ...
```

```python
sage: K.decomposition_type(2)
[(2, 4, 3), (2, 12, 2), (2, 36, 1)]
```

```python
sage: K.decomposition_type(3)
[(9, 3, 2), (9, 10, 1)]
```

```python
sage: K.decomposition_type(7)
[(1, 18, 1), (1, 90, 1), (2, 1, 6), (2, 3, 4)]
```

It also works for relative extensions:

```python
sage: K.<a> = QuadraticField(-143)
```

```python
sage: M.<c> = K.extension(x^10 - 6*x^8 + (a + 12)*x^6 + (-7/2*a - 89/2)*x^4
....: + (13/2*a - 77/2)*x^2 + 25])
```
There is a unique prime above 11 and above 13 in $K$, each of which is unramified in $M$:

```python
sage: M.decomposition_type(11)
[(1, 2, 5)]
sage: P11 = K.primes_above(11)[0]
sage: len(M.primes_above(P11))
5
sage: M.decomposition_type(13)
[(1, 1, 10)]
sage: P13 = K.primes_above(13)[0]
sage: len(M.primes_above(P13))
10
```

There are two primes above 2, each of which ramifies in $M$:

```python
sage: Q0, Q1 = K.primes_above(2)
sage: M.decomposition_type(Q0)
[(2, 5, 1)]
sage: q0 = M.primes_above(Q0)
sage: q0.residue_class_degree()
5
sage: q0.relative_ramification_index()
2
sage: M.decomposition_type(Q1)
[(2, 5, 1)]
```

(continues on next page)
Check that Issue #34514 is fixed:

```python
sage: K.<a> = NumberField(x^4 + 18*x^2 - 1)
sage: R.<y> = K[]
sage: L.<b> = K.extension(y^2 + 9*a^3 - 2*a^2 + 162*a - 38)
sage: [L.decomposition_type(i) for i in K.primes_above(3)]
```

```python
[(1, 1, 2), (1, 1, 2), (1, 2, 1)]
```

The `decomposition_type()` function returns a list of tuples, where each tuple represents the type of the extension at a prime ideal. The first element of the tuple is the degree of the extension, the second element is the ramification index, and the third element is the inertia degree.

**defining_polynomial()**

Return the defining polynomial of this number field.

This is exactly the same as `polynomial()`.

**degree()**

Return the degree of this number field.
sage: NumberField(x + 1, 'a').degree()
1
sage: NumberField(x^997 + 17*x + 3, 'a', check=False).degree()
997

>>> from sage.all import *

sage: x = polygen(QQ, x)
sage: k.<a> = NumberField(x**Integer(3) + x**Integer(2) + Integer(997)*x + Integer(1), 'a').degree()
3
sage: NumberField(x + Integer(1), 'a').degree()
1
sage: NumberField(x**Integer(997) + Integer(17)*x + Integer(3), 'a', check=False).degree()
997

different()

Compute the different fractional ideal of this number field.

The codifferent is the fractional ideal of all $x$ in $K$ such that the trace of $xy$ is an integer for all $y \in O_K$.

The different is the integral ideal which is the inverse of the codifferent.

See Wikipedia article Different_ideal

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^2 + 23)
sage: d = k.different()
sage: d
Fractional ideal (-a)
sage: d.norm()
23
sage: k.disc()
-23

>>> from sage.all import *

sage: x = polygen(QQ, 'x')
sage: k = NumberField(x**Integer(2) + Integer(23), names=('a',)); (a,) = k._first_ngens(1)
sage: d = k.different()
sage: d
Fractional ideal (-a)
sage: d.norm()
23
sage: k.disc()
-23

The different is cached:

sage: d is k.different()
True

>>> from sage.all import *

sage: d is k.different()
True
Another example:

```python
sage: k.<b> = NumberField(x^2 - 123)
sage: d = k.different(); d
Fractional ideal (2*b)
sage: d.norm()
492
sage: k.disc()
492
```

```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K.<t> = NumberField(x^3 + x^2 - 36*x - 4)
>>> K.conductor()
109
>>> K.dirichlet_group()  # optional - gap_package_polycyclic
[Dirichlet character modulo 109 of conductor 1 mapping 6 |--> 1,
  Dirichlet character modulo 109 of conductor 109 mapping 6 |--> zeta3,
  Dirichlet character modulo 109 of conductor 109 mapping 6 |--> -zeta3 - 1]
```

```python
sage: # needs sage.groups sage.modular
sage: x = polygen(QQ, 'x')
```

1.1. Number fields 93
K = NumberField(x**Integer(3) + x**Integer(2) - Integer(36)*x - Integer(4), names=('t',)); (t,) = K._first_ngens(1)
K.conductor()
109
K.dirichlet_group()  # optional - gap_package_polycyclic
[Dirichlet character modulo 109 of conductor 1 mapping 6 |--> 1,
 Dirichlet character modulo 109 of conductor 109 mapping 6 |--> zeta3,
 Dirichlet character modulo 109 of conductor 109 mapping 6 |--> -zeta3 - 1]

# needs sage.modular
K = CyclotomicField(Integer(44))
L = K.subfields(Integer(5))[Integer(0)][Integer(0)]
X = L.dirichlet_group(); X  # optional - gap_package_polycyclic
[Dirichlet character modulo 11 of conductor 1 mapping 2 |--> 1,
 Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5,
 Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5^2,
 Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5^3,
 Dirichlet character modulo 11 of conductor 11
 mapping 2 |--> -zeta5^3 - zeta5^2 - zeta5 - 1]
X[Integer(4)]**Integer(2)  # optional - gap_package_polycyclic
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5^3
X[Integer(4)]**Integer(2) in X  # optional - gap_package_polycyclic
True

disc (v=None)
Shortcut for discriminant().

EXAMPLES:
sage: x = polygen(QQ, 'x')
sage: k.<b> = NumberField(x^2 - 123)
sage: k.disc()
492

from sage.all import *

sage: x = polygen(QQ, 'x')
sage: k = NumberField(x**Integer(2) - Integer(123), names=('b',)); (b,) = k._first_ngens(1)
sage: k.disc()
492

discriminant (v=None)
Return the discriminant of the ring of integers of the number field, or if v is specified, the determinant of the trace pairing on the elements of the list v.

INPUT:
• v – (optional) list of elements of this number field

OUTPUT:
Integer if v is omitted, and Rational otherwise.

EXAMPLES:
sage: x = polygen(QQ, 'x')
sage: K.<t> = NumberField(x^3 + x^2 - 2*x + 8)
sage: K.disc()
elements_of_norm\( (n, proof=None) \)

Return a list of elements of norm \( n \).

INPUT:

- \( n \) – integer
- \( proof \) – boolean (default: True, unless you called proof.number_field() and set it otherwise)

OUTPUT:

A complete system of integral elements of norm \( n \), modulo units of positive norm.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')

sage: K.<a> = NumberField(x^2 + 1)
sage: K.elements_of_norm(3)
[]
sage: K.elements_of_norm(50)
[-a - 7, 5*a - 5, 7*a + 1]
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> K.elements_of_norm(Integer(3))
[]
>>> K.elements_of_norm(Integer(50))
[-a - 7, 5*a - 5, 7*a + 1]
```
extension (poly, name=None, names=None, latex_name=None, latex_names=None, *args, **kwds)

Return the relative extension of this field by a given polynomial.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^3 - 2)
sage: R.<t> = K[]
sage: L.<b> = K.extension(t^2 + a); L
Number Field in b with defining polynomial t^2 + a over its base field
```

We create another extension:

```
sage: k.<a> = NumberField(x^2 + 1); k
Number Field in a with defining polynomial x^2 + 1
sage: y = polygen(QQ,y)
sage: m.<b> = k.extension(y^2 + 2); m
Number Field in b with defining polynomial y^2 + 2 over its base field
```

Note that \( b \) is a root of \( y^2 + 2 \):

```
sage: b.minpoly()
x^2 + 2
sage: b.minpoly('z')
z^2 + 2
```

A relative extension of a relative extension:

```
sage: k.<a> = NumberField([x^2 + 1, x^3 + x + 1])
sage: R.<z> = k[]
sage: L.<b> = NumberField(z^3 + 3 + a); L
Number Field in b with defining polynomial z^3 + a0 + 3 over its base field
```
Extension fields with given defining data are unique (Issue #20791):

```python
sage: K.<a> = NumberField(x^2 + 1)
sage: K.extension(x^2 - 2, 'b') is K.extension(x^2 - 2, 'b')
True
```

**factor(n)**

Ideal factorization of the principal ideal generated by \( n \).

**EXAMPLES:**

Here we show how to factor Gaussian integers (up to units). First we form a number field defined by \( x^2 + 1 \):

```python
sage: x = polygen(QQ, 'x')
sage: K.<I> = NumberField(x^2 + 1); K
Number Field in I with defining polynomial x^2 + 1
```

Here are the factors:

```python
sage: fi, fj = K.factor(17); fi,fj
((Fractional ideal (I + 4), 1), (Fractional ideal (I - 4), 1))
```

Now we extract the reduced form of the generators:

```python
sage: zi = fi[0].gens_reduced()[0]; zi
I + 4
sage: zj = fj[0].gens_reduced()[0]; zj
I - 4
```

(continues on next page)
We recover the integer that was factored in \( \mathbb{Z}[i] \) (up to a unit):

```sage
sage: zi*zj
-17
```

One can also factor elements or ideals of the number field:

```sage
sage: K.<a> = NumberField(x^2 + 1)
sage: K.factor(1/3)
(Fractional ideal (3))^-1
sage: K.factor(1+a)
Fractional ideal (a + 1)
sage: K.factor(1+a/5)
(Fractional ideal (a + 1)) * (Fractional ideal (-a - 2))^-1
* (Fractional ideal (2*a + 1))^-1 * (Fractional ideal (-2*a + 3))
```

An example over a relative number field:

```sage
sage: pari('setrand(2)')
sage: L.<b> = K.extension(x^2 - 7)
sage: f = L.factor(a + 1)
sage: f  # representation varies, not tested
(Fractional ideal (1/2*a*b - a + 1/2)) * (Fractional ideal (-1/2*a*b - a + 1/2))
sage: f.value() == a+1
True
```

```sage
>>> from sage.all import *
>>> K = NumberField(x**Integer(2) - Integer(7), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> f = L.factor(a + Integer(1))
>>> f  # representation varies, not tested
(Fractional ideal (1/2*a*b - a + 1/2)) * (Fractional ideal (-1/2*a*b - a + 1/2))
>>> f.value() == a+Integer(1)
True
```
It doesn’t make sense to factor the ideal (0), so this raises an error:

```sage
L.factor(0)
```

Traceback (most recent call last):
...
AttributeError: 'NumberFieldIdeal' object has no attribute 'factor'...

```>>> from sage.all import *
>>> L.factor(Integer(0))
```

Traceback (most recent call last):
...
AttributeError: 'NumberFieldIdeal' object has no attribute 'factor'...

**AUTHORS:**


### fractional_ideal(*gens, **kwds)

Return the ideal in $\mathcal{O}_K$ generated by $\text{gens}$. This overrides the `sage.rings.ring.Field` method to use the `sage.rings.ring.Ring` one instead, since we’re not really concerned with ideals in a field but in its ring of integers.

**INPUT:**

- `gens` — a list of generators, or a number field ideal.

**EXAMPLES:**

```sage
x = polygen(QQ, 'x')
K.<a> = NumberField(x^3 - 2)
sage: K.fractional_ideal([1/a])
Fractional ideal (1/2*a^2)
```

```>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(2), names=(a,)); (a,) = K._first_ngens(1)
>>> K.fractional_ideal([Integer(1)/a])
Fractional ideal (1/2*a^2)
```

One can also input a number field ideal itself, or, more usefully, for a tower of number fields an ideal in one of the fields lower down the tower.

```sage
K.fractional_ideal(K.ideal(a))
```

Fractional ideal (a)

```sage
L.<b> = K.extension(x^2 - 3, x^2 + 1)
sage: L.ideal(K.ideal(2, a)) == L.ideal(a*(b - c)/2)
```

True

```>>> from sage.all import *
>>> K.fractional_ideal(K.ideal(a))
Fractional ideal (a)
```

```>>> L = K.extension(x**Integer(2) - Integer(3), x**Integer(2) + Integer(1), names=('b',)); (b, ) = L._first_ngens(1)
>>> M = L.extension(x**Integer(2) + Integer(1), names=('c',)); (c, ) = M._
```

(continues on next page)
The zero ideal is not a fractional ideal!

```python
sage: K.fractional_ideal(0)
Traceback (most recent call last):
... ValueError: gens must have a nonzero element (zero ideal is not a fractional_ideal)
```

```python
from sage.all import *

>>> K.fractional_ideal(Integer(0))
Traceback (most recent call last):
...
ValueError: gens must have a nonzero element (zero ideal is not a fractional_ideal)
```

galois_group\(type=None, algorithm='pari', names=None, gc_numbering=None\)

Return the Galois group of the Galois closure of this number field.

**INPUT:**

- **type** – Deprecated; the different versions of Galois groups have been merged in Issue #28782.
- **algorithm** – 'pari', 'gap', 'kash', 'magma'. (default: 'pari'; for degrees between 12 and 15 default is 'gap', and when the degree is >= 16 it is 'kash'.)
- **names** – a string giving a name for the generator of the Galois closure of self, when this field is not Galois.
- **gc_numbering** – if True, permutations will be written in terms of the action on the roots of a defining polynomial for the Galois closure, rather than the defining polynomial for the original number field. This is significantly faster; but not the standard way of presenting Galois groups. The default currently depends on the algorithm (True for 'pari', False for 'magma') and may change in the future.

The resulting group will only compute with automorphisms when necessary, so certain functions (such as `sage.rings.number_field.galois_group.GaloisGroup_v2.order()`) will still be fast.

For more (important!) documentation, see the documentation for Galois groups of polynomials over \(\mathbb{Q}\), e.g., by typing `K.polynomial().galois_group?`, where \(K\) is a number field.

**EXAMPLES:**

```python
sage: # needs sage.groups
sage: x = polygen(QQ, 'x')
sage: k.<b> = NumberField(x^2 - 14)  # a Galois extension
sage: G = k.galois_group(); G
Galois group 2T1 (S2) with order 2 of x^2 - 14
sage: G.gen(0)
(1,2)
sage: G.gen(0)(b)
-b
sage: G.artin_symbol(k.primes_above(3)[0])
(1,2)
```
### 1.1. Number fields

**EXPLICIT GALOIS GROUP:** We compute the Galois group as an explicit group of automorphisms of the Galois closure of a field.

```python
sage: # needs sage.groups
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<b1> = K.galois_closure(); L
Number Field in b1 with defining polynomial x^6 + 108
sage: G = End(L); G
Automorphism group of Number Field in b1 with defining polynomial x^6 + 108
sage: G.list()
[Ring endomorphism of Number Field in b1 with defining polynomial x^6 + 108
  Defn: b1 |---> b1,
  ...]
```

(continues on next page)
Many examples for higher degrees may be found in the online databases http://galoisdb.math.upb.de/ by Jürgen Klüners and Gunter Malle and https://www.lmfdb.org/NumberField/ by the LMFDB collaboration, although these might need a lot of computing time.

If \( L/K \) is a relative number field, this method will currently return \( \text{Gal}(L/\mathbb{Q}) \). This behavior will change in the future, so it's better to explicitly call \( \text{absolute_field()} \) if that is the desired behavior:
• \( n \) – must be 0 (the default), or an exception is raised.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: k.<theta> = NumberField(x^14 + 2); k
Number Field in theta with defining polynomial x^14 + 2
sage: k.gen()
theta
sage: k.gen(1)
Traceback (most recent call last):
  ... IndexError: Only one generator.
```

```
>>> from sage.all import *

from sage.all import *

sage: x = polygen(QQ, 'x')
sage: k = NumberField(x**Integer(14) + Integer(2), names=(theta,)); (theta,)
  ...
  = k._first_ngens(1); k
Number Field in theta with defining polynomial x^14 + 2
sage: k.gen()
theta
sage: k.gen(Integer(1))
Traceback (most recent call last):
  ... IndexError: Only one generator.
```

\textbf{gen\_embedding()} 

If an embedding has been specified, return the image of the generator under that embedding. Otherwise return \texttt{None}.

EXAMPLES:

```
sage: QuadraticField(-7, 'a').gen_embedding()
2.645751311064591*I
sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + 7, 'a').gen_embedding()  # None
```

```
>>> from sage.all import *

from sage.all import *

sage: QuadraticField(-Integer(7), a).gen_embedding()
2.645751311064591*I
```

```
>>> from sage.all import *

from sage.all import *

sage: NumberField(x**Integer(2) + Integer(7), a).gen_embedding()  # None
```

\textbf{ideal(*gens, **kwds)}

Return a fractional ideal of the field, except for the zero ideal, which is not a fractional ideal.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(2)
Fractional ideal (2)
sage: K.ideal(2 + i)
Fractional ideal (i + 2)
sage: K.ideal(0)
Ideal (0) of Number Field in i with defining polynomial x^2 + 1
```

1.1. Number fields
idealchinese(ideals, residues)

Return a solution of the Chinese Remainder Theorem problem for ideals in a number field.

This is a wrapper around the pari function pari:idealchinese.

INPUT:

• ideals—a list of ideals of the number field.

• residues—a list of elements of the number field.

OUTPUT:

Return an element \( b \) of the number field such that \( b \equiv x_i \mod I_i \) for all residues \( x_i \) and respective ideals \( I_i \).

See also:

• crt()

EXAMPLES:

This is the example from the pari page on idealchinese:

```
sage: # needs sage.symbolic
sage: K.<sqrt2> = NumberField(sqrt(2).minpoly())
sage: ideals = [K.ideal(4), K.ideal(3)]
sage: residues = [sqrt2, 1]
sage: r = K.idealchinese(ideals, residues); r
-3*sqrt2 + 4
sage: all((r - a) in I for I, a in zip(ideals, residues))
True
```

The result may be non-integral if the results are non-integral:

```
sage: # needs sage.symbolic
sage: K.<sqrt2> = NumberField(sqrt(2).minpoly())
sage: ideals = [K.ideal(4), K.ideal(21)]
```
Algebraic Numbers and Number Fields, Release 10.4

(continued from previous page)

```python
sage: residues = [1/sqrt2, 1]
sage: r = K.idealchinese(ideals, residues); r
-63/2*sqrt2 - 20

sage: all(
    ....:   (r - a).valuation(P) >= k
    ....:   for I, a in zip(ideals, residues)
    ....:   for P, k in I.factor()
    ....:)
True
```

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> K = NumberField(sqrt(Integer(2)).minpoly(), names=('sqrt2',)); (sqrt2,) = K._first_ngens(1)
>>> ideals = [K.ideal(Integer(4)), K.ideal(Integer(21))]
>>> residues = [Integer(1)/sqrt2, Integer(1)]
>>> r = K.idealchinese(ideals, residues); r
-63/2*sqrt2 - 20

>>> all(
    ...   (r - a).valuation(P) >= k
    ...   for I, a in zip(ideals, residues)
    ...   for P, k in I.factor()
    ...
)
True
```

ideals_of_bdd_norm(bound)

Return all integral ideals of bounded norm.

INPUT:

• bound – a positive integer

OUTPUT: A dict of all integral ideals \( I \) such that \( \text{Norm}(I) \leq \text{bound} \), keyed by norm.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 + 23)
sage: d = K.ideals_of_bdd_norm(10)
sage: for n in d:
    ....:   print(n)
    ....:   for I in sorted(d[n]):
    ....:       print(I)
1
  Fractional ideal (1)
  2
  Fractional ideal (2, 1/2*a - 1/2)
  Fractional ideal (2, 1/2*a + 1/2)
  3
  Fractional ideal (3, 1/2*a - 1/2)
  Fractional ideal (3, 1/2*a + 1/2)
  4
  Fractional ideal (2)
  Fractional ideal (4, 1/2*a + 3/2)
  Fractional ideal (4, 1/2*a + 5/2)
```

(continues on next page)
Fractional ideal (1/2*a - 1/2)
Fractional ideal (1/2*a + 1/2)
Fractional ideal (6, 1/2*a + 5/2)
Fractional ideal (6, 1/2*a + 7/2)
7
8
Fractional ideal (4, a - 1)
Fractional ideal (4, a + 1)
Fractional ideal (1/2*a + 3/2)
Fractional ideal (1/2*a - 3/2)
9
Fractional ideal (3)
Fractional ideal (9, 1/2*a + 7/2)
Fractional ideal (9, 1/2*a + 11/2)
10

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(23), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> d = K.ideals_of_bdd_norm(Integer(10))
>>> for n in d:
...    print(n)
...    for I in sorted(d[n]):
...        print(I)
1
Fractional ideal (1)
2
Fractional ideal (2, 1/2*a - 1/2)
Fractional ideal (2, 1/2*a + 1/2)
3
Fractional ideal (3, 1/2*a - 1/2)
Fractional ideal (3, 1/2*a + 1/2)
4
Fractional ideal (2)
Fractional ideal (4, 1/2*a + 3/2)
Fractional ideal (4, 1/2*a + 5/2)
5
6
Fractional ideal (1/2*a - 1/2)
Fractional ideal (1/2*a + 1/2)
Fractional ideal (6, 1/2*a + 5/2)
Fractional ideal (6, 1/2*a + 7/2)
7
8
Fractional ideal (4, a - 1)
Fractional ideal (4, a + 1)
Fractional ideal (1/2*a + 3/2)
Fractional ideal (1/2*a - 3/2)
9
Fractional ideal (3)
Fractional ideal (9, 1/2*a + 7/2)
Fractional ideal (9, 1/2*a + 11/2)
10

integral_basis (v=None)

Return a list containing a \ZZ[-basis for the full ring of integers of this number field.
INPUT:

- \( v \) - None, a prime, or a list of primes. See the documentation for `maximal_order()`.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^5 + 10*x + 1)
sage: K.integral_basis()
[1, a, a^2, a^3, a^4]
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(5) + Integer(10)*x + Integer(1), names=('a',)); (a,)

>>> K.integral_basis()
[1, a, a^2, a^3, a^4]
```

Next we compute the ring of integers of a cubic field in which 2 is an “essential discriminant divisor”, so the ring of integers is not generated by a single element.

```python
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: K.integral_basis()
[1, 1/2*a^2 + 1/2*a, a^2]
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(3) + x**Integer(2) - Integer(2)*x + Integer(8), names=('a',)); (a,)

>>> K.integral_basis()
[1, 1/2*a^2 + 1/2*a, a^2]
```

ALGORITHM: Uses the PARI library (via `pari:pari_integral_basis`).

`is_CM()`

Return True if self is a CM field (i.e., a totally imaginary quadratic extension of a totally real field).

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: Q.<a> = NumberField(x - 1)
sage: Q.is_CM()
False
```

```python
sage: K.<i> = NumberField(x^2 + 1)
sage: K.is_CM()
True
```

```python
sage: L.<zeta20> = CyclotomicField(20)
sage: L.is_CM()
True
```

```python
sage: K.<omega> = QuadraticField(-3)
sage: K.is_CM()
True
```

```python
sage: L.<sqrt5> = QuadraticField(5)
sage: L.is_CM()
False
```

```python
sage: F.<a> = NumberField(x^3 - 2)
sage: F.is_CM()
False
```

```python
sage: F.<a> = NumberField(x^4 - x^3 - 3*x^2 + x + 1)
```

(continues on next page)
Algebraic Numbers and Number Fields, Release 10.4

The following are non-CM totally imaginary fields.

```python
sage: F.<a> = NumberField(x^4 + x^3 - x^2 - x + 1)
sage: F.is_totally_imaginary()
True
sage: F.is_CM()
False
sage: F2.<a> = NumberField(x^12 - 5*x^11 + 8*x^10 - 5*x^9 - x^8 + 9*x^7 + 7*x^6 - 3*x^5 + 5*x^4 + 7*x^3 - 4*x^2 - 7*x + 7)
sage: F2.is_totally_imaginary()
True
sage: F2.is_CM()
False
```

(continues on next page)
The following is a non-cyclotomic CM field.

```python
sage: M.<a> = NumberField(x^4 - x^3 - x^2 - 2*x + 4)
sage: M.is_CM()
True
```

Now, we construct a totally imaginary quadratic extension of a totally real field (which is not cyclotomic).

```python
sage: E_0.<a> = NumberField(x^7 - 4*x^6 - 4*x^5 + 10*x^4 + 4*x^3 - 6*x^2 - x + 1)
sage: E_0.is_totally_real()
True
sage: E.<b> = E_0.extension(x^2 + 1)
sage: E.is_CM()
True
```

Finally, a CM field that is given as an extension that is not CM.

```python
sage: E_0.<a> = NumberField(x^2 - 4*x + 16)
sage: y = polygen(E_0)
sage: E.<z> = E_0.extension(y^2 - E_0.gen() / 2)
sage: E.is_CM()
True
sage: E.is_CM_extension()
False
```
is_abelian()  
Return True if this number field is an abelian Galois extension of \( \mathbb{Q} \).

EXAMPLES:

```sage
sage: # needs sage.groups
sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + 1, 'i').is_abelian()
True
sage: NumberField(x^3 + 2, 'a').is_abelian()
False
sage: NumberField(x^3 + x^2 - 2*x - 1, 'a').is_abelian()
True
sage: NumberField(x^6 + 40*x^3 + 1372, 'a').is_abelian()
False
sage: NumberField(x^6 + x^5 - 5*x^4 - 4*x^3 + 6*x^2 + 3*x - 1, 'a').is_abelian()
True
```

is_absolute()  
Return True if self is an absolute field.

This function will be implemented in the derived classes.

EXAMPLES:

```sage
sage: K = CyclotomicField(5)
sage: K.is_absolute()
True
```
is_field (proof=True)

Return True since a number field is a field.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: NumberField(x^5 + x + 3, 'x').is_field()
True
```

is_galois()

Return True if this number field is a Galois extension of Q.

EXAMPLES:

```
sage: # needs sage.groups
sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + 1, 'i').is_galois()
True
sage: NumberField(x^3 + 2, 'a').is_galois()
False
```

(continues on next page)
is_isomorphic (other, isomorphism_maps=False)

Return True if self is isomorphic as a number field to other.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^2 + 1)
sage: m.<b> = NumberField(x^2 + 4)
sage: k.is_isomorphic(m)
True
sage: m.<b> = NumberField(x^2 + 5)
sage: k.is_isomorphic (m)
False

sage: k = NumberField(x^3 + 2, a)
sage: k.is_isomorphic(NumberField((x+1/3)^3 + 2, 'b'))
True
sage: k.is_isomorphic(NumberField(x^3 + 4, 'b'))
True
sage: k.is_isomorphic(NumberField(x^3 + 5, 'b'))
False

sage: k = NumberField(x^2 - x - 1, b)
sage: l = NumberField(x^2 - 7, a)
sage: k.is_isomorphic(l, True)
(False, [])
sage: k.is_isomorphic(l, True)
False
is_relative()  

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^10 - 2)
sage: K.is_absolute() 
True
sage: K.is_relative() 
False

is_totally_imaginary()  

Return True if self is totally imaginary, and False otherwise.

Totally imaginary means that no isomorphic embedding of self into the complex numbers has image contained in the real numbers.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + 2, 'alpha').is_totally_imaginary() 
True
sage: NumberField(x^2 - 2, 'alpha').is_totally_imaginary() 
False
is_totally_imaginary()

Return \texttt{True} if \texttt{self} is totally imaginary, and \texttt{False} otherwise.

Totally imaginary means that every isomorphic embedding of \texttt{self} into the complex numbers has image contained in the real numbers.

EXAMPLES:

\begin{verbatim}
sage: NumberField(x^4 - 2, 'alpha').is_totally_imaginary()
False

>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> NumberField(x^4 - 2, 'alpha').is_totally_imaginary()
True
</verbatim>

is_totally_real()

Return \texttt{True} if \texttt{self} is totally real, and \texttt{False} otherwise.

Totally real means that every isomorphic embedding of \texttt{self} into the complex numbers has image contained in the real numbers.

EXAMPLES:

\begin{verbatim}
sage: x = polygen(QQ, 'x')

sage: NumberField(x^2 + 2, 'alpha').is_totally_real()
False

sage: NumberField(x^2 - 2, 'alpha').is_totally_real()
True

sage: NumberField(x^4 - 2, 'alpha').is_totally_real()
False
</verbatim>

lmfdb_page()

Open the LMFDB web page of the number field in a browser.

See https://www.lmfdb.org

EXAMPLES:

\begin{verbatim}
sage: E = QuadraticField(-1)

sage: E.lmfdb_page() # optional -- webbrowser

>>> from sage.all import *

>>> E = QuadraticField(-Integer(1))

>>> E.lmfdb_page() # optional -- webbrowser
</verbatim>

Even if the variable name is different it works:

\begin{verbatim}
sage: R.<y>= PolynomialRing(QQ, "y")

sage: K = NumberField(y^2 + 1, "i")

sage: K.lmfdb_page() # optional -- webbrowser
</verbatim>
maximal_order (v=\text{None}, \text{assume\_maximal}=\text{non-maximal-non-unique})

Return the maximal order, i.e., the ring of integers, associated to this number field.

INPUT:

- \text{v} = \text{None}, \text{a prime, or a list of integer primes (default: None)}
  - if None, return the maximal order.
  - if a prime \( p \), return an order that is \( p \)-maximal.
  - if a list, return an order that is maximal at each prime of these primes

- \text{assume\_maximal} – True, False, None, or "non-maximal-non-unique" (default: "non-maximal-non-unique") ignored when \( v \) is None; otherwise, controls whether we assume that the order \text{order.is\_maximal()} outside of \( v \).
  - if True, the order is assumed to be maximal at all primes.
  - if False, the order is assumed to be non-maximal at some prime not in \( v \).
  - if None, no assumptions are made about primes not in \( v \).
  - if "non-maximal-non-unique" (deprecated), like False, however, the order is not a unique parent, so creating the same order later does typically not poison caches with the information that the order is not maximal.

EXAMPLES:

In this example, the maximal order cannot be generated by a single element:

```
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: o = k.maximal_order()
sage: o
Maximal Order generated by [1/2*a^2 + 1/2*a, a^2] in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
```

We compute \( p \)-maximal orders for several \( p \). Note that computing a \( p \)-maximal order is much faster in general than computing the maximal order:

```
sage: p = next_prime(10^22)
sage: q = next_prime(10^23)
sage: K.<a> = NumberField(x^3 - p*q)
sage: K.maximal_order([3], assume_maximal=None).basis()
```
sage: K.maximal_order([2], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

sage: K.maximal_order([p], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

sage: K.maximal_order([q], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

sage: K.maximal_order([p, 3], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

>>> from sage.all import *

>>> p = next_prime(10**22)

>>> q = next_prime(10**23)

>>> K.<a> = NumberField(x^3 - p*q, names=('a',)); (a,) = K._first_
˓→ngens(1)

>>> K.maximal_order([Integer(3)], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

>>> K.maximal_order([Integer(2)], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

>>> K.maximal_order([p], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

>>> K.maximal_order([q], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]

An example with bigger discriminant:

sage: p = next_prime(10^97)

sage: q = next_prime(10^99)

sage: K.<a> = NumberField(x^3 - p*q)

sage: K.maximal_order(prime_range(10000), assume_maximal=None).basis()
[1, a, a^2]

>>> from sage.all import *

>>> p = next_prime(Integer(10)**Integer(97))

>>> q = next_prime(Integer(10)**Integer(99))

>>> K = NumberField(x^3 - p*q, names=('a',)); (a,) = K._first_
˓→ngens(1)

>>> K.maximal_order(prime_range(Integer(10000)), assume_maximal=None).basis()
[1, a, a^2]

An example in a relative number field:

sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 3])

sage: OK = K.maximal_order()

sage: OK.basis()
An order that is maximal at a prime. We happen to know that it is actually maximal and mark it as such:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.maximal_order(v=2, assume_maximal=True)
Gaussian Integers generated by i in Number Field in i with defining polynomial x^2 + 1
```

It is an error to create a maximal order and declare it non-maximal, however, such mistakes are only caught automatically if they evidently contradict previous results in this session:

```
sage: K.maximal_order(v=2, assume_maximal=False)
Traceback (most recent call last):
...  
ValueError: cannot assume this order to be non-maximal because we already found it to be a maximal order
```

```
sage: K.maximal_order(v=2, assume_maximal=False)
Traceback (most recent call last):
...  
ValueError: cannot assume this order to be non-maximal because we already found it to be a maximal order
```
maximal_totally_real_subfield()

Return the maximal totally real subfield of \textit{self} together with an embedding of it into \textit{self}.

**EXAMPLES:**

```python
sage: F.<a> = QuadraticField(11)
sage: F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^2 - 11 with a = 3. →316624790355400?,
    Identity endomorphism of
    Number Field in a with defining polynomial x^2 - 11 with a = 3. →316624790355400?]
sage: F.<a> = QuadraticField(-15)
sage: F.maximal_totally_real_subfield()
[Rational Field, Natural morphism:
    From: Rational Field
    To: Number Field in a with defining polynomial x^2 + 15
        with a = 3.872983346207417?*I]
sage: F.<a> = CyclotomicField(29)
sage: F.maximal_totally_real_subfield()
(Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 - 12*x^11
  + 66*x^10 + 55*x^9 - 165*x^8 - 120*x^7 + 210*x^6 + 126*x^5 - 126*x^4
  - 56*x^3 + 28*x^2 + 7*x - 1 with a0 = 1.95324111420174?,
    Ring morphism:
    From: Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 -...
    To: Cyclotomic Field of order 29 and degree 28
    Defn: a0 |--> -a^27 - a^26 - a^25 - a^24 - a^23 - a^22 - a^21 - a^20 - a^19
        - a^18 - a^17 - a^16 - a^15 - a^14 - a^13 - a^12 - a^11 - a^10
        - a^9 - a^8 - a^7 - a^6 - a^5 - a^4 - a^3 - a^2 - 1)
sage: x = polygen(QQ, 'x')
sage: F.<a> = NumberField(x^3 - 2)
sage: F.maximal_totally_real_subfield()
[Rational Field, Coercion map:
    From: Rational Field
    To: Number Field in a with defining polynomial x^3 - 2]
sage: F.<a> = NumberField(x^4 - x^3 - x^2 + x + 1)
sage: F.maximal_totally_real_subfield()
[Rational Field, Coercion map:
    From: Rational Field
    To: Number Field in a with defining polynomial x^4 - x^3 - x^2 + x + 1]
sage: F.<a> = NumberField(x^4 - 4*x^2 - x + 1)
sage: F.maximal_totally_real_subfield()
[Number Field in a1 with defining polynomial x^2 - x - 1,
    Ring morphism:
    From: Number Field in a1 with defining polynomial x^2 - x - 1
    To: Number Field in a with defining polynomial x^4 - x^3 + 2*x^2 + x + 1
    Defn: a1 |--> -1/2*a^3 - 1/2]
sage: F.<a> = NumberField(x^4 - 4*x^2 - x + 1)
sage: F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^4 - 4*x^2 - x + 1,
    Identity endomorphism of
    Number Field in a with defining polynomial x^4 - 4*x^2 - x + 1]```
```python
>>> from sage.all import *
>>> F = QuadraticField(Integer(11), names=('a',)); (a,) = F._first_ngens(1)
>>> F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^2 - 11 with a = 3.
→\sqrt{11} \\
Identity endomorphism of
Number Field in a with defining polynomial x^2 - 11 with a = 3.
→\sqrt{11}]
>>> F = QuadraticField(-Integer(15), names=('a',)); (a,) = F._first_ngens(1)
>>> F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^2 + 15 with a = 3.872983346207417?*I
→I \\
Identity endomorphism of
Number Field in a with defining polynomial x^2 + 15 with a = 3.872983346207417?*I]
>>> F = CyclotomicField(Integer(29), names=('a',)); (a,) = F._first_ngens(1)
>>> F.maximal_totally_real_subfield()
[Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 - 12*x^11 + 66*x^10 + 55*x^9 - 165*x^8 - 120*x^7 + 210*x^6 + 126*x^5 - 126*x^4 - 56*x^3 + 28*x^2 + 7*x - 1 with a0 = 1.953241111420174?,
→\sqrt{29}/2 - \sqrt{29}/2 \\
Ring morphism:
From: Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 - 12*x^11 + 66*x^10 + 55*x^9 - 165*x^8 - 120*x^7 + 210*x^6 + 126*x^5 - 126*x^4 - 56*x^3 + 28*x^2 + 7*x - 1 with a0 = 1.953241111420174?
To: Cyclotomic Field of order 29 and degree 28
Defn: a0 |--> -a^27 - a^26 - a^25 - a^24 - a^23 - a^22 - a^21 - a^20 - a^19 - a^18 - a^17 - a^16 - a^15 - a^14 - a^13 - a^12 - a^11 - a^10 - a^9 - a^8 - a^7 - a^6 - a^5 - a^4 - a^3 - a^2 - 1]
>>> x = polygen(QQ, 'x')
>>> F = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = F._first_ngens(1)
>>> F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^3 - 2 with a = \sqrt{2},
→\sqrt{2} \\
Identity endomorphism of
Number Field in a with defining polynomial x^3 - 2 with a = \sqrt{2}]
>>> F = NumberField(x**Integer(4) - x**Integer(3) - x**Integer(2) + x + Integer(1), names=('a',)); (a,) = F._first_ngens(1)
>>> F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^4 - x^3 - x^2 + x + 1 with a = \sqrt{-2},
→\sqrt{-2} + 1/2i \\
Identity endomorphism of
Number Field in a with defining polynomial x^4 - x^3 - x^2 + x + 1 with a = \sqrt{-2},
→\sqrt{-2} + 1/2i]
>>> F = NumberField(x**Integer(4) - Integer(4)*x**Integer(2) - x + Integer(1), names=('a',)); (a,) = F._first_ngens(1)
>>> F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^4 - 4*x^2 - x + 1 with a = \sqrt{-1} + \sqrt{-4}/2,
→\sqrt{-1} + \sqrt{-4}/2 \\
Identity endomorphism of
Number Field in a with defining polynomial x^4 - 4*x^2 - x + 1 with a = \sqrt{-1} + \sqrt{-4}/2]
```
An example of a relative extension where the base field is not the maximal totally real subfield.

```python
sage: E_0.<a> = NumberField(x^2 - 4*x + 16)

sage: y = polygen(E_0)

sage: E.<z> = E_0.extension(y^2 - E_0.gen() / 2)

sage: E.maximal_totally_real_subfield()

[Number Field in z1 with defining polynomial x^2 - 2*x - 5,
  Composite map:
  From: Number Field in z1 with defining polynomial x^2 - 2*x - 5
  To: Number Field in z with defining polynomial x^2 - 1/2*a over its base_field
  Defn: Ring morphism:
    From: Number Field in z1 with defining polynomial x^2 - 2*x - 5
    To: Number Field in z with defining polynomial x^4 - 2*x^3 + x^2 + 6*x + 3
    Defn: z1 |--> -1/3*z^3 + 1/3*z^2 + z - 1
  then
  Isomorphism map:
    From: Number Field in z with defining polynomial x^4 - 2*x^3 + x^2 + 6*x + 3
    To: Number Field in z with defining polynomial x^2 - 1/2*a over its base_field]
```

```python
>>> from sage.all import *

>>> E_0 = NumberField(x**Integer(2) - Integer(4)*x + Integer(16), names=('a',)); (a,) = E_0._first_ngens(1)

>>> y = polygen(E_0)

>>> E = E_0.extension(y**Integer(2) - E_0.gen() / Integer(2), names=('z',)); (z,) = E._first_ngens(1)

>>> E.maximal_totally_real_subfield()

[Number Field in z1 with defining polynomial x^2 - 2*x - 5,
  Composite map:
  From: Number Field in z1 with defining polynomial x^2 - 2*x - 5
  To: Number Field in z with defining polynomial x^4 - 2*x^3 + x^2 + 6*x + 3
  Defn: z1 |--> -1/3*z^3 + 1/3*z^2 + z - 1
  then
  Isomorphism map:
    From: Number Field in z with defining polynomial x^4 - 2*x^3 + x^2 + 6*x + 3
    To: Number Field in z with defining polynomial x^2 - 1/2*a over its base_field]
```

`narrow_class_group(proof=None)`

Return the narrow class group of this field.

**INPUT:**

- `proof` — default: None (use the global proof setting, which defaults to True).

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')

sage: NumberField(x^3 + x + 9, 'a').narrow_class_group()

Multiplicative Abelian group isomorphic to C2
```
ngens()  
Return the number of generators of this number field (always 1).  

OUTPUT: the python integer 1.  

EXAMPLES:

```sage
definitions:
x = polygen(QQ, 'x')
F.<alpha> = NumberField(x^22 + 3)
F.<alpha> = NumberField(x^2 - 7)
```

number_of_roots_of_unity()  
Return the number of roots of unity in this field.  

**Note:** We do not create the full unit group since that can be expensive, but we do use it if it is already known.  

EXAMPLES:

```sage
definitions:
x = polygen(QQ, 'x')
F.<alpha> = NumberField(x^22 + 3)
F.<alpha> = NumberField(x^2 - 7)
```
order()

Return the order of this number field (always +infinity).

OUTPUT: always positive infinity

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + 19, 'a').order()
+Infinity
```

```python
>>> from sage.all import *
```

```python
>>> x = polygen(QQ, 'x')
```

```python
>>> NumberField(x**Integer(2) + Integer(19), 'a').order()
+Infinity
```

pari_bnf (proof=None, units=True)

PARI big number field corresponding to this field.

INPUT:

- proof – If False, assume GRH. If True, run PARI’s pari:bnfcertify to make sure that the results are correct.

- units – (default: True) If `True`, insist on having fundamental units. If False, the units may or may not be computed.

OUTPUT:

The PARI bnf structure of this number field.

Warning: Even with proof=True, I wouldn’t trust this to mean that everything computed involving this number field is actually correct.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^2 + 1); k
Number Field in a with defining polynomial x^2 + 1
sage: len(k.pari_bnf())
10
sage: k.pari_bnf()[:4]
[[], matrix(0,3), [], [], ...]
```

```python
sage: len(k.pari_nf())
9
```

```python
sage: k.<a> = NumberField(x^7 + 7); k
Number Field in a with defining polynomial x^7 + 7
sage: dummy = k.pari_bnf(proof=True)
```
```python
>>> from sage.all import *

```n
```python
>>> x = polygen(QQ, 'x')

```n
```python
>>> k = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = k._first_ngens(1); k
```
```n
```python
Number Field in a with defining polynomial x^2 + 1

```n
```python
>>> len(k.pari_bnf())
10

```n
```python
>>> k.pari_bnf()[:Integer(4)]
[;], matrix(0,3), [;], ...

```n
```python
>>> len(k.pari_nf())
9

```n
```python
>>> k = NumberField(x**Integer(7) + Integer(7), names=('a',)); (a,) = k._first_ngens(1); k
```
```n
```python
Number Field in a with defining polynomial x^7 + 7

```n
```python
>>> dummy = k.pari_bnf(proof=True)

```n
```python
pari_nf(important=True)

```n
Return the PARI number field corresponding to this field.

**INPUT:**

- **important** – boolean (default: True). If False, raise a `RuntimeError` if we need to do a difficult discriminant factorization. This is useful when an integral basis is not strictly required, such as for factoring polynomials over this number field.

**OUTPUT:**

The PARI number field obtained by calling the PARI function `pari:nfinit` with `self.pari_polynomial(y)` as argument.

**Note:** This method has the same effect as `pari(self)`.

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^4 - 3*x + 7); k
Number Field in a with defining polynomial x^4 - 3*x + 7
sage: k.pari_nf()[:4]
y^4 - 3*y + 7, [0, 2], 85621, 1
sage: pari(k)[:4]
y^4 - 3*y + 7, [0, 2], 85621, 1
```

```python
sage: k.<a> = NumberField(x^4 - 3/2*x + 5/3); k
Number Field in a with defining polynomial x^4 - 3/2*x + 5/3
sage: k.pari_nf()[:4]
y^4 - 324*y + 2160, [0, 2], 48918708, 216, ..., [36, 36*y, y^3 + 6*y^2 - 252,
```
(continues on next page)
\[
\begin{align*}
\text{\rightarrow} & \ 6y^2, [1, 0, 0, 252; 0, 1, 0, 0; 0, 0, 0, 36; 0, 0, 6, -36], [1, 0, 0, 0, \\
\text{\rightarrow} & \ 0, 0, -18, 42, 0, -18, -66, 0, 42, -60, -60; 0, 1, 0, 0, 1, 0, 2, 0, \\
\text{\rightarrow} & \ 0, 2, -11, -1, 0, 0, -1, 9; 0, 0, 1, 0, 0, 0, 6, 1, 6, -5, 0, 0, 6, 0, 0; \\
\text{\rightarrow} & \ 0, 0, 0, 1, 0, 6, -6, -6, 0, -6, -1, 2, 1, -6, 2, 0]\]
\]

\textbf{sage:} pari(k)
\[
[y^4 - 324y + 2160, [0, 2], 48918708, 216, ...]
\]

\textbf{sage:} gp(k)
\[
[y^4 - 324y + 2160, [0, 2], 48918708, 216, ...]
\]

\textbf{>>> from sage.all import *}
\textbf{>>=} k = NumberField(x**Integer(4) - Integer(3)/Integer(2)*x + Integer(5)/Integer(3), names=('a',)); (a,) = k._first_ngens(1); k
\textbf{Number Field in a with defining polynomial x^4 - 3/2*x + 5/3}
\textbf{>>=} k.pari_nf()
\[
[y^4 - 324y + 2160, [0, 2], 48918708, 216, ..., [36, 36*y, y^3 + 6*y^2 - 252, \rightarrow 6*y^2], [1, 0, 0, 252; 0, 1, 0, 0; 0, 0, 0, 36; 0, 0, 6, -36], [1, 0, 0, 0, \\
\text{\rightarrow} & \ 0, 0, -18, 42, 0, -18, -66, 0, 42, -60, -60; 0, 1, 0, 0, 1, 0, 2, 0, \\
\text{\rightarrow} & \ 0, 2, -11, -1, 0, 0, -1, 9; 0, 0, 1, 0, 0, 0, 6, 1, 6, -5, 0, 0, 6, 0, 0; \\
\text{\rightarrow} & \ 0, 0, 0, 1, 0, 6, -6, -6, 0, -6, -1, 2, 1, -6, 2, 0]]
\]

\textbf{>>> pari(k)}
\[
[y^4 - 324y + 2160, [0, 2], 48918708, 216, ...]
\]

\textbf{>>> gp(k)}
\[
[y^4 - 324y + 2160, [0, 2], 48918708, 216, ...]
\]

With \texttt{important=False}, we simply bail out if we cannot easily factor the discriminant:

\textbf{sage:} p = next_prime(10^40); q = next_prime(10^41)
\textbf{sage:} K.<a> = NumberField(x^2 - p*q)
\textbf{sage:} K.pari_nf(important=False)
Traceback (most recent call last):
... 
RuntimeError: Unable to factor discriminant with trial division

\textbf{>>> from sage.all import *}
\textbf{>>=} p = next_prime(Integer(10)**Integer(40)); q = next_
\textbf{\rightarrow prime(Integer(10)**Integer(41))}
\textbf{>>=} K = NumberField(x**Integer(2) - p*q, names=('a',)); (a,) = K._first_
\textbf{\rightarrow ngens(1)}
\textbf{>>=} K.pari_nf(important=False)
Traceback (most recent call last):
... 
RuntimeError: Unable to factor discriminant with trial division

Next, we illustrate the \texttt{maximize_at_primes} and \texttt{assume_disc_small} parameters of the \texttt{NumberField} constructor. The following would take a very long time without the \texttt{maximize_at_primes} option:

\textbf{sage:} K.<a> = NumberField(x^2 - p*q, maximize_at_primes=[p])
\textbf{sage:} K.pari_nf()
\[
[y^2 - 100000000000000000(249,797),(761,838)
\]

\textbf{>>> from sage.all import *}
\textbf{>>=} K = NumberField(x**Integer(2) - p*q, maximize_at_primes=[p], names=('a', 
\texttt{\rightarrow}))
\textbf{>>=} K.pari_nf()
\[
[y^2 - 100000000000000000(249,851),(761,892)
\]
Since the discriminant is square-free, this also works:

```python
sage: K.<a> = NumberField(x^2 - p*q, assume_disc_small=True)
sage: K.pari_nf()
[y^2 - 100000000000000000000...]
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(2) - p*q, assume_disc_small=True, names=('a',)); (a,) = K._first_ngens(1)
>>> K.pari_nf()
[y^2 - 100000000000000000000...]
```

```python
pari_polynomial (name='x')
```

Return the PARI polynomial corresponding to this number field.

**INPUT:**

- `name` – variable name (default: `'x'`)

**OUTPUT:**

A monic polynomial with integral coefficients (PARI t_Pol) defining the PARI number field corresponding to `self`.

**Warning:** This is not the same as simply converting the defining polynomial to PARI.

**EXAMPLES:**

```python
sage: y = polygen(QQ)
sage: k.<a> = NumberField(y^2 - 3/2*y + 5/3)
sage: k.pari_polynomial()
x^2 - x + 40
sage: k.polynomial().__pari__()
x^2 - 3/2*x + 5/3
sage: k.pari_polynomial(a)
a^2 - a + 40
```

```python
>>> from sage.all import *

>>> y = polygen(QQ)

>>> k = NumberField(y**Integer(2) - Integer(3)/Integer(2)*y + Integer(5)/Integer(3), names=('a',)); (a,) = k._first_ngens(1)

>>> k.pari_polynomial()
x^2 - x + 40
>>> k.polynomial().__pari__()
x^2 - 3/2*x + 5/3
>>> k.pari_polynomial(a)
a^2 - a + 40
```

Some examples with relative number fields:

```python
sage: x = polygen(ZZ, 'x')
sage: k.<a, c> = NumberField([x^2 + 3, x^2 + 1])
sage: k.pari_polynomial()
x^4 + 8*x^2 + 4
sage: k.pari_polynomial('a')
a^4 + 8*a^2 + 4
```

(continues on next page)
This fails with arguments which are not a valid PARI variable name:

```
sage: k = QuadraticField(-1)
sage: k.pari_polynomial('I')
Traceback (most recent call last):
...
PariError: I already exists with incompatible valence
sage: k.pari_polynomial('i')
i^2 + 1
sage: k.pari_polynomial('theta')
Traceback (most recent call last):
...
PariError: theta already exists with incompatible valence
```
pari_rnfnorm_data \((L, \text{proof}=\text{True})\)

Return the PARI \texttt{pari:rnfnorminit} data corresponding to the extension \(L / \text{self}\).

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(QQ)
sage: K = NumberField(x^2 - 2, 'alpha')
sage: L = K.extension(x^2 + 5, 'gamma')
sage: ls = K.pari_rnfnorm_data(L); len(ls) 8

sage: K.<a> = NumberField(x^2 + x + 1)
sage: P.<X> = K[]
sage: L.<b> = NumberField(X^3 + a)
sage: ls = K.pari_rnfnorm_data(L); len(ls) 8
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> x = polygen(QQ)
>>> K = NumberField(x**Integer(2) - Integer(2), 'alpha')
>>> L = K.extension(x**Integer(2) + Integer(5), 'gamma')
>>> ls = K.pari_rnfnorm_data(L) ; len(ls)
8

>>> K = NumberField(x**Integer(2) + x + Integer(1), names=(a,)); (a,) = K._first_ngens(1)
>>> P = K['X']; (X,) = P._first_ngens(1)
>>> L = NumberField(X**Integer(3) + a, names=('b',)); (b,) = L._first_ngens(1)
>>> ls = K.pari_rnfnorm_data(L); len(ls)
8
\end{verbatim}

pari_zk()

Integral basis of the PARI number field corresponding to this field.

This is the same as \texttt{pari_nf().getattr('zk')}, but much faster.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^3 - 17)
sage: k.pari_zk()
[1, 1/3*y^2 - 1/3*y + 1/3, y]
sage: k.pari_nf().getattr('zk')
[1, 1/3*y^2 - 1/3*y + 1/3, y]

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> k = NumberField(x**Integer(3) - Integer(17), names=('a',)); (a,) = k._first_ngens(1)
>>> k.pari_zk()
[1, 1/3*y^2 - 1/3*y + 1/3, y]
>>> k.pari_nf().getattr('zk')
[1, 1/3*y^2 - 1/3*y + 1/3, y]
\end{verbatim}

1.1. Number fields
polynomial()

Return the defining polynomial of this number field.

This is exactly the same as defining_polynomial().

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + (2/3)*x - 9/17, 'a').polynomial()
x^2 + 2/3*x - 9/17
```

```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> NumberField(x**Integer(2) + (Integer(2)/Integer(3))*x - Integer(9)/Integer(17), 'a').polynomial()
x^2 + 2/3*x - 9/17
```

polynomial_ntl()

Return defining polynomial of this number field as a pair, an ntl polynomial and a denominator.

This is used mainly to implement some internal arithmetic.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: NumberField(x^2 + (2/3)*x - 9/17, 'a').polynomial_ntl()
([-27 34 51], 51)
```

```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> NumberField(x**Integer(2) + (Integer(2)/Integer(3))*x - Integer(9)/Integer(17), 'a').polynomial_ntl()
([-27 34 51], 51)
```

polynomial_quotient_ring()

Return the polynomial quotient ring isomorphic to this number field.

EXAMPLES:

```
sage: x = polygen(QQ, 'x')
sage: K = NumberField(x^3 + 2*x - 5, alpha)
sage: K.polynomial_quotient_ring()
Univariate Quotient Polynomial Ring in alpha over Rational Field
with modulus x^3 + 2*x - 5
```

```
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(3) + Integer(2)*x - Integer(5), 'alpha')
>>> K.polynomial_quotient_ring()
Univariate Quotient Polynomial Ring in alpha over Rational Field
with modulus x^3 + 2*x - 5
```

polynomial_ring()

Return the polynomial ring that we view this number field as being a quotient of (by a principal ideal).

EXAMPLES: An example with an absolute field:
sage: x = polygen(QQ, 'x')
sage: k.<a> = NumberField(x^2 + 3)
sage: y = polygen(QQ, 'y')
sage: k.<a> = NumberField(y^2 + 3)
sage: k.polynomial_ring()
Univariate Polynomial Ring in y over Rational Field

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> k = NumberField(x^Integer(2) + Integer(3), names=(a,)); (a,) = k._first_ngens(1)
>>> y = polygen(QQ, 'y')
>>> k = NumberField(y^Integer(2) + Integer(3), names=(a,)); (a,) = k._first_ngens(1)
>>> k.polynomial_ring()
Univariate Polynomial Ring in y over Rational Field

An example with a relative field:

sage: y = polygen(QQ, 'y')
sage: M.<a> = NumberField([y^3 + 97, y^2 + 1]); M
Number Field in a0 with defining polynomial y^3 + 97 over its base field
sage: M.polynomial_ring()
Univariate Polynomial Ring in y over
Number Field in a1 with defining polynomial y^2 + 1

>>> from sage.all import *
>>> y = polygen(QQ, 'y')
>>> M = NumberField([y**Integer(3) + Integer(97), y**Integer(2) + Integer(1)],
xiety=names=('a',)); (a,) = M._first_ngens(1); M
Number Field in a0 with defining polynomial y^3 + 97 over its base field
>>> M.polynomial_ring()
Univariate Polynomial Ring in y over
Number Field in a1 with defining polynomial y^2 + 1

power_basis()

Return a power basis for this number field over its base field.

If this number field is represented as \( k[t]/f(t) \), then the basis returned is \( 1, t, t^2, \ldots, t^{d-1} \) where \( d \) is the degree of this number field over its base field.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^5 + 10*x + 1)
sage: K.power_basis()
[1, a, a^2, a^3, a^4]

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x^Integer(5) + Integer(10)*x + Integer(1), names=('a',));
xiety=(a,) = K._first_ngens(1)
>>> K.power_basis()
[1, a, a^2, a^3, a^4]
prime_above \(x, degree=None\)

Return a prime ideal of self lying over \(x\).

INPUT:

- \(x\) – usually an element or ideal of self. It should be such that \(self.ideal(x)\) is sensible. This excludes \(x = 0\).
- \(degree\) – (default: None) None or an integer. If one, find a prime above \(x\) of any degree. If an integer, find a prime above \(x\) such that the resulting residue field has exactly this degree.

OUTPUT: A prime ideal of self lying over \(x\). If degree is specified and no such ideal exists, raises a ValueError.

EXAMPLES:

```python
sage: x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)

>>> from sage.all import *
>>> x = ZZ['x'].gen()
>>> F = NumberField(x**Integer(3) - Integer(2), names=('t',)); (t,) = F._
˓→first_ngens(1)
```
The ideal $(3)$ is totally ramified in $F$, so there is no degree 2 prime above 3:

```
sage: F.prime_above(3, degree=2)
Traceback (most recent call last):
...
ValueError: No prime of degree 2 above Fractional ideal (3)
sage: [ id.residue_class_degree() for id, _ in F.ideal(3).factor() ]
[1]
```

Asking for a specific degree works:

```
sage: P5_1 = F.prime_above(5, degree=1)
sage: P5_1 # random
Fractional ideal (-t^2 - 1)
sage: P5_1.residue_class_degree()
1
```
Relative number fields are ok:

```
sage: G = F.extension(x^2 - 11, b)
sage: G.prime_above(7)
Fractional ideal (b + 2)
```

It doesn't make sense to factor the ideal (0):

```
sage: F.prime_above(0)
Traceback (most recent call last):
  ... AttributeError: 'NumberFieldIdeal' object has no attribute 'prime_factors'
```

```
prime_factors(x)
```

Return a list of the prime ideals of self which divide the ideal generated by x.

**OUTPUT:** list of prime ideals (a new list is returned each time this function is called)

**EXAMPLES:**

```
sage: x = polygen(QQ, 'x')
sage: K.<w> = NumberField(x^2 + 23)
sage: K.prime_factors(w + 1)
[Fractional ideal (2, 1/2*w - 1/2),
 Fractional ideal (2, 1/2*w + 1/2),
 Fractional ideal (3, 1/2*w + 1/2)]
```
primes_above\((x, \text{degree}=\text{None})\)

Return prime ideals of self lying over \(x\).

**INPUT:**

- \(x\) – usually an element or ideal of self. It should be such that \(\text{self.ideal}(x)\) is sensible. This excludes \(x = 0\).
- \(\text{degree}\) – (default: \text{None}) \text{None} or an integer. If \text{None}, find all primes above \(x\) of any degree. If an integer, find all primes above \(x\) such that the resulting residue field has exactly this degree.

**OUTPUT:** A list of prime ideals of self lying over \(x\). If \text{degree} is specified and no such ideal exists, returns the empty list. The output is sorted by residue degree first, then by underlying prime (or equivalently, by norm).

**EXAMPLES:**

```python
sage: x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)

sage: F.primes_above(2)
[Fractional ideal (-t)]
sage: all(2 in P2 for P2 in F.primes_above(2))
True
sage: all(P2.is_prime() for P2 in F.primes_above(2))
True
sage: [P2.norm() for P2 in F.primes_above(2)]
[2]
```

```python
sage: P3s = F.primes_above(3)
sage: P3s
[Fractional ideal (-t)]
```

(continues on next page)
The ideal $(3)$ is totally ramified in $F$, so there is no degree 2 prime above $3$:

sage: F.primes_above(3, degree=2)
[]

sage: [ id.residue_class_degree() for id, _ in F.ideal(3).factor() ]
[1]

Asking for a specific degree works:

sage: P5_1s = F.primes_above(5, degree=1)
sage: P5_1s # random
[Fractional ideal (-t^2 - 1)]
sage: P5_1 = P5_1s[0]; P5_1.residue_class_degree()
1

sage: P5_2s = F.primes_above(5, degree=2)
sage: P5_2s # random
[Fractional ideal (t^2 - 2*t - 1)]
sage: P5_2 = P5_2s[0]; P5_2.residue_class_degree()
2

(continues on next page)
>>> P5_2s # random
[Fractional ideal (t^2 - 2*t - 1)]
>>> P5_2 = P5_2s[Integer(0)]; P5_2.residue_class_degree()
2
Works in relative extensions too:

```python
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = F.ideal(a + 2*b)
sage: P, Q = K.primes_above(I)
sage: K.ideal(I) == P^4*Q
True
sage: K.primes_above(I, degree=Integer(1)) == [P]
True
sage: K.primes_above(I, degree=Integer(4)) == [Q]
True
```

It doesn’t make sense to factor the ideal (0), so this raises an error:

```python
sage: F.prime_above(0)
Traceback (most recent call last):
  ... AttributeError: 'NumberFieldIdeal' object has no attribute 'prime_factors'
```

```python
>>> from sage.all import *
```

```python
>>> F.prime_above(Integer(0))
Traceback (most recent call last):
  ... AttributeError: 'NumberFieldIdeal' object has no attribute 'prime_factors'
```

**primes_of_bounded_norm**(B)

Return a sorted list of all prime ideals with norm at most B.

**INPUT:**

- B – a positive integer or real; upper bound on the norms of the primes generated.

**OUTPUT:**
A list of all prime ideals of this number field of norm at most $B$, sorted by norm. Primes of the same norm are sorted using the comparison function for ideals, which is based on the Hermite Normal Form.

**Note:** See also `primes_of_bounded_norm_iter()` for an iterator version of this, but note that the iterator sorts the primes in order of underlying rational prime, not by norm.

**EXAMPLES:**

```
sage: K.<i> = QuadraticField(-1)
sage: K.primes_of_bounded_norm(10)
[Fractional ideal (i + 1), Fractional ideal (-i - 2),
 Fractional ideal (2*i + 1), Fractional ideal (3)]
sage: K.primes_of_bounded_norm(1)
[]
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^3 - 2)
sage: P = K.primes_of_bounded_norm(30)
sage: P
[Fractional ideal (a),
 Fractional ideal (a + 1),
 Fractional ideal (-a^2 - 1),
 Fractional ideal (a^2 + a - 1),
 Fractional ideal (2*a + 1),
 Fractional ideal (-2*a^2 - a - 1),
 Fractional ideal (a^2 - 2*a - 1),
 Fractional ideal (a + 3)]
sage: [p.norm() for p in P]
[2, 3, 5, 11, 17, 23, 25, 29]
```

```
>>> from sage.all import *
>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> K.primes_of_bounded_norm(Integer(10))
[Fractional ideal (i + 1), Fractional ideal (-i - 2),
 Fractional ideal (2*i + 1), Fractional ideal (3)]
>>> K.primes_of_bounded_norm(Integer(1))
[]
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> P = K.primes_of_bounded_norm(Integer(30))
>>> P
[Fractional ideal (a),
 Fractional ideal (a + 1),
 Fractional ideal (-a^2 - 1),
 Fractional ideal (a^2 + a - 1),
 Fractional ideal (2*a + 1),
 Fractional ideal (-2*a^2 - a - 1),
 Fractional ideal (a^2 - 2*a - 1),
 Fractional ideal (a + 3)]
>>> [p.norm() for p in P]
[2, 3, 5, 11, 17, 23, 25, 29]
```

**primes_of_bounded_norm_iter**($B$)

Iterator yielding all prime ideals with norm at most $B$.

**INPUT:**
• $B$ – a positive integer or real; upper bound on the norms of the primes generated.

**OUTPUT:**

An iterator over all prime ideals of this number field of norm at most $B$.

**Note:** The output is not sorted by norm, but by size of the underlying rational prime.

**EXAMPLES:**

```python
sage: K.<i> = QuadraticField(-1)
sage: it = K.primes_of_bounded_norm_iter(10)
sage: list(it)
[Fractional ideal (i + 1),
 Fractional ideal (3),
 Fractional ideal (-i - 2),
 Fractional ideal (2*i + 1)]
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> it = K.primes_of_bounded_norm_iter(Integer(10))
>>> list(it)
[Fractional ideal (i + 1),
 Fractional ideal (3),
 Fractional ideal (-i - 2),
 Fractional ideal (2*i + 1)]
```

**primes_of_degree_one_iter** (`num_integer_primes=10000`, `max_iterations=100`)

Return an iterator yielding prime ideals of absolute degree one and small norm.

**Warning:** It is possible that there are no primes of $K$ of absolute degree one of small prime norm, and it possible that this algorithm will not find any primes of small norm.

See module `sage.rings.number_field.small_primes_of_degree_one` for details.

**INPUT:**

• `num_integer_primes` – (default: 10000) an integer. We try to find primes of absolute norm no greater than the `num_integer_primes`-th prime number. For example, if `num_integer_primes` is 2, the largest norm found will be 3, since the second prime is 3.

• `max_iterations` – (default: 100) an integer. We test `max_iterations` integers to find small primes before raising `StopIteration`.

**EXAMPLES:**

```python
sage: K.<z> = CyclotomicField(10)
sage: it = K.primes_of_degree_one_iter()
sage: Ps = [ next(it) for i in range(3) ]
sage: Ps # random
[Fractional ideal (z^3 + z + 1),
 Fractional ideal (3*z^3 - z^2 + z - 1),
] (continues on next page)
```
Fractional ideal \((2z^3 - 3z^2 + z - 2)\)

```
sage: [P.norm() for P in Ps] # random
[11, 31, 41]
sage: [P.residue_class_degree() for P in Ps]
[1, 1, 1]
```

```
>>> from sage.all import *

>>> K = CyclotomicField(Integer(10), names=('z',)); (z,) = K._first_ngens(1)
>>> Ps = [ next(it) for i in range(Integer(3)) ]
>>> Ps # random
[Fractional ideal (-z^3 - z^2 + 1),
 Fractional ideal (3*z^3 - z^2 + z - 1),
 Fractional ideal (2*z^3 - 3*z^2 + z - 2)]
```

```
>>> [P.norm() for P in Ps] # random
[11, 31, 41]
>>> [P.residue_class_degree() for P in Ps]
[1, 1, 1]
```

\textbf{primes\_of\_degree\_one\_list}\((n, \text{num}\_\text{integer}\_\text{primes}=10000, \text{max}\_\text{iterations}=100)\)

Return a list of \(n\) prime ideals of absolute degree one and small norm.

\textbf{Warning:} It is possible that there are no primes of \(K\) of absolute degree one of small prime norm, and it is possible that this algorithm will not find any primes of small norm.

See module \texttt{sage.rings.number_field.small_primes\_of\_degree\_one} for details.

\textbf{INPUT:}

- \texttt{num\_integer\_primes} – (default: 10000) an integer. We try to find primes of absolute norm no greater than the \texttt{num\_integer\_primes}-th prime number. For example, if \texttt{num\_integer\_primes} is 2, the largest norm found will be 3, since the second prime is 3.

- \texttt{max\_iterations} – (default: 100) an integer. We test \texttt{max\_iterations} integers to find small primes before raising \textit{StopIteration}.

\textbf{EXAMPLES:}

```
sage: K.<z> = CyclotomicField(10)
sage: Ps = K.primes\_of\_degree\_one\_list(3)
sage: Ps # random output
[Fractional ideal (-z^3 - z^2 + 1),
 Fractional ideal (3*z^3 - z^2 + z - 1),
 Fractional ideal (2*z^3 - 3*z^2 + z - 2)]
sage: [P.norm() for P in Ps] # random
[11, 31, 41]
```

```
>>> from sage.all import *

>>> K = CyclotomicField(Integer(10), names=('z',)); (z,) = K._first_ngens(1)
>>> Ps = K.primes\_of\_degree\_one\_list(Integer(3))
>>> Ps # random output
[Fractional ideal (-z^3 - z^2 + 1),
 Fractional ideal (2*z^3 - 2*z^2 + 2*z - 3),
 Fractional ideal (2*z^3 - 3*z^2 + z - 2)]
```
Fractional ideal \((2z^3 - 3z^2 + z - 2)\)

```python
>>> [P.norm() for P in Ps]
[11, 31, 41]
```

```python
>>> [P.residue_class_degree() for P in Ps]
[1, 1, 1]
```

**primitive_element()**

Return a primitive element for this field, i.e., an element that generates it over \(\mathbb{Q}\).

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2)
sage: K.primitive_element()
a
sage: alpha = K.primitive_element(); alpha
a
sage: alpha.minpoly()
x^2 + (2*b - 2*c)*x - 2*c*b + 6
sage: alpha.absolute_minpoly()
x^8 - 40*x^6 + 352*x^4 - 960*x^2 + 576
```

```python
>>> from sage.all import *
```

```python
>>> x = polygen(ZZ, 'x')
```

```python
>>> K = NumberField(x**Integer(3) + Integer(2), names=('a',)); (a,) = K._first_ngens(1)
```

```python
>>> K.primitive_element()
a
```

```python
>>> K = NumberField([x**Integer(2) - Integer(2), x**Integer(2) - Integer(3), x**Integer(2) - Integer(5)], names=(a, b, c,)); (a, b, c,) = K._first_ngens(3)
```

```python
>>> K.primitive_element()
a - b + c
```

```python
>>> alpha = K.primitive_element(); alpha
a - b + c
```

```python
>>> alpha.minpoly()
x^2 + (2*b - 2*c)*x - 2*c*b + 6
```

```python
>>> alpha.absolute_minpoly()
x^8 - 40*x^6 + 352*x^4 - 960*x^2 + 576
```

**primitive_root_of_unity()**

Return a generator of the roots of unity in this field.

**OUTPUT:** a primitive root of unity. No guarantee is made about which primitive root of unity this returns, not even for cyclotomic fields. Repeated calls of this function may return a different value.

**Note:** We do not create the full unit group since that can be expensive, but we do use it if it is already known.

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
```

(continues on next page)
We do not special-case cyclotomic fields, so we do not always get the most obvious primitive root of unity:

```
sage: K.<a> = CyclotomicField(3)
sage: z = K.primitive_root_of_unity(); z
a + 1
sage: z.multiplicative_order()
6
```

```bash
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> z = K.primitive_root_of_unity(); z
i
>>> z.multiplicative_order()
4
```

```bash
>>> K = NumberField(x**Integer(2) + x + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> z = K.primitive_root_of_unity(); z
a + 1
>>> z.multiplicative_order()
6
```

```bash
>>> x = polygen(QQ)
>>> F = NumberField((x**Integer(2) - Integer(2), x**Integer(2) - Integer(3)),
                  names=('a', 'b')); (a, b,) = F._first_ngens(2)
>>> y = polygen(F)
>>> K = F.extension(y**Integer(2) - (Integer(1) + a)*(a + b)*a*b, names=('c',
                 )); (c,) = K._first_ngens(1)
>>> K.primitive_root_of_unity()
-1
```
from sage.all import *
K = CyclotomicField(Integer(3), names=('a',)); (a,) = K._first_ngens(1)
z = K.primitive_root_of_unity(); z
a + 1
z.multiplicative_order()
6

K = CyclotomicField(Integer(3))
z = K.primitive_root_of_unity(); z
zeta3 + 1
z.multiplicative_order()
6

quadratic_defect (a, p, check=True)

Return the valuation of the quadratic defect of \( a \) at \( p \).

INPUT:

- \( a \) – an element of \( self \)
- \( p \) – a prime ideal
- \( check \) – (default: True); check if \( p \) is prime

ALGORITHM:

This is an implementation of Algorithm 3.1.3 from [Kir2016]

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x**2 + 2)
sage: p = K.primes_above(2)[0]
sage: K.quadratic_defect(5, p)
4
sage: K.quadratic_defect(0, p)
+Infinity
sage: K.quadratic_defect(a, p)
1
sage: K.<a> = CyclotomicField(5)
sage: p = K.primes_above(2)[0]
sage: K.quadratic_defect(5, p)
+Infinity

Note: The output for the quadratic defect depends on the specific implementation and may not directly match the expectations from the original code snippet.
random_element (num_bound=None, den_bound=None, integral_coefficients=False, distribution=None)

Return a random element of this number field.

INPUT:

- num_bound – Bound on numerator of the coefficients of the resulting element
- den_bound – Bound on denominators of the coefficients of the resulting element
- integral_coefficients – (default: False) If True, then the resulting element will have integral coefficients. This option overrides any value of den_bound.
- distribution – Distribution to use for the coefficients of the resulting element

OUTPUT: Element of this number field

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<j> = NumberField(x^8 + 1)
sage: K.random_element().parent() is K
True

sage: while K.random_element().list()[0] != 0:
    ....:     pass
sage: while K.random_element().list()[0] == 0:
    ....:     pass
sage: while K.random_element().is_prime():
    ....:     pass
sage: while not K.random_element().is_prime():
    ....:     pass

sage: K.<a,b,c> = NumberField([x^2 - 2, x^2 - 3, x^2 - 5])
sage: K.random_element().parent() is K
True

sage: while K.random_element().is_prime():
    ....:     pass
sage: while not K.random_element().is_prime(): # long time
    ....:     pass

sage: K.<a> = NumberField(x^5 - 2)
sage: p = K.random_element(integral_coefficients=True)
sage: p.is_integral()
True

sage: while K.random_element().is_integral():
    ....:     pass
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(8) + Integer(1), names=('j',)); (j,) = K._first_ngens(1)
>>> K.random_element().parent() is K
True

>>> while K.random_element().list()[Integer(0)] != Integer(0):
    ...     pass
>>> while K.random_element().list()[Integer(0)] == Integer(0):
    ...     pass
>>> while K.random_element('').is_prime():
```
... pass

>>> while not K.random_element().is_prime():
... pass

>>> K = NumberField([x**Integer(2) - Integer(2), x**Integer(2) - Integer(3),
                                         ˓→x**Integer(2) - Integer(5)], names=('a', 'b', 'c',)); (a, b, c,) = K._first_ngens(3)
>>> K.random_element().parent() is K
True

>>> while K.random_element().is_prime():
... pass
>>> while not K.random_element().is_prime():  # long time
... pass

>>> K = NumberField(x**Integer(5) - Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> p = K.random_element(integral_coefficients=True)
>>> p.is_integral()
True

real_embeddings (prec=53)

Return all homomorphisms of this number field into the approximate real field with precision prec.

If prec is 53 (the default), then the real double field is used; otherwise the arbitrary precision (but slow)
real field is used. If you want embeddings into the 53-bit double precision, which is faster, use self. embeddings(RDF).

Note: This function uses finite precision real numbers. In functions that should output proven results, one
could use self.embeddings(AA) instead.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^3 + 2)
sage: K.real_embeddings()
[Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Real Field with 53 bits of precision
  Defn: a |--> -1.25992104989487]
sage: K.real_embeddings(16)
[
  Ring morphism:
    From: Number Field in a with defining polynomial x^3 + 2
    To:   Real Field with 16 bits of precision
    Defn: a |--> -1.260
]
sage: K.real_embeddings(100)
[
  Ring morphism:
    From: Number Field in a with defining polynomial x^3 + 2
    To:   Real Field with 100 bits of precision
    Defn: a |--> -1.2599210498948731707...

(continues on next page)
To: Real Field with 100 bits of precision
Defn: a |--> -1.2599210498948731647672106073

```python
>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> K = NumberField(x**Integer(3) + Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)

>>> K.real_embeddings()
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Real Field with 53 bits of precision
        Defn: a |--> -1.25992104989487
]

>>> K.real_embeddings(Integer(16))
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Real Field with 16 bits of precision
        Defn: a |--> -1.260
]

>>> K.real_embeddings(Integer(100))
[
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Real Field with 100 bits of precision
        Defn: a |--> -1.2599210498948731647672106073
]

As this is a numerical function, the number of embeddings may be incorrect if the precision is too low:

```python
sage: K = NumberField(x^2 + 2*10^1000*x + 10^2000 + 1, 'a')

sage: len(K.real_embeddings())
2

sage: len(K.real_embeddings(100))
2

sage: len(K.real_embeddings(10000))
0

sage: len(K.embeddings(AA))
0
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(2) + Integer(2)*Integer(10)**Integer(1000)*x +
˓→Integer(10)**Integer(2000) + Integer(1), 'a')

>>> len(K.real_embeddings())
2

>>> len(K.real_embeddings(Integer(100)))
2

>>> len(K.real_embeddings(Integer(10000)))
0

>>> len(K.embeddings(AA))
0
```

**reduced_basis** (*prec=None*)

Return an LLL-reduced basis for the Minkowski-embedding of the maximal order of a number field.
INPUT:

- \texttt{prec} (default: \texttt{None}) – the precision with which to compute the Minkowski embedding.

OUTPUT:

An LLL-reduced basis for the Minkowski-embedding of the maximal order of a number field, given by a sequence of (integral) elements from the field.

\textbf{Note:} In the non-totally-real case, the LLL routine we call is currently PARI’s \texttt{pari:qflll}, which works with floating point approximations, and so the result is only as good as the precision promised by PARI. The matrix returned will always be integral; however, it may only be only “almost” LLL-reduced when the precision is not sufficiently high.

\textbf{EXAMPLES:}

```sage
sage: x = polygen(QQ, 'x')
sage: F.<t> = NumberField(x^6 - 7*x^4 - x^3 + 11*x^2 + x - 1)
sage: F.maximal_order().basis()
[1/2*t^5 + 1/2*t^4 + 1/2*t^2 + 1/2, t, t^2, t^3, t^4, t^5]
sage: F.reduced_basis()
[-1, -1/2*t^5 + 1/2*t^4 + 3*t^3 - 3/2*t^2 - 4*t - 1/2, t, 1/2*t^5 + 1/2*t^4 - 4*t^3 - 5/2*t^2 + 7*t + 1/2, 1/2*t^5 - 1/2*t^4 - 2*t^3 + 3/2*t^2 - 1/2, 1/2*t^5 - 1/2*t^4 + 3*t^3 + 5/2*t^2 + 4*t - 5/2]
sage: CyclotomicField(12).reduced_basis()
[1, zeta12^2, zeta12, zeta12^3]
```

```python
>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> F = NumberField(x**Integer(6) - Integer(7)*x**Integer(4) - x**Integer(3) - Integer(11)*x**Integer(2) + x - Integer(1), names=('t',)); (t,) = F._first_ngens(1)

>>> F.maximal_order().basis()
[1/2*t^5 + 1/2*t^4 + 1/2*t^2 + 1/2, t, t^2, t^3, t^4, t^5]

>>> F.reduced_basis()
[-1, -1/2*t^5 + 1/2*t^4 + 3*t^3 - 3/2*t^2 - 4*t - 1/2, t, 1/2*t^5 + 1/2*t^4 - 4*t^3 - 5/2*t^2 + 7*t + 1/2, 1/2*t^5 - 1/2*t^4 - 2*t^3 + 3/2*t^2 - 1/2, 1/2*t^5 - 1/2*t^4 + 3*t^3 + 5/2*t^2 + 4*t - 5/2]

>>> CyclotomicField(Integer(12)).reduced_basis()
[1, zeta12^2, zeta12, zeta12^3]
```

\texttt{reduced\_gram\_matrix}(\texttt{prec=\texttt{None}})

Return the Gram matrix of an LLL-reduced basis for the Minkowski embedding of the maximal order of a number field.

INPUT:

- \texttt{prec} (default: \texttt{None}) – the precision with which to calculate the Minkowski embedding. (See \textbf{NOTE} below.)

OUTPUT: The Gram matrix \([x_i, x_j]\) of an LLL reduced basis for the maximal order of \texttt{self}, where the integral basis for \texttt{self} is given by \{x_0, …, x_{n-1}\}. Here \langle, \rangle is the usual inner product on \( \mathbb{R}^n \), and \texttt{self} is embedded in \( \mathbb{R}^n \) by the Minkowski embedding. See the docstring for \texttt{NumberField\_absolute.minkowski\_embedding()} for more information.

1.1. Number fields
Note: In the non-totally-real case, the LLL routine we call is currently PARI’s `pari:qflll`, which works with floating point approximations, and so the result is only as good as the precision promised by PARI. In particular, in this case, the returned matrix will not be integral, and may not have enough precision to recover the correct Gram matrix (which is known to be integral for theoretical reasons). Thus the need for the `prec` parameter above.

If the following run-time error occurs: “PariError: not a definite matrix in lllgram (42)”, try increasing the `prec` parameter.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: F.<t> = NumberField(x^6 - 7*x^4 - x^3 + 11*x^2 + x - 1)
sage: F.reduced_gram_matrix()
[ 6 3 0 2 0 1]
[ 3 9 0 1 0 -2]
[ 0 0 14 6 -2 3]
[ 2 1 16 6 -3 3]
[ 0 0 -2 -3 16 6]
[ 1 -2 3 3 6 19]
sage: Matrix(6, [(x*y).trace()
    ....: for x in F.integral_basis() for y in F.integral_basis()
]
[2550 133 259 664 1368 3421]
[133 14 3 54 30 233]
[259 3 54 30 233 217]
[664 54 30 233 217 1078]
[1368 30 233 217 1078 1371]
[3421 233 217 1078 1371 5224]
```

```python
>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> F = NumberField(x**Integer(6) - Integer(7)*x**Integer(4) - x**Integer(3)...
 ˓→ Integer(11)*x**Integer(2) + x - Integer(1), names=('t',)); (t,) = F._
 ˓→ first_ngens(1)
>>> F.reduced_gram_matrix()
[ 6 3 0 2 0 1]
[ 3 9 0 1 0 -2]
[ 0 0 14 6 -2 3]
[ 2 1 16 6 -3 3]
[ 0 0 -2 -3 16 6]
[ 1 -2 3 3 6 19]
```

```python
sage: x = polygen(QQ)
sage: F.<alpha> = NumberField(x^4 + x^2 + 712312*x + 131001238)
sage: F.reduced_gram_matrix(prec=128)
[ 4.00000000000000000000000000000000000000 0.00000000000000000000000000000000000000 0.00000000000000000000000000000000000000 -1.00000000000000000000000000000000000000]
```

(continues on next page)
\[ \begin{pmatrix} 4.0000000000000000000000000000000000000 & 0. \\ 0.0000000000000000000000000000000000000 & 1.9999999999999999999999999999999999037 -11488. \\ 910026517242751227497036149666768 -418. \\ 12718083977141198754424579680468382 \\
\end{pmatrix} \]

\[ \begin{pmatrix} 0.0000000000000000000000000000000000000 & 539331563218381658483353092335550 -11488. \\ 910026517242751227497036149666768 -418. \\ 12718083977141198754424579680468382 \\
\end{pmatrix} \]

\[ \begin{pmatrix} -1.9999999999999999999999999999999999037 -11488. \\ 910026517242751227497036149666768 5. \\ 5658915310500611768713076521847709187e8 1. \\ 417909227149407005043336847682152174e8 1. \\ -0.9999999999999999999999999999999383702 -418. \\ 12718083977141198754424579680468382 1. \\ 417909227149407005043336847682152174e8 1. \\ 3665897267919181137884111201405279175e12 \\
\end{pmatrix} \]

```python
>>> from sage.all import *
```

```python
>>> x = polygen(QQ)
```

```python
>>> F = NumberField(x**Integer(4) + x**Integer(2) + Integer(712312)*x + Integer(131001238), names=('alpha',)); (alpha,) = F._first_ngens(1)
```

```python
>>> F.reduced_gram_matrix(prec=Integer(128))
```

### regulator

**proof=None**

Return the regulator of this number field.

Note that PARI computes the regulator to higher precision than the Sage default.

**INPUT:**

- proof = default: True, unless you set it otherwise.

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
```

```python
sage: NumberField(x^2 - 2, 'a').regulator()
```

```
0.881373587019543
```

```python
sage: NumberField(x^4 + x^3 + x^2 + x + 1, 'a').regulator()
```

```
0.962423650119207
```

```python
>>> from sage.all import *
```

```python
>>> x = polygen(QQ, 'x')
```

```python
>>> NumberField(x**Integer(2) - Integer(2), a).regulator()
```

(continues on next page)
residue_field (prime, names=None, check=True)

Return the residue field of this number field at a given prime, i.e. \( O_K / pO_K \).

**INPUT:**

- prime – a prime ideal of the maximal order in this number field, or an element of the field which generates a principal prime ideal.
- names – the name of the variable in the residue field
- check – whether or not to check the primality of `prime`

**OUTPUT:** The residue field at this prime.

**EXAMPLES:**

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4 + 3*x^2 - 17)
sage: P = K.ideal(61).factor()[0][0]
sage: K.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)
```

```python
>>> from sage.all import *
```

```python
R = QQ[x]; (x,) = R._first_ngens(1)
K = NumberField(x**Integer(4) + Integer(3)*x**Integer(2) - Integer(17),
               names=('a',)); (a,) = K._first_ngens(1)
P = K.ideal(Integer(61)).factor()[Integer(0)][Integer(0)]
K.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)
```

```python
sage: K.<i> = NumberField(x^2 + 1)
sage: K.residue_field(1+i)
Residue field of Fractional ideal (i + 1)
```

```python
>>> from sage.all import *
```

```python
K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
               first_ngens(1)
K.residue_field(Integer(1)+i)
Residue field of Fractional ideal (i + 1)
```

roots_of_unity()

Return all the roots of unity in this field, primitive or not.

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: K.<b> = NumberField(x^2 + 1)
sage: zs = K.roots_of_unity(); zs
[1, -1, I, -I]
sage: [z**K.number_of_roots_of_unity() for z in zs]
[1, 1, 1, 1]
```
selmer_generators \((S, m, \text{proof}=\text{True}, \text{orders}=\text{False})\)

Compute generators of the group \(K(S, m)\).

INPUT:
- \(S\) – a set of primes of self
- \(m\) – a positive integer
- \(\text{proof}\) – if False, assume the GRH in computing the class group
- \(\text{orders}\) – (default: False) if True, output two lists, the generators and their orders

OUTPUT:
A list of generators of \(K(S, m)\), and (optionally) their orders as elements of \(K^\times/(K^\times)^m\). This is the subgroup of \(K^\times/(K^\times)^m\) consisting of elements \(a\) such that the valuation of \(a\) is divisible by \(m\) at all primes not in \(S\). It fits in an exact sequence between the units modulo \(m\)-th powers and the \(m\)-torsion in the \(S\)-class group:

\[ 1 \rightarrow O_{K,S}^\times/(O_{K,S}^\times)^m \rightarrow K(S, m) \rightarrow \text{Cl}_{K,S}[m] \rightarrow 0. \]

The group \(K(S, m)\) contains the subgroup of those \(a\) such that \(K(\sqrt[m]{a})/K\) is unramified at all primes of \(K\) outside of \(S\), but may contain it properly when not all primes dividing \(m\) are in \(S\).

See also:

- `NumberField_generic.selmer_space()`, which gives additional output when \(m = p\) is prime: as well as generators, it gives an abstract vector space over \(\mathbb{F}_p\) isomorphic to \(K(S, p)\) and maps implementing the isomorphism between this space and \(K(S, p)\) as a subgroup of \(K^\times/(K^\times)^p\).

EXAMPLES:

```python
sage: K.<a> = QuadraticField(-5)
sage: K.selmer_generators((), 2)
[-1, 2]
```

```python
sage: K = QuadraticField(-Integer(5), names=('a',)); (a,) = K._first_ngens(1)
sage: K.selmer_generators((), Integer(2))
[-1, 2]
```

The previous example shows that the group generated by the output may be strictly larger than the group of elements giving extensions unramified outside \(S\), since that has order just 2, generated by \(-1\):

```python
sage: K.class_number() 2
sage: K.hilbert_class_field('b')
Number Field in b with defining polynomial x^2 + 1 over its base field
```
When $m$ is prime all the orders are equal to $m$, but in general they are only divisors of $m$:

```
sage: K.<a> = QuadraticField(-5)
sage: P2 = K.ideal(2, -a + 1)
sage: P3 = K.ideal(3, a + 1)
sage: K.selmer_generators([], 2, orders=True)  
([-1, 2], [2, 2])
sage: K.selmer_generators([], 4, orders=True)  
([-1, 4], [2, 2])
sage: K.selmer_generators([P2], 2)  
[2, -1]
sage: K.selmer_generators([P2, P3], 4)  
[2, -a - 1, -1]
sage: K.selmer_generators([P2, P3], 4, orders=True)  
([2, -a - 1, -1], [4, 4, 2])
sage: K.selmer_generators([P2], 3)  
[2]
sage: K.selmer_generators([P2, P3], 3)  
[2, -a - 1]
sage: K.selmer_generators([P2, P3, K.ideal(a)], 3)  # random signs  
[2, a + 1, a]
```

Example over $\mathbb{Q}$ (as a number field):

```
sage: K.<a> = NumberField(polygen(QQ))
sage: K.selmer_generators([],5)  
[]
sage: K.selmer_generators([K.prime_above(p) for p in [2,3,5]], 2)  
[2, 3, 5, -1]
sage: K.selmer_generators([K.prime_above(p) for p in [2,3,5]], 6, orders=True)  
```

(continues on next page)
selmer_group_iterator \((S, m, proof=True)\)

Return an iterator through elements of the finite group \(K(S, m)\).

INPUT:

- \(S\) – a set of primes of self
- \(m\) – a positive integer
- \(proof\) – if False, assume the GRH in computing the class group

OUTPUT:

An iterator yielding the distinct elements of \(K(S, m)\). See the docstring for NumberField_generic.

selmer_generators() for more information.

EXAMPLES:

```python
sage: K.\langle a \rangle = QuadraticField\((-5)\)
sage: list(K.selmer_group_iterator(), 2))
[1, 2, -1, -2]
sage: list(K.selmer_group_iterator(), 4))
[1, 4, -1, -4]
sage: list(K.selmer_group_iterator([K.ideal(2, a + 1)], 2))
[1, -1, 2, -2]
sage: list(K.selmer_group_iterator([K.ideal(2, a + 1), K.ideal(3, a + 1)], 2))
[1, -1, -a - 1, a + 1, 2, -2, -2*a - 2, 2*a + 2]
```

Examples over \(\mathbb{Q}\) (as a number field):

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> list(K.selmer_group_iterator(), Integer(2)))
[1, 2, -1, -2]
>>> list(K.selmer_group_iterator(), Integer(4)))
[1, 4, -1, -4]
>>> list(K.selmer_group_iterator([K.ideal(Integer(2), a + Integer(1))], Integer(2)))
[1, -1, 2, -2]
>>> list(K.selmer_group_iterator([K.ideal(Integer(2), a + Integer(1)), K.
   .ideal(Integer(3), a + Integer(1))], Integer(2)))
[1, -1, -a - 1, a + 1, 2, -2, -2*a - 2, 2*a + 2]
```
Compute the group $K(S, p)$ as a vector space with maps to and from $K^*$.  

**INPUT:**  
- $S$ – a set of primes ideals of self  
- $p$ – a prime number  
- proof – if False, assume the GRH in computing the class group  

**OUTPUT:**  
(tuple) $K_S, K_S\_gens, from\_K_S, to\_K_S$ where  
- $K_S$ is an abstract vector space over $GF(p)$ isomorphic to $K(S, p)$;  
- $K_S\_gens$ is a list of elements of $K^*$ generating $K(S, p)$;  
- $from\_K_S$ is a function from $K_S$ to $K^*$ implementing the isomorphism from the abstract $K(S, p)$ to $K(S, p)$ as a subgroup of $K^*/(K^*)^p$;  
- $to\_K_S$ is a partial function from $K^*$ to $K_S$, defined on elements $a$ whose image in $K^*/(K^*)^p$ lies in $K(S, p)$, mapping them via the inverse isomorphism to the abstract vector space $K_S$.

The group $K(S, p)$ is the finite subgroup of $K^*/(K^*)^p$ consisting of elements whose valuation at all primes not in $S$ is a multiple of $p$. It contains the subgroup of those $a \in K^*$ such that $K(\sqrt[p]{a})/K$ is unramified at all primes of $K$ outside of $S$, but may contain it properly when not all primes dividing $p$ are in $S$.

**EXAMPLES:**

A real quadratic field with class number 2, where the fundamental unit is a generator, and the class group provides another generator when $p = 2$:

```
sage: K.<a> = QuadraticField(-5)
sage: K.class_number()
2
sage: P2 = K.ideal(2, -a + 1)
sage: P3 = K.ideal(3, a + 1)
sage: P5 = K.ideal(a)
sage: KS2, gens, fromKS2, toKS2 = K.selmer_space([P2, P3, P5], 2)
sage: KS2
Vector space of dimension 4 over Finite Field of size 2
```
To continue from the previous page,

```python
sage: gens
[a + 1, a, 2, -1]
```

```python
>>> from sage.all import *

>>> K = QuadraticField(-Integer(5), names=('a',)); (a,) = K._first_ngens(1)

>>> K.class_number()
2

>>> P2 = K.ideal(Integer(2), -a + Integer(1))

>>> P3 = K.ideal(Integer(3), a + Integer(1))

>>> P5 = K.ideal(a)

>>> KS2, gens, fromKS2, toKS2 = K.selmer_space([P2, P3, P5], Integer(2))

>>> KS2
Vector space of dimension 4 over Finite Field of size 2

>>> gens
[a + 1, a, 2, -1]

Each generator must have even valuation at primes not in $S$:

```python
sage: [K.ideal(g).factor() for g in gens]
[(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1)),
 Fractional ideal (a),
 (Fractional ideal (2, a + 1))^2,
 1]

sage: toKS2(10)
(0, 0, 1, 1)

sage: fromKS2([0,0,1,1])
-2

sage: K(10/(-2)).is_square()
True

sage: KS3, gens, fromKS3, toKS3 = K.selmer_space([P2, P3, P5], 3)

sage: KS3
Vector space of dimension 3 over Finite Field of size 3

sage: gens
[1/2, 1/4*a + 1/4, a]
```

```python
>>> from sage.all import *

>>> [K.ideal(g).factor() for g in gens]
[(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1)),
 Fractional ideal (a),
 (Fractional ideal (2, a + 1))^2,
 1]

>>> toKS2(Integer(10))
(0, 0, 1, 1)

>>> fromKS2([Integer(0),Integer(0),Integer(1),Integer(1)])
-2

>>> K(Integer(10)/(-Integer(2))).is_square()
True

>>> KS3, gens, fromKS3, toKS3 = K.selmer_space([P2, P3, P5], Integer(3))

>>> KS3
Vector space of dimension 3 over Finite Field of size 3

>>> gens
[1/2, 1/4*a + 1/4, a]
```
An example to show that the group $K(S, 2)$ may be strictly larger than the group of elements giving extensions unramified outside $S$. In this case, with $K$ of class number 2 and $S$ empty, there is only one quadratic extension of $K$ unramified outside $S$, the Hilbert Class Field $K(\sqrt{-1})$:

```
sage: K.<a> = QuadraticField(-5)
sage: KS2, gens, fromKS2, toKS2 = K.selmer_space([], 2)
sage: KS2
Vector space of dimension 2 over Finite Field of size 2
sage: gens
[2, -1]
sage: x = polygen(ZZ, 'x')
sage: for v in KS2:
    ....:     if not v:
    ....:         continue
    ....:     a = fromKS2(v)
    ....:     print((a, K.extension(x^2 - a, 'roota').relative_discriminant().factor()))
(2, (Fractional ideal (2, a + 1))^4)
(-1, 1)
(-2, (Fractional ideal (2, a + 1))^4)
sage: K.hilbert_class_field('b')
Number Field in b with defining polynomial x^2 + 1 over its base field
```
solve_CRT (reslist, Ilist, check=True)
Solve a Chinese remainder problem over this number field.

INPUT:

- reslist – a list of residues, i.e. integral number field elements
- Ilist – a list of integral ideals, assumed pairwise coprime
- check – (boolean, default True) if True, result is checked

OUTPUT:
An integral element $x$ such that $x - \text{reslist}[i]$ is in $\text{Ilist}[i]$ for all $i$.

Note: The current implementation requires the ideals to be pairwise coprime. A more general version would be possible.

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 - 10)
sage: Ilist = [K.primes_above(p)[0] for p in prime_range(10)]
sage: b = K.solve_CRT([1,2,3,4], Ilist, True)
sage: all(b - i - 1 in Ilist[i] for i in range(4))
True
sage: Ilist = [K.ideal(a), K.ideal(2)]
sage: K.solve_CRT([0,1], Ilist, True)
Traceback (most recent call last):
... ArithmeticError: ideals in solve_CRT() must be pairwise coprime
```
some_elements()  
Return a list of elements in the given number field.

EXAMPLES:

```
sage: R.<t> = QQ[]
sage: K.<a> = QQ.extension(t^2 - 2); K
Number Field in a with defining polynomial t^2 - 2
sage: K.some_elements()
[1, a, 2*a, 3*a - 4, 1/2, 1/3*a, 1/6*a, 0, 1/2*a, 2, ..., 12, -12*a + 18]
sage: T.<u> = K[

>>> from sage.all import *
>>> R = QQ['t']; (t,) = R._first_ngens(1)
>>> K = QQ.extension(t**Integer(2) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1); K
Number Field in a with defining polynomial t^2 - 2
>>> K.some_elements()
[1, a, 2*a, 3*a - 4, 1/2, 1/3*a, 1/6*a, 0, 1/2*a, 2, ..., 12, -12*a + 18]

>>> T = K['u']; (u,) = T._first_ngens(1)
>>> M = K.extension(t**Integer(3) - Integer(5), names=('b',)); (b,) = M._
˓→first_ngens(1); M
Number Field in b with defining polynomial t^3 - 5 over its base field
>>> M.some_elements()
[1, b, 1/2*a*b, ..., 2/5*b^2 + 2/5, 1/6*b^2 + 5/6*b + 13/6, 2]
```

specified_complex_embedding()  
Return the embedding of this field into the complex numbers which has been specified.

Fields created with the QuadraticField() or CyclotomicField() constructors come with an implicit embedding. To get one of these fields without the embedding, use the generic NumberField constructor.

EXAMPLES:

```
sage: QuadraticField(-1, 'I').specified_complex_embedding()
Generic morphism:
  From: Number Field in I with defining polynomial x^2 + 1 with I = 1*I
  To:  Complex Lazy Field
  Defn: I -> 1*I

>>> from sage.all import *
>>> QuadraticField(-Integer(1), 'I').specified_complex_embedding()
Generic morphism:
  From: Number Field in I with defining polynomial x^2 + 1 with I = 1*I
  To:  ComplexLazyField
  Defn: I -> 1*I

sage: QuadraticField(3, 'a').specified_complex_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^2 - 3

(continues on next page)
Algebraic Numbers and Number Fields, Release 10.4

with \( a = 1.732050807568878? \)
To: Real Lazy Field
Defn: \( a \rightarrow 1.732050807568878? \)

```python
>>> from sage.all import *
```

```python
QuadraticField(Integer(3), 'a').specified_complex_embedding()
```
Generic morphism:
From: Number Field in a with defining polynomial \( x^2 - 3 \)
with \( a = 1.732050807568878? \)
To: Real Lazy Field
Defn: \( a \rightarrow 1.732050807568878? \)

```python
sage: CyclotomicField(13).specified_complex_embedding()
```
Generic morphism:
From: Cyclotomic Field of order 13 and degree 12
To: Complex Lazy Field
Defn: \( zeta_{13} \rightarrow 0.885456025653210? + 0.464723172043769?*I \)

```python
>>> from sage.all import *
```

```python
CyclotomicField(Integer(13)).specified_complex_embedding()
```
Generic morphism:
From: Cyclotomic Field of order 13 and degree 12
To: Complex Lazy Field
Defn: \( zeta_{13} \rightarrow 0.885456025653210? + 0.464723172043769?*I \)

Most fields don’t implicitly have embeddings unless explicitly specified:

```python
sage: x = polygen(QQ, 'x')
sage: NumberField(x**Integer(2) - Integer(2), 'a').specified_complex_embedding() is None
```
True

```python
sage: NumberField(x**Integer(3) - x + Integer(5), 'a').specified_complex_embedding() is None
```
True

```python
sage: NumberField(x**Integer(3) - x + Integer(5), 'a', embedding=Integer(2)).specified_complex_embedding()
```
Generic morphism:
From: Number Field in a with defining polynomial \( x^3 - x + 5 \)
with \( a = -1.904160859134921? \)
To: Real Lazy Field
Defn: \( a \rightarrow -1.904160859134921? \)

```python
sage: NumberField(x**Integer(3) - x + Integer(5), 'a', embedding=CDF.0).specified_complex_embedding()
```
Generic morphism:
From: Number Field in a with defining polynomial \( x^3 - x + 5 \)
with \( a = 0.952080429567461? + 1.311248044077123?*I \)
To: Complex Lazy Field
Defn: \( a \rightarrow 0.952080429567461? + 1.311248044077123?*I \)

(continues from previous page)
This function only returns complex embeddings:

```
sage: # needs sage.rings.padics
sage: K.<a> = NumberField(x^2 - 2, embedding=Qp(7)(2).sqrt())
sage: K.specified_complex_embedding() is None
True
sage: K.gen_embedding()
3 + 7 + 2*7^2 + 6*7^3 + 7^4 + 2*7^5 + 7^6 + 2*7^7 + 4*7^8 + 6*7^9 + 6*7^10 + 2*7^11 + 7^12 + 7^13 + 2*7^15 + 7^16 + 7^17 + 4*7^18 + 6*7^19 + O(7^20)
sage: K.coerce_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^2 - 2
  with a = 3 + 7 + 2*7^2 + 6*7^3 + 7^4 + 2*7^5 + 7^6 + 2*7^7 + 4*7^8 + 6*7^9 + 6*7^10 + 2*7^11 + 7^12 + 7^13 + 2*7^15 + 7^16 + 7^17 + 4*7^18 + 6*7^19 + O(7^20)
  To: 7-adic Field with capped relative precision 20
  Defn: a -> 3 + 7 + 2*7^2 + 6*7^3 + 7^4 + 2*7^5 + 7^6 + 2*7^7 + 4*7^8 + 6*7^9 + 6*7^10 + 2*7^11 + 7^12 + 7^13 + 2*7^15 + 7^16 + 7^17 + 4*7^18 + 6*7^19 + O(7^20)
```
This is used to record various isomorphisms or embeddings that arise naturally in other constructions.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<z> = NumberField(x^2 + 3)
sage: L.<a> = K.absolute_field(); L
Number Field in a with defining polynomial x^2 + 3
sage: L.structure()
(Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 + 3
  To: Number Field in z with defining polynomial x^2 + 3,
Isomorphism given by variable name change map:
  From: Number Field in z with defining polynomial x^2 + 3
  To: Number Field in a with defining polynomial x^2 + 3)
sage: K.<a> = QuadraticField(-3)
sage: R.<y> = K[]
sage: D.<x0> = K.extension(y)
sage: D_abs.<y0> = D.absolute_field()
sage: D_abs.structure()[0](y0)
-a
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(3), names=('z',)); (z,) = K._first_ngens(1)
>>> L = K.absolute_field(names=('a',)); (a,) = L._first_ngens(1); L
Number Field in a with defining polynomial x^2 + 3
>>> L.structure()
(Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 + 3
  To: Number Field in z with defining polynomial x^2 + 3,
Isomorphism given by variable name change map:
  From: Number Field in z with defining polynomial x^2 + 3
  To: Number Field in a with defining polynomial x^2 + 3)
```

```python
>>> K = QuadraticField(-Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> D = K.extension(y, names=('x0',)); (x0,) = D._first_ngens(1)
>>> D_abs = D.absolute_field(names=('y0',)); (y0,) = D_abs._first_ngens(1)
>>> D_abs.structure()[Integer(0)](y0)
-a
```

### subfield (alpha, name=None, names=None)

Return a number field $K$ isomorphic to $\mathbb{Q}(\alpha)$ (if this is an absolute number field) or $L(\alpha)$ (if this is a relative extension $M/L$) and a map from $K$ to self that sends the generator of $K$ to alpha.

**INPUT:**

- $\alpha$ – an element of self, or something that coerces to an element of self.

**OUTPUT:**

- $K$ – a number field

- from$_K$ – a homomorphism from $K$ to self that sends the generator of $K$ to alpha.

**EXAMPLES:**
A relative example. Note that the result returned is the subfield generated by $\alpha$ over `self.base_field()`, not over $\mathbb{Q}$ (see Issue #5392):

```python
sage: L.<a> = NumberField(x^2 - 3)
sage: M.<b> = L.extension(x^4 + 1)
sage: K, phi = M.subfield(b^2)
sage: K.base_field() is L
True
```

Subfields inherit embeddings:

```python
sage: K.<z> = CyclotomicField(5)
sage: L, K_from_L = K.subfield(z - z^2 - z^3 + z^4)
sage: L
Number Field in z0 with defining polynomial x^2 - 5 with z0 = 236079749970?
```

(continues on next page)
1.1. Number fields

Generic morphism:
From: Cyclotomic Field of order 5 and degree 4
To: Complex Lazy Field
Defn: z -> 0.309016994374948? + 0.951056516295154?*I

\[ \text{sage: } CLF_{\text{from } L} = L\text{.coerce_embedding()}; CLF_{\text{from } L} \]

Generic morphism:
From: Number Field in \( z_0 \) with defining polynomial \( x^2 - 5 \)
with \( z_0 = 2.236067977499790? \)
To: Complex Lazy Field
Defn: \( z_0 \rightarrow 2.236067977499790? \)

Check transitivity:
\[ \text{sage: } CLF_{\text{from } L}(L\text{.gen()}) \]
\[ 2.236067977499790? \]
\[ \text{sage: } CLF_{\text{from } K}(K\text{.from } L(L\text{.gen()})) \]
\[ 2.23606797749979? + 0.?e-14*I \]

If \( \text{self} \) has no specified embedding, then \( K \) comes with an embedding in \( \text{self} \):

\[ \text{sage: } K.<a> = NumberField(x^6 - 6*x^4 + 8*x^2 - 1) \]
\[ \text{sage: } L.<b>, from } L = K\text{.subfield(a^2)} \]
\[ \text{sage: } L \]
Number Field in \( b \) with defining polynomial \( x^3 - 6*x^2 + 8*x - 1 \) with \( b = a^2 \)
\[ \text{sage: } L\text{.gen}\_embedding() \]
\[ a^2 \]

(continues on next page)
You can also view a number field as having a different generator by just choosing the input to generate the whole field; for that it is better to use \texttt{change_generator()}, which gives isomorphisms in both directions.

\textbf{subfield}\textunderscore\textit{from}\textunderscore\textit{elements} \texttt{(alpha, name=None, polred=True, threshold=None)}

Return the subfield generated by the elements \texttt{alpha}.

If the generated subfield by the elements \texttt{alpha} is either the rational field or the complete number field, the field returned is respectively \texttt{QQ} or \texttt{self}.

\textbf{INPUT:}

- \texttt{alpha} – list of elements in this number field
- \texttt{name} – a name for the generator of the new number field
- \texttt{polred} – (boolean, default True); whether to optimize the generator of the newly created field
- \texttt{threshold} – (positive number, default None) threshold to be passed to the \texttt{do\_polred} function

\textbf{OUTPUT:} a triple \texttt{(field, beta, hom)} where

- \texttt{field} – a subfield of this number field
- \texttt{beta} – a list of elements of \texttt{field} corresponding to \texttt{alpha}
- \texttt{hom} – inclusion homomorphism from \texttt{field} to \texttt{self}

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(QQ)
sage: poly = x^4 - 4*x^2 + 1
sage: emb = AA.polynomial_root(poly, RIF(0.51, 0.52))
sage: K.<a> = NumberField(poly, embedding=emb)
sage: sqrt2 = -a^3 + 3*a
sage: sqrt3 = -a^2 + 2
sage: assert sqrt2 ** 2 == 2 and sqrt3 ** 2 == 3
sage: L, elts, phi = K.subfield_from_elements([sqrt2, 1 - sqrt2/3])
sage: L
Number Field in a0 with defining polynomial x^2 - 2 with a0 = 1.414213562373095?
sage: elts
[a0, -1/3*a0 + 1]
sage: phi
Ring morphism:
From: Number Field in a0 with defining polynomial x^2 - 2 with a0 = 1.414213562373095?
To:   Number Field in a with defining polynomial x^4 - 4*x^2 + 1 with a = 0.5176380902050415?
Defn: a0 |--> -a^3 + 3*a
sage: assert phi(elts[0]) == sqrt2
sage: assert phi(elts[1]) == 1 - sqrt2/3
\end{verbatim}
sage: L, elts, phi = K.subfield_from_elements([1, sqrt3])
sage: assert phi(elts[0]) == 1
sage: assert phi(elts[1]) == sqrt3

sage: L, elts, phi = K.subfield_from_elements([sqrt2, sqrt3])
sage: phi
Identity endomorphism of Number Field in a with defining polynomial
x^4 - 4*x^2 + 1 with a = 0.5176380902050415?

>>> from sage.all import *
>>> x = polygen(QQ)
>>> poly = x**Integer(4) - Integer(4)*x**Integer(2) + Integer(1)
>>> emb = AA.polynomial_root(poly, RIF(RealNumber('0.51'), RealNumber('0.52')))
>>> K = NumberField(poly, embedding=emb, names=('a',)); (a,) = K._first_ngens(1)
>>> sqrt2 = -a**Integer(3) + Integer(3)*a
>>> sqrt3 = -a**Integer(2) + Integer(2)

>>> assert sqrt2 ** Integer(2) == Integer(2)
and sqrt3 ** Integer(2) == Integer(3)

>>> L, elts, phi = K.subfield_from_elements([sqrt2, Integer(1) - sqrt2/Integer(3)])
>>> L
Number Field in a0 with defining polynomial x^2 - 2 with a0 = 1.414213562373095?
>>> elts
[a0, -1/3*a0 + 1]
>>> phi
Ring morphism:
    From: Number Field in a0 with defining polynomial x^2 - 2 with a0 = 1.414213562373095?
    To:   Number Field in a with defining polynomial x^4 - 4*x^2 + 1 with a = 0.5176380902050415?
    Defn: a0 |--> -a^3 + 3*a

>>> assert phi(elts[Integer(0)]) == sqrt2
>>> assert phi(elts[Integer(1)]) == Integer(1) - sqrt2/Integer(3)

trace_dual_basis(b)
Compute the dual basis of a basis of self with respect to the trace pairing.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^3 + x + 1)
sage: b = [1, 2*a, 3*a^2]
sage: T = K.trace_dual_basis(b); T
[4/31*a^2 - 6/31*a + 13/31, -9/62*a^2 - 1/31*a - 3/31, 2/31*a^2 - 3/31*a + 4/93]
sage: [(b[i]*T[j]).trace() for i in range(3) for j in range(3)]
[1, 0, 0, 0, 1, 0, 0, 0, 1]

>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> K = NumberField(x*x^Integer(3) + x + Integer(1), names=('a',)); (a,) = K._first_ngens(1)

>>> b = [Integer(1), Integer(2)*a, Integer(3)*a**Integer(2)]

>>> T = K.trace_dual_basis(b); T
[4/31*a^2 - 6/31*a + 13/31, -9/62*a^2 - 1/31*a - 3/31, 2/31*a^2 - 3/31*a + 4/93]

>>> [(b[i]*T[j]).trace() for i in range(Integer(3)) for j in range(Integer(3))]
[1, 0, 0, 0, 1, 0, 0, 0, 1]

\[ \text{trace_pairing} \]

Return the matrix of the trace pairing on the elements of the list \( v \).

\textbf{EXAMPLES:}

sage: x = polygen(QQ, 'x')

sage: K.<zeta3> = NumberField(x^2 + 3)

sage: K.trace_pairing([1, zeta3])
[  2 0]
[  0 -6]

>>> from sage.all import *

>>> x = polygen(QQ, 'x')

>>> K = NumberField(x*x^Integer(2) + Integer(3), names=('zeta3',)); (zeta3,) = K._first_ngens(1)

>>> K.trace_pairing([Integer(1), zeta3])
[  2 0]
[  0 -6]

\[ \text{uniformizer} \ (P, \text{others='positive'}) \]

Return an element of \( \text{self} \) with valuation 1 at the prime ideal \( P \).

\textbf{INPUT:}

\begin{itemize}
  \item \( \text{self} \) – a number field
  \item \( P \) – a prime ideal of \( \text{self} \)
  \item \text{others} – either "positive" (default), in which case the element will have non-negative valuation at all other primes of \( \text{self} \), or "negative", in which case the element will have non-positive valuation at all other primes of \( \text{self} \).
\end{itemize}

\textbf{Note:} When \( P \) is principal (e.g., always when \text{self} has class number one) the result may or may not be a generator of \( P \! \).

\textbf{EXAMPLES:}
1.1. Number fields

sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 + 5); K
Number Field in a with defining polynomial x^2 + 5

sage: p, q = K.ideal(3).prime_factors()
sage: p
Fractional ideal (3, a + 1)

sage: pi = K.uniformizer(p); pi
a + 1

sage: K.ideal(pi).factor()
(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1))

sage: pi = K.uniformizer(p, 'negative'); pi
1/2*a + 1/2

sage: K.ideal(pi).factor()
(Fractional ideal (2, a + 1))^-1 * (Fractional ideal (3, a + 1))

sage: K = CyclotomicField(9)
sage: Plist = K.ideal(17).prime_factors()
sage: pilist = [K.uniformizer(P) for P in Plist]
sage: [pi.is_integral() for pi in pilist]
[True, True, True]

sage: [pi.valuation(P) for pi, P in zip(pilist, Plist)]
[1, 1, 1]

sage: [ pilist[i] in Plist[i] for i in range(len(Plist)) ]
[True, True, True]

sage: K.<t> = NumberField(x^4 - x^3 - 3*x^2 - x + 1)
sage: [K.uniformizer(P) for P, e in factor(K.ideal(2))]
[2]
sage: [K.uniformizer(P) for P, e in factor(K.ideal(3))]  
[t - 1]  
sage: [K.uniformizer(P) for P, e in factor(K.ideal(5))]  
[t^2 - t + 1, t + 2, t - 2]  
sage: [K.uniformizer(P) for P, e in factor(K.ideal(7))]  
# representation varies, not tested  
[t^2 + 3*t + 1]  
sage: [K.uniformizer(P) for P, e in factor(K.ideal(67))]  
[t + 23, t + 26, t - 32, t - 18]  

>>> from sage.all import *  
>>> K = NumberField(x^4 - x^3 - 3*x^2 - x + 1, names=('a',)); (t,) = K._first_ngens(1)  
>>> [K.uniformizer(P) for P, e in factor(K.ideal(2))]  
[2]  
>>> [K.uniformizer(P) for P, e in factor(K.ideal(3))]  
[t - 1]  
>>> [K.uniformizer(P) for P, e in factor(K.ideal(5))]  
[t^2 - t + 1, t + 2, t - 2]  
>>> [K.uniformizer(P) for P, e in factor(K.ideal(7))]  
# representation varies, not tested  
[t^2 + 3*t + 1]  
>>> [K.uniformizer(P) for P, e in factor(K.ideal(67))]  
[t + 23, t + 26, t - 32, t - 18]  

ALGORITHM:
Use PARI. More precisely, use the second component of pari:idealprimedec in the “positive” case.  
Use pari:idealappr with exponent of −1 and invert the result in the “negative” case.

unit_group (proof=None)
Return the unit group (including torsion) of this number field.

ALGORITHM: Uses PARI’s pari:bnfinit command.

INPUT:
• proof (bool, default True) flag passed to PARI.

Note: The group is cached.

See also:
units() S_unit_group() S_units()
sage: U.invariants()
(10, 0)
sage: [u.multiplicative_order() for u in U.gens()]
[10, +Infinity]

>>> from sage.all import *
>>> x = QQ('x').gen(0)
>>> A = x**Integer(4) - Integer(10)*x**Integer(3) +
˓→Integer(20)*x**Integer(5)*x**Integer(2) - Integer(15)*Integer(5)**Integer(2)*x+
˓→Integer(11)*Integer(5)**Integer(3)
>>> K = NumberField(A, 'a')
>>> U = K.unit_group(); U
Unit group with structure C10 x Z of Number Field in a
with defining polynomial x^4 - 10*x^3 + 100*x^2 - 375*x + 1375
>>> U.gens()
(u0, u1)
>>> U.gens_values()  # random
[-1/275*a^3 + 7/55*a^2 - 6/11*a + 4, 1/275*a^3 + 4/55*a^2 - 5/11*a + 3]
>>> U.invariants()
(10, 0)
>>> [u.multiplicative_order() for u in U.gens()]
[10, +Infinity]

For big number fields, provably computing the unit group can take a very long time. In this case, one can ask
for the conjectural unit group (correct if the Generalized Riemann Hypothesis is true):

sage: K = NumberField(x^17 + 3, 'a')
sage: K.unit_group(proof=True)  # takes forever, not tested
...
sage: U = K.unit_group(proof=False)
sage: U
Unit group with structure C2 x Z x Z x Z x Z x Z x Z x Z x Z of
Number Field in a with defining polynomial x^17 + 3
sage: U.gens()
(u0, u1, u2, u3, u4, u5, u6, u7, u8)
sage: U.gens_values()  # result not independently verified
[-1,
-\(a^9 - a + 1\),
-\(a^16 + a^15 - a^14 + a^12 - a^11 + a^10 + a^8 - a^7 + 2*a^6 - a^4 + 3*a^3 -
˓→2*a^2 + 2*a - 1\),
2*a^16 - a^14 - a^13 + 3*a^12 - 2*a^10 + a^9 + 3*a^8 - 3*a^6 + 3*a^5 + 3*a^4-
˓→2*a^3 - 2*a^2 + 3*a + 4, a^15 + a^14 + 2*a^11 + a^10 - a^9 + a^8 + 2*a^7 - a^5 + 2*a^3 - a^2 - 3*a +
˓→1, a^16 - a^15 - a^14 - a^12 - a^11 - a^10 - a^9 - a^8 - a^7 - a^6 - a^5 - a^4 - a^3 - a^2 + 2, 
˓→2*a^16 + 3*a^15 - 3*a^14 + 3*a^13 - 3*a^12 + a^11 - a^9 + 3*a^8 - 4*a^7 +
˓→5*a^6 - 5*a^5 + 4*a^4 - 3*a^3 + 2*a^2 + 2*a - 4, a^15 - a^12 + a^10 - a^9 - 2*a^8 + 3*a^7 + a^6 - 3*a^5 + a^4 + 4*a^3 - 3*a^2-
˓→2*a + 2, 2*a^16 + a^15 - a^11 - 3*a^10 - 4*a^9 - 4*a^8 - 4*a^7 - 5*a^6 - 7*a^5 - 8*a^4-
˓→4 - 6*a^3 - 5*a^2 - 6*a - 7]

>>> from sage.all import *
>>> K = NumberField(x**Integer(17) + Integer(3), 'a')
>>> K.unit_group(proof=True)  # takes forever, not tested
(continues on next page)
units \((proof=None)\)

Return generators for the unit group modulo torsion.

**ALGORITHM:** Uses PARI’s `pari:bnfinit` command.

**INPUT:**

- \(proof\) (bool, default `True`) flag passed to PARI.

**Note:** For more functionality see `unit_group()`.

**See also:**

`unit_group()` \(\text{S}_\text{unit_group()}\) \(\text{S}_\text{units}()\)

**EXAMPLES:**

```python
sage: x = polygen(QQ)
sage: A = x**4 - 10*x**3 + 20*5*x**2 - 15*5**2*x + 11*5**3
sage: K = NumberField(A, a')
sage: K.units()
(-1/275*a^3 - 4/55*a^2 + 5/11*a - 3,)
```

```python
from sage.all import *

>>> x = polygen(QQ)

>>> A = x**4 - 10*x^3 + 20*5*x^2 - 15*5^2*x + 11*5^3

>>> K = NumberField(A, a')

>>> K.units()
(-1/275*a^3 - 4/55*a^2 + 5/11*a - 3,)
```
For big number fields, provably computing the unit group can take a very long time. In this case, one can ask for the conjectural unit group (correct if the Generalized Riemann Hypothesis is true):

```
sage: K = NumberField(x^17 + 3, 'a')
sage: K.units(proof=True)  # takes forever, not tested
...
sage: K.units(proof=False)  # result not independently verified
(-a^9 - a + 1,
 -a^16 + a^15 - a^14 + a^12 - a^11 + a^10 + a^8 - a^7 + 2*a^6 - a^4 + 3*a^3 -
   2*a^2 + 2*a - 1,
  2*a^16 - a^14 + 3*a^12 - 2*a^10 + a^9 + 3*a^8 - 3*a^6 + 3*a^5 + 3*a^4 -
   2*a^3 - 2*a^2 + 3*a + 4,
  a^15 + a^14 + 2*a^11 + a^10 - a^9 + a^8 + 2*a^7 - a^5 + 2*a^3 - a^2 - 3*a +
   1,
  -a^16 - a^15 + a^14 - a^13 - a^12 - a^11 + a^10 - a^9 - a^8 - a^7 - a^6 - a^5 -
   a^4 - a^3 - a^2 + 2,
   -2*a^16 + 3*a^15 - 3*a^14 + 3*a^13 - 3*a^12 + a^11 - a^9 + 3*a^8 - 4*a^7 +
   5*a^6 - 6*a^5 + 4*a^4 - 3*a^3 + 2*a^2 + 2*a - 4,
   a^15 - a^12 + a^10 - a^9 - 2*a^8 + 3*a^7 + a^6 - 3*a^5 + a^4 + 4*a^3 - 3*a^2 +
   2*a + 2,
   2*a^16 + a^15 - a^11 - 3*a^10 - 4*a^9 - 4*a^8 - 4*a^7 - 5*a^6 - 7*a^5 - 8*a^4 - 6*a^3 - 5*a^2 - 6*a - 7)
```

```
>>> from sage.all import *
>>> K = NumberField(x**Integer(17) + Integer(3), 'a')
>>> K.units(proof=True)  # takes forever, not tested
...
>>> K.units(proof=False)  # result not independently verified
(-a^9 - a + 1,
 -a^16 + a^15 - a^14 + a^12 - a^11 + a^10 + a^8 - a^7 + 2*a^6 - a^4 + 3*a^3 -
   2*a^2 + 2*a - 1,
  2*a^16 - a^14 + 3*a^12 - 2*a^10 + a^9 + 3*a^8 - 3*a^6 + 3*a^5 + 3*a^4 -
   2*a^3 - 2*a^2 + 3*a + 4,
  a^15 + a^14 + 2*a^11 + a^10 - a^9 + a^8 + 2*a^7 - a^5 + 2*a^3 - a^2 - 3*a +
   1,
  -a^16 - a^15 + a^14 - a^13 - a^12 - a^11 + a^10 - a^9 - a^8 - a^7 - a^6 - a^5 -
   a^4 - a^3 - a^2 + 2,
   -2*a^16 + 3*a^15 - 3*a^14 + 3*a^13 - 3*a^12 + a^11 - a^9 + 3*a^8 - 4*a^7 +
   5*a^6 - 6*a^5 + 4*a^4 - 3*a^3 + 2*a^2 + 2*a - 4,
   a^15 - a^12 + a^10 - a^9 - 2*a^8 + 3*a^7 + a^6 - 3*a^5 + a^4 + 4*a^3 - 3*a^2 +
   2*a + 2,
   2*a^16 + a^15 - a^11 - 3*a^10 - 4*a^9 - 4*a^8 - 4*a^7 - 5*a^6 - 7*a^5 - 8*a^4 - 6*a^3 - 5*a^2 - 6*a - 7)
```

```
valuation (prime)

Return the valuation on this field defined by prime.

INPUT:

• prime – a prime that does not split, a discrete (pseudo-)valuation or a fractional ideal

EXAMPLES:

The valuation can be specified with an integer prime that is completely ramified in \(R\):
```
sage: x = polygen(QQ, 'x')
sage: K.<a> = NumberField(x^2 + 1)
sage: K.valuation(2)  # ...
(continues on next page)
```
It can also be unramified in $R$:

```
sage: K.valuation(3)  # needs sage.rings.padics
3-adic valuation
```

A prime that factors into pairwise distinct factors, results in an error:

```
sage: K.valuation(5)  # needs sage.rings.padics
Traceback (most recent call last):
  ...
ValueError: The valuation Gauss valuation induced by 5-adic valuation does not approximate a unique extension of 5-adic valuation with respect to $x^2 + 1$
```

The valuation can also be selected by giving a valuation on the base ring that extends uniquely:

```
sage: CyclotomicField(5).valuation(ZZ.valuation(5))  # needs sage.rings.padics
5-adic valuation
```

When the extension is not unique, this does not work:

```
sage: K.valuation(ZZ.valuation(5))  # needs sage.rings.padics
Traceback (most recent call last):
  ...
```

(continues on next page)
... Value Error: The valuation Gauss valuation induced by 5-adic valuation does not approximate a unique extension of 5-adic valuation with respect to \( x^2 + 1 \)

```python
>>> from sage.all import *

K = ZZ.valuation(Integer(5))

Traceback (most recent call last):
...
ValueError: The valuation Gauss valuation induced by 5-adic valuation does not approximate a unique extension of 5-adic valuation with respect to \( x^2 + 1 \)
```

For a number field which is of the form \( K[x]/(G) \), you can specify a valuation by providing a discrete pseudo-valuation on \( K[x] \) which sends \( G \) to infinity. This lets us specify which extension of the 5-adic valuation we care about in the above example:

```python
sage: # needs sage.rings.padics
sage: R.<x> = QQ[]

sage: G5 = GaussValuation(R, QQ.valuation(Integer(5)))

sage: v = K.valuation(G5.augmentation(x + 2, infinity))

sage: w = K.valuation(G5.augmentation(x + 1/2, infinity))

sage: v == w
False
```

Note that you get the same valuation, even if you write down the pseudo-valuation differently:

```python
sage: # needs sage.rings.padics

sage: v = G5.augmentation(x + 3, 1)

sage: K.valuation(v)
[ 5-adic valuation, v(x + 3) = 1 ]-adic valuation
```

The valuation `prime` does not need to send the defining polynomial \( G \) to infinity. It is sufficient if it singles out one of the valuations on the number field. This is important if the prime only factors over the completion, i.e., if it is not possible to write down one of the factors within the number field:
>>> from sage.all import *
>>> # needs sage.rings.padics
>>> v = G5.augmentation(x + Integer(3), Integer(1))
>>> K.valuation(v)
[ 5-adic valuation, v(x + 3) = 1 ]-adic valuation

Finally, prime can also be a fractional ideal of a number field if it singles out an extension of a $p$-adic valuation of the base field:

```python
sage: K.valuation(K.fractional_ideal(a + 1))
2-adic valuation
```

See also:

- `Order.valuation()`, `pAdicGeneric.valuation()`

**zeta** ($n=2$, `all=False`)

Return one, or a list of all, primitive $n$-th root of unity in this field.

**INPUT:**

- $n$ – positive integer

- `all` – boolean. If `False` (default), return a primitive $n$-th root of unity in this field, or raise a `ValueError` exception if there are none. If `True`, return a list of all primitive $n$-th roots of unity in this field (possibly empty).

**Note:** To obtain the maximal order of a root of unity in this field, use `number_of_roots_of_unity()`.

**Note:** We do not create the full unit group since that can be expensive, but we do use it if it is already known.

**EXAMPLES:**

```python
sage: x = polygen(QQ, 'x')
sage: K.<z> = NumberField(x^2 + 3)
sage: K.zeta(1)
1
sage: K.zeta(2)
-1
sage: K.zeta(2, all=True)
[-1]
sage: K.zeta(3)
-1/2*z - 1/2
sage: K.zeta(3, all=True)
[-1/2*z - 1/2, 1/2*z - 1/2]
sage: K.zeta(4)
Traceback (most recent call last):
(continues on next page)
...``

```
>> from sage.all import *
```

```
>>> x = polygen(QQ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(3), names=('z',)); (z,) = K._
˓→first_ngens(1)
>>> K.zeta(Integer(1))
1
>>> K.zeta(Integer(2))
-1
>>> K.zeta(Integer(2), all=True)
[-1]
>>> K.zeta(Integer(3))
-1/2*z - 1/2
>>> K.zeta(Integer(3), all=True)
[-1/2*z - 1/2, 1/2*z - 1/2]
>>> K.zeta(Integer(4))
Traceback (most recent call last):
...```

```
ValueError: there are no 4th roots of unity in self
```

```
sage: r.<x> = QQ[]
sage: K.<b> = NumberField(x^2 + 1)
sage: K.zeta(4)
b
sage: K.zeta(4, all=True)
[b, -b]
sage: K.zeta(3)
Traceback (most recent call last):
...```

```
ValueError: there are no 3rd roots of unity in self
```

```
sage: K.<a> = NumberField(1/2*x^2 + 1/6)
sage: K.zeta(3)
-3/2*a - 1/2
```

Number fields defined by non-monic and non-integral polynomials are supported (Issue #252):

```
>>> from sage.all import *
```

```
>>> r = QQ['x']; (x,) = r._first_ngens(1)
>>> K = NumberField(x**Integer(2) + Integer(1), names=('b',)); (b,) = K._
˓→first_ngens(1)
>>> K.zeta(Integer(4))
b
>>> K.zeta(Integer(4), all=True)
[b, -b]
>>> K.zeta(Integer(3))
Traceback (most recent call last):
...```

```
ValueError: there are no 3rd roots of unity in self
```

```
>>> K.zeta(Integer(3), all=True)
[]
```

```
```
>>> from sage.all import *
>>> K = NumberField(Integer(1)/Integer(2)*x**Integer(2) + Integer(1) /
-Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> K.zeta(Integer(3))
-3/2*a - 1/2

zeta_coefficients(n)

Compute the first $n$ coefficients of the Dedekind zeta function of this field as a Dirichlet series.

EXAMPLES:

sage: x = QQ['x'].0
sage: NumberField(x^2 + 1, 'a').zeta_coefficients(10)
[1, 1, 0, 1, 2, 0, 0, 1, 1, 2]

>>> from sage.all import *
>>> x = QQ['x'].gen(0)
>>> NumberField(x**Integer(2) + Integer(1), 'a').zeta_coefficients(Integer(10))
[1, 1, 0, 1, 2, 0, 0, 1, 1, 2]

zeta_order()

Return the number of roots of unity in this field.

Note: We do not create the full unit group since that can be expensive, but we do use it if it is already known.

EXAMPLES:

sage: x = polygen(QQ, 'x')
sage: F.<alpha> = NumberField(x^22 + 3)
sage: F.zeta_order()
6
sage: F.<alpha> = NumberField(x^2 - 7)
sage: F.zeta_order()
2

>>> from sage.all import *
>>> x = polygen(QQ, 'x')
>>> F = NumberField(x**Integer(22) + Integer(3), names=('alpha',)); (alpha,) = F._first_ngens(1)
>>> F.zeta_order()
6
>>> F = NumberField(x**Integer(2) - Integer(7), names=('alpha',)); (alpha,) = F._first_ngens(1)
>>> F.zeta_order()
2

sage.rings.number_field.number_field.NumberField_generic_v1 (poly, name, latex_name,
canonical_embedding=None)

Used for unpickling old pickles.

EXAMPLES:
```python
sage: from sage.rings.number_field.number_field import NumberField_absolute_v1
sage: R.<x> = QQ[]
sage: NumberField_absolute_v1(x^2 + 1, 'i', 'i')
Number Field in i with defining polynomial x^2 + 1
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.number_field import NumberField_absolute_v1
>>> R = QQ[x]; (x,) = R._first_ngens(1)
>>> NumberField_absolute_v1(x**Integer(2) + Integer(1), 'i', 'i')
Number Field in i with defining polynomial x^2 + 1
```

class sage.rings.number_field.number_field.NumberField_quadratic(polynomial, name=None, latex_name=None, check=True, embedding=None, assume_disc_small=False, maximize_at_primes=None, structure=None)

Bases: NumberField_absolute, NumberField_quadratic

Create a quadratic extension of the rational field.

The command QuadraticField(a) creates the field \( \mathbb{Q}(\sqrt{a}) \).

EXAMPLES:

```python
sage: QuadraticField(3, 'a')
Number Field in a with defining polynomial x^2 - 3 with a = 1.732050807568878?
```

```python
sage: QuadraticField(-4, 'b')
Number Field in b with defining polynomial x^2 + 4 with b = 2*I
```

```python
>>> from sage.all import *
>>> QuadraticField(Integer(3), 'a')
Number Field in a with defining polynomial x^2 - 3 with a = 1.732050807568878?
```

```python
>>> QuadraticField(-Integer(4), 'b')
Number Field in b with defining polynomial x^2 + 4 with b = 2*I
```

class_number (proof=None)

Return the size of the class group of self.

INPUT:

- proof – boolean (default: True, unless you called proof.number_field() and set it otherwise). If proof is False (not the default!), and the discriminant of the field is negative, then the following warning from the PARI manual applies:

**Warning:** For \( D < 0 \), this function may give incorrect results when the class group has a low exponent (has many cyclic factors), because implementing Shank’s method in full generality slows it down immensely.

EXAMPLES:
These are all the primes so that the class number of \( \mathbb{Q}(\sqrt{-p}) \) is 1:

\[
\text{sage: } [d \text{ for } d \text{ in } \text{prime_range}(2, 300) \\
\text{......: } \text{if not is_square(d) and QuadraticField(-d, 'a').class_number() == 1}]
\]

\[
[2, 3, 7, 11, 19, 43, 67, 163]
\]

It is an open problem to prove that there are infinity many positive square-free \( d \) such that \( \mathbb{Q}(\sqrt{d}) \) has class number 1:

\[
\text{sage: } \text{len([d \text{ for } d \text{ in } \text{range}(2, 200) \\
\text{......: } \text{if not is_square(d) and QuadraticField(d, 'a').class_number() == 1}])}
\]

\[
121
\]

\textbf{discriminant (v=None)}

Return the discriminant of the ring of integers of the number field, or if \( v \) is specified, the determinant of the trace pairing on the elements of the list \( v \).

\textbf{INPUT:}
- \( v \) – (optional) list of element of this number field

\textbf{OUTPUT:} Integer if \( v \) is omitted, and Rational otherwise.

\textbf{EXAMPLES:}

\[
\text{sage: } x = \text{polygen(QQ, 'x')}
\]

\[
\text{sage: } K.<i> = \text{NumberField}(x^2 + 1)
\]

\[
\text{sage: } K.\text{discriminant()}
\]

\-
4

\[
\text{sage: } K.<a> = \text{NumberField}(x^2 + 5)
\]

\[
\text{sage: } K.\text{discriminant()}
\]

\-
20

\[
\text{sage: } K.<a> = \text{NumberField}(x^2 - 5)
\]

\[
\text{sage: } K.\text{discriminant()}
\]

5
hilbert_class_field(names)

Return the Hilbert class field of this quadratic field as a relative extension of this field.

Note: For the polynomial that defines this field as a relative extension, see the method hilbert_class_field_defining_polynomial(), which is vastly faster than this method, since it doesn’t construct a relative extension.

EXAMPLES:

```sage
x = polygen(QQ, 'x')
K.<a> = NumberField(x^2 + 23)
L = K.hilbert_class_field('b'); L
Number Field in b with defining polynomial x^3 - x^2 + 1 over its base field
```

hilbert_class_field_defining_polynomial(name='x')

Return a polynomial over \( \mathbb{Q} \) whose roots generate the Hilbert class field of this quadratic field as an extension of this quadratic field.

Note: Computed using PARI via Schertz’s method. This implementation is quite fast.

EXAMPLES:
sage: K.<b> = QuadraticField(-23)
sage: K.hilbert_class_field_defining_polynomial()
x^3 - x^2 + 1

>>> from sage.all import *
>>> K = QuadraticField(-Integer(23), names=('b',)); (b,) = K._first_ngens(1)
>>> K.hilbert_class_field_defining_polynomial()
x^3 - x^2 + 1

Note that this polynomial is not the actual Hilbert class polynomial: see hilbert_class_polynomial:

sage: K.hilbert_class_polynomial()
#...
˓→needs sage.schemes
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375

>>> from sage.all import *
>>> K = QuadraticField(-Integer(431), names=('a',)); (a,) = K._first_ngens(1)
>>> K.class_number()
21

ComputetheHilbertclasspolynomialofthisquadraticfield.
Rightnow,thisisonlyimplementedforimaginaryquadraticfields.

EXAMPLES:

sage: K.<a> = QuadraticField(-3)
sage: K.hilbert_class_polynomial() ˓→needs sage.schemes
x

sage: K.<a> = QuadraticField(-31)
sage: K.hilbert_class_polynomial(name='z') ˓→needs sage.schemes
z^3 + 39491307*z^2 - 58682638134*z + 1566028350940383

hilbert_class_polynomial(name='x')

Compute the Hilbert class polynomial of this quadratic field.
Right now, this is only implemented for imaginary quadratic fields.
is_galois()  
Return True since all quadratic fields are automatically Galois.

EXAMPLES:

```python
sage: QuadraticField(1234, 'd').is_galois()
True

>>> from sage.all import *

>>> QuadraticField(Integer(1234), 'd').is_galois()
True
```

number_of_roots_of_unity()  
Return the number of roots of unity in this quadratic field.

This is always 2 except when \( d \) is \(-3\) or \(-4\).

EXAMPLES:

```python
sage: QF = QuadraticField
sage: [QF(d).number_of_roots_of_unity() for d in range(-7, -2)]
[2, 2, 2, 4, 6]

>>> from sage.all import *

>>> QF = QuadraticField

>>> [QF(d).number_of_roots_of_unity() for d in range(-Integer(7), -Integer(2))]
[2, 2, 2, 4, 6]
```

order_of_conductor(f)  
Return the unique order with the given conductor in this quadratic field.

See also:

`sage.rings.number_field.order.Order.conductor()`  

EXAMPLES:

```python
sage: K.<t> = QuadraticField(-123)
sage: K.order_of_conductor(1) is K.maximal_order()
True
sage: K.order_of_conductor(2).gens()
(1, t)
sage: K.order_of_conductor(44).gens()
(1, 22*t)
sage: K.order_of_conductor(9001).conductor()
9001
```
sage.rings.number_field.number_field.NumberField_quadratic_v1(poly, name, canonical_embedding=None)

Used for unpicking old pickles.

EXAMPLES:

```python
sage: from sage.rings.number_field.number_field import NumberField_quadratic_v1
sage: NumberField_quadratic_v1(x^2 - 2, 'd')
Number Field in d with defining polynomial x^2 - 2
```

sage.rings.number_field.number_field.QuadraticField(D, name='a', check=True, embedding=True, latex_name='sqrt', **args)

Return a quadratic field obtained by adjoining a square root of $D$ to the rational numbers, where $D$ is not a perfect square.

INPUT:

- $D$ - a rational number
- name - variable name (default: 'a')
- check - bool (default: True)
- embedding - bool or square root of $D$ in an ambient field (default: True)
- latex_name - latex variable name (default: $\sqrt{D}$)

OUTPUT: A number field defined by a quadratic polynomial. Unless otherwise specified, it has an embedding into $\mathbb{R}$ or $\mathbb{C}$ by sending the generator to the positive or upper-half-plane root.

EXAMPLES:

```python
sage: QuadraticField(3, 'a')
Number Field in a with defining polynomial x^2 - 3 with a = 1.732050807568878?
```

(continues on next page)
Traceback (most recent call last):
...
ValueError: D must not be a perfect square.
sage: QuadraticField(9, 'a', check=False)
Number Field in a with defining polynomial x^2 - 9 with a = 3

```python
>>> from sage.all import *

>>> QuadraticField(Integer(3), 'a')
Number Field in a with defining polynomial x^2 - 3 with a = 1.732050807568878?

>>> K = QuadraticField(Integer(3), names=('theta',)); (theta,) = K._first_ngens(1); K
Number Field in theta with defining polynomial x^2 - 3 with theta = 1.732050807568878?

>>> RR(theta)
1.73205080756888

>>> QuadraticField(Integer(9), 'a')
Traceback (most recent call last):
...
ValueError: D must not be a perfect square.

>>> QuadraticField(Integer(9), 'a', check=False)
Number Field in a with defining polynomial x^2 - 9 with a = 3
```

Quadratic number fields derive from general number fields.

```python
sage: from sage.rings.number_field.number_field_base import NumberField
sage: type(K)
<class 'sage.rings.number_field.number_field.NumberField_quadratic_with_category'>
sage: isinstance(K, NumberField)
True
```

Quadratic number fields are cached:

```python
sage: QuadraticField(-11, 'a') is QuadraticField(-11, 'a')
True
```

By default, quadratic fields come with a nice latex representation:

```python
sage: K.<a> = QuadraticField(-7)
sage: latex(K)
\Bold{Q}(\sqrt{-7})
sage: latex(a)
\sqrt{-7}
sage: latex(1/(1+a))
-\frac{1}{8} \sqrt{-7} + \frac{1}{8}
```

(continues on next page)
We can provide our own name as well:

```python
sage: K.<a> = QuadraticField(next_prime(Integer(10)**Integer(10)), latex_name=r'\sqrt{D}'); (a,) = K._first_ngens(1)
sage: 1 + a
a + 1
sage: latex(1 + a)
\sqrt{D} + 1
sage: latex(QuadraticField(-1, a, latex_name=None).gen())
a
```
Note that, in presence of two different names for the generator, the name given by the preparser takes precedence:

```
sage: K4.<y> = QuadraticField(5, 'x'); K4
Number Field in y with defining polynomial x^2 - 5 with y = 2.236067977499790?
sage: K1 == K4
False
```

```
>>> from sage.all import *
>>> K4 = QuadraticField(Integer(5), 'x', names=('y',)); (y,) = K4._first_ngens(1);
˓→ K4
Number Field in y with defining polynomial x^2 - 5 with y = 2.236067977499790?
>>> K1 == K4
False
```

```
sage.rings.number_field.number_field.is_AbsoluteNumberField(x)
Return True if x is an absolute number field.

EXAMPLES:
```
sage: from sage.rings.number_field.number_field import is_AbsoluteNumberField
sage: x = polygen(ZZ, x)
sage: is_AbsoluteNumberField(NumberField(x^2 + 1, 'a'))
doctest:warning...
DeprecationWarning: The function is_AbsoluteNumberField is deprecated; use...
˓→'isinstance(..., NumberField_absolute)' instead.
See https://github.com/sagemath/sage/issues/38124 for details.
True
sage: is_AbsoluteNumberField(NumberField([x^3 + 17, x^2 + 1], 'a'))
False
>>> from sage.all import *
>>> from sage.rings.number_field.number_field import is_AbsoluteNumberField
>>>
is_AbsoluteNumberField(NumberField(x**Integer(2) + Integer(1), 'a'))
doctest:warning...
DeprecationWarning: The function is_AbsoluteNumberField is deprecated; use...
˓→'isinstance(..., NumberField_absolute)' instead.
See https://github.com/sagemath/sage/issues/38124 for details.
True
```

```
The rationals are a number field, but they're not of the absolute number field class.
```
sage: is_AbsoluteNumberField(QQ)
False
```

```
>>> from sage.all import *
>>> is_AbsoluteNumberField(QQ)
False
```

```
sage.rings.number_field.number_field.is_NumberFieldHomsetCodomain(codomain)
Return whether codomain is a valid codomain for a number field homset. This is used by NumberField._Hom_ to determine whether the created homsets should be a sage.rings.number_field.homset.NumberFieldHomset.

EXAMPLES:
```
This currently accepts any parent (CC, RR, ...) in \texttt{Fields}:

```python
sage: from sage.rings.number_field.number_field import is_NumberFieldHomsetCodomain
    # NumberFieldHomsetCodomain
sage: is_NumberFieldHomsetCodomain(QQ)
True
sage: x = polygen(ZZ, 'x')
sage: is_NumberFieldHomsetCodomain(NumberField(x^2 + 1, 'x'))
True
sage: is_NumberFieldHomsetCodomain(ZZ)
False
sage: is_NumberFieldHomsetCodomain(3)
False
sage: is_NumberFieldHomsetCodomain(MatrixSpace(QQ, 2))
False
sage: is_NumberFieldHomsetCodomain(InfinityRing)
False
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.number_field import is_NumberFieldHomsetCodomain
>>> is_NumberFieldHomsetCodomain(QQ)
True
>>> x = polygen(ZZ, 'x')
>>> is_NumberFieldHomsetCodomain(NumberField(x**Integer(2) + Integer(1), 'x'))
True
>>> is_NumberFieldHomsetCodomain(ZZ)
False
>>> is_NumberFieldHomsetCodomain(Integer(3))
False
>>> is_NumberFieldHomsetCodomain(MatrixSpace(QQ, Integer(2)))
False
>>> is_NumberFieldHomsetCodomain(InfinityRing)
False
```

Question: should, for example, QQ-algebras be accepted as well?

Caveat: Gap objects are not (yet) in \texttt{Fields}, and therefore not accepted as number field homset codomains:

```python
sage: is_NumberFieldHomsetCodomain(gap.Rationals)  # needs sage.libs.gap
False
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.number_field import is_NumberFieldHomsetCodomain
>>> is_NumberFieldHomsetCodomain(gap.Rationals)  # needs sage.libs.gap
False
```

\texttt{sage.rings.number_field.number_field.is\_fundamental\_discriminant}(D)

Return \texttt{True} if the integer \( D \) is a fundamental discriminant, i.e., if \( D \not\equiv 0, 1 \pmod{4} \), and \( D \neq 0, 1 \) and either (1) \( D \) is square free or (2) we have \( D \equiv 0 \pmod{4} \) with \( D/4 \equiv 2, 3 \pmod{4} \) and \( D/4 \) square free. These are exactly the discriminants of quadratic fields.

\textbf{EXAMPLES:}

```python
sage: [D for D in range(-15,15) if is_fundamental_discriminant(D)]
...
DeprecationWarning: is_fundamental_discriminant(D) is deprecated;
```

(continues on next page)
please use \( D \text{.is_fundamental_discriminant() } \)
...
[-15, -11, -8, -7, -4, -3, 5, 8, 12, 13]
sage: [D \text{ for } D \text{ in range}(-15, 15)]
....: \text{if not is_square(D) and QuadraticField(D, \( a \)).disc() == D]}
[-15, -11, -8, -7, -4, -3, 5, 8, 12, 13]

```python
from sage.all import *

[D \text{ for } D \text{ in range}(-\text{Integer}(15), \text{Integer}(15)) \text{ if is_fundamental_discriminant(D)}]

DeprecationWarning: is_fundamental_discriminant(D) is deprecated; please use D.is_fundamental_discriminant()
...
[-15, -11, -8, -7, -4, -3, 5, 8, 12, 13]
```

sage.rings.number_field.number_field.is_real_place(v)

Return \( True \) if \( \nu \) is real, \( False \) if \( \nu \) is complex

INPUT:

- \( \nu \) – an infinite place of self

OUTPUT:

A boolean indicating whether a place is real (\( True \)) or complex (\( False \)).

EXAMPLES:

```python
sage: x = polygen(QQ, 'x')
sage: K.<xi> = NumberField(x^3 - 3)
sage: phi_real = K.places()[0]
sage: phi_complex = K.places()[1]
sage: v_fin = tuple(K.primes_above(3))[0]

sage: is_real_place(phi_real)
True
sage: is_real_place(phi_complex)
False
```

```python
from sage.all import *

x = polygen(QQ, 'x')
K = NumberField(x**Integer(3) - Integer(3), names=('xi',)); (xi,) = K._first_ngens(1)
phi_real = K.places()[Integer(0)]
phi_complex = K.places()[Integer(1)]
v_fin = tuple(K.primes_above(Integer(3)))[Integer(0)]

is_real_place(phi_real)
True
is_real_place(phi_complex)
False
```

It is an error to put in a finite place
sage: is_real_place(v_fin)
Traceback (most recent call last):
  ... 
AttributeError: 'NumberFieldFractionalIdeal' object has no attribute 'im_gens'...

```python
>>> from sage.all import *
>>> is_real_place(v_fin)
Traceback (most recent call last):
  ... 
AttributeError: 'NumberFieldFractionalIdeal' object has no attribute 'im_gens'...
```

`sage.rings.number_field.number_field.proof_flag(t)`

Used for easily determining the correct proof flag to use.

Return `t` if `t` is not `None`, otherwise return the system-wide proof-flag for number fields (default: `True`).

**EXAMPLES:**

```python
sage: from sage.rings.number_field.number_field import proof_flag
sage: proof_flag(True)
True
sage: proof_flag(False)
False
sage: proof_flag(None)
True
sage: proof_flag("banana")
'banana'
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.number_field import proof_flag
>>> proof_flag(True)
True
>>> proof_flag(False)
False
>>> proof_flag(None)
True
>>> proof_flag("banana")
'banana'
```

`sage.rings.number_field.number_field.put_natural_embedding_first(v)`

Helper function for embeddings() functions for number fields.

**INPUT:**

- `v` – a list of embeddings of a number field

**OUTPUT:** `None`. The list is altered in-place, so that, if possible, the first embedding has been switched with one of the others, so that if there is an embedding which preserves the generator names then it appears first.

**EXAMPLES:**

```python
sage: K.<a> = CyclotomicField(7)
sage: embs = K.embeddings(K)
sage: [e(a) for e in embs]  # random - there is no natural sort order
[a, a^2, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1]
sage: id = [e for e in embs if e(a) == a][0]; id
Ring endomorphism of Cyclotomic Field of order 7 and degree 6
Defn: a |--> a
```

(continues on next page)
sage: permuted_embs = list(embs); permuted_embs.remove(id); permuted_embs.
˓→append(id)
sage: [e(a) for e in permuted_embs]  # random - but natural map is not first
[a^2, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1, a]
sage: permuted_embs[0] != a
True
sage: from sage.rings.number_field.number_field import put_natural_embedding_first
sage: put_natural_embedding_first(permuted_embs)
sage: [e(a) for e in permuted_embs]  # random - but natural map is first
[a, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1, a^2]
sage: permuted_embs[0] == id
True

>>> from sage.all import *
>>> K = CyclotomicField(Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> embs = K.embeddings(K)
>>> [e(a) for e in embs]  # random - there is no natural sort order
[a, a^2, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1]
>>> id = [e for e in embs if e(a) == a][Integer(0)]; id
Ring endomorphism of Cyclotomic Field of order 7 and degree 6
  Defn: a |--> a
>>> permuted_embs = list(embs); permuted_embs.remove(id); permuted_embs.append(id)
>>> [e(a) for e in permuted_embs]  # random - but natural map is not first
[a^2, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1, a]
>>> permuted_embs[Integer(0)] != a
True
>>> from sage.rings.number_field.number_field import put_natural_embedding_first
>>> put_natural_embedding_first(permuted_embs)
>>> [e(a) for e in permuted_embs]  # random - but natural map is first
[a, a^3, a^4, a^5, -a^5 - a^4 - a^3 - a^2 - a - 1, a^2]
>>> permuted_embs[Integer(0)] == id
True

sage.rings.number_field.number_field.refine_embedding(e, prec=None)
Given an embedding from a number field to either R or C, return an equivalent embedding with higher precision.

INPUT:

• e – an embedding of a number field into either R or C (with some precision)

• prec – (default None) the desired precision; if None, current precision is doubled; if Infinity, the equivalent embedding into either QQbar or AA is returned.

EXAMPLES:

sage: from sage.rings.number_field.number_field import refine_embedding
sage: K = CyclotomicField(3)
sage: e10 = K.complex_embedding(10)
sage: e10.codomain().precision()
10
sage: e25 = refine_embedding(e10, prec=25)
sage: e25.codomain().precision()
25

>>> from sage.all import *
>>> from sage.rings.number_field.number_field import refine_embedding
(continues on next page)
An example where we extend a real embedding into $\mathbb{A}$:

```
 sage: x = polygen(QQ, 'x')
 sage: K.<a> = NumberField(x^3 - 2)
 sage: K.signature()
 (1, 1)
 sage: e = K.embeddings(RR)[0]; e
 Ring morphism:
 From: Number Field in a with defining polynomial x^3 - 2
 To:   Real Field with 53 bits of precision
 Defn: a |--> 1.25992104989487
 sage: e = refine_embedding(e, Infinity); e
 Ring morphism:
 From: Number Field in a with defining polynomial x^3 - 2
 To:   Algebraic Real Field
 Defn: a |--> 1.259921049894873?
```

Now we can obtain arbitrary precision values with no trouble:

```
 sage: RealField(150)(e(a))
 1.2599210498948731647672106072782283505702515
 sage: _**3
 2.0000000000000000000000000000000000000000000
 sage: RealField(200)(e(a^2 - 3*a + 7))
 4.8076379022835799804500738174376232086807389337953290695624
```

(continues on next page)
Complex embeddings can be extended into \texttt{QQbar}:

\begin{verbatim}
>>> RealField(Integer(200))(e(a**Integer(2) - Integer(3)*a + Integer(7)))
4.8076379022835799845007738174376232086807389337953290695624

4.8076379022835799804500738174376232086807389337953290695624

4.8076379022835799804500738174376232086807389337953290695624

4.8076379022835799804500738174376232086807389337953290695624

sage: e = K.embeddings(CC)[0]; e
Ring morphism:
From: Number Field in a with defining polynomial \(x^3 - 2\)
To: Complex Field with 53 bits of precision
Defn: a |--> -0.62996052494743... - 1.09112363597172*I
sage: e = refine_embedding(e,Infinity); e
Ring morphism:
From: Number Field in a with defining polynomial \(x^3 - 2\)
To: Algebraic Field
Defn: a |--> -0.62996052494743657... - 1.091123635971722*I

sage: ComplexField(200)(e(a))
-0.62996052494743658238360530363911417528512573235075399004099
- 1.091123635971721403560072614189808881325873338740309407036*I
sage: e(a)**Integer(3)
2

sage: e = K.embeddings(CC)[Integer(0)]; e
Ring morphism:
From: Number Field in a with defining polynomial \(x^3 - 2\)
To: Complex Field with 53 bits of precision
Defn: a |--> -0.62996052494743... - 1.091123635971722*I
sage: e = refine_embedding(e,Infinity); e
Ring morphism:
From: Number Field in a with defining polynomial \(x^3 - 2\)
To: Algebraic Field
Defn: a |--> -0.62996052494743657... - 1.091123635971722*I

sage: ComplexField(Integer(200))(e(a))
-0.62996052494743658238360530363911417528512573235075399004099
- 1.091123635971721403560072614189808881325873338740309407036*I
sage: e(a)**Integer(3)
2

Embeddings into lazy fields work:

\begin{verbatim}
sage: L = CyclotomicField(7)
sage: x = L.specified_complex_embedding(); x
Generic morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To: Complex Lazy Field
Defn: \(\zeta_7\) |--> 0.6234898018587347 + 0.7818314824680307*I
sage: refine_embedding(x, 300)
Ring morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To: Complex Field with 300 bits of precision
Defn: \(\zeta_7\) |--> 0.62348980185873350525004884004239810632274730896402105365549439096853652456487286759425076
+ 0.781831482468029807844526674057575023233451870868752898063495804509173163393641700868007*I
sage: refine_embedding(x, infinity)

(continues on next page)
\end{verbatim}
From: Cyclotomic Field of order 7 and degree 6
To: Algebraic Field
Defn: zeta7 |--> 0.6234898018587335? + 0.7818314824680299?*I

```
>>> from sage.all import *
>>> L = CyclotomicField(Integer(7))
>>> x = L.specified_complex_embedding(); x
Generic morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To: Complex Lazy Field
  Defn: zeta7 |--> 0.623489801858734? + 0.781831482468030?*I

>>> refine_embedding(x, Integer(300))
Ring morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To: Complex Field with 300 bits of precision
  Defn: zeta7 |--> 0.
    6234898018587335305250048840042398106323274730896402105365549439096853652456487286375942507
    + 0.
    7813148246802980870844452667405775023233451870868752898063495804509173163393644700868007*I

>>> refine_embedding(x, infinity)
Ring morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To: Algebraic Field
  Defn: zeta7 |--> 0.6234898018587335? + 0.7818314824680299?*I
```

When the old embedding is into the real lazy field, then only real embeddings should be considered. See Issue #17495:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3 + x - 1, embedding=0.68, \n    names=(a,)); (a,) = K._first_ngens(1)
sage: from sage.rings.number_field.number_field import refine_embedding

sage: refine_embedding(K.specified_complex_embedding(), 100)
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + x - 1 with a = 0.
  To: Real Field with 100 bits of precision
  Defn: a |--> 0.68232780382801932736948373971

sage: refine_embedding(K.specified_complex_embedding(), Infinity)
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + x - 1 with a = 0.
  To: Algebraic Real Field
  Defn: a |--> 0.68232780382801932736948373971
```

```
>>> from sage.all import *

>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(3) + x - Integer(1), embedding=RealNumber('0.68'),
    \n    names=('a',)); (a,) = K._first_ngens(1)

>>> from sage.rings.number_field.number_field import refine_embedding

>>> refine_embedding(K.specified_complex_embedding(), Integer(100))
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + x - 1 with a = 0.
  To: Real Field with 100 bits of precision
  Defn: a |--> 0.68232780382801932736948373971

>>> refine_embedding(K.specified_complex_embedding(), Infinity)
```

(continues on next page)
1.2 Base class of number fields

AUTHORS:

• William Stein (2007-09-04): initial version

class sage.rings.number_field.number_field_base.NumberField
Bases: Field
Base class for all number fields.

OK (*args, **kwds)
   Synonym for maximal_order().

EXAMPLES:

```sage
x = polygen(ZZ)
sage: NumberField(x^3 - 2,'a').OK()
Maximal Order generated by a in Number Field in a with defining polynomial x^3 - 2
```

```sage
>>> from sage.all import *

sage: K = QQ[sqrt(5)]

sage: K.minkowski_bound()
1/2*sqrt(5)
```

bach_bound()

Return the Bach bound associated to this number field.

Assuming the General Riemann Hypothesis, this is a bound $B$ so that every integral ideal is equivalent modulo principal fractional ideals to an integral ideal of norm at most $B$.

See also:

minkowski_bound()

OUTPUT:

symbolic expression or the Integer 1

EXAMPLES:

We compute both the Minkowski and Bach bounds for a quadratic field, where the Minkowski bound is much better:

```sage
# needs sage.symbolic
sage: K = QQ[sqrt(5)]
sage: K.minkowski_bound()
1/2*sqrt(5)
sage: K.minkowski_bound().n()
```

(continues on next page)
We compute both the Minkowski and Bach bounds for a bigger degree field, where the Bach bound is much better:

```python
sage: # needs sage.symbolic
sage: K = CyclotomicField(Integer(37))
sage: K.minkowski_bound().n() 7.50857335698544e14
sage: K.bach_bound().n() 191669.304126267
```

The bound of course also works for the rational numbers:

```python
sage: QQ.bach_bound() 1
```

```python
>>> from sage.all import *
```
Discriminant()

Return the discriminant of this number field.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: NumberField(x^3 + 9, 'a').discriminant()
-243
```

Is absolute()

Return True if self is viewed as a single extension over \( \mathbb{Q} \).

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^3 + 2)
sage: K.is_absolute()
True
sage: y = polygen(K)
sage: L.<b> = NumberField(y^2 + 1)
sage: L.is_absolute()
False
sage: QQ.is_absolute()
True
```

Maximal order()

Return the maximal order, i.e., the ring of integers of this number field.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: NumberField(x^3 - 2, 'b').maximal_order()
(continues on next page)
```
Maximal Order generated by $b$ in Number Field in $b$ with defining polynomial $x^3 - 2$

```python
>>> from sage.all import *

>>> x = polygen(ZZ)

>>> NumberField(x**Integer(3) - Integer(2), 'b').maximal_order()
Maximal Order generated by $b$ in Number Field in $b$ with defining polynomial $x^3 - 2$
```

**minkowski_bound()**

Return the Minkowski bound associated to this number field.

This is a bound $B$ so that every integral ideal is equivalent modulo principal fractional ideals to an integral ideal of norm at most $B$.

**See also:**

**bach_bound()**

**OUTPUT:**

symbolic expression or Rational

**EXAMPLES:**

The Minkowski bound for $\mathbb{Q}[i]$ tells us that the class number is 1:

```python
sage: # needs sage.symbolic
sage: K = QQ[I]

sage: B = K.minkowski_bound(); B
4/pi

sage: B.n()
1.27323954473516
```

We compute the Minkowski bound for $\mathbb{Q}[\sqrt[3]{2}]$:

```python
sage: # needs sage.symbolic
sage: K = QQ[2^(1/3)]

sage: B = K.minkowski_bound(); B
16/3*sqrt(3)/pi

sage: B.n()
2.94042077558289

sage: int(B)
2
```
We compute the Minkowski bound for $\mathbb{Q}[\sqrt{10}]$, which has class number 2:

```python
sage: # needs sage.symbolic
sage: K = QQ[sqrt(10)]
sage: B = K.minkowski_bound(); B
sqrt(10)
sage: int(B)
3
sage: K.class_number()
2
```

We compute the Minkowski bound for $\mathbb{Q}[\sqrt{2} + \sqrt{3}]$:

```python
sage: # needs sage.symbolic
sage: x = polygen(ZZ)
sage: K.<y,z> = NumberField([x**2 - 2, x**2 - 3])
sage: L.<w> = QQ[sqrt(2) + sqrt(3)]
sage: B = K.minkowski_bound(); B
9/2
sage: int(B)
4
sage: B == L.minkowski_bound()
True
sage: K.class_number()
1
```
The bound of course also works for the rational numbers:

```
sage: QQ.minkowski_bound()
sage: QQ.minkowski_bound()
1
```

```
>>> from sage.all import*
>>> QQ.minkowski_bound()
1
```

**ring_of_integers(**`*args`, **`*kwds`)**

Synonym for `maximal_order()`.

**EXAMPLES:**

```
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^2 + 1)
sage: K.ring_of_integers()
Gaussian Integers generated by a in Number Field in a with defining polynomial x^2 + 1
```

```
>>> from sage.all import*
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> K.ring_of_integers()
Gaussian Integers generated by a in Number Field in a with defining polynomial x^2 + 1
```

**signature()**

Return \((r_1, r_2)\), where \(r_1\) and \(r_2\) are the number of real embeddings and pairs of complex embeddings of this field, respectively.

**EXAMPLES:**

```
sage: x = polygen(ZZ)
sage: NumberField(x^3 - 2, 'a').signature()
(1, 1)
```

```
>>> from sage.all import*
>>> x = polygen(ZZ)
>>> NumberField(x**Integer(3) - Integer(2), 'a').signature()
(1, 1)
```

`sage.rings.number_field.number_field_base.is_NumberField(x)`

Return `True` if `x` is of number field type.

This function is deprecated.

**EXAMPLES:**

```
sage: from sage.rings.number_field.number_field_base import is_NumberField
sage: x = polygen(ZZ)
sage: is_NumberField(NumberField(x^2 + 1, 'a'))
doctest:...: DeprecationWarning: the function is_NumberField is deprecated; use isinstance(x, sage.rings.number_field.number_field_base.NumberField) instead
See https://github.com/sagemath/sage/issues/35283 for details.
True
```
sage: is_NumberField(QuadraticField(-97, 'theta'))
True
sage: is_NumberField(CyclotomicField(97))
True

>>> from sage.all import *
>>> from sage.rings.number_field.number_field_base import is_NumberField
>>> x = polygen(ZZ)
>>> is_NumberField(NumberField(x**Integer(2) + Integer(1), 'a'))
doctest:...: DeprecationWarning: the function is_NumberField is deprecated; use isinstance(x, sage.rings.number_field.number_field_base.NumberField) instead
See https://github.com/sagemath/sage/issues/35283 for details.
True
>>> is_NumberField(QuadraticField(-Integer(97), 'theta'))
True
>>> is_NumberField(CyclotomicField(Integer(97)))
True

Note that the rational numbers \( \mathbb{Q} \) are a number field:

sage: is_NumberField(QQ)
True
sage: is_NumberField(ZZ)
False

1.3 Relative number fields

This example constructs a quadratic extension of a quartic number field:

sage: x = polygen(ZZ, 'x')
sage: K.<y> = NumberField(x^4 - 420*x^2 + 40000)
sage: z = y^5/11; z
420/11*y^3 - 40000/11*y
sage: R.<y> = PolynomialRing(K)
sage: f = y^2 + y + 1
sage: L.<a> = K.extension(f); L
Number Field in a with defining polynomial y^2 + y + 1 over its base field
sage: KL.<b> = NumberField([x^4 - 420*x^2 + 40000, x^2 + x + 1]); KL
Number Field in b0 with defining polynomial x^4 - 420*x^2 + 40000 over its base field

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(420)*x**Integer(2) + Integer(40000),˓
    \text{names=('y',))}; (y,) = K._first_ngens(1)
>>> z = y**Integer(5)/Integer(11); z
420/11*y^3 - 40000/11*y
>>> R = PolynomialRing(K, names=('y',)); (y,) = R._first_ngens(1)
We do some arithmetic in a tower of relative number fields:

```python
sage: K.<cuberoot2> = NumberField(x^3 - 2)
sage: L.<cuberoot3> = K.extension(x^3 - 3)
sage: S.<sqrt2> = L.extension(x^2 - 2)
sage: S
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
sage: sqrt2 * cuberoot3
cuberoot3*sqrt2
sage: (sqrt2 + cuberoot3)^5
(20*cuberoot3^2 + 15*cuberoot3 + 4)*sqrt2 + 3*cuberoot3^2 + 20*cuberoot3 + 60
sage: a = S(cuberoot2); a
cuberoot2
sage: a.parent()  # output will be the same as the previous line
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
```

(continues on next page)
AUTHORS:

- Steven Sivek (2006-05-12): added support for relative extensions
- William Stein (2007-09-04): major rewrite and documentation
- Robert Bradshaw (2008-10): specified embeddings into ambient fields
- Nick Alexander (2009-01): modernized coercion implementation
- Robert Harron (2012-08): added is_CM_extension
- Julian Rüth (2014-04): absolute number fields are unique parents

```
>>> a.parent()
Number Field in sqrt2 with defining polynomial x^2 - 2 over its base field
```

```
sage.rings.number_field.number_field_rel.
NumberField_extension_v1 (base_field, poly,
name, latex_name, canonical_embedding=None)
```

Used for unpickling old pickles.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_rel import NumberField_relative_v1
sage: R.<x> = CyclotomicField(3)[]
```

```
sage: NumberField_relative_v1(CyclotomicField(3), x^2 + 7, 'a', 'a')
Number Field in a with defining polynomial x^2 + 7 over its base field
```

```
>>> from sage.all import *
```

```
>>> from sage.rings.number_field.number_field_rel import NumberField_relative_v1
```

```
>>> R = CyclotomicField(Integer(3))['x']; (x,) = R._first_ngens(1)
```

```
>>> NumberField_relative_v1(CyclotomicField(Integer(3)), x**Integer(2) +
˓→Integer(7), 'a', 'a')
Number Field in a with defining polynomial x^2 + 7 over its base field
```

```
class sage.rings.number_field.number_field_rel.NumberField_relative (base,
polynomial, name, latex_name=None, names=None, check=True, embedding=None,
structure=None)
```

Bases: NumberField_generic

INPUT:

- base – the base field
- polynomial – a polynomial which must be defined in the ring \( K[x] \), where \( K \) is the base field.
- name – a string, the variable name
- latex_name – a string or None (default: None), variable name for latex printing
• check – a boolean (default: True), whether to check irreducibility of polynomial
• embedding – currently not supported, must be None
• structure – an instance of structure.NumberFieldStructure or None (default: None), provides additional information about this number field, e.g., the absolute number field from which it was created

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 2)
sage: t = polygen(K)
sage: L.<b> = K.extension(t^2 + t + a); L
Number Field in b with defining polynomial x^2 + x + a over its base field
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> t = polygen(K)
>>> L = K.extension(t**Integer(2) + t + a, names=('b',)); (b,) = L._first_ngens(1); L
Number Field in b with defining polynomial x^2 + x + a over its base field
```

```python
absolute_base_field()

Return the base field of this relative extension, but viewed as an absolute field over \( \mathbb{Q} \).

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b,c> = NumberField([x^2 + 2, x^3 + 3, x^3 + 2])
sage: K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: K.base_field()
Number Field in b with defining polynomial x^3 + 3 over its base field
sage: K.absolute_base_field()[0]
Number Field in a0 with defining polynomial x^9 + 3*x^6 + 165*x^3 + 1
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + Integer(2), x**Integer(3) + Integer(3),
...
                        x**Integer(3) + Integer(2)], names=('a', 'b', 'c',)); (a, b, c,) = K._first_ngens(3)
>>> K
Number Field in a with defining polynomial x^2 + 2 over its base field
>>> K.base_field()
Number Field in b with defining polynomial x^3 + 3 over its base field
>>> K.absolute_base_field()[0]
Number Field in a0 with defining polynomial x^9 + 3*x^6 + 165*x^3 + 1
```

absolute_degree()

The degree of this relative number field over the rational field.

EXAMPLES:
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.absolute_degree()
6

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**Integer(2) - Integer(17), x**Integer(3) -
˓→Integer(2)], names=('a',)); (a,) = K._first_ngens(1)
>>> K.absolute_degree()
6

absolute_different ()
Return the absolute different of this relative number field $L$, as an ideal of $L$. To get the relative different of $L/K$, use relative_different ().

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<b> = K.extension(t^4 - i)
sage: L.absolute_different()
Fractional ideal (8)

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
˓→first_ngens(1)
>>> t = K['t'].gen()
>>> L = K.extension(t**Integer(4) - i, names=('b',)); (b,) = L._first_ngens(1)
>>> L.absolute_different()
Fractional ideal (8)

absolute_discriminant (v=None)
Return the absolute discriminant of this relative number field or if $v$ is specified, the determinant of the trace pairing on the elements of the list $v$.

INPUT:

- $v$ (optional) – list of element of this relative number field.

OUTPUT: Integer if $v$ is omitted, and Rational otherwise.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<b> = K.extension(t^4 - i)
sage: L.absolute_discriminant()
16777216
sage: L.absolute_discriminant([[(b + i)^j for j in range(8)]])
61911970349056

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
(continues on next page)
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> t = K['t'].gen()
>>> L = K.extension(t**Integer(4) - i, names=('b',)); (b,) = L._first_ngens(1)
>>> L.absolute_discriminant()
16777216
>>> L.absolute_discriminant([(b + i)**j for j in range(Integer(8))])
61911970349056

**absolute_field**(names)

Return self as an absolute number field.

**INPUT:**

- names – string; name of generator of the absolute field

**OUTPUT:**

An absolute number field $K$ that is isomorphic to this field.

Also, $K$.structure() returns from$_K$ and to$_K$, where from$_K$ is an isomorphism from $K$ to self and to$_K$ is an isomorphism from self to $K$.

**EXAMPLES:**

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: L.<xyz> = K.absolute_field(); L
Number Field in xyz with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
sage: L.<c> = K.absolute_field(); L
Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
sage: from_L, to_L = L.structure()
sage: from_L
Isomorphism map:
  From: Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
  To: Number Field in a with defining polynomial x^4 + 3 over its base field
sage: from_L(c)
a - b
sage: to_L
Isomorphism map:
  From: Number Field in a with defining polynomial x^4 + 3 over its base field
  To: Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
sage: to_L(a)
-5/182*c^7 - 87/364*c^5 - 185/182*c^3 + 323/364*c
sage: to_L(b)
-5/182*c^7 - 87/364*c^5 - 185/182*c^3 - 41/364*c
sage: to_L(a)^4
-3
sage: to_L(b)^2
-2
```

```bash
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
(continues on next page)
```
K = NumberField([x**Integer(4) + Integer(3), x**Integer(2) + Integer(2)], names=('a', 'b')); (a, b,) = K._first_ngens(2); K

Number Field in a with defining polynomial x^4 + 3 over its base field

L = K.absolute_field(names=('xyz',)); (xyz,) = L._first_ngens(1); L

Number Field in xyz with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49

L = K.absolute_field(names=('c',)); (c,) = L._first_ngens(1); L

Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49

from_L, to_L = L.structure()

from_L

Isomorphism map:
From: Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49
To:   Number Field in a with defining polynomial x^4 + 3 over its base field

to_L

Isomorphism map:
From: Number Field in a with defining polynomial x^4 + 3 over its base field
To:   Number Field in c with defining polynomial x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49

to_L(c)

a - b

to_L(a)
-5/182*c^7 + 87/364*c^5 + 185/182*c^3 + 323/364*c

to_L(b)
-5/182*c^7 + 87/364*c^5 - 185/182*c^3 - 41/364*c

to_L(a)**Integer(4)
-3

to_L(b)**Integer(2)
-2

absolute_generator()

Return the chosen generator over Q for this relative number field.

EXAMPLES:

sage: y = polygen(QQ,'y')
sage: k.<a> = NumberField([y**2 + 2, y**4 + 3])
sage: g = k.absolute_generator(); g
a0 - a1

sage: g.minpoly()
x^2 + 2*a1*x + a1^2 + 2

sage: g.absolute_minpoly()
x^8 + 8*x^6 + 30*x^4 - 40*x^2 + 49

absolute_polynomial()

1.3. Relative number fields 203
Return the polynomial over \( \mathbb{Q} \) that defines this field as an extension of the rational numbers.

**Note:** The absolute polynomial of a relative number field is chosen to be equal to the defining polynomial of the underlying PARI absolute number field (it cannot be specified by the user). In particular, it is always a monic polynomial with integral coefficients. On the other hand, the defining polynomial of an absolute number field and the relative polynomial of a relative number field are in general different from their PARI counterparts.

**EXAMPLES:**

```python
c = polygen(ZZ, 'c')
d = polygen(ZZ, 'd')
k.<a, b> = NumberField([x^2 + 1, x^3 + x + 1]); k
Number Field in a with defining polynomial x^2 + 1 over its base field
k.absolute_polynomial()
x^6 + 5*x^4 - 2*x^3 + 4*x^2 + 4*x + 1
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField([x**Integer(2) + Integer(1), x**Integer(3) + x + Integer(1)], names=('a', 'b',)); (a, b,) = k._first_ngens(2); k
Number Field in a with defining polynomial x^2 + 1 over its base field
>>> k.absolute_polynomial()
x^6 + 5*x^4 - 2*x^3 + 4*x^2 + 4*x + 1
```

An example comparing the various defining polynomials to their PARI counterparts:

```python
c = polygen(ZZ, 'c')
d = polygen(ZZ, 'd')
k.<a, c> = NumberField([x^2 + 1/3, x^2 + 1/4])
k.absolute_polynomial()
x^4 - x^2 + 1
k.pari_polynomial()
x^4 - x^2 + 1
k.base_field().absolute_polynomial()
x^2 + 1/4
k.pari_absolute_base_polynomial()
y^2 + 1
k.relative_polynomial()
x^2 + Mod(y, y^2 + 1)*x - 1
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField([x**Integer(2) + Integer(1)/Integer(3), x**Integer(2) + Integer(1)/Integer(4)], names=('a', 'c',)); (a, c,) = k._first_ngens(2)
>>> k.absolute_polynomial()
x^4 - x^2 + 1
>>> k.pari_polynomial()
x^4 - x^2 + 1
>>> k.base_field().absolute_polynomial()
x^2 + 1/4
>>> k.pari_absolute_base_polynomial()
y^2 + 1
>>> k.relative_polynomial()
x^2 + Mod(y, y^2 + 1)*x - 1
```

(continues on next page)
absolute_polynomial_ntl()

Return defining polynomial of this number field as a pair, an ntl polynomial and a denominator.

This is used mainly to implement some internal arithmetic.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: NumberField(x^2 + (2/3)*x - 9/17, 'a').absolute_polynomial_ntl()
([-27 34 51], 51)
```

absolute_vector_space(base=None, *args, **kwds)

Return vector space over $\mathbb{Q}$ of self and isomorphisms from the vector space to self and in the other direction.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^3 + 3, x^3 + 2]); K
Number Field in a with defining polynomial x^3 + 3 over its base field
sage: V,from_V,to_V = K.absolute_vector_space(); V
Vector space of dimension 9 over Rational Field
sage: from_V
Isomorphism map:
  From: Vector space of dimension 9 over Rational Field
  To:  Number Field in a with defining polynomial x^3 + 3 over its base field
sage: to_V
Isomorphism map:
  From: Number Field in a with defining polynomial x^3 + 3 over its base field
  To:  Vector space of dimension 9 over Rational Field
sage: c = (a+1)^5; c
7*a^2 - 10*a - 29
sage: to_V(c)
(-29, -712/9, 19712/45, 0, -14/9, 364/45, 0, -4/9, 119/45)
```

1.3. Relative number fields
>>> from_V
Isomorphism map:
From: Vector space of dimension 9 over Rational Field
To: Number Field in a with defining polynomial x^3 + 3 over its base field

>>> to_V
Isomorphism map:
From: Number Field in a with defining polynomial x^3 + 3 over its base field
To: Vector space of dimension 9 over Rational Field

>>> c = (a+Integer(1))**Integer(5); c
7*a^2 - 10*a - 29

>>> to_V(c)
(-29, -712/9, 19712/45, 0, -14/9, 364/45, 0, -4/9, 119/45)

>>> from_V(to_V(c))
7*a^2 - 10*a - 29

>>> from_V(Integer(3)*to_V(b))
3*b

automorphisms()

Compute all Galois automorphisms of self over the base field. This is different from computing the embeddings of self into self; there, automorphisms that do not fix the base field are considered.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + 10000, x^2 + x + 50]); K
Number Field in a with defining polynomial x^2 + 10000 over its base field
sage: K.automorphisms()
[Relative number field endomorphism of Number Field in a with defining polynomial x^2 + 10000 over its base field
 Defn: a |--> a
 b |--> b,
 Relative number field endomorphism of Number Field in a with defining polynomial x^2 + 10000 over its base field
 Defn: a |--> -a
 b |--> b]
sage: rho, tau = K.automorphisms()
sage: tau(a)
-a
sage: tau(b) == b
True

sage: L.<b, a> = NumberField([x^2 + x + 50, x^2 + 10000, ]); L
Number Field in b with defining polynomial x^2 + x + 50 over its base field
sage: L.automorphisms()
[Relative number field endomorphism of Number Field in b with defining polynomial x^2 + x + 50 over its base field
 Defn: b |--> b
 a |--> a,
 Relative number field endomorphism of Number Field in b with defining polynomial x^2 + x + 50 over its base field
 Defn: b |--> -b - 1
 a |--> a]
sage: rho, tau = L.automorphisms()
sage: tau(a) == a
True
sage: tau(b)
-b - 1

sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.automorphisms()
[Relative number field endomorphism of Number Field in c
 with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its base field
 Defn: c |--> c
 a |--> a
 b |--> b,
 Relative number field endomorphism of Number Field in c
 with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its base field
 Defn: c |--> -c
 a |--> a
 b |--> b]

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + Integer(10000), x**Integer(2) + x + Integer(50),], names=('a', 'b',)); (a, b,) = K._first_ngens(2); K
Number Field in a with defining polynomial x^2 + 10000 over its base field

>>> K.automorphisms()
[Relative number field endomorphism of Number Field in a
 with defining polynomial x^2 + 10000 over its base field
 Defn: a |--> a
 b |--> b,
 Relative number field endomorphism of Number Field in a
 with defining polynomial x^2 + 10000 over its base field
 Defn: a |--> -a
 b |--> b]

>>> rho, tau = K.automorphisms()
>>> tau(a)
-a
>>> tau(b) == b
True

>>> L = NumberField([x**Integer(2) + x + Integer(50), x**Integer(2) +
 Integer(10000), ], names=('b', 'a',)); (b, a,) = L._first_ngens(2); L
Number Field in b with defining polynomial x^2 + x + 50 over its base field

>>> L.automorphisms()
[Relative number field endomorphism of Number Field in b
 with defining polynomial x^2 + x + 50 over its base field
 Defn: b |--> b
 a |--> a,
 Relative number field endomorphism of Number Field in b
 with defining polynomial x^2 + x + 50 over its base field

(continues on next page)
Defn: \( b \mapsto -b - 1 \)
\( a \mapsto a \)

\[
\begin{align*}
\text{rho}, \tau &= \text{L.automorphisms()} \\
\text{tau}(a) &= a \\
\text{tau}(b) &= -b - 1 \\
\text{PQ} &= \text{QQ[}'X'\text{]}; (X,) = \text{PQ._first_ngens(1)} \\
\text{F} &= \text{NumberField([}'X'^\text{2} - \text{Integer}(2), 'X'^\text{2} - \text{Integer}(3)]\}, \\
\text{PF} &= \text{F['Y']}; (Y,) = \text{PF._first_ngens(1)} \\
\text{K} &= \text{F.extension(}'Y'^\text{2} - (\text{Integer}(1) + a) \cdot (a + b) \cdot a \cdot b\text{, names=('c', -1))}; (c,) = \text{K._first_ngens(1)} \\
\text{K.automorphisms()} &= \\
\text{Relative number field endomorphism of Number Field in c} \\
\text{with defining polynomial } Y^2 + (-2*b - 3)*a - 2*b - 6 \text{ over its base field} \\
\text{Defn: } c \mapsto c \\
\text{a \mapsto a} \\
\text{b \mapsto b} \\
\text{Relative number field endomorphism of Number Field in c} \\
\text{with defining polynomial } Y^2 + (-2*b - 3)*a - 2*b - 6 \text{ over its base field} \\
\text{Defn: } c \mapsto -c \\
\text{a \mapsto a} \\
\text{b \mapsto b} \\
\end{align*}
\]

\textbf{base_field()}

Return the base field of this relative number field.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField([x^3 + x + 1])
sage: R.<z> = k[]
sage: L.<b> = NumberField(z^3 + a)
sage: L.base_field() \\
Number Field in a with defining polynomial x^3 + x + 1 \\
sage: L.base_field() \text{ is } k \\
True
\end{verbatim}

\begin{verbatim}
>>> from sage.all import * 
>>> x = polygen(ZZ, 'x') 
>>> k = NumberField([x**Integer(3) + x + Integer(1)], names=('a',)); (a,) = k._first_ngens(1) 
>>> R = k['z']; (z,) = R._first_ngens(1) 
>>> L = NumberField(z**Integer(3) + a, names=('b',)); (b,) = L._first_ngens(1) 
>>> L.base_field() \\
Number Field in a with defining polynomial x^3 + x + 1 \\
>>> L.base_field() \text{ is } k \\
True
\end{verbatim}

This is very useful because the print representation of a relative field doesn’t describe the base field.:
### base_ring()

This is exactly the same as base_field.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: a = NumberField([x**2 + 1, x**3 + x + 1])
sage: a.base_ring()
Number Field in a1 with defining polynomial x^3 + x + 1
```

### change_names(names)

Return relative number field isomorphic to self but with the given generator names.

**INPUT:**

- names – number of names should be at most the number of generators of self, i.e., the number of steps in the tower of relative fields.

Also, K.structure() returns from_K and to_K, where from_K is an isomorphism from K to self and to_K is an isomorphism from self to K.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x**4 + 3, x**2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: L.<c,d> = K.change_names(); (c, d) = L._first_ngens(2)
Number Field in c with defining polynomial x^4 + 3 over its base field
```
Number Field in c with defining polynomial \( x^4 + 3 \) over its base field

```
Number Field in d with defining polynomial \( x^2 + 2 \)
```

An example with a 3-level tower:

```
sage: K.<a,b,c> = NumberField([x^2 + 17, x^2 + x + 1, x^3 - 2]); K
Number Field in a with defining polynomial \( x^2 + 17 \) over its base field
sage: L.<m,n,r> = K.change_names()
sage: L
Number Field in m with defining polynomial \( x^2 + 17 \) over its base field
sage: L.base_field()
Number Field in n with defining polynomial \( x^2 + x + 1 \) over its base field
sage: L.base_field().base_field()
Number Field in r with defining polynomial \( x^3 - 2 \)
```

And a more complicated example:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b, names=('m', 'n', 'r',)); L
Number Field in m with defining polynomial \( x^2 + (-2*r - 3)*n - 2*r - 6 \) over its base field
sage: L.structure()
(Isomorphism given by variable name change map:
 From: Number Field in m with defining polynomial
 \( x^2 + (-2*r - 3)*n - 2*r - 6 \) over its base field
 To: Number Field in c with defining polynomial
 \( Y^2 + (-2*b - 3)*a - 2*b - 6 \) over its base field,
Isomorphism given by variable name change map:
 From: Number Field in c with defining polynomial
 \( Y^2 + (-2*b - 3)*a - 2*b - 6 \) over its base field
 To: Number Field in m with defining polynomial
 \( x^2 + (-2*r - 3)*n - 2*r - 6 \) over its base field)
```

(continues on next page)
\[ \text{composite_fields}(\text{other}, \text{names} = \text{None}, \text{both_maps} = \text{False}, \text{preserve_embedding} = \text{True}) \]

List of all possible composite number fields formed from \text{self} and \text{other}, together with (optionally) embeddings into the compositum; see the documentation for both \text{maps} below.

Since relative fields do not have ambient embeddings, \text{preserve_embedding} has no effect. In every case all possible composite number fields are returned.

\text{INPUT:}

\begin{itemize}
  \item \text{other} – a number field
  \item \text{names} – generator name for composite fields
  \item \text{both_maps} – (default: False) if True, return quadruples \((F, \text{self_into_F}, \text{other_into_F}, \text{k})\) such that \text{self_into_F} maps \text{self} into \(F\), \text{other_into_F} maps \text{other} into \(F\). For relative number fields, \(k\) is always None.
  \item \text{preserve_embedding} – (default: True) has no effect, but is kept for compatibility with the absolute version of this method. In every case the list of all possible compositums is returned.
\end{itemize}

\text{OUTPUT:}

list of the composite fields, possibly with maps.

\text{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + 5, x^2 - 2])
sage: L.<c, d> = NumberField([x^2 + 5, x^2 - 3])
sage: K.composite_fields(L, e)
[Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600]
sage: K.composite_fields(L, e, both_maps=True)
[[Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600, Relative number field morphism: From: Number Field in a with defining polynomial x^2 + 5 over its base_field
  To: Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600
  Defn: a |--> -9/66560*e^7 + 11/4160*e^5 - 241/4160*e^3 - 101/104*e
  b |--> -21/166400*e^7 + 73/20800*e^5 - 779/10400*e^3 + 7/260*e,

(continues on next page)\end{verbatim}
Relative number field morphism:
From: Number Field in c with defining polynomial $x^2 + 5$ over its base
To: Number Field in e with defining polynomial
$x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600$
Defn: $c |\mapsto -9/66560*e^7 + 11/4160*e^5 - 241/4160*e^3 - 101/104*e$
d $|\mapsto -3/25600*e^7 + 7/1600*e^5 - 147/1600*e^3 + 1/40*e,$
None]

```python
>>> from sage.all import *
``` 
```python
>>> x = polygen(ZZ, 'x')
``` 
```python
>>> K = NumberField([x**Integer(2) + Integer(5), x**Integer(2) - Integer(2)],
˓→names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> L = NumberField([x**Integer(2) + Integer(5), x**Integer(2) - Integer(3)],
˓→names=('c', 'd',)); (c, d,) = L._first_ngens(2)
>>> K.composite_fields(L, 'e')
``` 
```python
[Number Field in e with defining polynomial
$x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600$
]``` 

```python
>>> K.composite_fields(L, 'e', both_maps=True)
``` 
```python
[[Number Field in e with defining polynomial
$x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600,$
Relative number field morphism:
From: Number Field in a with defining polynomial $x^2 + 5$ over its base
To: Number Field in e with defining polynomial
$x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600$
Defn: $a |\mapsto -9/66560*e^7 + 11/4160*e^5 - 241/4160*e^3 - 101/104*e$
b $|\mapsto -21/166400*e^7 + 73/20800*e^5 - 779/10400*e^3 + 7/260*e,$
Relative number field morphism:
From: Number Field in c with defining polynomial $x^2 + 5$ over its base
To: Number Field in e with defining polynomial
$x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600$
Defn: $c |\mapsto -9/66560*e^7 + 11/4160*e^5 - 241/4160*e^3 - 101/104*e$
d $|\mapsto -3/25600*e^7 + 7/1600*e^5 - 147/1600*e^3 + 1/40*e,$
None]]
```

**defining_polynomial()**

Return the defining polynomial of this relative number field.

This is exactly the same as **relative_polynomial()**.

**EXAMPLES:**

```python
sage: C.<z> = CyclotomicField(5)
sage: PC.<X> = C[]
sage: K.<a> = C.extension(X^2 + X + z); K
Number Field in a with defining polynomial X^2 + X + z over its base field
sage: K.defining_polynomial()
X^2 + X + z
``` 

```python
>>> from sage.all import *
``` 
```python
>>> C = CyclotomicField(Integer(5), names=('z',)); (z,) = C._first_ngens(1)
``` 
```python
>>> PC = C['X']; (X,) = PC._first_ngens(1)
``` 
```python
>>> K = C.extension(X**Integer(2) + X + z, names=('a',)); (a,) = K._first_ ˓→ngens(1); K
```
**Number Field in a with defining polynomial X^2 + X + z over its base field**

```python
>>> K.defining_polynomial()
X^2 + X + z
```

**degree()**

The degree, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute degree for the relative degree, or vice versa.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.degree()
Traceback (most recent call last):
  ...  
NotImplementedError: For a relative number field you must use relative_degree or absolute_degree as appropriate
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**2 + x + 1, x**3 + x + 1], names=('a',)); (a,) = K._first_ngens(1)
>>> K.degree()
Traceback (most recent call last):
  ...  
NotImplementedError: For a relative number field you must use relative_degree or absolute_degree as appropriate
```

**different()**

The different, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute different for the relative different, or vice versa.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.different()
Traceback (most recent call last):
  ...  
NotImplementedError: For a relative number field you must use relative_differenve or absolute_differenve as appropriate
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**2 - 17, x**3 - 2], names=('a',)); (a,) = K._first_ngens(1)
>>> K.different()
Traceback (most recent call last):
  ...  
NotImplementedError: For a relative number field you must use relative_differenve or absolute_differenve as appropriate
```

**disc()**

The discriminant, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute discriminant for the relative discriminant, or vice versa.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.discriminant()
Traceback (most recent call last):
  ...  
NotImplementedError: For a relative number field you must use relative_discriminant or absolute_discriminant as appropriate
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**2 + x + 1, x**3 + x + 1], names=('a',)); (a,) = K._first_ngens(1)
>>> K.discriminant()
Traceback (most recent call last):
  ...  
NotImplementedError: For a relative number field you must use relative_discriminant or absolute_discriminant as appropriate
```
The discriminant, unqualified, of a relative number field is deliberately not implemented, so that a user cannot mistake the absolute discriminant for the relative discriminant, or vice versa.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.discriminant()
Traceback (most recent call last):
... NotImplementedError: For a relative number field you must use relative_discriminant or absolute_discriminant as appropriate
```
sage: f = K.embeddings(ComplexField(58)); f
[Relative number field morphism:
  From: Number Field in a with defining polynomial x^3 - 2 over its base field
  To:   Complex Field with 58 bits of precision
  Defn: a |--> -0.62996052494743676 - 1.0911236359717214*I
        b |--> -1.9428902930940239e-16 + 1.0000000000000000*I,
...]
Relative number field morphism:
  From: Number Field in a with defining polynomial x^3 - 2 over its base field
  To:   Complex Field with 58 bits of precision
  Defn: a |--> 1.2599210498948731
        b |--> -0.99999999999999999*I
]
sage: f[0](a)^3
2.0000000000000002 - 8.6389229103644993e-16*I
sage: f[0](b)^2
-1.0000000000000001 - 3.8857805861880480e-16*I
sage: f[0](a+b)
-0.62996052494743693 - 0.091123635971721295*I

```python
>>> from sage.all import *
```

```python
>>> x = polygen(ZZ, 'x')
```

```python
>>> K = NumberField([x**Integer(3) - Integer(2), x**Integer(2) + Integer(1)],
    names=('a', 'b')); (a, b) = K._first_ngens(2)
>>> f = K.embeddings(ComplexField(Integer(58))); f
[Relative number field morphism:
  From: Number Field in a with defining polynomial x^3 - 2 over its base field
  To:   Complex Field with 58 bits of precision
  Defn: a |--> -0.62996052494743676 - 1.0911236359717214*I
        b |--> -1.9428902930940239e-16 + 1.0000000000000000*I,
...]
Relative number field morphism:
  From: Number Field in a with defining polynomial x^3 - 2 over its base field
  To:   Complex Field with 58 bits of precision
  Defn: a |--> 1.2599210498948731
        b |--> -0.99999999999999999*I
]
```

```python
>>> f[Integer(0)](a)**Integer(3)
2.0000000000000002 - 8.6389229103644993e-16*I
```

```python
>>> f[Integer(0)](b)**Integer(2)
-1.0000000000000001 - 3.8857805861880480e-16*I
```

```python
>>> f[Integer(0)](a+b)
-0.62996052494743693 - 0.091123635971721295*I
```

**free_module** *(base=None, basis=None, map=True)*

Return a vector space over a specified subfield that is isomorphic to this number field, together with the isomorphisms in each direction.

**INPUT:**

- **base** - a subfield
- **basis** - (optional) a list of elements giving a basis over the subfield
- **map** - (default True) whether to return isomorphisms to and from the vector space

### 1.3. Relative number fields
EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b,c> = NumberField([x^2 + 2, x^3 + 2, x^3 + 3]); K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: V, from_V, to_V = K.free_module()
sage: to_V(K.0)
(0, 1)
sage: W, from_W, to_W = K.free_module(base=QQ)
sage: w = to_W(K.0); len(w)
18
sage: w[0]
-127917622658689792301282/48787705559800061938765
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**2 + x**Integer(2) + Integer(2), x**Integer(3) + Integer(3)],
                  names=('a', 'b', 'c')); (a, b, c) = K._first_ngens(3); K
Number Field in a with defining polynomial x^2 + 2 over its base field
>>> V, from_V, to_V = K.free_module()
>>> to_V(K.gen(0))
(0, 1)
>>> W, from_W, to_W = K.free_module(base=QQ)
>>> w = to_W(K.gen(0)); len(w)
18
>>> w[Integer(0)]
-127917622658689792301282/48787705559800061938765
```

`galois_closure(names=None)`

Return the absolute number field $K$ that is the Galois closure of this relative number field.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.galois_closure(c)  # needs sage.groups
Number Field in c with defining polynomial x^16 + 16*x^14 + 28*x^12 + 784*x^10 + 19846*x^8 - 595280*x^6 + 2744476*x^4 + 3212848*x^2 + 29953729
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(4) + x**Integer(3), x**Integer(2) + Integer(2)],
                  names=('a', 'b')); (a, b,) = K._first_ngens(2); K
Number Field in a with defining polynomial x^4 + 3 over its base field
>>> K.galois_closure(c)  # needs sage.groups
Number Field in c with defining polynomial x^16 + 16*x^14 + 28*x^12 + 784*x^10 + 19846*x^8 - 595280*x^6 + 2744476*x^4 + 3212848*x^2 + 29953729
```

`gen(n=0)`

Return the $n$’th generator of this relative number field.

EXAMPLES:
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.gens()
(a, b)
sage: K.gen(0)
a

>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField([x**Integer(4) + Integer(3), x**Integer(2) + Integer(2)],
˓→names=('a', 'b',)); (a, b,) = K._first_ngens(2); K
Number Field in a with defining polynomial x^4 + 3 over its base field

>>> K.gens()
(a, b)

>>> K.gen(Integer(0))
a

gens()

Return the generators of this relative number field.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.gens()
(a, b)

>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField([x**Integer(4) + Integer(3), x**Integer(2) + Integer(2)],
˓→names=('a', 'b',)); (a, b,) = K._first_ngens(2); K
Number Field in a with defining polynomial x^4 + 3 over its base field

>>> K.gens()
(a, b)

is_CM_extension()

Return True is this is a CM extension, i.e. a totally imaginary quadratic extension of a totally real field.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: F.<a> = NumberField(x^2 - 5)
sage: K.<z> = F.extension(x^2 + 7)
sage: K.is_CM_extension()
True
sage: K = CyclotomicField(7)
sage: K_rel = K.relativize(K.gen() + K.gen()^(-1), 'z')
sage: K_rel.is_CM_extension()
True
sage: F = CyclotomicField(3)
sage: K.<z> = F.extension(x^3 - 2)
sage: K.is_CM_extension()
False
A CM field \( K \) such that \( K/F \) is not a CM extension

```python
sage: F.<a> = NumberField(x^2 + 1)
sage: K.<z> = F.extension(x^2 - 3)
sage: K.is_CM_extension()
False
sage: K.is_CM()
True
```

is_absolute() Return False, since this is not an absolute field.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.is_absolute()
False
sage: K.is_relative()
True
```

(continues on next page)
is_free (proof=\texttt{None})

Determine whether or not \( L/K \) is free.

(i.e. if \( \mathcal{O}_L \) is a free \( \mathcal{O}_K \)-module).

INPUT:

\begin{itemize}
  \item proof – default: \texttt{True}
\end{itemize}

EXAMPLES:

\begin{verbatim}
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^2 + 6)
sage: x = polygen(K)
sage: L.<b> = K.extension(x^2 + 3) # extend by \( x^2+3 \)
sage: L.is_free()
False
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> x = polygen(QQ)
>>> K = NumberField(x**Integer(2) + Integer(6), names=('a',)); (a,) = K._first_ngens(1)
>>> x = polygen(K)
>>> L = K.extension(x**Integer(2) + Integer(3), names=('b',)); (b,) = L._first_ngens(1) # extend by \( x^2+3 \)
>>> L.is_free()
False
\end{verbatim}

is_galois ()

For a relative number field, \texttt{is_galois()} is deliberately not implemented, since it is not clear whether this would mean “Galois over \( \mathbb{Q} \)” or “Galois over the given base field”. Use either \texttt{is_galois_absolute()} or \texttt{is_galois_relative()}, respectively.

EXAMPLES:

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField([x**3 - 2, x**2 + x + 1])
sage: k.is_galois() Traceback (most recent call last):
...
NotImplementedError: For a relative number field \( L \) you must use either \( L.is_galois\_relative() \) or \( L.is_galois\_absolute() \) as appropriate
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField([x**Integer(3) - Integer(2), x**Integer(2) + x + Integer(1)], names=('a',)); (a,) = k._first_ngens(1)
>>> k.is_galois()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field \( L \) you must use either \( L.is_galois\_relative() \) or \( L.is_galois\_absolute() \) as appropriate
\end{verbatim}
is_galois_absolute()  
Return True if for this relative extension $L/K$, $L$ is a Galois extension of $\mathbb{Q}$.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 2)
sage: y = polygen(K); L.<b> = K.extension(y^2 - a)
sage: L.is_galois_absolute()  # needs sage.groups
False
```

is_galois_relative()  
Return True if for this relative extension $L/K$, $L$ is a Galois extension of $K$.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 2)
sage: y = polygen(K)
sage: L.<b> = K.extension(y^2 - a)
sage: L.is_galois_relative()  
True
```

The next example previously gave a wrong result; see Issue #9390:

```
sage: F.<a, b> = NumberField([x^2 - 2, x^2 - 3])
sage: F.is_galois_relative()  
True
```

(continues on next page)
is_isomorphic_relative (other, base_isom=None)

For this relative extension $L/K$ and another relative extension $M/K$, return True if there is a $K$-linear isomorphism from $L$ to $M$. More generally, other can be a relative extension $M/K'$ with base_isom an isomorphism from $K$ to $K'$.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<z9> = NumberField(x^6 + x^3 + 1)
sage: R.<z> = PolynomialRing(K)
sage: m1 = 3*z9^4 - 4*z9^3 - 4*z9^2 + 3*z9 - 8
sage: L1 = K.extension(z^2 - m1, b1)
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(6) + x**Integer(3) + Integer(1), names=('z9',))
>>> R = PolynomialRing(K, names=('z',)); (z,) = R._first_ngens(1)
>>> m1 = Integer(3)*z9**Integer(4) - Integer(4)*z9**Integer(3) -
>>> Integer(4)*z9**Integer(2) + Integer(3)*z9 - Integer(8)
>>> L1 = K.extension(z**Integer(2) - m1, 'b1')
>>> L1.is_isomorphic_relative(L2)
False
>>> L1.is_isomorphic(L2)
True
>>> L3 = K.extension(z**Integer(4) - m1, 'b3')
>>> L1.is_isomorphic_relative(L3)
False
```

If we have two extensions over different, but isomorphic, bases, we can compare them by letting base_isom be an isomorphism from self’s base field to other’s base field:

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(6) + x**Integer(3) + Integer(1), names=('z9',))
>>> R = PolynomialRing(K, names=('z',)); (z,) = R._first_ngens(1)
>>> m1 = Integer(3)*z9**Integer(4) - Integer(4)*z9**Integer(3) -
>>> Integer(4)*z9**Integer(2) + Integer(3)*z9 - Integer(8)
>>> L1 = K.extension(z**Integer(2) - m1, 'b1')
>>> G = K.galois_group(); gamma = G.gen()
>>> m2 = (gamma**Integer(2))(m1)
>>> L2 = K.extension(z**Integer(2) - m2, 'b2')
>>> L1.is_isomorphic_relative(L2)
False
>>> L1.is_isomorphic(L2)
True
>>> L3 = K.extension(z**Integer(4) - m1, 'b3')
>>> L1.is_isomorphic_relative(L3)
False
```
Omitting `base_isom` raises a `ValueError` when the base fields are not identical:

```
sage: L1.is_isomorphic_relative(L1cyc)
Traceback (most recent call last):
  ... ValueError: other does not have the same base field as self,
   so an isomorphism from self's base_field to other's base_field
   must be provided using the base_isom parameter.
```
Sage: for g in G:
needs sage.groups
if L1.is_isomorphic_relative(L2, g.as_hom()):
print(g.as_hom())
Ring endomorphism of Number Field in z9 with defining polynomial x^6 + x^3 + 1
Defn: z9 |--> z9^4

lift_to_base(element)

Lift an element of this extension into the base field if possible, or raise a ValueError if it is not possible.

EXAMPLES:

sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^3 - 2)
sage: R.<y> = K[]
sage: L.<b> = K.extension(y^2 - a)
sage: L.lift_to_base(b^4)
a^2
sage: L.lift_to_base(b^6)
2
sage: L.lift_to_base(355/113)
355/113
sage: L.lift_to_base(b)
Traceback (most recent call last):
... ValueError: The element b is not in the base field

logarithmic_embedding(prec=53)

Return the morphism of self under the logarithmic embedding in the category Set.

The logarithmic embedding is defined as a map from the relative number field self to \( \mathbb{R}^n \).

It is defined under Definition 4.9.6 in [Coh1993].
INPUT:

- prec – desired floating point precision.

OUTPUT:

the morphism of self under the logarithmic embedding in the category Set.

EXAMPLES:

```python
sage: K.<k> = CyclotomicField(3)
sage: R.<x> = K[]
sage: L.<l> = K.extension(x^5 + 5)
sage: f = L.logarithmic_embedding()
sage: f(0)
(-1, -1, -1, -1, -1)
sage: f(5)
(3.21887582486820, 3.21887582486820, 3.21887582486820, 3.21887582486820, 3.21887582486820)
```

```python
>>> from sage.all import *

>>> K = CyclotomicField(Integer(3), names=('k',)); (k,) = K._first_ngens(1)
>>> R = K['x']; (x,) = R._first_ngens(1)
>>> L = K.extension(x**Integer(5) + Integer(5), names=('l',)); (l,) = L._
\_first_ngens(1)
>>> f = L.logarithmic_embedding()
>>> f(Integer(0))
(-1, -1, -1, -1, -1)
>>> f(Integer(5))
(3.21887582486820, 3.21887582486820, 3.21887582486820, 3.21887582486820, 3.21887582486820)
```

```python
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<a> = K.extension(t^4 - i)
sage: f = L.logarithmic_embedding()
sage: f(0)
(-1, -1, -1, -1, -1, -1, -1, -1)
sage: f(3)
(2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622)
```

```python
>>> from sage.all import *

>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
\_first_ngens(1)
>>> t = K['t'].gen()
>>> L = K.extension(t**Integer(4) - i, names=('a',)); (a,) = L._first_ngens(1)
>>> f = L.logarithmic_embedding()
>>> f(Integer(0))
(-1, -1, -1, -1, -1, -1, -1, -1)
>>> f(Integer(3))
(2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622, 2.19722457733622)
```

ngens ()

Return the number of generators of this relative number field.

EXAMPLES:
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: K.gens()
(a, b)
sage: K.ngens()
2

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(4) + Integer(3), x**Integer(2) + Integer(2)],
˓→names=('a', 'b')); (a, b,) = K._first_ngens(2); K
Number Field in a with defining polynomial x^4 + 3 over its base field
>>> K.gens()
(a, b)
>>> K.ngens()
2

number_of_roots_of_unity()

Return the number of roots of unity in this relative field.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + x + 1, x^4 + 1])
sage: K.number_of_roots_of_unity()
24

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + x + Integer(1), x**Integer(4) +
˓→Integer(1)], names=('a', 'b')); (a, b,) = K._first_ngens(2)
>>> K.number_of_roots_of_unity()
24

order(*gens, **kwds)

Return the order with given ring generators in the maximal order of this number field.

INPUT:

- gens – list of elements of self; if no generators are given, just returns the cardinality of this number field (∞) for consistency.
- check_is_integral – bool (default: True), whether to check that each generator is integral.
- check_rank – bool (default: True), whether to check that the ring generated by gens is of full rank.
- allow_subfield – bool (default: False), if True and the generators do not generate an order, i.e., they generate a subring of smaller rank, instead of raising an error, return an order in a smaller number field.

The check_is_integral and check_rank inputs must be given as explicit keyword arguments.

EXAMPLES:

sage: P.<a,b,c> = QQ[2^(1/2), 2^(1/3), 3^(1/2)]  # needs sage.symbolic
sage: R = P.order([a,b,c]); R

(continues on next page)
Relative Order generated by
\[((-36372*sqrt3 + 371270)*a^2 + (-89082*sqrt3 + 384161)*a - 422504*sqrt3 \_\_\_\_\_\_
\rightarrow 66959)*sqrt2 + (303148*sqrt3 - 89080)*a^2 + (313664*sqrt3 - 218211)*a \_\_\_\_\_\_
\rightarrow 38053*sqrt3 - 1034933, ((-65954*sqrt3 + 323491)*a^2 + (-110591*sqrt3 + 350011)*a - 351557*sqrt3 +\_\_\_\_\_\_
\rightarrow 77507)*sqrt2 + (264138*sqrt3 - 161552)*a^2 + (285784*sqrt3 - 270906)*a \_\_\_\_\_\_
\rightarrow 63287*sqrt3 - 861151, ((-89292*sqrt3 + 406648)*a^2 + (-137274*sqrt3 + 457033)*a - 449503*sqrt3 +\_\_\_\_\_\_
\rightarrow 102712)*sqrt2 + (332036*sqrt3 - 218718)*a^2 + (373172*sqrt3 - 336261)*a \_\_\_\_\_\_
\rightarrow 83862*sqrt3 - 1101079, ((-164204*sqrt3 + 553344)*a^2 + (-225111*sqrt3 + 646064)*a - 594724*sqrt3 +\_\_\_\_\_\_
\rightarrow 280879)*sqrt2 + (451819*sqrt3 - 402227)*a^2 + (527524*sqrt3 - 551431)*a \_\_\_\_\_\_
\rightarrow 229346*sqrt3 - 1456815, ((-73815*sqrt3 + 257278)*a^2 + (-102896*sqrt3 + 298046)*a - 277080*sqrt3 +\_\_\_\_\_\_
\rightarrow 123726)*sqrt2 + (210072*sqrt3 - 180812)*a^2 + (243357*sqrt3 - 252052)*a \_\_\_\_\_\_
\rightarrow 101026*sqrt3 - 678718]\in Number Field in sqrt2 with defining polynomial \(x^2 - 2\) over its base field

>>> from sage.all import *
>>> P = QQ[(Integer(2)**(Integer(1)/Integer(2)), Integer(2)**(Integer(1)/Integer(3)), Integer(3)**(Integer(1)/Integer(2)))]; (a, b, c,) = P._first_ngens(3) # needs sage.symbolic
>>> R = P.order([a,b,c]); R
Relative Order generated by
\[((-36372*sqrt3 + 371270)*a^2 + (-89082*sqrt3 + 384161)*a - 422504*sqrt3 \_\_\_\_\_\_
\rightarrow 66959)*sqrt2 + (303148*sqrt3 - 89080)*a^2 + (313664*sqrt3 - 218211)*a \_\_\_\_\_\_
\rightarrow 38053*sqrt3 - 1034933, ((-65954*sqrt3 + 323491)*a^2 + (-110591*sqrt3 + 350011)*a - 351557*sqrt3 +\_\_\_\_\_\_
\rightarrow 77507)*sqrt2 + (264138*sqrt3 - 161552)*a^2 + (285784*sqrt3 - 270906)*a \_\_\_\_\_\_
\rightarrow 63287*sqrt3 - 861151, ((-89292*sqrt3 + 406648)*a^2 + (-137274*sqrt3 + 457033)*a - 449503*sqrt3 +\_\_\_\_\_\_
\rightarrow 102712)*sqrt2 + (332036*sqrt3 - 218718)*a^2 + (373172*sqrt3 - 336261)*a \_\_\_\_\_\_
\rightarrow 83862*sqrt3 - 1101079, ((-164204*sqrt3 + 553344)*a^2 + (-225111*sqrt3 + 646064)*a - 594724*sqrt3 +\_\_\_\_\_\_
\rightarrow 280879)*sqrt2 + (451819*sqrt3 - 402227)*a^2 + (527524*sqrt3 - 551431)*a \_\_\_\_\_\_
\rightarrow 229346*sqrt3 - 1456815, ((-73815*sqrt3 + 257278)*a^2 + (-102896*sqrt3 + 298046)*a - 277080*sqrt3 +\_\_\_\_\_\_
\rightarrow 123726)*sqrt2 + (210072*sqrt3 - 180812)*a^2 + (243357*sqrt3 - 252052)*a \_\_\_\_\_\_
\rightarrow 101026*sqrt3 - 678718]\in Number Field in sqrt2 with defining polynomial \(x^2 - 2\) over its base field

The base ring of an order in a relative extension is still \(\mathbb{Z}\): 

```
sage: R.base_ring() # needs sage.symbolic
Integer Ring
```

One must give enough generators to generate a ring of finite index in the maximal order:
pari_absolute_base_polynomial()

Return the PARI polynomial defining the absolute base field, in $y$.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 3]); K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: K.pari_absolute_base_polynomial()
y^2 + 3
sage: type(K.pari_absolute_base_polynomial())
<class 'cypari2.gen.Gen'>
sage: z = ZZ['z'].gen(0)
sage: K = NumberField([z^2 + 2, z^2 + 3, z^2 + 5], names=(a, b, c)); (a, b, c) = K._first_ngens(3); K
Number Field in a with defining polynomial z^2 + 2 over its base field
sage: K.pari_absolute_base_polynomial()
y^4 + 16*y^2 + 4
sage: K.base_field()
Number Field in b with defining polynomial z^2 + 3 over its base field
sage: len(QQ['y'](K.pari_absolute_base_polynomial()).roots(K.base_field()))
4
sage: type(K.pari_absolute_base_polynomial())
<class 'cypari2.gen.Gen'>
```
pari_relative_polynomial()

Return the PARI relative polynomial associated to this number field.

This is always a polynomial in $x$ and $y$, suitable for PARI's pari:rnfinit function. Notice that if this is a relative extension of a relative extension, the base field is the absolute base field.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: k.<i> = NumberField(x^2 + 1)
sage: m.<z> = k.extension(k[w](i,0,1))
>>> m
Number Field in z with defining polynomial w^2 + i over its base field
```

pari_rnf()

Return the PARI relative number field object associated to this relative extension.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: k = NumberField([x**Integer(4) + Integer(3), x**Integer(2) + Integer(2)], names=('a',)); (a,) = k._first_ngens(1)
sage: k.pari_rnf()
[x^4 + 3, [364, -10*x^7 - 87*x^5 - 370*x^3 - 41*x^1, [108, 3], ...]
```
places \( \text{(all\_complex=False, \text{prec}=None)} \)

Return the collection of all infinite places of self.

By default, this returns the set of real places as homomorphisms into \( \text{RIF} \) first, followed by a choice of one of each pair of complex conjugate homomorphisms into \( \text{CIF} \).

On the other hand, if \( \text{prec} \) is not \( \text{None} \), we simply return places into \( \text{RealField}(\text{prec}) \) and \( \text{ComplexField}(\text{prec}) \) (or \( \text{RDF}, \text{CDF} \) if \( \text{prec}=53 \)).

There is an optional flag all\_complex, which defaults to False. If all\_complex is True, then the real embeddings are returned as embeddings into \( \text{CIF} \) instead of \( \text{RIF} \).

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: L.<b, c> = NumberFieldTower([x^2 - 5, x^3 + x + 3])
sage: L.places()  # needs sage.libs.linbox
[Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Real Field with 106 bits of precision
  Defn: b |--> -2.236067977499789696409173668937
  c |--> -1.2134116627622963413213177426,
Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Real Field with 106 bits of precision
  Defn: b |--> 2.236067977499789696409173668937
  c |--> -1.2134116627622963413213177426,
Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Complex Field with 53 bits of precision
  Defn: b |--> -2.236067977499789696409173668937...e-1...*I
  c |--> 0.606705831381... - 1.45061224918844*I,
Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Complex Field with 53 bits of precision
  Defn: b |--> 2.236067977499789696409173668937 - 4.4089209850063e-16*I
  c |--> 0.606705831381115 - 1.45061224918844*I]
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> L = NumberFieldTower([x**Integer(2) - Integer(5), x**Integer(3) + x +...
  Integer(3)], names=('b', 'c',)); (b, c,) = L._first_ngens(2)
>>> L.places()  # needs sage.libs.linbox
[Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Real Field with 106 bits of precision
  Defn: b |--> -2.236067977499789696409173668937
  c |--> -1.2134116627622963413213177426,
Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Real Field with 106 bits of precision
  Defn: b |--> 2.236067977499789696409173668937
  c |--> -1.2134116627622963413213177426,
Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Complex Field with 53 bits of precision
  Defn: b |--> -2.236067977499789696409173668937...e-1...*I
  c |--> 0.606705831381... - 1.45061224918844*I,
Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base...
  To:   Complex Field with 53 bits of precision
  Defn: b |--> 2.236067977499789696409173668937 - 4.4089209850063e-16*I
  c |--> 0.606705831381115 - 1.45061224918844*I]
```
Defn: \( b \mapsto 2.236067977499789696411548005367 \)
\( c \mapsto -1.213411662762229634130492421800 \),

Relative number field morphism:
From: Number Field in \( b \) with defining polynomial \( x^2 - 5 \) over its base...
\( \rightarrow \) field
To: Complex Field with 53 bits of precision
\( b \mapsto -2.23606797749979 \cdots 
\times e-1 \cdots *I
\( c \mapsto 0.606705831381115 - 1.45061224918844*I \),

Relative number field morphism:
From: Number Field in \( b \) with defining polynomial \( x^2 - 5 \) over its base...
\( \rightarrow \) field
To: Complex Field with 53 bits of precision
\( b \mapsto 2.23606797749979 - 4.44089209850063e-16*I
\( c \mapsto 0.606705831381115 - 1.45061224918844*I \)

\textit{polynomial()} 

For a relative number field, \textit{polynomial()} is deliberately not implemented. Either \textit{relative_polynomial()} or \textit{absolute_polynomial()} must be used.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.polynomial()
Traceback (most recent call last):
... NotImplementedError: For a relative number field L you must use either L.relative_polynomial() or L.absolute_polynomial() as appropriate

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**Integer(2) + x + Integer(1), x**Integer(3) + x + Integer(1)], names=('a',)); (a,) = K._first_ngens(1)
>>> K.polynomial()
Traceback (most recent call last):
... NotImplementedError: For a relative number field L you must use either L.relative_polynomial() or L.absolute_polynomial() as appropriate
\end{verbatim}

\textit{relative_degree()} 

Returns the relative degree of this relative number field.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.relative_degree()
2

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**Integer(2) - Integer(17), x**Integer(3) - Integer(2)], names=('a',)); (a,) = K._first_ngens(1)
>>> K.relative_degree()
2
\end{verbatim}
**relative_different()**

Return the relative different of this extension $L/K$ as an ideal of $L$. If you want the absolute different of $L/Q$, use `absolute_different()`.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: PK.<t> = K[

sage: L.<a> = K.extension(t^4 - i)
sage: L.relative_different()
Fractional ideal (4)
```

**relative_discriminant()**

Return the relative discriminant of this extension $L/K$ as an ideal of $K$. If you want the (rational) discriminant of $L/Q$, use e.g. `L.absolute_discriminant()`.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<b> = K.extension(t^4 - i)
sage: L.relative_discriminant()
Fractional ideal (256)
sage: PQ.<X> = QQ[

sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[

sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.relative_discriminant() == F.ideal(4*b)
True
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._

>>> t = K['t'].gen()

>>> L = K.extension(t^4 - i, names=('a',)); (a,) = L._first_ 

>>> L.relative_discriminant()
Fractional ideal (4)
```

```python
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._

>>> t = K['t'].gen()

>>> L = K.extension(t^4 - i, names=('a',)); (a,) = L._first_ 

>>> L.relative_discriminant()
Fractional ideal (256)
```

```python
>>> PQ = QQ['X']; (X,) = PQ._first_ngens(1)

>>> F = NumberField([X**Integer(2) - Integer(2), X**Integer(2) - Integer(3)],

>>> PF = F['Y']; (Y,) = PF._first_ngens(1)

>>> K = F.extension(Y**Integer(2) - (Integer(1) + a)*(a + b)*a*b, names=('c', 

>>> K.relative_discriminant() == F.ideal(Integer(4)*b)
True
```
relative_polynomial()
Return the defining polynomial of this relative number field over its base field.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.relative_polynomial()
x^2 + x + 1

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**Integer(2) + x + Integer(1), x**Integer(3) + x +...
˓→Integer(1)], names=('a',)); (a,) = K._first_ngens(1)
>>> K.relative_polynomial()
x^2 + x + 1
```

Use absolute_polynomial() for a polynomial that defines the absolute extension:

```
sage: K.absolute_polynomial()
x^6 + 3*x^5 + 8*x^4 + 9*x^3 + 7*x^2 + 6*x + 3

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + Integer(2), x**Integer(3) +...
˓→Integer(3)], names=(a, b, c,)); (a, b, c,) = K._first_ngens(3); K
Number Field in a with defining polynomial x^2 + 2 over its base field
>>> V, from_V, to_V = K.relative_vector_space()
>>> from_V(V.gen(0))
1
>>> to_V(K.gen(0))
(0, 1)
```

relative_vector_space (base=None, *args, **kwds)
Return vectorspace over the base field of self and isomorphisms from the vector space to self and in the other direction.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a,b,c> = NumberField([x^2 + 2, x^3 + 2, x^3 + 3]); K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: V, from_V, to_V = K.relative_vector_space()
```

(continues on next page)
The underlying vector space and maps is cached:

```python
sage: W, from_V, to_V = K.relative_vector_space()
```

relativize\((\alpha, \text{names})\)

Given an element in \(\text{self}\) or an embedding of a subfield into \(\text{self}\), return a relative number field \(K\) isomorphic to \(\text{self}\) that is relative over the absolute field \(\mathbb{Q}(\alpha)\) or the domain of \(\alpha\), along with isomorphisms from \(K\) to \(\text{self}\) and from \(\text{self}\) to \(K\).

**INPUT:**

- \(\alpha\) – an element of \(\text{self}\), or an embedding of a subfield into \(\text{self}\)
- \(\text{names}\) – name of generator for output field \(K\).

**OUTPUT:** \(K\) – a relative number field

Also, \(K.\text{structure}()\) returns \(\text{from}_K\) and \(\text{to}_K\), where \(\text{from}_K\) is an isomorphism from \(K\) to \(\text{self}\) and \(\text{to}_K\) is an isomorphism from \(\text{self}\) to \(K\).

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^4 + 3, x^2 + 2]); K
Number Field in a with defining polynomial x^4 + 3 over its base field
sage: L.<z,w> = K.relativize(a^2)
sage: z^2
z^2
sage: w^2
-3
sage: L
Number Field in z with defining polynomial x^4 + (-2*w + 4)*x^2 + 4*w + 1 over its base field
sage: L.base_field()
Number Field in w with defining polynomial x^2 + 3
```

(continued from previous page)
Number Field in $z$ with defining polynomial $x^4 + (-2w + 4)x^2 + 4w + 1$ over its base field

Number Field in $w$ with defining polynomial $x^2 + 3$

Now suppose we have $K$ below $L$ below $M$:

```python
sage: M = NumberField(x^8 + 2, 'a'); M
Number Field in a with defining polynomial x^8 + 2
sage: L, L_into_M, _ = M.subfields(4)[0]; L
Number Field in a0 with defining polynomial x^4 + 2
sage: K, K_into_L, _ = L.subfields(2)[0]; K
Number Field in a0_0 with defining polynomial x^2 + 2
sage: K_into_M = L_into_M * K_into_L
sage: L_over_K = L.relativize(K_into_L, c); L_over_K
Number Field in c with defining polynomial x^2 + a0_0 over its base field
sage: L_over_K_to_L, L_to_L_over_K = L_over_K.structure()
```

Test relativizing a degree 6 field over its degree 2 and degree 3 subfields, using both an explicit element:

```python
sage: K.<a> = NumberField(x^6 + 2); K
Number Field in a with defining polynomial x^6 + 2
```
Here we explicitly relativize over an element of $K_2$ (not the generator):

```python
sage: L = K.relativize(K3_into_K, 'b'); L
Number Field in b with defining polynomial $x^2 + a_0$ over its base field
sage: L_to_K, K_to_L = L.structure()
sage: L_over_K2 = L.relativize(K_to_L(K2_into_K(K2.gen() + 1)), 'c'); L_over_K2
Number Field in c0 with defining polynomial $x^3 - c1 + 1$ over its base field
sage: L_over_K2.base_field()
Number Field in c1 with defining polynomial $x^2 - 2*x + 3$
```

Here we use a morphism to preserve the base field information:

```python
sage: K2_into_L = K_to_L * K2_into_K
sage: L_over_K2 = L.relativize(K2_into_L, 'c'); L_over_K2
Number Field in c with defining polynomial $x^3 - a0$ over its base field
sage: L_over_K2.base_field() is K2
True
```

`roots_of_unity()`

Return all the roots of unity in this relative field, primitive or not.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + x + 1, x^4 + 1])
sage: rts = K.roots_of_unity()
sage: len(rts)
24
sage: all(u in rts for u in [b*a, -b^2*a - b^2, b^3, -a, b*a + b])
True
```
subfields (degree=0, name=None)

Return all subfields of this relative number field self of the given degree, or of all possible degrees if degree is 0. The subfields are returned as absolute fields together with an embedding into self. For the case of the field itself, the reverse isomorphism is also provided.

EXAMPLES:

```python
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.subfields(2)
[(Number Field in c0 with defining polynomial x^2 - 24*x + 96,
  Ring morphism:
   From: Number Field in c0 with defining polynomial x^2 - 24*x + 96
   To: Number Field in c with defining polynomial
      Y^2 + (-2*b - 3)*a - 2*b - 6 over its base field
   Defn: c0 |--> -4*b + 12,
  None),
(Number Field in c1 with defining polynomial x^2 - 24*x + 120,
  Ring morphism:
   From: Number Field in c1 with defining polynomial x^2 - 24*x + 120
   To: Number Field in c with defining polynomial
      Y^2 + (-2*b - 3)*a - 2*b - 6 over its base field
   Defn: c1 |--> 2*b*a + 12,
  None),
(Number Field in c2 with defining polynomial x^2 - 24*x + 72,
  Ring morphism:
   From: Number Field in c2 with defining polynomial x^2 - 24*x + 72
   To: Number Field in c with defining polynomial
      Y^2 + (-2*b - 3)*a - 2*b - 6 over its base field
   Defn: c2 |--> -6*a + 12,
  None)]
```

`sage: K.subfields(8, 'w')
[(Number Field in w0 with defining polynomial x^8 - 12*x^6 + 36*x^4 - 36*x^2 + 9,
  Ring morphism:
   From: Number Field in w0 with defining polynomial
      x^8 - 12*x^6 + 36*x^4 - 36*x^2 + 9
   To: Number Field in c with defining polynomial
      Y^2 + (-2*b - 3)*a - 2*b - 6 over its base field
   Defn: w0 |--> (-1/2*b*a + 1/2*b + 1/2)*c,
  Relative number field morphism:```
From: Number Field in $c$ with defining polynomial $Y^2 + (-2*b - 3)*a - 2*b - 6$ over its base field
To: Number Field in $w_0$ with defining polynomial $x^8 - 12*x^6 + 36*x^4 - 36*x^2 + 9$
Defn: $c |\mapsto -1/3*w_0^7 + 4*w_0^5 - 12*w_0^3 + 11*w_0$
a |\mapsto 1/3*w_0^6 - 10/3*w_0^4 + 5*w_0^2
b |\mapsto -2/3*w_0^6 + 7*w_0^4 - 14*w_0^2 + 6)\]

sage: K.subfields(3)
[]

from sage.all import *

PQ = QQ['X']; (X,) = PQ._first_ngens(1)
F = NumberField([X**Integer(2) - Integer(2), X**Integer(2) - Integer(3)],
˓→names=('a', 'b')); (a, b,) = F._first_ngens(2)
PF = F['Y']; (Y,) = PF._first_ngens(1)
K = F.extension(Y**Integer(2) - (Integer(1) + a)*(a + b)*a*b, names=('c',
˓→)); (c,) = K._first_ngens(1)
K.subfields(Integer(2))

(Number Field in $c_0$ with defining polynomial $x^2 - 24*x + 96$,
Ring morphism:
  From: Number Field in $c_0$ with defining polynomial $x^2 - 24*x + 96$
  To:   Number Field in $c$ with defining polynomial $Y^2 + (-2*b - 3)*a - 2*b - 6$ over its base field
  Defn: $c_0 |\mapsto -4*b + 12$,
None),
(Number Field in $c_1$ with defining polynomial $x^2 - 24*x + 120$,
Ring morphism:
  From: Number Field in $c_1$ with defining polynomial $x^2 - 24*x + 120$
  To:   Number Field in $c$ with defining polynomial $Y^2 + (-2*b - 3)*a - 2*b - 6$ over its base field
  Defn: $c_1 |\mapsto 2*b*a + 12$,
None),
(Number Field in $c_2$ with defining polynomial $x^2 - 24*x + 72$,
Ring morphism:
  From: Number Field in $c_2$ with defining polynomial $x^2 - 24*x + 72$
  To:   Number Field in $c$ with defining polynomial $Y^2 + (-2*b - 3)*a - 2*b - 6$ over its base field
  Defn: $c_2 |\mapsto -6*a + 12$,
None)

>>> K.subfields(Integer(8), 'w')

(Number Field in $w_0$ with defining polynomial $x^8 - 12*x^6 + 36*x^4 - 36*x^2 + 9$,
Ring morphism:
  From: Number Field in $w_0$ with defining polynomial $x^8 - 12*x^6 + 36*x^4 - 36*x^2 + 9$
  To:   Number Field in $c$ with defining polynomial $Y^2 + (-2*b - 3)*a - 2*b - 6$ over its base field
  Defn: $w_0 |\mapsto (-1/2*b*a + 1/2*b + 1/2)*c$,
Relative number field morphism:
  From: Number Field in $c$ with defining polynomial $Y^2 + (-2*b - 3)*a - 2*b - 6$ over its base field
  To:   Number Field in $w_0$ with defining polynomial $x^8 - 12*x^6 + 36*x^4 - 36*x^2 + 9$)
uniformizer \((P, \text{others='positive'})\)

Returns an element of \(self\) with valuation 1 at the prime ideal \(P\).

**INPUT:**

- \(self\) – a number field
- \(P\) – a prime ideal of \(self\)
- \(\text{others}\) – either "positive" (default), in which case the element will have non-negative valuation at all other primes of \(self\), or "negative", in which case the element will have non-positive valuation at all other primes of \(self\).

**Note:** When \(P\) is principal (e.g., always when \(self\) has class number one), the result may or may not be a generator of \(P\)!

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + 23, x^2 - 3])
sage: P = K.prime_factors(5)[0]; P
Fractional ideal (5, 1/2*a + b - 5/2)
sage: u = K.uniformizer(P)
sage: u.valuation(P)
1
sage: (P, 1) in K.factor(u)
True
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + Integer(23), x**Integer(2) - Integer(3)],
                 names=('a', 'b')); (a, b) = K._first_ngens(2)
>>> P = K.prime_factors(Integer(5))[Integer(0)]; P
Fractional ideal (5, 1/2*a + b - 5/2)
>>> u = K.uniformizer(P)
>>> u.valuation(P)
1
>>> (P, Integer(1)) in K.factor(u)
True
```

vector_space \(^{(*args, **kwargs)}\)

For a relative number field, \(vector_space()\) is deliberately not implemented, so that a user cannot confuse \(relative_vector_space()\) with \(absolute_vector_space()\).

**EXAMPLES:**
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.vector_space()
Traceback (most recent call last):
  ... 
NotImplementedError: For a relative number field L you must use either L.relative_vector_space() or L.absolute_vector_space() as appropriate

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**Integer(2) - Integer(17), x**Integer(3) - Integer(2)], names=('a',)); (a,) = K._first_ngens(1)
>>> K.vector_space()
Traceback (most recent call last):
  ... 
NotImplementedError: For a relative number field L you must use either L.relative_vector_space() or L.absolute_vector_space() as appropriate

sage.rings.number_field.number_field_rel.NumberField_relative_v1(base_field, poly, name, latex_name, canonical_embedding=None)

Used for unpickling old pickles.

EXAMPLES:

sage: from sage.rings.number_field.number_field_rel import NumberField_relative_v1
sage: R.<x> = CyclotomicField(3)[]
sage: NumberField_relative_v1(CyclotomicField(3), x^2 + 7, 'a', 'a')
Number Field in a with defining polynomial x^2 + 7 over its base field

sage: from sage.all import *
>>> from sage.rings.number_field.number_field_rel import NumberField_relative_v1
>>> R = CyclotomicField(Integer(3))[x]; (x,) = R._first_ngens(1)
>>> NumberField_relative_v1(CyclotomicField(Integer(3)), x**Integer(2) + Integer(7), 'a', 'a')
Number Field in a with defining polynomial x^2 + 7 over its base field

sage.rings.number_field.number_field_rel.is_RelativeNumberField(x)

Return True if x is a relative number field.

EXAMPLES:

sage: from sage.rings.number_field.number_field_rel import is_RelativeNumberField
sage: x = polygen(ZZ, 'x')

sage: is_RelativeNumberField(NumberField(x^2+1,'a'))
doctest:warning...
DeprecationWarning: The function is_RelativeNumberField is deprecated; use 'isinstance(..., NumberField_relative)' instead.
See https://github.com/sagemath/sage/issues/38124 for details.
False

sage: k.<a> = NumberField(x^3 - 2)
sage: l.<b> = k.extension(x^3 - 3); l
Number Field in b with defining polynomial x^3 - 3 over its base field

sage: is_RelativeNumberField(l)
True
1.4 Splitting fields of polynomials over number fields

AUTHORS:

- Jeroen Demeyer (2014-01-02): initial version for Issue #2217
- Jeroen Demeyer (2014-01-03): added abort_degree argument, Issue #15626

class sage.rings.number_field.splitting_field.SplittinData(_pol, _dm)

Bases: object

A class to store data for internal use in splitting_field(). It contains two attributes \( \text{pol} \) (polynomial), \( \text{dm} \) (degree multiple), where \( \text{pol} \) is a PARI polynomial and \( \text{dm} \) a Sage Integer.

\( \text{dm} \) is a multiple of the degree of the splitting field of \( \text{pol} \) over some field \( E \). In splitting_field(), \( E \) is the field containing the current field \( K \) and all roots of other polynomials inside the list \( L \) with \( \text{dm} \) less than this \( \text{dm} \).

key()

Return a sorting key. Compare first by degree bound, then by polynomial degree, then by discriminant.

EXAMPLES:

```python
sage: from sage.rings.number_field.splitting_field import SplittingData
sage: L = []
sage: L.append(SplittingData(pari("x^2 + 1"), 1))
sage: L.append(SplittingData(pari("x^3 + 1"), 1))
sage: L.append(SplittingData(pari("x^2 + 7"), 2))
sage: L.append(SplittingData(pari("x^3 + 1"), 2))
sage: L.append(SplittingData(pari("x^3 + x^2 + x + 1"), 2))
sage: L.sort(key=lambda x: x.key()); L
[SplittingData(x^2 + 1, 1), SplittingData(x^3 + 1, 1), SplittingData(x^2 + 7, 2), SplittingData(x^3 + x^2 + x + 1, 2)]
```
>>> from sage.all import *
>>> from sage.rings.number_field.splitting_field import SplittingData
>>> L = []
>>> L.append(SplittingData(pari("x^2 + 1"), Integer(1)))
>>> L.append(SplittingData(pari("x^3 + 1"), Integer(1)))
>>> L.append(SplittingData(pari("x^2 + 7"), Integer(2)))
>>> L.append(SplittingData(pari("x^3 + x^2 + x + 1"), Integer(2)))
>>> L.sort(key=lambda x: x.key()); L
[SplittingData(x^2 + 1, 1), SplittingData(x^3 + 1, 1), SplittingData(x^2 + 7, 2), SplittingData(x^3 + x^2 + x + 1, 2), SplittingData(x^3 + 1, 2)]

poldegree()

Return the degree of self.pol

EXAMPLES:

sage: from sage.rings.number_field.splitting_field import SplittingData
sage: SplittingData(pari("x^123 + x + 1"), 2).poldegree()
123

exception sage.rings.number_field.splitting_field.SplittingFieldAbort (div, mult)

Special exception class to indicate an early abort of splitting_field().

EXAMPLES:

sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: raise SplittingFieldAbort(20, 60)
Traceback (most recent call last):
... SplittingFieldAbort: degree of splitting field is a multiple of 20
sage: raise SplittingFieldAbort(12, 12)
Traceback (most recent call last):
... SplittingFieldAbort: degree of splitting field equals 12

(continues on next page)
SplittngFieldAbort: degree of splitting field equals 12

sage.rings.number_field.splitting_field.splitting_field(poly, name=None, map=False, degree_multiple=None, abort_degree=None, simplify=True, simplify_all=False)

Compute the splitting field of a given polynomial, defined over a number field.

**INPUT:**

- **poly** – a monic polynomial over a number field
- **name** – a variable name for the number field
- **map** – (default: False) also return an embedding of **poly** into the resulting field. Note that computing this embedding might be expensive.
- **degree_multiple** – a multiple of the absolute degree of the splitting field. If degree_multiple equals the actual degree, this can enormously speed up the computation.
- **abort_degree** – abort by raising a `SplittingFieldAbort` if it can be determined that the absolute degree of the splitting field is strictly larger than abort_degree.
- **simplify** – (default: True) during the algorithm, try to find a simpler defining polynomial for the inter-
medi ate number fields using PARI's `polredbest()` . This usually speeds up the computation but can also considerably slow it down. Try and see what works best in the given situation.
- **simplify_all** – (default: False) If True, simplify intermediate fields and also the resulting number field.

**OUTPUT:**

If **map** is False, the splitting field as an absolute number field. If **map** is True, a tuple (K, phi) where phi is an embedding of the base field in K.

**EXAMPLES:**

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = (x^3 + 2).splitting_field(); K
Number Field in a with defining polynomial x^6 + 3*x^5 + 6*x^4 + 11*x^3 + 12*x^2 - 3*x + 1
sage: K.<a> = (x^3 - 3*x + 1).splitting_field(); K
Number Field in a with defining polynomial x^3 - 3*x + 1
```

```python
>>> from sage.all import *
>>> R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
>>> K = (x^3 - Integer(3)*x + Integer(1)).splitting_field(names=('a',)); (a,) = K._
˓→first_ngens(1); K
Number Field in a with defining polynomial x^3 - 3*x + 1
```

The **simplify** and simplify_all flags usually yield fields defined by polynomials with smaller coefficients. By default, simplify is True and simplify_all is False.
Reducible polynomials also work:

```
sage: pol = (x^4 - 1)*(x^2 + 1/2)*(x^2 + 1/3)
sage: pol.splitting_field('a', simplify_all=True)
```

Number Field in a with defining polynomial

\[ x^8 - x^4 + 1 \]

```python
>>> from sage.all import *
```
Relative situation:

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^3 + 2)
sage: S.<t> = PolynomialRing(K)
sage: L.<b> = (t^2 - a).splitting_field()
sage: L
Number Field in b with defining polynomial t^6 + 2
```

With `map=True`, we also get the embedding of the base field into the splitting field:

```python
sage: L, phi = (t^2 - a).splitting_field(map=True)
sage: phi
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + 2
  To:   Number Field in b with defining polynomial t^6 + 2
  Defn: a |--> b^2
```

(continues on next page)
We can enable verbose messages:

```python
from sage.misc.verbose import setVerbose
setVerbose(2)
K.<a> = (x^3 - x + 1).splitting_field()
```

```python
verbose 1 (...: splitting_field.py, splitting_field) Starting field: y
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to factor: [(3, 0)]
verbose 2 (...: splitting_field.py, splitting_field) Done factoring (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to handle: [(2, 2), (3, 3)]
verbose 1 (...: splitting_field.py, splitting_field) Bounds for absolute degree: [6, 6]
verbose 2 (...: splitting_field.py, splitting_field) Handling polynomial x^2 + 23
verbose 1 (...: splitting_field.py, splitting_field) New field before simplifying: x^2 + 23 (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) New field: y^2 - y + 6 (time = ...)
verbose 2 (...: splitting_field.py, splitting_field) Converted polynomials to new field (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to factor: []
verbose 2 (...: splitting_field.py, splitting_field) Done factoring (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to handle: [(3, 3)]
verbose 1 (...: splitting_field.py, splitting_field) Bounds for absolute degree: [6, 6]
verbose 2 (...: splitting_field.py, splitting_field) Handling polynomial x^3 - x + 1
verbose 1 (...: splitting_field.py, splitting_field) New field: y^6 + 3*y^5 + ...
verbose 2 (...: splitting_field.py, splitting_field) Converted polynomials to new field (time = ...)
```

(continues on next page)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to handle: [(3, -3)]
verbose 1 (...: splitting_field.py, splitting_field) Bounds for absolute degree: [6, 6]
verbose 2 (...: splitting_field.py, splitting_field) Handling polynomial x^3 - x + 1
verbose 1 (...: splitting_field.py, splitting_field) New field: y^6 + 3*y^5 + 19*y^4 + 35*y^3 + 127*y^2 + 73*y + 271 (time = ...)

>>> setVerbose(Integer(0))

Try all Galois groups in degree 4. We use a quadratic base field such that polgalois() cannot be used:

```python
sage: R.<x> = PolynomialRing(QuadraticField(-11))
sage: C2C2pol = x^4 - 10*x^2 + 1
sage: C2C2pol.splitting_field(x)
Number Field in x with defining polynomial x^8 + 24*x^6 + 608*x^4 + 9792*x^2 + 53824
sage: C4pol = x^4 + x^3 + x^2 + x + 1
sage: C4pol.splitting_field(x)
Number Field in x with defining polynomial x^8 - x^7 - 2*x^6 + 5*x^5 + x^4 + 15*x^3 - 18*x^2 - 27*x + 81
sage: D8pol = x^4 - 2
sage: D8pol.splitting_field(x)
Number Field in x with defining polynomial x^16 + 8*x^15 + 68*x^14 + 336*x^13 + 1514*x^12 + 5080*x^11 + 14912*x^10 + 35048*x^9 + 64959*x^8 + 93416*x^7 + 88216*x^6 + 41608*x^5 - 25586*x^4 - 60048*x^3 - 16628*x^2 + 12008*x + 34961
sage: A4pol = x^4 - 4*x^3 + 14*x^2 - 28*x + 21
sage: A4pol.splitting_field(x)
Number Field in x with defining polynomial x^24 - 20*x^23 + 290*x^22 - 3048*x^21 + 26147*x^20 - 186132*x^19 + 1130626*x^18 - 5913784*x^17 + 26899345*x^16 - 106792132*x^15 + 371066538*x^14 - 1127792656*x^13 + 2991524876*x^12 - 6888328132*x^11 + 13655960064*x^10 - 2300783036*x^9 + 3224479382*x^8 - 36347834476*x^7 + 30850898884*x^6 - 16707053128*x^5 + 1896946429*x^4 + 4832907884*x^3 - 3038258802*x^2 - 200383596*x + 593179173
```

```python
>>> from sage.all import *
R = PolynomialRing(QuadraticField(-Integer(11)), names=('x',)); (x,) = R._first_ngens(1)
C2C2pol = x**Integer(4) - Integer(10)*x**Integer(2) + Integer(1)
C2C2pol.splitting_field('x')
Number Field in x with defining polynomial x^8 + 24*x^6 + 608*x^4 + 9792*x^2 + 53824
C4pol = x**Integer(4) + x**Integer(3) + x**Integer(2) + x + Integer(1)
C4pol.splitting_field('x')
Number Field in x with defining polynomial x^8 - x^7 - 2*x^6 + 5*x^5 + x^4 + 15*x^3 - 18*x^2 - 27*x + 81
D8pol = x**Integer(4) - Integer(2)
D8pol.splitting_field('x')
Number Field in x with defining polynomial x^16 + 8*x^15 + 68*x^14 + 336*x^13 + 1514*x^12 + 5080*x^11 + 14912*x^10 + 35048*x^9 + 64959*x^8 + 93416*x^7 + 88216*x^6 + 41608*x^5 - 25586*x^4 - 60048*x^3 - 16628*x^2 + 12008*x + 34961
```

(continues on next page)
Some bigger examples:

```
sage: R.<x> = PolynomialRing(QQ)
sage: pol15 = chebyshev_T(31, x) - 1  # 2^30*(x-1)*minpoly(cos(2*pi/31))^2
sage: pol15.splitting_field('a')
Number Field in a with defining polynomial
x^15 - x^14 - 14*x^13 + 13*x^12 + 78*x^11 - 66*x^10 + 220*x^9 + 165*x^8 + 330*x^7 - 210*x^6 - 252*x^5 + 126*x^4 + 84*x^3 - 28*x^2 - 8*x + 1
```

If you somehow know the degree of the field in advance, you should add a degree_multiple argument. This can speed up the computation, in particular for polynomials of degree >= 12 or for relative extensions:

```
sage: pol15.splitting_field('a', degree_multiple=15)
Number Field in a with defining polynomial
x^15 + x^14 - 14*x^13 - 13*x^12 + 78*x^11 + 66*x^10 - 220*x^9 - 165*x^8 + 330*x^7 - 210*x^6 - 252*x^5 - 126*x^4 + 84*x^3 + 28*x^2 - 8*x - 1
```

A value for degree_multiple which isn’t actually a multiple of the absolute degree of the splitting field can either result in a wrong answer or the following exception:
Compute the Galois closure as the splitting field of the defining polynomial:

```python
sage: R.<x> = PolynomialRing(QQ)
sage: pol48 = x^6 - 4*x^4 + 12*x^2 - 12
sage: K.<a> = NumberField(pol48)
sage: L.<b> = pol48.change_ring(K).splitting_field()
sage: L
Number Field in b with defining polynomial x^48 ...
```

Try all Galois groups over $\mathbb{Q}$ in degree 5 except for $S_5$ (the latter is infeasible with the current implementation):

```python
sage: C5pol = x^5 + x^4 - 4*x^3 - 3*x^2 + 3*x + 1
sage: C5pol.splitting_field(x)
Number Field in x with defining polynomial x^5 + x^4 - 4*x^3 - 3*x^2 + 3*x + 1
sage: A5pol = x^5 - x^4 + 2*x^2 - 2*x + 2
sage: A5pol.splitting_field(x)
Number Field in x with defining polynomial x^60 ...
```
We can use the `abort_degree` option if we don’t want to compute fields of too large degree (this can be used to check whether the splitting field has small degree):

```
sage: (x^5 + x + 3).splitting_field('b', abort_degree=119)
```
```
Traceback (most recent call last):
  ...  
SplittingFieldAbort: degree of splitting field equals 120
```
```
sage: (x^10 + x + 3).splitting_field('b', abort_degree=60). # long time (10s on sage.math, 2014)
```
```
Traceback (most recent call last):
  ...  
SplittingFieldAbort: degree of splitting field is a multiple of 180
```

Use the `degree_divisor` attribute to recover the divisor of the degree of the splitting field or `degree_multiple` to recover a multiple:

```
sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: try:
    # long time (4s on sage.math, 2014)
    (x^8 + x + 1).splitting_field('b', abort_degree=Integer(60), simplify=False)
    except SplittingFieldAbort as e:
        print(e.degree_divisor)
        print(e.degree_multiple)
```
```
120
1440
```

```
sage: from sage.all import *
```
```
sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: try:
    # long time (4s on sage.math, 2014)
    (x^8 + x + 1).splitting_field('b', abort_degree=Integer(60), simplify=False)
```
```
(continues on next page)
... except SplittingFieldAbort as e:
    print(e.degree_divisor)
    print(e.degree_multiple)
120
1440

1.5 Galois groups of number fields

AUTHORS:

- David Loeffler (2009): rewrote to give explicit homomorphism groups

sage.rings.number_field.galois_group.GaloisGroup
alias of GaloisGroup_v1

class sage.rings.number_field.galois_group.GaloisGroupElement

Bases: PermutationGroupElement

An element of a Galois group. This is stored as a permutation, but may also be made to act on elements of the field (generally returning elements of its Galois closure).

EXAMPLES:

```
sage: K.<w> = QuadraticField(-7); G = K.galois_group()
sage: G[1]
(1,2)
sage: G[1](w + 2)
-w + 2
sage: x = polygen(ZZ, 'x')
sage: L.<v> = NumberField(x^3 - 2); G = L.galois_group(names='y')
sage: G[4]
(1,5)(2,4)(3,6)
sage: G[4](v)
1/18*y^4
sage: G[4](G[4](v))
-1/36*y^4 - 1/2*y
sage: G[4](G[4](G[4](v)))
1/18*y^4
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(7), names=('w',)); (w,) = K._first_ngens(1); G =
    → K.galois_group()
>>> G[Integer(1)]
(1,2)
>>> G[Integer(1)](w + Integer(2))
-w + 2
>>> x = polygen(ZZ, 'x')
>>> L = NumberField(x**Integer(3) - Integer(2), names=('v',)); (v,) = L._first_
    → ngens(1); G = L.galois_group(names='y')
>>> G[Integer(4)]
(1,5)(2,4)(3,6)
```
as_hom()

Return the homomorphism $L \rightarrow L$ corresponding to self, where $L$ is the Galois closure of the ambient number field.

EXAMPLES:

```python
sage: G = QuadraticField(-7, 'w').galois_group()
```
```python
sage: G[1].as_hom()
```
```
Ring endomorphism of Number Field in w with defining polynomial x^2 + 7
with w = 2.645751311064591?*I
Defn: w |--> -w
```

ramification_degree($P$)

Return the greatest value of $v$ such that $s$ acts trivially modulo $P^v$. Should only be used if $P$ is prime and $s$ is in the decomposition group of $P$.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
```
```python
sage: K.<b> = NumberField(x^3 - 3, 'a').galois_closure()  
```
```python
sage: G = K.galois_group()  
```
```python
sage: P = K.primes_above(3)[0]  
```
```python
sage: s = hom(K, K, 1/18*b^4 - 1/2*b)
```
```python
sage: G(s).ramification_degree(P)  
```
```
4
```

```python
sage: x = polygen(ZZ, 'x')
```
```python
sage: K = NumberField(x**Integer(3) - Integer(3), 'a').galois_closure()  
```
```python
sage: G = K.galois_group()  
```
```python
sage: P = K.primes_above(Integer(3))[Integer(0)]
```
```python
sage: s = hom(K, K, Integer(1)/Integer(18)*b**Integer(4) - Integer(1)/Integer(2)*b)
```
```python
sage: G(s).ramification_degree(P)  
```
```
4
```

class sage.rings.number_field.galois_group.GaloisGroup_subgroup

1.5. Galois groups of number fields 251
Bases: GaloisSubgroup_perm

A subgroup of a Galois group, as returned by functions such as decomposition_group.

INPUT:

- ambient – the ambient Galois group
- gens – a list of generators for the group
- gap_group – a gap or libgap permutation group, or a string defining one (default: None)
- domain – a set on which this permutation group acts; extracted from ambient if not specified
- category – the category for this object
- canonicalize – if True, sorts and removes duplicates
- check – whether to check that generators actually lie in the ambient group

EXAMPLES:

```python
sage: from sage.rings.number_field.galois_group import GaloisGroup_subgroup
sage: x = polygen(ZZ, 'x')

sage: G = NumberField(x^3 - x - 1, 'a').galois_closure('b').galois_group()

sage: GaloisGroup_subgroup( G, [G([(1,2,3),(4,5,6)])])
Subgroup generated by [(1,2,3)(4,5,6)] of (Galois group 6T2 ([3]2) with order 6 of x^6 - 6*x^4 + 9*x^2 + 23)

sage: K.<a> = NumberField(x^6 - 3*x^2 - 1)

sage: L.<b> = K.galois_closure()

sage: G = L.galois_group()

sage: P = L.primes_above(3)[0]

sage: H = G.decomposition_group(P)

sage: H.order()
3

sage: G = NumberField(x^3 - x - 1, 'a').galois_closure('b').galois_group()

sage: H = G.subgroup([G([(1,2,3),(4,5,6)])])

Subgroup generated by [(1,2,3)(4,5,6)] of (Galois group 6T2 ([3]2) with order 6 of x^6 - 6*x^4 + 9*x^2 + 23)
```

(continues on next page)
>>> H.order()
3

>>> G = NumberField(x**Integer(3) - x - Integer(1), 'a').galois_closure('b').
galois_group()

>>> H = G.subgroup([G([[Integer(1),Integer(2),Integer(3)],
→Integer(4),Integer(5),
→Integer(6)])]])

Subgroup generated by [(1,2,3)(4,5,6)] of
(Galois group 6T2 ([3]2) with order 6 of x^6 - 6*x^4 + 9*x^2 + 23)

Element
alias of :class:`GaloisGroupElement`

**fixed_field**(name=None, polred=None, threshold=None)

Return the fixed field of this subgroup (as a subfield of the Galois closure of the number field associated to
the ambient Galois group).

**INPUT:**

- **name** – a variable name for the new field.
- **polred** – whether to optimize the generator of the newly created field
  for a simpler polynomial, using PARI’s :func:`pari:polredbest`. Defaults to True when the degree of the
  fixed field is at most 8.
- **threshold** – positive number; polred only performed if the cost is at most this threshold

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')

sage: L.<a> = NumberField(x**4 + 1)

sage: G = L.galois_group()

sage: H = G.decomposition_group(L.primes_above(3)[0])

sage: H.fixed_field()
(Number Field in a0 with defining polynomial x^2 + 2 with a0 = a^3 + a,
Ring morphism:
  From: Number Field in a0 with defining polynomial x^2 + 2 with a0 = a^3 + a
  To:   Number Field in a with defining polynomial x^4 + 1
  Defn: a0 |--> a^3 + a)
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> L = NumberField(x**Integer(4) + Integer(1), names=('a',)); (a,) = L._
→first_ngens(1)

>>> G = L.galois_group()

>>> H = G.decomposition_group(L.primes_above(Integer(3))[Integer(0)])

>>> H.fixed_field()
(Number Field in a0 with defining polynomial x^2 + 2 with a0 = a^3 + a,
Ring morphism:
  From: Number Field in a0 with defining polynomial x^2 + 2 with a0 = a^3 + a
  To:   Number Field in a with defining polynomial x^4 + 1
  Defn: a0 |--> a^3 + a)
```

You can use the **polred** option to get a simpler defining polynomial:
sage: K.<a> = NumberField(x^5 - 5*x^2 - 3)
sage: G = K.galois_group(); G
Galois group 5T2 (5:2) with order 10 of x^5 - 5*x^2 - 3
sage: sigma, tau = G.gens()
sage: H = G.subgroup([tau])
sage: H.fixed_field(polred=False)
(Number Field in a0 with defining polynomial x^2 + 84375
 with a0 = 5*ac^5 + 25*ac^3,
 Ring morphism:
 From: Number Field in a0 with defining polynomial x^2 + 84375
 with a0 = 5*ac^5 + 25*ac^3
 To: Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + ...
 \rightarrow 3375
 Defn: a0 |--> 5*ac^5 + 25*ac^3)
sage: H.fixed_field(polred=True)
(Number Field in a0 with defining polynomial x^2 - x + 4
 with a0 = -1/30*ac^5 - 1/6*ac^3 + 1/2,
 Ring morphism:
 From: Number Field in a0 with defining polynomial x^2 - x + 4
 with a0 = -1/30*ac^5 - 1/6*ac^3 + 1/2
 To: Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + ...
 \rightarrow 3375
 Defn: a0 |--> -1/30*ac^5 - 1/6*ac^3 + 1/2)
sage: G.splitting_field()
Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + 3375

```
>>> from sage.all import *

K = NumberField(x**Integer(5) - Integer(5)*x**Integer(2) - Integer(3),
                names=('a',)); (a,)=K._first_ngens(1)
>>> G = K.galois_group(); G
Galois group 5T2 (5:2) with order 10 of x^5 - 5*x^2 - 3
>>> sigma, tau = G.gens()
>>> H = G.subgroup([tau])
>>> H.fixed_field(polred=False)
(Number Field in a0 with defining polynomial x^2 + 84375
 with a0 = 5*ac^5 + 25*ac^3,
 Ring morphism:
 From: Number Field in a0 with defining polynomial x^2 + 84375
 with a0 = 5*ac^5 + 25*ac^3
 To: Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + ...
 \rightarrow 3375
 Defn: a0 |--> 5*ac^5 + 25*ac^3)
>>> H.fixed_field(polred=True)
(Number Field in a0 with defining polynomial x^2 - x + 4
 with a0 = -1/30*ac^5 - 1/6*ac^3 + 1/2,
 Ring morphism:
 From: Number Field in a0 with defining polynomial x^2 - x + 4
 with a0 = -1/30*ac^5 - 1/6*ac^3 + 1/2
 To: Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + ...
 \rightarrow 3375
 Defn: a0 |--> -1/30*ac^5 - 1/6*ac^3 + 1/2)
>>> G.splitting_field()
Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + 3375
```

An embedding is returned also if the subgroup is trivial (Issue #26817):
sage: H = G.subgroup([])
sage: H.fixed_field()
(\text{Number Field in } a \text{ with defining polynomial } x^{10} + 10 x^8 + 25 x^6 + 3375, \\
\text{Identity endomorphism of} \\
\text{Number Field in } a \text{ with defining polynomial } x^{10} + 10 x^8 + 25 x^6 + 3375)

>>> from sage.all import *
>>> H = G.subgroup([])
>>> H.fixed_field()
(\text{Number Field in } a \text{ with defining polynomial } x^{10} + 10 x^8 + 25 x^6 + 3375, \\
\text{Identity endomorphism of} \\
\text{Number Field in } a \text{ with defining polynomial } x^{10} + 10 x^8 + 25 x^6 + 3375)

class \text{sage.rings.number_field.galois_group.GaloisGroup_v1}(\text{group, number_field})
Bases: \text{SageObject}

A wrapper around a class representing an abstract transitive group.

This is just a fairly minimal object at present. To get the underlying group, do \text{G.group()}, and to get the corresponding number field do \text{G.number_field()}. For a more sophisticated interface use the \text{type=None} option.

EXAMPLES:

sage: # needs sage.symbolic
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: K = QQ[2^(1/3)]
sage: pK = K.absolute_polynomial()
sage: G = GaloisGroup_v1(pK.galois_group(pari_group=True), K); G
...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://github.com/sagemath/sage/issues/28782 for details.
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the 
Number Field in a with defining polynomial \(x^3 - 2\) with \(a = 1.259921049894873\)
sage: G.order()
6
sage: G.group()
PARI group [6, -1, 2, "S3"] of degree 3
sage: G.number_field()
Number Field in a with defining polynomial \(x^3 - 2\) with \(a = 1.259921049894873\)

>>> from sage.all import *
>>> # needs sage.symbolic
>>> from sage.rings.number_field.galois_group import GaloisGroup_v1
>>> K = QQ[Integer(2)**(Integer(1)/Integer(3))]
>>> pK = K.absolute_polynomial()
>>> G = GaloisGroup_v1(pK.galois_group(pari_group=True), K); G
...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://github.com/sagemath/sage/issues/28782 for details.
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the 
Number Field in a with defining polynomial \(x^3 - 2\) with \(a = 1.259921049894873\)
>>> G.order()
6
>>> G.group()
PARI group [6, -1, 2, "S3"] of degree 3
>>> G.number_field()
Number Field in a with defining polynomial \(x^3 - 2\) with \(a = 1.259921049894873\)

1.5. Galois groups of number fields

255
Return the underlying abstract group.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: x = polygen(ZZ, 'x')
sage: K = NumberField(x^3 + 2*x + 2, 'theta')
sage: G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True), K)
...
DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://github.com/sagemath/sage/issues/28782 for details.
sage: H = G.group(); H
PARI group [6, -1, 2, "S3"] of degree 3
sage: P = H.permutation_group(); P
Transitive group number 2 of degree 3
sage: sorted(P)
[(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.galois_group import GaloisGroup_v1
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x^6 + 2, t)
>>> G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True), K); G
...
DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://github.com/sagemath/sage/issues/28782 for details.
>>> H = G.group(); H
PARI group [6, -1, 3, "D(6) = S(3)[x]2"] of degree 6 of the
Number Field in t with defining polynomial x^6 + 2
>>> sorted(P)
[(), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
```

`number_field()`

Return the number field of which this is the Galois group.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: x = polygen(ZZ, 'x')
sage: K = NumberField(x^6 + 2, 't')
sage: G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True), K);
...
DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://github.com/sagemath/sage/issues/28782 for details.
Galois group PARI group [12, -1, 3, "D(6) = S(3)[x]2"] of degree 6 of the
Number Field in t with defining polynomial x^6 + 2
sage: G.number_field()
Number Field in t with defining polynomial x^6 + 2
```

(continues on next page)
See https://github.com/sagemath/sage/issues/28782 for details.

Galois group PARI group \([12, -1, 3, "D(6) = S(3)[x]2"]\) of degree 6 of the
Number Field in \(t\) with defining polynomial \(x^6 + 2\)

>>> G.number_field()
Number Field in \(t\) with defining polynomial \(x^6 + 2\)

**order()**

Return the order of this Galois group.

**EXAMPLES:**

```
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: x = polygen(ZZ, 'x')
  ...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
  See https://github.com/sagemath/sage/issues/28782 for details.
```

Galois group PARI group \([20, -1, 3, "F(5) = 5:4"]\) of degree 5 of the
Number Field in \(\theta_1\) with defining polynomial \(x^5 + 2\)

```
sage: G.order()
20
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.galois_group import GaloisGroup_v1
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(5) + Integer(2), 'theta_1')
>>> G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True), K); G
  ...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
  See https://github.com/sagemath/sage/issues/28782 for details.
```

Galois group PARI group \([20, -1, 3, "F(5) = 5:4"]\) of degree 5 of the
Number Field in \(\theta_1\) with defining polynomial \(x^5 + 2\)

```python
>>> G.order()
20
```

```
class sage.rings.number_field.galois_group.GaloisGroup_v2(number_field, algorithm='pari', names=None, gc_numbering=None, _type=None)
  
Bases: GaloisGroup_perm

The Galois group of an (absolute) number field.

**Note:** We define the Galois group of a non-normal field \(K\) to be the Galois group of its Galois closure \(L\), and elements are stored as permutations of the roots of the defining polynomial of \(L\), *not* as permutations of the roots (in \(L\)) of the defining polynomial of \(K\). The latter would probably be preferable, but is harder to implement. Thus the permutation group that is returned is always simply-transitive.

The ‘arithmetical’ features (decomposition and ramification groups, Artin symbols etc) are only available for Galois fields.

**EXAMPLES:**
Subgroups can be specified using generators (Issue #26816):

```python
sage: K.<a> = NumberField(x^6 - 6*x^4 + 9*x^2 + 23)
sage: G = K.galois_group()
sage: list(G)
[(), (1,2,3)(4,5,6), (1,3,2)(4,6,5), (1,4)(2,6)(3,5), (1,5)(2,4)(3,6), (1,6)(2,5)(3,4)]
sage: g = G[1]
sage: h = G[3]
sage: list(G.subgroup([g]))
[(), (1,2,3)(4,5,6), (1,3,2)(4,6,5)]
sage: list(G.subgroup([h]))
[(), (1,4)(2,6)(3,5)]
sage: sorted(G.subgroup([g,h])) == sorted(G)
True
```
Element
alias of GaloisGroupElement

Subgroup
alias of GaloisGroup_subgroup

artin_symbol(P)

Return the Artin symbol \( \left( \frac{K/\mathcal{P}}{\mathcal{P}} \right) \), where \( K \) is the number field of \( \text{self} \), and \( \mathcal{P} \) is an unramified prime ideal. This is the unique element \( s \) of the decomposition group of \( \mathcal{P} \) such that \( s(x) = x^p \mod \mathcal{P} \), where \( p \) is the residue characteristic of \( \mathcal{P} \).

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<b> = NumberField(x^4 - 2*x^2 + 2, 'a').galois_closure()
sage: G = K.galois_group()
sage: [G.artin_symbol(P) for P in K.primes_above(7)]
[(1,4)(2,3)(5,8)(6,7), (1,4)(2,3)(5,8)(6,7),
 (1,5)(2,6)(3,7)(4,8), (1,5)(2,6)(3,7)(4,8)]
sage: G.artin_symbol(17)
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not prime
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(2)*x**Integer(2) + Integer(2), 'a').galois_closure(names=('b',)); (b,) = K._first_ngens(1)
>>> G = K.galois_group()
>>> [G.artin_symbol(P) for P in K.primes_above(Integer(7))]
[(1,4)(2,3)(5,8)(6,7), (1,4)(2,3)(5,8)(6,7),
 (1,5)(2,6)(3,7)(4,8), (1,5)(2,6)(3,7)(4,8)]
>>> G.artin_symbol(Integer(17))
Traceback (most recent call last):
...
ValueError: Fractional ideal (...) is ramified
```

complex_conjugation (P=None)

Return the unique element of \( \text{self} \) corresponding to complex conjugation, for a specified embedding \( P \) into the complex numbers. If \( P \) is not specified, use the “standard” embedding, whenever that is well-defined.

EXAMPLES:
An example where the field is not CM, so complex conjugation really depends on the choice of embedding:

```python
sage: x = polygen(ZZ, 'x')
sage: L = NumberField(x^6 + 40*x^3 + 1372, 'a')
sage: G = L.galois_group()
sage: [G.complex_conjugation(x) for x in L.places()]
[(1,3)(2,6)(4,5), (1,5)(2,4)(3,6), (1,2)(3,4)(5,6)]
```

**decomposition_group** \((P)\)

Decomposition group of a prime ideal \(P\), i.e., the subgroup of elements that map \(P\) to itself. This is the same as the Galois group of the extension of local fields obtained by completing at \(P\).

This function will raise an error if \(P\) is not prime or the given number field is not Galois.

\(P\) can also be an infinite prime, i.e., an embedding into \(\mathbb{R}\) or \(\mathbb{C}\).

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 - 2*x^2 + 2, 'b').galois_closure()
sage: P = K.ideal([17, a^2])
sage: G = K.galois_group()
sage: G.decomposition_group(P)
Subgroup generated by [(1,8)(2,7)(3,6)(4,5)] of (Galois group 8T4 ([4]2) with order 8 of x^8 - 20*x^6 + 104*x^4 - 40*x^2 +\
˓→1156)
sage: G.decomposition_group(P^2)
Traceback (most recent call last):
...
ValueError: Fractional ideal (...) is not a prime ideal
sage: G.decomposition_group(17)
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not a prime ideal
```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(2)*x**Integer(2) + Integer(2), 'b')
>>> K.galois_closure(names=('a',)); (a,) = K._first_ngens(1)
>>> P = K.ideal([Integer(17), a**Integer(2)])
>>> G = K.galois_group()
>>> G.decomposition_group(P)
Subgroup generated by [(1,8)(2,7)(3,6)(4,5)] of Galois group 8T4 ([4]2) with order 8 of x^8 - 20*x^6 + 104*x^4 - 40*x^2 +...
>>> G.decomposition_group(P**Integer(2))
Traceback (most recent call last):
...
ValueError: Fractional ideal (...) is not a prime ideal
>>> G.decomposition_group(Integer(17))
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not a prime ideal

An example with an infinite place:

sage: x = polygen(ZZ, 'x')
sage: L.<b> = NumberField(x^3 - 2,'a').galois_closure(); G = L.galois_group()
sage: x = L.places()[0]
sage: G.decomposition_group(x).order()
2

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> L = NumberField(x**Integer(3) - Integer(2),'a').galois_closure(names=('b',))
>>> (b,) = L._first_ngens(1); G = L.galois_group()
>>> x = L.places()[Integer(0)]
>>> G.decomposition_group(x).order()
2

\textbf{easy_order} \ (algorithm=None)

Return the order of this Galois group if it's quick to compute.

\textbf{EXAMPLES:}

sage: R.<x> = ZZ[]
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group()
sage: G.easy_order()
6
sage: x = polygen(ZZ, 'x')
sage: L.<b> = NumberField(x^72 + 2*x + 2)
sage: H = L.galois_group()
sage: H.easy_order()

>>> from sage.all import *
>>> R = ZZ['x']; (x,) = R._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) + Integer(2)*x + Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> G = K.galois_group()

(continues on next page)
G.eval_order()
6
>>> x = polygen(ZZ, 'x')
>>> L = NumberField(x**72 + Integer(2)*x + Integer(2), names=('b',));
    (b,) = L._first_ngens(1)
>>> H = L.galois_group()
>>> H.eval_order()

GaloisGroup_v1 is being deprecated, this provides public access to the PARI/GAP group in order to keep all aspects of that API.

EXAMPLES:

sage: R.<x> = ZZ[]
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group(type="pari")
...DeprecationWarning: the different Galois types have been merged into one...
    ~class
See https://github.com/sagemath/sage/issues/28782 for details.
sage: G.eval_group()
...DeprecationWarning: the group method is deprecated;
you can use _pol_galgp if you really need it
See https://github.com/sagemath/sage/issues/28782 for details.
PARI group [6, -1, 2, "S3"] of degree 3

inertia_group(P)

Return the inertia group of the prime $P$, i.e., the group of elements acting trivially modulo $P$. This is just the 0th ramification group of $P$.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<b> = NumberField(x^2 - 3, 'a')
sage: G = K.galois_group()  # type="pari"
...DeprecationWarning: the different Galois types have been merged into one...
    ~class
See https://github.com/sagemath/sage/issues/28782 for details.
sage: G.eval_inertia_group(K.primes_above(2)[0])
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 - 3)
sage: G.eval_inertia_group(K.primes_above(5)[0])
Subgroup generated by [] of (Galois group 2T1 (S2) with order 2 of x^2 - 3)
is_galois()  
Whether the underlying number field is Galois.

EXAMPLES:

```sage
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(3), 'a', names=('b',)); (b,) = K._first_ngens(1)
>>> G = K.galois_group()
>>> G.inertia_group(K.primes_above(Integer(2))[0])
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 - 3)

```  

```sage
>>> G.inertia_group(K.primes_above(Integer(5))[0])
Subgroup generated by [()] of (Galois group 2T1 (S2) with order 2 of x^2 - 3)
```  

list()  
List of the elements of self.

EXAMPLES:

```sage
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - x + Integer(1), 'a').galois_group().list()
[((), (1,2,3), (1,3,2))]
```  

```sage
>>> K = NumberField(x**Integer(3) - x + Integer(1), 'a').galois_group().list()
[((), (1,2,3), (1,3,2))]
```  

timeout

number_field()  
The ambient number field.

EXAMPLES:

```sage
sage: K.galois_group(names='b').number_field() is K
True
```
```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - x + Integer(1), 'a')
>>> K.galois_group(names='b').number_field() is K
True
```

**order** *(algorithm=None, recompute=False)*

Return the order of this Galois group.

**EXAMPLES:**

```python
sage: R.<x> = ZZ[]
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group()
sage: G.order()
6
```

```python
>>> from sage.all import *
>>> R = ZZ['x']; (x,) = R._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) + Integer(2)*x + Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> G = K.galois_group()
>>> G.order()
6
```

**pari_label()**

Return the label assigned by PARI for this Galois group, an attempt at giving a human readable description of the group.

**EXAMPLES:**

```python
sage: R.<x> = ZZ[]
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^8 - x^5 + x^4 - x^3 + 1)
sage: G = K.galois_group()
sage: G.transitive_label()
'8T44'
sage: G.pari_label()
'[2^4]S(4)'
```

```python
from sage.all import *
>>> R = ZZ['x']; (x,) = R._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(8) - x**Integer(5) + x**Integer(4) - x**Integer(3) + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> G = K.galois_group()
>>> G.transitive_label()
'8T44'
>>> G.pari_label()
'[2^4]S(4)'
```

**ramification_breaks(P)**

Return the set of ramification breaks of the prime ideal $P$, i.e., the set of indices $i$ such that the ramification group $G_{i+1} \neq G_i$. This is only defined for Galois fields.
EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<b> = NumberField(x^8 - 20*x^6 + 104*x^4 - 40*x^2 + 1156)
sage: G = K.galois_group()
sage: P = K.primes_above(2)[0]
sage: G.ramification_breaks(P) {1, 3, 5}
sage: min(G.ramification_group(P, i).order() / G.ramification_group(P, i + 1).order() for i in G.ramification_breaks(P)) 2
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(8) - Integer(20)*x**Integer(6) +...
˓→Integer(104)*x**Integer(4) - Integer(40)*x**Integer(2) + Integer(1156),...
˓→names=('b',)); (b,) = K._first_ngens(1)
>>> G = K.galois_group()
>>> P = K.primes_above(Integer(2))[Integer(0)]
>>> G.ramification_breaks(P) {1, 3, 5}
>>> min(G.ramification_group(P, i).order() / G.ramification_group(P, i + Integer(1)).order() for i in G.ramification_breaks(P)) 2
```

```
ramification_group(P, v)
```

Return the \( v \)-th ramification group of `self` for the prime \( P \), i.e., the set of elements \( s \) of `self` such that \( s \) acts trivially modulo \( P^{(v+1)} \). This is only defined for Galois fields.

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: K.<b> = NumberField(x^3 - 3, 'a').galois_closure()
sage: G=K.galois_group()
sage: P = K.primes_above(3)[0]
sage: G.ramification_group(P, 3)
Subgroup generated by [(1,2,4)(3,5,6)] of (Galois group 6T2 ([3]2) with order 6 of x^6 + 243)
sage: G.ramification_group(P, 5)
Subgroup generated by [()] of (Galois group 6T2 ([3]2) with order 6 of x^6 + 243)
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3), 'a').galois_closure(names=('b',)); (b,) = K._first_ngens(1)
>>> G=K.galois_group()
>>> P = K.primes_above(Integer(3))[Integer(0)]
>>> G.ramification_group(P, Integer(3))
Subgroup generated by [(1,2,4)(3,5,6)] of (Galois group 6T2 ([3]2) with order 6 of x^6 + 243)
>>> G.ramification_group(P, Integer(5))
Subgroup generated by [()] of (Galois group 6T2 ([3]2) with order 6 of x^6 + 243)
```

```
signature()
```

1.5. Galois groups of number fields 265
Return $1$ if contained in the alternating group, $-1$ otherwise.

EXAMPLES:

```
sage: R.<x> = ZZ[]
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 2)
sage: K.galois_group().signature()
-1
sage: K.<a> = NumberField(x^3 - 3*x - 1)
sage: K.galois_group().signature()
1
```

```python
>>> from sage.all import *
>>> R = ZZ['x']; (x,) = R._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> K.galois_group().signature()
-1
>>> K = NumberField(x**Integer(3) - Integer(3)*x - Integer(1), names=('a',));␣
˓→(a,) = K._first_ngens(1)
>>> K.galois_group().signature()
1
```

**transitive_number** (*algorithm=None, recompute=False*)

Regardless of the value of gc_numbering, give the transitive number for the action on the roots of the defining polynomial of the original number field, not the Galois closure.

**INPUT:**

- *algorithm*—string, specify the algorithm to be used
- *recompute*—boolean, whether to recompute the result even if known by another algorithm

**EXAMPLES:**

```
sage: R.<x> = ZZ[]
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group()
sage: G.transitive_number()
2
```

```python
>>> from sage.all import *
>>> R = ZZ['x']; (x,) = R._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) + Integer(2)*x + Integer(2), names=('a',));␣
˓→(a,) = K._first_ngens(1)
>>> G = K.galois_group()
>>> G.transitive_number()
2
```

(continues on next page)
unrank\( (i) \)

Return the \( i \)-th element of \( \text{self} \).

INPUT:

- \( i \) – integer between 0 and \( n - 1 \) where \( n \) is the cardinality of this set

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: G = NumberField(x^3 - 3*x + 1, 'a').galois_group()
sage: [G.unrank(i) for i in range(G.cardinality())]
[(1,2,3), (1,3,2)]
```

```python
from sage.all import *

sage: x = polygen(ZZ, 'x')
sage: G = NumberField(x**Integer(3) - Integer(3)*x + Integer(1), 'a').galois_group()
sage: [G.unrank(i) for i in range(G.cardinality())]
[(), (1,2,3), (1,3,2)]
```
2.1 Elements of number fields (implemented using NTL)

AUTHORS:
- William Stein: initial version
- Joel B. Mohler (2007-03-09): reimplemented in Cython
- William Stein (2007-09-04): added doctests
- Robert Bradshaw (2007-09-15): specialized classes for relative and absolute elements
- John Cremona (2009-05-15): added support for local and global logarithmic heights
- Robert Harron (2012-08): conjugate() now works for all fields contained in CM fields

```python
class sage.rings.number_field.number_field_element.CoordinateFunction(alpha, W, to_V):
    Bases: object

    This class provides a callable object which expresses elements in terms of powers of a fixed field generator α.

    EXAMPLES:

    sage: x = polygen(ZZ, x)
    sage: K.<a> = NumberField(x^2 + x + 3)
    sage: f = (a + 1).coordinates_in_terms_of_powers(); f
    Coordinate function that writes elements in terms of the powers of a + 1
    sage: f.__class__
    <class 'sage.rings.number_field.number_field_element.CoordinateFunction'>
    sage: f(a)
    [-1, 1]
    sage: f == loads(dumps(f))
    True
```

(continues on next page)
>>> f == loads(dumps(f))
True

alpha()

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2)
sage: (a + 2).coordinates_in_terms_of_powers().alpha()
a + 2
```

```python
>>> from sage.all import *
```  
```python
>>> x = polygen(ZZ, 'x')
```  
```python
>>> K = NumberField(x**Integer(3) + Integer(2), names=('a',)); (a,) = K._  
˓→first_ngens(1)
```  
```python
>>> (a + Integer(2)).coordinates_in_terms_of_powers().alpha()
a + 2
```

class sage.rings.number_field.number_field_element.NumberFieldElement

Bases: NumberFieldElement_base

An element of a number field.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^3 + x + 1)
sage: a^3
-a - 1
```

```python
>>> from sage.all import *
```  
```python
>>> x = polygen(ZZ, 'x')
```  
```python
>>> k = NumberField(x**Integer(3) + x + Integer(1), names=('a',)); (a,) = k._  
˓→first_ngens(1)
```  
```python
>>> a**Integer(3)
-a - 1
```

abs(prec=None, i=None)

Return the absolute value of this element.

If $i$ is provided, then the absolute value of the $i$-th embedding is given.

Otherwise, if the number field has a coercion embedding into $\mathbb{R}$, the corresponding absolute value is returned as an element of the same field (unless $\text{prec}$ is given). Otherwise, if it has a coercion embedding into $\mathbb{C}$, then the corresponding absolute value is returned. Finally, if there is no coercion embedding, $i$ defaults to 0.

For the computation, the complex field with $\text{prec}$ bits of precision is used, defaulting to 53 bits of precision if $\text{prec}$ is not provided. The result is in the corresponding real field.

INPUT:

- $\text{prec}$ – (default: None) integer bits of precision
- $i$ – (default: None) integer, which embedding to use

EXAMPLES:
sage: z = CyclotomicField(7).gen()
sage: abs(z)
1.00000000000000
sage: abs(z^2 + 17*z - 3)
16.0604426799931
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 17)
sage: abs(a)
2.57128159065824
sage: a.abs(prec=100)
2.571281590658235351872087
sage: a.abs(prec=100, i=1)
2.571281590658235351872087
sage: a.abs(100, 2)
2.571281590658235351872087

Here's one where the absolute value depends on the embedding:

sage: K.<b> = NumberField(x^2 - 2)
sage: a = 1 + b
sage: a.abs(i=0)
0.414213562373095
sage: a.abs(i=1)
2.41421356237309

Check that Issue #16147 is fixed:

sage: x = polygen(ZZ)
sage: f = x^3 - x - 1
sage: beta = f.complex_roots()[0]; beta

(continues on next page)
1.32471795724475
\( \text{sage: } K.<b> = NumberField(f, embedding=\textit{beta}) \)
\( \text{sage: } b.abs() \)
1.32471795724475

\[ \text{from sage.all import } * \]
\[ \text{from sage.all import } * \]
\[ \text{x = polygen(ZZ)} \]
\[ \text{f = x**Integer(3) - x - Integer(1)} \]
\[ \text{beta = f\.complex_roots()[0]}; \beta \]
1.32471795724475
\[ \text{K = NumberField(f, embedding=\beta, names=('b',)); (b,) = K\.first_ngens(1)} \]
\[ \text{b.abs()} \]
1.32471795724475

Check that for fields with real coercion embeddings, absolute values are in the same field (Issue #21105):

\[ \text{sage: } x = \text{polygen(ZZ)} \]
\[ \text{sage: } f = x**3 - x - 1 \]
\[ \text{sage: } K.<b> = \text{NumberField(f, embedding=1.3)} \]
\[ \text{sage: } b.abs() \]
\[ b \]

\[ \text{from sage.all import } * \]
\[ \text{from sage.all import } * \]
\[ \text{x = polygen(ZZ)} \]
\[ \text{f = x^3 - x - 1} \]
\[ \text{K.<b> = NumberField(f, embedding=1.3)} \]
\[ \text{b.abs()} \]
\[ b \]

However, if a specific embedding is requested, the behavior reverts to that of number fields without a coercion embedding into \( \mathbb{R} \):

\[ \text{sage: } b.abs(i=2) \]
1.32471795724475

\[ \text{from sage.all import } * \]
\[ \text{from sage.all import } * \]
\[ \text{b.abs(i=Integer(2))} \]
1.32471795724475

Also, if a precision is requested explicitly, the behavior reverts to that of number fields without a coercion embedding into \( \mathbb{R} \):

\[ \text{sage: } b.abs(prec=Integer(53)) \]
1.32471795724475

\[ \text{from sage.all import } * \]
\[ \text{from sage.all import } * \]
\[ \text{b.abs(prec=Integer(53))} \]
1.32471795724475

**abs_non_arch** \((P, \text{prec}=None)\)

Return the non-archimedean absolute value of this element with respect to the prime \( P \), to the given precision.

**INPUT:**

- \( P \) – a prime ideal of the parent of \textit{self}
• prec (int) – desired floating point precision (default: default RealField precision).

OUTPUT:

(real) the non-archimedean absolute value of this element with respect to the prime \( P \), to the given precision. This is the normalised absolute value, so that the underlying prime number \( p \) has absolute value \( 1/p \).

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 5)
sage: [1/K(2).abs_non_arch(P) for P in K.primes_above(2)]
[2.00000000000000]
sage: [1/K(3).abs_non_arch(P) for P in K.primes_above(3)]
[3.00000000000000, 3.00000000000000]
sage: [1/K(5).abs_non_arch(P) for P in K.primes_above(5)]
[5.00000000000000]
```

A relative example:

```
sage: L.<b> = K.extension(x^2 - 5)
sage: [b.abs_non_arch(P) for P in L.primes_above(b)]
[0.447213595499958, 0.447213595499958]
```

**absolute_different()**

Return the absolute different of this element.

This means the different with respect to the base field \( \mathbb{Q} \).

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: a.absolute_different()
0
```

(continues on next page)
>>> K = NumberFieldTower([x**Integer(2) - Integer(17), x**Integer(3) - Integer(2)], names=('a',)); (a,) = K._first_ngens(1)
>>> a.absolute_different()
0

See also:

\texttt{different()}

\texttt{absolute_norm()}

Return the absolute norm of this number field element.

**EXAMPLES:**

\begin{verbatim}
sage: K1.<a1> = CyclotomicField(11)
sage: x = polygen(ZZ, 'x')
sage: K2.<a2> = K1.extension(x^2 - 3)
sage: K3.<a3> = K2.extension(x^2 + 1)
>>> (a1 + a2 + a3).absolute_norm()
1353244757701
>>> QQ(7/5).absolute_norm()
7/5
\end{verbatim}

\texttt{additive_order()}

Return the additive order of this element (i.e., infinity if \texttt{self} \neq 0 and 1 if \texttt{self} == 0)

**EXAMPLES:**

\begin{verbatim}
>>> from sage.all import *
>>> K1 = CyclotomicField(Integer(11), names=('a1',)); (a1,) = K1._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K2 = K1.extension(x**Integer(2) - Integer(3), names=('a2',)); (a2,) = K2._first_ngens(1)
>>> K3 = K2.extension(x**Integer(2) + Integer(1), names=('a3',)); (a3,) = K3._first_ngens(1)
>>> (a1 + a2 + a3).additive_order()
1353244757701
>>> QQ(Integer(7)/Integer(5)).additive_order()
7/5
\end{verbatim}
ceil()

Return the ceiling of this number field element.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: p = x**7 - 5*x**2 + x + 1
sage: a_AA = AA.polynomial_root(p, RIF(1,2))
sage: K.<a> = NumberField(p, embedding=a_AA)
sage: b = a**5 + a/2 - 1/7
sage: RR(b)
4.1344473767055
sage: b.ceil()
5
```

This function always succeeds even if a tremendous precision is needed:

```
sage: c = b - 5065701199253/1225243417356 + 2
sage: c.ceil()
3
sage: RIF(c).unique_ceil()
Traceback (most recent call last):
  ...
ValueError: interval does not have a unique ceil
```

```
sage: c = b - Integer(5065701199253)/Integer(1225243417356) + Integer(2)
sage: c.ceil()
3
sage: RIF(c).unique_ceil()
Traceback (most recent call last):
  ...
ValueError: interval does not have a unique ceil
```

If the number field is not embedded, this function is valid only if the element is rational:

```
sage: p = x**5 - 3
sage: K.<a> = NumberField(p)
sage: K(2/3).ceil()
```

(continues on next page)
1
sage: a.ceil()
Traceback (most recent call last):
... 
TypeError: ceil not uniquely defined since no real embedding is specified

>>> from sage.all import *
... p = x**Integer(3) - Integer(3)
... K = NumberField(p, names=('a',)); (a,) = K._first_ngens(1)
... K(Integer(2)/Integer(3)).ceil()
1
... a.ceil()
Traceback (most recent call last):
... 
TypeError: ceil not uniquely defined since no real embedding is specified

**charpoly** *(var='x')*

Return the characteristic polynomial of this number field element.

**EXAMPLES:**

```sage
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^3 - 2)
sage: a.charpoly()
x^3 - 3*x^2 + 3*x - 1
```

```sage
... from sage.all import *
... x = polygen(ZZ, 'x')
... K = NumberField(x**Integer(3) + Integer(7), names=('a',)); (a,) = K._first_ngens(1)
... a.charpoly()
x^3 + 7
```

**complex_embedding** *(prec=53, i=0)*

Return the *i*-th embedding of *self* in the complex numbers, to the given precision.

**EXAMPLES:**

```sage
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^3 - 2)
sage: a.complex_embedding()
-0.629960524947437 - 1.09112363597172*I
sage: a.complex_embedding(10)
-0.63 - 1.1*I
```

```sage
... from sage.all import *
... x = polygen(ZZ, 'x')
... K = NumberField(x**Integer(3) + Integer(7), names=('a',)); (a,) = K._first_ngens(1)
... a.complex_embedding()
x^3 + 7
```

```sage
... K(Integer(1)).charpoly()
x^3 - 3*x^2 + 3*x - 1
```

```sage
... a.complex_embedding(20, 1)
-0.62996 + 1.0911*I
```

```sage
... a.complex_embedding(20, 2)
1.2599
```
The code snippet demonstrates the use of the `NumberField` class in SageMath to work with algebraic numbers and number fields. The snippet shows how to create a number field, access its complex embeddings, and compute the conjugate of an element in the field.

### complex_embeddings (prec=53)

Return the images of this element in the floating point complex numbers, to the given bits of precision.

**INPUT:**
- `prec` – integer (default: 53) bits of precision

**EXAMPLES:**

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = k._first_ngens(1)
>>> a.complex_embeddings()
[-0.629960524947437 - 1.09112363597172*I,
 -0.629960524947437 + 1.09112363597172*I,
  1.25992104989487]
```

```python
>>> a.complex_embeddings(Integer(10))
[-0.63 - 1.1*I, -0.63 + 1.1*I, 1.3]
```

```python
>>> a.complex_embeddings(Integer(100))
[-0.62996052494743658238360530364 - 1.0911236359717214035600726142*I,
 -0.62996052494743658238360530364 + 1.0911236359717214035600726142*I,
  1.2599210498948731647672106073]
```

### conjugate()

Return the complex conjugate of the number field element.

This is only well-defined for fields contained in CM fields (i.e. for totally real fields and CM fields). Recall that a CM field is a totally imaginary quadratic extension of a totally real field. For other fields, a `ValueError` is raised.
EXAMPLES:

```
sage: k.<I> = QuadraticField(-1)
sage: I.conjugate()
-I
sage: (I/(1+I)).conjugate()
-1/2*I + 1/2
sage: z6 = CyclotomicField(6).gen(0)
sage: (2*z6).conjugate()
-2*zeta6 + 2
```

```
>>> from sage.all import *
>>> k = QuadraticField(Integer(-1), names=(I,)); (I,) = k._first_ngens(1)
>>> I.conjugate()
-I
>>> (I/(Integer(1)+I)).conjugate()
-1/2*I + 1/2
>>> z6 = CyclotomicField(Integer(6)).gen(Integer(0))
>>> (Integer(2)*z6).conjugate()
-2*zeta6 + 2
```

The following example now works.

```
sage: x = polygen(ZZ, 'x')
sage: F.<b> = NumberField(x^2 - 2)
sage: K.<j> = F.extension(x^2 + 1)
sage: j.conjugate()
-j
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> F = NumberField(x**Integer(2) - Integer(2), names=('b',)); (b,) = F._
―first_ngens(1)
>>> K = F.extension(x**Integer(2) + Integer(1), names=('j',)); (j,) = K._
―first_ngens(1)
>>> j.conjugate()
-j
```

Raise a `ValueError` if the field is not contained in a CM field.

```
sage: K.<b> = NumberField(x^3 - 2)
sage: b.conjugate()
Traceback (most recent call last):
...
ValueError: Complex conjugation is only well-defined for fields contained in CM fields.
```

```
>>> from sage.all import *
>>> K = NumberField(x**Integer(3) - Integer(2), names=('b',)); (b,) = K._
―first_ngens(1)
>>> b.conjugate()
Traceback (most recent call last):
...
ValueError: Complex conjugation is only well-defined for fields contained in CM fields.
```

An example of a non-quadratic totally real field.
### 2.1. Elements of number fields (implemented using NTL)

#### \texttt{sage}

\begin{verbatim}
sage: F.<a> = NumberField(x^4 + x^3 - 3*x^2 - x + 1)
sage: a.conjugate()
a
>>> from sage.all import *
>>> F = NumberField(x**Integer(4) + x**Integer(3) - Integer(3)*x**Integer(2) -
→ x + Integer(1), names=('a',)); (a,) = F._first_ngens(1)
>>> a.conjugate()
a
An example of a non-cyclotomic CM field.

\begin{verbatim}
sage: K.<a> = NumberField(x^4 - x^3 + 2*x^2 + x + 1)
sage: a.conjugate()
-1/2*a^3 - a - 1/2
sage: (2*a^2 - 1).conjugate()
a^3 - 2*a^2 - 2

>>> from sage.all import *
>>> K = NumberField(x**Integer(4) - x**Integer(3) + Integer(2)*x**Integer(2)
→ + x + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> a.conjugate()
-1/2*a^3 - a - 1/2
>>> (Integer(2)*a**Integer(2) - Integer(1)).conjugate()
a^3 - 2*a^2 - 2
\end{verbatim}

coordinates\_in\_terms\_of\_powers()

Let \(\alpha\) be self. Return a callable object (of type \texttt{CoordinateFunction}) that takes any element of the parent of self in \(\mathbb{Q}(\alpha)\) and writes it in terms of the powers of \(\alpha\): \(1, \alpha, \alpha^2, \ldots\).

(NOT CACHED).

EXAMPLES:

This function allows us to write elements of a number field in terms of a different generator without having to construct a whole separate number field.

\begin{verbatim}
sage: y = polygen(QQ,'y'); K.<beta> = NumberField(y^3 - 2); K
Number Field in beta with defining polynomial y^3 - 2
sage: alpha = beta^2 + beta + 1
sage: c = alpha.coordinates_in_terms_of_powers(); c
Coordinate function that writes elements in terms of the powers of beta^2 +
→ beta + 1
sage: c(beta)
[-2, -3, 1]
sage: c(alpha)
[0, 1, 0]
sage: c((1+beta)^5)
[3, 3, 3]
sage: c((1+beta)^10)
[54, 162, 189]

>>> from sage.all import *
>>> y = polygen(QQ,'y'); K = NumberField(y**Integer(3) - Integer(2), names=(
→ 'beta',)); (beta,) = K._first_ngens(1); K
Number Field in beta with defining polynomial y^3 - 2
>>> alpha = beta**Integer(2) + beta + Integer(1)
\end{verbatim}

(continues on next page)
This function works even if `self` only generates a subfield of this number field.

```python
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^6 - 5)
sage: alpha = a^3
sage: c = alpha.coordinates_in_terms_of_powers()
sage: c((2/3)*a^3 - 5/3)
[-5/3, 2/3]
sage: c
Coordinate function that writes elements in terms of the powers of a^3
sage: c(a)
Traceback (most recent call last):
... ArithmeticError: vector is not in free module
```

**denominator()**

Return the denominator of this element, which is by definition the denominator of the corresponding polynomial representation. I.e., elements of number fields are represented as a polynomial (in reduced form) modulo the modulus of the number field, and the denominator is the denominator of this polynomial.

**EXAMPLES:**

```python
sage: K.<z> = CyclotomicField(3)
sage: a = 1/3 + (1/5)*z
sage: a.denominator()
15
```

```python
>>> from sage.all import *
```

```python
>>> K = CyclotomicField(Integer(3), names=('z',)); (z,) = K._first_ngens(1)
```
denominator_ideal()

Return the denominator ideal of this number field element.

The denominator ideal of a number field element \( a \) is the integral ideal consisting of all elements of the ring of integers \( R \) whose product with \( a \) is also in \( R \).

See also:

numerator_ideal()

EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 5)
sage: b = (1+a)/2
sage: b.norm()
3/2
sage: D = b.denominator_ideal(); D
Fractional ideal (2, a + 1)
sage: D.norm()
2
sage: (1/b).denominator_ideal()
Fractional ideal (3, a + 1)
sage: K(0).denominator_ideal()
Fractional ideal (1)
```

```python
from sage.all import *
```

```sage
sage: x = polygen(ZZ, 'x')
sage: K = NumberField(x**Integer(2) + Integer(5), names=('a',)); (a,) = K._˓→first_ngens(1)
sage: b = (Integer(1)+a)/Integer(2)
sage: b.norm()
3/2
sage: D = b.denominator_ideal(); D
Fractional ideal (2, a + 1)
sage: D.norm()
2
sage: (Integer(1)/b).denominator_ideal()
Fractional ideal (3, a + 1)
sage: K(Integer(0)).denominator_ideal()
Fractional ideal (1)
```

descend_mod_power (K='QQ', d=2)

Return a list of elements of the subfield \( K \) equal to \self modulo \( d \)'th powers.

INPUT:

- \( K \) (number field, default \( \mathbb{Q} \)) – a subfield of the parent number field \( L \) of \self
- \( d \) (positive integer, default 2) – an integer at least 2

OUTPUT:

A list, possibly empty, of elements of \( K \) equal to \self modulo \( d \)'th powers, i.e. the preimages of \self under the map \( K^*/(K^*)^d \rightarrow L^*/(L^*)^d \) where \( L \) is the parent of \self. A ValueError is raised if \( K \)
does not embed into $L$.

**ALGORITHM:**

All preimages must lie in the Selmer group $K(S, d)$ for a suitable finite set of primes $S$, which reduces the question to a finite set of possibilities. We may take $S$ to be the set of primes which ramify in $L$ together with those for which the valuation of $\text{self}$ is not divisible by $d$.

**EXAMPLES:**

A relative example:

```python
sage: Qi.<i> = QuadraticField(-1)
sage: K.<zeta> = CyclotomicField(8)
sage: f = Qi.embeddings(K)[0]
sage: a = f(2+3*i) * (2-zeta)^2
sage: a.descend_mod_power(Qi,2)
[-2*i + 3, 3*i + 2]
```

An absolute example:

```python
sage: K.<zeta> = CyclotomicField(8)
sage: K(1).descend_mod_power(QQ,2)
[1, 2, -1, -2]
sage: a = 17 * K._random_nonzero_element()^2
sage: a.descend_mod_power(QQ,2)
[17, 34, -17, -34]
```

**different** ($K$=`None`)

Return the different of this element with respect to the given base field.

The different of an element $a$ in an extension of number fields $L/K$ is defined as $\text{Diff}_{L/K}(a) = f'(a)$ where $f$ is the characteristic polynomial of $a$ over $K$.

**INPUT:**

- $K$ – a subfield (embedding of a subfield) of the parent number field of $\text{self}$. Default: `None`, which will amount to $\text{self}.parent().base_field()$.

**EXAMPLES:**
The optional argument \( K \) can be an embedding of a subfield:

```python
sage: K.<b> = NumberField(x^4 - 2)
sage: (b^2).different()
0
sage: phi = K.base_field().embeddings(K)[0]
sage: b.different(K=phi)
4*b^3
```

Compare the relative different and the absolute different for an element in a relative number field:

```python
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: a.different()
2*a0
sage: a.different(K=QQ)
0
sage: a.absolute_different()
0
```

Observe that for the field extension \( \mathbb{Q}(i)/\mathbb{Q} \), the different of the field extension is the ideal generated by the different of \( i \):

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> a.different()
3*a^2
>>> a.different(K=K)
1
>>> K = NumberField(x**Integer(4) - Integer(2), names=('b',)); (b,) = K._first_ngens(1)
>>> (b**Integer(2)).different()
0
>>> phi = K.base_field().embeddings(K)[Integer(0)]
>>> b.different(K=phi)
4*b^3
```
sage: K.<c> = NumberField(x**2 + 1)
sage: K.different() == K.ideal(c.different())
True

>>> from sage.all import *
>>> K = NumberField(x**Integer(2) + Integer(1), names=('c',)); (c,) = K._
˓→first_ngens(1)
>>> K.different() == K.ideal(c.different())
True

See also:

absolute_different()  sage.rings.number_field.number_field_rel.
NumberField_relative.different()

factor()

Return factorization of this element into prime elements and a unit.

OUTPUT:

(Factorization) If all the prime ideals in the support are principal, the output is a Factorization
as a product of prime elements raised to appropriate powers, with an appropriate unit factor.

Raise ValueError if the factorization of the ideal (self) contains a non-principal prime ideal.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x**2 + 1)
sage: (6*i + 6).factor()
(-i) * (i + 1)^3 * 3

>>> from sage.all import *
>>>
x = polygen(ZZ, 'x')
>>>
K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
˓→first_ngens(1)
>>> (Integer(6)*i + Integer(6)).factor()
(-i) * (i + 1)^3 * 3

In the following example, the class number is 2. If a factorization in prime elements exists, we will find it:

sage: K.<a> = NumberField(x**2 - 10)
sage: factor(169*a + 531)
(-6*a - 19) * (-3*a - 1) * (-2*a + 9)
sage: factor(K(3))
Traceback (most recent call last):
...
ArithmeticError: non-principal ideal in factorization

>>> from sage.all import *
>>> K = NumberField(x**Integer(2) - Integer(10), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> factor(Integer(169)*a + Integer(531))
(-6*a - 19) * (-3*a - 1) * (-2*a + 9)
>>> factor(K(Integer(3)))
Traceback (most recent call last):
...
ArithmeticError: non-principal ideal in factorization
Factorization of 0 is not allowed:

```python
sage: K.<i> = QuadraticField(-1)
sage: K(0).factor()
Traceback (most recent call last):
  ...  
ArithmeticError: factorization of 0 is not defined
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> K(Integer(0)).factor()
Traceback (most recent call last):
  ...  
ArithmeticError: factorization of 0 is not defined
```

**floor()**

Return the floor of this number field element.

**EXAMPLES:**

```python
sage: x = polygen(ZZ)
sage: p = x**7 - 5*x**2 + x + 1
sage: a_AA = AA.polynomial_root(p, RIF(1,2))
sage: K.<a> = NumberField(p, embedding=a_AA)
sage: b = a**5 + a/2 - 1/7
sage: RR(b)
4.13444473767055
sage: b.floor()
4
sage: K(Integer(125)/Integer(7)).floor()
17
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> p = x**Integer(7) - Integer(5)*x**Integer(2) + x + Integer(1)
>>> a_AA = AA.polynomial_root(p, RIF(Integer(1),Integer(2)))
>>> K = NumberField(p, embedding=a_AA, names=('a',)); (a,) = K._first_ngens(1)
>>> b = a**Integer(5) + a/Integer(2) - Integer(1)/Integer(7)
>>> RR(b)
4.1344473767055
>>> b.floor()
4
>>> K(Integer(125)/Integer(7)).floor()
17
```

This function always succeeds even if a tremendous precision is needed:

```python
sage: c = b - 4772404052447/1154303505127 + 2
sage: c.floor()
1
sage: RIF(c).unique_floor()
Traceback (most recent call last):
  ...  
ValueError: interval does not have a unique floor
```
If the number field is not embedded, this function is valid only if the element is rational:

```python
sage: p = x**5 - 3
sage: K.<a> = NumberField(p)
```

```python
sage: K(2/3).floor()
```

```
0
```

```python
sage: a.floor()
```

```
Traceback (most recent call last):
...
TypeError: floor not uniquely defined since no real embedding is specified
```

**galois_conjugates** (*K*)

Return all Gal($\overline{\mathbb{Q}}$/$\mathbb{Q}$)-conjugates of this number field element in the field $K$.

**EXAMPLES:**

In the first example the conjugates are obvious:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 2)
sage: a.galois_conjugates(K)
```

```
[a, -a]
```

```python
sage: K(Integer(3)).galois_conjugates(K)
```

```
[3]
```

In this example the field is not Galois, so we have to pass to an extension to obtain the Galois conjugates.

```python
sage: K.<a> = NumberField(x^3 - 2)
sage: c = a.galois_conjugates(K); c
```

(continues on next page)
There is only one Galois conjugate of $\sqrt[3]{2}$ in $\mathbb{Q}(\sqrt[3]{2})$.

```
sage: a.galois_conjugates(K)
[a]
```

Galois conjugates of $\sqrt[3]{2}$ in the field $\mathbb{Q}(\zeta_3, \sqrt[3]{2})$:

```
sage: L.<a> = CyclotomicField(3).extension(x^3 - 2)
sage: a.galois_conjugates(L)
[a, (-zeta3 - 1)*a, zeta3*a]
```

```
from sage.all import *
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> c = a.galois_conjugates(K); c
[a]
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> c = a.galois_conjugates(K.galois_closure('a1')); c
[1/18*a1^4, -1/36*a1^4 + 1/2*a1, -1/36*a1^4 - 1/2*a1]
>>> c[Integer(0)]**Integer(3)
2
>>> parent(c[Integer(0)])
Number Field in a1 with defining polynomial x^6 + 108
>>> parent(c[Integer(0)]).is_galois()
# needs sage.groups
True
```

```
from sage.all import *
>>> L = CyclotomicField(Integer(3)).extension(x**Integer(3) - Integer(2),␣
˓→names=('a',)); (a,) = L._first_ngens(1)
>>> a.galois_conjugates(L)
[a, (-zeta3 - 1)*a, zeta3*a]
```

```
gcd(other)
```

Return the greatest common divisor of self and other.

INPUT:
• self, other – elements of a number field or maximal order.

OUTPUT:

• A generator of the ideal (self, other). If the parent is a number field, this always returns 0 or 1. For maximal orders, this raises ArithmeticError if the ideal is not principal.

EXAMPLES:

```
sage: K.<i> = QuadraticField(-1)
sage: (i+1).gcd(2)
1
sage: K(i).gcd(0)
1
sage: K(0).gcd(0)
0
sage: R = K.maximal_order()
sage: R(i+1).gcd(2)
i + 1
sage: R = K.order(2*i)
sage: R(1).gcd(R(4*i))
1
```

The following field has class number 3, but if the ideal (self, other) happens to be principal, this still works:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 7)
sage: K.class_number()
3
sage: a.gcd(7)
1
sage: R = K.maximal_order()
sage: R(a).gcd(7)
a
sage: R(a+1).gcd(2)
Traceback (most recent call last):
...
ArithmeticError: ideal (a + 1, 2) is not principal, gcd is not defined
sage: R(2*a - a^2).gcd(0)
a
sage: R(a).gcd(R(2*a)).parent()
Maximal Order generated by a in Number Field in a with defining polynomial x^3 - 7
```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(7), names=('a',)); (a,) = K._first_ngens()
>>> K.class_number()
3
>>> a.gcd(Integer(7))
1
>>> R = K.maximal_order()
>>> R(a).gcd(Integer(7))
a
>>> R(a+Integer(1)).gcd(Integer(2))
Traceback (most recent call last):
...
ArithmeticError: ideal (a + 1, 2) is not principal, gcd is not defined
>>> R(Integer(2)*a - a**Integer(2)).gcd(Integer(0))
a
Maximal Order generated by a in Number Field in a with defining polynomial x^3 - 7

global_height (prec=None)
Returns the absolute logarithmic height of this number field element.

INPUT:

- prec (int) – desired floating point precision (default: default RealField precision).

OUTPUT:

(real) The absolute logarithmic height of this number field element; that is, the sum of the local heights at all finite and infinite places, scaled by the degree to make the result independent of the parent field.

EXAMPLES:

sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4 + 3*x^2 - 17)
sage: b = a/2
sage: b.global_height()
0.789780699008...

sage: b.global_height(prec=Integer(200))
0.78978069900813892060267152032141577237037181070060784564457

The global height of an algebraic number is absolute, i.e. it does not depend on the parent field:

sage: QQ(6).global_height()  
1.79175946922805

sage: K(6).global_height()  
(continues on next page)
1.79175946922805

\texttt{sage: L.<b> = NumberField((a^2).minpoly())}
\texttt{sage: L.degree()}
2
\texttt{sage: b.global_height() \# element of } L (\text{degree 2 field})
1.41660667202811
\texttt{sage: (a^2).global_height() \# element of } K (\text{degree 4 field})
1.41660667202811

\texttt{>>> from sage.all import *}
\texttt{>>> QQ(Integer(6)).global_height()}
1.79175946922805
\texttt{>>> K(Integer(6)).global_height()}
1.79175946922805

\texttt{>>> L = NumberField((a*Integer(2)).minpoly(), names=('b',)); (b,) = L._first_ngens(1)}
\texttt{>>> L.degree()}
2
\texttt{>>> b.global_height() \# element of } L (\text{degree 2 field})
1.41660667202811
\texttt{>>> (a*Integer(2)).global_height() \# element of } K (\text{degree 4 field})
1.41660667202811

And of course every element has the same height as its inverse:

\texttt{sage: K.<s> = QuadraticField(2)}
\texttt{sage: s.global_height()}
0.346573590279973
\texttt{sage: (1/s).global_height() \# make sure that 11758 is fixed}
0.346573590279973

\texttt{>>> from sage.all import *}
\texttt{>>> K = QuadraticField(Integer(2), names=('s',)); (s,) = K._first_ngens(1)}
\texttt{>>> s.global_height()}
0.346573590279973
\texttt{>>> (Integer(1)/s).global_height() \# make sure that 11758 is fixed}
0.346573590279973

\textbf{global\_height\_arch(}\texttt{prec=None)}

Returns the total archimedean component of the height of \texttt{self}.

\textbf{INPUT:}

\begin{itemize}
  \item \texttt{prec (int)} – desired floating point precision (default: default \texttt{RealField} precision).
\end{itemize}

\textbf{OUTPUT:}

(real) The total archimedean component of the height of this number field element; that is, the sum of the local heights at all infinite places.

\textbf{EXAMPLES:}

\texttt{sage: R.<x> = QQ[]}
\texttt{sage: K.<a> = NumberField(x^4 + 3*x^2 - 17)}
\texttt{sage: b = a/2}
global_height_non_arch (prec=None)

Return the total non-archimedean component of the height of self.

INPUT:

• prec (int) – desired floating point precision (default: default RealField precision).

OUTPUT:

(real) The total non-archimedean component of the height of this number field element; that is, the sum of the local heights at all finite places, weighted by the local degrees.

ALGORITHM:

An alternative formula is log(d) where d is the norm of the denominator ideal; this is used to avoid factorization.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4 + 3*x^2 - 17)
sage: b = a/6
sage: b.global_height_non_arch()
7.16703787691222
```

Check that this is equal to the sum of the non-archimedean local heights:

```python
sage: [b.local_height(P) for P in b.support()]
[0.000000000000000, 0.693147180559945, 1.09861228866811, 1.09861228866811]
sage: sum([b.local_height(P, weighted=True) for P in b.support()])
7.16703787691222
```

(continues on next page)
A relative example:

```
sage: PK.<y> = K[]
sage: L.<c> = NumberField(y^2 + a)
sage: (c/10).global_height_non_arch()
18.4206807439524
```

**inverse_mod(\(I\))**

Returns the inverse of self mod the integral ideal \(I\).

**INPUT:**

- \(I\) – may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A **ValueError** is raised if \(I\) is non-integral or zero. A **ZeroDivisionError** is raised if \(I + (x) \neq (1)\).

**Note:** It's not implemented yet for non-integral elements.

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3)
sage: d = 3*a + 1
sage: d.inverse_mod(N)
1
```

```
sage: x = polygen(ZZ, 'x')
sage: k = NumberField(x**2 + 23, names=('a',)); (a,) = k._first_ngens(1)
sage: N = k.ideal(3)
sage: d = Integer(3)*a + Integer(1)
sage: d.inverse_mod(N)
1
```

```
sage: k.<a> = NumberField(x^3 + 11)
sage: d = a + 13
sage: d.inverse_mod(a^2)*d - 1 in k.ideal(a^2)
True
sage: d.inverse_mod((5, a + 1))*d - 1 in k.ideal(5, a + 1)
True
sage: K.<b> = k.extension(x^2 + 3)
sage: b.inverse_mod([37, a - b])
```
is_integer()

Test whether this number field element is an integer.

See also:

- is_rational() to test if this element is a rational number
- is_integral() to test if this element is an algebraic integer

EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<cbrt3> = NumberField(x^3 - 3)
sage: cbrt3.is_integer()
False
sage: (cbrt3^2 - cbrt3 + 2).is_integer()
False
sage: K(-12).is_integer()
False
sage: K(0).is_integer()
True
sage: K(1/2).is_integer()
False
```

```sage
>>> from sage.all import *  
>>> x = polygen(ZZ, 'x')  
>>> K = NumberField(x**Integer(3) - Integer(3), names=('cbrt3',)); (cbrt3,) = K._first_ngens(1)  
>>> cbrt3.is_integer()
False
>>> (cbrt3**Integer(2) - cbrt3 + Integer(2)).is_integer()
False
```
is_integral()

Determine if a number is in the ring of integers of this number field.

EXAMPLES:

```sage
x = polygen(ZZ, 'x')
K.<a> = NumberField(x^2 + 23)
a.is_integral()  # True

sage: t = (1+a)/2
t.is_integral()  # True

sage: t.minpoly()
x^2 - x + 6

sage: t = a/2
t.is_integral()  # False

t.minpoly()
x^2 + 23/4
```

An example in a relative extension:

```sage
K.<a,b> = NumberField([x^2 + 1, x^2 + 3])
(a + b).is_integral()  # True

sage: ((a-b)/2).is_integral()  # False
```

```sage
from sage.all import *

x = polygen(ZZ, 'x')
K = NumberField([x^2 + 1, x^2 + 3], names=('a', 'b')); (a, b) = K._first_ngens(2)
(a + b).is_integral()
```
is_norm\(^{(L, \text{element}=False, \text{proof}=True)}\)

Determine whether self is the relative norm of an element of \(L/K\), where \(K\) is self.parent().

**INPUT:**
- \(L\) – a number field containing \(K = \text{self.parent()}\).
- element – True or False, whether to also output an element of which self is a norm.
- proof – If True, then the output is correct unconditionally. If False, then the output is correct under GRH.

**OUTPUT:**
If element is False, then the output is a boolean \(B\), which is True if and only if self is the relative norm of an element of \(L\) to \(K\).

If element is True, then the output is a pair \((B, x)\), where \(B\) is as above. If \(B\) is True, then \(x\) is an element of \(L\) such that self == x.norm(K). Otherwise, \(x\) is None.

**ALGORITHM:**
Uses PARI's pari:rnfisnorm. See _rnfisnorm().

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: K.<beta> = NumberField(x^3 + 5)
sage: Q.<X> = K[]
sage: L = K.extension(X^2 + X + beta, 'gamma')
sage: (beta/2).is_norm(L)
False
sage: beta.is_norm(L)
True
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K.<beta> = NumberField(x**Integer(3) + Integer(5), names=('beta',)); (beta,) = K._first_ngens(1)
>>> Q = K['X']; (X,) = Q._first_ngens(1)
>>> L = K.extension(X**Integer(2) + X + beta, 'gamma')
>>> (beta/Integer(2)).is_norm(L)
False
>>> beta.is_norm(L)
True
```

With a relative base field:

```
sage: K.<a, b> = NumberField([x^2 - 2, x^2 - 3])
sage: L.<c> = K.extension(x^2 - 5)
sage: (2*a*b).is_norm(L)
True
sage: _, v = (2*b*a).is_norm(L, element=True)
sage: v.norm(K) == 2*a*b
True
```

2.1. Elements of number fields (implemented using NTL) 295
>>> from sage.all import *
>>> K = NumberField([x**Integer(2) - Integer(2), x**Integer(2) - Integer(3)],
    names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> L = K.extension(x**Integer(2) - Integer(5), names=('c',)); (c,) = L._
    first_ngens(1)
>>> (Integer(2)*a*b).is_norm(L)
True
>>> _, v = (Integer(2)*b*a).is_norm(L, element=True)
>>> v.norm(K) == Integer(2)*a*b
True

Non-Galois number fields:

sage: K.<a> = NumberField(x^2 + x + 1)
sage: Q.<X> = K[]
sage: L.<b> = NumberField(X^4 + a + 2)
sage: (a/4).is_norm(L)
True
sage: (a/2).is_norm(L)
# needs sage.groups
Traceback (most recent call last):
... Not Implemented Error: is_norm is not implemented unconditionally
for norms from non-Galois number fields
sage: (a/2).is_norm(L, proof=False)
# needs sage.groups
False

sage: K.<a> = NumberField(x^3 + x + 1)
sage: Q.<X> = K[]
sage: L.<b> = NumberField(X^4 + a)
sage: t, u = (-a).is_norm(L, element=True); u  # random (not unique)
b^3 + 1
sage: t
and u.norm(K) == -a
True

>>> from sage.all import *
>>> K = NumberField([x**Integer(2) - Integer(1), names=('a',))]; (a,) = K._
    first_ngens(1)
>>> Q = K['X']; (X,) = Q._first_ngens(1)
>>> L = NumberField([x**Integer(2) - a + Integer(2), names=('b',)); (b,) = L._
    first_ngens(1)
>>> (a/Integer(4)).is_norm(L)
True
>>> (a/Integer(2)).is_norm(L)
# needs sage.groups
Traceback (most recent call last):
... Not Implemented Error: is_norm is not implemented unconditionally
for norms from non-Galois number fields
>>> (a/Integer(2)).is_norm(L, proof=False)
# needs sage.groups
False

sage: K = NumberField([x**Integer(3) + x + Integer(1), names=('a',)); (a,) = K._
    first_ngens(1)
>>> Q = K['X']; (X,) = Q._first_ngens(1)

(continues on next page)
Verify that Issue #27469 has been fixed:

```
sage: L.<z24> = CyclotomicField(24); L
Cyclotomic Field of order 24 and degree 8
sage: K = L.subfield(z24^3, 'z8')[0]; K
Number Field in z8 with defining polynomial x^4 + 1
  with z8 = 0.7071067811865475? + 0.7071067811865475?*I
sage: flag, c = K(-7).is_norm(K, element=True); flag
True
sage: c.norm(K)
-7
sage: c in L
True
```

AUTHORS:

- Craig Citro (2008-04-05)
- Marco Streng (2010-12-03)

`is_nth_power(n)`

Return `True` if `self` is an `n`th power in its parent `K`.

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 - 7)
sage: K(7).is_nth_power(2)
True
sage: K(7).is_nth_power(4)
True
sage: K(7).is_nth_power(8)
False
sage: K((a-3)^5).is_nth_power(5)
True
```
**is_one()**

Test whether this number field element is 1.

**EXAMPLES:**

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K.<a> = NumberField(x^3 + 3)
>>> K(1).is_one()
True
>>> K(0).is_one()
False
>>> K(-1).is_one()
False
>>> K(1/2).is_one()
False
>>> a.is_one()
False
```

**is_padic_square** *(P, check=True)*

Return if self is a square in the completion at the prime *P*.

**INPUT:**

- *P* – a prime ideal
- *check* – *(default: True)*; check if *P* is prime

**EXAMPLES:**

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K.<a> = NumberField(x^3 + 3)
>>> K(Integer(1)).is_padic_square()  # check if 1 is a square in the p-adic completion
True
>>> K(Integer(0)).is_padic_square()  # check if 0 is a square in the p-adic completion
False
>>> K(-Integer(1)).is_padic_square()  # check if -1 is a square in the p-adic completion
False
>>> K(Integer(1)/Integer(2)).is_padic_square()  # check if 1/2 is a square in the p-adic completion
False
>>> a.is_padic_square()  # check if a is a square in the p-adic completion
False
```
is_prime()

Test whether this number-field element is prime as an algebraic integer.

Note that the behavior of this method differs from the behavior of is_prime() in a general ring, according to which (number) fields would have no nonzero prime elements.

EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 2)
sage: p = K.primes_above(2)[0]
sage: K(5).is_padic_square(p)
False
```

is_rational()

Test whether this number field element is a rational number.

See also:

- is_integer() to test if this element is an integer
- is_integral() to test if this element is an algebraic integer

EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<cbrt3> = NumberField(x^3 - 3)
sage: cbrt3.is_rational()
False
sage: (cbrt3^2 - cbrt3 + 1/2).is_rational()
False
sage: K(-12).is_rational()
True
```

(continues on next page)
is_square \ (\text{root}=\text{False})

Return True if self is a square in its parent number field and otherwise return False.

INPUT:

\begin{itemize}
    \item root – if True, also return a square root (or None if self is not a perfect square)
\end{itemize}

EXAMPLES:

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: m.<b> = NumberField(x^4 - 1789)
sage: b.is_square()
False
sage: c = (2/3*b + 5)^2; c
4/9*b^2 + 20/3*b + 25
sage: c.is_square()
True
sage: c.is_square(\text{True})
(True, 2/3*b + 5)
\end{verbatim}

We also test the functional notation.

\begin{verbatim}
sage: is_square(c, True)
(True, 2/3*b + 5)
\end{verbatim}
### is_square() Function

This function checks if a given element is a perfect square in the context of a number field. It returns `True` if `c` is a perfect square, and `False` otherwise.

#### Examples:

```python
sage: is_square(c)
True
sage: is_square(c + 1)
False
```

### is_totally_positive() Function

This function returns `True` if the element is positive in all real embeddings of its parent number field. It returns `False` otherwise.

#### Examples:

```python
>>> from sage.all import *
>>> is_square(c, True)
(True, 2/3*b + 5)
```

### is_unit() Function

This function returns `True` if the element is a unit in the ring where it is defined.

#### Examples:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - x - 1)
sage: OK = K.ring_of_integers()
sage: OK(a).is_unit()
True
```

(continues on next page)
It also works for relative fields and orders:

```python
sage: K.<a,b> = NumberField([x^2 - 3, x^4 + x^3 + x^2 + x + 1])
sage: OK = K.ring_of_integers()
sage: OK(a).is_unit()
True
sage: OK(b).is_unit()
False
sage: a.is_unit()
True
```

```
>>> from sage.all import *
```
• \( P \) – a prime ideal of the parent of \texttt{self}.
• \texttt{prec} (int) – desired floating point precision (default: \texttt{default RealField} precision).
• \texttt{weighted} (bool, default \texttt{False}) – if \texttt{True}, apply local degree weighting.

**OUTPUT:**

(real) The local height of this number field element at the place \( P \). If \texttt{weighted} is \texttt{True}, this is multiplied by the local degree (as required for global heights).

**EXAMPLES:**

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4 + 3*x^2 - 17)
sage: P = K.ideal(61).factor()[0][0]
sage: b = 1/(a^2 + 30)
sage: b.local_height(P)
4.11087386417331
sage: b.local_height(P, weighted=True)
8.22174772834662
sage: b.local_height(P, 200)
4.110873864173311248751891034256147463156817430812610629374
sage: (b^2).local_height(P)
8.22174772834662
sage: (b**-1).local_height(P)
0.000000000000000
```

A relative example:

```python
sage: PK.<y> = K[]
sage: L.<c> = NumberField(y^2 + a)
sage: L(1/4).local_height(L.ideal(2, c - a + 1))
1.38629436111989
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(4) + Integer(3)*x**Integer(2) - Integer(17),
˓→names=('a',)); (a,) = K._first_ngens(1)
>>> P = K.ideal(Integer(61)).factor()[Integer(0)][Integer(0)]
>>> b = Integer(1)/(a**Integer(2) + Integer(30))
>>> b.local_height(P)
4.11087386417331
>>> b.local_height(P, weighted=True)
8.22174772834662
>>> b.local_height(P, Integer(200))
4.110873864173311248751891034256147463156817430812610629374
>>> (b**Integer(2)).local_height(P)
8.22174772834662
>>> (b**-Integer(1)).local_height(P)
0.000000000000000
```

2.1. Elements of number fields (implemented using NTL)
local_height_arch\((i, \text{prec}=None, \text{weighted}=False)\)

Returns the local height of self at the \(i\)th infinite place.

**INPUT:**

- \(i\) (int) – an integer in range \((r+s)\) where \((r,s)\) is the signature of the parent field (so \(n = r + 2s\) is the degree).
- \(\text{prec}\) (int) – desired floating point precision (default: default RealField precision).
- \(\text{weighted}\) (bool, default False) – if True, apply local degree weighting, i.e. double the value for complex places.

**OUTPUT:**

(real) The archimedean local height of this number field element at the \(i\)th infinite place. If \(\text{weighted}\) is True, this is multiplied by the local degree (as required for global heights), i.e. 1 for real places and 2 for complex places.

**EXAMPLES:**

```python
data: R.<x> = QQ[]
data: K.<a> = NumberField(x^4 + 3*x^2 - 17)
data: [p.codomain() for p in K.places()]
[Real Field with 106 bits of precision,
  Real Field with 106 bits of precision,
  Complex Field with 53 bits of precision]
data: [a.local_height_arch(i) for i in range(3)]
[0.5301924545717755083366563897519,
  0.5301924545717755083366563897519,
  0.886414217456333]
data: [a.local_height_arch(i, weighted=True) for i in range(3)]
[0.5301924545717755083366563897519,
  0.5301924545717755083366563897519,
  1.77282843491267]
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(4) + Integer(3)*x**Integer(2) - Integer(17),˓→names=('a',)); (a,) = K._first_ngens(1)
>>> [p.codomain() for p in K.places()]
[Real Field with 106 bits of precision,
  Real Field with 106 bits of precision,
  Complex Field with 53 bits of precision]
>>> [a.local_height_arch(i) for i in range(Integer(3))]  
[0.5301924545717755083366563897519,
  0.5301924545717755083366563897519,
  0.886414217456333]
>>> [a.local_height_arch(i, weighted=True) for i in range(Integer(3))]  
[0.5301924545717755083366563897519,
  0.5301924545717755083366563897519,
  1.77282843491267]
```

A relative example:

```python
data: L.<b, c> = NumberFieldTower([x^2 - 5, x^3 + x + 3])
data: [(b + c).local_height_arch(i) for i in range(4)]
[1.238223390757884911842206617439,
  0.022403472299578578076974691491,
```
0.780028961749618,
1.16048938497298

```python
>>> from sage.all import *
>>>
L = NumberFieldTower([x**Integer(2) - Integer(5), x**Integer(3) + x → Integer(3)], names=('b', 'c',)); (b, c,) = L._first_ngens(2)
[1.238223390757884911842206617439,
0.022403472299578757875780769746914391,
0.780028961749618,
1.16048938497298]
```

```
matrix(base=None)
```

If `base` is `None`, return the matrix of right multiplication by the element on the power basis

\[ 1, x, x^2, \ldots, x^{d-1} \]

for the number field. Thus the rows of this matrix give the images of each of the \( x^i \).

If `base` is not `None`, then `base` must be either a field that embeds in the parent of `self` or a morphism to the parent of `self`, in which case this function returns the matrix of multiplication by `self` on the power basis, where we view the parent field as a field over `base`.

Specifying `base` as the base field over which the parent of `self` is a relative extension is equivalent to `base` being `None`.

**INPUT:**

- `base` – field or morphism

**EXAMPLES:**

Regular number field:

```python
sage: K.<a> = NumberField(QQ['x'].0^3 - 5)
sage: M = a.matrix(); M
[0 1 0]
[0 0 1]
[5 0 0]
sage: M.base_ring() is QQ
True
```

Relative number field:

```python
>>> from sage.all import *
>>> K = NumberField(QQ['x'].gen(0)**Integer(3) - Integer(5), names=('a',));...
>>> M = a.matrix(); M
[0 1 0]
[0 0 1]
[5 0 0]
>>> M.base_ring() is QQ
True
```

```
2.1. Elements of number fields (implemented using NTL) 305
```
from sage.all import *

L = K.extension(K['x'].gen(0)**Integer(2) - Integer(2), names=('b',)); (b, ) = L._first_ngens(1)
M = b.matrix(); M

M.base_ring() is K
True

Absolute number field:

sage: M = L.absolute_field('c').gen().matrix(); M
[ 0 1 0 0 0 0]
[ 0 0 1 0 0 0]
[ 0 0 0 1 0 0]
[ 0 0 0 0 1 0]
[ 0 0 0 0 0 1]
[-17 -60 -12 -10 6 0]
sage: M.base_ring() is QQ
True

>>> from sage.all import *

>>> M = L.absolute_field('c').gen().matrix(); M
[ 0 1 0 0 0 0]
[ 0 0 1 0 0 0]
[ 0 0 0 1 0 0]
[ 0 0 0 0 1 0]
[ 0 0 0 0 0 1]
[-17 -60 -12 -10 6 0]
>>> M.base_ring() is QQ
True

More complicated relative number field:

sage: L.<b> = K.extension(K['x'].0^2 - a); L
Number Field in b with defining polynomial x^2 - a over its base field
sage: M = b.matrix(); M
[0 1]
[a 0]
sage: M.base_ring() is K
True

>>> from sage.all import *

>>> L = K.extension(K['x'].gen(0)**Integer(2) - a, names=('b',)); (b,) = L._first_ngens(1); L
Number Field in b with defining polynomial x^2 - a over its base field
>>> M = b.matrix(); M
[0 1]
[a 0]
>>> M.base_ring() is K
True

An example where we explicitly give the subfield or the embedding:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 + 1); L.<a2> = NumberField(x^2 + 1)
sage: a.matrix(L)
Notice that if we compute all embeddings and choose a different one, then the matrix is changed as it should be:

```
sage: v = L.embeddings(K)
sage: a.matrix(v[1])
[ 0  1]
[-a2  0]
```

The norm is also changed:

```
sage: a.norm(v[1])
a2
sage: a.norm(v[0])
-a2
```

```
>>> from sage.all import *
>>> v = L.embeddings(K)
>>> a.matrix(v[Integer(1)])
[ 0  1]
[-a2  0]
```

```
>>> from sage.all import *
>>> v = L.embeddings(K)
>>> a.matrix(v[Integer(1)])
[ 0  1]
[-a2  0]
```

**minpoly** (*var=*’x’)

Return the minimal polynomial of this number field element.

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 3)
sage: a.minpoly('x')
x^2 + 3
sage: R.<X> = K[X]
sage: L.<b> = K.extension(X^2 - (22 + a))
sage: b.minpoly('t')
t^2 - a - 22
sage: b.absolute_minpoly('t')
t^4 - 44*t^2 + 487
sage: b^2 - (22+a)
0
```
```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> a.minpoly('x')
x^2 + 3
>>> R = K['X']; (X,) = R._first_ngens(1)
>>> L = K.extension(X**Integer(2) - (Integer(22) + a), names=('b',)); (b,) = L._first_ngens(1)
>>> b.minpoly('t')
t^2 - a - 22
>>> b.absolute_minpoly('t')
t^4 - 44*t^2 + 487
```

### multiplicative_order()

Return the multiplicative order of this number field element.

**EXAMPLES:**

```python
sage: K.<z> = CyclotomicField(5)
sage: z.multiplicative_order()
5
sage: (-z).multiplicative_order()
10
sage: (1+z).multiplicative_order()
+Infinity
```

```python
sage: x = polygen(QQ)
sage: K = NumberField(x**Integer(40) - x**Integer(20) + Integer(4), names=('a',))
>>> (Integer(1)+z).multiplicative_order()
+Infinity
```

An example in a relative extension:
```python
sage: K.<a, b> = NumberField([x^2 + x + 1, x^2 - 3])
sage: z = (a - 1)*b/3
sage: z.multiplicative_order()
12
sage: z^12==1 and z^6!=1 and z^4!=1
True
```

```python
>>> from sage.all import *
>>>
>>> K = NumberField([x**Integer(2) + x + Integer(1), x**Integer(2) -
˓→Integer(3)], names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> z = (a - Integer(1))*b/Integer(3)
>>> z.multiplicative_order()
12
>>> z**Integer(12)==Integer(1) and z**Integer(6)!=Integer(1) and
˓→z**Integer(4)!=Integer(1)
True
```

```
norm (K=None)

Return the absolute or relative norm of this number field element.

If K is given, then K must be a subfield of the parent L of self, in which case the norm is the relative
from L to K. In all other cases, the norm is the absolute norm down to Q.

EXAMPLES:
```
```
We illustrate that norm is compatible with towers:

```
sage: z = (a + b + c).norm(L); z.norm(M)
-11
```

If we are in an order, the norm is an integer:

```
sage: K.<a> = NumberField(x^3 - 2)
sage: a.norm().parent()
Rational Field
sage: R = K.ring_of_integers()
sage: R(a).norm().parent()
Integer Ring
```

```
When the base field is given by an embedding:

```
sage: K.<a> = NumberField(x^4 + 1)
sage: L.<a2> = NumberField(x^2 + 1)
sage: v = L.embeddings(K)
sage: a.norm(v[1])
a2
sage: a.norm(v[0])
-a2
```

```
>>> from sage.all import *
>>> K = NumberField(x**Integer(4) + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> L = NumberField(x**Integer(2) + Integer(1), names=('a2',)); (a2,) = L._first_ngens(1)
>>> v = L.embeddings(K)
>>> a.norm(v[Integer(1)])
a2
>>> a.norm(v[Integer(0)])
-a2
```

```
nth_root (n, all=False)
Return an n’th root of self in its parent K.
EXAMPLES:
```
```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 5)
sage: b = (1+a)/2
(continues on next page)
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> K(Integer(7)).nth_root(Integer(2))
a^2
```

```
ALGORITHM: Use PARI to factor x^n - self in K.
```

```
numerator_ideal()
Return the numerator ideal of this number field element.
The numerator ideal of a number field element a is the ideal of the ring of integers R obtained by intersecting aR with R.
See also:
denominator_ideal()  
```

```
EXAMPLES:
```
```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 5)
sage: b = (1+a)/2
(continues on next page)
```

2.1. Elements of number fields (implemented using NTL) 311
ord($P$)

Return the valuation of self at a given prime ideal $P$.

INPUT:

- $P$ – a prime ideal of the parent of self

Note: The method $ord()$ is an alias for $valuation()$.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4 + 3*x^2 - 17)
sage: P = K.ideal(61).factor()[0][0]
sage: b = a^2 + 30
sage: b.valuation(P)
1
sage: b.ord(P)
1
sage: type(b.valuation(P))
<class 'sage.rings.integer.Integer'>
```

```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(4) + Integer(3)*x**Integer(2) - Integer(17),
˓→names=('a',)); (a,) = K._first_ngens(1)
>>> P = K.ideal(Integer(61)).factor()[Integer(0)][Integer(0)]
>>> b = a**Integer(2) + Integer(30)
```
The function can be applied to elements in relative number fields:

```python
sage: L.<b> = K.extension(x^2 - 3)
```

```python
sage: [L(6).valuation(P) for P in L.primes_above(2)]
[4]
```

```python
sage: [L(6).valuation(P) for P in L.primes_above(3)]
[2, 2]
```

```python
>>> from sage.all import *
```

```python
L = K.extension(x**Integer(2) - Integer(3), names=('b',)); (b,) = L._
˓→first_ngens(1)
```

```python
sage: [L(Integer(6)).valuation(P) for P in L.primes_above(Integer(2))]
[4]
```

```python
sage: [L(Integer(6)).valuation(P) for P in L.primes_above(Integer(3))]
[2, 2]
```

```python
polynomial(var='x')
```

Return the underlying polynomial corresponding to this number field element.

The resulting polynomial is currently not cached.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
```

```python
sage: K.<a> = NumberField(x^5 - x - 1)
```

```python
sage: f = (-2/3 + 1/3*a)^4; f
1/81*a^4 - 8/81*a^3 + 8/27*a^2 - 32/81*a + 16/81
```

```python
sage: g = f.polynomial(); g
1/81*x^4 - 8/81*x^3 + 8/27*x^2 - 32/81*x + 16/81
```

```python
sage: parent(g)
Univariate Polynomial Ring in x over Rational Field
```

```python
>>> from sage.all import *
```

```python
x = polygen(ZZ, 'x')
```

```python
>>> K = NumberField(x**Integer(5) - x - Integer(1), names=('a',)); (a,) = K._
˓→first_ngens(1)
```

```python
>>> f = (-Integer(2)/Integer(3) + Integer(1)/Integer(3)*a)**Integer(4); f
1/81*a^4 - 8/81*a^3 + 8/27*a^2 - 32/81*a + 16/81
```

```python
>>> g = f.polynomial(); g
1/81*x^4 - 8/81*x^3 + 8/27*x^2 - 32/81*x + 16/81
```

```python
>>> parent(g)
Univariate Polynomial Ring in x over Rational Field
```

Note that the result of this function is not cached (should this be changed?):

```python
sage: g is f.polynomial()
False
```
Note that in relative number fields, this produces the polynomial of the internal representation of this element:

```python
>>> from sage.all import *
>>> g is f.polynomial()
False
```

```python
sage: R.<y> = K[]
sage: L.<b> = K.extension(y^2 - a)
sage: b.polynomial()
x
```

In some cases this might not be what you are looking for:

```python
sage: K.<a> = NumberField(x^2 + x + 1)
sage: R.<y> = K[]
sage: L.<b> = K.extension(y^2 + y + 2)
sage: b.polynomial()
1/2*x^3 + 3*x - 1/2
sage: R(list(b))
y
```

```python
>>> from sage.all import *
>>> R = K['y']; (y,) = R._first_ngens(1)
>>> L = K.extension(y**Integer(2) - a, names=('b',)); (b,) = L._first_ngens(1)
>>> b.polynomial()
x
```

**relative_norm()**

Return the relative norm of this number field element over the next field down in some tower of number fields.

**EXAMPLES:**

```python
sage: K1.<a1> = CyclotomicField(11)
sage: x = polygen(ZZ, 'x')
sage: K2.<a2> = K1.extension(x^2 - 3)
sage: (a1 + a2).relative_norm()
a1^2 - 3
sage: (a1 + a2).relative_norm().relative_norm() == (a1 + a2).absolute_norm()
True
sage: K.<x,y,z> = NumberField([x^2 + 1, x^3 - 3, x^2 - 5])
sage: (x + y + z).relative_norm()
y^2 + 2*z*y + 6
```
```
>>> from sage.all import *
>>> K1 = CyclotomicField(Integer(11), names=('a1',)); (a1,) = K1._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K2 = K1.extension(x**Integer(2) - Integer(3), names=('a2',)); (a2,) = K2._first_ngens(1)
>>> (a1 + a2).relative_norm()
a1^2 - 3
>>> (a1 + a2).relative_norm().relative_norm() == (a1 + a2).absolute_norm()
True
>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(3) - Integer(3), x**Integer(2) - Integer(5)], names=('x', 'y', 'z',)); (x, y, z,) = K._first_ngens(3)
>>> (x + y + z).relative_norm()
y^2 + 2*z*y + 6
```

`residue_symbol(P, m, check=True)`

The \( m \)-th power residue symbol for an element `self` and proper ideal `P`.

\[
\left( \frac{\alpha}{P} \right) \equiv \alpha^\frac{N(P)-1}{m} \mod P
\]

**Note:** accepts \( m = 1 \), in which case returns 1

**Note:** can also be called for an ideal from `sage.rings.number_field_ideal.residue_symbol`

**Note:** `self` is coerced into the number field of the ideal `P`

**Note:** if \( m = 2, \) `self` is an integer, and `P` is an ideal of a number field of absolute degree 1 (i.e. it is a copy of the rationals), then this calls `kronecker_symbol()`, which is implemented using GMP.

**INPUT:**

- `P` – proper ideal of the number field (or an extension)
- `m` – positive integer

**OUTPUT:**

- an \( m \)-th root of unity in the number field

**EXAMPLES:**

Quadratic Residue (11 is not a square modulo 17):

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x - 1)
sage: K(11).residue_symbol(K.ideal(17), 2)
-1
sage: kronecker_symbol(11, 17)
-1
```
The result depends on the number field of the ideal:

```
sage: K.<a> = NumberField(x - 1)
sage: L.<b> = K.extension(x^2 + 1)
sage: K(7).residue_symbol(K.ideal(11),2)
-1
sage: K(7).residue_symbol(L.ideal(11),2) # needs sage.libs.gap
1
```

Cubic Residue:

```
sage: K.<w> = NumberField(x^2 - x + 1)
sage: (w^2 + 3).residue_symbol(K.ideal(17),3)
-w
```

The field must contain the m-th roots of unity:

```
sage: K.<w> = NumberField(x^2 - x + 1)
sage: (w^2 + 3).residue_symbol(K.ideal(17),5)
Traceback (most recent call last):
  ... ValueError: The residue symbol to that power is not defined for the number_field
```

(continues on next page)
round()

Return the round (nearest integer) of this number field element. In case of ties, this relies on the default rounding for rational numbers.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: p = x**7 - 5*x**2 + x + 1
sage: a_AA = AA.polynomial_root(p, RIF(1,2))
sage: K.<a> = NumberField(p, embedding=a_AA)
sage: b = a**5 + a/2 - 1/7
sage: RR(b)
4.13444473767055
sage: b.round()
4
sage: (-b).round()
-4
sage: (b + 1/2).round()
5
sage: (-b - 1/2).round()
-5
```

This function always succeeds even if a tremendous precision is needed:

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> p = x**Integer(7) - Integer(5)*x**Integer(2) + x + Integer(1)
>>> a_AA = AA.polynomial_root(p, RIF(Integer(1),Integer(2)))
>>> K = NumberField(p, embedding=a_AA, names=('a',)); (a,) = K._first_ngens(1)
>>> b = a**Integer(5) + a/Integer(2) - Integer(1)/Integer(7)
>>> RR(b)
4.1344473767055
>>> b.round()
4
>>> (-b).round()
-4
>>> (b + Integer(1)/Integer(2)).round()
5
>>> (-b - Integer(1)/Integer(2)).round()
-5
```

(continues on next page)
If the number field is not embedded, this function is valid only if the element is rational:

```python
sage: p = x**5 - 3
sage: K.<a> = NumberField(p)
sage: [K(k/3).round() for k in range(-3,4)]
[-1, -1, 0, 0, 0, 1, 1]
sage: a.round()
Traceback (most recent call last):
... TypeError: floor not uniquely defined since no real embedding is specified
```

**sign()**

Return the sign of this algebraic number (if a real embedding is well defined)

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 2, embedding=AA(2)**(1/3))
sage: K.zero().sign()
0
sage: K.one().sign()
1
sage: (-K.one()).sign()
-1
sage: a.sign()
1
sage: (a - 234917380309015/186454048314072).sign()
1
sage: (a - 3741049304830488/2969272800976409).sign()
-1
```

```python
>> from sage.all import *
>> p = x**Integer(5) - Integer(3)
>> K = NumberField(p, names=('a',)); (a,) = K._first_ngens(1)
>> [K(k/Integer(3)).round() for k in range(-Integer(3),Integer(4))]
[-1, -1, 0, 0, 0, 1, 1]
>> a.round()
Traceback (most recent call last):
... TypeError: floor not uniquely defined since no real embedding is specified
```
If the field is not embedded in real numbers, this method will only work for rational elements:

```python
sage: L.<b> = NumberField(x^4 - x - 1)
sage: b.sign()
Traceback (most recent call last):
  ...TypeError: sign not well defined since no real embedding is specified
sage: L(-33/125).sign()
-1
sage: L.zero().sign()
0
```

```python
>>> from sage.all import *
```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> K(Integer(3)).sqrt()
a
>>> K(Integer(3)).sqrt(all=True)
[a, -a]
>>> K(a**Integer(10)).sqrt()
9*a
>>> K(Integer(49)).sqrt()
7
>>> K(Integer(1)+a).sqrt(extend=False)
Traceback (most recent call last):
...
ValueError: a + 1 not a square in Number Field in a with defining polynomial x^2 - 3
>>> K(Integer(0)).sqrt()
0
>>> K((Integer(7)+a)**Integer(2)).sqrt(all=True)
[a + 7, -a - 7]

Using the `extend` keyword:

```python
sage: K.<a> = QuadraticField(-5)
sage: z = K(-7).sqrt(extend=True); z
```
(continues on next page)
needs sage.symbolic
sqrt(-7)
sage: CyclotomicField(4)(4).sqrt(extend=False)
2

>>> from sage.all import *
>>> K = QuadraticField(-Integer(5))
>>> z = K(-Integer(7)).sqrt(extend=True); z
˓→ # needs sage.symbolic
sqrt(-7)
>>> CyclotomicField(Integer(4))(Integer(4)).sqrt(extend=False)
2

If extend=False an error is raised, if self is not a square:

sage: K = QuadraticField(-5)
sage: K(-7).sqrt(extend=False)
Traceback (most recent call last):
...
ValueError: -7 not a square in Number Field in a
with defining polynomial x^2 + 5 with a = 2.236067977499790?*I

ALGORITHM: Use PARI to factor \( x^2 - \text{self} \) in \( K \).

support()

Return the support of this number field element.

OUTPUT: A sorted list of the prime ideals at which this number field element has nonzero valuation. An error is raised if the element is zero.

EXAMPLES:

sage: x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)

>>> from sage.all import *
>>> x = ZZ['x'].gen()
>>> F = NumberField(x**Integer(3) - Integer(2), names=('t',)); (t,) = F._
˓→first_ngens(1)

sage: P5s = F(5).support()
sage: P5s
[Fractional ideal (-t^2 - 1), Fractional ideal (t^2 - 2*t - 1)]
sage: all(5 in P5 for P5 in P5s)
True
sage: all(P5.is_prime() for P5 in P5s)
True
trace ($K=\text{None}$)

Return the absolute or relative trace of this number field element.

If $K$ is given, then $K$ must be a subfield of the parent $L$ of $\text{self}$, in which case the trace is the relative trace from $L$ to $K$. In all other cases, the trace is the absolute trace down to $\mathbb{Q}$.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')

sage: K.<a> = NumberField(x^3 - 132/7*x^2 + x + 1); K
Number Field in a with defining polynomial x^3 - 132/7*x^2 + x + 1
sage: a.trace()
132/7
sage: (a + 1).trace() == a.trace() + 3
True
```

If we are in an order, the trace is an integer:

```python
sage: K.<zeta> = CyclotomicField(17)

sage: R = K.ring_of_integers()

sage: R(zeta).trace().parent()
Integer Ring
```

valuation ($P$)

Return the valuation of $\text{self}$ at a given prime ideal $P$. 

---
INPUT:

• \( P \) – a prime ideal of the parent of \( self \)

Note: The method \( \text{ord()} \) is an alias for \( \text{valuation()} \).

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4 + 3*x^2 - 17)
sage: P = K.ideal(61).factor()[0][0]
sage: b = a^2 + 30
sage: b.valuation(P)
1
sage: b.ord(P)
1
sage: type(b.valuation(P))
<class 'sage.rings.integer.Integer'>
```

The function can be applied to elements in relative number fields:

```python
sage: L.<b> = K.extension(x^2 - 3)
sage: [L(6).valuation(P) for P in L.primes_above(2)]
[4]
sage: [L(6).valuation(P) for P in L.primes_above(3)]
[2, 2]
```

```
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(4) + Integer(3)*x**Integer(2) - Integer(17),
˓→ names=('a',)); (a,) = K._first_ngens(1)
>>> P = K.ideal(Integer(61)).factor()[Integer(0)][Integer(0)]
>>> b = a**Integer(2) + Integer(30)
>>> b.valuation(P)
1
>>> b.ord(P)
1
>>> type(b.valuation(P))
<class 'sage.rings.integer.Integer'>
```

```
>>> from sage.all import *
>>> L = K.extension(x**Integer(2) - Integer(3), names=('b',)); (b,) = L._
˓→first_ngens(1)
>>> [L(Integer(6)).valuation(P) for P in L.primes_above(Integer(2))]
[4]
>>> [L(Integer(6)).valuation(P) for P in L.primes_above(Integer(3))]
[2, 2]
```

vector()

Return vector representation of \( self \) in terms of the basis for the ambient number field.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 1)
sage: (2/3*a - 5/6).vector()
```

(continues on next page)
(-5/6, 2/3)
sage: (-5/6, 2/3)
(-5/6, 2/3)
sage: O = K.order(2*a)
sage: (O.1).vector()
(0, 2)
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: (a + b).vector()
(b, 1)
sage: O = K.order([a,b])
sage: (O.1).vector()
(-b, 1)
sage: (O.2).vector()
(1, -b)

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> (Integer(2)/Integer(3)*a - Integer(5)/Integer(6)).vector()
(-5/6, 2/3)
>>> (-Integer(5)/Integer(6), Integer(2)/Integer(3))
(-5/6, 2/3)
>>> O = K.order(Integer(2)*a)
>>> (O.gen(1)).vector()
(0, 2)
>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(3)],
    names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> (a + b).vector()
(b, 1)
>>> O = K.order([a,b])
>>> (O.gen(1)).vector()
(-b, 1)
>>> (O.gen(2)).vector()
(1, -b)

class sage.rings.number_field.number_field_element.NumberFieldElement_absolute
Bases: NumberFieldElement

absolute_charpoly (var='x', algorithm=None)

Return the characteristic polynomial of this element over \( \mathbb{Q} \).

For the meaning of the optional argument algorithm, see charpoly().

EXAMPLES:

sage: x = ZZ['x'].0
sage: K.<a> = NumberField(x^4 + 2, 'a')
sage: a.absolute_charpoly()
x^4 + 2
sage: a.absolute_charpoly('y')
y^4 + 2
sage: (-a^2).absolute_charpoly()
x^4 + 4*x^2 + 4
sage: (-a^2).absolute_minpoly()
x^2 + 2
absolute_charpoly(algorithm='pari') == absolute_charpoly(algorithm='sage')
True

>>> from sage.all import *
>>> x = ZZ['x'].gen(0)
>>> K = NumberField(x**Integer(4) + Integer(2), 'a', names=('a',)); (a,) = K._first_ngens(1)
>>> a.absolute_charpoly()
x^4 + 2
>>> a.absolute_charpoly('y')
y^4 + 2
>>> (-a**Integer(2)).absolute_charpoly()
x^4 + 4*x^2 + 4
>>> (-a**Integer(2)).absolute_minpoly()
x^2 + 2
>>> a.absolute_charpoly(algorithm='pari') == a.absolute_charpoly(algorithm='sage')
True

absolute_minpoly(var='x', algorithm=None)

Return the minimal polynomial of this element over \( \mathbb{Q} \).

For the meaning of the optional argument algorithm, see charpoly().

EXAMPLES:

sage: x = ZZ['x'].0
sage: f = (x^10 - 5*x^9 + 15*x^8 - 68*x^7 + 81*x^6 - 221*x^5 + 141*x^4 - 242*x^3 - 13*x^2 - 33*x - 135)
```
sage: K.<a> = NumberField(f, 'a')
sage: a.absolute_charpoly()
x^10 - 5*x^9 + 15*x^8 - 68*x^7 + 81*x^6 - 221*x^5 + 141*x^4 - 242*x^3 - 13*x^2 - 33*x - 135
```
sage: a.absolute_charpoly('y')
y^10 - 5*y^9 + 15*y^8 - 68*y^7 + 81*y^6 - 221*y^5 + 141*y^4 - 242*y^3 - 13*y^2 - 33*y - 135
sage: b = (-79/9995*a^9 + 52/9995*a^8 + 271/9995*a^7 + 1663/9995*a^6 + 13204/9995*a^5 + 5573/9995*a^4 + 8435/1999*a^3 - 3116/9995*a^2 + 7734/1999*a + 1620/1999)
```
sage: b.absolute_charpoly()
x^5 + 5*x^4 - 40*x^2 - 19*x + 135
```
sage: b.absolute_minpoly()
x^5 + 5*x^4 - 40*x^2 - 19*x + 135
sage: b.absolute_minpoly(algorithm='pari') == b.absolute_minpoly(algorithm='sage')
# needs sage.libs.pari
True

>>> from sage.all import *
>>> x = ZZ['x'].gen(0)
>>> f = (x**Integer(10) - Integer(5)*x**Integer(9) + Integer(15)*x**Integer(8) - Integer(68)*x**Integer(7) + Integer(221)*x**Integer(5)
`charpoly` (var=`x`, algorithm=None)

The characteristic polynomial of this element, over \( \mathbb{Q} \) if `self` is an element of a field, and over \( \mathbb{Z} \) is `self` is an element of an order.

This is the same as `absolute_charpoly()` since this is an element of an absolute extension.

The optional argument `algorithm` controls how the characteristic polynomial is computed: 'pari' uses PARI, 'sage' uses `charpoly` for Sage matrices. The default value `None` means that 'pari' is used for small degrees (up to the value of the constant `TUNE_CHARPOLY_NF`, currently at 25), otherwise 'sage' is used. The constant `TUNE_CHARPOLY_NF` should give reasonable performance on all architectures; however, if you feel the need to customize it to your own machine, see Issue #5213 for a tuning script.

EXAMPLES:

We compute the characteristic polynomial of the cube root of 2.

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3 - 2)
sage: a.charpoly('x')
x^3 - 2
sage: a.charpoly('y').parent()
Univariate Polynomial Ring in y over Rational Field
```
**is_real_positive (min_prec=53)**

Using the \( n \) method of approximation, return `True` if `self` is a real positive number and `False` otherwise. This method is completely dependent of the embedding used by the \( n \) method.

The algorithm first checks that `self` is not a strictly complex number. Then if `self` is not zero, by approximation more and more precise, the method answers `True` if the number is positive. Using `RealInterval`, the result is guaranteed to be correct.

For `CyclotomicField`, the embedding is the natural one sending \( \zeta_n \) on \( \cos(2 * \pi / n) \).

**EXAMPLES:**

```sage
sage: K.<a> = CyclotomicField(3)
sage: (a + a^2).is_real_positive()
False
sage: (-a - a^2).is_real_positive()
True
sage: K.<a> = CyclotomicField(1000)
sage: (a + a^(-1)).is_real_positive()
True
sage: K.<a> = CyclotomicField(1009)

sage: d = a^252
sage: (d + d.conjugate()).is_real_positive()
True
sage: d = a^253
sage: (d + d.conjugate()).is_real_positive()
False
sage: K.<a> = QuadraticField(3)
sage: a.is_real_positive()
True
sage: K.<a> = QuadraticField(-3)
sage: a.is_real_positive()
False
```

```python
>>> from sage.all import *
>>> K = CyclotomicField(Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> (a + a**Integer(2)).is_real_positive()
False
>>> (-a - a**Integer(2)).is_real_positive()
True
>>> K = CyclotomicField(Integer(1000), names=('a',)); (a,) = K._first_ngens(1)
>>> (a + a**(-Integer(1))).is_real_positive()
True
>>> K = CyclotomicField(Integer(1009), names=('a',)); (a,) = K._first_ngens(1)
>>> d = a**Integer(252)
>>> (d + d.conjugate()).is_real_positive()
True
>>> d = a**Integer(253)
>>> (d + d.conjugate()).is_real_positive()
False
>>> K = QuadraticField(Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> a.is_real_positive()
True
>>> K = QuadraticField(-Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> a.is_real_positive()
False
```

(continues on next page)
lift \((var='x')\)

Return an element of \(\mathbb{Q}[x]\), where this number field element lives in \(\mathbb{Q}[x]/(f(x))\).

**EXAMPLES:**

```python
sage: K.<a> = QuadraticField(-3)
sage: a.lift()
x
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> a.lift()
x
```

list()

Return the list of coefficients of \(self\) written in terms of a power basis.

**EXAMPLES:**

```python
sage: K.<z> = CyclotomicField(3)
sage: (2 + 3/5*z).list()
[2, 3/5]
sage: (5*z).list()
[0, 5]
sage: K(3).list()
[3, 0]
```

```python
>>> from sage.all import *
>>> K = CyclotomicField(Integer(3), names=('z',)); (z,) = K._first_ngens(1)
>>> (Integer(2) + Integer(3)/Integer(5)*z).list()
[2, 3/5]
>>> (Integer(5)*z).list()
[0, 5]
>>> K(Integer(3)).list()
[3, 0]
```

minpoly \((var='x', algorithm=None)\)

Return the minimal polynomial of this number field element.

For the meaning of the optional argument \(algorithm\), see \(charpoly()\).

**EXAMPLES:**

We compute the characteristic polynomial of cube root of 2.

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3 - 2)
sage: a.minpoly('x')
x^3 - 2
sage: a.minpoly('y').parent()
Univariate Polynomial Ring in y over Rational Field
```
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> a.minpoly('x')
x^3 - 2
>>> a.minpoly('y').parent()
Univariate Polynomial Ring in y over Rational Field

2.1. Elements of number fields (implemented using NTL)
absolute_minpoly (var='x', algorithm=None)

Return the minimal polynomial over \(\mathbb{Q}\) of this element.

For the meaning of the optional argument algorithm, see absolute_charpoly().

EXAMPLES:

\[
\begin{align*}
\text{sage: } & x = \text{polygen}(\mathbb{Z}, 'x') \\
\text{sage: } & K.<a, b> = \text{NumberField}([x^2 + 2, x^2 + 1000*x + 1]) \\
\text{sage: } & y = K['y'].0 \\
\text{sage: } & L.<c> = K.extension(y^2 + a*y + b) \\
\text{sage: } & c.absolute_charpoly() \\
& x^8 - 1996*x^6 + 996006*x^4 + 1997996*x^2 + 1 \\
\text{sage: } & c.absolute_minpoly() \\
& x^8 - 1996*x^6 + 996006*x^4 + 1997996*x^2 + 1 \\
\text{sage: } & L(a).absolute_charpoly() \\
& x^8 + 8*x^6 + 24*x^4 + 32*x^2 + 16 \\
\text{sage: } & L(a).absolute_minpoly() \\
& x^2 + 2 \\
\text{sage: } & L(b).absolute_charpoly() \\
& x^8 + 4000*x^7 + 6000004*x^6 + 4000012000*x^5 + 1000012000006*x^4 \\
& + 40000120000*x^3 + 6000004*x^2 + 4000*x + 1 \\
\text{sage: } & L(b).absolute_minpoly() \\
& x^2 + 1000*x + 1
\end{align*}
\]
charpoly (var='x')

The characteristic polynomial of this element over its base field.

EXAMPLES:

```
sage: x = ZZ['x'].0
sage: K.<a, b> = QQ.extension([x^2 + 2, x^5 + 400*x^4 + 11*x^2 + 2])
sage: a.charpoly()
x^2 + 2
sage: b.charpoly()
x^2 - 2*b*x + b^2
sage: b.minpoly()
x - b
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: y = K['y'].0
sage: L.<c> = K.extension(y^2 + a*y + b)
sage: c.charpoly()
x^2 + a*x + b
sage: c.minpoly()
x^2 + a*x + b
sage: L(a).charpoly()
x^2 - 2*a*x - 2
sage: L(a).minpoly()
x - a
sage: L(b).charpoly()
x^2 - 2*b*x - 1000*b - 1
sage: L(b).minpoly()
x - b
```

>>> from sage.all import *

```python
>>> x = ZZ['x'].gen(0)
>>> K = QQ.extension([x**Integer(2) + Integer(2), x**Integer(5) + Integer(400)*x**Integer(4) + Integer(11)*x**Integer(2) + Integer(2)], names=(a, b,)); (a, b,) = K._first_ngens(2)
>>> a.charpoly()
x^2 + 2
>>> b.charpoly()
x^2 - 2*b*x + b^2
>>> b.minpoly()
x - b
```

```python
>>> K = NumberField([x**Integer(2) + Integer(2), x**Integer(2) + Integer(1000)*x + Integer(1)], names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> y = K['y'].gen(0)
>>> L = K.extension(y**Integer(2) + a*y + b, names=('c',)); (c,) = L._first_ngens(1)
>>> c.charpoly()
x^2 + a*x + b
>>> c.minpoly()
x^2 + a*x + b
```

(continues on next page)
L(a).charpoly()
x^2 - 2*a*x - 2
L(a).minpoly()
x - a
L(b).charpoly()
x^2 - 2*b*x - 1000*b - 1
L(b).minpoly()
x - b

\textbf{lift}(var='x')

Return an element of $K[x]$, where this number field element lives in the relative number field $K[x]/(f(x))$.

\textbf{EXAMPLES:}

sage: K.<a> = QuadraticField(-3)
sage: x = polygen(K)
sage: L.<b> = K.extension(x^7 + 5)
sage: u = L(1/2*a + 1/2 + b + (a-9)*b^5)
sage: u.lift()
(a - 9)*x^5 + x + 1/2*a + 1/2

\textbf{list()}

Return the list of coefficients of \texttt{self} written in terms of a power basis.

\textbf{EXAMPLES:}

sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^3 + 2, x^2 + 1])
sage: a.list()
[0, 1, 0]
sage: v = (K.base_field().gen(0) + a)**2; v
a^2 + 2*b*a - 1
sage: v.list()
[-1, 2*b, 1]
valuation($P$)

Return the valuation of self at a given prime ideal $P$.

INPUT:

- $P$ – a prime ideal of a relative number field which is the parent of self

EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<a, b, c> = NumberField([x^2 - 2, x^2 - 3, x^2 - 5])
sage: P = K.prime_factors(5)[1]
sage: (2*a + b - c).valuation(P)
1
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField([x**Integer(2) - Integer(2), x**Integer(2) - Integer(3),
                  x**Integer(2) - Integer(5)], names=('a', 'b', 'c',)); (a, b, c,) = K._first_ngens(3)

>>> P = K.prime_factors(Integer(5))[Integer(1)]

>>> (Integer(2)*a + b - c).valuation(P)
1
```

class sage.rings.number_field.number_field_element.OrderElement_absolute

Bases: NumberFieldElement_absolute

Element of an order in an absolute number field.

EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 1)
sage: O2 = K.order(2*a)
sage: w = O2.1; w
2*a
sage: parent(w)
Order of conductor 2 generated by 2*a in Number Field in a with defining polynomial x^2 + 1

sage: w.absolute_charpoly()
x^2 + 4
sage: w.absolute_charpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
sage: w.absolute_minpoly()
x^2 + 4
sage: w.absolute_minpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = K._first_ngens(1)

>>> O2 = K.order(Integer(2)*a)

>>> w = O2.gen(1); w
2*a

>>> parent(w)
Order of conductor 2 generated by 2*a in Number Field in a with defining polynomial x^2 + 1
```

(continues on next page)
"polynomial x^2 + 1

|>>> w.absolute_charpoly() |
x^2 + 4
|>>> w.absolute_charpoly().parent() |
Univariate Polynomial Ring in x over Integer Ring
|>>> w.absolute_minpoly() |
x^2 + 4
|>>> w.absolute_minpoly().parent() |
Univariate Polynomial Ring in x over Integer Ring

inverse_mod(I)

Return an inverse of self modulo the given ideal.

INPUT:

- I – may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A ValueError is raised if I is non-integral or is zero. A ZeroDivisionError is raised if I + (x) ≠ (1).

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: OE.<w> = EquationOrder(x^3 - x + 2)
sage: w.inverse_mod(13)
6*w^2 - 6
sage: w * (w.inverse_mod(13)) - 1 in 13*OE.number_field()
True
sage: w.inverse_mod(13).parent() == OE
True
sage: w.inverse_mod(2)
Traceback (most recent call last):
... ZeroDivisionError: w is not invertible modulo Fractional ideal (2)
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> OE = EquationOrder(x**Integer(3) - x + Integer(2), names=('w',)); (w,) = OE._first_ngens(1)

>>> w.inverse_mod(Integer(13))
6*w^2 - 6

>>> w * (w.inverse_mod(Integer(13))) - Integer(1) in Integer(13)*OE.number_field()
True
>>> w.inverse_mod(Integer(13)).parent() == OE
True
>>> w.inverse_mod(Integer(2))
Traceback (most recent call last):
...
ZeroDivisionError: w is not invertible modulo Fractional ideal (2)
```

class sage.rings.number_field.number_field_element.OrderElement_relative

   Bases: NumberFieldElement_relative

   Element of an order in a relative number field.

   EXAMPLES:
absolute_charpoly (\textit{var}='x')

The absolute characteristic polynomial of this order element over \( \mathbb{Z} \).

EXAMPLES:

```python
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order()
sage: _, u, _, v = OK.basis()
sage: t = 2*u - v; t
-b
sage: t.absolute_charpoly()
x^4 - 6*x^2 + 9
sage: t.absolute_minpoly()
x^2 - 3
sage: t.absolute_charpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
```

absolute_minpoly (\textit{var}='x')

The absolute minimal polynomial of this order element over \( \mathbb{Z} \).

EXAMPLES:

```python
>>> from sage.all import *
>>> x = ZZ['x'].gen(0)
>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(3)],
˓→names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> OK = K.maximal_order()
>>> _, u, _, v = OK.basis()
>>> t = Integer(2)*u - v; t
-b
>>> t.absolute_charpoly()
x^4 - 6*x^2 + 9
>>> t.absolute_minpoly()
x^2 - 3
>>> t.absolute_charpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
```
The characteristic polynomial of this order element over its base ring.

This special implementation works around Issue #4738. At this time the base ring of relative order elements is \( \mathbb{Z} \); it should be the ring of integers of the base field.

EXAMPLES:

```python
sage: x = ZZ['x'].gen(0)

sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])

sage: OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]

sage: charpoly(OK.1)
x^2 + b*x + 1

sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order generated by b
in Number Field in b with defining polynomial x^2 - 3
```

```
[ x^2 - 2*x + 1, x^2 + b*x + 1, x^2 - x + 1, x^2 + 1 ]
```
[charpoly(t) for t in OK.basis()]
[x^2 - 2*x + 1, x^2 + b*x + 1, x^2 - x + 1, x^2 + 1]

\textbf{inverse\_mod(I)}

Return an inverse of \texttt{self} modulo the given ideal.

\textbf{INPUT:}

- \texttt{I} – may be an ideal of \texttt{self.parent()}, or an element or list of elements of \texttt{self.parent()} generating a nonzero ideal. A \texttt{ValueError} is raised if \texttt{I} is non-integral or is zero. A \texttt{ZeroDivisionError} is raised if \texttt{I + (x) \neq (1)}.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: E.<a,b> = NumberField([x^2 - x + 2, x^2 + 1])
sage: OE = E.ring_of_integers()
sage: t = OE(b - a).inverse_mod(17*b)
sage: t*(b - a) - 1 in E.ideal(17*b)
True
sage: t.parent() == OE
True
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> E = NumberField([x**Integer(2) - x + Integer(2), x**Integer(2) + Integer(1)], names=(a, b,)); (a, b,) = E._first_ngens(2)
>>> OE = E.ring_of_integers()
>>> t = OE(b - a).inverse_mod(Integer(17)*b)
>>> t*(b - a) - Integer(1) in E.ideal(Integer(17)*b)
True
>>> t.parent() == OE
True
\end{verbatim}

\textbf{minpoly(var='x')}\footnote{Continued from previous page}

The minimal polynomial of this order element over its base ring.

This special implementation works around Issue \#4738. At this time the base ring of relative order elements is \( \mathbb{Z} \); it should be the ring of integers of the base field.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = 2Z['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
sage: minpoly(OK.1)
x^2 + b*x + 1
sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order generated by b
in Number Field in b with defining polynomial x^2 - 3
sage: _, u, _, v = OK.basis()
sage: t = 2*u - v; t
-b
sage: t.charpoly()
x^2 + 2*b*x + 3
sage: t.minpoly()
\end{verbatim}

(continues on next page)
x + b

sage: t.absolute_charpoly()
x^4 - 6*x^2 + 9
sage: t.absolute_minpoly()
x^2 - 3

>>> from sage.all import *
>>> x = ZZ['x'].gen(0)
>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(3)],
˓→names=('a', 'b')); (a, b,) = K._first_ngens(2)
>>> OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
>>> minpoly(OK.gen(1))
x^2 + b*x + 1
>>> charpoly(OK.gen(1)).parent()
Univariate Polynomial Ring in x over Maximal Order generated by b
in Number Field in b with defining polynomial x^2 - 3
>>> u, _, v = OK.basis()
>>> t = Integer(2)*u - v; t
-b
>>> t.charpoly()
x^2 + 2*b*x + 3
>>> t.minpoly()
x + b
>>> t.absolute_charpoly()
x^4 - 6*x^2 + 9
>>> t.absolute_minpoly()
x^2 - 3

sage.rings.number_field.number_field_element.is_NumberFieldElement(x)
Return True if x is of type NumberFieldElement, i.e., an element of a number field.

EXAMPLES:

sage: from sage.rings.number_field.number_field_element import is_
˓→NumberFieldElement
sage: is_NumberFieldElement(2)
doctest:warning...
DeprecationWarning: is_NumberFieldElement is deprecated; use isinstance(..., sage.structure.element.NumberFieldElement) instead
See https://github.com/sagemath/sage/issues/34931 for details.
False
sage: x = polygen(ZZ, 'x')
\sagen: k.<a> = NumberField(x^7 + 17*x + 1)
\sage: is_NumberFieldElement(a+1)
True

>>> from sage.all import *
>>> from sage.rings.number_field.number_field_element import is_NumberFieldElement
>>> is_NumberFieldElement(Integer(2))
doctest:warning...
DeprecationWarning: is_NumberFieldElement is deprecated; use isinstance(..., sage.structure.element.NumberFieldElement) instead
See https://github.com/sagemath/sage/issues/34931 for details.
2.2 Elements optimized for quadratic number fields

This module defines a Cython class \texttt{NumberFieldElement\_quadratic} to speed up computations in quadratic extensions of \( \mathbb{Q} \).

\textbf{Todo:} The \texttt{_new()} method should be overridden in this class to copy the \texttt{D} and \texttt{standard\_embedding} attributes.

\textbf{AUTHORS:}

- Robert Bradshaw (2007-09): initial version
- David Harvey (2007-10): fixed up a few bugs, polish around the edges
- David Loeffler (2009-05): added more documentation and tests
- Vincent Delecroix (2012-07): added comparisons for quadratic number fields (Issue \#13213), abs, floor and ceil functions (Issue \#13256)

\begin{code}
\begin{verbatim}
class sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian:
    Bases: NumberFieldElement_quadratic_sqrt

    An element of \( \mathbb{Q}[\sqrt{i}] \).

    Some methods of this class behave slightly differently than the corresponding methods of general elements of quadratic number fields, especially with regard to conversions to parents that can represent complex numbers in rectangular form.

    In addition, this class provides some convenience methods similar to methods of symbolic expressions to make the behavior of \( a + \sqrt{-1} \cdot b \) with rational \( a, b \) closer to that when \( a, b \) are expressions.

    EXAMPLES:

    sage: type(I)
    <class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian'>

    sage: mi = QuadraticField(-1, embedding=CC(0,-1)).gen()
    sage: type(mi)
    <class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian'>
    sage: CC(mi)
    -1.00000000000000*I

    >>> from sage.all import *
    >>> type(I)
    <class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian'>
\end{verbatim}
\end{code}
\texttt{imag()} \hspace{1cm} \text{(Imaginary part.)}

\textbf{EXAMPLES:}

\begin{verbatim}
sage: (1 + 2*I).imag()
sage: (1 + 2*I).imag().parent()
sage: K.<mi> = QuadraticField(-1, embedding=CC(0,-1))
sage: (1 - mi).imag()
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> (Integer(1) + Integer(2)*I).imag()
>>> (Integer(1) + Integer(2)*I).imag().parent()
>>> K = QuadraticField(-Integer(1), embedding=CC(Integer(0),-Integer(1)),
                        names=('mi',)); (mi,) = K._first_ngens(1)
>>> (Integer(1) - mi).imag()
\end{verbatim}

\texttt{imag_part()} \hspace{1cm} \text{(Imaginary part.)}

\textbf{EXAMPLES:}

\begin{verbatim}
sage: (1 + 2*I).imag()
sage: (1 + 2*I).imag().parent()
sage: K.<mi> = QuadraticField(-1, embedding=CC(0,-1))
sage: (1 - mi).imag()
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> (Integer(1) + Integer(2)*I).imag()
>>> (Integer(1) + Integer(2)*I).imag().parent()
>>> K = QuadraticField(-Integer(1), embedding=CC(Integer(0),-Integer(1)),
                        names=('mi',)); (mi,) = K._first_ngens(1)
\end{verbatim}
log(*args, **kwds)
Complex logarithm (standard branch).

EXAMPLES:
```
sage: I.log()  # needs sage.symbolic
1/2*I*pi
```

real()
Real part.

EXAMPLES:
```
sage: (1 + 2*I).real()
1
sage: (1 + 2*I).real().parent()
Rational Field
```

real_part()
Real part.

EXAMPLES:
```
sage: (1 + 2*I).real()
1
sage: (1 + 2*I).real().parent()
Rational Field
```

class sage.rings.number_field.number_field_element_quadratic.
NumberFieldElement_quadratic
Bases: NumberFieldElement_absolute

A NumberFieldElement_quadratic object gives an efficient representation of an element of a quadratic extension of $\mathbb{Q}$.
Elements are represented internally as triples \((a, b, c)\) of integers, where \(\gcd(a, b, c) = 1\) and \(c > 0\), representing the element \((a + b\sqrt{D})/c\). Note that if the discriminant \(D\) is 1 mod 4, integral elements do not necessarily have \(c = 1\).

**ceil()**

Return the ceil.

**EXAMPLES:**

```
sage: K.<sqrt7> = QuadraticField(7, name='sqrt7')
sage: sqrt7.ceil()
3
sage: (-sqrt7).ceil()
-2
sage: (1022/313*sqrt7 - 14/23).ceil()
9
```

```
>>> from sage.all import *
>>> K = QuadraticField(Integer(7), name='sqrt7', names=('sqrt7',)); (sqrt7,)
˓→= K._first_ngens()
>>> sqrt7.ceil()
3
>>> (-sqrt7).ceil()
-2
>>> (Integer(1022)/Integer(313)*sqrt7 - Integer(14)/Integer(23)).ceil()
9
```

**charpoly**(\(\text{var}=x^{'}, \text{algorithm}=\text{None}\))

The characteristic polynomial of this element over \(\mathbb{Q}\).

**INPUT:**

- **\text{var}** – the minimal polynomial is defined over a polynomial ring
  in a variable with this name. If not specified, this defaults to 'x'
- **\text{algorithm}** – for compatibility with general number field elements; ignored

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - x + 13)
sage: a.charpoly()
x^2 - x + 13
sage: b = 3 - a/2
sage: f = b.charpoly(); f
x^2 - 11/2*x + 43/4
sage: f(b)
0
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - x + Integer(13), names=('a',)); (a,)
˓→K._first_ngens()
>>> a.charpoly()
x^2 - x + 13
>>> b = Integer(3) - a/Integer(2)
>>> f = b.charpoly(); f
x^2 - 11/2*x + 43/4
```

(continues on next page)
continued_fraction()
Return the (finite or ultimately periodic) continued fraction of self.

EXAMPLES:

```sage
sage: K.<sqrt2> = QuadraticField(2)
sage: cf = sqrt2.continued_fraction(); cf
[1; (2)*]
sage: cf.n()  
1.41421356237310
sage: sqrt2.n()  
1.41421356237309
sage: cf.value()  
sqrt2
sage: (sqrt2/3 + 1/4).continued_fraction()  
[0; 1, (2, 1, 1, 2, 3, 2, 1, 1, 2, 5, 1, 1, 14, 1, 1, 5)*]
```

```python
>>> from sage.all import *
>>> K = QuadraticField(Integer(2), names=('sqrt2',)); (sqrt2,) = K._first_ngens(1)
```

```sage
>>> cf = sqrt2.continued_fraction(); cf
[1; (2)*]
```

```sage
>>> cf.n()  
1.41421356237310
>>> sqrt2.n()  
1.41421356237309
>>> cf.value()  
sqrt2
>>> (sqrt2/3 + 1/4).continued_fraction()  
[0; 1, (2, 1, 1, 2, 3, 2, 1, 1, 2, 5, 1, 1, 14, 1, 1, 5)*]
```

continued_fraction_list()
Return the preperiod and the period of the continued fraction expansion of self.

EXAMPLES:

```sage
sage: K.<sqrt2> = QuadraticField(2)
sage: sqrt2.continued_fraction_list()  
((1,), (2,))
sage: (1/2 + sqrt2/3).continued_fraction_list()  
((0, 1, 33), (1, 32))
```

```python
>>> from sage.all import *
```

```sage
>>> K = QuadraticField(Integer(2), names=('sqrt2',)); (sqrt2,) = K._first_ngens(1)
```

```sage
>>> cf = sqrt2.continued_fraction(); cf
[1; (2)*]
```

```sage
>>> cf.n()  
1.41421356237310
>>> sqrt2.n()  
1.41421356237309
>>> cf.value()  
sqrt2
>>> (sqrt2/3 + 1/4).continued_fraction()  
[0; 1, (2, 1, 1, 2, 3, 2, 1, 1, 2, 5, 1, 1, 14, 1, 1, 5)*]
```

For rational entries a pair of tuples is also returned but the second one is empty:
denominator()

Return the denominator of self.

This is the LCM of the denominators of the coefficients of self, and thus it may well be \( > 1 \) even when the element is an algebraic integer.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 5)
sage: b = (a + 1)/2
sage: b.denominator()
2
sage: b.is_integral()
True

sage: K.<c> = NumberField(x^2 - x + 7)
sage: c.denominator()
1
```

floor()

Returns the floor of self.

EXAMPLES:

```python
sage: K.<sqrt2> = QuadraticField(2, name='sqrt2')
sage: sqrt2.floor()
1
sage: (-sqrt2).floor()
-2
sage: (13/197 + 3702/123*sqrt2).floor()
42
sage: (13/197 - 3702/123*sqrt2).floor()
-43
```
```python
>>> from sage.all import *
>>> K = QuadraticField(Integer(2), name='sqrt2', names=('sqrt2',)); (sqrt2,)
˓→ K._first_ngens(1)
>>> sqrt2.floor()
1
>>> (-sqrt2).floor()
-2
>>> (Integer(13)/Integer(197) + Integer(3702)/Integer(123)*sqrt2).floor()
42
>>> (Integer(13)/Integer(197) - Integer(3702)/Integer(123)*sqrt2).floor()
-43

**galois_conjugate()**

Return the image of this element under action of the nontrivial element of the Galois group of this field.

**EXAMPLES:**

```python
sage: K.<a> = QuadraticField(23)
sage: a.galois_conjugate()
-a
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 5*x + 1)
sage: a.galois_conjugate()
-a + 5
sage: b = 5*a + 1/3
sage: b.galois_conjugate()
-5*a + 76/3
sage: b.norm() == b * b.galois_conjugate()
True
sage: b.trace() == b + b.galois_conjugate()
True
```

```python
>>> from sage.all import *
>>> K = QuadraticField(Integer(23), names=('a',)); (a,)
˓→ K._first_ngens(1)
>>> a.galois_conjugate()
-a
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x^2 - Integer(5)*x + Integer(1), names=('a',));
˓→ (a,)
˓→ K._first_ngens(1)
>>> a.galois_conjugate()
-a + 5
>>> b = Integer(5)*a + Integer(1)/Integer(3)
>>> b.galois_conjugate()
-5*a + 76/3
>>> b.norm() == b * b.galois_conjugate()
True
>>> b.trace() == b + b.galois_conjugate()
True
```

**imag()**

Return the imaginary part of self.

**EXAMPLES:**

```python
```
is_integer()  
Check whether this number field element is an integer.

See also:

• *is_rational()* to test if this element is a rational number
• `is_integral()` to test if this element is an algebraic integer

EXAMPLES:

```python
sage: K.<sqrt3> = QuadraticField(3)
sage: sqrt3.is_integer()
False
sage: (sqrt3 - 1/2).is_integer()
False
sage: K(0).is_integer()
True
sage: K(-12).is_integer()
True
sage: K(1/3).is_integer()
False
```

```python
>>> from sage.all import *
>>> K = QuadraticField(Integer(3), names=('sqrt3',)); (sqrt3,) = K._first_ngens(1)

>>> sqrt3.is_integer()
False
>>> (sqrt3 - Integer(1)/Integer(2)).is_integer()
False
```

`is_integral()`

Return whether this element is an algebraic integer.

`is_one()`

Check whether this number field element is 1.

EXAMPLES:

```python
sage: K = QuadraticField(-2)
sage: K(1).is_one()
True
sage: K(-1).is_one()
False
sage: K(2).is_one()
False
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(2))

>>> K(Integer(1)).is_one()
True
```

(continues on next page)
continues from previous page)

```python
False
>>> K(Integer(2)).is_one()
False
>>> K(Integer(0)).is_one()
False
>>> K(Integer(1)/Integer(2)).is_one()
False
>>> K.gen().is_one()
False
```

**is_rational()**

Check whether this number field element is a rational number.

See also:

- `is_integer()` to test if this element is an integer
- `is_integral()` to test if this element is an algebraic integer

**EXAMPLES:**

```python
sage: K.<sqrt3> = QuadraticField(3)
sage: sqrt3.is_rational()
False
sage: (sqrt3 - 1/2).is_rational()
False
sage: K(0).is_rational()
True
sage: K(-12).is_rational()
True
sage: K(1/3).is_rational()
True
```

```python
>>> from sage.all import *
```

```python
>>> K = QuadraticField(Integer(3), names=('sqrt3',)); (sqrt3,) = K._first_ngens(1)
```

```python
>>> sqrt3.is_rational()
False
>>> (sqrt3 - Integer(1)/Integer(2)).is_rational()
False
>>> K(Integer(0)).is_rational()
True
>>> K(-Integer(12)).is_rational()
True
>>> K(Integer(1)/Integer(3)).is_rational()
True
```

**minpoly**(var='x', algorithm=None)

The minimal polynomial of this element over \( \mathbb{Q} \).

**INPUT:**

- `var` – the minimal polynomial is defined over a polynomial ring in a variable with this name. If not specified, this defaults to 'x'
- `algorithm` – for compatibility with general number field elements: and ignored

**EXAMPLES:**
norm (K=None)

Return the norm of self.

If the second argument is None, this is the norm down to \( \mathbb{Q} \). Otherwise, return the norm down to \( K \) (which had better be either \( \mathbb{Q} \) or this number field).

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - x + 3)
sage: a.norm()
3
sage: a.matrix()
[ 0 1]
[-3 1]
sage: K.<a> = NumberField(x^2 + 5)
sage: (1 + a).norm()
6
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - x + Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> a.norm()
3
>>> a.matrix()
[ 0 1]
[-3 1]
>>> K = NumberField(x**Integer(2) + Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> (Integer(1) + a).norm()
6
```

The norm is multiplicative:

```
sage: K.<a> = NumberField(x^2 - 3)
sage: a.norm()
(continues on next page)```
We test that the optional argument is handled sensibly:

```python
sage: (3*a).norm(QQ)
-27
sage: (3*a).norm(K)
3*a
sage: (3*a).norm(CyclotomicField(3))
Traceback (most recent call last):
  ... ValueError: no way to embed L into parent's base ring K
```

The `numerator()` method returns `self * self.denominator()`.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + x + 41)
sage: b = (2*a+1)/6
sage: b.numerator()
2*a + 1
```

(continues on next page)
parts()

Return a pair of rationals $a$ and $b$ such that $\text{self} = a + b\sqrt{D}$.

This is much closer to the internal storage format of the elements than the polynomial representation coefficients (the output of self.list()), unless the generator with which this number field was constructed was equal to $\sqrt{D}$. See the last example below.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 13)
sage: K.discriminant()
13
sage: a.parts()
(0, 1)
sage: (a/2 - 4).parts()
(-4, 1/2)
sage: K.<a> = NumberField(x^2 - 7)
sage: K.discriminant()
28
sage: a.parts()
(0, 1)
sage: a._coefficients()
[0, 1]
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(13), names=('a',)); (a,) = K._first_ngens(1)
>>> K.discriminant()
13
>>> a.parts()
(0, 1)
>>> (a/Integer(2) - Integer(4)).parts()
(-4, 1/2)
>>> K = NumberField(x**Integer(2) - Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> K.discriminant()
28
>>> a.parts()
(0, 1)
>>> K = NumberField(x**Integer(2) - x + Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> a.parts()
(1/2, 3/2)
>>> a._coefficients()
[0, 1]
```
Return the real part of self, which is either self (if self lives in a totally real field) or a rational number.

**EXAMPLES:**

```sage
sage: K.<sqrt2> = QuadraticField(2)
sage: sqrt2.real()
sqrt2
sage: K.<a> = QuadraticField(-3)
sage: a.real()
0
sage: (a + 1/2).real()
1/2
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + x + 1)
sage: a.real()
-1/2
sage: parent(a.real())
Rational Field
sage: K.<i> = QuadraticField(-1)
sage: i.real()
0
```

```python
>>> from sage.all import *
>>> K = QuadraticField(Integer(2), names=('sqrt2',)); (sqrt2,) = K._first_ngens(1)
>>> sqrt2.real()
sqrt2
>>> K = QuadraticField(-Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> a.real()
0
>>> (a + Integer(1)/Integer(2)).real()
1/2
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x^2 + x + Integer(1), names=('a',)); (a,) = K._first_ngens(1)
>>> a.real()
-1/2
>>> parent(a.real())
Rational Field
>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> i.real()
0
```

**round()**

Return the round (nearest integer) of this number field element. In case of ties, this relies on the default rounding for rational numbers.

**EXAMPLES:**

```sage
sage: K.<sqrt7> = QuadraticField(7, name='sqrt7')
sage: sqrt7.round()
3
sage: (-sqrt7).round()
-3
sage: (12/313*sqrt7 - 1745917/2902921).round()
0
```

(continues on next page)


from sage.all import *

K = QuadraticField(Integer(7), name='sqrt7', names=('sqrt7',)); (sqrt7,)

sqrt7.round()
3

(-sqrt7).round()
-3

(Integer(12)/Integer(313)*sqrt7 - Integer(1745917)/Integer(2902921)).round()
0

(Integer(12)/Integer(313)*sqrt7 - Integer(1745918)/Integer(2902921)).round()
-1

sign()

Returns the sign of self (0 if zero, +1 if positive, and -1 if negative).

EXAMPLES:

sage: K.<sqrt2> = QuadraticField(2, name='sqrt2')
sage: K(0).sign()
0
sage: sqrt2.sign()
1
sage: (sqrt2+1).sign()
1
sage: (sqrt2-1).sign()
1
sage: (sqrt2-2).sign()
-1
sage: (-sqrt2).sign()
-1
sage: (-sqrt2+1).sign()
-1
sage: (-sqrt2+2).sign()
1

sage: K.<a> = QuadraticField(2, embedding=-1.4142)
sage: K(0).sign()
0
sage: a.sign()
-1
sage: (a+1).sign()
-1
sage: (a+2).sign()
1
sage: (a-1).sign()
-1
sage: (-a).sign()
1
sage: (-a-1).sign()
1
sage: (-a-2).sign()
-1

(continues on next page)
sage: # needs sage.symbolic
sage: x = polygen(ZZ, 'x')

sage: K.<b> = NumberField(x^2 + 2*x + 7, 'b', embedding=CC(-1,-sqrt(6)))

sage: b.sign()
Traceback (most recent call last):
  ... ValueError: a complex number has no sign!

sage: K(1).sign()
1
sage: K(0).sign()
0
sage: K(-2/3).sign()
-1

>>> from sage.all import *

>>> K = QuadraticField(Integer(2), name='sqrt2', names=('sqrt2',)); (sqrt2,) = K._first_ngens(1)

>>> K(Integer(0)).sign()
0

>>> sqrt2.sign()
1

>>> (sqrt2+Integer(1)).sign()
1

>>> (sqrt2-Integer(1)).sign()
1

>>> (sqrt2-Integer(2)).sign()
-1

>>> (-sqrt2).sign()
-1

>>> (-sqrt2+Integer(1)).sign()
-1

>>> (-sqrt2+Integer(2)).sign()
1

>>> K = QuadraticField(Integer(2), embedding=-RealNumber('1.4142'), names=('a',)); (a,) = K._first_ngens(1)

>>> K(Integer(0)).sign()
0

>>> a.sign()
-1

>>> (a+Integer(1)).sign()
-1

>>> (a+Integer(2)).sign()
1

>>> (a-Integer(1)).sign()
-1

>>> (-a).sign()
1

>>> (-a-Integer(1)).sign()
1

>>> (-a-Integer(2)).sign()
-1

>>> # needs sage.symbolic

>>> x = polygen(ZZ, 'x')

(continues on next page)
trace()

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + x + 41)
sage: a.trace()
-1
sage: a.matrix()
[ 0 1]
[-41 -1]
```

```
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + x + Integer(41), names=('a',)); (a,) = K._first_ngens(1)
>>> a.trace()
-1
>>> a.matrix()
[ 0 1]
[-41 -1]
```

The trace is additive:

```
sage: K.<a> = NumberField(x^2 + 7)
sage: (a + 1).trace()
2
sage: K(3).trace()
6
sage: (a + 4).trace()
8
sage: (a/3 + 1).trace()
2
```

```
>>> from sage.all import *

>>> K = NumberField(x**Integer(2) + Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> (a + Integer(1)).trace()
2
>>> K(Integer(3)).trace()
6
>>> (a + Integer(4)).trace()
8
```
class sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic_sqrt

Bases: NumberFieldElement_quadratic

A \texttt{NumberFieldElement\_quadratic\_sqrt} object gives an efficient representation of an element of a quadratic extension of \(\mathbb{Q}\) for the case when \texttt{is\_sqrt\_disc()} is True.

denominator()

Return the denominator of \texttt{self}.

This is the LCM of the denominators of the coefficients of \texttt{self}, and thus it may well be > 1 even when the element is an algebraic integer.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + x + 41)
sage: a.denominator()
1
sage: b = (2*a+1)/6
sage: b.denominator()
6
sage: K(1).denominator()
1
sage: K(1/2).denominator()
2
sage: K(0).denominator()
1
sage: K.<a> = NumberField(x^2 - 5)
sage: b = (a + 1)/2
sage: b.denominator()
2
sage: b.is_integral()
True
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + x + Integer(41), names=('a',)); (a,) = K._first_ngens(1)
>>> a.denominator()
1
>>> b = (Integer(2)*a+Integer(1))/Integer(6)
>>> b.denominator()
6
>>> K(Integer(1)).denominator()
1
>>> K(Integer(1)/Integer(2)).denominator()
2
>>> K(Integer(0)).denominator()
1
>>> K = NumberField(x**Integer(2) - Integer(5), names=('a',)); (a,) = K._first_ngens(1)
(continues on next page)
```python
>>> b = (a + Integer(1))/Integer(2)
>>> b.denominator()
2
>>> b.is_integral()
True
```

class `sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic`

Bases: `NumberFieldElement_quadratic`

Element of an order in a quadratic field.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 1)
sage: O2 = K.order(2*a)
sage: w = O2.1; w
2*a
sage: parent(w)
Order of conductor 2 generated by 2*a in Number Field in a with defining polynomial x^2 + 1
```

```
from sage.all import *
```

```python
>>> x = polygen(ZZ, 'x')
```

```python
>>> K = NumberField(x**Integer(2) + Integer(1), names=(a,)); (a,) = K._first_ngens(1)
>>> O2 = K.order(Integer(2)*a)
>>> w = O2.gen(1); w
2*a
>>> parent(w)
Order of conductor 2 generated by 2*a in Number Field in a with defining polynomial x^2 + 1
```

**charpoly** *(var='x', algorithm=None)*

The characteristic polynomial of this element, which is over \( \mathbb{Z} \) because this element is an algebraic integer.

**INPUT:**

- **var** – the minimal polynomial is defined over a polynomial ring in a variable with this name. If not specified, this defaults to 'x'
- **algorithm** – for compatibility with general number field elements; ignored

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 5)
sage: R = K.ring_of_integers()
sage: b = R((5+a)/2)
sage: f = b.charpoly('x'); f
x^2 - 5*x + 5
sage: f.parent()
Univariate Polynomial Ring in x over Integer Ring
```

2.2. Elements optimized for quadratic number fields 357
denominator()

Return the denominator of self.

This is the LCM of the denominators of the coefficients of self, and thus it may well be > 1 even when the element is an algebraic integer.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 27)
sage: R = K.ring_of_integers()
sage: aa = R.gen(1)
sage: aa.denominator()
3
```

inverse_mod(I)

Return an inverse of self modulo the given ideal.

INPUT:

- I -- may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A ValueError is raised if I is non-integral or is zero. A ZeroDivisionError is raised if I + (x) ≠ (1).

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: OE.<w> = EquationOrder(x^2 - x + 2)
sage: w.inverse_mod(13) == 6*w - 6
True
sage: w*(6*w - 6) - 1
-13
sage: w.inverse_mod(13).parent() == OE
True
sage: w.inverse_mod(2*OE)
Traceback (most recent call last):
```

(continues on next page)
from sage.all import *

OE = EquationOrder(x**Integer(2) - x + Integer(2), names=('w',)); (w,) = OE._first_ngens(1)
w.inverse_mod(Integer(13)) == Integer(6)*w - Integer(6)
True
w*(Integer(6)*w - Integer(6)) - Integer(1)
-13
w.inverse_mod(Integer(13)).parent() == OE
True
w.inverse_mod(Integer(2)*OE)
Traceback (most recent call last):
...
ZeroDivisionError: w is not invertible modulo Fractional ideal (2)

minpoly(var='x', algorithm=None)

The minimal polynomial of this element over \( \mathbb{Z} \).

INPUT:

- **var** – the minimal polynomial is defined over a polynomial ring
  in a variable with this name. If not specified, this defaults to 'x'
- **algorithm** – for compatibility with general number field elements; ignored

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 163)
sage: R = K.ring_of_integers()
sage: f = R(a).minpoly('x'); f
x^2 + 163
sage: f.parent()
Univariate Polynomial Ring in x over Integer Ring

sage: K.<a> = NumberField(x^2 + 163, names=('a',)); (a,) = K._first_ngens(1)

norm()

The norm of an element of the ring of integers is an Integer.

EXAMPLES:
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 3)
sage: O2 = K.order(2*a)
sage: w = O2.gen(1); w
2*a
sage: w.norm()

12
sage: parent(w.norm())
Integer Ring

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> O2 = K.order(Integer(2)*a)
>>> w = O2.gen(Integer(1)); w
2*a
>>> w.norm()
12
>>> parent(w.norm())
Integer Ring

trace()

The trace of an element of the ring of integers is an Integer.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = QuadraticField(-3)
sage: f = K.coerce_map_from(QQ); f

(continues on next page)
Natural morphism:
  From: Rational Field
  To:  Number Field in a with defining polynomial x^2 + 3 with a = 1.
  \Rightarrow 732050807568878?*I
sage: f(3/1)
3
sage: f(1/2).parent() is K
True

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> f = K.coerce_map_from(QQ); f
Natural morphism:
  From: Rational Field
  To:  Number Field in a with defining polynomial x^2 + 3 with a = 1.
  \Rightarrow 732050807568878?*I
>>> f(Integer(3)/Integer(1))
3
>>> f(Integer(1)/Integer(2)).parent() is K
True
```

class sage.rings.number_field.number_field_element_quadratic.
Z_to_quadratic_field_element
Bases: Morphism

Morphism that coerces from integers to elements of a quadratic number field \( K \).

EXAMPLES:

```python
sage: K.<a> = QuadraticField(3)
sage: phi = K.coerce_map_from(ZZ); phi
Natural morphism:
  From: Integer Ring
  To:  Number Field in a with defining polynomial x^2 - 3 with a = 1.
  \Rightarrow 732050807568878?
sage: phi(4)
4
sage: phi(5).parent() is K
True
```

```python
>>> from sage.all import *
>>> K = QuadraticField(Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> phi = K.coerce_map_from(ZZ); phi
Natural morphism:
  From: Integer Ring
  To:  Number Field in a with defining polynomial x^2 - 3 with a = 1.
  \Rightarrow 732050807568878?
>>> phi(Integer(4))
4
>>> phi(Integer(5)).parent() is K
True
```

sage.rings.number_field.number_field_element_quadratic.is_sqrt_disc(ad, bd)

Return True if the pair \((ad, bd)\) is \( \sqrt{D} \).

EXAMPLES:
2.3 Elements of bounded height in number fields

This module provides functions to list all elements of a given number field with height less than a specified bound.

REFERENCES:

• [DK2013]

AUTHORS:

• John Doyle, David Krumm (2013): initial version

• TJ Combs, Raghukul Raman (2018): added Doyle-Krumm algorithm-4

sage.rings.number_field.bdd_height.bdd_height(K, height_bound, tolerance=0.01, precision=53)

Compute all elements in the number field $K$ which have relative multiplicative height at most $height\_bound$.

The function can only be called for number fields $K$ with positive unit rank. An error will occur if $K$ is $\mathbb{Q}$ or an imaginary quadratic field.

This algorithm computes 2 lists: $L$, containing elements $x$ in $K$ such that $H_k(x) \leq B$, and a list $L'$ containing elements $x$ in $K$ that, due to floating point issues, may be slightly larger than the bound. This can be controlled by lowering the tolerance.

In current implementation both lists $(L, L')$ are merged and returned in form of iterator.

ALGORITHM:

This is an implementation of the revised algorithm (Algorithm 4) in [DK2013].

INPUT:

• height_bound -- real number

• tolerance -- (default: 0.01) a rational number in (0,1]

• precision -- (default: 53) positive integer

OUTPUT:

an iterator of number field elements

EXAMPLES:

There are no elements of negative height:
The only nonzero elements of height 1 are the roots of unity:

The only nonzero elements of height 1 are the roots of unity:
sage.rings.number_field.bdd_height.bdd_height_iq(K, height_bound)

Compute all elements in the imaginary quadratic field $K$ which have relative multiplicative height at most $height\_bound$.

The function will only be called with $K$ an imaginary quadratic field.

If called with $K$ not an imaginary quadratic, the function will likely yield incorrect output.

ALGORITHM:
This is an implementation of Algorithm 5 in [DK2013].

INPUT:
• $K$ – an imaginary quadratic number field
• $height\_bound$ – a real number

OUTPUT:
• an iterator of number field elements

EXAMPLES:

```python
>>> from sage.all import *
>>> from sage.rings.number_field.bdd_height import bdd_height
>>> K = NumberField(x**Integer(4) - x**Integer(3) - Integer(3)*x**Integer(2) + x... + Integer(1), names=('g',)); (g,) = K._first_ngens(1)
>>> len(list(bdd_height(K, Integer(10))))
99
```

Continues on next page...

```python
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 191)
sage: for t in bdd_height_iq(K, 8):
    print(exp(2*t.global_height()))
1.00000000000000
1.00000000000000
1.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
```

Continues on next page...

```python
>>> from sage.all import *
>>> from sage.rings.number_field.bdd_height import bdd_height_iq
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(191), names=('a',)); (a,) = K._first_ngens(1)
>>> for t in bdd_height_iq(K, Integer(8)):
...    print(exp(Integer(2)*t.global_height()))
1.00000000000000
(continues on next page)
```
There are 175 elements of height at most 10 in \( QQ(\sqrt{3}) \):

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + 3)
sage: len(list(bdd_height_iq(K,10)))
175
```

The only elements of multiplicative height 1 in a number field are 0 and the roots of unity:

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + x + 1)
sage: list(bdd_height_iq(K,1))
[0, a + 1, a, -1, -a - 1, -a, 1]
```

A number field has no elements of multiplicative height less than 1:

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + 5)
sage: list(bdd_height_iq(K,0.9))
[]
```

2.3. Elements of bounded height in number fields
sage.rings.number_field.bdd_height.bdd_norm_pr_gens_iq($K$, $\text{norm\_list}$)

Compute generators for all principal ideals in an imaginary quadratic field $K$ whose norms are in $\text{norm\_list}$.  
The only keys for the output dictionary are integers $n$ appearing in $\text{norm\_list}$.  
The function will only be called with $K$ an imaginary quadratic field.  
The function will return a dictionary for other number fields, but it may be incorrect.

INPUT:

- $K$ – an imaginary quadratic number field
- $\text{norm\_list}$ – a list of positive integers

OUTPUT:

- a dictionary of number field elements, keyed by norm

EXAMPLES:

In $\mathbb{Q}(i)$, there is one principal ideal of norm 4, two principal ideals of norm 5, but no principal ideals of norm 7:

```python
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens_iq
sage: x = polygen(ZZ, 'x')

sage: K.<g> = NumberField(x^2 + 1)
sage: L = range(10)

sage: bdd_pr_ideals = bdd_norm_pr_gens_iq(K, L)

sage: bdd_pr_ideals[4][2]
sage: [-g - 2, -g + 2]

sage: bdd_pr_ideals[5][2]
sage: [-g - 2, -g + 2]

sage: bdd_pr_ideals[7][2]
sage: []
```

There are no ideals in the ring of integers with negative norm:

```python
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens_iq
sage: x = polygen(ZZ, 'x')

sage: K.<g> = NumberField(x^2 + 10)

sage: L = range(-5,-1)

sage: bdd_pr_ideals = bdd_norm_pr_gens_iq(K, L)

sage: bdd_pr_ideals[-5][2]
sage: []

sage: bdd_pr_ideals[-4][2]
sage: []

sage: bdd_pr_ideals[-3][2]
sage: []

sage: bdd_pr_ideals[-2][2]
sage: []
```

(continues on next page)
Calling a key that is not in the input norm_list raises a KeyError:

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens
sage: K.<g> = NumberField(x^2 + 20)
sage: L = range(100)
sage: bdd_pr_ideals = bdd_norm_pr_gens(K, L)
sage: bdd_pr_ideals[100]
Traceback (most recent call last):
  ...  
KeyError: 100
```

```
sage.rings.number_field.bdd_height.bdd_norm_pr_ideal_gens(K, norm_list)

Compute generators for all principal ideals in a number field \( K \) whose norms are in \( \text{norm_list} \).

INPUT:

- \( K \) – a number field
- \( \text{norm_list} \) – a list of positive integers

OUTPUT:

- a dictionary of number field elements, keyed by norm

EXAMPLES:

There is only one principal ideal of norm 1, and it is generated by the element 1:

```
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
sage: K.<g> = QuadraticField(101)
sage: bdd_norm_pr_ideal_gens(K, [1])
{1: [1]}
```

```
sage: from sage.all import *
>>> from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
>>> from sage.rings.number_field.bdd_height import bdd_norm_pr_gens
>>> K = QuadraticField(Integer(101), names=('g',)); (g,) = K._first_ngens(1)
>>> bdd_norm_pr_ideal_gens(K, [Integer(1)])
{1: [1]}
```

2.3. Elements of bounded height in number fields
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
sage: K.<g> = QuadraticField(123)
sage: bdd_norm_pr_ideal_gens(K, range(5))
{0: [0], 1: [1], 2: [g + 11], 3: [], 4: [2]}

>>> from sage.all import *
>>> from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
>>> K = QuadraticField(Integer(123), names=('g',)); (g,) = K._first_ngens(1)
>>> bdd_norm_pr_ideal_gens(K, range(Integer(5)))
{0: [0], 1: [1], 2: [g + 11], 3: [], 4: [2]}

sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
sage: x = polygen(ZZ, 'x')
sage: K.<g> = NumberField(x^5 - x + 19)
sage: b = bdd_norm_pr_ideal_gens(K, range(30))
sage: key = ZZ(28)
sage: b[key]
[(157*g^4 - 139*g^3 - 369*g^2 + 848*g + 158, g^4 + g^3 - g - 7)]

>>> from sage.all import *
>>> from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal_gens
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(5) - x + Integer(19), names=('g',)); (g,) = K._first_ngens(1)
>>> b = bdd_norm_pr_ideal_gens(K, range(Integer(30)))
>>> key = ZZ(Integer(28))
>>> b[key]
[(157*g^4 - 139*g^3 - 369*g^2 + 848*g + 158, g^4 + g^3 - g - 7)]

sage.rings.number_field.bdd_height.integer_points_in_polytope(matrix, interval_radius)

Return the set of integer points in the polytope obtained by acting on a cube by a linear transformation.

Given an $r$-by-$r$ matrix $\text{matrix}$ and a real number $\text{interval_radius}$, this function finds all integer lattice points in the polytope obtained by transforming the cube $[-\text{interval_radius}, \text{interval_radius}]^r$ via the linear map induced by matrix.

INPUT:

- matrix -- a square matrix of real numbers
- interval_radius -- a real number

OUTPUT:

- a list of tuples of integers

EXAMPLES:

Stretch the interval $[-1,1]$ by a factor of 2 and find the integers in the resulting interval:
Integer points inside a parallelogram:

```python
>>> from sage.rings.number_field.bdd_height import integer_points_in_polytope
>>> m = matrix([[Integer(2)]])
>>> r = Integer(1)
>>> integer_points_in_polytope(m, r)
[(-2), (-1), (0), (1), (2)]
```

```python
Integer points inside a parallelepiped:
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.bdd_height import integer_points_in_polytope
>>> m = matrix([[Integer(1), Integer(2), Integer(3), Integer(7)],
              [Integer(4), Integer(5), Integer(6), Integer(2)],
              [Integer(7), Integer(8), Integer(9), Integer(3)],
              [Integer(0), Integer(3), Integer(4), Integer(5)]]).
>>> integer_points_in_polytope(m, Integer(0))
[(0, 0, 0, 0)]
```

2.3. Elements of bounded height in number fields
3.1 Morphisms between number fields

This module provides classes to represent ring homomorphisms between number fields (i.e. field embeddings).

```python
class sage.rings.number_field.morphism.CyclotomicFieldHomomorphism_im gens
    Bases: NumberFieldHomomorphism_im gens

class sage.rings.number_field.morphism.NumberFieldHomomorphism_im gens
    Bases: RingHomomorphism_im gens

preimage(y)
    Compute a preimage of \( y \) in the domain, provided one exists. Raises a \( \text{ValueError} \) if \( y \) has no preimage.
    
    INPUT:
    
    • \( y \) – an element of the codomain of self.
    
    OUTPUT:
    
    Returns the preimage of \( y \) in the domain, if one exists. Raises a \( \text{ValueError} \) if \( y \) has no preimage.
    
    EXAMPLES:
```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 7)
sage: L.<b> = NumberField(x^4 - 7)
sage: f = K.embeddings(L)[0]
sage: f.preimage(3*b^2 - 12/7)
3*a - 12/7
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> L = NumberField(x**Integer(4) - Integer(7), names=('b',)); (b,) = L._first_ngens(1)
>>> f = K.embeddings(L)[Integer(0)]
>>> f.preimage(Integer(3)*b**Integer(2) - Integer(12)/Integer(7))
3*a - 12/7
>>> f.preimage(b)
```

(continues on next page)
Traceback (most recent call last):
...
ValueError: Element 'b' is not in the image of this homomorphism.

 sage: # needs sage.libs.linbox
 sage: F.<b> = QuadraticField(23)
 sage: G.<a> = F.extension(x^3 + 5)
 sage: f = F.embeddings(G)[0]
 sage: f.preimage(a^3 + 2*b + 3)
 2*b - 2

>>> from sage.all import *
>>> # needs sage.libs.linbox
>>> F = QuadraticField(Integer(23), names=('b',)); (b,) = F._first_ngens(1)
>>> G = F.extension(x**Integer(3) + Integer(5), names=('a',)); (a,) = G._
˓→first_ngens(1)
>>> f = F.embeddings(G)[Integer(0)]
>>> f.preimage(a**Integer(3) + Integer(2)*b + Integer(3))
2*b - 2

class sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs(par-
˓→ent, abs_hom)

Bases: RingHomomorphism

A homomorphism from a relative number field to some other ring, stored as a homomorphism from the correspond-
˓→ing absolute field.

abs_hom()  
Return the corresponding homomorphism from the absolute number field.

EXAMPLES:

 sage: x = polygen(ZZ, 'x')
 sage: K.<a, b> = NumberField([x^3 + 2, x^2 + x + 1])
 sage: K.hom(a, K).abs_hom()
 Ring morphism:
   From: Number Field in a with defining polynomial
   x^6 - 3*x^5 + 6*x^4 - 3*x^3 - 9*x + 9
   To:   Number Field in a with defining polynomial x^3 + 2 over its base field
   Defn: a |--> a - b

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(3) + Integer(2), x**Integer(2) + x +
˓→Integer(1)], names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> K.hom(a, K).abs_hom()
 Ring morphism:
   From: Number Field in a with defining polynomial
   x^6 - 3*x^5 + 6*x^4 - 3*x^3 - 9*x + 9
   To:   Number Field in a with defining polynomial x^3 + 2 over its base field
   Defn: a |--> a - b

im_gens()  
Return the images of the generators under this map.

EXAMPLES:
3.2 Sets of homomorphisms between number fields

class sage.rings.number_field.homset.CyclotomicFieldHomset (R, S, category=None)
Bases: NumberFieldHomset
Set of homomorphisms with domain a given cyclotomic field.

EXAMPLES:

sage: End(CyclotomicField(16))
Automorphism group of Cyclotomic Field of order 16 and degree 8

>>> from sage.all import *
>>> End(CyclotomicField(Integer(16)))
Automorphism group of Cyclotomic Field of order 16 and degree 8

Element
alias of CyclotomicFieldHomomorphism_im_gens

list()
Return a list of all the elements of self (for which the domain is a cyclotomic field).

EXAMPLES:

sage: K.<z> = CyclotomicField(12)
sage: G = End(K); G
Automorphism group of Cyclotomic Field of order 12 and degree 4

sage: [g(z) for g in G]
[z, z^3 - z, -z, -z^3 + z]
sage: x = polygen(ZZ, 'x')
sage: L.<a, b> = NumberField([x^2 + x + 1, x^4 + 1])
sage: L
Number Field in a with defining polynomial x^2 + x + 1 over its base field
sage: Hom(CyclotomicField(12), L)[3]
Ring morphism:
  From: Cyclotomic Field of order 12 and degree 4
  To:  Number Field in a with defining polynomial x^2 + x + 1 over its base_field
  Defn: zeta12 |--> -b^2*a
sage: list(Hom(CyclotomicField(5), K))
[]
sage: Hom(CyclotomicField(11), L).list()
[]
>>> from sage.all import *
>>> K = CyclotomicField(Integer(12), names=('z',)); (z,) = K._first_ngens(1)
>>> G = End(K); G
Automorphism group of Cyclotomic Field of order 12 and degree 4

>>> [g(z) for g in G]
z, z^3 - z, -z, -z^3 + z

>>> x = polygen(ZZ, 'x')
>>> L = NumberField([x**Integer(2) + x + Integer(1), x**Integer(4) +...
˓→Integer(1)], names=('a', 'b',)); (a, b,) = L._first_ngens(2)

>>> L
Number Field in a with defining polynomial x^2 + x + 1 over its base field

>>> Hom(CyclotomicField(Integer(12)), L)[Integer(3)]
Ring morphism:
  From: Cyclotomic Field of order 12 and degree 4
  To:   Number Field in a with defining polynomial x^2 + x + 1 over its base field
  Defn: zeta12 |--> -b^2*a

>>> list(Hom(CyclotomicField(Integer(5)), K))
[]

>>> list(Hom(CyclotomicField(Integer(11)), L))
[]

class sage.rings.number_field.homset.NumberFieldHomset (R, S, category=None)

Set of homomorphisms with domain a given number field.

Element

alias of NumberFieldHomomorphism_im_gens

cardinality()

Return the order of this set of field homomorphism.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^2 + 1)
sage: End(k)
Automorphism group of Number Field in a with defining polynomial x^2 + 1

sage: End(k).order()
2

sage: k.<a> = NumberField(x^3 + 2)
sage: End(k).order()
1

sage: K.<a> = NumberField([x^3 + 2, x^2 + x + 1])
sage: End(K).order()
6

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = k._
˓→first_ngens(1)

>>> End(k)
Automorphism group of Number Field in a with defining polynomial x^2 + 1

>>> End(k).order()
2

(continues on next page)
list()

Return a list of all the elements of self.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 3*x + 1)
sage: End(K).list()
[Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
  Defn: a |--> a,
  Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
  Defn: a |--> a^2 - 2,
  Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
  Defn: a |--> -a^2 - a + 2]
sage: Hom(K, CyclotomicField(9))[0] # indirect doctest
Ring morphism:
  From: Number Field in a with defining polynomial x^3 - 3*x + 1
  To:   Cyclotomic Field of order 9 and degree 6
  Defn: a |--> -zeta9^4 + zeta9^2 - zeta9
```

An example where the codomain is a relative extension:

```python
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<b> = K.extension(x^2 + 3)
sage: Hom(K, L).list()
```
Ring morphism:
  From: Number Field in a with defining polynomial \(x^3 - 2\)
  To:   Number Field in b with defining polynomial \(x^2 + 3\) over its base field
  Defn: \(a \mapsto a\),

Ring morphism:
  From: Number Field in a with defining polynomial \(x^3 - 2\)
  To:   Number Field in b with defining polynomial \(x^2 + 3\) over its base field
  Defn: \(a \mapsto -1/2*a*b - 1/2*a\),

Ring morphism:
  From: Number Field in a with defining polynomial \(x^3 - 2\)
  To:   Number Field in b with defining polynomial \(x^2 + 3\) over its base field
  Defn: \(a \mapsto 1/2*a*b - 1/2*a\)

>>> from sage.all import *

>>> K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._
               \text{first_ngens}(1)

>>> L = K.extension(x**Integer(2) + Integer(3), names=('b',)); (b,) = L._
                 \text{first_ngens}(1)

>>> Hom(K, L).list()

\[
\begin{align*}
\text{Ring morphism:} & \\
\text{From: Number Field in a with defining polynomial } x^3 - 2 & \\
\text{To: Number Field in b with defining polynomial } x^2 + 3 \text{ over its base field} & \\
\text{Defn: } a \mapsto a, \\
\text{Ring morphism:} & \\
\text{From: Number Field in a with defining polynomial } x^3 - 2 & \\
\text{To: Number Field in b with defining polynomial } x^2 + 3 \text{ over its base field} & \\
\text{Defn: } a \mapsto -1/2*a*b - 1/2*a, \\
\text{Ring morphism:} & \\
\text{From: Number Field in a with defining polynomial } x^3 - 2 & \\
\text{To: Number Field in b with defining polynomial } x^2 + 3 \text{ over its base field} & \\
\text{Defn: } a \mapsto 1/2*a*b - 1/2*a
\end{align*}
\]

order()

Return the order of this set of field homomorphism.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^2 + 1)
sage: End(k)
Automorphism group of Number Field in a with defining polynomial x^2 + 1
sage: End(k).order()
2
sage: k.<a> = NumberField(x^3 + 2)
sage: End(k).order()
1
sage: K.<a> = NumberField([x^3 + 2, x^2 + x + 1])
sage: End(K).order()
6
```
```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = k._first_ngens(1)
>>> End(k)
Automorphism group of Number Field in a with defining polynomial x^2 + 1
>>> End(k).order()
2
>>> k = NumberField(x**Integer(3) + Integer(2), names=('a',)); (a,) = k._first_ngens(1)
>>> End(k).order()
1
>>> K = NumberField([x**Integer(3) + Integer(2), x**Integer(2) + x + Integer(1)], names=('a',)); (a,) = K._first_ngens(1)
>>> End(K).order()
6
```

**class sage.rings.number_field.homset.RelativeNumberFieldHomset (R, S, category=None)**

Set of homomorphisms with domain a given relative number field.

**EXAMPLES:**

We construct a homomorphism from a relative field by giving the image of a generator:

```python
sage: x = polygen(ZZ, 'x')
sage: L.<cuberoot2, zeta3> = CyclotomicField(3).extension(x^3 - 2)
sage: phi = L.hom([cuberoot2 * zeta3]); phi
Relative number field endomorphism of
    Number Field in cuberoot2 with defining polynomial x^3 - 2 over its base field
    Defn: cuberoot2 |--> zeta3*cuberoot2
    zeta3 |--> zeta3
sage: phi(cuberoot2 + zeta3)
zeta3*cuberoot2 + zeta3
```

In fact, this phi is a generator for the Kummer Galois group of this cyclic extension:

```python
sage: phi(phi(cuberoot2 + zeta3))
(-zeta3 - 1)*cuberoot2 + zeta3
sage: phi(phi(phi(cuberoot2 + zeta3)))
cuberoot2 + zeta3
```

3.2. Sets of homomorphisms between number fields
>>> from sage.all import *
>>> phi(phi(cuberoot2 + zeta3))
(-zeta3 - 1)*cuberoot2 + zeta3
>>> phi(phi(phi(cuberoot2 + zeta3)))
cuberoot2 + zeta3

Element

alias of RelativeNumberFieldHomomorphism_from_abs
default_base_hom()

Pick an embedding of the base field of self into the codomain of this homset. This is done in an essentially arbitrary way.

EXAMPLES:

sage: x = polygen(ZZ, x)
sage: L.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
sage: M.<c> = NumberField(x^4 + 80*x^2 + 36)
sage: Hom(L, M).default_base_hom()
Ring morphism:
  From: Number Field in b with defining polynomial x^2 + 23
  To:   Number Field in c with defining polynomial x^4 + 80*x^2 + 36
  Defn: b |--> 1/12*c^3 + 43/6*c

sage: x = polygen(ZZ, x)
sage: L = NumberField([x**Integer(3) - x + Integer(1), x**Integer(2) +
˓→Integer(23)], names=(a, b,)); (a, b,) = L._first_ngens(2)
sage: M = NumberField(x**Integer(4) + Integer(80)*x**Integer(2) + Integer(36),␣
˓→names=('c',)); (c,) = M._first_ngens(1)
sage: Hom(L, M).default_base_hom()
Ring morphism:
  From: Number Field in b with defining polynomial x^2 + 23
  To:   Number Field in c with defining polynomial x^4 + 80*x^2 + 36
  Defn: b |--> 1/12*c^3 + 43/6*c

list()

Return a list of all the elements of self (for which the domain is a relative number field).

EXAMPLES:

sage: x = polygen(ZZ, x)
sage: K.<a, b> = NumberField([x^2 + x + 1, x^3 + 2])
sage: End(K).list()
[
  Relative number field endomorphism of
  Number Field in a with defining polynomial x^2 + x + 1 over its base field
  Defn: a |--> a
  b |--> b,
  ...
  Relative number field endomorphism of
  Number Field in a with defining polynomial x^2 + x + 1 over its base field
  Defn: a |--> a
  b |--> -b*a - b
]
An example with an absolute codomain:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 - 3, x^2 + 2])
sage: Hom(K, CyclotomicField(24, 'z')).list()
```

Relative number field morphism:
- From: Number Field in a with defining polynomial $x^2 - 3$ over its base field
- To: Cyclotomic Field of order 24 and degree 8
- Defn: $a \mapsto z^6 - 2z^2$
- $b \mapsto -z^5 - z^3 + z$

Relative number field morphism:
- From: Number Field in a with defining polynomial $x^2 - 3$ over its base field
- To: Cyclotomic Field of order 24 and degree 8
- Defn: $a \mapsto -z^6 + 2z^2$
- $b \mapsto z^5 + z^3 - z$

3.2. Sets of homomorphisms between number fields
3.3 Embeddings into ambient fields

This module provides classes to handle embeddings of number fields into ambient fields (generally \( \mathbb{R} \) or \( \mathbb{C} \)).

```
class sage.rings.number_field.number_field_morphisms.CyclotomicFieldConversion
    Bases: sage.rings.number_field.number_field_morphisms.Map

    This allows one to cast one cyclotomic field in another consistently.

    EXAMPLES:

    sage: from sage.rings.number_field.number_field_morphisms import CyclotomicFieldConversion
    sage: K1.<z1> = CyclotomicField(12)
    sage: K2.<z2> = CyclotomicField(18)
    sage: f = CyclotomicFieldConversion(K1, K2)
    sage: f(z1^2)
    z2^3
    sage: f(z1)
    Traceback (most recent call last):
    ... ValueError: Element z1 has no image in the codomain

>>> from sage.all import *
>>> from sage.rings.number_field.number_field_morphisms import CyclotomicFieldConversion
>>> K1 = CyclotomicField(Integer(12), names=(z1,)); (z1,) = K1._first_ngens(1)
>>> K2 = CyclotomicField(Integer(18), names=(z2,)); (z2,) = K2._first_ngens(1)
>>> f = CyclotomicFieldConversion(K1, K2)
>>> f(z1**Integer(2))
z2^3
>>> f(z1)
Traceback (most recent call last):
... ValueError: Element z1 has no image in the codomain

Tests from Issue #29511:

```sage
sage: K.<z> = CyclotomicField(12)
sage: K1.<z1> = CyclotomicField(3)
sage: K(2) in K1 # indirect doctest
True
sage: K1(K(2)) # indirect doctest
2
```

```
>>> from sage.all import *
>>> K = CyclotomicField(Integer(12), names=('z',)); (z,) = K._first_ngens(1)
>>> K1 = CyclotomicField(Integer(3), names=('z1',)); (z1,) = K1._first_ngens(1)
>>> K(Integer(2)) in K1 # indirect doctest
True
>>> K1(K(Integer(2))) # indirect doctest
2
```

```
class sage.rings.number_field.number_field_morphisms.CyclotomicFieldEmbedding
    Bases: sage.rings.number_field.number_field_morphisms.NumberFieldEmbedding

    Specialized class for converting cyclotomic field elements into a cyclotomic field of higher order. All the real work is done by _lift_cyclotomic_element().
```

section()
Return the section of self.

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_morphisms import...
    - CyclotomicFieldEmbedding
sage: K = CyclotomicField(7)
sage: L = CyclotomicField(21)
sage: f = CyclotomicFieldEmbedding(K, L)
sage: h = f.section()
sage: h(f(K.gen()))
# indirect doctest
zeta7
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.number_field_morphisms import...
>>> K = CyclotomicField(Integer(7))
>>> L = CyclotomicField(Integer(21))
>>> f = CyclotomicFieldEmbedding(K, L)
>>> h = f.section()
>>> h(f(K.gen()))
# indirect doctest
zeta7
```

class sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldConversion
Bases: Map
This allows one to cast one number field in another consistently, assuming they both have specified embeddings into an ambient field (by default it looks for an embedding into $\mathbb{C}$).

This is done by factoring the minimal polynomial of the input in the number field of the codomain. This may fail if the element is not actually in the given field.

ambient_field
class sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldMorphism
Bases: NumberFieldEmbedding
This allows one to go from one number field in another consistently, assuming they both have specified embeddings into an ambient field.

If no ambient field is supplied, then the following ambient fields are tried:

- the pushout of the fields where the number fields are embedded;
- the algebraic closure of the previous pushout;
- $\mathbb{C}$.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1, embedding=QQbar(I))
sage: L.<i> = NumberField(x^2 + 1, embedding=-QQbar(I))
sage: from sage.rings.number_field.number_field_morphisms import...
    - EmbeddedNumberFieldMorphism
sage: EmbeddedNumberFieldMorphism(K, L, CDF)
Generic morphism:
```

(continues on next page)
From: Number Field in i with defining polynomial $x^2 + 1$ with $i = I$
To: Number Field in i with defining polynomial $x^2 + 1$ with $i = -I$
Defn: $i \mapsto -i$

\[
\text{sage: } \text{EmbeddedNumberFieldMorphism}(K, L, \text{QQbar})
\]
Generic morphism:
From: Number Field in i with defining polynomial $x^2 + 1$ with $i = I$
To: Number Field in i with defining polynomial $x^2 + 1$ with $i = -I$
Defn: $i \mapsto -i$

\[
\text{sage: } \text{from sage.all import *}
\]
\[
\text{sage: } x = \text{polygen}(%Z, 'x')
\]
\[
\text{sage: } K = \text{NumberField}(x^2 + 1, \text{names}=('i',)); (i,) = K._first_ngens(1)
\]
\[
\text{sage: } L = \text{NumberField}(x^2 + 1, \text{embedding}=-\text{QQbar}(I), \text{names}=('i',))
\]
\[
\text{sage: } \text{EmbeddedNumberFieldMorphism}(K, L, \text{QQbar})
\]
Generic morphism:
From: Number Field in i with defining polynomial $x^2 + 1$ with $i = I$
To: Number Field in i with defining polynomial $x^2 + 1$ with $i = -I$
Defn: $i \mapsto -i$

ambient_field

section()
```python
>>> g = f.section()
>>> g(Integer(2)*b**Integer(3) - Integer(1))
2*a - 1
```

```python
class sage.rings.number_field.number_field_morphisms.NumberFieldEmbedding
    Bases: Morphism

    If R is a lazy field, the closest root to gen_embedding will be chosen.

    EXAMPLES:

    sage: x = polygen(QQ)
    sage: from sage.rings.number_field.number_field_morphisms import ...
    => NumberFieldEmbedding
    sage: K.<a> = NumberField(x^3-2)
    sage: f = NumberFieldEmbedding(K, RLF, 1)
    sage: f(a)**3
    2.0000000000000000000000000000000000000000000000000000000000
    sage: RealField(200)(f(a)**3)
    2.0000000000000000000000000000000000000000000000000000000000
    sage: sigma_a = K.polynomial().change_ring(CC).roots()[1][0]; sigma_a
    -0.62996052494743... - 1.09112363597172*I
    sage: g = NumberFieldEmbedding(K, CC, sigma_a)
    sage: g(a+1)
    0.37003947505256... - 1.09112363597172*I
```

```python
>>> from sage.all import *
```

```python
>>> x = polygen(QQ)
>>> from sage.rings.number_field.number_field_morphisms import ...
>>> K = NumberField(x**Integer(3)-Integer(2), names=('a',)); (a,) = K._first_...
>>> f(a)**Integer(3)
2.0000000000000000000000000000000000000000000000000000000000
```

```python
>>> sigma_a = K.polynomial().change_ring(CC).roots()[Integer(1)][Integer(0)]; ...
>>> g = NumberFieldEmbedding(K, CC, sigma_a)
>>> g(a+Integer(1))
0.37003947505256... - 1.09112363597172*I
```

```python
gen_image()

    Return the image of the generator under this embedding.

    EXAMPLES:

    sage: f = QuadraticField(7, 'a', embedding=2).coerce_embedding()
    sage: f.gen_image()
    2.645751311064591?
```

3.3. Embeddings into ambient fields 383
sage.rings.number_field.number_field_morphisms.closest(target, values, margin=1)

This is a utility function that returns the item in values closest to target (with respect to the abs function). If margin is greater than 1, and \( x \) and \( y \) are the first and second closest elements to target, then only return \( x \) if \( x \) is margin times closer to target than \( y \), i.e. margin * abs(target-x) < abs(target-y).

sage.rings.number_field.number_field_morphisms.create_embedding_from_approx(K, gen_image)

Return an embedding of \( K \) determined by \( \text{gen_image} \).

The codomain of the embedding is the parent of \( \text{gen_image} \) or, if \( \text{gen_image} \) is not already an exact root of the defining polynomial of \( K \), the corresponding lazy field. The embedding maps the generator of \( K \) to a root of the defining polynomial of \( K \) closest to \( \text{gen_image} \).

EXAMPLES:

```python
>>> from sage.rings.number_field.number_field_morphisms import create_embedding_from_approx
>>> x = polygen(ZZ, 'x')
>>> K.<a> = NumberField(x**Integer(3) - x + Integer(1)/Integer(10), names=('a',)); (a, ) = K._first_ngens(1)
>>> create_embedding_from_approx(K, Integer(1))
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To: Real Lazy Field
  Defn: a -> 0.9456492739235915?
>>> create_embedding_from_approx(K, Integer(0))
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To: Real Lazy Field
  Defn: a -> 0.10103125788101081?
>>> create_embedding_from_approx(K, -Integer(1))
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To: Real Lazy Field
  Defn: a -> -1.046680531804603?
```
We can define embeddings from one number field to another:

```python
sage: L.<b> = NumberField(x^6-x^2+1/10)
sage: create_embedding_from_approx(K, b^2)
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To:   Number Field in b with defining polynomial x^6 - x^2 + 1/10
  Defn: a -> b^2
```

If the embedding is exact, it must be valid:

```python
sage: create_embedding_from_approx(K, b)
Traceback (most recent call last):
  ... ValueError: b is not a root of x^3 - x + 1/10

>>> from sage.all import *
>>> L = NumberField(x**Integer(6)-x**Integer(2)+Integer(1)/Integer(10), names=('b',)); (b,) = L._first_ngens(1)
>>> create_embedding_from_approx(K, b**Integer(2))
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 1/10
  To:   Number Field in b with defining polynomial x^6 - x^2 + 1/10
  Defn: a -> b^2
```

sage.rings.number_field.number_field_morphisms.matching_root(poly, target, ambient_field=None, margin=1, max_prec=None)

Given a polynomial and a target, choose the root that target best approximates as compared in ambient_field.

If the parent of target is exact, the equality is required, otherwise find closest root (with respect to the abs function) in the ambient field to the target, and return the root of poly (if any) that approximates it best.

EXAMPLES:

```python
sage: from sage.rings.number_field.number_field_morphisms import matching_root
sage: R.<x> = CC[]
sage: matching_root(x^2-2, 1.5) 1.41421356237310
sage: matching_root(x^2-2, -100.0) -1.41421356237310
```
sage.rings.number_field.number_field_morphisms.root_from_approx(f, a)

Return an exact root of the polynomial $f$ closest to $a$.

**INPUT:**

- $f$ – polynomial with rational coefficients
- $a$ – element of a ring

**OUTPUT:**

A root of $f$ in the parent of $a$ or, if $a$ is not already an exact root of $f$, in the corresponding lazy field. The root is taken to be closest to $a$ among all roots of $f$.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.number_field_morphisms import root_from_approx
sage: R.<x> = QQ[]
sage: root_from_approx(x^2 - 1, -1)
-1
sage: root_from_approx(x^2 - 2, 1)
1.414213562373095?

sage: root_from_approx(x^3 - x - 1, RR(1))
1.324717957244746?

sage: root_from_approx(x^3 - x - 1, CC.gen())
-0.6623589786223730? + 0.5622795120623013?*I

sage: root_from_approx(x^2 + 1, 0)
Traceback (most recent call last):
  ... ValueError: x^2 + 1 has no real roots
sage: root_from_approx(x^2 + 1, CC(0))
-1*I

sage: root_from_approx(x^2 + 1, sqrt(2))  # needs sage.symbolic
```

(continues on next page)
sage: root_from_approx(x^2 - 2, sqrt(3))    # needs sage.symbolic
Traceback (most recent call last):
...
ValueError: sqrt(3) is not a root of x^2 - 2

3.4 Structure maps for number fields

This module provides isomorphisms between relative and absolute presentations, to and from vector spaces, name changing maps, etc.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: L.<cuberoot2, zeta3> = CyclotomicField(3).extension(x^3 - 2)
sage: K = L.absolute_field('a')
sage: from_K, to_K = K.structure()
sage: from_K
Isomorphism map:
From: Number Field in a with defining polynomial
x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1
To: Number Field in cuberoot2 with defining polynomial
x^3 - 2 over its base field
sage: to_K
```
Isomorphism map:
From: Number Field in cuberoot2 with defining polynomial
x^3 - 2 over its base field
To: Number Field in a with defining polynomial
x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> L = CyclotomicField(Integer(3)).extension(x**Integer(3) - Integer(2), names=(
˓→'cuberoot2', 'zeta3',)); (cuberoot2, zeta3,) = L._first_ngens(2)

>>> K = L.absolute_field('a')

>>> from_K, to_K = K.structure()

>>> from_K

Isomorphism map:
From: Number Field in a with defining polynomial
x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1
To: Number Field in cuberoot2 with defining polynomial
x^3 - 2 over its base field

>>> to_K

Isomorphism map:
From: Number Field in cuberoot2 with defining polynomial
x^3 - 2 over its base field
To: Number Field in a with defining polynomial
x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1
```

```python
class sage.rings.number_field.maps.MapAbsoluteToRelativeNumberField(A, R)

Bases: NumberFieldIsomorphism

See MapRelativeToAbsoluteNumberField for examples.

class sage.rings.number_field.maps.MapNumberFieldToVectorSpace(K, V)

Bases: Map

A class for the isomorphism from an absolute number field to its underlying Q-vector space.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: L.<a> = NumberField(x^3 - x + 1)
sage: V, fr, to = L.vector_space()
sage: type(to)
<class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>
```
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
sage: V, fr, to = K.relative_vector_space()
sage: type(to)
<class 'sage.rings.number_field.maps.MapRelativeNumberFieldToRelativeVectorSpace'>

>>> from sage.all import *
>>>
>>> x = polygen(ZZ, 'x')
>>>
>>> K = NumberField([x**Integer(3) - x + Integer(1), x**Integer(2) + Integer(23)],
˓→ names=('a', 'b'),); (a, b) = K._first_ngens(2)
>>> V, fr, to = K.relative_vector_space()
>>> type(to)
<class 'sage.rings.number_field.maps.MapRelativeNumberFieldToRelativeVectorSpace'>

class sage.rings.number_field.maps.MapRelativeNumberFieldToVectorSpace (L, V, to_K, to_V)

Bases: NumberFieldIsomorphism

The isomorphism from a relative number field to its underlying \( \mathbb{Q} \)-vector space. Compare `MapRelativeNumberFieldToRelativeVectorSpace`.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^8 + 100*x^6 + x^2 + 5)
sage: L = K.relativize(K.subfields(Integer(4))[Integer(0)][Integer(1)], b); L
Number Field in b with defining polynomial x^2 + a0 over its base field
sage: L_to_K, K_to_L = L.structure()
sage: V, fr, to = L.absolute_vector_space()
sage: V
Vector space of dimension 8 over Rational Field
sage: fr
Isomorphism map:
    From: Vector space of dimension 8 over Rational Field
    To:    Number Field in b with defining polynomial x^2 + a0 over its base field
sage: to
Isomorphism map:
    From: Number Field in b with defining polynomial x^2 + a0 over its base field
    To:    Vector space of dimension 8 over Rational Field
sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField'>,
 <class 'sage.rings.number_field.maps.MapRelativeNumberFieldToVectorSpace'>)

sage: v = V([1, 1, 1, 1, 0, 1, 1, 1])
sage: fr(v), to(fr(v)) == v
((-a0^3 + a0^2 - a0 + 1)*b - a0^3 - a0 + 1, True)

>>> from sage.all import *
>>>
>>> x = polygen(ZZ, 'x')
>>>
>>> K = NumberField([x**Integer(8) + Integer(100)*x**Integer(6) + x**Integer(2) +
˓→ Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> L = K.relativize(K.subfields(Integer(4))[Integer(0)][Integer(1)], b); L
Number Field in b with defining polynomial x^2 + a0 over its base field

(continues on next page)
>>> L_to_K, K_to_L = L.structure()

>>> V, fr, to = L.absolute_vector_space()

Vector space of dimension 8 over Rational Field

>>> fr
Isomorphism map:
    From: Vector space of dimension 8 over Rational Field
    To: Number Field in b with defining polynomial x^2 + a0 over its base field

>>> to
Isomorphism map:
    From: Number Field in b with defining polynomial x^2 + a0 over its base field
    To: Vector space of dimension 8 over Rational Field

>>> type(fr), type(to)
(<class sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField>,
 <class sage.rings.number_field.maps.MapRelativeNumberFieldToVectorSpace>)

>>> v = V([Integer(1), Integer(1), Integer(1), Integer(1), Integer(0), Integer(1),
                          →
                          Integer(1), Integer(1)])

>>> fr(v), to(fr(v)) == v
((-a0^3 + a0^2 - a0 + 1)*b - a0^3 - a0 + 1, True)

>>> to(L.gen()), fr(to(L.gen())) == L.gen()
((0, 1, 0, 0, 0, 0, 0, 0), True)

```python

class sage.rings.number_field.maps.MapRelativeToAbsoluteNumberField(R, A)

Bases: NumberFieldIsomorphism

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^6 + 4*x^2 + 200)
sage: L = K.relativize(K.subfields(3)[0][1], 'b'); L
Number Field in b with defining polynomial x^2 + a0 over its base field

sage: fr, to = L.structure()

sage: fr
Relative number field morphism:
    From: Number Field in b with defining polynomial x^2 + a0 over its base field
    To: Number Field in a with defining polynomial x^6 + 4*x^2 + 200
    Defn: b |--> a
            a0 |--> -a^2

sage: to
Ring morphism:
    From: Number Field in a with defining polynomial x^6 + 4*x^2 + 200
    To: Number Field in b with defining polynomial x^2 + a0 over its base field
    Defn: a |--> b

sage: type(fr), type(to)
(<class 'sage.rings.number_field.homset.RelativeNumberFieldHomset_with_category.element_class'>,
 <class 'sage.rings.number_field.homset.NumberFieldHomset_with_category.element_class'>)

sage: M.<c> = L.absolute_field(); M
Number Field in c with defining polynomial x^6 + 4*x^2 + 200

sage: fr, to = M.structure()

sage: fr
Isomorphism map:
    From: Number Field in c with defining polynomial x^6 + 4*x^2 + 200
    To: Number Field in a with defining polynomial x^3 + a0 + 1
    Defn: c |--> a
            a0 |--> -a^2
```

To: Number Field in b with defining polynomial \(x^2 + a0\) over its base field

\begin{verbatim}
sage: to
Isomorphism map:
  From: Number Field in b with defining polynomial x^2 + a0 over its base field
  To: Number Field in c with defining polynomial x^6 + 4*x^2 + 200
sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapAbsoluteToRelativeNumberField'>,
 <class 'sage.rings.number_field.maps.MapRelativeToAbsoluteNumberField'>)
sage: fr(M.gen()), to(fr(M.gen())) == M.gen()
(b, True)
sage: to(L.gen()), fr(to(L.gen())) == L.gen()
(c, True)
sage: (to * fr)(M.gen()) == M.gen(), (fr * to)(L.gen()) == L.gen()
(True, True)
\end{verbatim}

\[
\begin{align*}
\textbf{3.4. Structure maps for number fields} & \quad 391 \\
\end{align*}
\]

(continues on next page)
(c, True)
>>> (to * fr)(M.gen()) == M.gen(), (fr * to)(L.gen()) == L.gen()
(True, True)

class sage.rings.number_field.maps.MapRelativeVectorSpaceToRelativeNumberField(V, K)

Bases: NumberFieldIsomorphism

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: L.<b> = NumberField(x^4 + 3*x^2 + 1)
sage: K = L.relativize(L.subfields(2)[0][1], a); K
Number Field in a with defining polynomial x^2 - b0*x + 1 over its base field
sage: V, fr, to = K.relative_vector_space()
sage: V
Vector space of dimension 2 over Number Field in b0 with defining polynomial x^2 + 1
sage: fr
Isomorphism map:
   From: Vector space of dimension 2 over Number Field in b0 with defining polynomial x^2 + 1
   To: Number Field in a with defining polynomial x^2 - b0*x + 1 over its base field
sage: type(fr)
<class 'sage.rings.number_field.maps.MapRelativeVectorSpaceToRelativeNumberField'>

sage: a0 = K.gen(); b0 = K.base_field().gen()

sage: fr(to(a0 + 2*b0)), fr(V([[0, 1]])), fr(V([[b0, 2*b0]]))
(a + 2*b0, a, 2*b0*a + b0)

sage: (fr * to)(K.gen()) == K.gen()
True
sage: (to * fr)(V([[1, 2]])) == V([[1, 2]])
True

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> L = NumberField(x**Integer(4) + Integer(3)*x**Integer(2) + Integer(1), names=(
˓→'b',)); (b,) = L._first_ngens(1)

>>> K = L.relativize(L.subfields(Integer(2))[Integer(0)][Integer(1)], a); K
Number Field in a with defining polynomial x^2 - b0*x + 1 over its base field

>>> V, fr, to = K.relative_vector_space()

>>> V
Vector space of dimension 2 over Number Field in b0 with defining polynomial x^2 + 1

>>> fr
Isomorphism map:
   From: Vector space of dimension 2 over Number Field in b0 with defining polynomial x^2 + 1
   To: Number Field in a with defining polynomial x^2 - b0*x + 1 over its base field

>>> type(fr)
<class 'sage.rings.number_field.maps.MapRelativeVectorSpaceToRelativeNumberField'>

>>> a0 = K.gen(); b0 = K.base_field().gen()

>>> fr(to(a0 + Integer(2)*b0)), fr(V([Integer(0), Integer(1)])), fr(V([b0,␣
˓→])

(continues on next page)
class sage.rings.number_field.maps.MapVectorSpaceToNumberField(V, K)

Bases: NumberFieldIsomorphism

The map to an absolute number field from its underlying \( \mathbb{Q} \)-vector space.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: V
Vector space of dimension 4 over Rational Field
sage: fr
Isomorphism map:
    From: Vector space of dimension 4 over Rational Field
    To: Number Field in a with defining polynomial x^4 + 3*x + 1
sage: to
Isomorphism map:
    From: Number Field in a with defining polynomial x^4 + 3*x + 1
    To: Vector space of dimension 4 over Rational Field
sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapVectorSpaceToNumberField'>,
 <class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>)

sage: fr.is_injective(), fr.is_surjective()
(True, True)

sage: fr.domain(), to.codomain()
(Vector space of dimension 4 over Rational Field,
 Vector space of dimension 4 over Rational Field)

sage: to.domain(), fr.codomain()
(Number Field in a with defining polynomial x^4 + 3*x + 1,
 Number Field in a with defining polynomial x^4 + 3*x + 1)
```

3.4. Structure maps for number fields
To: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

then

Isomorphism map:

From: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

To: Vector space of dimension 4 over Rational Field

```
sage: to(a), to(a + 1)
((0, 1, 0, 0), (1, 1, 0, 0))
sage: fr(to(a)), fr(V([0, 1, 2, 3]))
(a, 3*a^3 + 2*a^2 + a)
```

```
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x**Integer(4) + Integer(3)*x + Integer(1), names=('a',)); (a, →) = K._first_ngens(1)

>>> V, fr, to = K.vector_space()

>>> V
Vector space of dimension 4 over Rational Field

>>> fr
Isomorphism map:

From: Vector space of dimension 4 over Rational Field

To: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

>>> to
Isomorphism map:

From: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

To: Vector space of dimension 4 over Rational Field

>>> type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapVectorSpaceToNumberField'>,
<class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>)

>>> fr.is_injective(), fr.is_surjective()
(True, True)

>>> fr.domain(), to.codomain()
(Vector space of dimension 4 over Rational Field,
Vector space of dimension 4 over Rational Field)

>>> to.domain(), fr.codomain()
(Number Field in a with defining polynomial \( x^4 + 3x + 1 \),
Number Field in a with defining polynomial \( x^4 + 3x + 1 \))

>>> fr * to
Composite map:

From: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

To: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

Defn: Isomorphism map:

From: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

To: Vector space of dimension 4 over Rational Field

then

Isomorphism map:

From: Vector space of dimension 4 over Rational Field

To: Number Field in a with defining polynomial \( x^4 + 3x + 1 \)

>>> to * fr
Composite map:

From: Vector space of dimension 4 over Rational Field

To: Vector space of dimension 4 over Rational Field

Defn: Isomorphism map:

From: Vector space of dimension 4 over Rational Field

(continues on next page)
To: Number Field in a with defining polynomial $x^4 + 3x + 1$
then
Isomorphism map:
From: Number Field in a with defining polynomial $x^4 + 3x + 1$
To: Vector space of dimension 4 over Rational Field

```python
>>> to(a), to(a + Integer(1))
((0, 1, 0, 0), (1, 1, 0, 0))
```

```python
>>> fr(to(a)), fr(V([Integer(0), Integer(1), Integer(2), Integer(3)]))
(a, 3*a^3 + 2*a^2 + a)
```

```python
from sage.rings.number_field.maps import MapVectorSpaceToRelativeNumberField

class sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField(V, L, from_V, from_K):
    Bases: NumberFieldIsomorphism

    The isomorphism to a relative number field from its underlying $\mathbb{Q}$-vector space. Compare MapRelativeVectorSpaceToRelativeNumberField.

    EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 3)
sage: from_L, to_L = L.structure()
sage: from_L
Isomorphism given by variable name change map:
From: Number Field in b with defining polynomial $x^2 - 3$
To: Number Field in a with defining polynomial $x^2 - 3$
sage: to_L
Isomorphism given by variable name change map:
From: Number Field in a with defining polynomial $x^2 - 3$
To: Number Field in b with defining polynomial $x^2 - 3$
sage: type(from_L), type(to_L)
(<class 'sage.rings.number_field.maps.NameChangeMap'>,
<class 'sage.rings.number_field.maps.NameChangeMap'>)
```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> L = K.change_names(names=('b',)); (b,) = L._first_ngens(1)
>>> from_L, to_L = L.structure()

Isomorphism given by variable name change map:
From: Number Field in b with defining polynomial x^2 - 3
To: Number Field in a with defining polynomial x^2 - 3

Isomorphism given by variable name change map:
From: Number Field in a with defining polynomial x^2 - 3
To: Number Field in b with defining polynomial x^2 - 3

>>> type(from_L), type(to_L)
(<class 'sage.rings.number_field.maps.NameChangeMap'>, <class 'sage.rings.number_field.maps.NameChangeMap'>)

class sage.rings.number_field.maps.NumberFieldIsomorphism

Bases: Map

A base class for various isomorphisms between number fields and vector spaces.

EXAMPLES:

sage: x = polygen(ZZ, x)
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: isinstance(fr, sage.rings.number_field.maps.NumberFieldIsomorphism)
True

sage: x = polygen(ZZ, x)
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_injective()
True

is_injective()

EXAMPLES:

sage: x = polygen(ZZ, x)
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_injective()
True

is_surjective()
EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_surjective()
True
```

```sage
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x^4 + Integer(3)*x + Integer(1), names=('a',));
\rightarrow (a,)
>>> V, fr, to = K.vector_space()
>>> fr.is_surjective()
True
```

3.5 Helper classes for structural embeddings and isomorphisms of number fields

Consider the following fields $L$ and $M$:

```sage
sage: L.<a> = QuadraticField(2)
sage: M.<a> = L.absolute_field()
```

```sage
>>> from sage.all import *
>>> L = QuadraticField(Integer(2), names=('a',)); (a,) = L._first_ngens(1)
>>> M = L.absolute_field(names=('a',)); (a,) = M._first_ngens(1)
```

Both produce the same extension of $\mathbb{Q}$. However, they should not be identical because $M$ carries additional information:

```sage
sage: L.structure()
(Identity endomorphism of Number Field in a with defining polynomial x^2 - 2
with a = 1.414213562373095?,
Identity endomorphism of Number Field in a with defining polynomial
x^2 - 2 with a = 1.414213562373095?)
sage: M.structure()
(Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 - 2
  To:  Number Field in a with defining polynomial x^2 - 2 with
       a = 1.414213562373095?,
Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 - 2
  To:  Number Field in a with defining polynomial x^2 - 2 with
       a = 1.414213562373095?)
```

```sage
>>> from sage.all import *
>>> L.structure()
(Identity endomorphism of Number Field in a with defining polynomial
x^2 - 2 with a = 1.414213562373095?,
Identity endomorphism of Number Field in a with defining polynomial
x^2 - 2 with a = 1.414213562373095?)
>>> M.structure()
```

(continues on next page)
This used to cause trouble with caching and made (absolute) number fields not unique when they should have been. The underlying technical problem is that the morphisms returned by structure() can only be defined once the fields in question have been created. Therefore, these morphisms cannot be part of a key which uniquely identifies a number field.

The classes defined in this file encapsulate information about these structure morphisms which can be passed to the factory creating number fields. This makes it possible to distinguish number fields which only differ in terms of these structure morphisms:

```sage
sage: L is M
False
sage: N.<a> = L.absolute_field()
sage: M is N
True
```

```sage
>>> from sage.all import *
>>> L is M
False
>>> N = L.absolute_field(names=('a',)); (a,) = N._first_ngens(1)
>>> M is N
True
```

AUTHORS:

- Julian Rueth (2014-04-03): initial version

```python
class sage.rings.number_field.structure.AbsoluteFromRelative(other)
    Bases: NumberFieldStructure
    Structure for an absolute number field created from a relative number field.

    INPUT:
    - other – the number field from which this field has been created.

    create_structure(field)
    Return a pair of isomorphisms which go from field to other and vice versa.

class sage.rings.number_field.structure.NameChange(other)
    Bases: NumberFieldStructure
    Structure for a number field created by a change in variable name.

    INPUT:
    - other – the number field from which this field has been created.

    create_structure(field)
    Return a pair of isomorphisms which send the generator of field to the generator of other and vice versa.
```
Abstract base class encapsulating information about a number fields relation to other number fields.

\texttt{create\_structure(field)}

Return a tuple encoding structural information about \texttt{field}.

\textbf{OUTPUT:}

Typically, the output is a pair of morphisms. The first one from \texttt{field} to a field from which \texttt{field} has been constructed and the second one its inverse. In this case, these morphisms are used as conversion maps between the two fields.

\texttt{class sage.rings.number_field.structure.RelativeFromAbsolute(other, gen)}

Structure for a relative number field created from an absolute number field.

\textbf{INPUT:}

- \texttt{other} – the (absolute) number field from which this field has been created.
- \texttt{gen} – the generator of the intermediate field

\texttt{create\_structure(field)}

Return a pair of isomorphisms which go from \texttt{field} to \texttt{other} and vice versa.

\textbf{INPUT:}

- \texttt{field} – a relative number field

\texttt{class sage.rings.number_field.structure.RelativeFromRelative(other)}

Structure for a relative number field created from another relative number field.

\textbf{INPUT:}

- \texttt{other} – the relative number field used in the construction, see \texttt{create\_structure()}; there this field will be called \texttt{field\_}.

\texttt{create\_structure(field)}

Return a pair of isomorphisms which go from \texttt{field} to the relative number field (called \texttt{other} below) from which \texttt{field} has been created and vice versa.

The isomorphism is created via the relative number field \texttt{field\_} which is identical to \texttt{field} but is equipped with an isomorphism to an absolute field which was used in the construction of \texttt{field}.

\textbf{INPUT:}

- \texttt{field} – a relative number field
4.1 Orders in number fields

EXAMPLES:

We define an absolute order:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 1); O = K.order(2*a)
sage: O.basis()
[1, 2*a]
```

We compute a basis for an order in a relative extension that is generated by 2 elements:

```
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: O = K.order([3*a, 2*b])
sage: O.basis()
[1, 3*a - 2*b, -6*b*a + 6, 3*a]
```

We compute a maximal order of a degree 10 field:

```
sage: K.<a> = NumberField((x+1)^10 + 17)
sage: K.maximal_order()
Maximal Order generated by a in Number Field in a with defining polynomial
x^10 + 10*x^9 + 45*x^8 + 120*x^7 + 210*x^6 + 252*x^5 + 210*x^4 + 120*x^3 + 45*x^2 +
-10*x + 18
```

(continues on next page)
We compute a suborder, which has index a power of 17 in the maximal order:

```python
sage: O = K.order(17*a); O
Order generated by 17*a in Number Field in a with defining polynomial
x^10 + 10*x^9 + 45*x^8 + 120*x^7 + 210*x^6 + 252*x^5 + 210*x^4 + 120*x^3 + 45*x^2 +
    10*x + 18

sage: m = O.index_in(K.maximal_order()); m
23453165165327788911665591944416226304630809183732482257

sage: factor(m)
17^45
```

AUTHORS:


```python
create_key_and_extra_args (K, module_rep, is_maximal=None, check=True, is_maximal_at=())
```

Return normalized arguments to create an absolute order.

```python
create_object (version, key, is_maximal=None, is_maximal_at=())
```

Create an absolute order.
**reduce_data** *(order)*

Return the data that can be used to pickle an order created by this factory.

This overrides the default implementation to update the latest knowledge about primes at which the order is maximal.

EXAMPLES:

This also works for relative orders since they are wrapping absolute orders:

```
sage: x = polygen(ZZ, 'x')
sage: L.<a, b> = NumberField([x^2 - 1000003, x^2 - 5*1000099^2])
sage: O = L.maximal_order([5], assume_maximal=None)
sage: s = dumps(O)
sage: loads(s) is O
True

sage: N = L.maximal_order([7], assume_maximal=None)
sage: dumps(N) == s
False
sage: loads(dumps(N)) is O
True
```

```bash
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> L = NumberField([x**Integer(2) - Integer(1000003), x**Integer(2) -
˓→Integer(5)*Integer(1000099)**Integer(2)], names=('a', 'b',)); (a, b,) = L._
˓→first_ngens(2)
>>> O = L.maximal_order([Integer(5)], assume_maximal=None)

>>> s = dumps(O)
>>> loads(s) is O
True

>>> N = L.maximal_order([Integer(7)], assume_maximal=None)
>>> dumps(N) == s
False
>>> loads(dumps(N)) is O
True
```

**sage.rings.number_field.order.EisensteinIntegers** *(names='omega')*

Return the ring of Eisenstein integers.

This is the ring of all complex numbers of the form \(a + b\omega\) with \(a\) and \(b\) integers and \(\omega = (-1 + \sqrt{-3})/2\).

EXAMPLES:

```
sage: R.<omega> = EisensteinIntegers()
sage: R
Eisenstein Integers generated by omega in Number Field in omega
with defining polynomial x^2 + x + 1
with omega = -0.5000000000000000? + 0.866025403784439?*I
sage: factor(3 + omega)
(-1) * (-omega - 3)
sage: CC(omega)
-0.500000000000000 + 0.866025403784439*I
```

(continues on next page)
sage: omega.minpoly()
x^2 + x + 1
sage: EisensteinIntegers().basis()
[1, omega]

>>> from sage.all import *
>>> R = EisensteinIntegers(names=('omega',)); (omega,) = R._first_ngens(1)
>>> R
Eisenstein Integers generated by omega in Number Field in omega
with defining polynomial x^2 + x + 1
with omega = -0.50000000000000000? + 0.866025403784439?*I
>>> factor(Integer(3) + omega)
(-1) * (-omega - 3)
>>> CC(omega)
-0.500000000000000 + 0.866025403784439*I
>>> omega.minpoly()
x^2 + x + 1
>>> EisensteinIntegers().basis()
[1, omega]

sage.rings.number_field.order.EquationOrder(f, names, **kwds)

Return the equation order generated by a root of the irreducible polynomial $f$ or list $f$ of polynomials (to construct a relative equation order).

IMPORTANT: Note that the generators of the returned order need not be roots of $f$, since the generators of an order are – in Sage – module generators.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: O.<a,b> = EquationOrder([x^2 + 1, x^2 + 2])
sage: O
Relative Order generated by [-b*a - 1, -3*a + 2*b] in
Number Field in a with defining polynomial x^2 + 1 over its base field
sage: O.0
-b*a - 1
sage: O.1
-3*a + 2*b

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> O = EquationOrder([x**Integer(2) + Integer(1), x**Integer(2) + Integer(2)],
˓→names=('a', 'b',)); (a, b,) = O._first_ngens(2)
>>> O
Relative Order generated by [-b*a - 1, -3*a + 2*b] in
Number Field in a with defining polynomial x^2 + 1 over its base field
>>> O.gen(0)
-b*a - 1
>>> O.gen(1)
-3*a + 2*b

Of course the input polynomial must be integral:

sage: R = EquationOrder(x^3 + x + 1/3, 'alpha'); R
Traceback (most recent call last):
...
ValueError: each generator must be integral

```sage
R = EquationOrder([x^3 + x + 1, x^2 + 1/2], 'alpha'); R
Traceback (most recent call last):
...  
ValueError: each generator must be integral
```

```sage
def from sage.all import *
R = EquationOrder(x**Integer(3) + x + Integer(1)/Integer(3), 'alpha'); R
Traceback (most recent call last):
...  
ValueError: each generator must be integral
```

```sage
R = EquationOrder([x**Integer(3) + x + Integer(1), x**Integer(2) + Integer(1)/Integer(2)], 'alpha'); R
Traceback (most recent call last):
...  
ValueError: each generator must be integral
```

```sage
sage: R = EquationOrder([x^3 + x + 1, x^2 + 1/2], 'alpha'); R
```

```sage
>>> from sage.all import *
>>> R = EquationOrder(x**Integer(3) + x + Integer(1)/Integer(3), 'alpha'); R
Traceback (most recent call last):
...  
ValueError: each generator must be integral
```

```sage
R = EquationOrder([x**Integer(3) + x + Integer(1), x**Integer(2) + Integer(1)/Integer(2)], 'alpha'); R
Traceback (most recent call last):
...  
ValueError: each generator must be integral
```

```sage
sage: R = EquationOrder([x^3 + x + 1, x^2 + 1/2], 'alpha'); R
```

```sage
>>> from sage.all import *
>>> R = EquationOrder(x**Integer(3) + x + Integer(1)/Integer(3), 'alpha'); R
```

```sage
from sage.rings.number_field.order import GaussianIntegers

GaussianIntegers(names='I', latex_name='i')

Return the ring of Gaussian integers.

This is the ring of all complex numbers of the form \( a + bI \) with \( a \) and \( b \) integers and \( I = \sqrt{-1} \).

EXAMPLES:

```sage
sage: ZZI.<I> = GaussianIntegers()
sage: ZZI
Gaussian Integers generated by I in Number Field in I with defining polynomial x^2 + 1 with I = 1*I
sage: factor(3 + I)
(-I) * (I + 1) * (2*I + 1)
sage: CC(I)
1.00000000000000*I
sage: I.minpoly()
x^2 + 1
sage: GaussianIntegers().basis()
[1, I]
```

```sage
>>> from sage.all import *
>>> ZZI = GaussianIntegers(names=('I',)); (I,) = ZZI._first_ngens(1)
```

```sage
>>> ZZI = GaussianIntegers(names=('I',)); (I,) = ZZI._first_ngens(1)
```

```sage
class sage.rings.number_field.order.Order(K)

Bases: IntegralDomain, Order

An order in a number field.

4.1. Orders in number fields
An order is a subring of the number field that has $\mathbb{Z}$-rank equal to the degree of the number field over $\mathbb{Q}$.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<theta> = NumberField(x^4 + x + 17)
sage: K.maximal_order()
Maximal Order generated by theta in Number Field in theta with defining polynomial $x^4 + x + 17$

sage: R = K.order(17*theta); R
Order generated by 17*theta in Number Field in theta with defining polynomial $x^4 + x + 17$

sage: R.basis()
[1, 17*theta, 289*theta^2, 4913*theta^3]
```

```python
sage: K.<b> = NumberField(x^4 + x^2 + 2)
sage: (b^2).charpoly().factor()
(x^2 + x + 2)^2

sage: K.order(b^2)
Traceback (most recent call last):
  ... ValueError: the rank of the span of gens is wrong
```
absolute_degree()

Return the absolute degree of this order, i.e., the degree of this order over \( \mathbf{Z} \).

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2)
sage: O = K.maximal_order()
sage: O.absolute_degree()
3
```

ambient()

Return the ambient number field that contains self.

This is the same as number_field() and fraction_field()

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: k.<z> = NumberField(x^2 - 389)
sage: o = k.order(389*z + 1)
sage: o.ambient()
Number Field in z with defining polynomial x^2 - 389
```

basis()

Return a basis over \( \mathbf{Z} \) of this order.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + x^2 - 16*x + 16)
sage: O = K.maximal_order(); O
```

(continues on next page)
Maximal Order generated by 1/4*a^2 + 1/4*a in Number Field in a with defining polynomial x^3 + x^2 - 16*x + 16
sage: O.basis()
[1, 1/4*a^2 + 1/4*a, a^2]

```
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x**Integer(3) + x**Integer(2) - Integer(16)*x + Integer(16), names=('a',)); (a,) = K._first_ngens(1)

Maximal Order generated by 1/4*a^2 + 1/4*a in Number Field in a with defining polynomial x^3 + x^2 - 16*x + 16
>>> O = K.maximal_order(); O

Maximal Order generated by 1/4*a^2 + 1/4*a in Number Field in a with defining polynomial x^3 + x^2 - 16*x + 16
>>> O.basis()
[1, 1/4*a^2 + 1/4*a, a^2]
```

**class_group** *(proof=None, names='a')*

Return the class group of this order.

(Currently only implemented for the maximal order.)

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^2 + 5077)
sage: O = k.maximal_order(); O

Maximal Order generated by a in Number Field in a with defining polynomial x^2 + 5077
sage: O.class_group()
Class group of order 22 with structure C22 of Number Field in a with defining polynomial x^2 + 5077
```

```
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> k = NumberField(x**Integer(2) + Integer(5077), names=(a,)); (a,) = k._first_ngens(1)

Maximal Order generated by a in Number Field in a with defining polynomial x^2 + 5077
>>> O = k.maximal_order(); O

Maximal Order generated by a in Number Field in a with defining polynomial x^2 + 5077
>>> O.class_group()
Class group of order 22 with structure C22 of Number Field in a with defining polynomial x^2 + 5077
```

**class_number** *(proof=None)*

Return the class number of this order.

**EXAMPLES:**

```
sage: ZZ[2^(1/3)].class_number()  # needs sage.symbolic
1
sage: QQ[sqrt(-23)].maximal_order().class_number()  # needs sage.symbolic
3
sage: ZZ[120*sqrt(-23)].class_number()  # needs sage.symbolic
288
```
>> from sage.all import *
>>> ZZ[Integer(2)**(Integer(1)/Integer(3))].class_number()
    # needs sage.symbolic
1
>>> QQ[sqrt(-Integer(23))].maximal_order().class_number()
    # needs sage.symbolic
3
>>> ZZ[Integer(120)*sqrt(-Integer(23))].class_number()
    # needs sage.symbolic
288

Note that non-maximal orders are only supported in quadratic fields:

sage: ZZ[120*sqrt(-23)].class_number()
    # needs sage.symbolic
288
sage: ZZ[100*sqrt(3)].class_number()
    # needs sage.symbolic
4
sage: ZZ[11*2^(1/3)].class_number()
    # needs sage.symbolic
Traceback (most recent call last):
  ... Not ImplementedError: computation of class numbers of non-maximal orders not in quadratic fields is not implemented

sage: ZZ[Integer(120)*sqrt(-Integer(23))].class_number()
    # needs sage.symbolic
288
sage: ZZ[Integer(100)*sqrt(Integer(3))].class_number()
    # needs sage.symbolic
4
sage: ZZ[Integer(11)*Integer(2)**(Integer(1)/Integer(3))].class_number()
    # needs sage.symbolic
Traceback (most recent call last):
  ... Not ImplementedError: computation of class numbers of non-maximal orders not in quadratic fields is not implemented

conductor()
For orders in quadratic number fields, return the conductor of this order.
The conductor is the unique positive integer \( f \) such that the discriminant of this order is \( f^2 \) times the discriminant of the containing quadratic field.
Not implemented for orders in number fields of degree \( \neq 2 \).

See also:
sage.rings.number_field.number_field.NumberField_quadratic.order_of_conductor()

EXAMPLES:
sage: K.<t> = QuadraticField(-101)
sage: K.maximal_order().conductor()
1

(continues on next page)
coordinates \( (x) \)

Return the coordinate vector of \( x \) with respect to this order.

INPUT:

- \( x \) – an element of the number field of this order.

OUTPUT:

A vector of length \( n \) (the degree of the field) giving the coordinates of \( x \) with respect to the integral basis of the order. In general this will be a vector of rationals; it will consist of integers if and only if \( x \) is in the order.

AUTHOR: John Cremona 2008-11-15

ALGORITHM:

Uses linear algebra. The change-of-basis matrix is cached. Provides simpler implementations for \_contains\_(), is_integral() and smallest_integer().

EXAMPLES:

```python
sage: K.<i> = QuadraticField(-1)
sage: OK = K.ring_of_integers()
sage: OK_basis = OK.basis(); OK_basis
[1, i]
sage: a = 23-14*i
sage: acoords = OK.coordinates(a); acoords
(23, -14)
sage: sum([OK_basis[j]*acoords[j] for j in range(2)]) == a
True
sage: OK.coordinates((120+340*i)/8)
(15, 85/2)

sage: O = K.order(3*i)
sage: O.is_maximal()
False
sage: O.index_in(OK)
3
sage: acoords = O.coordinates(a); acoords
(23, -14/3)
```
sage: sum([OK.basis()[j]*acoords[j] for j in range(2)]) == a
True

>>> from sage.all import *
>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> OK = K.ring_of_integers()
>>> OK_basis = OK.basis(); OK_basis
[1, i]
>>> a = Integer(23) - Integer(14)*i
>>> acoords = OK.coordinates(a); acoords
(23, -14)
>>> sum([OK_basis[j]*acoords[j] for j in range(Integer(2))]) == a
True
>>> OK.coordinates((Integer(120) + Integer(340)*i)/Integer(8))
(15, 85/2)
>>> O = K.order(Integer(3)*i)
>>> O.is_maximal()
False
>>> O.index_in(OK)
3
>>> acoords = O.coordinates(a); acoords
(23, -14/3)
>>> sum([O.basis()[j]*acoords[j] for j in range(Integer(2))]) == a
True

\textbf{degree()}

Return the degree of this order, which is the rank of this order as a \(\mathbb{Z}\)-module.

\textbf{EXAMPLES:}

sage: x = polygen(ZZ, 'x')
sage: K.<c> = NumberField(x^3 + x^2 - 2*x+8)
sage: o = K.maximal_order()
sage: o.degree()
3
sage: o.rank()
3

\textbf{fraction_field()}

Return the fraction field of this order, which is the ambient number field.

\textbf{EXAMPLES:}

sage: x = polygen(ZZ, 'x')
sage: K.<c> = NumberField(x^4 + 17*x^2 + 17)
fractional_ideal(*args, **kwds)

Return the fractional ideal of the maximal order with given generators.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 2)
sage: R = K.maximal_order()
sage: R.fractional_ideal(2/3 + 7*a, a)
Fractional ideal (1/3*a)
```

free_module()

Return the free \( \mathbb{Z} \)-module contained in the vector space associated to the ambient number field, that corresponds to this order.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: O = K.maximal_order(); O.basis()
[1, 1/2*a^2 + 1/2*a, a^2]
sage: O.free_module()
Free module of degree 3 and rank 3 over Integer Ring
User basis matrix:
[ 1 0 0]
[ 0 1/2 1/2]
[ 0 0 1]
```

(continues from previous page)
An example in a relative extension. Notice that the module is a \( \mathbb{Z} \)-module in the absolute field associated to the relative field:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 2])
sage: O = K.maximal_order(); O.basis()
[(-3/2*b - 5)*a + 7/2*b - 2, -3*a + 2*b, -2*b*a - 3, -7*a + 5*b]
sage: O.free_module()
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
[1/4 1/4 3/4 3/4]
[0 1/2 0 1/2]
[0 0 1 0]
[0 0 0 1]
```

\texttt{gen}(i)

Return \( i \)th module generator of this order.

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: K.<c> = NumberField(x^3 + 2*x + 17)
sage: O = K.maximal_order(); O
Maximal Order generated by \( c \) in Number Field in \( c \) with defining polynomial \( x^3 + 2*x + 17 \)
sage: O.basis()
[1, c, c^2]
sage: O.gen(1)
c
sage: O.gen(2)
c^2
sage: O.gen(5)
```

(continues on next page)
Traceback (most recent call last):
...  
IndexError: no 5th generator

sage: O.gen(-1)

Traceback (most recent call last):
...  
IndexError: no -1th generator

>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x**Integer(3) + Integer(2)*x + Integer(17), names=('c',));

(continued on next page)

→ (c,) = K._first_ngens(1)

>>> O = K.maximal_order(); O

Maximal Order generated by c in Number Field in c with defining polynomial x^3 + 2*x + 17

>>> O.basis()

[1, c, c^2]

>>> O.gen(Integer(1))
c

>>> O.gen(Integer(2))
c^2

>>> O.gen(Integer(5))

Traceback (most recent call last):

...  
IndexError: no 5th generator

>>> O.gen(-Integer(1))

Traceback (most recent call last):

...  
IndexError: no -1th generator

ideal (*args, **kwds)

Return the integral ideal with given generators.

EXAMPLES:

sage: x = polygen(ZZ, 'x')

sage: K.<a> = NumberField(x^2 + 7)

sage: R = K.maximal_order()

sage: R.ideal(2/3 + 7*a, a)

Traceback (most recent call last):

...  
ValueError: ideal must be integral;
use fractional_ideal to create a non-integral ideal.

sage: R.ideal(7*a, 77 + 28*a)

Fractional ideal (7)

sage: R.ideal(8)

Traceback (most recent call last):

...  
ValueError: the conductor of the order is not known.

sage: R = K.order(4*a)

sage: R.ideal(8)

doctest:warning ... FutureWarning: ...

Ideal (8, 32*a) of Order of conductor 8 generated by 4*a
in Number Field in a with defining polynomial x^2 + 7

(continues on next page)
This function is called implicitly below:

```
sage: R = EquationOrder(x**2 + 2, 'a'); R
Maximal Order generated by a in Number Field in a with defining polynomial x^2 + 2
sage: (3,15)*R
doctest:warning ... DeprecationWarning: ...
Fractional ideal (3)
```

The zero ideal is handled properly:

```
sage: R.ideal(0)
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
```

```
>>> from sage.all import *
```

```
>>> R = EquationOrder(x**Integer(2) + Integer(2), 'a'); R
Maximal Order generated by a in Number Field in a with defining polynomial x^2 + 2
>>> (Integer(3),Integer(15))*R
doctest:warning ... DeprecationWarning: ...
Fractional ideal (3)
```

```
integral_closure()
return the integral closure of this order.
```

```
EXAMPLES:
```
```
>> from sage.all import *
>> K = QuadraticField(Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>> O2 = K.order(Integer(2)*a); O2
Order of conductor 4 generated by 2*a in Number Field in a
with defining polynomial x^2 - 5 with a = 2.236067977499790?
>> O2.integral_closure()
Maximal Order generated by 1/2*a + 1/2 in Number Field in a
with defining polynomial x^2 - 5 with a = 2.236067977499790?
>> OK = K.maximal_order()
>> OK is OK.integral_closure()
True

is_field (proof=True)

Return False (because an order is never a field).

EXAMPLES:

```sage
x = polygen(ZZ, 'x')
sage: L.<alpha> = NumberField(x**4 - x**2 + 7)
sage: O = L.maximal_order(); O.is_field()
False
```

```sage
from sage.all import *

x = polygen(ZZ, 'x')

L = NumberField(x**Integer(4) - x**Integer(2) + Integer(7), names=('alpha ->',)); (alpha,) = L._first_ngens(1)

O = L.maximal_order(); O.is_field()
False
```

is_integrally_closed()

Return True if this ring is integrally closed, i.e., is equal to the maximal order.

EXAMPLES:

```sage
x = polygen(ZZ, 'x')

K.<a> = NumberField(x^2 + 189*x + 394)

R = K.order(Integer(2)*a)

R.is_integrally_closed()
False
```

```sage
from sage.all import *

x = polygen(ZZ, 'x')

K = NumberField(x**Integer(2) + Integer(189)*x + Integer(394), names=('a',, -1)); (a,) = K._first_ngens(1)

R = K.order(Integer(2)*a)
```
Algebraic Numbers and Number Fields, Release 10.4

>>> R.is_integrally_closed()
False

Order of conductor 2 generated by 2*a in Number Field in a with defining polynomial:
    x^2 + 189*x + 394

S = K.maximal_order(); S
Maximal Order generated by a in Number Field in a with defining polynomial:
    x^2 + 189*x + 394

>>> S.is_integrally_closed()
True

is_noetherian()

Return True (because orders are always Noetherian)

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: L.<alpha> = NumberField(x**4 - x**2 + 7)
sage: O = L.maximal_order() ; O.is_noetherian()
True
sage: E.<w> = NumberField(x**2 - x + 2)
sage: OE = E.ring_of_integers(); OE.is_noetherian()
True
```

is_suborder(other)

Return True if self and other are both orders in the same ambient number field and self is a subset of other.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: W.<i> = NumberField(x^2 + 1)
sage: O5 = W.order(5*i)
sage: O10 = W.order(10*i)
sage: O15 = W.order(15*i)

>>> from sage.all import *

>>> from sage.all import *
```

(continued from previous page)
We create another isomorphic but different field:

```python
sage: W2.<j> = NumberField(x^2 + 1)
sage: P5 = W2.order(5*j)
```

This is False because the ambient number fields are not equal:

```python
>>> from sage.all import *
>>> W2 = NumberField(x**Integer(2) + Integer(1), names=('j',)); (j,) = W2._
>>> P5 = W2.order(Integer(5)*j)
```

We create a field that contains (in no natural way!) \( W \), and of course again \( is_suborder() \) returns False:

```python
sage: K.<z> = NumberField(x^4 + 1)
sage: M = K.order(5*z)
sage: O5.is_suborder(M)
False
```

```
>>> from sage.all import *
>>> K = NumberField(x**Integer(4) + Integer(1), names=('z',)); (z,) = K._
>>> M = K.order(Integer(5)*z)
>>> O5.is_suborder(M)
False
```

\textbf{krull_dimension()}

Return the Krull dimension of this order, which is 1.

**EXAMPLES:**

```python
sage: K.<a> = QuadraticField(5)
sage: OK = K.maximal_order()
sage: OK.krull_dimension()
1
sage: O2 = K.order(2*a)
sage: O2.krull_dimension()
1
```
>>> from sage.all import *
>>> K = QuadraticField(Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> OK = K.maximal_order()
>>> OK.krull_dimension()
1
>>> O2 = K.order(Integer(2)*a)
>>> O2.krull_dimension()
1
ngens()

Return the number of module generators of this order.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: O = K.maximal_order()
sage: O.ngens()
3
```

number_field()

Return the number field of this order, which is the ambient number field that this order is embedded in.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<b> = NumberField(x^4 + x^2 + 2)
sage: O = K.order(2*b); O
Order generated by 2*b in Number Field in b with defining polynomial x^4 + x^2 + 2
sage: O.basis()
[1, 2*b, 4*b^2, 8*b^3]
sage: O.number_field()
Number Field in b with defining polynomial x^4 + x^2 + 2
sage: O.number_field() is K
True
```
random_element(*args, **kwds)

Return a random element of this order.

INPUT:

* args, kwds – parameters passed to the random integer function. See the documentation for \ZZ\_. random_element() for details.

OUTPUT:

A random element of this order, computed as a random \ZZ\_linear combination of the basis.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 + 2)
sage: OK = K.ring_of_integers()
sage: OK.random_element() # random output
-2*a^2 - a - 2
sage: OK.random_element(distribution="uniform") # random output
-a^2 - 1
sage: OK.random_element(-10,10) # random output
-10*a^2 - 9*a - 2
sage: K.order(a).random_element() # random output
a^2 - a - 3
```

```python
>>> from sage.all import *
```
z^{15} - z^{11} - z^{10} - 4z^{9} + z^{8} + 2z^{7} + z^{6} - 2z^{5} - z^{4} - 445z^{3} - 2z^{2} - 15z - 2

```python
>>> OK.random_element().is_integral()
True
>>> OK.random_element().parent() is OK
True
```

A relative example:

```python
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: OK = K.ring_of_integers()
sage: OK.random_element() # random output
(42221/2*b + 61/2)*a + 7037384*b + 7041
sage: OK.random_element().is_integral() # random output
True
sage: OK.random_element().parent() is OK # random output
True
```

```python
from sage.all import *
```

```python
K = NumberField([x**Integer(2) + Integer(2), x**Integer(2) + \rightarrow Integer(1000)*x + Integer(1)], names=('a', 'b',)); (a, b,) = K._first_ \rightarrow ngens(2)
```

```python
>>> from sage.all import *
```

```python
K = NumberField([-Integer(3), names=(a,)); (a,) = K._first_ngens(1)
```

An example in a non-maximal order:

```python
sage: K.<a> = QuadraticField(-3)
sage: R = K.ring_of_integers()
sage: A = K.order(a)
sage: A.index_in(R)
2
sage: R.random_element() # random output
-39/2*a - 1/2
sage: A.random_element() # random output
2*a - 1
sage: A.random_element().is_integral()
True
sage: A.random_element().parent() is A
True
```

```python
from sage.all import *
```

```python
K = QuadraticField(-Integer(3), names=('a',)); (a,) = K._first_ngens(1)
```

```python
R = K.ring_of_integers()
```

```python
A = K.order(a)
```

```python
>>> from sage.all import *
```

(continues on next page)
> A.random_element().is_integral()
True
>>> A.random_element().parent() is A
True

**rank()**

Return the rank of this order, which is the rank of the underlying \( \mathbb{Z} \)-module, or the degree of the ambient number field that contains this order.

This is a synonym for `degree()`.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: k.<c> = NumberField(x^5 + x^2 + 1)
sage: o = k.maximal_order(); o
Maximal Order generated by c in Number Field in c with defining polynomial x^5 + x^2 + 1
sage: o.rank()
5
```

```python
>>> from sage.all import *
```
```python
>>> from sage.all import *
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(4) + Integer(3)*x**Integer(2) - Integer(17),
    names=('a',)); (a,) = K._first_ngens(1)
>>> P = K.ideal(Integer(61)).factor()[Integer(0)][Integer(0)]
>>> OK = K.maximal_order()
>>> OK.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)
>>> Fp = OK.residue_field(P, names=('b',)); (b,) = Fp._first_ngens(1)
>>> Fp
Residue field in b of Fractional ideal (61, a^2 + 30)
```

### ring_generators()

Return generators for self as a ring.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: O = K.maximal_order(); O
Gaussian Integers generated by i in Number Field in i with defining...
    polynomial x^2 + 1
sage: O.ring_generators()
[i]
```

This is an example where 2 generators are required (because 2 is an essential discriminant divisor):.

```python
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: O = K.maximal_order(); O.basis()
[1, 1/2*a^2 + 1/2*a, a^2]
sage: O.ring_generators()
[i]
```

An example in a relative number field:

```python
sage: K.<a, b> = NumberField([x^2 + x + 1, x^3 - 3])
sage: O = K.maximal_order()
```

(continues on next page)
some_elements()

Return a list of elements of the given order.

EXAMPLES:

```sage
G = GaussianIntegers(); G
Gaussian Integers generated by I in Number Field in I with defining polynomial x^2 + 1 with I = 1*I
sage: G.some_elements()
[1, I, 2*I, -1, 0, -I, 2, 4*I, -2, -2*I, -4]
```

```
sage: (t,) = QQ[t].first_ngens(1)
sage: K.<a> = QQ.extension(t^3 - 2); K
Number Field in a with defining polynomial t^3 - 2
sage: Z = K.ring_of_integers(); Z
Maximal Order generated by a in Number Field in a with defining polynomial t^3 - 2
sage: Z.some_elements()
[1, a, a^2, 2*a, 0, 2, a^2 + 2*a + 1, ..., a^2 + 1, 2*a^2 + 2, a^2 + 2*a, 4*a^2 + 2 + 4]
```

valuation(p)

Return the p-adic valuation on this order.

EXAMPLES:

```sage
>>> from sage.all import *
>>> G = GaussianIntegers(); G
Gaussian Integers generated by I in Number Field in I with defining polynomial x^2 + 1 with I = 1*I
```

```sage
>>> R = QQ['t']; (t,) = R.first_ngens(1)
>>> K = QQ.extension(t**Integer(3) - Integer(2), names=('a',)); (a,) = K.first_ngens(1); K
Number Field in a with defining polynomial t^3 - 2
>>> Z = K.ring_of_integers(); Z
Maximal Order generated by a in Number Field in a with defining polynomial t^3 - 2
>>> Z.some_elements()
[1, a, a^2, 2*a, 0, 2, a^2 + 2*a + 1, ..., a^2 + 1, 2*a^2 + 2, a^2 + 2*a, 4*a^2 + 2 + 4]
```
The valuation can be specified with an integer prime \( p \) that is completely ramified or unramified:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x**2 + 1)
sage: O = K.order(2*a)
sage: valuations.pAdicValuation(O, 2)
2-adic valuation
sage: GaussianIntegers().valuation(2)
2-adic valuation
```

A prime \( p \) that factors into pairwise distinct factors, results in an error:

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('a',)); (a,) = K._first_ngens()
>>> O = K.order(Integer(2)*a)
>>> valuations.pAdicValuation(O, Integer(2))
2-adic valuation
>>> GaussianIntegers().valuation(Integer(2))
2-adic valuation
```

The valuation can also be selected by giving a valuation on the base ring that extends uniquely:

```python
sage: CyclotomicField(5).ring_of_integers().valuation(ZZ.valuation(5))
5-adic valuation
>>> from sage.all import *
>>> CyclotomicField(Integer(5)).ring_of_integers().valuation(ZZ.valuation(Integer(5)))
5-adic valuation
```

When the extension is not unique, this does not work:
sage: GaussianIntegers().valuation(ZZ.valuation(5))
Traceback (most recent call last):
...
ValueError: The valuation Gauss valuation induced by 5-adic valuation does not approximate a unique extension of 5-adic valuation with respect to \(x^2 + 1\)

```python
>>> from sage.all import *

>>> GaussianIntegers().valuation(ZZ.valuation(Integer(5)))
Traceback (most recent call last):
...
ValueError: The valuation Gauss valuation induced by 5-adic valuation does not approximate a unique extension of 5-adic valuation with respect to \(x^2 + 1\)
```

If the fraction field is of the form \(K[x]/(G)\), you can specify a valuation by providing a discrete pseudo-valuation on \(K[x]\) which sends \(G\) to infinity:

```python
sage: R.<x> = QQ[]
sage: GV5 = GaussValuation(R, QQ.valuation(5))
sage: v = GaussianIntegers().valuation(GV5.augmentation(x + 2, infinity))
sage: w = GaussianIntegers().valuation(GV5.augmentation(x + 1/2, infinity))
sage: v == w
False
```

```python
>>> from sage.all import *

>>> R = QQ[x]; (x,) = R._first_ngens(1)

>>> GV5 = GaussValuation(R, QQ.valuation(Integer(5)))

>>> v = GaussianIntegers().valuation(GV5.augmentation(x + Integer(2), \→ infinity))

>>> w = GaussianIntegers().valuation(GV5.augmentation(x + Integer(1)/\→ Integer(2), infinity))

>>> v == w
False
```

See also:

`NumberField_generic.valuation()`, `pAdicGeneric.valuation()`

### zeta

\(n=2, \text{all}=False\)

Return a primitive \(n\)-th root of unity in this order, if it contains one. If \text{all} is \text{True}, return all of them.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: F.<alpha> = NumberField(x**2 + 3)
sage: F.ring_of_integers().zeta(6)
-1/2*alpha + 1/2
```

```python
sage: O = F.order([3*alpha])
sage: O.zeta(3)
Traceback (most recent call last):
...
ArithmeticError: there are no 3rd roots of unity in self
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> F = NumberField(x**Integer(2) + Integer(3), names=('alpha',)); (alpha,) = F._first_ngens(1)

>>> F.ring_of_integers().zeta(Integer(6))
```

(continues on next page)
\(-\frac{1}{2}\alpha + \frac{1}{2}\)

```python
>>> O = F.order([Integer(3)*alpha])
>>> O.zeta(Integer(3))
Traceback (most recent call last):
  ... ArithmeticError: there are no 3rd roots of unity in self
```

class sage.rings.number_field.order.OrderFactory

Bases: UniqueFactory

Abstract base class for factories creating orders, such as AbsoluteOrderFactory and RelativeOrderFactory.

get_object (version, key, extra_args)

Create the order identified by key.

This overrides the default implementation to update the maximality of the order if it was explicitly specified.

EXAMPLES:

Even though orders are unique parents, this lets us update their internal state when they are recreated with more additional information available about them:

```python
sage: x = polygen(ZZ, 'x')
sage: L.<a, b> = NumberField([x^2 - 1000003, x^2 - 5*1000099^2])
sage: O = L.maximal_order([2], assume_maximal=None)

sage: O._is_maximal_at(2)
True
sage: O._is_maximal_at(3) is None
True

sage: N = L.maximal_order([3], assume_maximal=None)

sage: N is O
True
sage: N._is_maximal_at(2)
True
sage: N._is_maximal_at(3)
True

>>> from sage.all import *

sage: L = NumberField([x^2 - Integer(1000003), x^2 - Integer(5)*Integer(1000099)^2], names=('a', 'b')); (a, b) = L._first_ngens(2)

sage: O = L.maximal_order([Integer(2)], assume_maximal=None)

sage: O._is_maximal_at(Integer(2))
True
sage: O._is_maximal_at(Integer(3)) is None
True

sage: N = L.maximal_order([Integer(3)], assume_maximal=None)

sage: N is O
True
sage: N._is_maximal_at(Integer(2))
True
```

(continues on next page)
class sage.rings.number_field.order.Order_absolute(K, module_rep)

Bases: Order

EXAMPLES:

```python
sage: from sage.rings.number_field.order import *
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3 + 2)
sage: V, from_v, to_v = K.vector_space()
sage: M = span([to_v(a^2), to_v(a), to_v(1)], ZZ)
sage: O = AbsoluteOrder(K, M); O
Maximal Order generated by a in Number Field in a with defining polynomial x^3 + 2
sage: M = span([to_v(a^2), to_v(a), to_v(2)], ZZ)
sage: O = AbsoluteOrder(K, M); O
Traceback (most recent call last):
... ValueError: 1 is not in the span of the module, hence not an order
```

```
>>> from sage.all import *
>>> from sage.rings.number_field.order import *

sage: x = polygen(QQ)

sage: K = NumberField(x**Integer(3) + Integer(2), names=(a,)); (a,) = K._first_
˓→ngens(1)
sage: V, from_v, to_v = K.vector_space()
sage: M = span([to_v(a**Integer(2)), to_v(a), to_v(Integer(1))], ZZ)
sage: O = AbsoluteOrder(K, M); O
Maximal Order generated by a in Number Field in a with defining polynomial x^3 + 2

sage: M = span([to_v(a**Integer(2)), to_v(a), to_v(Integer(2))], ZZ)
sage: O = AbsoluteOrder(K, M); O
Traceback (most recent call last):
... ValueError: 1 is not in the span of the module, hence not an order
```

absolute_discriminant()

Return the discriminant of this order.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^8 + x^3 - 13*x + 26)
sage: O = K.maximal_order()
sage: factor(O.discriminant())
3 * 11 * 13^2 * 613 * 1575917857
sage: L = K.order(13*a^2)
sage: factor(L.discriminant())
3^3 * 5^2 * 11 * 13^60 * 613 * 733^2 * 1575917857
sage: factor(L.index_in(O))
3 * 5 * 13^29 * 733
sage: L.discriminant() / O.discriminant() == L.index_in(O)^2
True
```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(8) + x**Integer(3) - Integer(13)*x + Integer(26), names=('a',)); (a,) = K._first_ngens(1)
>>> O = K.maximal_order()
>>> factor(O.discriminant())
3 * 11 * 13^2 * 613 * 1575917857
>>> L = K.order(Integer(13)*a**Integer(2))
>>> factor(L.discriminant())
3^3 * 5^2 * 11 * 13^60 * 613 * 733^2 * 1575917857
>>> factor(L.index_in(O))
3 * 5 * 13^29 * 733
>>> L.discriminant() / O.discriminant() == L.index_in(O)**Integer(2)
True

absolute_order()  
Return the absolute order associated to this order, which is just this order again since this is an absolute order.

EXAMPLES:

sage: x = polygen(ZZ, 'x')  
sage: k.<c> = NumberField(x^3 + 2)  
sage: O1 = k.order(a); O1  
Maximal Order generated by a in Number Field in a with defining polynomial x^3 + 2  
sage: O1.absolute_order() is O1  
True

basis()  
Return the basis over \( \mathbb{Z} \) for this order.

EXAMPLES:

sage: x = polygen(ZZ, 'x')  
sage: k.<c> = NumberField(x^3 + x^2 + 1)  
sage: O = k.maximal_order(); O  
Maximal Order generated by c in Number Field in c with defining polynomial x^3 + x^2 + 1  
sage: O.basis()  
[1, c, c^2]

(continues on next page)
\[ \rightarrow 3 + x^2 + 1 \]

```
>>> 0.basis()
[1, c, c^2]
```

The basis is an immutable sequence:

```
sage: type(0.basis())
<class 'sage.structure.sequence.Sequence_generic'>
```

```
>>> from sage.all import *
>>> type(0.basis())
<class 'sage.structure.sequence.Sequence_generic'>
```

The generator functionality uses the basis method:

```
sage: 0.0
1
sage: 0.1
c
sage: 0.basis()
[1, c, c^2]
sage: 0.ngens()
3
```

```
>>> from sage.all import *
>>> 0.gen(0)
1
>>> 0.gen(1)
c
>>> 0.basis()
[1, c, c^2]
>>> 0.ngens()
3
```

\textbf{change\_names (names)}

Return a new order isomorphic to this one in the number field with given variable names.

\textbf{EXAMPLES:}

```
sage: x = polygen(ZZ, 'x')
sage: R = EquationOrder(x^3 + x + 1, 'alpha'); R
Order generated by alpha in Number Field in alpha with defining polynomial x^3 + x + 1
sage: R.basis()
[1, alpha, alpha^2]
sage: S = R.change_names('gamma'); S
Order generated by gamma in Number Field in gamma with defining polynomial x^3 + x + 1
sage: S.basis()
[1, gamma, gamma^2]
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> R = EquationOrder(x**Integer(3) + x + Integer(1), 'alpha'); R
Order generated by alpha in Number Field in alpha with defining polynomial x^3 + x + 1
```

(continues on next page)
discriminant()  
Return the discriminant of this order.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^8 + x^3 - 13*x + 26)
sage: O = K.maximal_order()
sage: factor(O.discriminant())
3 * 11 * 13^2 * 613 * 1575917857
sage: L = K.order(13*a^2)
sage: factor(L.discriminant())
3^3 * 5^2 * 11 * 13^60 * 613 * 733^2 * 1575917857
sage: factor(L.index_in(O))
3 * 5 * 13^29 * 733
sage: L.discriminant() / O.discriminant() == L.index_in(O)^2
True
```
```python
sage: x = polygen(ZZ, 'x')
sage: k.<i> = NumberField(x^2 + 1)
sage: O1 = k.order(i)
sage: O5 = k.order(5*i)
sage: O5.index_in(O1)
5
sage: k.<a> = NumberField(x^3 + x^2 - 2*x+8)
sage: o = k.maximal_order()
sage: o
Maximal Order generated by [1/2*a^2 + 1/2*a, a^2] in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1 = k.order(a); O1
Order generated by a in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1.index_in(o)
2
sage: O2 = k.order(1+2*a); O2
Order generated by 2*a in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1.basis()
[1, a, a^2]
sage: O2.basis()
[1, 2*a, 4*a^2]
sage: o.index_in(O2)
1/16

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> k = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = k._first_ngens(1)
>>> O1 = k.order(i)
>>> O5 = k.order(Integer(5)*i)
>>> O5.index_in(O1)
5
>>> k = NumberField(x**Integer(3) + x**Integer(2) - Integer(2)*x+Integer(8), names=('a',)); (a,) = k._first_ngens(1)
>>> o = k.maximal_order()
>>> o
Maximal Order generated by [1/2*a^2 + 1/2*a, a^2] in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
>>> O1 = k.order(a); O1
Order generated by a in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
>>> O1.index_in(o)
2
>>> O2 = k.order(Integer(1)+Integer(2)*a); O2
Order generated by 2*a in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
>>> O1.basis()
[1, a, a^2]
>>> O2.basis()
[1, 2*a, 4*a^2]
>>> o.index_in(O2)
1/16
```

**intersection** (other)
Return the intersection of this order with another order.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)
sage: O6 = K.order(6*i)
sage: O9 = K.order(9*i)
sage: O6.basis()
[1, 6*i]
sage: O9.basis()
[1, 9*i]
sage: O6.intersection(O9).basis()
[1, 18*i]
sage: (O6 & O9).basis()
[1, 18*i]
sage: (O6 + O9).basis()
[1, 3*i]
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> O6 = K.order(Integer(6)*i)
>>> O9 = K.order(Integer(9)*i)
>>> O6.basis()
[1, 6*i]
>>> O9.basis()
[1, 9*i]
>>> O6.intersection(O9).basis()
[1, 18*i]
>>> (O6 & O9).basis()
[1, 18*i]
>>> (O6 + O9).basis()
[1, 3*i]
```

**is_maximal** *(p=None)*

Return whether this is the maximal order.

**INPUT:**

- **p** – an integer prime or **None** (default: **None**); if set, return whether this order is maximal at the prime **p**.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<i> = NumberField(x^2 + 1)

sage: K.order(3*i).is_maximal()
False
sage: K.order(5*i).is_maximal()
False
sage: (K.order(3*i) + K.order(5*i)).is_maximal()
True
sage: K.maximal_order().is_maximal()
True
```
Maximality can be checked at primes when the order is maximal at that prime by construction:

```
sage: K.maximal_order().is_maximal(p=3)
True
```

And also at other primes:

```
sage: K.order(3*i).is_maximal(p=3)
False
```

An example involving a relative order:

```
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 3])
sage: O = K.order([3*a, 2*b])
sage: O.is_maximal()
False
```

```
module()
```

Return the underlying free module corresponding to this order, embedded in the vector space corresponding to the ambient number field.

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^3 + x + 3)
sage: m = k.order(3*a); m
Order generated by 3*a in Number Field in a with defining polynomial x^3 + x + 3
```

```
Echelon basis matrix:
```
```
[1 0 0]
[0 3 0]
[0 0 9]
```

```
class sage.rings.number_field.order.Order_relative(K, absolute_order)

A relative order in a number field.

A relative order is an order in some relative number field.

Invariants of this order may be computed with respect to the contained order.

absolute_discriminant()

Return the absolute discriminant of self, which is the discriminant of the absolute order associated to self.

OUTPUT:

an integer

EXAMPLES:

sage: x = polygen(ZZ, 'x')

sage: R = EquationOrder([x**2 + 1, x**3 + x**2 + 3], 'a,b')

sage: d = R.absolute_discriminant(); d
-746496

sage: d is R.absolute_discriminant()
True

sage: factor(d)
-1 * 2^10 * 3^6

absolute_order(names='z')

Return underlying absolute order associated to this relative order.

INPUT:

* names - string (default: 'z'); name of generator of absolute extension.

Note: There is a default variable name, since this absolute order is frequently used for internal algorithms.
EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: R = EquationOrder([x^2 + 1, x^2 - 5], 'i', 'g'); R
Relative Order generated by [6*i - g, -g*i + 2, 7*i - g] in
Number Field in i with defining polynomial x^2 + 1 over its base field
sage: R.basis()
[1, 6*i - g, -g*i + 2, 7*i - g]
sage: S = R.absolute_order(); S
Order generated by [5/12*z^3 + 1/6*z, 1/2*z^2, 1/2*z^3] in Number Field in z with defining polynomial x^4 - 8*x^2 + 36
sage: S.basis()
[1, 5/12*z^3 + 1/6*z, 1/2*z^2, 1/2*z^3]
```

```
>>> from sage.all import *

>>> R = EquationOrder([x**Integer(2) + Integer(1), x**Integer(2) - Integer(5)], 'i', 'g'); R
Relative Order generated by [6*i - g, -g*i + 2, 7*i - g] in
Number Field in i with defining polynomial x^2 + 1 over its base field
>>> R.basis()
[1, 6*i - g, -g*i + 2, 7*i - g]

>>> S = R.absolute_order(); S
Order generated by [5/12*z^3 + 1/6*z, 1/2*z^2, 1/2*z^3] in Number Field in z with defining polynomial x^4 - 8*x^2 + 36
>>> S.basis()
[1, 5/12*z^3 + 1/6*z, 1/2*z^2, 1/2*z^3]
```

We compute a relative order in alpha0, alpha1, then make the generator of the number field that contains the absolute order be called gamma:

```python
sage: R = EquationOrder([x^2 + 2, x^2 - 3], 'alpha'); R
Relative Order generated by [-alpha1*alpha0 + 1, 5*alpha0 + 2*alpha1, 7*alpha0 + 3*alpha1] in
Number Field in alpha0 with defining polynomial x^2 + 2 over its base field
sage: R.absolute_order('gamma')
Order generated by [1/2*gamma^2 + 1/2, 7/10*gamma^3 + 1/10*gamma, gamma^3] in
Number Field in gamma with defining polynomial x^4 - 2*x^2 + 25
sage: R.absolute_order('gamma').basis()
[1/2*gamma^2 + 1/2, 7/10*gamma^3 + 1/10*gamma, gamma^2, gamma^3]
```

```python
>>> from sage.all import *

>>> R = EquationOrder([x**Integer(2) + Integer(2), x**Integer(2) - Integer(3)], 'alpha'); R
Relative Order generated by [-alpha1*alpha0 + 1, 5*alpha0 + 2*alpha1, 7*alpha0 + 3*alpha1] in
Number Field in alpha0 with defining polynomial x^2 + 2 over its base field
>>> R.absolute_order('gamma')
Order generated by [1/2*gamma^2 + 1/2, 7/10*gamma^3 + 1/10*gamma, gamma^3] in
Number Field in gamma with defining polynomial x^4 - 2*x^2 + 25
>>> R.absolute_order('gamma').basis()
[1/2*gamma^2 + 1/2, 7/10*gamma^3 + 1/10*gamma, gamma^2, gamma^3]
```

`basis()`

Return a basis for this order as \(\mathbb{Z}\)-module.
EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 3])
sage: O = K.order([a,b])
sage: O.basis()
[1, -2*a + b, -b*a - 2, -5*a + 3*b]
sage: z = O.1; z
-2*a + b
sage: z.absolute_minpoly()
x^4 + 14*x^2 + 1
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) + Integer(3)],
                 names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> O = K.order([a,b])
>>> O.basis()
[1, -2*a + b, -b*a - 2, -5*a + 3*b]
>>> z = O.gen(1); z
-2*a + b
>>> z.absolute_minpoly()
x^4 + 14*x^2 + 1
```

**index_in** (*other*)

Return the index of self in other.

This is a lattice index, so it is a rational number if self is not contained in other.

INPUT:

- *other* — another order with the same ambient absolute number field.

OUTPUT:

a rational number

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^3 + x + 3, x^2 + 1])
sage: R1 = K.order([3*a, 2*b])
sage: R2 = K.order([a, 4*b])
sage: R1.index_in(R2)
729/8
sage: R2.index_in(R1)
8/729
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(3) + x + Integer(3), x**Integer(2) +
                 Integer(1)], names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> R1 = K.order([Integer(3)*a, Integer(2)*b])
>>> R2 = K.order([a, Integer(4)*b])
>>> R1.index_in(R2)
729/8
>>> R2.index_in(R1)
8/729
```
is_maximal \( (p = \text{None}) \)

Return whether this is the maximal order.

INPUT:

- \( p \) – an integer prime or \text{None} (default: \text{None}); if set, return whether this order is maximal at the prime \( p \).

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 5])
sage: K.order(3*a, b).is_maximal()
False
sage: K.order(5*a, b/2 + 1/2).is_maximal()
False
sage: (K.order(3*a, b) + K.order(5*a, b/2 + 1/2)).is_maximal()
True
sage: K.maximal_order().is_maximal()
True
```

Maximality can be checked at primes when the order is maximal at that prime by construction:

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(5)],
˓→ names=('a', 'b',)); (a, b,) = K._first_ngens(2)

>>> K.order(Integer(3)*a, b).is_maximal()
False
>>> K.order(Integer(5)*a, b/Integer(2) + Integer(1)/Integer(2)).is_maximal()
False
>>> (K.order(Integer(3)*a, b) + K.order(Integer(5)*a, b/Integer(2) +
˓→Integer(1)/Integer(2))).is_maximal()
True
>>> K.maximal_order().is_maximal()
True
```

And at other primes:

```python
sage: K.order(3*a, b).is_maximal(p=3)
False

>>> from sage.all import *
>>> K.maximal_order().is_maximal(p=Integer(3))
True
```

is_suborder \((\text{other})\)

Return \text{True} if self is a subset of the order \text{other}.
## 4.1. Orders in number fields

### Example:

```sage
x = polygen(ZZ, 'x')
K.<a,b> = NumberField([x^2 + 1, x^3 + 2])
R1 = K.order([a, b])
R2 = K.order([2*a, b])
R3 = K.order([a + b, b + 2*a])
R1.is_suborder(R2)
False
R2.is_suborder(R1)
True
R3.is_suborder(R1)
True
R1.is_suborder(R3)
True
R1 == R3
True
```

```python
class sage.rings.number_field.order.RelativeOrderFactory
    Bases: OrderFactory
    
    An order in a relative number field extension.

    EXAMPLES:
```

```sage
x = polygen(ZZ, 'x')
K.<i> = NumberField(x^2 + 1)
R.<j> = K[]
L.<j> = K.extension(j^2 - 2)
L.order([i, j])
Relative Order generated by [-i*j + 1, -i] in Number Field in j with defining polynomial j^2 - 2 over its base field
```
create_key_and_extra_args \((K, \text{absolute\_order}, \text{is\_maximal}=\text{None}, \text{check}=\text{True}, \text{is\_maximal\_at}())\)

Return normalized arguments to create a relative order.

create_object \((\text{version}, \text{key}, \text{is\_maximal}=\text{None}, \text{is\_maximal\_at}())\)

Create a relative order.

sage.rings.number_field.order.absolute_order_from_module_generators \((\text{gens}, \text{check\_integral}=\text{True}, \text{check\_rank}=\text{True}, \text{check\_is\_ring}=\text{True}, \text{is\_maximal}=\text{None}, \text{allow\_subfield}=\text{False}, \text{is\_maximal\_at}())\)

INPUT:

• \text{gens} – list of elements of an absolute number field that generates an order in that number field as a \(\mathbb{Z}\)-module.

• \text{check\_integral} – check that each generator is integral

• \text{check\_rank} – check that the \text{gens} span a module of the correct rank

• \text{check\_is\_ring} – check that the module is closed under multiplication (this is very expensive)

• \text{is\_maximal} – \text{bool (or None)}; set if maximality of the generated order is known

• \text{is\_maximal\_at} – a tuple of primes where this order is known to be maximal

OUTPUT:

an absolute order

EXAMPLES:

We have to explicitly import the function, since it is not meant for regular usage:

```python
sage: from sage.rings.number_field.order import absolute_order_from_module_generators

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 - 5)
sage: O = K.maximal_order(); O
Maximal Order generated by [1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a] in Number Field in a
with defining polynomial x^4 - 5

sage: O.basis()
[1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a, a^2, a^3]
sage: O.module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[ 1/2 0 1/2 0]
[ 0 1/2 0 1/2]
[ 0 0 1 0]
[ 0 0 0 1]
```

(continues on next page)
sage: g = O.basis(); g
[1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a, a^2, a^3]
sage: absolute_order_from_module_generators(g)
Maximal Order generated by [1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a] in Number Field in a
with defining polynomial x^4 - 5

We illustrate each check flag – the output is the same but in case the function would run ever so slightly faster:

sage: absolute_order_from_module_generators(g, check_is_ring=False)
Maximal Order generated by [1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a] in Number Field in a
with defining polynomial x^4 - 5
sage: absolute_order_from_module_generators(g, check_rank=False)
Maximal Order generated by [1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a] in Number Field in a
with defining polynomial x^4 - 5
sage: absolute_order_from_module_generators(g, check_integral=False)
Maximal Order generated by [1/2*a^2 + 1/2, 1/2*a^3 + 1/2*a] in Number Field in a
with defining polynomial x^4 - 5

Next we illustrate constructing “fake” orders to illustrate turning off various check flags:

>>> from sage.all import *

We illustrate each check flag – the output is the same but in case the function would run ever so slightly faster:
sage: k.<i> = NumberField(x^2 + 1)
sage: R = absolute_order_from_module_generators([2, 2*i],
.....:      check_is_ring=False); R
Order of conductor 4 generated by [2, 2*i]
in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[2, 2*i]

sage: R = absolute_order_from_module_generators([k(1)],
.....:      check_rank=False); R
Order of conductor I generated by []
in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1]

If the order contains a non-integral element, even if we do not check that, we will find that the rank is wrong or that the order is not closed under multiplication:

sage: absolute_order_from_module_generators([1/2, i],
.....:      check_integral=False)
Traceback (most recent call last):
...
ValueError: the module span of the gens is not closed under multiplication.
sage: R = absolute_order_from_module_generators([1/2, i],
.....:      check_is_ring=False,
.....:      check_integral=False); R
Order of conductor 0 generated by [1/2, i] in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1/2, 1]

>>> from sage.all import *
>>> k = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = k._first_ngens(1)
>>> R = absolute_order_from_module_generators([Integer(2), Integer(2)*i],
.....:      check_is_ring=False); R
Order of conductor 4 generated by [2, 2*i]
in Number Field in i with defining polynomial x^2 + 1
>>> R.basis()
[2, 2*i]

>>> R = absolute_order_from_module_generators([k(Integer(1))],
.....:      check_rank=False); R
Order of conductor I generated by []
in Number Field in i with defining polynomial x^2 + 1
>>> R.basis()
[1]

(continues on next page)
We turn off all check flags and make a really messed up order:

```
sage: R = absolute_order_from_module_generators([1/2, i],
....:     check_is_ring=False,
....:     check_integral=False,
....:     check_rank=False); R
Order of conductor 0 generated by [1/2, i] in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1/2, i]
```

An order that lives in a subfield:

```
sage: F.<alpha> = NumberField(x**4 + 3)
sage: F.order([alpha**2], allow_subfield=True)
Order of conductor 2 generated by ... in Number Field in beta with defining polynomial ... with beta = ...
```

```
sage: F = NumberField(x**Integer(4) + Integer(3), names=('alpha',)); (alpha,) = F._first_ngens(1)
sage: F.order([alpha**Integer(2)], allow_subfield=True)
Order of conductor 2 generated by ... in Number Field in beta with defining polynomial ... with beta = ...
```

```
sage.rings.number_field.order.absolute_order_from_ring_generators(gens, check_is_integral=True, check_rank=True, is_maximal=None, allow_subfield=False)
```

**INPUT:**

- `gens` — list of integral elements of an absolute order.
- `check_is_integral` — bool (default: True), whether to check that each generator is integral.
- `check_rank` — bool (default: True), whether to check that the ring generated by `gens` is of full rank.
- `is_maximal` — bool (or None); set if maximality of the generated order is known
- `allow_subfield` — bool (default: False), if True and the generators do not generate an order, i.e., they generate a subring of smaller rank, instead of raising an error, return an order in a smaller number field.
EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 - 5)
sage: K.order(a)
Order generated by a in Number Field in a with defining polynomial x^4 - 5
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> K.order(a)
Order generated by a in Number Field in a with defining polynomial x^4 - 5
```

We have to explicitly import this function, since typically it is called with `K.order` as above.

```
sage: from sage.rings.number_field.order import absolute_order_from_ring_generators
sage: absolute_order_from_ring_generators([a])
Order generated by a in Number Field in a with defining polynomial x^4 - 5
```

```
>>> from sage.all import *
>>> from sage.rings.number_field.order import absolute_order_from_ring_generators

>>> absolute_order_from_ring_generators([a])
Order generated by a in Number Field in a with defining polynomial x^4 - 5
```

If one of the inputs is non-integral, it is an error.

```
sage: absolute_order_from_ring_generators([a/2])
Traceback (most recent call last):
...
ValueError: each generator must be integral
```

```
>>> from sage.all import *

>>> absolute_order_from_ring_generators([a/Integer(2)])
Traceback (most recent call last):
...
ValueError: each generator must be integral
```

If the `gens` do not generate an order, i.e., generate a ring of full rank, then it is an error.

```
sage: absolute_order_from_ring_generators([a^2])
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
```

```
>>> from sage.all import *

>>> absolute_order_from_ring_generators([a**Integer(2)])
Traceback (most recent call last):
...
ValueError: the rank of the span of gens is wrong
```
Both checking for integrality and checking for full rank can be turned off in order to save time, though one can get nonsense as illustrated below:

```
sage: absolute_order_from_ring_generators([a/2], check_is_integral=False)
Order generated by [1, 1/2*a, 1/4*a^2, 1/8*a^3] in Number Field in a with defining polynomial x^4 - 5
```

```
sage: absolute_order_from_ring_generators([a^2], check_rank=False)
Order generated by a^2 in Number Field in a with defining polynomial x^4 - 5
```

```python
def absolute_order_from_ring_generators(v, check_is_integral=True, check_rank=True):
    # Implementation...
```

Return whether every element of the list $v$ of elements of a number field is integral.

**EXAMPLES:**

```
sage: x = polygen(ZZ, 'x')
sage: W.<sqrt5> = NumberField(x^2 - 5)
sage: from sage.rings.number_field.order import each_is_integral
sage: each_is_integral([sqrt5, 2, (1+sqrt5)/2])
True
sage: each_is_integral([sqrt5, (1+sqrt5)/3])
False
```

```
sage.rings.number_field.order.is_NumberFieldOrder(R)
Return True if $R$ is either an order in a number field or is the ring $\mathbb{Z}$ of integers.

**EXAMPLES:**

```
sage: from sage.rings.number_field.order import is_NumberFieldOrder
sage: x = polygen(ZZ, 'x')
sage: W = NumberField(x^2 + 1, 'a').maximal_order(); (sqrt5,) = W._first_ngens(1)
sage: from sage.rings.number_field.order import is_integral
sage: is_integral([sqrt5, Integer(1)+sqrt5/2])
True
sage: is_integral([sqrt5, Integer(1)+sqrt5/Integer(2)])
False
```

4.1. Orders in number fields
```python
>>> from sage.all import *
>>> from sage.rings.number_field.order import is_NumberFieldOrder

>>> x = polygen(ZZ, 'x')

>>> is_NumberFieldOrder(NumberField(x**Integer(2) + Integer(1), 'a').maximal_order())

doctest:warning...
DeprecationWarning: The function is_NumberFieldOrder is deprecated; use isinstance(..., sage.rings.abc.Order) or ... == ZZ instead. See https://github.com/sagemath/sage/issues/38124 for details.
True

>>> is_NumberFieldOrder(ZZ)
True

>>> is_NumberFieldOrder(QQ)
False

>>> is_NumberFieldOrder(Integer(45))
False

sage.rings.number_field.order.quadratic_order_class_number(disc)
Return the class number of the quadratic order of given discriminant.

EXAMPLES:

```
```python
sage: from sage.rings.number_field.order import relative_order_from_ring_generators
sage: x = polygen(ZZ, 'x')
sage: K.<i, a> = NumberField([x^2 + 1, x^2 - 17])
sage: R = K.base_field().maximal_order()
sage: S = relative_order_from_ring_generators([i,a]); S
Relative Order generated by [7*i - 2*a, -a*i + 8, 25*i - 7*a] in
Number Field in i with defining polynomial x^2 + 1 over its base field
```

Basis for the relative order, which is obtained by computing the algebra generated by i and a:

```python
sage: S.basis()
[1, 7*i - 2*a, -a*i + 8, 25*i - 7*a]
```

### 4.2 Ideals of number fields

AUTHORS:

- Steven Sivek (2005-05-16): initial version
- William Stein (2007-09-06): vastly improved the doctesting
- William Stein and John Cremona (2007-01-28): new class `NumberFieldFractionalIdeal` now used for all except the 0 ideal
- Radoslav Kirov and Alyson Deines (2010-06-22): `prime_to_S_part`, `is_S_unit`, `is_S_integral`

```python
class sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal(field, gens, coerce=True)
```

A fractional ideal in a number field.

```python
class sage.rings.number_field.number_field_ideal.LiftMap(OK, M_OK_map, Q, I)
```

Class to hold data needed by lifting maps from residue fields to number field orders.

```python
class sage.rings.number_field.number_field_ideal.NumberFieldIdeal(field, gens, coerce=True)
```

Bases: `MultiplicativeGroupElement, NumberFieldIdeal, Ideal_fractional`

EXAMPLES:
denominator()

Return the denominator ideal of this fractional ideal. Each fractional ideal has a unique expression as $N/D$ where $N, D$ are coprime integral ideals; the denominator is $D$.

EXAMPLES:

```python
dsage: x = polygen(ZZ)
dsage: R.<x> = PolynomialRing(QQ)
dsage: K.<a> = NumberField(x^3 - 2)
dsage: I = K.ideal(2/(5+a))
dsage: J = I^2
sage: Jinv = I^(-2)
sage: J*Jinv
Fractional ideal (1)
```

```python
denominator()
Return the denominator ideal of this fractional ideal. Each fractional ideal has a unique expression as $N/D$ where $N, D$ are coprime integral ideals; the denominator is $D$.

EXAMPLES:

```python
dsage: x = polygen(ZZ)
dsage: R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
dsage: K = NumberField(x**Integer(3) - Integer(2), names=('a',)); (a,) = K._first_ngens(1)
dsage: I = K.ideal(Integer(2)/(Integer(5)+a))
dsage: J = I**Integer(2)
dsage: Jinv = I**(-Integer(2))
dsage: J*Jinv
Fractional ideal (1)
```

```python
dsage: x = polygen(ZZ)
dsage: K.<i> = NumberField(x^2 + 1)
dsage: I = K.ideal((3+4*i)/5); I
Fractional ideal (4/5*i + 3/5)
dsage: I.denominator()
Fractional ideal (2*i + 1)
dsage: I.numerator()
Fractional ideal (-i - 2)
dsage: I.numerator().is_integral() and I.denominator().is_integral()
True
dsage: I.numerator() + I.denominator() == K.unit_ideal()
True
dsage: I.numerator()/I.denominator() == I
True
```

```python
denominator()
Return the denominator ideal of this fractional ideal. Each fractional ideal has a unique expression as $N/D$ where $N, D$ are coprime integral ideals; the denominator is $D$.

EXAMPLES:

```python
dsage: x = polygen(ZZ)
dsage: K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)
dsage: I = K.ideal((Integer(3)+Integer(4)*i)/Integer(5)); I
Fractional ideal (4/5*i + 3/5)
dsage: I.denominator()
Fractional ideal (2*i + 1)
dsage: I.numerator()
Fractional ideal (-i - 2)
dsage: I.numerator().is_integral() and I.denominator().is_integral()
True
dsage: I.numerator() + I.denominator() == K.unit_ideal()
True
dsage: I.numerator()/I.denominator() == I
True
```

```python
denominator()
Return the denominator ideal of this fractional ideal. Each fractional ideal has a unique expression as $N/D$ where $N, D$ are coprime integral ideals; the denominator is $D$.

EXAMPLES:

```python
denominator()
```
\textbf{divides}(\textit{other})

Return \textbf{True} if this ideal divides \textit{other} and \textbf{False} otherwise.

**EXAMPLES:**

```
sage: K.<a> = CyclotomicField(11); K
Cyclotomic Field of order 11 and degree 10
sage: I = K.factor(31)[0][0]; I
Fractional ideal (31, a^5 + 10*a^4 - a^3 + a^2 + 9*a - 1)
sage: I.divides(I)
True
sage: I.divides(31)
True
sage: I.divides(29)
False
```

\textbf{element\textunderscore 1\_mod}(\textit{other})

Return an element \(r\) in this ideal such that \(1 - r\) is in \textit{other}

An error is raised if either ideal is not integral of if they are not coprime.

**INPUT:**

- \textit{other} – another ideal of the same field, or generators of an ideal.

**OUTPUT:**

An element \(r\) of the ideal \textit{self} such that \(1 - r\) is in the ideal \textit{other}

**AUTHOR:** Maite Aranes (modified to use PARI’s \texttt{pari:idealaddtoone} by Francis Clarke)

**EXAMPLES:**

```
>>> from sage.all import *
>>> K = CyclotomicField(Integer(11), names=('a',)); (a,) = K._first_ngens(1);
>>> K
Cyclotomic Field of order 11 and degree 10
>>> I = K.factor(Integer(31))[Integer(0)][Integer(0)]; I
Fractional ideal (31, a^5 + 10*a^4 - a^3 + a^2 + 9*a - 1)
>>> I.divides(I)
True
>>> I.divides(Integer(31))
True
>>> I.divides(Integer(29))
False
```
euler_phi()

Return the Euler $\varphi$-function of this integral ideal.

This is the order of the multiplicative group of the quotient modulo the ideal.

An error is raised if the ideal is not integral.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: I = K.ideal(2 + i)
sage: [r for r in I.residues() if I.is_coprime(r)]
[-2*i, -i, i, 2*i]
sage: I.euler_phi()
4
sage: J = I^3
sage: J.euler_phi()
100
sage: len([r for r in J.residues() if J.is_coprime(r)])
100
sage: J = K.ideal(3 - 2*i)
sage: I.is_coprime(J)
True
sage: I.euler_phi()*J.euler_phi() == (I*J).euler_phi()
True
sage: L.<b> = K.extension(x^2 - 7)
sage: L.ideal(3).euler_phi()
64
```
100
>>> len([r for r in J.residues() if J.is_coprime(r)])
100
>>> J = K.ideal(Integer(3) - Integer(2)*i)
>>> J.is_coprime(J)
True
>>> I.euler_phi() * J.euler_phi() == (I * J).euler_phi()
True
>>> L = K.extension(x**Integer(2) - Integer(7), names=('b',)); (b,) = L._→first_ngens(1)
>>> L.ideal(Integer(3)).euler_phi()
64

factor()
Factorization of this ideal in terms of prime ideals.

EXAMPLES:

```sage
x = polygen(ZZ)
K.<a> = NumberField(x^4 + 23); K
Number Field in a with defining polynomial x^4 + 23
I = K.ideal(19); I
Fractional ideal (19)
F = I.factor(); F
(Fractional ideal (19, 1/2*a^2 + a - 17/2))
* (Fractional ideal (19, 1/2*a^2 - a - 17/2))
type(F)
<class 'sage.structure.factorization.Factorization'>
list(F)
[(Fractional ideal (19, 1/2*a^2 + a - 17/2), 1),
 (Fractional ideal (19, 1/2*a^2 - a - 17/2), 1)]
F.prod()
Fractional ideal (19)
```

```sage
from sage.all import *
x = polygen(ZZ)
K = NumberField(x**Integer(4) + Integer(23), names=('a',)); (a,) = K._←first_ngens(1); K
Number Field in a with defining polynomial x^4 + 23
I = K.ideal(Integer(19)); I
Fractional ideal (19)
F = I.factor(); F
(Fractional ideal (19, 1/2*a^2 + a - 17/2))
* (Fractional ideal (19, 1/2*a^2 - a - 17/2))
type(F)
<class 'sage.structure.factorization.Factorization'>
list(F)
[(Fractional ideal (19, 1/2*a^2 + a - 17/2), 1),
 (Fractional ideal (19, 1/2*a^2 - a - 17/2), 1)]
F.prod()
Fractional ideal (19)
```

idealcoprime(J)
Return l such that l * self is coprime to J.

INPUT:

- J – another integral ideal of the same field as self, which must also be integral.

4.2. Ideals of number fields
an element \( l \) such that \( l \text{self} \) is coprime to the ideal \( J \)

Todo: Extend the implementation to non-integral ideals.

EXAMPLES:

```sage
sage: x = polygen(ZZ)
sage: k.<a> = NumberField(x^2 + 23)
sage: A = k.ideal(a + 1)
sage: B = k.ideal(3)
sage: A.is_coprime(B)
False
sage: lam = A.idealcoprime(B)
# representation depends, not tested
-1/6*a + 1/6
sage: (lam*A).is_coprime(B)
True
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> k = NumberField(x**Integer(2) + Integer(23), names=('a',)); (a,) = k._
˓→first_ngens(1)
>>> A = k.ideal(a + Integer(1))
>>> B = k.ideal(Integer(3))
>>> A.is_coprime(B)
False
>>> lam = A.idealcoprime(B)
# representation depends, not tested
-1/6*a + 1/6
>>> (lam*A).is_coprime(B)
True
```

ALGORITHM: Uses Pari function pari:idealcoprime.

\textbf{ideallog}(x, gens=None, check=True)

Return the discrete logarithm of \( x \) with respect to the generators given in the bid structure of the ideal \( \text{self} \), or with respect to the generators \( \text{gens} \) if these are given.

INPUT:

- \( x \) – a non-zero element of the number field of \( \text{self} \), which must have valuation equal to 0 at all prime ideals in the support of the ideal \( \text{self} \).
- \( \text{gens} \) – a list of elements of the number field which generate \((R/I)^*\), where \( R \) is the ring of integers of the field and \( I \) is this ideal, or None. If None, use the generators calculated by \text{idealstar()}.
- \( \text{check} \) – if True, do a consistency check on the results. Ignored if \( \text{gens} \) is None.

OUTPUT:

a list of non-negative integers \( (x_i) \) such that \( x = \prod_i g_i^{x_i} \) in \((R/I)^*\), where \( g_i \) are the generators, and the list \( (x_i) \) is lexicographically minimal with respect to this requirement. If the \( x_i \) generate independent cyclic factors of order \( d_i \), as is the case for the default generators calculated by \text{idealstar()}, this just means that \( 0 \leq x_i < d_i \).

A \text{ValueError} will be raised if the elements specified in \( \text{gens} \) do not in fact generate the unit group (even if the element \( x \) is in the subgroup they generate).
EXAMPLES:

```
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> k.<a> = NumberField(x**Integer(3) - Integer(11), names=('a',)); (a,) = k._first_ngens(1)
>>> A = k.ideal(Integer(5))
>>> G = A.idealstar(Integer(2))
>>> l = A.ideallog(a**Integer(2) + Integer(3))
>>> r = G(l).value()
>>> (a**Integer(2) + Integer(3)) - r
True
>>> A.small_residue(r) # random
a^2 - 2
```

Examples with custom generators:

```
>>> K.<a> = NumberField(x**Integer(2) - Integer(7), names=('a',)); (a,) = K._first_ngens(1)
>>> I = K.ideal(Integer(17))
>>> I.ideallog(a + Integer(7), [1 + a, 2])
[10, 3]
>>> I.ideallog(a + Integer(7), [2, 1 + a])
[0, 118]
```

```
>>> L.<b> = NumberField(x**Integer(4) - x**Integer(3) - Integer(7)*x**Integer(2) + Integer(3)*x + Integer(2), names=('b',)); (b,) = L._first_ngens(1)
>>> J = L.ideal(-b**Integer(3) - b**Integer(2) - Integer(2))
>>> u = -Integer(14)*b**Integer(3) + Integer(21)*b**Integer(2) + b - Integer(1)
>>> v = Integer(4)*b**Integer(2) + Integer(2)*b - Integer(1)
```

(continues on next page)
A non-example:

```
sage: I.ideallog(a + 7, [2])
Traceback (most recent call last):
...
ValueError: Given elements do not generate unit group --
they generate a subgroup of index 36
```

ALGORITHM: Uses PARI function `pari:ideallog`, and (if `gens` is not None) a Hermite normal form calculation to express the result in terms of the generators `gens`.

```
idealstar (flag=1)
```

Return the finite abelian group `(OK/I)^*`, where `I` is the ideal self of the number field `K`, and `OK` is the ring of integers of `K`.

INPUT:

- `flag` (int default 1) – when `flag` = 2, it also computes the generators of the group `(OK/I)^*`, which takes more time. By default `flag` = 1 (no generators are computed). In both cases the special PARI structure `bid` is computed as well. If `flag` = 0 (deprecated) it computes only the group structure of `(OK/I)^*` (with generators) and not the special `bid` structure.

OUTPUT:

The finite abelian group `(OK/I)^*`.

Note: Uses the PARI function `pari:idealstar`. The PARI function outputs a special `bid` structure which is stored in the internal field `_bid` of the ideal (when `flag = 1,2`). The special structure `bid` is used in the PARI function `pari:ideallog` to compute discrete logarithms.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: k.<a> = NumberField(x^3 - 11)
sage: A = k.ideal(5)
sage: G = A.idealstar(); G
Multiplicative Abelian group isomorphic to C24 x C4
```

```
sage: G = A.idealstar(2)
sage: G.gens()
(f0, f1)
sage: G.gens_values()  # random output
(2*a^2 - 1, 2*a^2 + 2*a - 2)
```
sage: all(G.gen(i).value() in k for i in range(G.ngens()))
True

>>> from sage.all import *
>>> x = polygen(ZZ)
>>> k = NumberField(x**Integer(3) - Integer(11), names=('a',)); (a,)
   = k._first_ngens(1)
>>> A = k.ideal(Integer(5))
>>> G = A.idealstar(); G
Multiplicative Abelian group isomorphic to C24 x C4
>>> G.gens()
(f0, f1)
>>> G = A.idealstar(Integer(2))
>>> G.gens()
(f0, f1)
>>> G.gens_values()  # random output
(2*a^2 - 1, 2*a^2 + 2*a - 2)
>>> all(G.gen(i).value() in k for i in range(G.ngens()))
True

ALGORITHM: Uses Pari function pari:idealstar

invertible_residues (reduce=True)

Return an iterator through a list of invertible residues modulo this integral ideal.

An error is raised if this fractional ideal is not integral.

INPUT:
- reduce – bool. If True (default), use small_residue to get small representatives of the residues.

OUTPUT:
An iterator through a list of invertible residues modulo this ideal \( I \), i.e. a list of elements in the ring of integers \( R \) representing the elements of \( (R/I)^* \).

ALGORITHM: Use pari:idealstar to find the group structure and generators of the multiplicative group modulo the ideal.

EXAMPLES:

sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: ires = K.ideal(2).invertible_residues(); ires
xmrange_iter([[-1, 0]], <function ...<lambda> at 0x...>)
sage: list(ires)
[1, -i]
sage: list(K.ideal(2 + i).invertible_residues())
[1, 2, 4, 3]
sage: list(K.ideal(i).residues())
[0]
sage: list(K.ideal(i).invertible_residues())
[1]
sage: I = K.ideal(3 + 6*i)
sage: units = I.invertible_residues()
sage: len(list(units)) == I.euler_phi()
True
AUTHOR: John Cremona

invertible_residues_mod(subgp_gens=[], reduce=True)

Return a iterator through a list of representatives for the invertible residues modulo this integral ideal, modulo the subgroup generated by the elements in the list subgp_gens.

INPUT:

- subgp_gens – either None or a list of elements of the number field of self. These need not be integral, but should be coprime to the ideal self. If the list is empty or None, the function returns an iterator through a list of representatives for the invertible residues modulo the integral ideal self.

- reduce – bool. If True (default), use small_residues to get small representatives of the residues.

Note: See also invertible_residues() for a simpler version without the subgroup.

OUTPUT:

An iterator through a list of representatives for the invertible residues modulo self and modulo the group
generated by subgp_gens, i.e. a list of elements in the ring of integers \( R \) representing the elements of \((R/I)^*/U\), where \( I \) is this ideal and \( U \) is the subgroup of \((R/I)^*\) generated by subgp_gens.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: k.<a> = NumberField(x^2 + 23)
sage: I = k.ideal(a)
sage: list(I.invertible_residues_mod([-1]))
[1, 5, 2, 10, 4, 20, 8, 17, 16, 11, 9]
sage: list(I.invertible_residues_mod([1/2]))
[1, 5]
sage: list(I.invertible_residues_mod([23]))
Traceback (most recent call last):
... TypeError: the element must be invertible mod the ideal
```

```
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> k = NumberField(x**Integer(2) + Integer(23), names=('a',)); (a,) = k._first_ngens(1)
>>> I = k.ideal(a)
>>> list(I.invertible_residues_mod([-Integer(1)]))
[1, 5, 2, 10, 4, 20, 8, 17, 16, 11, 9]
>>> list(I.invertible_residues_mod([Integer(1)/Integer(2)]))
[1, 5]
>>> list(I.invertible_residues_mod([Integer(23)]))
Traceback (most recent call last):
... TypeError: the element must be invertible mod the ideal
```

```
sage: K.<a> = NumberField(x^3 - 10)
sage: I = K.ideal(a - 1)
sage: len(list(I.invertible_residues_mod([]))) == I.euler_phi()
True

sage: I = K.ideal(1)
sage: list(I.invertible_residues_mod([]))
[1]
```

```
>>> from sage.all import *
>>> K = NumberField(x**Integer(3) - Integer(10), names=('a',)); (a,) = K._first_ngens(1)
>>> I = K.ideal(a - Integer(1))
>>> len(list(I.invertible_residues_mod([]))) == K.euler_phi()
True

>>> I = K.ideal(Integer(1))
>>> list(I.invertible_residues_mod([]))
[1]
```

```
sage: K.<z> = CyclotomicField(10)
sage: len(list(K.primes_above(3)[0].invertible_residues_mod([])))
80
```

```
>>> from sage.all import *
(continues on next page)
```
AUTHOR: Maite Aranes.

is_S_integral (S)

Return True if this fractional ideal is integral with respect to the list of primes S.

INPUT:

• S – a list of prime ideals (not checked if they are indeed prime).

Note: This function assumes that S is a list of prime ideals, but does not check this. This function will fail if S is not a list of prime ideals.

OUTPUT:

True, if the ideal is S-integral: that is, if the valuations of the ideal at all primes not in S are non-negative. False, otherwise.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^2 + 23)
sage: I = K.ideal(1/2)
sage: P = K.ideal(2, 1/2*a - 1/2)
sage: I.is_S_integral([P])
False

sage: J = K.ideal(1/5)
sage: J.is_S_integral([K.ideal(5)])
True
```

is_S_unit (S)

Return True if this fractional ideal is a unit with respect to the list of primes S.

INPUT:

• S – a list of prime ideals (not checked if they are indeed prime).
Note: This function assumes that $S$ is a list of prime ideals, but does not check this. This function will fail if $S$ is not a list of prime ideals.

OUTPUT:

True, if the ideal is an $S$-unit: that is, if the valuations of the ideal at all primes not in $S$ are zero. False, otherwise.

EXAMPLES:

```sage
x = polygen(ZZ)
sage: K.<a> = NumberField(x^2 + 23)
sage: I = K.ideal(2)
sage: P = I.factor()[0][0]
sage: I.is_S_unit([P])
False
```

```python
>>> from sage.all import *
```
is_maximal()

Return True if this ideal is maximal. This is equivalent to self being prime, since it is nonzero.

EXAMPLES:

sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^3 + 3); K
Number Field in a with defining polynomial x^3 + 3
sage: K.ideal(5).is_maximal()
False
sage: K.ideal(7).is_maximal()
True
```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(3) + Integer(3), names=('a',)); (a,) = K._first_ngens(1)
Number Field in a with defining polynomial x^3 + 3
>>> K.ideal(Integer(5)).is_maximal()
False
>>> K.ideal(Integer(7)).is_maximal()
True

is_trivial (proof=None)
Return True if this is a trivial ideal.

EXAMPLES:

sage: F.<a> = QuadraticField(-5)
sage: I = F.ideal(3)
sage: I.is_trivial()  
False
sage: J = F.ideal(5)
sage: J.is_trivial()  
False
sage: (I + J).is_trivial()  
True

>>> from sage.all import *
>>> F = QuadraticField(-Integer(5), names=('a',)); (a,) = F._first_ngens(1)
>>> I = F.ideal(Integer(3))
>>> I.is_trivial()  
False
>>> J = F.ideal(Integer(5))
>>> J.is_trivial()  
False
>>> (I + J).is_trivial()  
True

numerator()
Return the numerator ideal of this fractional ideal.

Each fractional ideal has a unique expression as \( \frac{N}{D} \) where \( N, D \) are coprime integral ideals. The numerator is \( N \).

EXAMPLES:

sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: I = K.ideal((3+4*i)/5); I
Fractional ideal (4/5*i + 3/5)
sage: I.denominator()
Fractional ideal (2*i + 1)
sage: I.numerator()  
Fractional ideal (-i - 2)
sage: I.numerator().is_integral() and I.denominator().is_integral()  
True
sage: I.numerator() + I.denominator() == K.unit_ideal()  
True
sage: I.numerator()/I.denominator() == I  
True
```

4.2. Ideals of number fields
```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> I = K.ideal((Integer(3)+Integer(4)*i)/Integer(5)); I
Fractional ideal (4/5*i + 3/5)
>>> I.denominator()
Fractional ideal (2*i + 1)
>>> I.numerator()
Fractional ideal (-i - 2)
>>> I.numerator().is_integral() and I.denominator().is_integral()
True
>>> I.numerator() + I.denominator() == K.unit_ideal()
True
>>> I.numerator()/I.denominator() == I
True
```

**prime_factors()**

Return a list of the prime ideal factors of `self`.

**OUTPUT:**

list of prime ideals (a new list is returned each time this function is called)

**EXAMPLES:**

```python
sage: x = polygen(ZZ)
sage: K.<w> = NumberField(x^2 + 23)
sage: I = ideal(w+1)
sage: I.prime_factors()
[Fractional ideal (2, 1/2*w - 1/2),
 Fractional ideal (2, 1/2*w + 1/2),
 Fractional ideal (3, 1/2*w + 1/2)]
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(23), names=('w',)); (w,) = K._first_ngens(1)
>>> I = ideal(w+Integer(1))
>>> I.prime_factors()
[Fractional ideal (2, 1/2*w - 1/2),
 Fractional ideal (2, 1/2*w + 1/2),
 Fractional ideal (3, 1/2*w + 1/2)]
```

**prime_to_S_part(S)**

Return the part of this fractional ideal which is coprime to the prime ideals in the list `S`.

**Note:** This function assumes that `S` is a list of prime ideals, but does not check this. This function will fail if `S` is not a list of prime ideals.

**INPUT:**

- `S` – a list of prime ideals

**OUTPUT:**

A fractional ideal coprime to the primes in `S`, whose prime factorization is that of `self` with the primes in `S` removed.
EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^2 - 23)
sage: I = K.ideal(24)
sage: S = [K.ideal(-a + 5), K.ideal(5)]
sage: I.prime_to_S_part(S)
Fractional ideal (3)
sage: J = K.ideal(15)
sage: J.prime_to_S_part(S)
Fractional ideal (3)
```

```python
sage: K.<a> = NumberField(x^5 - 23)
sage: I = K.ideal(24)
sage: S = [K.ideal(15161*a^4 + 28383*a^3 + 53135*a^2 + 99478*a + 186250), ...
     K.ideal(2*a^4 + 3*a^3 + 4*a^2 + 15*a + 11), ...
     K.ideal(101)]
sage: I.prime_to_S_part(S)
Fractional ideal (24)
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) - Integer(23), names=('a',)); (a,) = K._
    first_ngens(1)
>>> I = K.ideal(Integer(24))
>>> S = [K.ideal(-a + Integer(5)), K.ideal(Integer(5))]
>>> I.prime_to_S_part(S)
Fractional ideal (3)
>>> J = K.ideal(Integer(15))
>>> J.prime_to_S_part(S)
Fractional ideal (3)
```

```python
>>> K = NumberField(x**Integer(5) - Integer(23), names=('a',)); (a,) = K._
    first_ngens(1)
>>> I = K.ideal(Integer(24))
>>> S = [K.ideal(Integer(15161)*a^4 + Integer(28383)*a^3 + Integer(53135)*a^2 + Integer(99478)*a + Integer(186250)), ...
     K.ideal(2*a^4 + 3*a^3 + 4*a^2 + 15*a + 11), ...
     K.ideal(Integer(101))]
>>> I.prime_to_S_part(S)
Fractional ideal (24)
```

`prime_to_idealM_part(M)`

Version for integral ideals of the `prime_to_m_part` function over $\mathbb{Z}$. Return the largest divisor of $\text{self}$ that is coprime to the ideal $M$.

**INPUT:**

- $M$ – an integral ideal of the same field, or generators of an ideal

**OUTPUT:**

An ideal which is the largest divisor of $\text{self}$ that is coprime to $M$.

**AUTHOR:** Maite Aranes

**EXAMPLES:**
sage: x = polygen(ZZ)
sage: k.<a> = NumberField(x^2 + 23)
sage: I = k.ideal(a + 1)
sage: M = k.ideal(2, 1/2*a - 1/2)
sage: J = I.prime_to_idealM_part(M); J  
Fractional ideal (12, 1/2*a + 13/2)
sage: J.is_coprime(M)  
True

sage: J = I.prime_to_idealM_part(2); J  
Fractional ideal (3, 1/2*a + 1/2)
sage: J.is_coprime(M)  
True

>>> from sage.all import *

>>> x = polygen(ZZ)

>>> k = NumberField(x**Integer(2) + Integer(23), names=('a',)); (a,) = k._
˓→first_ngens()

>>> I = k.ideal(a + Integer(1))

>>> M = k.ideal(Integer(2), Integer(1)/Integer(2)*a - Integer(1)/Integer(2))

>>> J = I.prime_to_idealM_part(M); J  
Fractional ideal (12, 1/2*a + 13/2)

>>> J.is_coprime(M)  
True

>>> J = I.prime_to_idealM_part(Integer(2)); J  
Fractional ideal (3, 1/2*a + 1/2)

>>> J.is_coprime(M)  
True

\textbf{ramification\_index()}  
Return the ramification index of this fractional ideal, assuming it is prime. Otherwise, raise a \texttt{ValueError}.  
The ramification index is the power of this prime appearing in the factorization of the prime in $\mathbb{Z}$ that this prime lies over.  

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^2 + 2); K
Number Field in a with defining polynomial x^2 + 2
sage: f = K.factor(2); f
(Fractional ideal (a))^2
sage: f[0][0].ramification_index()  
2
sage: K.ideal(13).ramification_index()  
1
sage: K.ideal(17).ramification_index()  
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not a prime ideal
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *

>>> x = polygen(ZZ)

>>> K = NumberField(x**Integer(2) + Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(); K
Number Field in a with defining polynomial x^2 + 2
\end{verbatim}
```python
>>> f = K.factor(Integer(2)); f
(Fractional ideal (a))^2
>>> f[Integer(0)][Integer(0)].ramification_index()
2
>>> K.ideal(Integer(13)).ramification_index()
1
>>> K.ideal(Integer(17)).ramification_index()
Traceback (most recent call last):
...
ValueError: Fractional ideal (17) is not a prime ideal
```

**ray_class_number()**

Return the order of the ray class group modulo this ideal. This is a wrapper around PARI's `pari:bnrclassno` function.

**EXAMPLES:**

```python
sage: K.<z> = QuadraticField(-23)
sage: p = K.primes_above(3)[0]
sage: p.ray_class_number()
3
sage: x = polygen(K)
sage: L.<w> = K.extension(x^3 - z)
sage: I = L.ideal(5)
sage: I.ray_class_number()
5184
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(23), names=('z',)); (z,) = K._first_ngens(1)
>>> p = K.primes_above(Integer(3))[Integer(0)]
>>> p.ray_class_number()
3
>>> x = polygen(K)
>>> L = K.extension(x**Integer(3) - z, names=('w',)); (w,) = L._first_ngens(1)
>>> I = L.ideal(Integer(5))
>>> I.ray_class_number()
5184
```

**reduce** *(f)*

Return the canonical reduction of the element *f* modulo the ideal *I* (= *self*). This is an element of *R* (the ring of integers of the number field) that is equivalent modulo *I* to *f*.

An error is raised if this fractional ideal is not integral or the element *f* is not integral.

**INPUT:**

- *f* – an integral element of the number field

**OUTPUT:**

An integral element *g*, such that *f* − *g* belongs to the ideal *self* and such that *g* is a canonical reduced representative of the coset *f* + *I* (where *I* = *self*) as described in the method `residues()`, namely an integral element with coordinates (*r*₀, ..., *r*ₙ₋₁), where:

- *r*₂ is reduced modulo *d*₂
- *d*₂ = *b*₂[*i*], with {*b*₀, *b*₁, ..., *b*ₙ} HNF basis of the ideal *self*.

---

4.2. Ideals of number fields 465
Note: The reduced element \( g \) is not necessarily small. To get a small \( g \) use the method \texttt{small_residue()}. 

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: k.<a> = NumberField(x^3 + 11)
sage: I = k.ideal(5, a^2 - a + 1)
sage: c = 4*a + 9
sage: I.reduce(c)
a^2 - 2*a
sage: c - I.reduce(c) in I
True
```

Sometimes the canonical reduced representative of 1 won’t be 1 (it depends on the choice of basis for the ring of integers):
AUTHOR: Maite Aranes.

residue_class_degree()
Return the residue class degree of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The residue class degree of a prime ideal $I$ is the degree of the extension $O_K/I$ of its prime subfield.

EXAMPLES:

```sage
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^5 + 2); K
Number Field in a with defining polynomial x^5 + 2
sage: f = K.factor(19); f
(Fractional ideal (a^2 + a - 3))
* (Fractional ideal (2*a^4 + a^2 - 2*a + 1))
* (Fractional ideal (a^2 + a - 1))
sage: [i.residue_class_degree() for i, _ in f]
[2, 2, 1]
```

residue_field(names=None)
Return the residue class field of this fractional ideal, which must be prime.

EXAMPLES:

```sage
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^3 - 7)
sage: P = K.ideal(29).factor()[0][0]
sage: P.residue_field()
```

(continues on next page)
Residue field in $\bar{a}$ of Fractional ideal $(2a^2 + 3a - 10)$

```python
sage: P.residue_field('z')
Residue field in $z$ of Fractional ideal $(2a^2 + 3a - 10)$
```

Another example:

```python
sage: K.<a> = NumberField(x^3 - 7)
sage: P = K.ideal(389).factor()[0][0]; P
Fractional ideal (389, $a^2 - 44a - 9$)
sage: P.residue_class_degree()
2
sage: P.residue_field()
Residue field in $\bar{a}$ of Fractional ideal (389, $a^2 - 44a - 9$)
sage: P.residue_field('z')
Residue field in $z$ of Fractional ideal (389, $a^2 - 44a - 9$)
sage: FF.<w> = P.residue_field()
sage: FF
Residue field in $w$ of Fractional ideal (389, $a^2 - 44a - 9$)
sage: FF((a+1)^390)
36
sage: FF(a)
$w$
```

An example of reduction maps to the residue field: these are defined on the whole valuation ring, i.e. the subring of the number field consisting of elements with non-negative valuation. This shows that the issue raised in Issue #1951 has been fixed:
Algebraic Numbers and Number Fields, Release 10.4

```
sage: K.<i> = NumberField(x^2 + 1)
sage: P1, P2 = [g[0] for g in K.factor(5)]; P1, P2
(Fractional ideal (-i - 2), Fractional ideal (2*i + 1))
sage: a = 1/(1+2*i)
sage: F1, F2 = [g.residue_field() for g in [P1, P2]]; F1, F2
(Residue field of Fractional ideal (-i - 2),
 Residue field of Fractional ideal (2*i + 1))
sage: a.valuation(P1)
0
sage: F1(i/7)
4
sage: F1(a)
3
sage: a.valuation(P2)
-1
sage: F2(a)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot reduce field element -2/5*i + 1/5
modulo Fractional ideal (2*i + 1): it has negative valuation
```

```
>>> from sage.all import *
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
˓→first_ngens(1)
>>> P1, P2 = [g[Integer(0)] for g in K.factor(Integer(5))]; P1, P2
(Fractional ideal (-i - 2), Fractional ideal (2*i + 1))
>>> a = Integer(1)/(Integer(1)+Integer(2)*i)
>>> F1, F2 = [g.residue_field() for g in [P1, P2]]; F1, F2
(Residue field of Fractional ideal (-i - 2),
 Residue field of Fractional ideal (2*i + 1))
>>> a.valuation(P1)
0
>>> F1(i/Integer(7))
4
>>> F1(a)
3
>>> a.valuation(P2)
-1
>>> F2(a)
Traceback (most recent call last):
...
ZeroDivisionError: Cannot reduce field element -2/5*i + 1/5
modulo Fractional ideal (2*i + 1): it has negative valuation
```

An example with a relative number field:

```
sage: L.<a,b> = NumberField([x^2 + 1, x^2 - 5])
sage: p = L.ideal((-1/2*b - 1/2)*a + 1/2*b - 1/2)
sage: R = p.residue_field(); R
Residue field in abar of Fractional ideal ((-1/2*b - 1/2)*a + 1/2*b - 1/2)
sage: R.cardinality()
9
sage: R(17)
2
sage: R((a + b)/17)
abar
sage: R(1/b)
```

(continues on next page)
We verify that Issue #8721 is fixed:

```python
from sage.all import *
L = NumberField([x**Integer(2) - Integer(3), x**Integer(2) - Integer(5)],
                names=('a', 'b',)); (a, b,) = L._first_ngens(2)
R = L.ideal(a).residue_field(); R
```

```
Residue field in abar of Fractional ideal (a)
```

```
>>> from sage.all import *
>>> L = NumberField([x**Integer(2) - Integer(3), x**Integer(2) - Integer(5)],
                  names=('a', 'b',)); (a, b,) = L._first_ngens(2)
>>> L.ideal(a).residue_field()
Residue field in abar of Fractional ideal (a)
```

residues()

Return an iterator through a complete list of residues modulo this integral ideal.

An error is raised if this fractional ideal is not integral.

OUTPUT:

An iterator through a complete list of residues modulo the integral ideal self. This list is the set of canonical reduced representatives given by all integral elements with coordinates \((r_0, \ldots, r_{n-1})\), where:

- \(r_i\) is reduced modulo \(d_i\)
- \(d_i = b_i[i]\), with \(\{b_0, b_1, \ldots, b_n\}\) HNF basis of the ideal.

AUTHOR: John Cremona (modified by Maite Aranes)

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: res = K.ideal(2).residues(); res
xmrange_iter(([0, 1], [0, 1]), <function ...<lambda> at 0x...>)
sage: list(res)
[0, i, 1, i + 1]
sage: list(K.ideal(2 + i).residues())
[-2*i, -i, 0, i, 2*i]
sage: list(K.ideal(i).residues())
[0]
```
sage: I = K.ideal(3 + 6*i)
sage: reps = I.residues()
sage: len(list(reps)) == I.norm()
True
sage: all(r == s or not (r-s) in I)  # long time (6s on sage.math, 2011)
....: for r in reps for s in reps)
True

sage: K.<a> = NumberField(x^3 - 10)
sage: I = K.ideal(a - 1)
sage: len(list(I.residues())) == I.norm()
True

sage: K.<z> = CyclotomicField(11)
sage: len(list(K.primes_above(3)[0].residues())) == 3**5  # long time (5s on...
˓→sage.math, 2011)
True

small_residue(f)

Given an element $f$ of the ambient number field, return an element $g$ such that $f - g$ belongs to the ideal self (which must be integral), and $g$ is small.

Note: The reduced representative returned is not uniquely determined.
ALGORITHM: Uses PARI function `pari:nfeltreduce`.

EXAMPLES:

```sage
tax = polygen(ZZ)
t.<a> = NumberField(x^2 + 5)
t.I = t.ideal(a)
t.I.small_residue(14)
4
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> k = NumberField(x**Integer(2) + Integer(5), names=('a',)); (a,) = k._
˓→first_ngens(1)
>>> I = k.ideal(a)
>>> I.small_residue(Integer(14))
4
```

```sage
K.<a> = NumberField(x^5 + 7*x^4 + 18*x^2 + x - 3)
I = K.ideal(5)
I.small_residue(a^2 - Integer(13))
a^2 + 5*a - 3
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(5) + Integer(7)*x**Integer(4) + Integer(18)*x**Integer(2) + x - Integer(3), names=('a',)); (a,) = K._first_˓→ngens(1)
>>> I = K.ideal(Integer(5))
>>> I.small_residue(a**Integer(2) - Integer(13))
a^2 + 5*a - 3
```

`support()`

Return a list of the prime ideal factors of `self`.

OUTPUT:

list of prime ideals (a new list is returned each time this function is called)

EXAMPLES:

```sage
x = polygen(ZZ)
t.<w> = NumberField(x^2 + 23)
t.I = ideal(w+1)
t.I.prime_factors()
[Fractional ideal (2, 1/2*w - 1/2),
 Fractional ideal (2, 1/2*w + 1/2),
 Fractional ideal (3, 1/2*w + 1/2)]
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(23), names=('w',)); (w,) = K._
˓→first_ngens(1)
>>> I = ideal(w+Integer(1))
>>> I.prime_factors()
[Fractional ideal (2, 1/2*w - 1/2),
 Fractional ideal (2, 1/2*w + 1/2),
 Fractional ideal (3, 1/2*w + 1/2)]
```
class sage.rings.number_field.number_field_ideal.NumberFieldIdeal (field, gens, coerce=True)

Bases:Ideal_generic

An ideal of a number field.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(7)
Fractional ideal (7)
```

Initialization from PARI:

```python
sage: K.ideal(pari(7))
Fractional ideal (7)
sage: K.ideal(pari(4), pari(4 + 2*i))
Fractional ideal (2)
sage: K.ideal(pari("i + 2"))
Fractional ideal (i + 2)
sage: K.ideal(pari("[3,0;0,3]"))
Fractional ideal (3)
sage: F = pari(K).idealprimedec(5)
sage: K.ideal(F[0])
Fractional ideal (2*i + 1)
```

S_ideal_class_log(S)

S-class group version of ideal_class_log().

EXAMPLES:

```python
sage: K.<a> = QuadraticField(-14)
sage: S = K.primes_above(2)
sage: I = K.ideal(3, a + 1)
sage: I.S_ideal_class_log(S)[1]
```

(continues on next page)
sage: I.S_ideal_class_log([])
[3]

>>> from sage.all import *

>>> K = QuadraticField(-Integer(14), names=('a',)); (a,) = K._first_ngens(1)
>>> S = K.primes_above(Integer(2))
>>> I = K.ideal(Integer(3), a + Integer(1))

>>> I.S_ideal_class_log(S)
[1]

>>> I.S_ideal_class_log([])
[3]

**absolute_norm()**

A synonym for norm().

**EXAMPLES:**

```sage
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + 2*i).absolute_norm()
5
```

```sage
>>> from sage.all import *

>>> x = polygen(ZZ)

>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
˓→first_ngens(1)

>>> K.ideal(Integer(1) + Integer(2)*i).absolute_norm()
5
```

**absolute_ramification_index()**

A synonym for ramification_index().

**EXAMPLES:**

```sage
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + i).absolute_ramification_index()
2
```

```sage
>>> from sage.all import *

>>> x = polygen(ZZ)

>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
˓→first_ngens(1)

>>> K.ideal(Integer(1) + i).absolute_ramification_index()
2
```

**artin_symbol()**

Return the Artin symbol \((K/Q, P)\), where \(K\) is the number field of \(P = \text{self}\). This is the unique element \(s\) of the decomposition group of \(P\) such that \(s(x) = x^p \pmod{P}\) where \(p\) is the residue characteristic of \(P\). (Here \(P\) should be prime and unramified.)

See the GaloisGroup_v2.artin_symbol() method for further documentation and examples.

**EXAMPLES:**
basis()

Return a basis for this ideal viewed as a \( \mathbb{Z} \)-module.

OUTPUT:

An immutable sequence of elements of this ideal (note: their parent is the number field) forming a basis for this ideal.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(7)
sage: I = K.factor(11)[0][0]
sage: I.basis()  # warning -- choice of basis can be somewhat random
[11, 11*z, 11*z^2, z^3 + 5*z^2 + 4*z + 10, z^4 + z^2 + z + 5, z^5 + z^4 + z^3 + 2*z^2 + 6*z + 5]
```

An example of a non-integral ideal:

```
sage: J = 1/I
sage: J  # warning -- choice of generators can be somewhat random
Fractional ideal (2/11*z^5 + 2/11*z^4 + 3/11*z^3 + 2/11)
sage: J.basis()  # warning -- choice of basis can be somewhat random
[1, z, z^2, 1/11*z^3 + 7/11*z^2 + 6/11*z + 10/11, 1/11*z^4 + 1/11*z^2 + 1/11*z + 7/11, 1/11*z^5 + 1/11*z^4 + 1/11*z^3 + 2/11*z^2 + 8/11*z + 7/11]
```

Number fields defined by non-monic and non-integral polynomials are supported (Issue #252):

```
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(2*x^2 - 1/3)
sage: K.ideal(a).basis()
[1, a]
```
coordinates \(x\)

Return the coordinate vector of \(x\) with respect to this ideal.

**INPUT:**

- \(x\) – an element of the number field (or ring of integers) of this ideal.

**OUTPUT:**

List giving the coordinates of \(x\) with respect to the integral basis of the ideal. In general this will be a vector of rationals; it will consist of integers if and only if \(x\) is in the ideal.

**AUTHOR:** John Cremona 2008-10-31

**ALGORITHM:**

Uses linear algebra. Provides simpler implementations for \_contains\_, \is_integral\() and \smallest_integer\().

**EXAMPLES:**

```python
case: K.<i> = QuadraticField(-1)
case: I = K.ideal(7 + 3*i)
case: Ibasis = I.integral_basis(); Ibasis [58, i + 41]
case: a = 23 - 14*i
case: acoords = I.coordinates(a); acoords (597/58, -14)
case: sum([Ibasis[j]*acoords[j] for j in range(2)]) == a True
case: b = 123 + 456*i
case: bcoords = I.coordinates(b); bcoords (-18573/58, 456)
case: sum([Ibasis[j]*bcoords[j] for j in range(2)]) == b True
case: J = K.ideal(0)
case: J.coordinates(0) ()
case: J.coordinates(1) Traceback (most recent call last):
... TypeError: vector is not in free module
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> I = K.ideal(Integer(7) + Integer(3)*i)
>>> Ibasis = I.integral_basis(); Ibasis [58, i + 41]
>>> a = Integer(23) - Integer(14)*i
>>> acoords = I.coordinates(a); acoords (597/58, -14)
>>> sum([Ibasis[j]*acoords[j] for j in range(Integer(2))]) == a True
```
```
True
>>> b = Integer(123) + Integer(456)*i
>>> bcoords = I.coordinates(b); bcoords
(-18573/58, 456)
>>> sum([Ibasis[j]*bcoords[j] for j in range(Integer(2))]) == b
True
>>> J = K.ideal(Integer(0))
>>> J.coordinates(Integer(0))
()
>>> J.coordinates(Integer(1))
Traceback (most recent call last):
...TypeError: vector is not in free module
```

decomposition_group()

Return the decomposition group of self, as a subset of the automorphism group of the number field of self. Raises an error if the field isn’t Galois. See the GaloisGroup_v2.decomposition_group() method for further examples and doctests.

EXAMPLES:

```
sage: QuadraticField(-23, 'w').primes_above(7)[0].decomposition_group()  # needs sage.groups
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
```

```
>>> from sage.all import *
>>> QuadraticField(-Integer(23), 'w').primes_above(Integer(7))[Integer(0)].decomposition_group()  # needs sage.groups
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
```

free_module()

Return the free \( \mathbb{Z} \)-module contained in the vector space associated to the ambient number field, that corresponds to this ideal.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(7)
sage: I = K.factor(11)[0][0]; I
Fractional ideal (-3*z^4 - 2*z^3 - 2*z^2 - 2)
sage: A = I.free_module()
sage: A  # warning -- choice of basis can be somewhat random
Free module of degree 6 and rank 6 over Integer Ring
User basis matrix:
[11 0 0 0 0 0]
[ 0 11 0 0 0 0]
[ 0 0 11 0 0 0]
[10 4 5 1 0 0]
[ 5 1 1 0 1 0]
[ 5 6 2 1 1 1]
```

```
>>> from sage.all import *
>>> K = CyclotomicField(Integer(7), names=('z',)); (z,) = K._first_ngens(1)
>>> I = K.factor(Integer(11))[Integer(0)][Integer(0)]; I
Fractional ideal (-3*z^4 - 2*z^3 - 2*z^2 - 2)
```
A = I.free_module()

Free module of degree 6 and rank 6 over Integer Ring

User basis matrix:
\[
\begin{bmatrix}
11 & 0 & 0 & 0 & 0 & 0 \\
11 & 0 & 0 & 0 & 0 & 0 \\
10 & 4 & 1 & 0 & 0 & 0 \\
5 & 1 & 1 & 0 & 1 & 0 \\
5 & 6 & 2 & 1 & 1 & 1 \\
\end{bmatrix}
\]


However, the actual \(\mathbb{Z}\)-module is not at all random:

```
A.basis_matrix().change_ring(ZZ).echelon_form()

\[
\begin{bmatrix}
1 & 0 & 0 & 5 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 7 \\
0 & 0 & 1 & 7 & 6 & 10 \\
0 & 0 & 0 & 11 & 0 & 0 \\
0 & 0 & 0 & 0 & 11 & 0 \\
0 & 0 & 0 & 0 & 0 & 11 \\
\end{bmatrix}
\]
```

```
from sage.all import *

A.basis_matrix().change_ring(ZZ).echelon_form()

\[
\begin{bmatrix}
1 & 0 & 0 & 5 & 1 & 1 \\
0 & 1 & 0 & 1 & 1 & 7 \\
0 & 0 & 1 & 7 & 6 & 10 \\
0 & 0 & 0 & 11 & 0 & 0 \\
0 & 0 & 0 & 0 & 11 & 0 \\
0 & 0 & 0 & 0 & 0 & 11 \\
\end{bmatrix}
\]
```

The ideal doesn’t have to be integral:

```
\text{sage:} \quad J = I^{(-1)}
\text{sage:} \quad B = J.free_module()
\text{sage:} \quad B.echelonized_basis_matrix()

\[
\begin{bmatrix}
1/11 & 0 & 0 & 7/11 & 1/11 & 1/11 \\
0 & 1/11 & 0 & 1/11 & 1/11 & 5/11 \\
0 & 0 & 1/11 & 5/11 & 4/11 & 10/11 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
```

```
from sage.all import *

\text{sage:} \quad J = I^{(-1)}
\text{sage:} \quad B = J.free_module()
\text{sage:} \quad B.echelonized_basis_matrix()

\[
\begin{bmatrix}
1/11 & 0 & 0 & 7/11 & 1/11 & 1/11 \\
0 & 1/11 & 0 & 1/11 & 1/11 & 5/11 \\
0 & 0 & 1/11 & 5/11 & 4/11 & 10/11 \\
0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 \\
\end{bmatrix}
\]
```

This also works for relative extensions:
sage: x = polygen(ZZ)
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 2])
sage: I = K.fractional_ideal(4)
sage: I.free_module()
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
[  4  0  0  0]
[ -3  7 -1  1]
[  3  7  1  1]
[  0 -10  0 -2]
sage: J = I^(-1); J.free_module()
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
[ 1/4  0  0  0]
[ -3/16  7/16 -1/16  1/16]
[ 3/16  7/16  1/16  1/16]
[  0 -5/8  0 -1/8]

An example of intersecting ideals by intersecting free modules:

sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: I = K.factor(2)
sage: p1 = I[0][0]; p2 = I[1][0]
sage: N = p1.free_module().intersection(p2.free_module()); N
Free module of degree 3 and rank 3 over Integer Ring
Echelon basis matrix:
[ 1  1/2  1/2]
[ 0  1  1]
[ 0  0  2]
sage: N.index_in(p1.free_module()).abs()
2

An example of intersecting ideals by intersecting free modules: (continues on next page)
 gens_reduced (proof=None)
Express this ideal in terms of at most two generators, and one if possible.
This function indirectly uses pari:bnfisprincipal, so set proof=True if you want to prove correctness (which is the default).
EXAMPLES:

```
sage: x = polygen(ZZ)
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 5)
sage: K.ideal(0).gens_reduced()
(0,)
sage: J = K.ideal([a + 2, 9])
sage: J.gens()
(a + 2, 9)
sage: J.gens_reduced()  # random sign
(a + 2,)
sage: K.ideal([a + 2, 3]).gens_reduced()
(3, a + 2)
```

 gens_two ()
Express this ideal using exactly two generators, the first of which is a generator for the intersection of the ideal with \( \mathbb{Q} \).
ALGORITHM: uses PARI’s pari:idealtowoelt function, which runs in randomized polynomial time and is very fast in practice.
EXAMPLES:

```
sage: x = polygen(ZZ)
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 5)
```
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> R = PolynomialRing(QQ, names=('x',)); (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(2) + Integer(5), names=('a',)); (a,) = K._
˓→first_ngens(1)
>>> J = K.ideal([a + Integer(2), Integer(9)])
>>> J.gens()
(a + 2, 9)
>>> J.gens_two()
(9, a + 2)
>>> K.ideal([a + Integer(5), a + Integer(8)]).gens_two()
(3, a + 2)
>>> K.ideal(Integer(0)).gens_two()
(0, 0)

The second generator is zero if and only if the ideal is generated by a rational, in contrast to the PARI function pari:idealtwoelt:

sage: I = K.ideal(12)
sage: pari(K).idealtwoelt(I)  # Note that second element is not zero
[12, [0, 12]-]

ideal_class_log (proof=None)

Return the output of PARI’s pari:bnfisprincipal for this ideal, i.e. a vector expressing the class of this ideal in terms of a set of generators for the class group.

Since it uses the PARI method pari:bnfisprincipal, specify proof=True (this is the default setting) to prove the correctness of the output.

EXAMPLES:

When the class number is 1, the result is always the empty list:
An example with class group of order 2. The first ideal is not principal, the second one is:

```python
sage: K.<a> = QuadraticField(-5)
sage: J = K.ideal(23).factor()[0][0]
sage: J.ideal_class_log()
[1]
sage: (J**10).ideal_class_log()
[0]
```

An example with a more complicated class group:

```python
sage: x = polygen(ZZ)
sage: K.<a, b> = NumberField([x^3 - x + 1, x^2 + 26])
sage: K.class_group()
Class group of order 18 with structure C6 x C3 of
Number Field in a with defining polynomial x^3 - x + 1 over its base field
sage: K.primes_above(7)[0].ideal_class_log() # random
[1, 2]
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField([x**Integer(3) - x + Integer(1), x**Integer(2) + Integer(26)], names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> K.class_group()
Class group of order 18 with structure C6 x C3 of
Number Field in a with defining polynomial x^3 - x + 1 over its base field
>>> K.primes_above(Integer(7))[Integer(0)].ideal_class_log() # random
[1, 2]
```

`inertia_group()`

Return the inertia group of `self`, i.e. the set of elements \( s \) of the Galois group of the number field of `self` (which we assume is Galois) such that \( s \) acts trivially modulo `self`. This is the same as the 0th ramification group of `self`. See the `GaloisGroup_v2.inertia_group()` method for further examples and doctests.

EXAMPLES:

```python
sage: QuadraticField(-23, 'w').primes_above(23)[0].inertia_group()  # needs sage.groups
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
```
integral_basis()

Return a list of generators for this ideal as a \( \mathbb{Z} \)-module.

**EXAMPLES:**

```python
sage: x = polygen(ZZ)
sage: R.<x> = PolynomialRing(QQ)
sage: K.<i> = NumberField(x^2 + 1)
sage: J = K.ideal(i + 1)
sage: J.integral_basis()
[2, i + 1]
```

integral_split()

Return a tuple \((I, d)\), where \(I\) is an integral ideal, and \(d\) is the smallest positive integer such that this ideal is equal to \(I/d\).

**EXAMPLES:**

```python
sage: x = polygen(ZZ)
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 - 5)
sage: I = K.ideal(2/(5+a))
sage: I.is_integral()
False
sage: J, d = I.integral_split()
sage: J
Fractional ideal (-1/2*a + 5/2)
sage: J.is_integral()
True
sage: d
5
sage: I == J/d
True
```
False

```python
>>> J, d = I.integral_split()
>>> J
Fractional ideal (-1/2*a + 5/2)
>>> J.is_integral()
True
>>> d
5
>>> I == J/d
True
```

**intersection** *(other)*

Return the intersection of self and other.

**EXAMPLES:**

```python
sage: K.<a> = QuadraticField(-11)
sage: p = K.ideal((a + 1)/2); q = K.ideal((a + 3)/2)
sage: p.intersection(q) == q.intersection(p) == K.ideal(a - 2)
True
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(11), names=('a',)); (a,) = K._first_ngens(1)
>>> p = K.ideal((a + Integer(1))/Integer(2)); q = K.ideal((a + Integer(3))/Integer(2))
>>> p.intersection(q) == q.intersection(p) == K.ideal(a - Integer(2))
True
```

An example with non-principal ideals:

```python
sage: x = polygen(ZZ)
sage: L.<a,b> = NumberField([x^2 + 11, x^2 - 5])
sage: A = L.ideal([15, (-3/2*b + 7/2)*a - 8])
sage: B = L.ideal([6, (-1/2*b + 1)*a - b - 5/2])
sage: A.intersection(B) == L.ideal(-1/2*a - 3/2*b - 1)
True
```

```python
>>> from sage.all import *
>>> L = NumberField([x**Integer(2) + Integer(11), x**Integer(2) - Integer(5)],
(continues on next page)
is_integral()  
Return True if this ideal is integral.  

EXAMPLES:

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^5 - x + 1)
sage: K.ideal(a).is_integral()
True
sage: (K.ideal(1) / (3*a+1)).is_integral()
False
```

is_maximal()  
Return True if this ideal is maximal. This is equivalent to self being prime and nonzero.  

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^3 + 3); K
Number Field in a with defining polynomial x^3 + 3
sage: K.ideal(5).is_maximal()
False
sage: K.ideal(7).is_maximal()
True
```

is_prime()  
Return True if this ideal is prime.  

EXAMPLES:

```python
sage: K.<a> = NumberField(x^3 + 3); K
Number Field in a with defining polynomial x^3 + 3
sage: K.ideal(5).is_prime()
False
sage: K.ideal(7).is_prime()
True
```
is_principal \( (proof=None) \)

Return True if this ideal is principal.

Since it uses the PARI method pari:bnfisprincipal, specify \( proof=True \) (this is the default setting) to prove the correctness of the output.

EXAMPLES:

```python
sage: K = QuadraticField(-119, 'a')
sage: P = K.factor(2)[1][0]
sage: P.is_principal()
False
sage: I = P^5
sage: I.is_principal()
True
```
is_zero()  
Return True if self is the zero ideal  
Note that (0) is a NumberFieldIdeal, not a NumberFieldFractionalIdeal.

Examples:

```python
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^2 + 2); K
Number Field in a with defining polynomial x^2 + 2
sage: K.ideal(3).is_zero()
False
sage: I = K.ideal(0); I.is_zero()
True
sage: I
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
```

>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(2), names=('a',)); (a,) = K._first_ngens(1); K
Number Field in a with defining polynomial x^2 + 2
>>> K.ideal(Integer(3)).is_zero()
False
>>> I = K.ideal(Integer(0)); I.is_zero()
True
>>> I
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
```

norm()  
Return the norm of this fractional ideal as a rational number.

Examples:

```python
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^4 + 23); K
Number Field in a with defining polynomial x^4 + 23
sage: I = K.ideal(19); I
Fractional ideal (19)
sage: factor(I.norm())
19^4
sage: F = I.factor()  
sage: F[0][0].norm().factor()
19^2
```

>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(4) + Integer(23), names=('a',)); (a,) = K._first_ngens(1); K
Number Field in a with defining polynomial x^4 + 23
>>> I = K.ideal(Integer(19)); I
Fractional ideal (19)
>>> factor(I.norm())
19^4
```

(continues on next page)
number_field()

Return the number field that this is a fractional ideal in.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^2 + 2); K
Number Field in a with defining polynomial x^2 + 2
sage: K.ideal(3).number_field()
Number Field in a with defining polynomial x^2 + 2
sage: K.ideal(0).number_field()  # not tested (not implemented)
Number Field in a with defining polynomial x^2 + 2
```

pari_hnf()

Return PARI’s representation of this ideal in Hermite normal form.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^3 - 2)
sage: I = K.ideal(2/(5+a))
sage: I.pari_hnf()
[2, 0, 50/127; 0, 2, 244/127; 0, 0, 2/127]
```

pari_prime()

Return a PARI prime ideal corresponding to the ideal self.

INPUT:

• `self` — a prime ideal.

OUTPUT: a PARI “prime ideal”, i.e. a five-component vector `[p, a, e, f, b]` representing the prime ideal `pO_K + aO_K`, `e`, `f` as usual, `a` as vector of components on the integral basis, `b` Lenstra’s constant.
EXAMPLES:

```python
sage: K.<i> = QuadraticField(-1)
sage: K.ideal(3).pari_prime()
[3, [3, 0], 1, 2, 1]
sage: K.ideal(2+i).pari_prime()
[5, [2, 1], 1, 1, [-2, -1; 1, -2]]
sage: K.ideal(2).pari_prime()
Traceback (most recent call last):
  ... ValueError: Fractional ideal (2) is not a prime ideal
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(1), names=('i',)); (i,) = K._first_ngens(1)
>>> K.ideal(Integer(3)).pari_prime()
[3, [3, 0], 1, 2, 1]
>>> K.ideal(Integer(2)+i).pari_prime()
[5, [2, 1], 1, 1, [-2, -1; 1, -2]]
>>> K.ideal(Integer(2)).pari_prime()
Traceback (most recent call last):
  ... ValueError: Fractional ideal (2) is not a prime ideal
```

`ramification_group(v)`

Return the $v$'th ramification group of `self`, i.e. the set of elements $s$ of the Galois group of the number field of `self` (which we assume is Galois) such that $s$ acts trivially modulo the $(v + 1)$'st power of self. See the `GaloisGroup.ramification_group()` method for further examples and doctests.

EXAMPLES:

```python
sage: QuadraticField(-23, 'w').primes_above(23)[0].ramification_group(0)  # needs sage.groups
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
sage: QuadraticField(-23, 'w').primes_above(23)[0].ramification_group(1)  # needs sage.groups
Subgroup generated by [(())] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
```

```python
>>> from sage.all import *
>>> QuadraticField(-Integer(23), 'w').primes_above(Integer(23))[0].ramification_group(Integer(0)).  # needs sage.groups
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
>>> QuadraticField(-Integer(23), 'w').primes_above(Integer(23))[0].ramification_group(Integer(1)).  # needs sage.groups
Subgroup generated by [(())] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
```

`random_element(*args, **kwds)`

Return a random element of this ideal.

INPUT:

* *args, **kwds – Parameters passed to the random integer function. See the documentation of `ZZ.random_element()` for details.

OUTPUT:

A random element of this fractional ideal, computed as a random $\mathbb{Z}$-linear combination of the basis.

4.2. Ideals of number fields
EXAMPLES:

```
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^3 + 2)
sage: I = K.ideal(1 - a)
sage: I.random_element() # random output
-a^2 - a - 19
sage: I.random_element(distribution="uniform") # random output
a^2 - 2*a - 8
sage: I.random_element(-30, 30) # random output
7*a^2 - 17*a - 75
sage: I.random_element(-100, 200).is_integral()
True
sage: I.random_element(-30, 30).parent() is K
True
```
ALGORITHM: Calls `pari:idealred` function.

EXAMPLES:

```sage
sage: x = polygen(ZZ)
sage: K.<w> = NumberField(x^2 + 23)
sage: I = ideal(w*23^5); I
Fractional ideal (6436343*w)
sage: I.reduce_equiv()
Fractional ideal (1)
sage: I = K.class_group().0.ideal()^10; I
Fractional ideal (1024, 1/2*w + 979/2)
sage: I.reduce_equiv()
Fractional ideal (2, 1/2*w - 1/2)
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(23), names=('w',)); (w,) = K._
˓→first_ngens(1)
>>> I = ideal(w*Integer(23)**Integer(5)); I
Fractional ideal (6436343*w)
>>> I.reduce_equiv()
Fractional ideal (1)
>>> I = K.class_group().gen(0).ideal()**Integer(10); I
Fractional ideal (1024, 1/2*w + 979/2)
>>> I.reduce_equiv()
Fractional ideal (2, 1/2*w - 1/2)
```

`relative_norm()`
A synonym for `norm()`.

EXAMPLES:

```sage
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + 2*i).relative_norm()
5
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._
˓→first_ngens(1)
>>> K.ideal(Integer(1) + Integer(2)*i).relative_norm()
5
```

`relative_ramification_index()`
A synonym for `ramification_index()`.

EXAMPLES:

```sage
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + i).relative_ramification_index()
2
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
```

(continues on next page)
residue_symbol \( (e, m, \text{check}=\text{True}) \)

The \( m \)-th power residue symbol for an element \( e \) and the proper ideal.

\[
\left( \frac{\alpha}{\mathfrak{p}} \right) \equiv \alpha^{\frac{\chi(\mathfrak{p}) - 1}{m}} \mod \mathfrak{p}
\]

**Note:** accepts \( m = 1 \), in which case returns 1

**Note:** can also be called for an element from sage.rings.number_field_element.residue_symbol

**Note:** \( e \) is coerced into the number field of \( \text{self} \)

**Note:** if \( m = 2 \), \( e \) is an integer, and \( \text{self}.\text{number_field}() \) has absolute degree 1 (i.e. it is a copy of the rationals), then this calls \( \text{kronecker}\_\text{symbol}() \), which is implemented using GMP.

**INPUT:**
- \( e \) – element of the number field
- \( m \) – positive integer

**OUTPUT:**
- an \( m \)-th root of unity in the number field

**EXAMPLES:**

Quadratic Residue (7 is not a square modulo 11):

```sage
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x - 1)
sage: K.ideal(11).residue_symbol(7,2)
-1
```

Cubic Residue:

```sage
sage: K.<w> = NumberField(x^2 - x + 1)
sage: K.ideal(17).residue_symbol(w^2 + 3, 3)
-w
```
The field must contain the $m$-th roots of unity:

```python
sage: K.<w> = NumberField(x^2 - x + 1)
sage: K.ideal(17).residue_symbol(w^2 + 3, 5)
Traceback (most recent call last):
  ...
ValueError: The residue symbol to that power is not defined for the number field
```

#### smallest_integer()

Return the smallest non-negative integer in $I \cap \mathbb{Z}$, where $I$ is this ideal. If $I = 0$, returns $0$.

**EXAMPLES:**

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 6)
sage: I = K.ideal([4, a])/7; I
Fractional ideal (2/7, 1/7*a)
sage: I.smallest_integer()
2
```

#### valuation($p$)

Return the valuation of self at $p$.

**INPUT:**

- $p$ – a prime ideal $p$ of this number field.

**OUTPUT:**

(integer) The valuation of this fractional ideal at the prime $p$. If $p$ is not prime, raise a `ValueError`.

**EXAMPLES:**
```
sage: x = polygen(ZZ)
sage: K.<a> = NumberField(x^5 + 2); K
Number Field in a with defining polynomial x^5 + 2
sage: i = K.ideal(38); i
Fractional ideal (38)
sage: i.valuation(K.factor(19)[0][0])
1
sage: i.valuation(K.factor(2)[0][0])
5
sage: i.valuation(K.factor(3)[0][0])
0
sage: i.valuation(0)
Traceback (most recent call last):
  ... ValueError: p (= Ideal (0) of Number Field in a
  with defining polynomial x^5 + 2) must be nonzero
sage: K.ideal(0).valuation(K.factor(2)[0][0])
+Infinity
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(5) + Integer(2), names=('a',)); (a,) = K._
˓→first_ngens(1); K
Number Field in a with defining polynomial x^5 + 2
>>> i = K.ideal(Integer(38)); i
Fractional ideal (38)
>>> i.valuation(K.factor(Integer(19))[Integer(0)][Integer(0)])
1
>>> i.valuation(K.factor(Integer(2))[Integer(0)][Integer(0)])
5
>>> i.valuation(K.factor(Integer(3))[Integer(0)][Integer(0)])
0
>>> i.valuation(Integer(0))
Traceback (most recent call last):
  ... ValueError: p (= Ideal (0) of Number Field in a
  with defining polynomial x^5 + 2) must be nonzero
>>> K.ideal(Integer(0)).valuation(K.
˓→factor(Integer(2))[Integer(0)][Integer(0)])
+Infinity
```

class sage.rings.number_field.number_field_ideal.QuotientMap(K, M_OK_change, Q, I)

Bases: object

Class to hold data needed by quotient maps from number field orders to residue fields. These are only partial maps: the exact domain is the appropriate valuation ring. For examples, see `residue_field()`.

sage.rings.number_field.number_field_ideal.basis_to_module(B, K)

Given a basis $B$ of elements for a $\mathbb{Z}$-submodule of a number field $K$, return the corresponding $\mathbb{Z}$-submodule.

EXAMPLES:

```
sage: x = polygen(ZZ)
sage: K.<w> = NumberField(x^4 + 1)
sage: from sage.rings.number_field.number_field_ideal import basis_to_module
sage: basis_to_module([K.0, K.0^2 + 3], K)
Free module of degree 4 and rank 2 over Integer Ring
```
User basis matrix:
\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
3 & 0 & 1 & 0 \\
\end{bmatrix}
\]

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(4) + Integer(1), names=('w',)); (w,) = K._first_
˓→ngens(1)
>>> from sage.rings.number_field.number_field_ideal import basis_to_module
>>> basis_to_module([K.gen(0), K.gen(0)**Integer(2) + Integer(3)], K)
Free module of degree 4 and rank 2 over Integer Ring
User basis matrix:
\[
\begin{bmatrix}
0 & 1 & 0 & 0 \\
3 & 0 & 1 & 0 \\
\end{bmatrix}
\]
sage.rings.number_field.number_field_ideal.is_NumberFieldFractionalIdeal(x)
Return True if \(x\) is a fractional ideal of a number field.

EXAMPLES:

```python
sage: from sage.rings.number_field.number_field_ideal import is_NumberFieldFractionalIdeal
sage: is_NumberFieldFractionalIdeal(2/3)
False
sage: is_NumberFieldFractionalIdeal(ideal(5))
False
sage: x = polygen(ZZ)
sage: k.<a> = NumberField(x^2 + 2)
\[
\text{Fractional ideal (a + 1)}
\]
True
sage: Z = k.ideal(0); Z
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
sage: is_NumberFieldFractionalIdeal(Z)
False
```
```python
>>> from sage.all import *
>>> from sage.rings.number_field.number_field_ideal import is_NumberFieldFractionalIdeal
>>> is_NumberFieldFractionalIdeal(Integer(2)/Integer(3))
False
>>> is_NumberFieldFractionalIdeal(ideal(Integer(5)))
False
>>> x = polygen(ZZ)
>>> k = NumberField(x**Integer(2) + Integer(2), names=('a',)); (a,) = k._first_
˓→ngens(1)
>>> I = k.ideal([a + Integer(1)]); I
```

(continues on next page)
Fractional ideal (a + 1)
>>> is_NumberFieldFractionalIdeal(I)
True
>>> Z = k.ideal(Integer(0)); Z
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
>>> is_NumberFieldFractionalIdeal(Z)
False

sage.rings.number_field.number_field_ideal.is_NumberFieldIdeal(x)

Return True if x is an ideal of a number field.

EXAMPLES:

sage: from sage.rings.number_field.number_field_ideal import is_NumberFieldIdeal
sage: is_NumberFieldIdeal(2/3)
False
sage: is_NumberFieldIdeal(ideal(5))
False
sage: x = polygen(ZZ)

sage: k.<a> = NumberField(x^2 + 2)

sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
>>> is_NumberFieldIdeal(I)
True
>>> Z = k.ideal(0); Z
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
>>> is_NumberFieldIdeal(Z)
True

>>> from sage.all import *

>>> from sage.rings.number_field.number_field_ideal import is_NumberFieldIdeal

>>> is_NumberFieldIdeal(Integer(2)/Integer(3))
False

>>> x = polygen(ZZ)

>>> k = NumberField(x**Integer(2) + Integer(2), names=('a',)); (a,) = k._first_ngens(1)

>>> I = k.ideal([a + Integer(1)]); I
Fractional ideal (a + 1)
>>> is_NumberFieldIdeal(I)
True
>>> Z = k.ideal(Integer(0)); Z
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
>>> is_NumberFieldIdeal(Z)
True
Given an integral ideal $I$ that contains a prime number $p$, compute a vector space $V = (O_K \mod p)/(I \mod p)$, along with a homomorphism $O_K \to V$ and a section $V \to O_K$.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.number_field_ideal import quotient_char_p
sage: x = polygen(ZZ)
sage: K.<i> = NumberField(x^2 + 1); O = K.maximal_order(); I = K.fractional_ideal(15)
sage: quotient_char_p(I, 5)[0]
Vector space quotient V/W of dimension 2 over Finite Field of size 5 where
V: Vector space of dimension 2 over Finite Field of size 5
W: Vector space of degree 2 and dimension 0 over Finite Field of size 5
Basis matrix:
[]

sage: quotient_char_p(I, 3)[0]
Vector space quotient V/W of dimension 2 over Finite Field of size 3 where
V: Vector space of dimension 2 over Finite Field of size 3
W: Vector space of degree 2 and dimension 0 over Finite Field of size 3
Basis matrix:
[]

sage: I = K.factor(13)[0][0]; I
Fractional ideal (-2*i + 3)
sage: I.residue_class_degree()
1
sage: quotient_char_p(I, 13)[0]
Vector space quotient V/W of dimension 1 over Finite Field of size 13 where
V: Vector space of dimension 2 over Finite Field of size 13
W: Vector space of degree 2 and dimension 1 over Finite Field of size 13
Basis matrix:
[1 8]
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.number_field_ideal import quotient_char_p
>>> x = polygen(ZZ)
>>> K = NumberField(x**Integer(2) + Integer(1), names=('i',)); (i,) = K._first_ngens(1); O = K.maximal_order(); I = K.fractional_ideal(Integer(15))
>>> quotient_char_p(I, Integer(5))[Integer(0)]
Vector space quotient V/W of dimension 2 over Finite Field of size 5 where
V: Vector space of dimension 2 over Finite Field of size 5
W: Vector space of degree 2 and dimension 0 over Finite Field of size 5
Basis matrix:
[]

>>> quotient_char_p(I, Integer(3))[Integer(0)]
Vector space quotient V/W of dimension 2 over Finite Field of size 3 where
V: Vector space of dimension 2 over Finite Field of size 3
W: Vector space of degree 2 and dimension 0 over Finite Field of size 3
Basis matrix:
[]

>>> I = K.factor(Integer(13))[Integer(0)][Integer(0)]; I
Fractional ideal (-2*i + 3)
>>> I.residue_class_degree()
1
```

(continues on next page)
4.3 Ideals of relative number fields

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 2])
sage: A = K.absolute_field('z')
sage: I = A.factor(7)[0][0]
sage: from_A, to_A = A.structure()
sage: G = [from_A(z) for z in I.gens()]; G
[7, -2*b*a - 1]
sage: K.fractional_ideal(G)
Fractional ideal ((1/2*b + 2)*a - 1/2*b + 2)
sage: K.fractional_ideal(G).absolute_norm().factor()
7^2
```

AUTHORS:

- Steven Sivek (2005-05-16)
- William Stein (2007-09-06)
- Nick Alexander (2009-01)
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField([x^2 + 1, x^2 + 2]); K
Number Field in a0 with defining polynomial x^2 + 1 over its base field
sage: i = K.ideal(38); i
Fractional ideal (38)

sage: K.<a0, a1> = NumberField([x^2 + 1, x^2 + 2]); K
Number Field in a0 with defining polynomial x^2 + 1 over its base field
sage: i = K.ideal([a0+1]); i
# random
Fractional ideal (-a1*a0)

sage: (g, ) = i.gens_reduced(); g
# random
-a1*a0

sage: ((a0 + 1) / g).is_integral()
True

absolute_ideal(names='a')

If this is an ideal in the extension \( L/K \), return the ideal with the same generators in the absolute field \( L/Q \).

INPUT:

- names (optional) – string; name of generator of the absolute field

EXAMPLES:

```python
sage: x = ZZ['x'].0
sage: K.<b> = NumberField(x^2 - 2)
sage: L.<c> = K.extension(x^2 - b)
sage: F.<m> = L.absolute_field()
```

```python
>>> from sage.all import *
>>> x = ZZ['x'].gen()
>>> K = NumberField(x**Integer(2) - Integer(2), names=('b',)); (b,) = K._first_ngens(1)
>>> L = K.extension(x**Integer(2) - b, names=('c',)); (c,) = L._first_ngens(1)
>>> F = L.absolute_field(names=('m',)); (m,) = F._first_ngens(1)
```
An example of an inert ideal:

```python
sage: P = F.factor(13)[0][0]; P
Fractional ideal (13)
sage: J = L.ideal(13)
sage: J.absolute_ideal()
Fractional ideal (13)
```

```python
>>> from sage.all import *

>>> P = F.factor(Integer(13))[Integer(0)][Integer(0)]; P
Fractional ideal (13)
>>> J = L.ideal(Integer(13))
>>> J.absolute_ideal()
Fractional ideal (13)
```

Now a non-trivial ideal in $L$ that is principal in the subfield $K$. Since the optional names argument is not passed, the generators of the absolute ideal $J$ are returned in terms of the default field generator $a$. This does not agree with the generator $m$ of the absolute field $F$ defined above:

```python
sage: J = L.ideal(b); J
Fractional ideal (b)
sage: J.absolute_ideal()
Fractional ideal ($a^2$)
sage: J.relative_norm()
Fractional ideal (2)
sage: J.absolute_norm() 4
sage: J.absolute_ideal().norm() 4
```

```python
>>> from sage.all import *

>>> J = L.ideal(b); J
Fractional ideal (b)
>>> J.absolute_ideal()
Fractional ideal ($a^2$)
>>> J.relative_norm()
Fractional ideal (2)
>>> J.absolute_norm() 4
>>> J.absolute_ideal().norm() 4
```

Now pass $m$ as the name for the generator of the absolute field:

```python
sage: J.absolute_ideal('m')
Fractional ideal ($m^2$)
```

```python
>>> from sage.all import *

>>> J.absolute_ideal('m')
Fractional ideal ($m^2$)
```

Now an ideal not generated by an element of $K$:

```python
sage: J = L.ideal(c); J
Fractional ideal (c)
sage: J.absolute_ideal()
Fractional ideal (a)
```

(continues on next page)
### absolute_norm()

Compute the absolute norm of this fractional ideal in a relative number field, returning a positive integer.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: L.<a, b, c> = QQ.extension([x^2 - 23, x^2 - 5, x^2 - 7])
sage: I = L.ideal(a + b)
sage: I.absolute_norm()
104976
sage: I.relative_norm().relative_norm().relative_norm()
104976
```

### absolute_ramification_index()

Return the absolute ramification index of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The absolute ramification index is the power of this prime appearing in the factorization of the rational prime that this prime lies over.

Use `relative_ramification_index()` to obtain the power of this prime occurring in the factorization of the prime ideal of the base field that this prime lies over.

**EXAMPLES:**

```python
>>> from sage.all import *

```

```python
from sage.all import *

>>> J = L.ideal(c); J
Fractional ideal (c)
>>> J.absolute_ideal()
Fractional ideal (a)
>>> J.absolute_norm()
2
>>> J.ideal_below()
Fractional ideal (b)
>>> J.ideal_below().norm()
2
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> L = QQ.extension([x**Integer(2) - Integer(23), x**Integer(2) - Integer(5), x**Integer(2) - Integer(7)], names=('a', 'b', 'c')); (a, b, c,) = L._first_ngens(3)

>>> I = L.ideal(a + b)

>>> I.absolute_norm()
104976

>>> I.relative_norm().relative_norm().relative_norm()
104976
```
element_1_mod

Returns an element \( r \) in this ideal such that \( 1 - r \) is in \( \text{other} \).

An error is raised if either ideal is not integral of if they are not coprime.

**INPUT:**

- \( \text{other} \) – another ideal of the same field, or generators of an ideal.

**OUTPUT:**

an element \( r \) of the ideal \( \text{self} \) such that \( 1 - r \) is in the ideal \( \text{other} \).

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberFieldTower([x**2 - 23, x**2 + 1])
sage: I = Ideal(2, (a - 3*b + 2)/2)
sage: J = K.ideal(a)
sage: z = I.element_1_mod(J)
sage: z in I
True
sage: 1 - z in J
True
```
factor()  
Factor the ideal by factoring the corresponding ideal in the absolute number field.

EXAMPLES:

```python
sage: factor(5)
Fractional ideal (5, (-1/4*b - 1/4)*a + 1/4*b - 3/4))^2
* (Fractional ideal (5, (-1/4*b - 1/4)*a + 1/4*b - 7/4))^2
sage: K.ideal(5).factor()
(Fractional ideal (5, (-1/4*b - 1/4)*a + 1/4*b - 3/4))^2
* (Fractional ideal (5, (-1/4*b - 1/4)*a + 1/4*b - 7/4))^2
sage: K.ideal(5).prime_factors()
[(Fractional ideal (5, (-1/4*b - 1/4)*a + 1/4*b - 3/4),
  Fractional ideal (5, (-1/4*b - 1/4)*a + 1/4*b - 7/4))]
sage: P = K.ideal((b*a - b - 1)*c/2 + a - 1)
sage: Q = K.ideal((b*a - b - 1)*c/2)
sage: list(I.factor()) == [(P, 2), (Q, 1)]
True
sage: I == P^2*Q
True
```

(continues from previous page)
\begin{verbatim}
from sage.all import *

x = polygen(ZZ, 'x')
K.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
I = K.ideal(a*b - 1)
I.free_module().is_submodule(K.maximal_order().free_module())
True

from sage.all import *

x = polygen(ZZ, 'x')
K = NumberField([x**Integer(3) - x + Integer(1), x**Integer(2) - Integer(2)],
names=('a', 'b',)); (a, b) = K._first_ngens(2)
I = K.ideal((a + Integer(1))*b/Integer(2) + Integer(1))
I.gens_reduced()
(1/2*b*a + 1/2*b + 1,)

from sage.all import *

x = polygen(ZZ, 'x')
K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(2)],
names=('a', 'b',)); (a, b) = K._first_ngens(2)
I = K.ideal((a + Integer(1))*b/Integer(2) + Integer(1))
I.gens_reduced()
(1/2*b*a + 1/2*b + 1,)
\end{verbatim}
ideal_below()

Compute the ideal of $K$ below this ideal of $L$.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2 + 6)
sage: L.<b> = K.extension(K[x].gen()^4 + a)
sage: N = L.ideal(b)
sage: M = N.ideal_below(); M == K.ideal([-a])
True
sage: Np = L.ideal([L(t) for t in M.gens()])
sage: Np.ideal_below() == M
True
sage: M.parent()
Monoid of ideals of Number Field in a with defining polynomial x^2 + 6
sage: M.ring()
Number Field in a with defining polynomial x^2 + 6
sage: M.ring() is K
True
```

This example concerns an inert ideal:

```
sage: K = NumberField(x^4 + 6*x^2 + 24, 'a')
sage: K.factor(7)
Fractional ideal (7)
sage: K0, K0_into_K, _ = K.subfields(2)[0]
sage: K0
Number Field in a0 with defining polynomial x^2 - 6*x + 24
sage: L = K.relativize(K0_into_K, 'c'); L
Number Field in c with defining polynomial x^2 + a0 over its base field
sage: L.base_field() is K0
True
sage: L.ideal(7)
(continues on next page)
```
This example concerns an ideal that splits in the quadratic field but each factor ideal remains inert in the extension:

```python
sage: len(K.factor(19))
2
sage: K0 = L.base_field(); a0 = K0.gen()
sage: len(K0.factor(19))
2
sage: w1 = -a0 + 1; P1 = K0.ideal([w1])
sage: P1.norm().factor(), P1.is_prime()
(19, True)
sage: L_into_K, K_into_L = L.structure()
sage: L.ideal(K_into_L(K0_into_K(w1))).ideal_below() == P1
True
```

The choice of embedding of quadratic field into quartic field matters:

```python
>>> from sage.all import *
>>> len(K.factor(19))
2
>>> K0 = L.base_field(); a0 = K0.gen()
>>> len(K0.factor(19))
2
>>> w1 = -a0 + Integer(1); P1 = K0.ideal([w1])
>>> P1.norm().factor(), P1.is_prime()
(19, True)
```

The choice of embedding of quadratic field into quartic field matters:
Algebraic Numbers and Number Fields, Release 10.4

sage: rho, tau = K0.embeddings(K)
sage: L1 = K.relativize(rho, 'b')
sage: L2 = K.relativize(tau, 'b')
sage: L1_into_K, K_into_L1 = L1.structure()
sage: L2_into_K, K_into_L2 = L2.structure()
sage: a = K.gen()
sage: P = K.ideal([a**Integer(2) + 5])
sage: K_into_L1(P).ideal_below() == K0.ideal([-a0 + 1])
True
sage: K_into_L2(P).ideal_below() == K0.ideal([-a0 + 5])
True
sage: K0.ideal([-a0 + 1]) == K0.ideal([-a0 + 5])
False

It works when the base field is itself a relative number field:

sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X**Integer(2) - Integer(2), X**Integer(2) - Integer(3)])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y**Integer(2) - (1 + a)*(a + b)*a*b)
sage: I = K.ideal(3, c)
sage: J = I.ideal_below()
sage: J == K.ideal(b)
True
sage: J.number_field() == F
True

Number fields defined by non-monic and non-integral polynomials are supported (Issue #252):

>>> from sage.all import *
>>> rho, tau = K0.embeddings(K)
>>> L1 = K.relativize(rho, 'b')
>>> L2 = K.relativize(tau, 'b')
>>> L1_into_K, K_into_L1 = L1.structure()
>>> L2_into_K, K_into_L2 = L2.structure()
>>> a = K.gen()
>>> P = K.ideal([a**Integer(2) + Integer(5)])
>>> K_into_L1(P).ideal_below() == K0.ideal([-a0 + Integer(1)])
True
>>> K_into_L2(P).ideal_below() == K0.ideal([-a0 + Integer(5)])
True
>>> K0.ideal([-a0 + Integer(1)]) == K0.ideal([-a0 + Integer(5)])
False

4.3. Ideals of relative number fields

507
```python
sage: K.<a> = NumberField(2*x^2 - 1/3)
sage: L.<b> = K.extension(5*x^2 + 1)
sage: P = L.primes_above(2)[0]
sage: P.ideal_below()
Fractional ideal (6*a + 2)
```

```python
>>> from sage.all import *
>>> K = NumberField(Integer(2)*x**Integer(2) - Integer(1)/Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> L = K.extension(Integer(5)*x**Integer(2) + Integer(1), names=('b',)); (b,)
>>> P = L.primes_above(Integer(2))[Integer(0)]
>>> P.ideal_below()
Fractional ideal (6*a + 2)
```

integral_basis()

Return a basis for `self` as a \(\mathbb{Z}\)-module.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: I = K.ideal(17*b - 3*a)
sage: x = I.integral_basis(); x
# random
[438, -b*a + 309, 219*a - 219*b, 156*a - 154*b]
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(3)], names=('a', 'b',)); (a, b,)
>>> I = K.ideal(Integer(17)*b - Integer(3)*a)
>>> x = I.integral_basis(); x
# random
[438, -b*a + 309, 219*a - 219*b, 156*a - 154*b]
```

The exact results are somewhat unpredictable, hence the `# random` flag, but we can test that they are indeed a basis:

```python
sage: V, _, phi = K.absolute_vector_space()
sage: V.span([phi(u) for u in x], ZZ) == I.free_module()
True
```

```python
>>> from sage.all import *
>>> V, _, phi = K.absolute_vector_space()
>>> V.span([phi(u) for u in x], ZZ) == I.free_module()
True
```

integral_split()

Return a tuple \((I, d)\), where \(I\) is an integral ideal, and \(d\) is the smallest positive integer such that this ideal is equal to \(I/d\).

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberFieldTower([x^2 - 23, x^2 + 1])
sage: I = K.ideal([a + b/3])
sage: J, d = I.integral_split()
(continues on next page)
```
sage: J.is_integral()
True
sage: J == d*I
True

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberFieldTower([x**Integer(2) - Integer(23), x**Integer(2) +
→Integer(1)], names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>>> I = K.ideal([a + b/Integer(3)])
>>> J, d = I.integral_split()
>>> J.is_integral()
True
>>> J == d*I
True

is_integral()

Return True if this ideal is integral.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = QQ.extension([x^2 + 11, x^2 - 5])
sage: I = K.ideal(7).prime_factors()[0]
sage: I.is_integral()
True
sage: (I/2).is_integral()
False

is_prime()

Return True if this ideal of a relative number field is prime.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 - 17, x^3 - 2])
sage: K.ideal(a + b).is_prime()
True
sage: K.ideal(13).is_prime()
False

(continues on next page)
is_prime()  
Return True if this ideal is prime. If so, set self.__reduced_generators, with length one.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 - 23, x^2 + 1])
sage: I = K.ideal([7, (-1/2*b - 3/2)*a + 3/2*b + 9/2])
sage: I.is_prime()
True
```

is_principal(proof=None)  
Return True if this ideal is principal. If so, set self.__reduced_generators, with length one.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x**2 - 23, x**2 + 1])
sage: I = K.ideal([7, (-1/2*b - 3/2)*a + 3/2*b + 9/2])
```

is_zero()  
Return True if this is the zero ideal.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x**2 + 3, x**3 + 4])
sage: K.ideal(17).is_zero()
False
```

norm()  
The norm of a fractional ideal in a relative number field is deliberately unimplemented, so that a user cannot mistake the absolute norm for the relative norm, or vice versa.

EXAMPLES:
Algebraic Numbers and Number Fields, Release 10.4

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 2])
sage: K.ideal(2).norm()
Traceback (most recent call last):
...
NotImplementedError: For a fractional ideal in a relative number field you must use relative_norm or absolute_norm as appropriate
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(2)],
... names=('a', 'b',)); (a, b,) = K._first_ngens(2)

>>> K.ideal(Integer(2)).norm()
Traceback (most recent call last):
...
NotImplementedError: For a fractional ideal in a relative number field you must use relative_norm or absolute_norm as appropriate
```

### pari_rhnf()

Return PARI's representation of this relative ideal in Hermite normal form.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')

sage: K.<a, b> = NumberField([x^2 + 23, x^2 - 7])
sage: I = K.ideal(2, (a + 2*b + 3)/2)
sage: I.pari_rhnf()
[[1, -2; 0, 1], [[2, 1; 0, 1], 1/2]]
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField([x**Integer(2) + Integer(23), x**Integer(2) - Integer(7)],
... names=('a', 'b',)); (a, b,) = K._first_ngens(2)

>>> I = K.ideal(Integer(2), (a + Integer(2)*b + Integer(3))/Integer(2))

>>> I.pari_rhnf()
[[1, -2; 0, 1], [[2, 1; 0, 1], 1/2]]
```

### ramification_index()

For ideals in relative number fields, `ramification_index()` is deliberately not implemented in order to avoid ambiguity. Either `relative_ramification_index()` or `absolute_ramification_index()` should be used instead.

**EXAMPLES:**

```python
sage: x = polygen(ZZ, 'x')

sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 2])
sage: K.ideal(2).ramification_index()
Traceback (most recent call last):
...
NotImplementedError: For an ideal in a relative number field you must use relative_ramification_index or absolute_ramification_index as appropriate
```

```python
>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField([x**Integer(2) + Integer(1), x**Integer(2) - Integer(2)],
... names=('a', 'b',)); (a, b,) = K._first_ngens(2)

(continues on next page)```
relative_norm()

Compute the relative norm of this fractional ideal in a relative number field, returning an ideal in the base field.

EXAMPLES:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2 + 6)
sage: L.<b> = K.extension(K['x'].gen()^4 + a)
sage: N = L.ideal(b).relative_norm(); N
Fractional ideal (-a)
sage: N.parent()
Monoid of ideals of Number Field in a with defining polynomial x^2 + 6
e
sage: N.ring()
Number Field in a with defining polynomial x^2 + 6
e
```

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.ideal(1).relative_norm()
Fractional ideal (1)
sage: K.ideal(13).relative_norm().relative_norm()
Fractional ideal (28561)
sage: K.ideal(13).relative_norm().relative_norm().relative_norm()
815730721
sage: K.ideal(13).absolute_norm()
815730721
```
Number fields defined by non-monic and non-integral polynomials are supported (Issue #252):

```python
sage: K.<a> = NumberField(2*x^2 - 1/3)
sage: L.<b> = K.extension(5*x^2 + 1)
sage: P = L.primes_above(2)[0]
sage: P.relative_norm()
Fractional ideal (6*a + 2)
```

relative_ramification_index()

Return the relative ramification index of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The relative ramification index is the power of this prime appearing in the factorization of the prime ideal of the base field that this prime lies over.

Use `absolute_ramification_index()` to obtain the power of this prime occurring in the factorization of the rational prime that this prime lies over.

EXAMPLES:

```python
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = K.ideal(3, c)
sage: I.relative_ramification_index()
2
sage: I.ideal_below()  # random sign
Fractional ideal (b)
sage: I.ideal_below() == K.ideal(b)
True
sage: K.ideal(b) == I^2
True
```

```python
>>> from sage.all import *
```

```python
>>> PQ = QQ[X]; (X,) = PQ._first_ngens(1)
>>> F = NumberFieldTower([X**Integer(2) - Integer(2), X**Integer(2) - Integer(3)], names=(
    -'a', 'b')); (a, b,) = F._first_ngens(2)
>>> PF = F[Y]; (Y,) = PF._first_ngens(1)
>>> K = F.extension(Y**Integer(2) - (Integer(1) + a)*(a + b)*a*b, names=('c',
    -)); (c,) = K._first_ngens(1)
>>> I = K.ideal(Integer(3), c)
```

(continues on next page)
I.relative_ramification_index()
2
I.ideal_below()  # random sign
Fractional ideal (b)
I.ideal_below() == K.ideal(b)  True
K.ideal(b) == I**Integer(2)  True

residue_class_degree()
Return the residue class degree of this prime.

EXAMPLES:

```python
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X**2 - 2, X**2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y**2 - (1 + a)*(a + b)*a*b)
sage: [I.residue_class_degree() for I in K.ideal(c).prime_factors()]
[1, 2]
```

residues()
Returns a iterator through a complete list of residues modulo this integral ideal.
An error is raised if this fractional ideal is not integral.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, w> = NumberFieldTower([x**2 - 3, x**2 + x + 1])
sage: I = K.ideal(5, -w*a - w + 4)
sage: list(I.residues())[:5]
[(25/3*w - 1/3)*a + 22*w + 1,
 (16/3*w - 1/3)*a + 13*w,
 (7/3*w - 1/3)*a + 4*w - 1,
 (-2/3*w - 1/3)*a - 5*w - 2,
 (-11/3*w - 1/3)*a - 14*w - 3]
```

(continues on previous page)
(continued from previous page)

\[ \begin{align*}
(7/3*w - 1/3)*a + 4*w - 1, \\
(-2/3*w - 1/3)*a - 5*w - 2, \\
(-11/3*w - 1/3)*a - 14*w - 3
\end{align*} \]

(smallest_integer())

Return the smallest non-negative integer in \( I \cap \mathbb{Z} \), where \( I \) is this ideal. If \( I = 0 \), returns 0.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberFieldTower([x^2 - 23, x^2 + 1])
sage: I = K.ideal([a + b])
sage: I.smallest_integer()
12
sage: [m for m in range(13) if m in I]
[0, 12]
```

(continued on next page)

>> from sage.all import *
>> x = polygen(ZZ, 'x')
>> K = NumberFieldTower([x^2 - Integer(23), x^2 + 1]
˓→Integer(1)], names=('a', 'b',)); (a, b,) = K._first_ngens(2)
>> I = K.ideal([a + b])
>> I.smallest_integer()
12
>> [m for m in range(Integer(13)) if m in I]
[0, 12]

(continues on next page)

(continued from previous page)

valuation \( (p) \)

Return the valuation of this fractional ideal at \( p \).

INPUT:

- \( p \) – a prime ideal \( p \) of this relative number field.

OUTPUT:

(integer) The valuation of this fractional ideal at the prime \( p \). If \( p \) is not prime, raise a ValueError.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a, b> = NumberField([x^2 - 17, x^3 - 2])
sage: A = K.ideal(a + b)
sage: A.is_prime()
True
sage: (A*K.ideal(3)).valuation(A)
1
sage: K.ideal(25).valuation(5)
Traceback (most recent call last):
  ... ValueError: p (= Fractional ideal (5)) must be a prime
```

```
```
sage.rings.number_field.number_field_ideal_rel.is_NumberFieldFractionalIdeal_rel(x)

Return True if \( x \) is a fractional ideal of a relative number field.

EXAMPLES:

```python
sage: from sage.rings.number_field.number_field_ideal_rel import is_
    →NumberFieldFractionalIdeal_rel
sage: from sage.rings.number_field.number_field_ideal import is_
    →NumberFieldFractionalIdeal
sage: is_NumberFieldFractionalIdeal_rel(2/3)
False
sage: is_NumberFieldFractionalIdeal_rel(ideal(5))
False
sage: x = polygen(ZZ, 'x')
sage: k.<a> = NumberField(x^2 + 2)
sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
sage: is_NumberFieldFractionalIdeal_rel(I)
False
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2 + 6)
sage: L.<b> = K.extension(K['x'].gen()^4 + a)
sage: I = L.ideal(b); I
Fractional ideal (6, b)
sage: is_NumberFieldFractionalIdeal_rel(I)
True
sage: N = I.relative_norm(); N
Fractional ideal (-a)
sage: is_NumberFieldFractionalIdeal_rel(N)
False
```
4.4 Ideals of (Not Necessarily Maximal) Orders in Number Fields

This module implements (integral) ideals of orders in number fields.

**Note:** Currently, Sage only offers very limited functionality for ideals of non-maximal orders (compared to the maximal case). This should hopefully change in the future.

**EXAMPLES:**

```python
sage: O = QuadraticField(-1).order(5*i)
sage: I = O.ideal([13, 5*i-1]); I
Ideal (60*a + 1, 65*a) of Order of conductor 5 generated by 5*a
in Number Field in a with defining polynomial x^2 + 1 with a = 1*I

>>> from sage.all import *

| O = QuadraticField(-Integer(1)).order(Integer(5)*i) |
| I = O.ideal([Integer(13), Integer(5)*i-Integer(1)]); I |
| Ideal (60*a + 1, 65*a) of Order of conductor 5 generated by 5*a |
| in Number Field in a with defining polynomial x^2 + 1 with a = 1*I |
```

An ideal of an order in a relative number field:

```python
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: O = K.order([3*a, 2*b])
sage: I = O.ideal([-6*b + 6]*a + 6*b + 18]; I
Ideal ((-60*b + 180)*a + 72, (-54*b + 174)*a - 6*b + 54, (-72*b + 288)*a + 72, 1872*a)
of Relative Order generated by [3*a - 2*b, -6*b*a + 6, 3*a]
in Number Field in a with defining polynomial x^2 + 1 over its base field
```
Perhaps the most useful functionality at this time is mapping ideals of quadratic orders to corresponding binary quadratic forms:

```python
sage: K.<t> = QuadraticField(-21463)
sage: O = K.order(t)
sage: I = O.ideal([123457, t + 45259]); I
Ideal (23058*t + 1, 123457*t) of Order of conductor 26 generated by t
    in Number Field in t with defining polynomial x^2 + 21463 with t = 146.5025597046004?

sage: I.quadratic_form()
123457*x^2 - 90518*x*y + 16592*y^2
```

Todo: Generalize more functionality (such as primality testing and factoring) from `NumberFieldFractionalIdeal` to ideals of not necessarily maximal orders.

AUTHORS:

- Lorenz Panny (2022)
```python
>>> from sage.all import *
>>> from sage.rings.number_field.order_ideal import NumberFieldOrderIdeal

>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> K = NumberField(x**Integer(3) - Integer(40), names=('t',)); (t,) = K._first_ngens(1)
>>> O = K.order(t)
>>> I = NumberFieldOrderIdeal(O, [Integer(13), t-Integer(1)]); I
Ideal (12*t^2 + 1, 12*t^2 + t, 13*t^2) of Order generated by t in Number Field in t with defining polynomial x^3 - 40
>>> K.absolute_degree()
3
>>> type(I)
<class 'sage.rings.number_field.order_ideal.NumberFieldOrderIdeal_generic'>
```

```python
sage: L.<u> = QuadraticField(-3)
sage: J = NumberFieldOrderIdeal(L.maximal_order(), [(u+5)/2])
sage: L.absolute_degree()
2
sage: type(J)
<class 'sage.rings.number_field.order_ideal.NumberFieldOrderIdeal_quadratic'>
```

```python
>>> from sage.all import *

>>> L = QuadraticField(-Integer(3), names=('u',)); (u,) = L._first_ngens(1)
>>> J = NumberFieldOrderIdeal(L.maximal_order(), [(u+Integer(5))/Integer(2)])
>>> L.absolute_degree()
2
>>> type(J)
<class 'sage.rings.number_field.order_ideal.NumberFieldOrderIdeal_quadratic'>
```

class sage.rings.number_field.order_ideal.NumberFieldOrderIdeal_generic(O, gens, *, coerce=True)

Bases: Ideal_generic

An ideal of a not necessarily maximal order in a number field.

free_module()

Return the free $\mathbb{Z}$-module corresponding to this ideal as a submodule of the vector space associated to the ambient number field.

EXAMPLES:

```python
sage: K.<t> = QuadraticField(-123)
sage: g, = K.ring_of_integers().ring_generators()
sage: O = K.order(g)
sage: I = O.ideal([191, 567*t-27]); I
Ideal (56133/2*t + 1/2, 108297*t) of Order of conductor 567 generated by 567/2*t + 1/2 in Number Field in t with defining polynomial x^2 + 123 with t = 11.

sage: I.free_module()
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[ 1/2 56133/2]
[ 0 108297]
sage: I.free_module().is_submodule(O.free_module())
True
```
>>> from sage.all import *
>>> K = QuadraticField(-Integer(123), names=('t',)); (t,) = K._first_ngens(1)
>>> g, = K.ring_of_integers().ring_generators()
>>> O = K.order(Integer(567)*g)
>>> I = O.ideal([Integer(191), Integer(567)*t-Integer(27)]); I
Ideal (56133/2*t + 1/2, 108297*t) of Order of conductor 567 generated by 567/ ˓→2*t + 1/2
in Number Field in t with defining polynomial x^2 + 123 with t = 11.
>>> O.ideal([Integer(191), Integer(567)*t-Integer(27)]); I
Ideal (56133/2*t + 1/2, 108297*t) of Order of conductor 567 generated by 567/ ˓→2*t + 1/2
in Number Field in t with defining polynomial x^2 + 123 with t = 11.
>>> I.free_module()
Free module of degree 2 and rank 2 over Integer Ring
Echelon basis matrix:
[ 1/2 56133/2]
[ 0 108297]
>>> I.free_module().is_submodule(O.free_module())
True

See also:

- `sage.rings.number_field.number_field.NumberField_absolute.absolute_vector_space()`
- `sage.rings.number_field.order.Order.free_module()`
- `sage.rings.number_field.number_field_ideal.NumberFieldIdeal.free_module()`

**norm()**

Return the norm of this ideal.

The norm is defined as the index (as an abelian group) of the ideal in its order.

EXAMPLES:

```python
sage: K.<t> = QuadraticField(-123)
sage: g, = K.ring_of_integers().ring_generators()
sage: O = K.order(567*g)
sage: I = O.ideal([191, 567*t-27])
sage: I.norm()
191
sage: (O.free_module() / I.free_module()).cardinality()
191
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(123), names=('t',)); (t,) = K._first_ngens(1)
>>> g, = K.ring_of_integers().ring_generators()
>>> O = K.order(Integer(567)*g)
>>> I = O.ideal([Integer(191), Integer(567)*t-Integer(27)]);
```

```python
class sage.rings.number_field.order_ideal.
NumberFieldOrderIdeal_quadratic(O, gens,
*, coerce=True)

Bases: NumberFieldOrderIdeal_generic
```

Chapter 4. Orders, Ideals and Ideal Classes
An ideal of a not necessarily maximal order in a quadratic number field.

**conjugate()**

Return the conjugate of this ideal, defined by conjugating the generators.

**EXAMPLES:**

```
sage: K.<t> = QuadraticField(-123)
sage: g, = K.ring_of_integers().ring_generators()
sage: O = K.order(567^g)
sage: I = O.ideal([191, 567*t-27])
sage: I.norm()
191
sage: I.norm() in I.conjugate() * I
True
sage: I.conjugate() * I == I.norm() * O
True
```

**gens_reduced()**

Express this ideal in terms of at most two generators, and one if possible (i.e., if the ideal is principal).

**EXAMPLES:**

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^2 + 11*x + 5)
sage: O = K.order(7*a)
sage: I = O.ideal([31915, -71145879*a - 32195694])
sage: I.gens_reduced()
(-63*a + 17,)
```

**ALGORITHM:**

Compute a reduction of the quadratic_form() to see if it represents 1, then use the transformation matrix to find an element in the ideal whose norm equals the norm of the ideal.

**gens_two()**

Express this ideal using exactly two generators, the first of which is a generator for the intersection of the ideal with \( \mathbb{Z} \).
EXAMPLES:

```sage
sage: K.<t> = QuadraticField(-100)
sage: O = K.order(t)
sage: I = O.ideal([123, 131-t, 21+23*t])
sage: I.gens_two()
(41, t - 8)
sage: I == O.ideal(I.gens_two())
True
```

These second generator is zero if and only if the ideal is generated by an integer:

```sage
sage: J = O.ideal([-33*t, 11*t-6589])
sage: J.gens_two()
(11, 0)
sage: J == O.ideal(11)
True
```

**Warning:** The returned generators do not necessarily form a \( \mathbb{Z} \)-basis of the ideal.

```sage
>>> from sage.all import *
>>> K = QuadraticField(-Integer(100), names=('t',)); (t,) = K._first_ngens(1)
>>> O = K.order(t)
>>> I = O.ideal([Integer(123), Integer(131)-t, Integer(21)+Integer(23)*t])
>>> I.gens_two()
(41, t - 8)
>>> I == O.ideal(I.gens_two())
True
```

**is_equivalent** *(other, narrow=False)*

Determine whether this ideal is equivalent to another ideal in the same order.

If `narrow` is `True`, test narrow equivalence instead.

(Two ideals are equivalent if they differ by multiplication by a non-zero element. They are narrowly equivalent if they differ by multiplication by an element of positive norm.)

EXAMPLES:

```sage
sage: K.<a> = QuadraticField(-163)
sage: O = K.order(7*a)
sage: I = O.ideal([47, 7*a-35])
sage: J = O.ideal([71, 7*a-65])
sage: I.is_equivalent(J)
False
sage: (I^10).is_equivalent(J)
True
```
>>> from sage.all import *
>>> K = QuadraticField(-Integer(163), names=('a',)); (a,) = K._first_ngens(1)
>>> O = K.order(Integer(7)*a)
>>> I = O.ideal([Integer(47), Integer(7)*a-Integer(35)])
>>> J = O.ideal([Integer(71), Integer(7)*a-Integer(65)])
>>> I.is_equivalent(J)
False
>>> (I**Integer(10)).is_equivalent(J)
True

sage: K.<a> = QuadraticField(229)
sage: O = K.order(7*a)
sage: O.class_number()
3
sage: I = O.ideal([3, 7*a-2])
sage: J = O.ideal([5, 7*a-4])
sage: I.is_equivalent(J)
True

>>> from sage.all import *
>>> K = QuadraticField(Integer(229), names=('a',)); (a,) = K._first_ngens(1)
>>> O = K.order(Integer(7)*a)
>>> O.class_number()
3
>>> I = O.ideal([Integer(3), Integer(7)*a-Integer(2)])
>>> J = O.ideal([Integer(5), Integer(7)*a-Integer(4)])
>>> I.is_equivalent(J)
True

sage: K.<a> = QuadraticField(273)
sage: O = K.order(11*a)
sage: O.class_number()
20
sage: I = O.ideal([17, 11*a-11])
sage: J = O.ideal([19, 11*a-12])
sage: I.is_equivalent(J)
False
sage: (I**3).is_equivalent(J)
False
sage: (I**6).is_equivalent(J**2)
True
sage: el = 177 + 11*a
sage: el.norm()
-1704
sage: (I**6).is_equivalent(J**2, narrow=True)
True
sage: (I**6).is_equivalent(J**2*el, narrow=True)
False

>>> from sage.all import *
>>> K = QuadraticField(Integer(273), names=('a',)); (a,) = K._first_ngens(1)
>>> O = K.order(Integer(11)*a)
>>> O.class_number()
20
>>> I = O.ideal([Integer(17), Integer(11)*a-Integer(11)])
>>> J = O.ideal([Integer(19), Integer(11)*a-Integer(12)])
is_principal()

Determine whether or not this ideal is principal.

See also:
To find a generator, use gens_reduced().

EXAMPLES:

sage: K.<a> = QuadraticField(-163)
sage: O = K.order(7*a)
sage: O.class_number() 24
sage: order = lambda v: next(e for e in range(1,99) if (v^e).is_principal())
sage: I = O.ideal([47,7*a-35])
sage: order(I) 24
sage: J = O.ideal([71,7*a-65])
sage: order(J) 12
sage: next(e for e in range(99) if (I^e * J.conjugate()).is_principal()) 10
sage: (I^10 * J.conjugate()).is_principal() True

sage: K.<a> = QuadraticField(229)
sage: O = K.order(7*a)
sage: I = O.ideal([3,7*a-2])
sage: J = O.ideal([5,7*a-4])
sage: (I * J.conjugate()).is_principal() True
sage: el = 104 + 7*a
sage: el.norm() -405
sage: (I * (J * el).conjugate()).is_principal() True

from sage.all import *

K = QuadraticField(Integer(229), names=('a',)); (a,) = K._first_ngens(1)
O = K.order(Integer(7)*a)
I = O.ideal([Integer(3), Integer(7)*a-Integer(2)])
J = O.ideal([Integer(5), Integer(7)*a-Integer(4)])
(I * J.conjugate()).is_principal() True
el = Integer(104) + Integer(7)*a
el.norm() -405
(I * (J * el).conjugate()).is_principal() True

quadratic_form(basis)

Return the binary quadratic form associated to this ideal.

This map induces an injective homomorphism from the narrow class group on ideals to the class group on quadratic forms.
If `basis` is set to `True` (default: `False`), the method additionally returns a \( \mathbb{Z} \)-basis \((a, b)\) of this ideal \( I \) such that \( f(x, y) \) equals \( \text{norm}(xa + yb)/\text{norm}(I) \), where \( f \) is the returned quadratic form.

**Note:** The narrow class group is the group of invertible ideals modulo the principal ideals generated by an element of positive norm.

- For *imaginary* quadratic orders, the narrow class group is identical to the class group.
- For *real* quadratic orders, identifying the classes of \( f(x, y) \) and \(-f(y, x)\) recovers a correspondence with the standard class group.

**REFERENCES:**

The correspondence itself is classical. Implemented after [Coh93], §5.2.

See also:
sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal
quadratic_form()

**EXAMPLES:**

```
sage: K.<t> = QuadraticField(-419)
sage: O = K.order(t)
sage: O.discriminant().factor()
-1 * 2^2 * 419
sage: I = O.ideal([t-1, 105]); I
Ideal (104*t + 1, 105*t) of Order of conductor 2 generated by t
in Number Field in t with defining polynomial x^2 + 419 with t = 20.
→ 46948949045873?*I
sage: f = I.quadratic_form(); f
105*x^2 - 208*x*y + 107*y^2
sage: f.discriminant().factor()
-1 * 2^2 * 419
sage: power(f,3).reduced_form()
x^2 + 419*y^2
>>> from sage.all import *
>>> K = QuadraticField(-Integer(419), names=('t',)); (t,) = K._first_ngens(1)
>>> O = K.order(t)
>>> O.discriminant().factor()
-1 * 2^2 * 419
>>> I = O.ideal([t-Integer(1), Integer(105)]); I
Ideal (104*t + 1, 105*t) of Order of conductor 2 generated by t
in Number Field in t with defining polynomial x^2 + 419 with t = 20.
→ 46948949045873?*I
>>> f = I.quadratic_form(); f
105*x^2 - 208*x*y + 107*y^2
>>> f.discriminant().factor()
-1 * 2^2 * 419
>>> power(f,Integer(3)).reduced_form()
x^2 + 419*y^2
sage: u = 23*t - 45
sage: J = I*u
sage: g = J.quadratic_form(); g
23485980*x^2 - 22795498*x*y + 5531329*y^2
```

(continues on next page)
4.5 Class groups of number fields

An element of a class group is stored as a pair consisting of both an explicit ideal in that ideal class, and a list of exponents giving that ideal class in terms of the generators of the parent class group. These can be accessed with the ideal() and exponents() methods respectively.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 23)
sage: I = K.class_group().gen(); I
Fractional ideal class (2, 1/2*a - 1/2)
sage: I.ideal()
Fractional ideal (2, 1/2*a - 1/2)
sage: I.exponents()
(1,)
sage: I.ideal() * I.ideal()
Fractional ideal (4, 1/2*a + 3/2)
sage: (I.ideal() * I.ideal()).reduce_equiv()
Fractional ideal (2, 1/2*a + 1/2)
sage: J = I * I; J  
# class group multiplication is automatically reduced
Fractional ideal class (2, 1/2*a + 1/2)
sage: J.ideal()
Fractional ideal (2, 1/2*a + 1/2)
```
sage: J.exponents()
(2,)

sage: I * I.ideal()  # ideal classes coerce to their representative ideal
Fractional ideal (4, 1/2*a + 3/2)

sage: K.fractional_ideal([2, 1/2*a + 1/2])
Fractional ideal (2, 1/2*a + 1/2)

sage: K.fractional_ideal([2, 1/2*a + 1/2]).is_principal()
False

sage: K.fractional_ideal([2, 1/2*a + 1/2])^3
Fractional ideal (1/2*a - 3/2)

>>> from sage.all import *

>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x^2 + Integer(23), names=('a',)); (a,) = K._first_ngens(1)

>>> I = K.class_group().gen(); I
Fractional ideal class (2, 1/2*a - 1/2)

>>> I.ideal()
Fractional ideal (2, 1/2*a - 1/2)

>>> I.exponents()
(1,)

>>> I.ideal() * I.ideal()
Fractional ideal (4, 1/2*a + 3/2)

>>> (I.ideal() * I.ideal()).reduce_equiv()
Fractional ideal (2, 1/2*a + 1/2)

>>> J = I * I; J  # class group multiplication is automatically reduced
Fractional ideal class (2, 1/2*a + 1/2)

>>> J.ideal()
Fractional ideal (2, 1/2*a + 1/2)

>>> J.exponents()
(2,)

>>> I * I.ideal()  # ideal classes coerce to their representative ideal
Fractional ideal (4, 1/2*a + 3/2)

>>> K.fractional_ideal([Integer(2), Integer(1)/Integer(2)*a + Integer(1)/Integer(2)])
Fractional ideal (2, 1/2*a + 1/2)

>>> K.fractional_ideal([Integer(2), Integer(1)/Integer(2)*a + Integer(1)/Integer(2)]).is_principal()
False

>>> K.fractional_ideal([Integer(2), Integer(1)/Integer(2)*a + Integer(1)/Integer(2)])^3*Integer(3)
Fractional ideal (1/2*a - 3/2)

class sage.rings.number_field.class_group.ClassGroup (gens_orders, names, number_field, gens, proof=True)

Bases: AbelianGroupWithValues_class

The class group of a number field.

EXAMPLES:

sage: x = polygen(ZZ, 'x')

sage: K.<a> = NumberField(x^2 + 23)
sage: G = K.class_group(); G
Class group of order 3 with structure C3 of
Number Field in a with defining polynomial x^2 + 23
sage: G.category()
Category of finite enumerated commutative groups

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) + Integer(23), names=('a',)); (a,) = K._first_

˓→ngens(1)
>>> G = K.class_group(); G
Class group of order 3 with structure C3 of
Number Field in a with defining polynomial x^2 + 23
>>> G.category()
Category of finite enumerated commutative groups

Note the distinction between abstract generators, their ideal, and exponents:

sage: C = NumberField(x^2 + 120071, 'a').class_group(); C
Class group of order 500 with structure C250 x C2
of Number Field in a with defining polynomial x^2 + 120071
sage: c = C.gen(0)
sage: c # random
Fractional ideal class (5, 1/2*a + 3/2)
sage: c.ideal() # random
Fractional ideal (5, 1/2*a + 3/2)
sage: c.ideal() is c.value() # alias
True
sage: c.exponents()
(1, 0)

>>> from sage.all import *
>>> C = NumberField(x**Integer(2) + Integer(120071), 'a').class_group(); C
Class group of order 500 with structure C250 x C2
of Number Field in a with defining polynomial x^2 + 120071
>>> c = C.gen(Integer(0))
>>> c # random
Fractional ideal class (5, 1/2*a + 3/2)
>>> c.ideal() # random
Fractional ideal (5, 1/2*a + 3/2)
>>> c.ideal() is c.value() # alias
True
>>> c.exponents()
(1, 0)

Element
alias of FractionalIdealClass
gens_ideals()

  Return generating ideals for the (S-)class group.

  This is an alias for gens_values().

OUTPUT:
A tuple of ideals, one for each abstract Abelian group generator.

EXAMPLES:
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 + 23)
sage: K.class_group().gens_ideals()  # random gens (platform dependent)
(Fractional ideal (2, 1/4*a^3 - 1/4*a^2 + 1/4*a - 1/4),)

sage: C = NumberField(x^2 + x + 23899, 'a').class_group(); C
Class group of order 68 with structure C34 x C2 of Number Field
in a with defining polynomial x^2 + x + 23899
sage: C.gens()
(Fractional ideal class (7, a + 5), Fractional ideal class (5, a + 3))
sage: C.gens_ideals()
(Fractional ideal (7, a + 5), Fractional ideal (5, a + 3))

number_field()
Return the number field that this (S-)class group is attached to.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: C = NumberField(x^2 + 23, 'w').class_group(); C
Class group of order 3 with structure C3 of Number Field in w with defining polynomial x^2 + 23
sage: C.number_field()
Number Field in w with defining polynomial x^2 + 23
sage: K.<a> = QuadraticField(-14)
sage: CS = K.S_class_group(K.primes_above(2))
sage: CS.number_field()
Number Field in a with defining polynomial x^2 + 14 with a = 3 + 74165738677394274*I

>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> C = NumberField(x^2 + Integer(23), 'w').class_group(); C
Class group of order 3 with structure C3 of Number Field in w with defining polynomial x^2 + 23
>>> C.number_field()
Number Field in w with defining polynomial x^2 + 23
>>> K = QuadraticField(-Integer(14), names=('a',)); (a,) = K._first_ngens(1)
>>> CS = K.S_class_group(K.primes_above(Integer(2)))
>>> CS.number_field()

(continues on next page)
Number Field in \(a\) with defining polynomial \(x^2 + 14\) with \(a = 3\):

\[
\text{741657386773942*I}
\]

class \texttt{sage.rings.number_field.class_group.FractionalIdealClass}\((parent, element,\)

\texttt{ideal=None)\}

Bases: \texttt{AbelianGroupWithValuesElement}\n
A fractional ideal class in a number field.

EXAMPLES:

\begin{verbatim}
sage: x = polygen(ZZ, 'x')
sage: G = NumberField(x**2 + 23,'a').class_group(); G
Class group of order 3 with structure C3 of Number Field in a with defining polynomial x^2 + 23
class: sage.rings.number_field.class_group.FractionalIdealClass

sage: I = G.0; I
Fractional ideal class (2, 1/2*a - 1/2)
sage: I.ideal()
Fractional ideal (2, 1/2*a - 1/2)
sage: K.<w> = QuadraticField(-23)
sage: OK = K.ring_of_integers()
sage: C = OK.class_group()
sage: P2a, P2b = [P for P, e in (2*K).factor()]
sage: c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.gens()[(2, 1/2*w - 1/2)]
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> G = NumberField(x**2 + 23,'a').class_group(); G
Class group of order 3 with structure C3 of Number Field in a with defining polynomial x^2 + 23
>>> I = G.gen(0); I
Fractional ideal class (2, 1/2*a - 1/2)
>>> I.ideal()
Fractional ideal (2, 1/2*a - 1/2)
>>> K = QuadraticField(-Integer(23), names=('w',)); (w,) = K._first_ngens(1)
>>> OK = K.ring_of_integers()
>>> C = OK.class_group()
>>> P2a, P2b = [P for P, e in (Integer(2)*K).factor()]
>>> c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
>>> c.gens()[(2, 1/2*w - 1/2)]
\end{verbatim}

gens()\n
Return generators for a representative ideal in this \((S-)\)ideal class.

EXAMPLES:

\begin{verbatim}
sage: K.<w> = QuadraticField(-23)
sage: OK = K.ring_of_integers()
sage: C = OK.class_group()
\end{verbatim}
sage: P2a, P2b = [P for P, e in (2*K).factor()]
sage: c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.gens()
(2, 1/2*w - 1/2)

>>> from sage.all import *
>>>
K = QuadraticField(-Integer(23), names=('w',)); (w,) = K._first_ngens(1)
>>> OK = K.ring_of_integers()
>>> C = OK.class_group()
>>> P2a, P2b = [P for P, e in (Integer(2)*K).factor()]
>>> c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
>>> c.gens()
(2, 1/2*w - 1/2)

ideal()

Return a representative ideal in this ideal class.

EXAMPLES:

sage: K.<w> = QuadraticField(-23)
sage: OK = K.ring_of_integers()
sage: C = OK.class_group()
sage: P2a, P2b = [P for P, e in (2*K).factor()]
sage: c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.ideal()
Fractional ideal (2, 1/2*w - 1/2)

inverse()

Return the multiplicative inverse of this ideal class.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 3*x + 8); G = K.class_group()
sage: G(2, a).inverse()
Fractional ideal class (2, a^2 + 2*a - 1)
sage: ~G(2, a)
Fractional ideal class (2, a^2 + 2*a - 1)

>>> from sage.all import *
>>>
x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3)*x + Integer(8), names=('a',));
(continues on next page)
\[\text{is\_principal}()\]

Return True iff this ideal class is the trivial (principal) class.

EXAMPLES:

\[
\text{sage: } K.<w> = \text{QuadraticField}(-23)\\
\text{sage: } OK = K.\text{ring\_of\_integers}()\\
\text{sage: } C = OK.\text{class\_group}()\\
\text{sage: } P2a, P2b = \left[ P \text{ for } P, e \text{ in } (2*K).\text{factor}() \right]\\
\text{sage: } c = C(P2a)\\
\text{sage: } c.\text{is\_principal}()\\
\text{False}\\
\text{sage: } (c^2).\text{is\_principal}()\\
\text{False}\\
\text{sage: } (c^3).\text{is\_principal}()\\
\text{True}
\]

\[
\text{reduce}()\\
\text{Return representative for this ideal class that has been reduced using PARI's pari:idealred.}
\]

EXAMPLES:

\[
\text{sage: } x = \text{polygen}(\mathbb{Z}, 'x')\\
\text{sage: } k.<a> = \text{NumberField}(x^2 + 20072); G = k.\text{class\_group}()\\
\text{Class group of order 76 with structure C38 x C2 of}\\
\text{Number Field in a with defining polynomial } x^2 + 20072\\
\text{sage: } I = \left( G.0 \right)^{11}; I\\
\text{Fractional ideal class } (33, 1/2*a + 8)\\
\text{sage: } J = G(I.\text{ideal()}^5); J\\
\text{Fractional ideal class } (39135393, 1/2*a + 13654253)\\
\text{sage: } J.\text{reduce}()\\
\text{Fractional ideal class } (73, 1/2*a + 47)\\
\text{sage: } J == I^5\\
\text{True}
\]

\[
\text{from sage.all import *}\\
\text{K = QuadraticField(-Integer(23), names=('w',)); } (w,) = K._\text{first\_ngens}(1)\\
\text{OK = K.\text{ring\_of\_integers}()}\\
\text{P2a, P2b = } \left[ P \text{ for } P, e \text{ in } (2*K).\text{factor}() \right]\\
\text{c = C(P2a)}\\
\text{c.\text{is\_principal}()}\\
\text{False}\\
\text{(c**Integer(2)).\text{is\_principal}()}\\
\text{False}\\
\text{(c**Integer(3)).\text{is\_principal}()}\\
\text{True}
\]
representative_prime \((\text{norm\_bound}=1000)\)

Return a prime ideal in this ideal class.

INPUT:

- \(\text{norm\_bound}\) – (positive integer) upper bound on the norm of primes tested.

EXAMPLES:

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 31)
sage: K.class_number()
3
sage: Cl = K.class_group()
sage: [c.representative_prime() for c in Cl]
[Fractional ideal (3),
 Fractional ideal (2, 1/2*a + 1/2),
 Fractional ideal (2, 1/2*a - 1/2)]
```

```python
sage: K.<a> = NumberField(x^2 + 223)
sage: K.class_number()
7
sage: Cl = K.class_group()
sage: [c.representative_prime() for c in Cl]
[Fractional ideal (3),
 Fractional ideal (2, 1/2*a + 1/2),
 Fractional ideal (17, 1/2*a + 7/2),
 Fractional ideal (7, 1/2*a - 1/2),
 Fractional ideal (7, 1/2*a + 1),
 Fractional ideal (17, 1/2*a + 27/2),
 Fractional ideal (2, 1/2*a - 1/2)]
```

```python
>>> from sage.all import *
```
\[
\texttt{>>> K = NumberField(x**Integer(2) + Integer(223), names=('a',)); (a,) = K._first_ngens(1)}
\]

\[
\texttt{>>> K.class_number()}
7
\]

\[
\texttt{>>> Cl = K.class_group()}
\]

\[
\texttt{[c.representative_prime() for c in Cl]}
[\text{Fractional ideal (3,)}, \text{Fractional ideal (2, 1/2*a + 1/2,)}, \text{Fractional ideal (17, 1/2*a + 7/2,)}, \text{Fractional ideal (7, 1/2*a - 1/2,)}, \text{Fractional ideal (7, 1/2*a + 1/2,)}, \text{Fractional ideal (17, 1/2*a + 27/2,)}, \text{Fractional ideal (2, 1/2*a - 1/2,)}]
\]

\[
\text{class sage.rings.number_field.class_group.SClassGroup(gens_orders, names, number_field, gens, S, proof=True)}
\]

Bases: \texttt{ClassGroup}

The \(S\)-class group of a number field.

\textbf{EXAMPLES:}

\[
\texttt{sage: K.<a> = QuadraticField(-14)}
\]

\[
\texttt{sage: S = K.primes_above(2)}
\]

\[
\texttt{sage: K.S_class_group(S).gens()} \# \text{random gens (platform dependent)}
(\text{Fractional S-ideal class (3, a + 2,)},)
\]

\[
\texttt{sage: K.<a> = QuadraticField(-974)}
\]

\[
\texttt{sage: CS = K.S_class_group(K.primes_above(2)); CS} \text{S-class group of order 18 with structure C6 x C3 of Number Field in a with defining polynomial x^2 + 974 with a = 31.20897306865447?} \rightarrow\text{I}
\]

\[
\texttt{sage: CS.gen(0)} \# \text{random}
\text{Fractional S-ideal class (3, a + 2)}
\]

\[
\texttt{sage: CS.gen(1)} \# \text{random}
\text{Fractional S-ideal class (31, a + 24)}
\]

\[
\texttt{Element} \quad \text{alias of SFractionalIdealClass}
\]

\[
\text{S()}
\]
Return the set (or rather tuple) of primes used to define this class group.

EXAMPLES:

```sage
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2, a)
sage: S = (I,)
sage: CS = K.S_class_group(S);CS
S-class group of order 2 with structure C2 of
Number Field in a with defining polynomial x^2 + 14 with a = 3.
    →741657386773942?I
sage: T = tuple()
sage: CT = K.S_class_group(T);CT
S-class group of order 4 with structure C4 of
Number Field in a with defining polynomial x^2 + 14 with a = 3.
    →741657386773942?I
sage: CS.S()
(Fractional ideal (2, a),)
sage: CT.S()
()
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(14), names=('a',)); (a,) = K._first_ngens(1)
>>> I = K.ideal(Integer(2), a)
>>> S = (I,)
>>> CS = K.S_class_group(S);CS
S-class group of order 2 with structure C2 of
Number Field in a with defining polynomial x^2 + 14 with a = 3.
    →741657386773942?I
>>> T = tuple()
>>> CT = K.S_class_group(T);CT
S-class group of order 4 with structure C4 of
Number Field in a with defining polynomial x^2 + 14 with a = 3.
    →741657386773942?I
>>> CS.S()
(Fractional ideal (2, a),)
>>> CT.S()
()
```

class sage.rings.number_field.class_group.SFractionalIdealClass(parent, element, ideal=None)

Bases: FractionalIdealClass

An $S$-fractional ideal class in a number field for a tuple $S$ of primes.

EXAMPLES:

```sage
sage: K.<a> = QuadraticField(-14)
sage: I = K.ideal(2, a)
sage: S = (I,)
sage: CS = K.S_class_group(S)
sage: J = K.ideal(7, a)
sage: G = K.ideal(3, a + 1)
sage: CS(I)
Trivial $S$-ideal class
sage: CS(J)
Trivial $S$-ideal class
```

(continues on next page)
sage: CS(G)
Fractional S-ideal class (3, a + 1)

```python
>>> from sage.all import *

>>> K = QuadraticField(-Integer(14), names=('a',)); (a,) = K._first_ngens(1)
>>> I = K.ideal(Integer(2), a)
>>> S = (I,)
>>> CS = K.S_class_group(S)
>>> J = K.ideal(Integer(7), a)
>>> G = K.ideal(Integer(3), a + Integer(1))
>>> CS(I)
Trivial S-ideal class
>>> CS(J)
Trivial S-ideal class
>>> CS(G)
Fractional S-ideal class (3, a + 1)
```

sage: K.<a> = QuadraticField(-14)

sage: I = K.ideal(2, a)

sage: S = (I,)

sage: CS = K.S_class_group(S)

sage: J = K.ideal(7, a)

sage: G = K.ideal(3, a + 1)

sage: CS(I).ideal()
Fractional ideal (2, a)

sage: CS(J).ideal()
Fractional ideal (7, a)

sage: CS(G).ideal()
Fractional ideal (3, a + 1)

```python
>>> from sage.all import *

>>> K = QuadraticField(-Integer(14), names=('a',)); (a,) = K._first_ngens(1)

>>> I = K.ideal(Integer(2), a)

>>> S = (I,)

>>> CS = K.S_class_group(S)

>>> J = K.ideal(Integer(7), a)

>>> G = K.ideal(Integer(3), a + Integer(1))

>>> CS(I).ideal()
Fractional ideal (2, a)

>>> CS(J).ideal()
Fractional ideal (7, a)

>>> CS(G).ideal()
Fractional ideal (3, a + 1)
```

sage: K.<a> = QuadraticField(-14)

sage: I = K.ideal(2, a)

sage: S = (I,)

sage: CS = K.S_class_group(S)

sage: G = K.ideal(3, a + 1)

sage: CS(G).inverse()
Fractional S-ideal class (3, a + 2)

```python
>>> from sage.all import *

>>> K = QuadraticField(-Integer(14), names=('a',)); (a,) = K._first_ngens(1)

>>> I = K.ideal(Integer(2), a)
```

(continues on next page)
4.6 Units and $S$-unit groups of number fields

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 - 8*x^2 + 36)
sage: UK = UnitGroup(K); UK
Unit group with structure C4 x Z of
Number Field in a with defining polynomial x^4 - 8*x^2 + 36
```

The first generator is a primitive root of unity in the field:

```
sage: UK.gens()
(u0, u1)
sage: UK.gens_values() # random
[-1/12*a^3 + 1/6*a, 1/24*a^3 + 1/4*a^2 - 1/12*a - 1]
sage: UK.gen(0).value()
1/12*a^3 - 1/6*a
sage: UK.gen(0) + K.one() # coerce abstract generator into number field
1/12*a^3 - 1/6*a + 1
```

```
Units in the field can be converted into elements of the unit group represented as elements of an abstract multiplicative group:

```
sage: UK(1)
1
sage: UK(-1)

u0^2
sage: [UK(u) for u in (x^4 - 1).roots(K, multiplicities=False)]

[1, u0^2, u0, u0^3]
```

Exp and log functions provide maps between units as field elements and exponent vectors with respect to the generators:

```
sage: u = UK.exp([13,10]); u

# random
-41/8*a^3 - 55/4*a^2 + 41/4*a + 55
```
sage: UK.log(u)
(1, 10)
sage: u = UK.fundamental_units()[0]
sage: [UK.log(u^k) == (0,k) for k in range(10)]
[True, True, True, True, True, True, True, True, True, True]
sage: all(UK.log(u^k) == (0,k) for k in range(10))
True

sage: K.<a> = NumberField(x^5 - 2, 'a')
sage: UK = UnitGroup(K)
sage: UK.rank()
2
sage: UK.fundamental_units()
[a^3 + a^2 - 1, a - 1]

S-unit groups may be constructed, where S is a set of primes:

sage: K.<a> = NumberField(x^6 + 2)
sage: S = K.ideal(3).prime_factors(); S
[Fractional ideal (3, a + 1), Fractional ideal (3, a - 1)]
sage: SUK = UnitGroup(K,S=tuple(S)); SUK
S-unit group with structure C2 x Z x Z x Z x Z of
Number Field in a with defining polynomial x^6 + 2
with S = (Fractional ideal (3, a + 1), Fractional ideal (3, a - 1))
sage: SUK.primes()
(Fractional ideal (3, a + 1), Fractional ideal (3, a - 1))
sage: SUK.rank()
4
sage: SUK.gens_values()
[-1, a^2 + 1, -a^5 - a^4 + a^2 + a + 1, a + 1, a - 1]
sage: u = 9*prod(SUK.gens_values()); u
-18*a^5 - 18*a^4 - 18*a^3 - 9*a^2 + 9*a + 27
sage: SUK.log(u)
(1, 3, 1, 7, 7)
sage: u == SUK.exp((1,3,1,7,7))
True

4.6. Units and S-unit groups of number fields
A relative number field example:

```python
sage: L.<a, b> = NumberField([x^2 + x + 1, x^4 + 1])
sage: UL = L.unit_group(); UL
Unit group with structure C24 x Z x Z x Z of
Number Field in a with defining polynomial x^2 + x + 1 over its base field
sage: UL.gens_values() # random
[-b^3*a - b^3, -b^3*a + b, (-b^3 - b^2 - b)*a - b - 1, (-b^3 - 1)*a - b^2 + b - 1]
sage: UL.zeta_order()
24
sage: UL.roots_of_unity()
[-b*a,  
  -b^2*a - b^2,  
  -b^3,  
  -a,  
  -b*a - b,  
  -b^2,  
  b^3*a,  
  -a - 1,  
  -b,  
  b^2*a,  
  b^3*a + b^3,  
  -1,  
  b*a,  
  b^2*a + b^2,  
  b^3,  
  a,  
  b*a + b,  
  b^2,  
  -b^3*a,  
  a + 1,  
  b,  
  -b^2*a,  
  -b^3*a - b^3,  
  1]
```
A relative extension example, which worked thanks to the code review by F.W.Clarke:

```
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: K.unit_group()
Unit group with structure C2 x Z x Z x Z x Z x Z x Z x Z of Number Field in c
with defining polynomial Y^2 + (-2*b - 3)*a - 2*b - 6 over its base field
```
The unit group or an $S$-unit group of a number field.

**exp**(exponents)

Return unit with given exponents with respect to group generators.

**INPUT:**

- $u$ – Any object from which an element of the unit group’s number field $K$ may be constructed; an error is raised if an element of $K$ cannot be constructed from $u$, or if the element constructed is not a unit.

**OUTPUT:** a list of integers giving the exponents of $u$ with respect to the unit group’s basis.

**EXAMPLES:**

```python
sage: x = polygen(QQ)
sage: K.<z> = CyclotomicField(13)
sage: UK = UnitGroup(K)
sage: [UK.log(u) for u in UK.gens()]
[(1, 0, 0, 0, 0, 0),
 (0, 1, 0, 0, 0, 0),
 (0, 0, 1, 0, 0, 0),
 (0, 0, 0, 1, 0, 0),
 (0, 0, 0, 0, 1, 0),
 (0, 0, 0, 0, 0, 1)]
sage: vec = [65,6,7,8,9,10]
sage: unit = UK.exp(vec)
sage: UK.log(unit)
(13, 6, 7, 8, 9, 10)
sage: u = UK.gens()[-1]
sage: UK.exp(UK.log(u)) == u.value()
True
```

```python
>>> from sage.all import *
>>> x = polygen(QQ)
>>> K = CyclotomicField(Integer(13), names=('z',)); (z,) = K._first_ngens(1)
>>> UK = UnitGroup(K)
>>> [UK.log(u) for u in UK.gens()]
[(1, 0, 0, 0, 0, 0),
 (0, 1, 0, 0, 0, 0),
 (0, 0, 1, 0, 0, 0),
 (0, 0, 0, 1, 0, 0),
 (0, 0, 0, 0, 1, 0),
 (0, 0, 0, 0, 0, 1)]
>>> vec = [Integer(65),Integer(6),Integer(7),Integer(8),Integer(9),
        Integer(10)]
>>> unit = UK.exp(vec)
>>> UK.log(unit)
(13, 6, 7, 8, 9, 10)
>>> u = UK.gens()[-Integer(1)]
>>> UK.exp(UK.log(u)) == u.value()
True
```

An $S$-unit example:
sage: SUK = UnitGroup(K,S=2)
sage: v = (3,1,4,1,5,9,2)
sage: u = SUK.exp(v); u
8732*z^11 - 15496*z^10 - 51840*z^9 - 68804*z^8 - 51840*z^7 - 15496*z^6
+ 8732*z^5 - 34216*z^3 - 64312*z^2 - 64312*z - 34216
sage: SUK.log(u)
(3, 1, 4, 1, 5, 9, 2)
sage: SUK.log(u) == v
True

>>> from sage.all import *
>>> SUK = UnitGroup(K,S=Integer(2))
>>> v = (Integer(3),Integer(1),Integer(4),Integer(1),Integer(5),Integer(9),
     →Integer(2))
>>> u = SUK.exp(v); u
8732*z^11 - 15496*z^10 - 51840*z^9 - 68804*z^8 - 51840*z^7 - 15496*z^6
+ 8732*z^5 - 34216*z^3 - 64312*z^2 - 64312*z - 34216
>>> SUK.log(u)
(3, 1, 4, 1, 5, 9, 2)
>>> SUK.log(u) == v
True

fundamental_units()

Return generators for the free part of the unit group, as a list.

EXAMPLES:

sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 + 23)
sage: U = UnitGroup(K)
sage: U.fundamental_units()  # random
[1/4*a^3 - 7/4*a^2 + 17/4*a - 19/4]

log(u)

Return the exponents of the unit u with respect to group generators.

INPUT:

• u – Any object from which an element of the unit group’s number field K may be constructed; an error is raised if an element of K cannot be constructed from u, or if the element constructed is not a unit.

OUTPUT: a list of integers giving the exponents of u with respect to the unit group’s basis.

EXAMPLES:

sage: x = polygen(QQ)
sage: K.<z> = CyclotomicField(13)
sage: UK = UnitGroup(K)
sage: [UK.log(u) for u in UK.gens()]

(continues on next page)
(0, 1, 0, 0, 0, 0),
(0, 0, 1, 0, 0, 0),
(0, 0, 0, 1, 0, 0),
(0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 1)]
\texttt{sage: vec = [65,6,7,8,9,10]}
\texttt{sage: unit = UK.exp(vec); unit \# random}
\begin{align*}
-253576z^{11} &+ 7003z^{10} - 395532z^9 - 35275z^8 - 500326z^7 - 35275z^6 \\
- 395532z^5 &+ 7003z^4 - 253576z^3 - 59925z - 59925
\end{align*}
\texttt{sage: UK.log(unit)}
\begin{align*}
(13, &6, 7, 8, 9, 10)
\end{align*}

An S-unit example:

\texttt{sage: SUK = UnitGroup(K, S=2)}
\texttt{sage: v = (3,1,4,1,5,9,2)}
\texttt{sage: u = SUK.exp(v); u}
\begin{align*}
8732z^{11} &- 15496z^{10} - 51840z^9 - 68804z^8 - 51840z^7 - 15496z^6 \\
+ 8732z^5 &- 34216z^3 - 64312z^2 - 64312z - 34216
\end{align*}
\texttt{sage: SUK.log(u)}
\begin{align*}
(3, &1, 4, 1, 5, 9, 2)
\end{align*}
\texttt{sage: SUK.log(u) == v}
\texttt{True}

\texttt{from sage.all import *}
\texttt{x = polygen(QQ)}
\texttt{K = CyclotomicField(Integer(13), names=('z',)); (z,) = K._first_ngens(1)}
\texttt{UK = UnitGroup(K)}
\texttt{[UK.log(u) for u in UK.gens()]}
Return the number field associated with this unit group.

EXAMPLES:

```sage
sage: U = UnitGroup(QuadraticField(-23, 'w')); U
Unit group with structure C2 of
Number Field in w with defining polynomial x^2 + 23 with w = 4.
->795831523312720*I
sage: U.number_field()
Number Field in w with defining polynomial x^2 + 23 with w = 4.
->795831523312720*I
```

primes()

Return the (possibly empty) list of primes associated with this S-unit group.

EXAMPLES:

```sage
sage: K.<a> = QuadraticField(-23)
sage: S = tuple(K.ideal(3).prime_factors()); S
(Fractional ideal (3, 1/2*a - 1/2), Fractional ideal (3, 1/2*a + 1/2))
sage: U = UnitGroup(K,S=tuple(S)); U
S-unit group with structure C2 x Z x Z of
Number Field in a with defining polynomial x^2 + 23 with a = 4.
->795831523312720*I
with S = (Fractional ideal (3, 1/2*a - 1/2), Fractional ideal (3, 1/2*a + 1/2))
sage: U.primes() == S
True
```

rank()

Return the rank of the unit group.

EXAMPLES:

```sage
sage: K.<z> = CyclotomicField(13)
sage: UnitGroup(K).rank()
```

(continues on next page)
roots_of_unity()
Return all the roots of unity in this unit group, primitive or not.

EXAMPLES:

```python
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x**4 - x**2 + 1)
sage: U = UnitGroup(K)
sage: U.torsion_generator()
u0
sage: U.torsion_generator().value() # random
-1/4*a^3 - 1/4*a + 1/2
```
zeta \((n=2, \text{all}=False)\)
Return one, or a list of all, primitive \(n\)-th root of unity in this unit group.

EXAMPLES:

```python
sage: x = polygen(QQ)
sage: K.<z> = NumberField(x^2 + 3)
sage: U = UnitGroup(K)
sage: U.zeta(1)
1
sage: U.zeta(2)
-1
sage: U.zeta(2, all=True)
[-1]
sage: U.zeta(3)
-1/2*z - 1/2
sage: U.zeta(3, all=True)
[-1/2*z - 1/2, 1/2*z - 1/2]
sage: U.zeta(4)
Traceback (most recent call last):
...  
ValueError: n (=4) does not divide order of generator
sage: r.<x> = QQ[]
sage: K.<b> = NumberField(x^2 + 1)
sage: U = UnitGroup(K)
sage: U.zeta(4)
b
sage: U.zeta(4, all=True)
[b, -b]
sage: U.zeta(3)
Traceback (most recent call last):
...  
ValueError: n (=3) does not divide order of generator
sage: U.zeta(3, all=True)
[]
```

```python
>>> from sage.all import *

>>> x = polygen(QQ)

>>> K = NumberField(x**Integer(2) + Integer(3), names=('z',)); (z,) = K._
˓→first_ngens(1)

>>> U = UnitGroup(K)

>>> U.zeta(Integer(1))
1

>>> U.zeta(Integer(2))
-1

>>> U.zeta(Integer(2), all=True)
[-1]

>>> U.zeta(Integer(3))
-1/2*z - 1/2

>>> U.zeta(Integer(3), all=True)
[-1/2*z - 1/2, 1/2*z - 1/2]

>>> U.zeta(Integer(4))
Traceback (most recent call last):
...

ValueError: n (=4) does not divide order of generator

>>> r = QQ['x']; (x,) = r._first_ngens(1)
```
K = NumberField(x**2 + 1, names=('b',)); (b,) = K._first_ngens(1)
U = UnitGroup(K)
U.zeta(Integer(4))
b
U.zeta(Integer(4), all=True)
[b, -b]
U.zeta(Integer(3))
Traceback (most recent call last):
  ...  ValueError: n (=3) does not divide order of generator
U.zeta(Integer(3), all=True)
[]

zeta_order()
Returns the order of the torsion part of the unit group.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 - x^2 + 4)
sage: U = UnitGroup(K)
sage: U.zeta_order()
6
```

```
from sage.all import *

>>> x = polygen(QQ)

>>> K = NumberField(x^4 - x^2 + 4, names=('a',));
˓→ (a,) = K._first_ngens(1)

>>> U = UnitGroup(K)

>>> U.zeta_order()
6
```

### 4.7 Solver for the \(S\)-unit equation \(x + y = 1\)

Inspired by works of Tzanakis–de Weger, Baker–Wustholz and Smart, we use the LLL methods to implement an algorithm that returns all \(S\)-unit solutions to the equation \(x + y = 1\).

EXAMPLES:

```
sage: from sage.rings.number_field.S_unit_solver import solve_S_unit_equation, eq_up_ ˓→to_order

sage: x = polygen(ZZ, 'x')
sage: K.<xi> = NumberField(x^2 + x + 1)
sage: S = K.primes_above(3)
sage: expected = [((0, 1), (4, 0), xi + 2, -xi - 1),
               ((1, -1), (0, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
               ((1, 0), (5, 0), xi + 1, -xi),
               ((2, 0), (5, 1), xi, -xi + 1)]

sage: sols = solve_S_unit_equation(K, S, 200)
sage: eq_up_to_order(sols, expected)
True
```
Todo:

- Use Cython to improve timings on the sieve

REFERENCES:

- [MR2016]
- [Sma1995]
- [Sma1998]
- [Yu2007]
- [AKMRVW]

AUTHORS:

- Alejandra Alvarado, Angelos Koutsianas, Beth Malmskog, Christopher Rasmussen, David Roe, Christelle Vincent, Mckenzie West (2018-04-25 to 2018-11-09): original version

sage.rings.number_field.S_unit_solver.K0_func(SUK, A, prec=106)

Return the constant $K_0$ from [AKMRVW].

INPUT:

- SUK – a group of $S$-units
- A – the set of the products of the coefficients of the $S$-unit equation with each root of unity of $K$
- prec – the precision of the real field (default: 106)

OUTPUT:

The constant $K_0$, a real number.

EXAMPLES:

continues on next page
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(6)))
sage: v = K.primes_above(3)[0]
sage: K0_func(SUK, K.roots_of_unity())
8.84763586062272e12

REFERENCES:
• [Sma1995] p. 824
• [AKMRVW] arXiv 1903.00977

sage.rings.number_field.S_unit_solver.K1_func(SUK, v, A, prec=106)
Return the constant $K_1$ from Smart’s TCDF paper, [Sma1995].

INPUT:
• SUK – a group of $S$-units
• $v$ – an infinite place of $K$ (element of SUK.number_field().places(prec))
• $A$ – a list of all products of each potential $a, b$ in the $S$-unit equation $ax + by + 1 = 0$ with each root of unity of $K$
• prec – the precision of the real field (default: 106)

OUTPUT:
The constant $K_1$, a real number

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import K1_func
generate code output

sage: from sage.rings.number_field.S_unit_solver import K1_func
generate code output

(continues on next page)
ngens(1)
>>> SUK = UnitGroup(K,S=tuple(K.primes_above(Integer(3))))
>>> phi_real = K.places()[Integer(0)]
>>> phi_complex = K.places()[Integer(1)]
>>> A = K.roots_of_unity()

>>> K1_func(SUK, phi_real, A)
4.483038368145048508970350163578e16

>>> K1_func(SUK, phi_complex, A)
2.073346189067285101984136298965e17

REFERENCES:
• [Sma1995] p. 825

\texttt{sage.rings.number_field.S_unit_solver.Omega_prime}(dK, v, mu_list, prec=106)

Return the constant $\Omega'$ appearing in [AKMRVW].

INPUT:
• $dK$ – the degree of a number field $K$
• $v$ – a finite place of $K$
• $\mu_list$ – a list of nonzero elements of $K$. It is assumed that the sublist $\mu_list[1:]$ is multiplicatively independent.
• $\text{prec}$ – the precision of the real field

OUTPUT:
The constant $\Omega'$.

EXAMPLES:

\begin{verbatim}
sage: from sage.rings.number_field.S_unit_solver import mus, Omega_prime
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 3)
sage: SUK = UnitGroup(K,S=tuple(K.primes_above(6)))
sage: v = K.primes_above(3)[0]
sage: mu_list = [-1] + mus(SUK,v)
sage: dK = K.degree()
sage: Omega_prime(dK, v, mu_list)
0.000487349679922696
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import mus, Omega_prime
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> SUK = UnitGroup(K,S=tuple(K.primes_above(Integer(6))))
>>> v = K.primes_above(Integer(3))[Integer(0)]
>>> mu_list = [-Integer(1)] + mus(SUK,v)
>>> dK = K.degree()
>>> Omega_prime(dK, v, mu_list)
0.000487349679922696
\end{verbatim}

REFERENCES:
• [AKMRVW] arXiv 1903.00977

4.7. Solver for the $S$-unit equation $x + y = 1$ 551
sage.rings.number_field.S_unit_solver.Yu_C1_star\(n, v, prec=106\)

Return the constant \(C_1^*\) appearing in [Yu2007] (1.23).

INPUT:

- \(n\) – the number of generators of a multiplicative subgroup of a field \(K\)
- \(v\) – a finite place of \(K\) (a fractional ideal)
- \(prec\) – the precision of the real field

OUTPUT:

The constant \(C_1^*\) as a real number.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 5)
sage: v11 = K.primes_above(11)[0]
sage: from sage.rings.number_field.S_unit_solver import Yu_C1_star
sage: Yu_C1_star(1,v11)
2.154667761574516556114215527020e6

from sage.all import *

>>> from sage.rings.number_field.S_unit_solver import Yu_C1_star
>>> Yu_C1_star(Integer(1),v11)
2.154667761574516556114215527020e6

REFERENCES:

- [Yu2007] p.189,193

sage.rings.number_field.S_unit_solver.Yu_al_kappa1_c1\(p, dK, ep\)

Compute the constants a(1), kappa1, and c(1) of [Yu2007].

INPUT:

- \(p\) – a rational prime number
- \(dK\) – the absolute degree of some number field \(K\)
- \(ep\) – the absolute ramification index of some prime \(\mathfrak{p}\) of \(K\) lying above \(p\)

OUTPUT:

The constants a(1), kappa1, and c(1).

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import Yu_al_kappa1_c1
sage: Yu_al_kappa1_c1(5, 10, 3)
(16, 20, 319)

from sage.all import *

>>> from sage.rings.number_field.S_unit_solver import Yu_al_kappa1_c1
>>> Yu_al_kappa1_c1(Integer(5), Integer(10), Integer(3))
(16, 20, 319)
REFERENCES:

- [Yu2007]

`sage.rings.number_field.S_unit_solver.Yu_bound(SUK, v, prec=106)`

Return $c_8$ such that $c_8 \geq \exp(2)/\log(2)$ and $\text{ord}_p(\Theta - 1) < c_8 \log B$, where $\Theta = \prod_{j=1}^{n} a_j b_j$ and $B \geq \max_j |b_j|$ and $B \geq 3$.

INPUT:

- `SUk` – a group of $S$-units
- `v` – a finite place of $K$ (a fractional ideal)
- `prec` – the precision of the real field

OUTPUT:

The constant $c_8$ as a real number.

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import Yu_bound
sage: x = polygen(ZZ, 'x')

sage: K.<a> = NumberField(x^2 + 11)

sage: SUK = UnitGroup(K, S=tuple(K.primes_above(6)))

sage: v = K.primes_above(3)[0]

sage: Yu_bound(SUK, v)
9.03984381033128e9
```

REFERENCES:

- [Sma1995] p. 825
- [AKMRVW] arXiv 1903.00977

`sage.rings.number_field.S_unit_solver.Yu_condition_115(K, v)`

Return `True` or `False`, as the number field $K$ and the finite place $v$ satisfy condition (1.15) of [Yu2007].

INPUT:

- `K` – a number field
- `v` – a finite place of $K$

OUTPUT:

`True` if (1.15) is satisfied, otherwise `False`.

EXAMPLES:
sage: from sage.rings.number_field.S_unit_solver import Yu_condition_115
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 5)
sage: v2 = K.primes_above(2)[0]
sage: v11 = K.primes_above(11)[0]
sage: Yu_condition_115(K, v2)
False
sage: Yu_condition_115(K, v11)
True

REFERENCES:

• [Yu2007] p. 188

sage.rings.number_field.S_unit_solver.Yu_modified_height (mu, n, v, prec=106)
Return the value of $h(n)(\mu)$ as appearing in [Yu2007] equation (1.21).

INPUT:

• $\mu$ – an element of a field $K$
• $n$ – number of $\mu_j$ to be considered in Yu’s Theorem.
• $v$ – a place of $K$
• $\text{prec}$ – the precision of the real field

OUTPUT:
The value $h_p(\mu)$.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 + 5)
sage: v11 = K.primes_above(11)[0]
sage: from sage.rings.number_field.S_unit_solver import Yu_modified_height
sage: Yu_modified_height(a, 3, v11)
0.8047189562170501873003796666131

>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import Yu_condition_115
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x^Integer(2) + Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> v2 = K.primes_above(Integer(2))[Integer(0)]
>>> v11 = K.primes_above(Integer(11))[Integer(0)]
>>> Yu_condition_115(K, v2)
False
>>> Yu_condition_115(K, v11)
True

554 Chapter 4. Orders, Ideals and Ideal Classes
If \( \mu \) is a root of unity, the output is not zero. ::

\[
\text{sage: Yu_modified_height(-1, 3, v11) 0.03425564675426243634374205111379}
\]

REFERENCES:
- [Yu2007] p. 192

\[
sage.rings.number_field.S_unit_solver.beta_k(betas_and_ns)
\]

Return a pair \([\beta_k, |\beta_k|_v]\), where \(\beta_k\) has the smallest nonzero valuation in absolute value of the list \(\text{betas_and_ns}\).

INPUT:
- \(\text{betas_and_ns}\) – a list of pairs \([\beta, \text{val}_v(\beta)]\) outputted from the function where \(\beta\) is an element of \(\text{SU}.\text{fundamental_units()}\)

OUTPUT:
The pair \([\beta_k, v(\beta_k)]\), where \(\beta_k\) is an element of \(K\) and \(v(\beta_k)\) is a integer

EXAMPLES:

\[
\begin{align*}
\text{sage: from sage.rings.number_field.S_unit_solver import beta_k} \\
\text{sage: x = polygen(ZZ, 'x')} \\
\text{sage: K.<xi> = NumberField(x^3 - 3)} \\
\text{sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))} \\
\text{sage: v_fin = tuple(K.primes_above(3))[0]} \\
\text{sage: betas = [[beta, beta.valuation(v_fin)] for beta in SUK.fundamental_units()]} \\
\text{sage: beta_k(betas)} \\
[\xi, 1]
\end{align*}
\]

\[
\begin{align*}
>>> \text{from sage.all import *} \\
>>> \text{from sage.rings.number_field.S_unit_solver import beta_k} \\
>>> x = polygen(ZZ, 'x') \\
>>> K = NumberField(x**Integer(3) - Integer(3), names=('xi',)); (xi,) = K._first_ngens(1) \\
>>> SUK = UnitGroup(K, S=tuple(K.primes_above(Integer(3)))) \\
>>> v_fin = tuple(K.primes_above(Integer(3)))[Integer(0)] \\
>>> betas = [[beta, beta.valuation(v_fin)] for beta in SUK.fundamental_units()] \\
>>> beta_k(betas) \\
[\xi, 1]
\end{align*}
\]

REFERENCES:
- [Sma1995] pp. 824-825

\[
sage.rings.number_field.S_unit_solver.c11_func(SUK, v, A, prec=106)
\]

Return the constant \(c_{11}\) from Smart's TCDF paper, [Sma1995].

INPUT:
- \(\text{SU}K\) – a group of \(S\)-units
- \(v\) – a place of \(K\), finite (a fractional ideal) or infinite (element of \(\text{SU}.\text{number_field()}.\text{places}(\text{prec})\))
- \(A\) – the set of the product of the coefficients of the \(S\)-unit equation with each root of unity of \(K\)
- \(\text{pre}c\) – the precision of the real field (default: 106)

4.7. Solver for the \(S\)-unit equation \(x + y = 1\)
OUTPUT:

The constant $c_{11}$, a real number

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import c11_func
sage: x = polygen(ZZ, 'x')

sage: K.<xi> = NumberField(x**3 - 3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: phi_real = K.places()[0]
sage: phi_complex = K.places()[1]
sage: A = K.roots_of_unity()

sage: c11_func(SUK, phi_real, A)  # abs tol 1e-29
3.255848343572896153455615423662

sage: c11_func(SUK, phi_complex, A)  # abs tol 1e-29
6.511696687145792306911230847323
```

REFERENCES:

- [Sma1995] p. 825

Sage.rings.number_field.S_unit_solver.c13_func(SUK, v, prec=106)

Return the constant $c_{13}$ from Smart's TCDF paper, [Sma1995].

INPUT:

- SUK – a group of $S$-units
- v – an infinite place of $K$ (element of SUK.number_field().places(prec))
- prec – the precision of the real field (default: 106)

OUTPUT:

The constant $c_{13}$, as a real number

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import c13_func
sage: x = polygen(ZZ, 'x')

sage: K.<xi> = NumberField(x**3 - 3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
```

(continues on next page)
It is an error to input a finite place.

REFERENCES:

- [Sma1995] p. 825

sage.rings.number_field.S_unit_solver.c3_func(SUK, prec=106)

Return the constant $c_3$ from [AKMRVW].

INPUT:

- SUK – a group of $S$-units
- prec – the precision of the real field (default: 106)

OUTPUT:

The constant $c_3$, as a real number

EXAMPLES:
sage: from sage.rings.number_field.S_unit_solver import c3_func
sage: x = polygen(ZZ, 'x')
sage: K.<xi> = NumberField(x^3 - 3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))

sage: c3_func(SUK)  # abs tol 1e-29
0.4257859134798034746197327286726

>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import c3_func
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3), names=(xi,)); (xi,) = K._first_ngens(1)
>>> SUK = UnitGroup(K, S=tuple(K.primes_above(Integer(3))))

>>> c3_func(SUK)  # abs tol 1e-29
0.4257859134798034746197327286726

Note: The numerator should be as close to 1 as possible, especially as the rank of the $S$-units grows large

REFERENCES:

- [AKMRVW] arXiv 1903.00977

sage.rings.number_field.S_unit_solver.c4_func(SUK, v, A, prec=106)

Return the constant $c_4$ from Smart's TCDF paper, [Sma1995].

INPUT:

- SUK – a group of $S$-units
- v – a place of $K$, finite (a fractional ideal) or infinite (element of $SUK.number_field().places(prec)$)
- A – the set of the product of the coefficients of the $S$-unit equation with each root of unity of $K$
- prec – the precision of the real field (default: 106)

OUTPUT:

The constant $c_4$, as a real number

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import c4_func
sage: x = polygen(ZZ, 'x')
sage: K.<xi> = NumberField(x^3 - 3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))

sage: phi_real = K.places()[0]
sage: phi_complex = K.places()[1]

sage: v_fin = tuple(K.primes_above(3))[0]
sage: A = K.roots_of_unity()

sage: c4_func(SUK, phi_real, A)
1.0000000000000000000000000000000
sage: c4_func(SUK, phi_complex, A)
1.0000000000000000000000000000000

(continues on next page)
import * from sage.rings.number_field.S_unit_solver

x = polygen(ZZ, 'x')
K = NumberField(x**Integer(3) - Integer(3), names=('xi',)); (xi,) = K._first_ngens(1)
SUK = UnitGroup(K, S=tuple(K.primes_above(Integer(3))))
phi_real = K.places()[Integer(0)]
phi_complex = K.places()[Integer(1)]
v_fin = tuple(K.primes_above(Integer(3)))[Integer(0)]
A = K.roots_of_unity()
c4_func(SUK, phi_real, A)
1.000000000000000000000000000000
c4_func(SUK, phi_complex, A)
1.000000000000000000000000000000
c4_func(SUK, v_fin, A)
1.000000000000000000000000000000

REFERENCES:

• [Sma1995] p. 824

sage.rings.number_field.S_unit_solver.clean_rfv_dict(rfv_dictionary)

Given a residue field vector dictionary, remove some impossible keys and entries.

INPUT:

• rfv_dictionary – a dictionary whose keys are exponent vectors and whose values are residue field vectors

OUTPUT:

None. But it removes some keys from the input dictionary.

Note:

• The keys of a residue field vector dictionary are exponent vectors modulo \( q - 1 \) for some prime \( q \).

• The values are residue field vectors. It is known that a residue field vector which comes from a solution to the \( S \)-unit equation cannot have 1 in any entry.

EXAMPLES:

In this example, we use a truncated list generated when solving the \( S \)-unit equation in the case that \( K \) is defined by the polynomial \( x^2 + x + 1 \) and \( S \) consists of the primes above 3:

sage: from sage.rings.number_field.S_unit_solver import clean_rfv_dict
sage: rfv_dict = [(1, 3): [3, 2], (3, 0): [6, 6], (5, 4): [3, 6], (2, 1): [4, 6],
               (5, 1): [3, 1], (2, 5): [1, 5], (0, 3): [1, 6]}
sage: len(rfv_dict)
7

(continues on next page)

4.7. Solver for the \( S \)-unit equation \( x + y = 1 \)
sage: clean_rfv_dict(rfv_dict)
sage: len(rfv_dict)
4
sage: rfv_dict
{(1, 3): [3, 2], (2, 1): [4, 6], (3, 0): [6, 6], (5, 4): [3, 6]}

sage.rings.number_field.S_unit_solver.clean_sfs(sfs_list)

Given a list of $S$-unit equation solutions, remove trivial redundancies.

**INPUT:**

- sfs_list – a list of solutions to the $S$-unit equation

**OUTPUT:**

A list of solutions to the $S$-unit equation

**Note:** The function looks for cases where $x + y = 1$ and $y + x = 1$ appear as separate solutions, and removes one.

**EXAMPLES:**

The function is not dependent on the number field and removes redundancies in any list.

sage: from sage.rings.number_field.S_unit_solver import clean_sfs
sage: sols = [((1, 0, 0), (0, 0, 1), -1, 2), ((0, 0, 1), (1, 0, 0), 2, -1)]
sage: clean_sfs( sols )
[((1, 0, 0), (0, 0, 1), -1, 2)]

sage.rings.number_field.S_unit_solver.column_Log(SUK, iota, U, prec=106)

Return the log vector of iota; i.e., the logs of all the valuations.

**INPUT:**
• SUK – a group of $S$-units
• $\iota$ – an element of $K$
• $U$ – a list of places (finite or infinite) of $K$
• $\text{prec}$ – the precision of the real field (default: 106)

OUTPUT:
The log vector as a list of real numbers

EXAMPLES:

```sage
column_Log(SUK, xi^2, U)
```

# abs tol 1e-29

```
[1.464816384890812968648768625966, -2.197224577336219382790490473845]
```

REFERENCES:

• [Sma1995] p. 823

sage.rings.number_field.S_unit_solver.compatible_system_lift

Given a compatible system of exponent vectors and complementary exponent vectors, return a lift to the integers.

INPUT:

• compatible_system – a list of pairs $\{[v_0, w_0], [v_1, w_1], \ldots, [v_k, w_k]\}$ where $[v_i, w_i]$ is a pair of complementary exponent vectors modulo $q_i - 1$, and all pairs are compatible.

• split_primes_list – a list of primes modulo $q_i - 1$, for all $i$ and all $j > 0$

OUTPUT:

A pair of vectors $[v, w]$ satisfying:

1. $v[0] = \text{vi}[0]$ for all $i$
2. $w[0] = \text{wi}[0]$ for all $i$
3. $v[j] = \text{vi}[j]$ modulo $q_i - 1$ for all $i$ and all $j > 0$
4. $w[j] = \text{wi}[j]$ modulo $q_i - 1$ for all $i$ and all $j > 0$
5. every entry of \( v \) and \( w \) is bounded by \( L/2 \) in absolute value, where \( L \) is the least common multiple of \( \{ q_i - 1 : q_i \text{ in split_primes_list} \} \)

EXAMPLES:

```
sage: from sage.rings.number_field.S_unit_solver import compatible_system_lift
sage: split_primes_list = [3, 7]
sage: comp_sys = [[[0, 1, 0], [0, 1, 0]], [[0, 3, 4], [0, 1, 2]]]
sage: compatible_system_lift(comp_sys, split_primes_list)
[(0, 3, -2), (0, 1, 2)]
```

```
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import compatible_system_lift
>>> split_primes_list = [Integer(3), Integer(7)]
>>> comp_sys = [[(Integer(0), Integer(1), Integer(0)), (Integer(0), Integer(1),...
  Integer(0))], [(Integer(0), Integer(3), Integer(4)), (Integer(0), Integer(1),...
  Integer(2))]]
>>> compatible_system_lift(comp_sys, split_primes_list)
[(0, 3, -2), (0, 1, 2)]
```

\[\text{sage.rings.number_field.S_unit_solver.compatible_systems} \text{(split_prime_list, complement_exp_vec_dict)}\]

Given dictionaries of complement exponent vectors for various primes that split in \( K \), compute all possible compatible systems.

INPUT:
- `split_prime_list` — a list of rational primes that split completely in \( K \)
- `complement_exp_vec_dict` — a dictionary of dictionaries. The keys are primes from `split_prime_list`.

OUTPUT:
A list of compatible systems of exponent vectors.

Note:
- For any \( q \) in `split_prime_list`, `complement_exp_vec_dict[q]` is a dictionary whose keys are exponent vectors modulo \( q - 1 \) and whose values are lists of exponent vectors modulo \( q - 1 \) which are complementary to the key.
- An item in `system_list` has the form \([ [v_0, w_0], [v_1, w_1], \ldots, [v_k, w_k] ]\), where:
  - \````
  - \``q_j = \text{split_prime_list}[j]````
  - \````
  - \``v_j`` and \``w_j`` are complementary exponent vectors modulo \``q_j - 1````
  - the pairs are all simultaneously compatible.
- \Let \( H = \text{lcm( q_j - 1 : q_j \text{ in split_primes_list} )} \). Then for any compatible system, there is at most one pair of integer exponent vectors \([v, w]\) such that:
  - every entry of \``v`` and \``w`` is bounded in absolute value by \``H```
  - for any \``q_j```, \``v`` and \``w`` agree modulo \``(q_j - 1)`
  - for any \``q_j```, \``v`` and \``w`` agree modulo \``(q_j - 1)`

EXAMPLES:
sage: from sage.rings.number_field.S_unit_solver import compatible_systems
sage: split_primes_list = [3, 7]
sage: checking_dict = {3: {(0, 1, 0): [(1, 0, 0)]}, 7: {(0, 1, 0): [(1, 0, 0)]}}
sage: compatible_systems(split_primes_list, checking_dict)
[[[(0, 1, 0), (1, 0, 0)], [(0, 1, 0), (1, 0, 0)]]]

sage.rings.number_field.S_unit_solver.compatible_vectors(a, m0, m1, g)

Given an exponent vector \(a\) modulo \(m_0\), return an iterator over the exponent vectors for the modulus \(m_1\), such that a lift to the lcm modulus exists.

**INPUT:**

- \(a\) – an exponent vector for the modulus \(m_0\)
- \(m_0\) – a positive integer (specifying the modulus for \(a\))
- \(m_1\) – a positive integer (specifying the alternate modulus)
- \(g\) – the gcd of \(m_0\) and \(m_1\)

**OUTPUT:**

A list of exponent vectors modulo \(m_1\) which are compatible with \(a\).

**Note:**

- Exponent vectors must agree exactly in the 0th position in order to be compatible.

**EXAMPLES:**

sage: from sage.rings.number_field.S_unit_solver import compatible_vectors
sage: a = (3, 1, 8, 1)
sage: list(compatible_vectors(a, 18, 12, gcd(18,12)))
[(3, 1, 2, 1), (3, 1, 2, 7), (3, 1, 8, 1), (3, 1, 8, 7), (3, 7, 2, 1), (3, 7, 2, 7), (3, 7, 8, 1), (3, 7, 8, 7)]

>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import compatible_vectors
>>> a = (Integer(3), Integer(1), Integer(8), Integer(1))
>>> list(compatible_vectors(a, Integer(18), Integer(12), gcd(Integer(18),
˓→Integer(12))))
[(3, 1, 2, 1), (3, 1, 2, 7), (3, 1, 8, 1), (3, 1, 8, 7), (3, 7, 2, 1), (3, 7, 2, 7), (3, 7, 8, 1), (3, 7, 8, 7)]

(continues on next page)
(3, 1, 2, 7),
(3, 1, 8, 1),
(3, 1, 8, 7),
(3, 7, 2, 1),
(3, 7, 2, 7),
(3, 7, 8, 1),
(3, 7, 8, 7)]

The order of the moduli matters.

sage: len(list(compatible_vectors(a, 18, 12, gcd(18,12))))
8
sage: len(list(compatible_vectors(a, 12, 18, gcd(18,12))))
27

>>> from sage.all import *

>>> len(list(compatible_vectors(a, Integer(18), Integer(12), gcd(Integer(18), Integer(12))))))
8
>>> len(list(compatible_vectors(a, Integer(12), Integer(18), gcd(Integer(18), Integer(12))))))
27

sage.rings.number_field.S_unit_solver.compatible_vectors_check(a0, a1, g, l)

Given exponent vectors with respect to two moduli, determine if they are compatible.

INPUT:

• a0 – an exponent vector modulo m0
• a1 – an exponent vector modulo m1 (must have the same length as a0)
• g – the gcd of m0 and m1
• l – the length of a0 and of a1

OUTPUT:

True if there is an integer exponent vector a satisfying

\[
\begin{align*}
a[0] &= a0[0] == a1[0] \\
\mod m_0 \\
a[1:] &= a0[1:] == a1[1:] \\
\mod m_1
\end{align*}
\]

and False otherwise.

Note:

• Exponent vectors must agree exactly in the first coordinate.
• If exponent vectors are different lengths, an error is raised.

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import compatible_vectors_check
sage: a0 = (3, 1, 8, 11)
sage: a1 = (3, 5, 6, 13)
sage: a2 = (5, 5, 6, 13)
sage: compatible_vectors_check(a0, a1, gcd(12, 22), 4r)
True
sage: compatible_vectors_check(a0, a2, gcd(12, 22), 4r)
False

```python
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import compatible_vectors_check

>>> a0 = (Integer(3), Integer(1), Integer(8), Integer(11))
>>> a1 = (Integer(3), Integer(5), Integer(6), Integer(13))
>>> a2 = (Integer(5), Integer(5), Integer(6), Integer(13))

>>> compatible_vectors_check(a0, a1, gcd(Integer(12), Integer(22)), 4)
True

>>> compatible_vectors_check(a0, a2, gcd(Integer(12), Integer(22)), 4)
False
```

`sage.rings.number_field.S_unit_solver.construct_comp_exp_vec(rfv_to_ev_dict, q)`

Constructs a dictionary associating complement vectors to residue field vectors.

**INPUT:**

- `rfv_to_ev_dict` — a dictionary whose keys are residue field vectors and whose values are lists of exponent vectors with the associated residue field vector.
- `q` — the characteristic of the residue field

**OUTPUT:**

A dictionary whose typical key is an exponent vector `a`, and whose associated value is a list of complementary exponent vectors to `a`.

**EXAMPLES:**

In this example, we use the list generated when solving the \(S\)-unit equation in the case that \(K\) is defined by the polynomial \(x^2 + x + 1\) and \(S\) consists of the primes above 3

```python
sage: from sage.rings.number_field.S_unit_solver import construct_comp_exp_vec
sage: rfv_to_ev_dict = {(6, 6): [(3, 0)], (5, 6): [(1, 2)], (5, 4): [(5, 3)], ...:
(6, 2): [(5, 5)], (2, 5): [(0, 1)], (5, 5): [(3, 4)], ...
......:
(4, 4): [(0, 2)], (6, 3): [(1, 4)], (3, 6): [(5, 4)], ...
......:
(2, 2): [(0, 4)], (3, 5): [(1, 0)], (6, 4): [(1, 1)], ...
......:
(3, 2): [(1, 3)], (2, 6): [(4, 5)], (4, 5): [(4, 3)], ...
......:
(2, 3): [(2, 3)], (4, 2): [(4, 0)], (6, 5): [(5, 2)], ...
......:
(3, 3): [(3, 2)], (5, 3): [(5, 0)], (4, 6): [(2, 1)], ...
......:
(3, 4): [(3, 5)], (4, 3): [(0, 5)], (5, 2): [(3, 1)], ...
......:
(2, 4): [(2, 0)]}

sage: construct_comp_exp_vec(rfv_to_ev_dict, 7)
((0, 1): [(1, 4)],
(0, 2): [(0, 2)],
(0, 4): [(3, 0)],
(0, 5): [(4, 3)],
(1, 0): [(5, 0)],
(1, 1): [(2, 0)],
(1, 2): [(1, 3)],
(1, 3): [(1, 2)],
(1, 4): [(0, 1)],
(2, 0): [(1, 1)],
(2, 1): [(4, 0)],
```

(continues on next page)
\[(2, 3): \{(5, 2)\},\]
\[(3, 0): \{(0, 4)\},\]
\[(3, 1): \{(5, 4)\},\]
\[(3, 2): \{(3, 4)\},\]
\[(3, 4): \{(3, 2)\},\]
\[(3, 5): \{(5, 3)\},\]
\[(4, 0): \{(2, 1)\},\]
\[(4, 3): \{(0, 5)\},\]
\[(4, 5): \{(5, 5)\},\]
\[(5, 0): \{(1, 0)\},\]
\[(5, 2): \{(2, 3)\},\]
\[(5, 3): \{(3, 5)\},\]
\[(5, 4): \{(3, 1)\},\]
\[(5, 5): \{(4, 5)\}\]

```python
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import construct_comp_exp_vec
>>>
rfv_to_ev_dict = {(Integer(6), Integer(6)): [(Integer(3), Integer(0))],
⟹(Integer(5), Integer(6)): [(Integer(1), Integer(2))], (Integer(5), Integer(4)): ⦃
⟹[(Integer(5), Integer(3))],
...
  (Integer(6), Integer(2)): [(Integer(5), Integer(5))],
  (Integer(5), Integer(5)): ⦃
  (Integer(3), Integer(4))],
...
  (Integer(5), Integer(5)): [(Integer(3), Integer(6))],
  (Integer(3), Integer(6)): ⦃
  (Integer(2), Integer(5))],
...
  (Integer(2), Integer(5)): [(Integer(0), Integer(1))], (Integer(6), Integer(4)): ⦃
  (Integer(0), Integer(3))],
...
  (Integer(3), Integer(4)): [(Integer(1), Integer(3))],
...  (Integer(1), Integer(3)): ⦃
...  (Integer(0), Integer(4))], (Integer(6), Integer(3)): ⦃
...
  (Integer(5), Integer(4)): [(Integer(4), Integer(5))],
  (Integer(4), Integer(5)): ⦃
  (Integer(2), Integer(6))],
...
  (Integer(2), Integer(6)): [(Integer(4), Integer(0))], (Integer(5), Integer(2)): ⦃
  (Integer(4), Integer(0))],
...
  (Integer(4), Integer(0)): [(Integer(5), Integer(3))],
...
  (Integer(3), Integer(3)): [(Integer(3), Integer(2))],
  (Integer(3), Integer(2)): ⦃
  (Integer(2), Integer(1))],
...
  (Integer(2), Integer(1)): [(Integer(0), Integer(5))], (Integer(5), Integer(1)): ⦃
  (Integer(0), Integer(2))],
...
  (Integer(0), Integer(2)): [(Integer(2), Integer(0))],
...
  (Integer(2), Integer(0)): [(Integer(2), Integer(0))], (Integer(3), Integer(1)): ⦃
  (Integer(2), Integer(0))],
...
  (Integer(2), Integer(0)): [(Integer(2), Integer(0))],
...
  (Integer(0), Integer(0)): [(Integer(2), Integer(0))],
...  (Integer(2), Integer(0)): ⦃
...  (Integer(0), Integer(0))],

>>> construct_comp_exp_vec(rfv_to_ev_dict, Integer(7))
{(0, 1): [(1, 4)],
 (0, 2): [(0, 2)],
 (0, 4): [(3, 0)],
 (0, 5): [(4, 3)],
 (1, 0): [(5, 0)],
 (1, 1): [(2, 0)],
 (1, 2): [(1, 3)],
 (1, 3): [(1, 2)],
 (1, 4): [(0, 1)],
 (2, 0): [(1, 1)],
 (2, 1): [(4, 0)],
 (2, 3): [(5, 2)],
 (3, 0): [(0, 4)],
(continues on next page)
sage.rings.number_field.S_unit_solver.construct_complement_dictionaries(split_primes_list, SUK, verbose=False)

Construct the complement exponent vector dictionaries.

INPUT:

• split_primes_list – a list of rational primes which split completely in the number field $K$
• SUK – the $S$-unit group for a number field $K$
• verbose – a boolean to provide additional feedback (default: False)

OUTPUT:

A dictionary of dictionaries. The keys coincide with the primes in split_primes_list For each $q$, comp_exp_vec[q] is a dictionary whose keys are exponent vectors modulo $q - 1$, and whose values are lists of exponent vectors modulo $q - 1$

If $w$ is an exponent vector in comp_exp_vec[q][v], then the residue field vectors modulo $q$ for $v$ and $w$ sum to $[1, 1, \ldots, 1]$

Note:

• The data of comp_exp_vec will later be lifted to $\mathbb{Z}$ to look for true $S$-Unit equation solutions.
• During construction, the various dictionaries are compared to each other several times to eliminate as many mod $q$ solutions as possible.
• The authors acknowledge a helpful discussion with Norman Danner which helped formulate this code.

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import construct_complement_dictionaries
sage: x = polygen(ZZ, 'x')

4.7. Solver for the $S$-unit equation $x + y = 1$
....: (0, 1, 2): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
....: (0, 3, 2): [(1, 0, 0), (1, 4, 4), (1, 2, 2)],
....: (0, 3, 4): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
....: (0, 5, 0): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
....: (0, 5, 4): [(1, 0, 0), (1, 4, 4), (1, 2, 2)],
....: (1, 0, 0): [(0, 5, 4), (0, 3, 2), (0, 1, 0)],
....: (1, 0, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (1, 0, 4): [(1, 2, 4), (1, 4, 0), (1, 0, 2)],
....: (1, 2, 0): [(1, 2, 4), (1, 4, 0), (1, 0, 2)],
....: (1, 2, 2): [(0, 5, 4), (0, 3, 2), (0, 1, 0)],
....: (1, 2, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (1, 4, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (1, 4, 2): [(1, 2, 4), (1, 4, 0), (1, 0, 2)],
....: (1, 4, 4): [(0, 5, 4), (0, 3, 2), (0, 1, 0)]}

sage: all(set(actual[p][vec]) == set(expected[p][vec])
....: for p in [3, 7] for vec in expected[p])
True

>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import construct_complement_dictionaries
>>> x = polygen(ZZ, 'x')
>>> f = x**Integer(2) + Integer(5)
>>> H = Integer(10)
>>> K = NumberField(f, names=('xi',)); (xi,)=K._first_ngens(1)
>>> SUK = K.S_unit_group(S=K.primes_above(H))
>>> split_primes_list = [Integer(3), Integer(7)]
>>> actual = construct_complement_dictionaries(split_primes_list, SUK)
>>> expected = {Integer(3): {(Integer(0), Integer(1), Integer(0)): [(Integer(1), Integer(0), Integer(0)), (Integer(0), Integer(1), Integer(0))],
... Integer(0), Integer(0)), (Integer(0), Integer(1), Integer(0))]},
... Integer(7): {Integer(0), Integer(1), Integer(0))]: [(Integer(1), Integer(0), Integer(0)), (Integer(0), Integer(1), Integer(0))],
... Integer(0), Integer(0), (Integer(0), Integer(1), Integer(0))}],
... Integer(2), Integer(2))],
... (Integer(0), Integer(1), Integer(2)): [(Integer(0), Integer(1), Integer(2)), (Integer(0), Integer(3), Integer(4), Integer(4)), (Integer(0), Integer(2), Integer(2))],
... (Integer(0), Integer(3), Integer(2)): [(Integer(1), Integer(4), Integer(4)), (Integer(1), Integer(2), Integer(2))],
... (Integer(0), Integer(3), Integer(4)): [(Integer(0), Integer(4), Integer(4)), (Integer(0), Integer(5), Integer(0))],
... (Integer(0), Integer(5), Integer(0)): [(Integer(0), Integer(5), Integer(0))],
... (Integer(1), Integer(2), Integer(2))],
... (Integer(0), Integer(5), Integer(4)): [(Integer(1), Integer(2), Integer(2))],
... (Integer(1), Integer(4), Integer(4)), (Integer(0), Integer(3), Integer(2))],
... (Integer(0), Integer(4), Integer(4)), (Integer(0), Integer(2), Integer(2))],
... (Integer(1), Integer(2), Integer(2))],
... (Integer(1), Integer(4), Integer(4)), (Integer(0), Integer(3), Integer(2))]

(continues on next page)
sage.rings.number_field.S_unit_solver.construct_rfv_to_ev(rfv_dictionary, q, d, verbose=False)

Return a reverse lookup dictionary, to find the exponent vectors associated to a given residue field vector.

INPUT:
- rfv_dictionary – a dictionary whose keys are exponent vectors and whose values are the associated residue field vectors
- q – a prime (assumed to split completely in the relevant number field)
- d – the number of primes in $K$ above the rational prime $q$
- verbose – a boolean flag to indicate more detailed output is desired (default: False)

OUTPUT:
A dictionary $P$ whose keys are residue field vectors and whose values are lists of all exponent vectors which correspond to the given residue field vector.

Note:
- For example, if $rfv\_dictionary[e0] = r0$, then $P[r0]$ is a list which contains $e0$.
- During construction, some residue field vectors can be eliminated as coming from solutions to the $S$-unit equation. Such vectors are dropped from the keys of the dictionary $P$.

EXAMPLES:
In this example, we use a truncated list generated when solving the $S$-unit equation in the case that $K$ is defined by the polynomial $x^2 + x + 1$ and $S$ consists of the primes above 3:
sage: from sage.rings.number_field.S_unit_solver import construct_rfv_to_ev
sage: rfv_dict = {(1, 3): [3, 2], (3, 0): [6, 6], (5, 4): [3, 6], (2, 1): [4, 6], ...
   (4, 0): [4, 2], (1, 2): [5, 6]}
sage: construct_rfv_to_ev(rfv_dict, 7, 2, False)
{(3, 2): [(1, 3)], (4, 2): [(4, 0)], (4, 6): [(2, 1)], (5, 6): [(1, 2)]}

sage: from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import construct_rfv_to_ev
>>> rfv_dict = {(Integer(1), Integer(3)): [Integer(3), Integer(2)], (Integer(3),...
               Integer(0)): [Integer(6), Integer(6)], (Integer(5), Integer(4)): [Integer(3),...
               Integer(1)): [Integer(4), Integer(4)], (Integer(1),...
               Integer(2)): [Integer(5), Integer(6)]}
>>> construct_rfv_to_ev(rfv_dict, Integer(7), Integer(2), False)
{(3, 2): [(1, 3)], (4, 2): [(4, 0)], (4, 6): [(2, 1)], (5, 6): [(1, 2)]}

sage.rings.number_field.S_unit_solver.cx_LLL_bound(SUK, A, prec=106)

Return the maximum of all of the $K_1$’s as they are LLL-optimized for each infinite place $v$.

INPUT:

- SUK – a group of $S$-units
- A – a list of all products of each potential $a, b$ in the $S$-unit equation $ax + by + 1 = 0$ with each root of unity of $K$
- prec – precision of real field (default: 106)

OUTPUT:

A bound for the exponents at the infinite place, as a real number

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import cx_LLL_bound
sage: x = polygen(ZZ, 'x')

570 Chapter 4. Orders, Ideals and Ideal Classes
OUTPUT:

A polynomial with integer coefficients that is equivalent \( \mod p^{\text{prec}} \) to a defining polynomial for the completion of \( K \) associated to the specified prime.

\begin{itemize}
  \item \textbf{Note:} \( K \) has to be an absolute extension
\end{itemize}

EXAMPLES:

```
sage: from sage.rings.number_field.S_unit_solver import defining_polynomial_for_Kp
sage: K.<a> = QuadraticField(2)
sage: p2 = K.prime_above(7); p2
Fractional ideal (-2*a + 1)
sage: defining_polynomial_for_Kp(p2, 10)
x + 266983762
```

```
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import defining_polynomial_for_Kp
>>> K = QuadraticField(Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> p2 = K.prime_above(Integer(7)); p2
Fractional ideal (-2*a + 1)
>>> defining_polynomial_for_Kp(p2, Integer(10))
x + 266983762
```

```
sage: K.<a> = QuadraticField(-6)
sage: p2 = K.prime_above(2); p2
Fractional ideal (2, a)
sage: defining_polynomial_for_Kp(p2, 100)
x^2 + 6
```

```
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import defining_polynomial_for_Kp
>>> K = QuadraticField(-Integer(6), names=('a',)); (a,) = K._first_ngens(1)
>>> p2 = K.prime_above(Integer(2)); p2
Fractional ideal (2, a)
>>> defining_polynomial_for_Kp(p2, Integer(100))
x^2 + 6
```

```
sage.rings.number_field.S_unit_solver.drop_vector\( \text{ev, p, q, complement\_ev\_dict} \)
```

Determine if the exponent vector, \( \text{ev} \), may be removed from the complement dictionary during construction. This will occur if \( \text{ev} \) is not compatible with an exponent vector \( \mod q - 1 \).

\begin{itemize}
  \item \textbf{INPUT:}
    \begin{itemize}
      \item \( \text{ev} \) – an exponent vector \( \mod p - 1 \)
      \item \( p \) – the prime such that \( \text{ev} \) is an exponent vector \( \mod p - 1 \)
      \item \( q \) – a prime, distinct from \( p \), that is a key in the \text{complement\_ev\_dict}
    \end{itemize}
\end{itemize}
• complement_ev_dict – a dictionary of dictionaries, whose keys are primes. complement_ev_dict[q] is a dictionary whose keys are exponent vectors modulo q − 1 and whose values are lists of complementary exponent vectors modulo q − 1.

OUTPUT:

Returns True if ev may be dropped from the complement exponent vector dictionary, and False if not.

Note:

• If ev is not compatible with any of the vectors modulo q − 1, then it can no longer correspond to a solution of the S-unit equation. It returns True to indicate that it should be removed.

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import drop_vector
drop_vector((1, 2, 5), 7, 11, {11: {(1, 1, 3): [(1, 1, 3), (2, 3, 4)]}})
True
```

```python
sage: from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import drop_vector
>>> drop_vector((Integer(1), Integer(2), Integer(5)), Integer(7), Integer(11),
....: (Integer(11): {(Integer(1), Integer(1), Integer(3)): [(Integer(1), Integer(1), Integer(3)), (Integer(2), Integer(3), Integer(4))]}))
True
```

```python
sage: P = {3: {(1, 0, 0): [(1, 0, 0), (0, 1, 0)],
....: (0, 1, 0): [(1, 0, 0), (0, 1, 0)]},
....: 7: {(0, 3, 4): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
....: (1, 2, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (0, 1, 2): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
....: (1, 4, 2): [(1, 2, 4), (1, 4, 0), (1, 0, 2)],
....: (1, 0, 4): [(1, 2, 4), (1, 4, 0), (1, 0, 2)],
....: (0, 3, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (1, 0, 0): [(0, 5, 4), (0, 3, 2), (0, 1, 0)],
....: (1, 2, 0): [(1, 2, 4), (1, 4, 0), (1, 0, 2)],
....: (0, 1, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (0, 5, 0): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
....: (1, 2, 2): [(0, 5, 4), (0, 3, 2), (0, 1, 0)],
....: (1, 4, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (1, 0, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
....: (1, 4, 4): [(0, 5, 4), (0, 3, 2), (0, 1, 0)]}
drop_vector((0, 1, 0), 3, 7, P)
False
```

```python
>>> from sage.all import *
>>> P = {Integer(3): {(Integer(1), Integer(0), Integer(0)): [(Integer(1),
....: Integer(0), Integer(0)), (Integer(0), Integer(1), Integer(0))]},
....: (Integer(7): {(Integer(0), Integer(3), Integer(4)): [(Integer(0),
....: Integer(1), Integer(2)), (Integer(0), Integer(3), Integer(4)), (Integer(0),
....: Integer(5), Integer(0))],
..... (Integer(1), Integer(2), Integer(4)): [(Integer(1), Integer(0),
....: Integer(4)), (Integer(1), Integer(4), Integer(2)), (Integer(1), Integer(2),
....: (continues on next page)
sage.rings.number_field.S_unit_solver.embedding_to_Kp(a, prime, prec)

INPUT:
- `a` – an element of a number field $K$
- `prime` – a prime ideal of $K$
- `prec` – a positive natural number

OUTPUT:
An element of $K$ that is equivalent to $a$ modulo $p^\text{prec}$ and the generator of $K$ appears with exponent less than $e \cdot f$, where $p$ is the rational prime below `prime` and $e$, $f$ are the ramification index and residue degree, respectively.

4.7. Solver for the $S$-unit equation $x + y = 1$
Note: $K$ has to be an absolute number field

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import embedding_to_Kp
sage: K.<a> = QuadraticField(17)
sage: p = K.prime_above(13); p
Fractional ideal (-a + 2)
sage: embedding_to_Kp(a-3, p, 15)
-20542890112375827
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import embedding_to_Kp
>>> K = QuadraticField(Integer(17), names=('a',)); (a,) = K._first_ngens(1)
>>> p = K.prime_above(Integer(13)); p
Fractional ideal (-a + 2)
>>> embedding_to_Kp(a-Integer(3), p, Integer(15))
-20542890112375827
```

```python
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^4 - 2)
sage: p = K.prime_above(7); p
Fractional ideal (-a^2 + a - 1)
sage: embedding_to_Kp(a^3 - 3, p, 15)
-1261985118949117459462968282807202378
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(4) - Integer(2), names=('a',)); (a,) = K._first_ngens(1)
>>> p = K.prime_above(Integer(7)); p
Fractional ideal (-a^2 + a - 1)
>>> embedding_to_Kp(a**Integer(3) - Integer(3), p, Integer(15))
-1261985118949117459462968282807202378
```

sage.rings.number_field.S_unit_solver.eq_up_to_order($A$, $B$)

If $A$ and $B$ are lists of four-tuples $[a_0,a_1,a_2,a_3]$ and $[b_0,b_1,b_2,b_3]$, check that there is some reordering so that either $a_i=b_i$ for all $i$ or $a_0==b_1,a_1==b_0,a_2==b_3,a_3==b_2$.

The entries must be hashable.

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import eq_up_to_order
sage: L = [(1,2,3,4), (5,6,7,8)]
sage: L1 = [L[1], L[0]]
sage: eq_up_to_order(L, L1)
True
sage: eq_up_to_order(L, L2)
True
sage: eq_up_to_order(L, [(1,2,4,3), (5,6,8,7)])
False
```
sage.rings.number_field.S_unit_solver.log_p(a, prime, prec)

INPUT:

• a – an element of a number field $K$
• prime – a prime ideal of the number field $K$
• prec – a positive integer

OUTPUT:
An element of $K$ which is congruent to the prime-adic logarithm of $a$ with respect to prime modulo $p^\text{prec}$, where $p$ is the rational prime below prime

Note: Here we take into account the other primes in $K$ above $p$ in order to get coefficients with small values

EXAMPLES:

```python
>>> from sage.rings.number_field.S_unit_solver import log_p
>>> x = polygen(ZZ, 'x')
>>> K.<a> = NumberField(x**2 + 14)
>>> p1 = K.primes_above(3)[0]
>>> p1
Fractional ideal (3, a + 1)
>>> log_p(a+2, p1, 20)
8255385638/3*a + 15567609440/3
```
Fractional ideal \((5, a + 1)\)

```python
sage: log_p(1/(a^2-4), p1, 30)
-42392683853751591352946/25*a^3 - 113099841599709611260219/25*a^2 - 8496494127064033599196/5*a - 18774052619501226990432/25
```

```python
>>> from sage.all import *
"""
>>> K = NumberField(x**Integer(4) + Integer(14), names=('a',)); (a,) = K._first_ngens(1)
>>> p1 = K.primes_above(Integer(5))[Integer(0)]
>>> p1
Fractional ideal (5, a + 1)
>>> log_p(Integer(1)/(a**Integer(2)-Integer(4)), p1, Integer(30))
-42392683853751591352946/25*a^3 - 113099841599709611260219/25*a^2 - 8496494127064033599196/5*a - 18774052619501226990432/25
```

```python
sage.rings.number_field.S_unit_solver.log_p_series_part(a, prime, prec)
```

**INPUT:**

- \(a\) – an element of a number field \(K\)
- \(prime\) – a prime ideal of the number field \(K\)
- \(prec\) – a positive integer

**OUTPUT:**

The \(prime\)-adic logarithm of \(a\) and accuracy \(p^{prec}\), where \(p\) is the rational prime below \(prime\)

**ALGORITHM:**

The algorithm is based on the algorithm on page 30 of [Sma1998]

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import log_p_series_part
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 5)
sage: p1 = K.primes_above(3)[0]
sage: p1
Fractional ideal (3)
sage: log_p_series_part(a^2 - a + 1, p1, 30)
120042736778562*a + 263389019530092
```

```python
>>> from sage.all import *
"""
>>> from sage.rings.number_field.S_unit_solver import log_p_series_part
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> p1 = K.primes_above(Integer(3))[Integer(0)]
>>> p1
Fractional ideal (3)
>>> log_p_series_part(a**Integer(2) - a + Integer(1), p1, Integer(30))
120042736778562*a + 263389019530092
```

```python
sage: K.<a> = NumberField(x^4 + 14)
sage: p1 = K.primes_above(5)[0]
sage: p1
Fractional ideal (5, a + 1)
```

(continues on next page)
```python
sage: log_p_series_part(1/(a^2-4), p1, 30)
562894088326458536922468804845989654349879320483965421501954860062122195091510657655581925236618
→ 18465957235571471561578165249366687569727244011302407020045580928059403805622385
2568951718462
→ 2 +
→ 2351432413692022254066438266577101081351482804415905040437326602004946930635942233146528817325
→ 18465957235571471561578165249366687569727244011302407020045580928059403805622385
2568951718462
"'
```

sage.rings.number_field.S_unit_solver.minimal_vector \((A, y, \text{prec}=106)\)

**INPUT:**
- \(A\) – a square \(n\) by \(n\) non-singular integer matrix whose rows generate a lattice \(\mathcal{L}\)
- \(y\) – a row \(1\) by \(n\) vector with integer coordinates
- \(\text{prec}\) – precision of real field (default: 106)

**OUTPUT:**

A lower bound for the square of

\[
\ell(\mathcal{L}, \vec{y}) = \begin{cases} 
\min_{\vec{x} \in \mathcal{L}} \|\vec{x} - \vec{y}\|, & \vec{y} \notin \mathcal{L}, \\
\min_{0 \neq \vec{x} \in \mathcal{L}} \|\vec{x}\|, & \vec{y} \in \mathcal{L}.
\end{cases}
\]

**ALGORITHM:**

The algorithm is based on V.9 and V.10 of [Sma1998]

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import minimal_vector
sage: A = matrix(ZZ, 2, [1,1,1,0])
```

```python
sage: y = vector(ZZ, [2,1])
```

```python
sage: minimal_vector(A, y)
1/2
```

4.7. Solver for the S-unit equation \(x + y = 1\)
```python
sage: B = random_matrix(ZZ, 3)
sage: while not B.determinant():
....:     B = random_matrix(ZZ, 3)
sage: B  # random
[-2 -1 -1]
[ 1 1 -2]
[ 6 1 -1]
sage: y = vector([1, 2, 100])
sage: minimal_vector(B, y)  # random
15/28
```

```python
>>> from sage.all import *
>>> B = random_matrix(ZZ, Integer(3))
>>> while not B.determinant():
...     B = random_matrix(ZZ, Integer(3))
>>> B  # random
[-2 -1 -1]
[ 1 1 -2]
[ 6 1 -1]
>>> y = vector([Integer(1), Integer(2), Integer(100)])
>>> minimal_vector(B, y)  # random
15/28
```

`sage.rings.number_field.S_unit_solver.mus`(SU\(\text{K}\), v)

Return a list \(\{\mu\}\), for \(\mu\) defined in [AKMRVW].

**INPUT:**

- SUK – a group of \(S\)-units
- v – a finite place of \(K\)

**OUTPUT:**

A list \([\mu]\) where each \(\mu\) is an element of \(K\)

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import mus
sage: x = polygen(ZZ, 'x')
sage: K.<xi> = NumberField(x^3 - 3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: v_fin = tuple(K.primes_above(Integer(3)))[0]
sage: mus(SUK, v_fin)
[xi^2 - 2]
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import mus
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3), names=('xi',)); (xi,) = K._first ngens(1)
>>> SUK = UnitGroup(K, S=tuple(K.primes_above(Integer(3))))
>>> v_fin = tuple(K.primes_above(Integer(3)))[Integer(0)]
>>> mus(SUK, v_fin)
[xi^2 - 2]
```

**REFERENCES:**
sage.rings.number_field.S_unit_solver.p_adic_LLL_bound(SUK, A, prec=106)

Return the maximum of all of the $K_0$'s as they are LLL-optimized for each finite place $v$.

INPUT:

- SUK – a group of $S$-units
- A – a list of all products of each potential $a, b$ in the $S$-unit equation $ax + by + 1 = 0$ with each root of unity of $K$
- prec – precision for p-adic LLL calculations (default: 106)

OUTPUT:

A bound for the max of exponents in the case that extremal place is finite (see [Sma1995]) as a real number

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import p_adic_LLL_bound
sage: x = polygen(ZZ, 'x')
sage: K.<xi> = NumberField(x^3 - 3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: A = SUK.roots_of_unity()
sage: prec = 100
sage: p_adic_LLL_bound(SUK, A, prec)
89
```

sage.rings.number_field.S_unit_solver.p_adic_LLL_bound_one_prime(prime, B0, M, M_logp, m0, c3, prec=106)

INPUT:

- prime – a prime ideal of a number field $K$
- B0 – the initial bound
- M – a list of elements of $K$, the $\mu_i$’s from Lemma IX.3 of [Sma1998]
- M_logp – the p-adic logarithm of elements in $M$
- m0 – an element of $K$, this is $\mu_0$ from Lemma IX.3 of [Sma1998]
- c3 – a positive real constant
- prec – the precision of the calculations (default: 106), i.e., values are known to $O(p^{\text{prec}})$

OUTPUT:

A pair consisting of:

1. a new upper bound, an integer

4.7. Solver for the $S$-unit equation $x + y = 1$ 579
2. a boolean value, True if we have to increase precision, otherwise False

**Note:** The constant $c_5$ is the constant $c_5$ at the page 89 of [Sma1998] which is equal to the constant $c_{10}$ at the page 139 of [Sma1995]. In this function, the $c_i$ constants are in line with [Sma1998], but generally differ from the constants in [Sma1995] and other parts of this code.

**EXAMPLES:**

This example indicates a case where we must increase precision:

```python
sage: from sage.rings.number_field.S_unit_solver import p_adic_LLL_bound_one_prime
sage: prec = 50
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^3 - 3)
sage: S = tuple(K.primes_above(3))
sage: SUK = UnitGroup(K, S=S)
sage: v = S[0]
sage: A = SUK.roots_of_unity()
sage: K0_old = 9.4755766731093e17
sage: Mus = [a^2 - 2]
sage: Log_p_Mus = [18505682459351109742400*a^2 + 1389583284398773572269676*a + 717897987691852588770249]
sage: mu0 = K(-1)
sage: c3_value = 0.42578591347980
sage: m0_Kv_new, increase_prec = p_adic_LLL_bound_one_prime(v, K0_old, Mus, Log_p_Mus, mu0, c3_value, prec)
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import p_adic_LLL_bound_one_prime
>>> prec = Integer(50)
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> S = tuple(K.primes_above(Integer(3)))
>>> SUK = UnitGroup(K, S=S)
>>> v = S[Integer(0)]
>>> A = SUK.roots_of_unity()
>>> K0_old = RealNumber('9.4755766731093e17')
>>> Mus = [a**Integer(2) - Integer(2)]
>>> Log_p_Mus = [Integer(18505682459351109742400)*a**Integer(2) + Integer(1389583284398773572269676)*a + Integer(717897987691852588770249)]
>>> mu0 = K(-Integer(1))
>>> c3_value = RealNumber('0.42578591347980')
>>> m0_Kv_new, increase_prec = p_adic_LLL_bound_one_prime(v, K0_old, Mus, Log_p_Mus, mu0, c3_value, prec)
```

>>> m0_Kv_new
0
>>> increase_prec
True
```
And now we increase the precision to make it all work:

```sage
prec = 106
K0_old = 9.47557667310927543280257946930e17
Log_p_Mus = [10295636043909867334686387890424583658678662701816*a^2
....: + 661450700156368458475507052066889190195530948403866*a]
c3_value = 0.4257859134798034746197327286726
m0_Kv_new, increase_prec = p_adic_LLL_bound_one_prime(v, K0_old, Mus, Log_p_Mus,
....: mu0, c3_value, prec)
m0_Kv_new
476
increase_prec
False
```

```sage
>>> from sage.all import *
```
K = NumberField(x**Integer(3) - Integer(3), names=('xi',)); (xi,) = K._first_ngens(1)
S = tuple(K.primes_above(Integer(3)))
SUK = UnitGroup(K, S=S)

v_fin = S[Integer(0)]

possible_mu0s(SUK, v_fin)

[-1, 1]

Note: $n_0$ is the valuation of the coefficient $\alpha_d$ of the $S$-unit equation such that $|\alpha_d \tau_d|_v = 1$. We have set $n_0 = 0$ here since the coefficients are roots of unity $\alpha_0$ is not defined in the paper, we set it to be 1.

REFERENCES:

• [AKMRVW]
• [Sma1995] pp. 824-825, but we modify the definition of $\sigma$ ($\sigma_tilde$) to make it easier to code sage.rings.number_field.S_unit_solver.reduction_step_complex_case(place, B0, list_of_gens, torsion_gen, c13)

INPUT:

• place – (ring morphism) an infinite place of a number field $K$
• B0 – the initial bound
• list_of_gens – a set of generators of the free part of the group
• torsion_gen – an element of the torsion part of the group
• c13 – a positive real number

OUTPUT:

A tuple consisting of:

1. a new upper bound, an integer
2. a boolean value, True if we have to increase precision, otherwise False

Note: The constant $c13$ in Section 5, [AKMRVW]. This function does handle both real and non-real infinite places.

REFERENCES:

See [Sma1998], [AKMRVW].

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import reduction_step_complex_case
sage: x = polygen(ZZ, 'x')

sage: K.<a> = NumberField([x^3 - 2])

sage: SK = sum([K.primes_above(p) for p in [2,3,5]],[])

sage: G = [g for g in K.S_unit_group(S=SK).gens_values()
.....: if g.multiplicative_order() == Infinity]

sage: p1 = K.places(prec=100)[1]
sage: reduction_step_complex_case(p1, 10^5, G, -1, 2)
(18, False)

```python
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import reduction_step_complex_case
>>>
x = polygen(ZZ, x)
>>>
K = NumberField([x**Integer(3) - Integer(2)], names=('a',)); (a,) = K._first_ngens(1)
>>>
SK = sum([K.primes_above(p) for p in [Integer(2),Integer(3),Integer(5)]],[])
>>>
G = [g for g in K.S_unit_group(S=SK).gens_values() ... if g.multiplicative_order() == Infinity]
>>>
p1 = K.places(prec=Integer(10)**Integer(5))[Integer(1)]
>>>
reduction_step_complex_case(p1, Integer(10)**Integer(5), G, -Integer(1), Integer(2))
(18, False)
```

`sage.rings.number_field.S_unit_solver.sieve_below_bound(K, S, bound=10, bump=10, split_primes_list=[], verbose=False)`

Return all solutions to the $S$-unit equation $x + y = 1$ over $K$ with exponents below the given bound.

**INPUT:**
- $K$ – a number field (an absolute extension of the rationals)
- $S$ – a list of finite primes of $K$
- **bound** – a positive integer upper bound for exponents, solutions with exponents having absolute value below this bound will be found (default: 10)
- **bump** – a positive integer by which the minimum LCM will be increased if not enough split primes are found in sieving step (default: 10)
- **split_primes_list** – a list of rational primes that split completely in the extension $K/Q$, used for sieving. For complete list of solutions should have lcm of $\{(p_i - 1)\}_{i=1}^{t}$ for primes $p_i$ greater than bound (default: [])
- **verbose** – an optional parameter allowing the user to print information during the sieving process (default: False)

**OUTPUT:**
A list of tuples $[(A_1, B_1, x_1, y_1), (A_2, B_2, x_2, y_2), \ldots, (A_n, B_n, x_n, y_n)]$ such that:
1. The first two entries are tuples $A_i = (a_0, a_1, \ldots, a_t)$ and $B_i = (b_0, b_1, \ldots, b_t)$ of exponents.
2. The last two entries are $S$-units $x_i$ and $y_i$ in $K$ with $x_i + y_i = 1$.
3. If the default generators for the $S$-units of $K$ are $(\rho_0, \rho_1, \ldots, \rho_t)$, then these satisfy $x_i = \prod (\rho_i)^{a_i}$ and $y_i = \prod (\rho_i)^{b_i}$.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import sieve_below_bound, eq_up_to_order
sage: x = polygen(ZZ, 'x')
sage: K.<xi> = NumberField(x^2 + x + 1)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: S = SUK.primes()
sage: sols = sieve_below_bound(K, S, 10)
```

4.7. Solver for the $S$-unit equation $x + y = 1$
sage: expected = [((1, -1), (0, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
.....: (0, 1), (4, 0), xi + 2, -xi - 1),
.....: ((2, 0), (5, 1), xi, -xi + 1),
.....: ((1, 0), (5, 0), xi + 1, -xi)]
sage: eq_up_to_order(sols, expected)
True

sage.rings.number_field.S_unit_solver.sieve_ordering(SUK, q)

Return ordered data for running sieve on the primes in SUK over the rational prime q.

INPUT:

- SUK – the S-unit group of a number field K
- q – a rational prime number which splits completely in K

OUTPUT:

A list of tuples, [ideals_over_q, residue_fields, rho_images, product_rho_orders], where

1. ideals_over_q is a list of the $d = [K : Q]$ ideals in K over q
2. residue_fields[i] is the residue field of ideals_over_q[i] 
3. rho_images[i] is a list of the reductions of the generators in of the S-unit group, modulo ideals_over_q[i]
4. product_rho_orders[i] is the product of the multiplicative orders of the elements in rho_images[i]

Note:

- The list ideals_over_q is sorted so that the product of orders is smallest for ideals_over_q[0], as this will make the later sieving steps more efficient.
- The primes of S must not lie over q.
EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import sieve_ordering
sage: x = polygen(ZZ, 'x')

sage: K.<xi> = NumberField(x**3 - 3*x + 1)

sage: SUK = K.S_unit_group(S=Integer(3))

sage: sieve_data = list(sieve_ordering(SUK, Integer(19)))

sage: sieve_data[0]
(Fractional ideal (-2*xi^2 + 3),
 Fractional ideal (-xi + 3),
 Fractional ideal (2*xi + 1))

sage: sieve_data[1]
(Residue field of Fractional ideal (-2*xi^2 + 3),
 Residue field of Fractional ideal (-xi + 3),
 Residue field of Fractional ideal (2*xi + 1))

sage: sieve_data[2]
([(18, 12, 16, 8), [18, 16, 10, 4], [18, 10, 12, 10])

sage: sieve_data[3]
(648, 2916, 3888)

>>> from sage.all import *

>>> from sage.rings.number_field.S_unit_solver import sieve_ordering

>>> x = polygen(ZZ, 'x')

>>> K = NumberField(x**Integer(3) - Integer(3)*x + Integer(1), names=(xi,)); →(xi,) = K._first_ngens(1)

>>> SUK = K.S_unit_group(S=Integer(3))

>>> sieve_data = list(sieve_ordering(SUK, Integer(19)))

>>> sieve_data[Integer(0)]
(Fractional ideal (-2*xi^2 + 3),
 Fractional ideal (-xi + 3),
 Fractional ideal (2*xi + 1))

>>> sieve_data[Integer(1)]
(Residue field of Fractional ideal (-2*xi^2 + 3),
 Residue field of Fractional ideal (-xi + 3),
 Residue field of Fractional ideal (2*xi + 1))

>>> sieve_data[Integer(2)]
([(18, 12, 16, 8), [18, 16, 10, 4], [18, 10, 12, 10])

>>> sieve_data[Integer(3)]
(648, 2916, 3888)
```

```python
sage.rings.number_field.S_unit_solver.solutions_from_systems(SUK, bound, cs_list, split_primes_list)
```

Lift compatible systems to the integers and return the $S$-unit equation solutions that the lifts yield.

INPUT:

- SUK – the group of $S$-units where we search for solutions
- bound – a bound for the entries of all entries of all lifts
- cs_list – a list of compatible systems of exponent vectors modulo $q - 1$ for various primes $q

4.7. Solver for the $S$-unit equation $x + y = 1$ 585
• split_primes_list – a list of primes giving the moduli of the exponent vectors in cs_list

OUTPUT:
A list of solutions to the S-unit equation. Each solution is a list:
1. an exponent vector over the integers, ev
2. an exponent vector over the integers, cv
3. the S-unit corresponding to ev, \( \iota_{\text{exp}} \)
4. the S-unit corresponding to cv, \( \iota_{\text{comp}} \)

Note:
• Every entry of ev is less than or equal to bound in absolute value
• Every entry of cv is less than or equal to bound in absolute value
• \( \iota_{\text{exp}} + \iota_{\text{comp}} = 1 \)

EXAMPLES:
Given a single compatible system, a solution can be found.

```python
sage: from sage.rings.number_field.S_unit_solver import solutions_from_systems
sage: x = polygen(ZZ, 'x')
sage: K.<xi> = NumberField(x^2 - 15)
sage: SUK = K.S_unit_group(S=K.primes_above(2))
sage: split_primes_list = [7, 17]
sage: a_compatible_system = [[[0, 0, 5], [0, 0, 5]], [[0, 0, 15], [0, 0, 15]]]
sage: solutions_from_systems(SUK, 20, a_compatible_system, split_primes_list)
[[(0, 0, -1), (0, 0, -1), 1/2, 1/2]]
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import solutions_from_systems
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2) - Integer(15), names=('xi',))
>>> (xi,) = K._first_ngens(1)
>>> SUK = K.S_unit_group(S=K.primes_above(Integer(2)))
>>> split_primes_list = [Integer(7), Integer(17)]
>>> a_compatible_system = [[[Integer(0), Integer(0), Integer(5)], [Integer(0), __Integer(0), Integer(15)]], [[Integer(0), Integer(0), Integer(15)], [Integer(0), __Integer(0), __Integer(15)]]]
>>> solutions_from_systems(SUK, Integer(20), a_compatible_system, split_primes_list)
[[(0, 0, -1), (0, 0, -1), 1/2, 1/2]]
```

sage.rings.number_field.S_unit_solver.solve_S_unit_equation(K, S, prec=106, include_exponents=True, include_bound=False, proof=None, verbose=False)

Return all solutions to the \( S \)-unit equation \( x + y = 1 \) over \( K \).

INPUT:
• \( K \) – a number field (an absolute extension of the rationals)
• \( S \) – a list of finite primes of \( K \)
• `prec` – precision used for computations in real, complex, and p-adic fields (default: 106)
• `include_exponents` – whether to include the exponent vectors in the returned value (default: True).
• `include_bound` – whether to return the final computed bound (default: False)
• `verbose` – whether to print information during the sieving step (default: False)

**OUTPUT:**
A list of tuples \([ (A_1, B_1, x_1, y_1), (A_2, B_2, x_2, y_2), \ldots, (A_n, B_n, x_n, y_n) ]\) such that:

1. The first two entries are tuples \(A_i = (a_0, a_1, \ldots, a_t)\) and \(B_i = (b_0, b_1, \ldots, b_t)\) of exponents. These will be omitted if `include_exponents` is False.
2. The last two entries are \(S\)-units \(x_i\) and \(y_i\) in \(K\) with \(x_i + y_i = 1\).
3. If the default generators for the \(S\)-units of \(K\) are \((\rho_0, \rho_1, \ldots, \rho_t)\)', then these satisfy \(x_i = \prod (\rho_i)^{a_i}\) and \(y_i = \prod (\rho_i)^{b_i}\).

If `include_bound`, will return a pair \((\text{sols}, \text{bound})\) where \(\text{sols}\) is as above and \(\text{bound}\) is the bound used for the entries in the exponent vectors.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import solve_S_unit_equation, eq_up_to_order
sage: x = polygen(ZZ, 'x')

sage: K.<xi> = NumberField(x^2 + x + 1)
sage: S = K.primes_above(3)
sage: sols = solve_S_unit_equation(K, S, 200)
sage: expected = 
[[(0, 1), (4, 0), xi + 2, -xi - 1],
...(1, 1), (0, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
...(1, 0), (5, 0), xi + 1, -xi),
...(2, 0), (5, 1), xi, -xi + 1]]
sage: eq_up_to_order(sols, expected)
True
```

In order to see the bound as well, use the optional parameter `include_bound`:
You can omit the exponent vectors:

```python
sage: sols = solve_S_unit_equation(K, S, 200, include_exponents=False)
sage: expected = 
[(xi + 2, -xi - 1), (1/3*xi + 2/3, -1/3*xi + 1/3), ...
(-xi, xi + 1), (-xi + 1, xi)]
sage: set(frozenset(a) for a in sols) == set(frozenset(b) for b in expected)
True
```
• SUK – the $S$-unit group of an absolute number field $K$.
• bound – a positive integer

OUTPUT:

A list $L$ of rational primes $q$, with the following properties:
• each prime $q$ in $L$ splits completely in $K$
• if $Q$ is a prime in $S$ and $q$ is the rational prime below $Q$, then $q$ is not in $L$
• the value $\text{lcm}\{q - 1 : q \in L\}$ is greater than or equal to $2 \times \text{bound} + 1$.

Note:
• A series of compatible exponent vectors for the primes in $L$ will lift to at most one integer exponent vector whose entries $a_i$ satisfy $|a_i|$ is less than or equal to $\text{bound}$.
• The ordering of this set is not very intelligent for the purposes of the later sieving processes.

EXAMPLES:

```sage
tfrom sage.rings.number_field.S_unit_solver import split_primes_large_lcm
tx = polygen(ZZ, 'x')
tK, <xi> = NumberField(x^3 - 3*x + 1)
tS = K.primes_above(3)
tSUK = UnitGroup(K, S=tuple(S))
tsplit_primes_large_lcm(SUK, 200)
[17, 19, 37, 53]
```

>>> from sage.all import *
>>> from sage.rings.number_field.S_unit_solver import split_primes_large_lcm
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(3) - Integer(3)*x + Integer(1), names=('xi',)); (xi,)
TINGS_first_ngens(1)
>>> S = K.primes_above(Integer(3))
>>> SUK = UnitGroup(K, S=tuple(S))
>>> split_primes_large_lcm(SUK, Integer(200))
[17, 19, 37, 53]

With a tiny bound, Sage may ask you to increase the bound.

```sage
tfrom sage.rings.number_field.S_unit_solver import split_primes_large_lcm
tK, <xi> = NumberField(x^2 + 163)
tSUK = UnitGroup(K, S=tuple(K.primes_above(23)))
tsplit_primes_large_lcm(SUK, Integer(8))
```

Traceback (most recent call last):
... 
ValueError: Not enough split primes found. Increase bound.

```sage
tfrom sage.all import *
tfrom sage.rings.number_field.S_unit_solver import split_primes_large_lcm
>>> K = NumberField(x**Integer(2) + Integer(163), names=('xi',)); (xi,) = K._
TINGS_first_ngens(1)
>>> SUK = UnitGroup(K, S=tuple(K.primes_above(Integer(23)))))
>>> split_primes_large_lcm(SUK, Integer(8))
```

Traceback (most recent call last):
(continues on next page)
4.8 Small primes of degree one

Iterator for finding several primes of absolute degree one of a number field of small prime norm.

**ALGORITHM:**

Let $P$ denote the product of some set of prime numbers. (In practice, we use the product of the first 10000 primes, because Pari computes this many by default.)

Let $K$ be a number field and let $f(x)$ be a polynomial defining $K$ over the rational field. Let $\alpha$ be a root of $f$ in $K$.

We know that $[O_K : \mathbb{Z}[\alpha]]^2 = |\Delta(f(x))/\Delta(O_K)|$, where $\Delta$ denotes the discriminant (see, for example, Proposition 4.4.4, p165 of [Coh1993]). Therefore, after discarding primes dividing $\Delta(f(x))$ (this includes all ramified primes), any integer $n$ such that $\gcd(f(n), P) > 0$ yields a prime $p|P$ such that $f(x)$ has a root modulo $p$. By the condition on discriminants, this root is a single root. As is well known (see, for example Theorem 4.8.13, p199 of [Coh1993]), the ideal generated by $(p, \alpha - n)$ is prime and of degree one.

**Warning:** It is possible that there are no primes of $K$ of absolute degree one of small prime norm, and it is possible that this algorithm will not find any primes of small norm.

**Todo:** There are situations when this will fail. There are questions of finding primes of relative degree one. There are questions of finding primes of exact degree larger than one. In short, if you can contribute, please do!

**EXAMPLES:**

```python
sage: x = ZZ['x'].gen()
sage: F.<a> = NumberField(x^2 - 2)
sage: Ps = F.primes_of_degree_one_list(3)
sage: Ps # random
[Fractional ideal (2*a + 1), Fractional ideal (-3*a + 1), Fractional ideal (-a + 5)]

sage: [ P.norm() for P in Ps ] # random
[7, 17, 23]

sage: all(ZZ(P.norm()).is_prime() for P in Ps)
True

sage: all(P.residue_class_degree() == 1 for P in Ps)
True
```

```python
>>> from sage.all import *

>>> x = ZZ['x'].gen()

>>> F = NumberField(x**Integer(2) - Integer(2), names=('a',)); (a,) = F._first_ngens(1)

>>> Ps = F.primes_of_degree_one_list(Integer(3))

>>> Ps # random
[Fractional ideal (2*a + 1), Fractional ideal (-3*a + 1), Fractional ideal (-a + 5)]

>>> [ P.norm() for P in Ps ] # random
[7, 17, 23]

>>> all(ZZ(P.norm()).is_prime() for P in Ps)
True
```
The next two examples are for relative number fields:

```python
sage: L.<b> = F.extension(x^3 - a)
sage: Ps = L.primes_of_degree_one_list(3)
sage: Ps # random
[Fractional ideal (17, b - 5), Fractional ideal (23, b - 4), Fractional ideal (31, b - 2)]
sage: [ P.absolute_norm() for P in Ps ] # random
[17, 23, 31]
sage: all(ZZ(P.absolute_norm()).is_prime() for P in Ps)
True
sage: all(P.residue_class_degree() == 1 for P in Ps)
True
sage: M.<c> = NumberField(x^2 - x*b^2 + b)
sage: Ps = M.primes_of_degree_one_list(3)
sage: Ps # random
[Fractional ideal (17, c - 2), Fractional ideal (c - 1), Fractional ideal (41, c - 15)]
sage: [ P.absolute_norm() for P in Ps ] # random
[17, 31, 41]
sage: all(ZZ(P.absolute_norm()).is_prime() for P in Ps)
True
sage: all(P.residue_class_degree() == 1 for P in Ps)
True
```

AUTHORS:
- Nick Alexander (2008): initial version
- David Loeffler (2009): fixed a bug with relative fields

4.8. Small primes of degree one
Maarten Derickx (2017): fixed a bug with number fields not generated by an integral element

```python
class sage.rings.number_field.small_primes_of_degree_one.Small_primes_of_degree_one_iter:
    def __init__(self, field, num_integer_primes=10000, max_iterations=100):
        pass

Bases: object

Iterator that finds primes of a number field of absolute degree one and bounded small prime norm.

INPUT:

- field -- a NumberField.
- num_integer_primes -- (default: 10000) an integer. We try to find primes of absolute norm no greater than the num_integer_primes-th prime number. For example, if num_integer_primes is 2, the largest norm found will be 3, since the second prime is 3.
- max_iterations -- (default: 100) an integer. We test max_iterations integers to find small primes before raising StopIteration.

AUTHOR:

- Nick Alexander

next ()

Return a prime of absolute degree one of small prime norm.

Raises StopIteration if such a prime cannot be easily found.

EXAMPLES:

```python
sage: x = QQ['x'].gen()
sage: K.<a> = NumberField(x^2 - 3)
sage: it = K.primes_of_degree_one_iter()
sage: [ next(it) for i in range(3) ] # random
[Fractional ideal (2*a + 1), Fractional ideal (-a + 4), Fractional ideal (3*a + 2)]

>>> from sage.all import *
>>> x = QQ['x'].gen()
>>> K = NumberField(x**Integer(2) - Integer(3), names=('a',)); (a,) = K._first_ngens(1)
>>> it = K.primes_of_degree_one_iter()
>>> [ next(it) for i in range(Integer(3)) ] # random
[Fractional ideal (2*a + 1), Fractional ideal (-a + 4), Fractional ideal (3*a + 2)]
```
4.9 $p$-Selmer groups of number fields

This file contains code to compute $K(S, p)$ where

- $K$ is a number field
- $S$ is a finite set of primes of $K$
- $p$ is a prime number

For $m \geq 2$, $K(S, m)$ is defined to be the finite subgroup of $K^*/(K^*)^m$ consisting of elements represented by $a \in K^*$ whose valuation at all primes not in $S$ is a multiple of $m$. It fits in the short exact sequence

$$1 \to O_{K,S}^*/(O_{K,S}^*)^m \to K(S, m) \to Cl_{K,S}[m] \to 1$$

where $O_{K,S}^*$ is the group of $S$-units of $K$ and $Cl_{K,S}$ the $S$-class group. When $m = p$ is prime, $K(S, p)$ is a finite-dimensional vector space over $GF(p)$. Its generators come from three sources: units (modulo $p$'th powers); generators of the $p$'th powers of ideals which are not principal but whose $p$'the powers are principal; and generators coming from the prime ideals in $S$.

The main function here is $pSelmerGroup()$. This will not normally be used by users, who instead will access it through a method of the NumberField class.

AUTHORS:

- John Cremona (2005-2021)

```
sage.rings.number_field.selmer_group.basis_for_p_cokernel(S, C, p)
```

Return a basis for the group of ideals supported on $S$ (mod $p$'th-powers) whose class in the class group $C$ is a $p$'th power, together with a function which takes the $S$-exponents of such an ideal and returns its coordinates on this basis.

INPUT:

- $S$ (list) – a list of prime ideals in a number field $K$.
- $C$ (class group) – the ideal class group of $K$.
- $p$ (prime) – a prime number.

OUTPUT:

(tuple) ($b$, $f$) where

- $b$ is a list of ideals which is a basis for the group of ideals supported on $S$ (modulo $p$'th powers) whose ideal class is a $p$'th power;
- $f$ is a function which takes such an ideal and returns its coordinates with respect to this basis.

EXAMPLES:

```
sage: from sage.rings.number_field.selmer_group import basis_for_p_cokernel
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - x + 58)
sage: S = K.ideal(30).support(); S
[Fractional ideal (2, a),
 Fractional ideal (2, a + 1),
 Fractional ideal (3, a + 1),
 Fractional ideal (5, a + 1),
 Fractional ideal (5, a + 3)]
sage: C = K.class_group()
sage: C.gens_orders()
```

(continues on next page)
(6, 2)

```
sage: [C(P).exponents() for P in S]
[(5, 0), (1, 0), (3, 1), (1, 1), (5, 1)]
sage: b, f = basis_for_p_cokernel(S, C, 2); b
[Fractional ideal (2), Fractional ideal (15, a + 13), Fractional ideal (5)]
sage: b, f = basis_for_p_cokernel(S, C, 3); b
[Fractional ideal (50, a + 18),
 Fractional ideal (10, a + 3),
 Fractional ideal (3, a + 1),
 Fractional ideal (5)]
sage: b, f = basis_for_p_cokernel(S, C, 5); b
[Fractional ideal (2, a),
 Fractional ideal (2, a + 1),
 Fractional ideal (3, a + 1),
 Fractional ideal (5, a + 1),
 Fractional ideal (5, a + 3)]
```

```python
from sage.all import *
from sage.rings.number_field.selmer_group import basis_for_p_cokernel
x = polygen(ZZ, 'x')
K = NumberField(x**2 - x + 5, names=('a',)); (a,) = K._first_ngens(1)
S = K.ideal(30).support(); S
[Fractional ideal (2, a),
 Fractional ideal (2, a + 1),
 Fractional ideal (3, a + 1),
 Fractional ideal (5, a + 1),
 Fractional ideal (5, a + 3)]
C = K.class_group()
C.gens_orders()
(6, 2)
```

```
sage: [C(P).exponents() for P in S]
[(5, 0), (1, 0), (3, 1), (1, 1), (5, 1)]
sage: b, f = basis_for_p_cokernel(S, C, 2); b
[Fractional ideal (2), Fractional ideal (15, a + 13), Fractional ideal (5)]
sage: b, f = basis_for_p_cokernel(S, C, 3); b
[Fractional ideal (50, a + 18),
 Fractional ideal (10, a + 3),
 Fractional ideal (3, a + 1),
 Fractional ideal (5)]
sage: b, f = basis_for_p_cokernel(S, C, 5); b
[Fractional ideal (2, a),
 Fractional ideal (2, a + 1),
 Fractional ideal (3, a + 1),
 Fractional ideal (5, a + 1),
 Fractional ideal (5, a + 3)]
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.selmer_group import basis_for_p_cokernel
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**2 - x + 5, names=('a',)); (a,) = K._first_ngens(1)
>>> S = K.ideal(30).support(); S
[Fractional ideal (2, a),
 Fractional ideal (2, a + 1),
 Fractional ideal (3, a + 1),
 Fractional ideal (5, a + 1),
 Fractional ideal (5, a + 3)]
>>> C = K.class_group()
>>> C.gens_orders()
(6, 2)
```

```python
>>> [C(P).exponents() for P in S]
[(5, 0), (1, 0), (3, 1), (1, 1), (5, 1)]
>>> b, f = basis_for_p_cokernel(S, C, 2); b
[Fractional ideal (2), Fractional ideal (15, a + 13), Fractional ideal (5)]
>>> b, f = basis_for_p_cokernel(S, C, 3); b
[Fractional ideal (50, a + 18),
 Fractional ideal (10, a + 3),
 Fractional ideal (3, a + 1),
 Fractional ideal (5)]
>>> b, f = basis_for_p_cokernel(S, C, 5); b
[Fractional ideal (2, a),
 Fractional ideal (2, a + 1),
 Fractional ideal (3, a + 1),
 Fractional ideal (5, a + 1),
 Fractional ideal (5, a + 3)]
```

sage.rings.number_field.selmer_group.coords_in_U_mod_p(u, U, p)

Return coordinates of a unit \(u\) with respect to a basis of the \(p\)-cotorsion \(U/U^p\) of the unit group \(U\).

**INPUT:**

- \(u\) (algebraic unit) – a unit in a number field \(K\).
- \(U\) (unit group) – the unit group of \(K\).
- \(p\) (prime) – a prime number.

**OUTPUT:**
The coordinates of the unit \( u \) in the \( p \)-cotorsion group \( U/U_p \).

ALGORITHM:

Take the coordinate vector of \( u \) with respect to the generators of the unit group, drop the coordinate of the roots of unity factor if it is prime to \( p \), and reduce the vector mod \( p \).

EXAMPLES:

```python
sage: from sage.rings.number_field.selmer_group import coords_in_U_mod_p
sage: x = polygen(ZZ, 'x')

sage: K.<a> = NumberField(x^4 - 5*x^2 + 1)
sage: U = K.unit_group()
sage: U
Unit group with structure C2 x Z x Z x Z of Number Field in a with defining polynomial x^4 - 5*x^2 + 1

sage: u0, u1, u2, u3 = U.gens_values()
sage: u = u1*u2^2*u3^3

sage: coords_in_U_mod_p(u, U, 2)
[0, 1, 0, 1]
sage: coords_in_U_mod_p(u, U, 3)
[1, 2, 0]

sage: u*=u0
sage: coords_in_U_mod_p(u, U, 2)
[1, 1, 0, 1]
sage: coords_in_U_mod_p(u, U, 3)
[1, 2, 0]
```

\[
\text{sage.rings.number_field.selmer_group.} \text{pSelmerGroup}(K, S, p, \text{proof=None, debug=False})
\]

Return the \( p \)-Selmer group \( K(S, p) \) of the number field \( K \) with respect to the prime ideals in \( S \).

INPUT:

- \( K \) – a number field or \( \mathbb{Q} \).
- \( S \) – a list of prime ideals in \( K \), or prime numbers when \( K \) is \( \mathbb{Q} \).
- \( p \) – a prime number.
• proof – if True, compute the class group provably correctly. Default is True. Call proof.
  number_field() to change this default globally.
• debug – (boolean, default False) debug flag.

OUTPUT:
(tuple) KSp, KSp_gens, from_KSp, to_KSp where
• KSp is an abstract vector space over \( GF(p) \) isomorphic to \( K(S, p) \);
• KSp_gens is a list of elements of \( K^* \) generating \( K(S, p) \);
• from_KSp is a function from KSp to \( K^* \) implementing the isomorphism from the abstract \( K(S, p) \) to
  \( K(S, p) \) as a subgroup of \( K^*/(K^*)^p \);
• to_KSp is a partial function from \( K^* \) to KSp, defined on elements \( a \) whose image in \( K^*/(K^*)^p \) lies in
  \( K(S, p) \), mapping them via the inverse isomorphism to the abstract vector space KSp.

ALGORITHM:
The list of generators of \( K(S, p) \) is the concatenation of three sublists, called alphalist, betalist and
ulist in the code. Only alphalist depends on the primes in \( S \).

• ulist is a basis for \( U/U^p \) where \( U \) is the unit group. This is the list of fundamental units, including the
generator of the group of roots of unity if its order is divisible by \( p \). These have valuation 0 at all primes.
• betalist is a list of the generators of the \( p \)'th powers of ideals which generate the \( p \)-torsion in the class
  group (so is empty if the class number is prime to \( p \)). These have valuation divisible by \( p \) at all primes.
• alphalist is a list of generators for each ideal \( A \) in a basis of those ideals supported on \( S \) (modulo \( p \)th
  powers of ideals) which are \( p \)'th powers in the class group. We find \( B \) such that \( A/B^p \) is principal and take
  a generator of it, for each \( A \) in a generating set. As a special case, if all the ideals in \( S \) are principal then
  alphalist is a list of their generators.

The map from the abstract space to \( K^* \) is easy: we just take the product of the generators to powers given by the
coefficient vector. No attempt is made to reduce the resulting product modulo \( p \)'th powers.

The reverse map is more complicated. Given \( a \in K^* \):

• write the principal ideal \( (a) \) in the form \( AB^p \) with \( A \) supported by \( S \) and \( p \)'th power free. If this fails, then
  \( a \) does not represent an element of \( K(S, p) \) and an error is raised.
• set \( I_S \) to be the group of ideals spanned by \( S \) mod \( p \)'th powers, and \( I_{S, p} \) the subgroup of \( I_S \) which maps to 0
  in \( C/C^p \).
• Convert \( A \) to an element of \( I_{S, p} \), hence find the coordinates of \( a \) with respect to the generators in alphal-
  list.
• after dividing out by \( A \), now \( (a) = B^p \) (with a different \( a \) and \( B \)). Write the ideal class \( [B] \), whose \( p \)'th power
  is trivial, in terms of the generators of \( C[p] \); then \( B = (b)B_1 \), where the coefficients of \( B_1 \) with respect to
  generators of \( C[p] \) give the coordinates of the result with respect to the generators in betalist.
• after dividing out by \( B \), and by \( b^p \), we now have \( (a) = (1) \), so \( a \) is a unit, which can be expressed in terms
  of the unit generators.

EXAMPLES:
Over \( \mathbb{Q} \) the unit contribution is trivial unless \( p = 2 \) and the class group is trivial:

```sage
def from sage.rings.number_field.selmer_group import pSelmerGroup
sage: QS2, gens, fromQS2, toQS2 = pSelmerGroup(QQ, [2, 3], 2)
sage: QS2
Vector space of dimension 3 over Finite Field of size 2
```
A real quadratic field with class number 2, where the fundamental unit is a generator, and the class group provides another generator when $p = 2$:
Vector space of dimension 4 over Finite Field of size 2
\[ (a + 1, a, 2, -1) \]

```python
from sage.all import *

K = QuadraticField(-Integer(5), names=('a',)); (a,) = K._first_ngens(1)

K.class_number()
2

P2 = K.ideal(Integer(2), -a+Integer(1))
P3 = K.ideal(Integer(3), a+Integer(1))
P5 = K.ideal(a)

KS2, gens, fromKS2, toKS2 = pSelmerGroup(K, [P2, P3, P5], Integer(2))

KS2
Vector space of dimension 4 over Finite Field of size 2

sage: gens
[a + 1, a, 2, -1]

Each generator must have even valuation at primes not in \( S \):

```python
sage: [K.ideal(g).factor() for g in gens]
[(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1)),
 Fractional ideal (a),
 (Fractional ideal (2, a + 1))^2,
 1]

sage: toKS2(10)
(0, 0, 1, 1)

sage: fromKS2([0,0,1,1])
-2

sage: K(10/(-2)).is_square()
True

sage: KS3, gens, fromKS3, toKS3 = pSelmerGroup(K, [P2, P3, P5], 3)

sage: KS3
Vector space of dimension 3 over Finite Field of size 3

sage: gens
[1/2, 1/4*a + 1/4, a]
```

```python
from sage.all import *

[K.ideal(g).factor() for g in gens]
[(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1)),
 Fractional ideal (a),
 (Fractional ideal (2, a + 1))^2,
 1]

> toKS2(Integer(10))
(0, 0, 1, 1)

> fromKS2([Integer(0),Integer(0),Integer(1),Integer(1)])
-2

> K(Integer(10)/(-Integer(2))).is_square()
True

> KS3, gens, fromKS3, toKS3 = pSelmerGroup(K, [P2, P3, P5], Integer(3))

> KS3
Vector space of dimension 3 over Finite Field of size 3

> gens
```
The \texttt{to} and \texttt{from} maps are inverses of each other:

```python
sage: K.<a> = QuadraticField(-5)
sage: S = K.ideal(30).support()
sage: KS2, gens, fromKS2, toKS2 = pSelmerGroup(K, S, 2)
sage: KS2
Vector space of dimension 5 over Finite Field of size 2
sage: assert all(toKS2(fromKS2(v))==v for v in KS2)
sage: KS3, gens, fromKS3, toKS3 = pSelmerGroup(K, S, 3)
sage: KS3
Vector space of dimension 4 over Finite Field of size 3
sage: assert all(toKS3(fromKS3(v))==v for v in KS3)
```

```python
>>> from sage.all import *
>>> K = QuadraticField(-Integer(5), names=('a',)); (a,) = K._first_ngens(1)
>>> S = K.ideal(Integer(30)).support()
>>> KS2, gens, fromKS2, toKS2 = pSelmerGroup(K, S, Integer(2))
>>> KS2
Vector space of dimension 5 over Finite Field of size 2
>>> assert all(toKS2(fromKS2(v))==v for v in KS2)
>>> KS3, gens, fromKS3, toKS3 = pSelmerGroup(K, S, Integer(3))
>>> KS3
Vector space of dimension 4 over Finite Field of size 3
>>> assert all(toKS3(fromKS3(v))==v for v in KS3)
```
5.1 Algebraic numbers

This module implements the algebraic numbers (the complex numbers which are the zero of a polynomial in $\mathbb{Z}[x]$; in other words, the algebraic closure of $\mathbb{Q}$, with an embedding into $\mathbb{C}$). All computations are exact. We also include an implementation of the algebraic reals (the intersection of the algebraic numbers with $\mathbb{R}$). The field of algebraic numbers $\overline{\mathbb{Q}}$ is available with abbreviation $\texttt{QQbar}$; the field of algebraic reals has abbreviation $\texttt{AA}$.

As with many other implementations of the algebraic numbers, we try hard to avoid computing a number field and working in the number field; instead, we use floating-point interval arithmetic whenever possible (basically whenever we need to prove non-equalities), and resort to symbolic computation only as needed (basically to prove equalities).

Algebraic numbers exist in one of the following forms:

- a rational number
- the sum, difference, product, or quotient of algebraic numbers
- the negation, inverse, absolute value, norm, real part, imaginary part, or complex conjugate of an algebraic number
- a particular root of a polynomial, given as a polynomial with algebraic coefficients together with an isolating interval (given as a $\texttt{RealIntervalFieldElement}$) which encloses exactly one root, and the multiplicity of the root
- a polynomial in one generator, where the generator is an algebraic number given as the root of an irreducible polynomial with integral coefficients and the polynomial is given as a $\texttt{NumberFieldElement}$.

An algebraic number can be coerced into $\texttt{ComplexIntervalField}$ (or $\texttt{RealIntervalField}$, for algebraic reals); every algebraic number has a cached interval of the highest precision yet calculated.

In most cases, computations that need to compare two algebraic numbers compute them with 128-bit precision intervals; if this does not suffice to prove that the numbers are different, then we fall back on exact computation.

Note that division involves an implicit comparison of the divisor against zero, and may thus trigger exact computation.

Also, using an algebraic number in the leading coefficient of a polynomial also involves an implicit comparison against zero, which again may trigger exact computation.

Note that we work fairly hard to avoid computing new number fields; to help, we keep a lattice of already-computed number fields and their inclusions.

EXAMPLES:

```
sage: sqrt(AA(2)) > 0
True
sage: (sqrt(5 + 2*sqrt(QQbar(6))) - sqrt(QQbar(3)))^2 == 2
True
sage: AA((sqrt(5 + 2*sqrt(6)) - sqrt(3))^2) == 2
#...
```
Algebraic Numbers and Number Fields, Release 10.4

needs sage.symbolic
True

from sage.all import *
from sage.all import *
sqrt(AA(Integer(2))) > Integer(0)
True

(sqrt(Integer(5) + Integer(2)*sqrt(QQbar(Integer(6)))) -
→ sqrt(QQbar(Integer(3))))**Integer(2) == Integer(2)
True

(sqrt(Integer(5) + Integer(2)*sqrt(Integer(6))) -
→ sqrt(Integer(3)))**Integer(2) == Integer(2)
needs sage.symbolic
True

For a monic cubic polynomial $x^3 + bx^2 + cx + d$ with roots $s_1, s_2, s_3$, the discriminant is defined as $(s_1 - s_2)^2(s_1 - s_3)^2(s_2 - s_3)^2$ and can be computed as $b^2c^2 - 4b^3d - 4c^3 + 18bcd - 27d^2$. We can test that these definitions do give the same result:

```
sage: def disc1(b, c, d):
    ... return b^2*c^2 - 4*b^3*d - 4*c^3 + 18*b*c*d - 27*d^2
sage: def disc2(s1, s2, s3):
    ... return ((s1-s2)*(s1-s3)*(s2-s3))^2
sage: x = polygen(AA)
sage: p = x*(x-2)*(x-4)
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(-1, 1))
sage: s2 = AA.polynomial_root(cp, RIF(1, 3))
sage: s3 = AA.polynomial_root(cp, RIF(3, 5))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
sage: p = p + 1
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(-1, 1))
sage: s2 = AA.polynomial_root(cp, RIF(1, 3))
sage: s3 = AA.polynomial_root(cp, RIF(3, 5))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
sage: p = (x-sqrt(AA(2)))*(x-AA(2).nth_root(3))*(x-sqrt(AA(3)))
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(1.4, 1.5))
sage: s2 = AA.polynomial_root(cp, RIF(1.7, 1.8))
sage: s3 = AA.polynomial_root(cp, RIF(1.2, 1.3))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
```

(continues on next page)
>>> cp = AA.common_polynomial(p)
>>> d, c, b, _ = p.list()
>>> s1 = AA.polynomial_root(cp, RIF(-Integer(1), Integer(1)))
>>> s2 = AA.polynomial_root(cp, RIF(Integer(1), Integer(3)))
>>> s3 = AA.polynomial_root(cp, RIF(Integer(3), Integer(5)))
>>> disc1(b, c, d) == disc2(s1, s2, s3)
True

>>> p = p + Integer(1)
>>> cp = AA.common_polynomial(p)
>>> d, c, b, _ = p.list()
>>> s1 = AA.polynomial_root(cp, RIF(RealNumber(1.4), RealNumber(1.5)))
>>> s2 = AA.polynomial_root(cp, RIF(RealNumber(1.7), RealNumber(1.8)))
>>> s3 = AA.polynomial_root(cp, RIF(RealNumber(1.2), RealNumber(1.3)))
>>> disc1(b, c, d) == disc2(s1, s2, s3)
True

We can convert from symbolic expressions:

```
sage: # needs sage.symbolic
sage: QQbar(sqrt(-Integer(5)))
2.236067977499790?*I
sage: AA(sqrt(Integer(2)) + sqrt(Integer(3)))
3.146264369941973?
```

5.1. Algebraic numbers 603
The coercion, however, goes in the other direction, since not all symbolic expressions are algebraic numbers:

```python
sage: QQbar(sqrt(2)) + sqrt(3)  # needs sage.symbolic
sqrt(3) + 1.414213562373095?
sage: QQbar(sqrt(2) + QQbar(sqrt(3)))  # needs sage.symbolic
3.146264369941973?
```

Note the different behavior in taking roots: for `AA` we prefer real roots if they exist, but for `QQbar` we take the principal root:

```python
sage: AA(-1)^(1/3)
-1
sage: QQbar(-1)^(1/3)
0.500000000000000? + 0.866025403784439?*I
```

However, implicit coercion from `QQbar` is only allowed when it is equipped with a complex embedding:

```python
from sage.all import *

>>> QQbar(I) + i
I + 1
```

(continues on next page)
sage: K.<im> = QuadraticField(-1, embedding=None)
sage: QQbar(1) + im
Traceback (most recent call last):
...
TypeError: unsupported operand parent(s) for +: 'Algebraic Field' and 'Number Field in im with defining polynomial x^2 + 1'

However, we can explicitly coerce from the abstract number field \(\mathbb{Q}[I]\). (Technically, this is not quite kosher, since we do not know whether the field generator is supposed to map to \(+I\) or \(-I\). We assume that for any quadratic field with polynomial \(x^2 + 1\), the generator maps to \(+I\):

sage: pythag = QQbar(3/5 + 4*im/5); pythag
4/5*I + 3/5
sage: pythag.abs() == 1
True

We can implicitly coerce from algebraic reals to algebraic numbers:

sage: a = QQbar(1); a, a.parent()
(1, Algebraic Field)
sage: b = AA(1); b, b.parent()
(1, Algebraic Real Field)
sage: c = a + b; c, c.parent()
(2, Algebraic Field)

Some computation with radicals:

5.1. Algebraic numbers
The Sage rings $\text{AA}$ and $\text{QQbar}$ can decide equalities between radical expressions (over the reals and complex numbers respectively):

```python
sage: a = AA((2/(3*sqrt(3)) + 10/27)**(1/3)
#   ← needs sage.symbolic
....: - 2/(9*(2/(3*sqrt(3)) + 10/27)**(1/3)) + 1/3)
sage: a
#   ← needs sage.symbolic
1.000000000000000?
```

```python
>>> from sage.all import *

>>> a = AA((Integer(2)/(Integer(3)*sqrt(Integer(3))) + Integer(10)/Integer(27))**(Integer(1)/Integer(3)))
#   ← needs sage.symbolic
....: - Integer(2)/(Integer(9)*(Integer(2)/(Integer(3)*sqrt(Integer(3))) + 10/27)**(Integer(1)/Integer(3))) + 1/3
```

```python
>>> a
#   ← needs sage.symbolic
1.000000000000000?
```

(continues on next page)
Algebraic numbers which are known to be rational print as rationals; otherwise they print as intervals (with 53-bit precision):

```
sage: AA(2)/3
2/3
sage: QQbar(5/7)
5/7
sage: QQbar(1/3 - 1/4*I)
-1/4*I + 1/3
sage: two = QQbar(4).nth_root(4)^2; two
2.000000000000000?
```

We can find the real and imaginary parts of an algebraic number (exactly):

```
sage: r = QQbar.polynomial_root(x^5 - x - 1, CIF(RIF(0.1, 0.2), RIF(1.0, 1.1))); r
0.1812324444698754? + 1.083954101317711?*I
```

5.1. Algebraic numbers
We can find the absolute value and norm of an algebraic number exactly. (Note that we define the norm as the product of a number and its complex conjugate; this is the algebraic definition of norm, if we view \( \mathbb{Q} \bar{\mathbb{Q}} \) as \( \mathbb{A}[I] \):

\[
\text{sage: } R.<x> = \mathbb{Q}[x] \text{; } (x,) = R._\text{first}\_\text{ngens}(1) \\
\text{sage: } r = (x^3 + 8).\text{roots}(\mathbb{Q} \bar{\mathbb{Q}}, \text{multiplicities=\text{False}})[2]; r \\
1.000000000000000? + 1.732050807568878?\text{I} \\
\text{sage: } r.\text{abs()} == 2 \\
\text{True} \\
\text{sage: } r.\text{norm()} == 4 \\
\text{True} \\
\text{sage: } (r+\mathbb{Q} \bar{\mathbb{Q}}(\text{I})).\text{norm().minpoly()} \\
x^2 - 10*x + 13 \\
\text{sage: } r = \mathbb{A}[\text{polynomial_root}(x^2 - x - 1, \text{RIF}(-1, 0)); r \\
-0.618033988749895? \\
\text{sage: } r.\text{abs().minpoly()} \\
x^2 + x - 1
\]

We can compute the multiplicative order of an algebraic number:

\[
\text{sage: } \mathbb{Q} \bar{\mathbb{Q}}(-1/2 + \text{I}\sqrt{3}/2).\text{multiplicative\_order()}
\]

3

\[
\text{sage: } \mathbb{Q} \bar{\mathbb{Q}}(-\sqrt{3}/2 + \text{I}/2).\text{multiplicative\_order()}
\]

12

\[
\text{sage: } (\mathbb{Q} \bar{\mathbb{Q}}.\text{zeta}(23)**5).\text{multiplicative\_order()}
\]

23
The paper “ARPREC: An Arbitrary Precision Computation Package” by Bailey, Yozo, Li and Thompson discusses this result. Evidently it is difficult to find, but we can easily verify it.

```python
sage: alpha = QQbar.polynomial_root(x^10 + x^9 - x^7 - x^6 - x^5 - x^4 - x^3 + x + 1, RIF(1, 1.2))
```

```
sage: lhs = alpha^630 - 1
sage: rhs_num = (alpha^315 - 1) * (alpha^210 - 1) * (alpha^126 - 1)^2 * (alpha^90 - 1) * (alpha^3 - 1)^3 * (alpha^2 - 1)^5 * (alpha - 1)^3
sage: rhs_den = (alpha^35 - 1) * (alpha^15 - 1)^2 * (alpha^14 - 1)^2 * (alpha^5 - 1)^6 * alpha^68
sage: rhs = rhs_num / rhs_den
sage: lhs
2.642040335819351?e44
sage: rhs
2.642040335819351?e44
sage: lhs - rhs
0.?e29
sage: lhs == rhs
True
sage: lhs - rhs
0
sage: lhs._exact_value()
-10648699402510886229334132989629606002223831*a^9 +
-2317456024910022861371818312802529035435800*a^8 -
-27257910625422520555847364695458901256*a^7 -
-21146499900465237691295705441004410518065*a^6 -
-14543082864016871805545108986578337637140321*a^5 -
-6458500500879664339372667222920521216589785*a^4 -
-30522190538007844912280171454923124998263*a^3 -
-1423896612862335681821644902045640915516176*a^2 -
-167490227832522054673732621893920439216101*a -
-9052854758155149578737247156588012516273410 where a^10 - a^9 + a^7 - a^6 + a^5 - a^4 + a^3 - a + 1 = 0 and a in -1.176280818259918
```
Given an algebraic number, we can produce a string that will reproduce that algebraic number if you type the string into Sage. We can see that until exact computation is triggered, an algebraic number keeps track of the computation steps used to produce that number:

```python
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: n = (rt2 + rt3)**5; n
308.3018001722975?
sage: sage_input(n)
R.<x> = AA[
    AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951)))
    + AA.polynomial_root(AA.common_polynomial(x^2 - 3), RIF(RR(1.7320508075688772), RR(1.7320508075688774)))
    v2 = v1*v1
    v2*v2*v1

>>> from sage.all import *
>>> rt2 = AA(sqrt(Integer(2)))
>>> rt3 = AA(sqrt(Integer(3)))
>>> n = (rt2 + rt3)**Integer(5); n
308.3018001722975?
>>> sage_input(n)
R.<x> = AA[
    AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951)))
    + AA.polynomial_root(AA.common_polynomial(x^2 - 3), RIF(RR(1.7320508075688772), RR(1.7320508075688774)))
    v2 = v1*v1
    v2*v2*v1
```

But once exact computation is triggered, the computation tree is discarded, and we get a way to produce the number directly:
We can also see that some computations (basically, those which are easy to perform exactly) are performed directly, instead of storing the computation tree:

```python
sage: z3_3 = QQbar.zeta(3) * 3
sage: z4_4 = QQbar.zeta(4) * 4
sage: z5_5 = QQbar.zeta(5) * 5
sage: sage_input(z3_3 * z4_4 * z5_5)
R.<y> = QQ[]
3*QQbar.polynomial_root(AA.common_polynomial(y^2 + y + 1), CIF(RIF(-RR(0.
˓→500000000000000001), -RR(0.499999999999999994)), RR(0.
˓→86602540378443871))))*QQbar(4*I)*(5*QQbar.polynomial_root(AA.common_polynomial(y^4
˓→+ y^3 + y^2 + y + 1), CIF(RIF(RR(0.3090169943749474), RR(0.30901699437494745)),
˓→RIF(RR(0.95105651629515353), RR(0.95105651629515364))))
```

Note that the `verify=True` argument to `sage_input` will always trigger exact computation, so running `sage_input` twice in a row on the same number will actually give different answers. In the following, running `sage_input` on `n` will also trigger exact computation on `rt2`, as you can see by the fact that the third output is different than the first:

```python
sage: # needs sage.symbolic
sage: rt2 = AA(sqrt(2))
sage: n = rt2^2
sage: sage_input(n, verify=True)
# Verified
R.<x> = AA[]
v = AA.polynomial_root(AA.common_polynomial(x^2 - 2), CIF(RR(1.4142135623730949),
˓→RR(1.4142135623730951)))
v*v
```

(continues on next page)
Just for fun, let's try `sage_input` on a very complicated expression. The output of this example changed with the rewriting of polynomial multiplication algorithms in Issue #10255:

```python
sage: rt2 = sqrt(AA(Integer(2)))
```

```python
sage: rt3 = sqrt(QQbar(Integer(3)))
```

```python
sage: x = polygen(QQbar)
```

```python
sage: nrt3 = AA.polynomial_root((x-rt2)*(x+rt3), RIF(-Integer(2), -Integer(1)))
```

```python
sage: one = AA.polynomial_root((x-rt2)*(x-rt3)*(x-nrt3)*(x-Integer(1)-rt3-nrt3), RIF(RealNumber(0.9), RealNumber(1.1)))
```

```python
sage: one
```

1
```
1.000000000000000?

```python
>>> sage_input(one, verify=True)
# Verified
R1.<x> = QQbar[]
R2.<y> = QQ[]
v = AA.polynomial_root(AA.common_polynomial(y^4 - 4*y^2 + 1), RIF(-RR(1.9318516525781364), -RR(1.9318516525781366))
AA.polynomial_root(AA.common_polynomial(x^4 + QQbar(v^3 - 3*v - 1)*x^3 + QQbar(-v^3 + 3*v - 3)*x^2 + QQbar(-3*v^3 + 9*v + 3)*x + QQbar(3*v^3 - 9*v)), RIF(RR(0.99999999999999989), RR(1.0000000000000002)))
```

```python
>>> one
1
```

We can pickle and unpickle algebraic fields (and they are globally unique):

```python
sage: loads(dumps(AlgebraicField())) is AlgebraicField()
True
sage: loads(dumps(AlgebraicRealField())) is AlgebraicRealField()
True
```

```python
>>> from sage.all import *
```

```python
>>> loads(dumps(AlgebraicField())) is AlgebraicField()
True
>>> loads(dumps(AlgebraicRealField())) is AlgebraicRealField()
True
```

We can pickle and unpickle algebraic numbers:

```python
sage: loads(dumps(QQbar(10))) == QQbar(10)
True
sage: loads(dumps(QQbar(5/2))) == QQbar(5/2)
True
sage: loads(dumps(QQbar.zeta(5))) == QQbar.zeta(5)
True
```

```python
sage: # needs sage.symbolic
sage: t = QQbar(sqrt(2)); type(t._descr)
<class 'sage.rings.qqbar.ANRoot'>
sage: loads(dumps(t)) == QQbar(sqrt(2))
True
```

```python
sage: t = ~QQbar(sqrt(2)); type(t._descr)
<class 'sage.rings.qqbar.ANUnaryExpr'>
sage: loads(dumps(t)) == 1/QQbar(sqrt(2))
True
```

```python
sage: t = QQbar(sqrt(2)) + QQbar(sqrt(3)); type(t._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: loads(dumps(t)) == QQbar(sqrt(2)) + QQbar(sqrt(3))
True
```

```python
>>> from sage.all import *
```

```python
>>> loads(dumps(QQbar(Integer(10)))) == QQbar(Integer(10))
True
```

5.1. Algebraic numbers
### We can convert elements of $\mathbb{Q}\bar{\mathbb{Q}}$ and $\mathbb{A}\mathbb{A}$ into the following types: float, complex, RDF, CDF, RR, CC, RIF, CIF, ZZ, and QQ, with a few exceptions. (For the arbitrary-precision types, RR, CC, RIF, and CIF, it can convert into a field of arbitrary precision.)

Converting from $\mathbb{Q}\bar{\mathbb{Q}}$ to a real type (float, RDF, RR, RIF, ZZ, or QQ) succeeds only if the $\mathbb{Q}\bar{\mathbb{Q}}$ is actually real (has an imaginary component of exactly zero). Converting from either AA or $\mathbb{Q}\bar{\mathbb{Q}}$ to ZZ or QQ succeeds only if the number actually is an integer or rational. If conversion fails, a ValueError will be raised.

Here are examples of all of these conversions:

```python
sage: # needs sage.symbolic
sage: all_vals = [AA(42), AA(22/7), AA(golden_ratio),
               QQbar(-13), QQbar(89/55), QQbar(~sqrt(7)), QQbar.zeta(5)]

sage: def convert_test_all(ty):
    ....:     def convert_test(v):
    ....:         try:
    ....:             return ty(v)
    ....:         except (TypeError, ValueError):
    ....:             return None
    ....:     return [convert_test(_) for _ in all_vals]

sage: convert_test_all(float)
[42.0, 6.283185307179586, 1.618033988749895, -13.0, 1.6181818181818182, -2.6457513110645907, None]

sage: convert_test_all(complex)
[(42+0j), (6.283185307179586+0j), (1.618033988749895+0j), (-13+0j), (1.6181818181818182+0j), (-2.6457513110645907+0j), (0.30901699437494745+0.9510565162951536j)]

sage: convert_test_all(RDF)
[42.0, 6.283185307179586, 1.618033988749895, -13.0, 1.6181818181818182, -2.6457513110645907, None]

sage: convert_test_all(CDF)
[(42+0j), (6.283185307179586+0j), (1.618033988749895+0j), (-13+0j), (1.6181818181818182+0j), (-2.6457513110645907+0j), (0.30901699437494745+0.9510565162951536*I)]

sage: convert_test_all(RR)
[42.0, 6.283185307179586, 1.618033988749895, -13.0, 1.6181818181818182, -2.6457513110645907, None]

```
5.1. Algebraic numbers

>>> from sage.all import *
>>> # needs sage.symbolic
>>> all_vals = [AA(Integer(42)), AA(Integer(22)/Integer(7)), AA(golden_ratio),
... QQbar(-Integer(13)), QQbar(Integer(89)/Integer(55)), QQbar(-
... sqrt(Integer(7))), QQbar.zeta(Integer(5))]
>>> def convert_test_all(ty):
...     def convert_test(v):
...         try:
...             return ty(v)
...         except (TypeError, ValueError):
...             return None
...     return [convert_test(_)
...            for _ in all_vals]
>>> convert_test_all(float)
[42.0, 3.1428571428571432, 1.6180339887498999, -13.0, 1.6181818181818182, -2.6457513110645907, None]
>>> convert_test_all(complex)
[(42+0j), (3.1428571428571432+0j), (1.618033988749895+0j), (-13+0j), (1.618181818181819+0j), (-2.6457513110645907+0j), (0.30901699437494745+0.9510565162951536j)]
Compute the exact coordinates of a 34-gon (the formulas used are from Weisstein, Eric W. “Trigonometry Angles–Pi/17.” and can be found at http://mathworld.wolfram.com/TrigonometryAnglesPi17.html):

```
sage: rt17 = AA(17).sqrt()
sage: rt2 = AA(2).sqrt()
sage: eps = (17 + rt17).sqrt()
sage: epss = (17 - rt17).sqrt()
sage: delta = rt17 - 1
sage: alpha = (34 + 6*rt17 + rt2*delta*epss - 8*rt2*eps).sqrt()
sage: beta = 2*(17 + 3*rt17 - 2*rt2*eps - rt2*epss).sqrt()
sage: x = rt2*(15 + rt17 + rt2*(alpha + epss)).sqrt()/8
sage: y = rt2*(epss**2 - rt2*(alpha + epss)).sqrt()/8
sage: cx, cy = 1, 0
sage: for i in range(34):
      ..:  cx, cy = x*cx-y*cy, x*cy+y*cx
sage: cx
1.000000000000000?
sage: cy
0.?e-15
sage: ax = polygen(AA)
sage: x2 = AA.polynomial_root(256*ax**8 - 128*ax**7 - 448*ax**6 + 192*ax**5
      + 240*ax**4 - 80*ax**3 - 40*ax**2 + 8*ax + 1,
      RIF(0.9829, 0.983))
```

(continues on next page)
Ideally, in the above example we should be able to test \( x == x_2 \) and \( y == y_2 \) but this is currently infinitely long.

AUTHOR:

- Carl Witty (2007-01-27): initial version
- Carl Witty (2007-10-29): massive rewrite to support complex as well as real numbers

```python
class sage.rings.qqbar.ANBinaryExpr(left, right, op)
Bases: ANDescr

Initialize this ANBinaryExpr.

EXAMPLES:

```
sage: sage_input(2 + (-1*sqrt(AA(2))), verify=True)  
# Verified
R.<x> = AA[]
2 - AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951)))
sage: sage_input(2*sqrt(AA(2)), verify=True)  
# Verified
R.<x> = AA[]
2*AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951)))
sage: rt2 = sqrt(AA(2))
sage: one = rt2/rt2
sage: n = one+3
sage: sage_input(n)  
R.<x> = AA[]
v = AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951)))
v/v + 3
sage: one == 1
True
sage: sage_input(n)
1 + AA(3)
sage: rt3 = QQbar(sqrt(3))  
# needs sage.symbolic
sage: one = rt3/rt3  
# needs sage.symbolic
sage: n = sqrt(AA(2)) + one
sage: one == 1  
# needs sage.symbolic
True
sage: sage_input(n)  
R.<x> = AA[]
QQbar.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951))) + 1
sage: from sage.rings.qqbar import *  
from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: binexp = ANBinaryExpr(AA(3), AA(5), operator.mul)
sage: binexp.handle_sage_input(sib, False, False)
({binop:* {atomic:3} {call: {atomic:AA}({atomic:5})}}, True)
sage: binexp.handle_sage_input(sib, False, True)
({call: {atomic:QQbar}({binop:* {atomic:3} {call: {atomic:AA}({atomic:5})}})}, True)
is_complex() (continued from previous page)

Whether this element is complex. Does not trigger exact computation, so may return True even if the element is real.

5.1. Algebraic numbers 619
EXAMPLES:

```python
sage: x = (QQbar(sqrt(-2)) / QQbar(sqrt(-5)))._descr
    # needs sage.symbolic
sage: x.is_complex()
    # needs sage.symbolic
True
```

```python
>>> from sage.all import *
>>> x = (QQbar(sqrt(-Integer(2))) / QQbar(sqrt(-Integer(5))))._descr
    # needs sage.symbolic
>>> x.is_complex()
    # needs sage.symbolic
True
```

class `sage.rings.qqbar.ANDescr`

Bases: `SageObject`

An `AlgebraicNumber` or `AlgebraicReal` is a wrapper around an `ANDescr` object. `ANDescr` is an abstract base class, which should never be directly instantiated; its concrete subclasses are `ANRational`, `ANBinaryExpr`, `ANUnaryExpr`, `ANRoot`, and `ANExtensionElement`. `ANDescr` and all of its subclasses are for internal use, and should not be used directly.

abs(n)

Absolute value of self.

EXAMPLES:

```python
sage: a = QQbar(sqrt(2))
    # needs sage.symbolic
sage: b = a._descr
    # needs sage.symbolic
sage: b.abs(a)
    # needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

```python
>>> from sage.all import *
>>> a = QQbar(sqrt(Integer(2)))
    # needs sage.symbolic
>>> b = a._descr
    # needs sage.symbolic
>>> b.abs(a)
    # needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

conjugate(n)

Complex conjugate of self.

EXAMPLES:

```python
sage: a = QQbar(sqrt(-7))
    # needs sage.symbolic
sage: b = a._descr
    # needs sage.symbolic
sage: b.conjugate(a)
    # needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

```python
>>> from sage.all import *
>>> a = QQbar(sqrt(-Integer(7)))
    # needs sage.symbolic
>>> b = a._descr
    # needs sage.symbolic
>>> b.conjugate(a)
    # needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```
imag (n)

Imaginary part of self.

EXAMPLES:

$$\text{sage: } a = \text{QQbar}(\sqrt{-\text{Integer}(7)})$$

$$\text{# needs sage.symbolic}$$

$$\text{sage: } b = a._\text{descr}$$

$$\text{# needs sage.symbolic}$$

$$\text{sage: } b.\text{imag}(a)$$

$$\text{# needs sage.symbolic}$$

invert (n)

1/self.

EXAMPLES:

$$\text{sage: } a = \text{QQbar}(\sqrt{2})$$

$$\text{# needs sage.symbolic}$$

$$\text{sage: } b = a._\text{descr}$$

$$\text{# needs sage.symbolic}$$

$$\text{sage: } b.\text{invert}(a)$$

$$\text{# needs sage.symbolic}$$

is_simple ()

Check whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

5.1. Algebraic numbers 621
This returns True if self is an ANRational, or a minimal ANExtensionElement.

EXAMPLES:

```python
sage: from sage.rings.qqbar import ANRational
sage: ANRational(1/2).is_simple()
True

sage: # needs sage.symbolic
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2.exactify()
sage: rt2._descr.is_simple()
True
sage: rt2b.exactify()
False
sage: rt2b.simplify()
True
```

>>> from sage.all import *
>>> from sage.rings.qqbar import ANRational
>>> ANRational(Integer(1)/Integer(2)).is_simple()
True

# needs sage.symbolic
>>> rt2 = AA(sqrt(Integer(2)))
>>> rt3 = AA(sqrt(Integer(3)))
>>> rt2b = rt3 + rt2 - rt3
>>> rt2.exactify()
>>> rt2._descr.is_simple()
True
>>> rt2b.exactify()
>>> rt2b._descr.is_simple()
False
>>> rt2b.simplify()
>>> rt2b._descr.is_simple()
True

neg \((n)\)
Negation of self.

EXAMPLES:

```python
sage: a = QQbar(sqrt(2))  # needs sage.symbolic
sage: b = a._descr  # needs sage.symbolic
sage: b.neg(a)  # needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

>>> from sage.all import *
>>> a = QQbar(sqrt(Integer(2)))  # needs sage.symbolic
>>> b = a._descr  # needs sage.symbolic
>>> b.neg(a)
... (continues on next page)
needs sage.symbolic

```python
b.neg(a)  #...
```

**norm**

Field norm of self from $\mathbb{Q}$ to its real subfield $A$, i.e. the square of the usual complex absolute value.

**EXAMPLES:**

```python
sage: a = QQbar(sqrt(-7))  #...
→ needs sage.symbolic
sage: b = a._descr  #...
→ needs sage.symbolic
sage: b.norm(a)  #...
→ needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

```python
>>> from sage.all import *
```

```python
a = QQbar(sqrt(-Integer(7)))  # needs sage.symbolic
b = a._descr  # needs sage.symbolic
b.norm(a)  # needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

**real**

Real part of self.

**EXAMPLES:**

```python
sage: a = QQbar(sqrt(-7))  #...
→ needs sage.symbolic
sage: b = a._descr  #...
→ needs sage.symbolic
sage: b.real(a)  #...
→ needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

```python
>>> from sage.all import *
```

```python
a = QQbar(sqrt(-Integer(7)))  # needs sage.symbolic
b = a._descr  # needs sage.symbolic
b.real(a)  # needs sage.symbolic
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

**class sage.rings.qqbar.ANExtensionElement**

**Bases:** ANDescr

The subclass of `ANDescr` that represents a number field element in terms of a specific generator. Consists of a polynomial with rational coefficients in terms of the generator, and the generator itself, an `AlgebraicGenerator`.  

---

**5.1. Algebraic numbers**
abs \((n)\)

Return the absolute value of \(self\) (square root of the norm).

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.abs(a)
Root 3.146264369941972342? of x^2 - 9.89897948556636?
```

```python
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-Integer(2))) + QQbar(sqrt(-Integer(3)))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.abs(a)
Root 3.146264369941972342? of x^2 - 9.89897948556636?
```

conjugate \((n)\)

Complex conjugate of \(self\).

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: c = b.conjugate(None); c
1/3*a^3 - 1/3*a^2 + a + 1 where a^4 - 2*a^3 + a^2 + 6*a + 3 = 0
and a in 1.724744871391589? - 1.573132184970987?*I
```

```python
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-Integer(2))) + QQbar(sqrt(-Integer(3)))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: c = b.conjugate(None); c
1/3*a^3 - 1/3*a^2 + a + 1 where a^4 - 2*a^3 + a^2 + 6*a + 3 = 0
and a in 1.724744871391589? - 1.573132184970987?*I
```

Internally, complex conjugation is implemented by taking the same abstract field element but conjugating the complex embedding of the field:

```python
sage: c.generator() == b.generator().conjugate()  # needs sage.symbolic
True
sage: c.field_element_value() == b.field_element_value()  # needs sage.symbolic
```

(continues on next page)
needs sage.symbolic
True

>>> from sage.all import *
>>> c.generator() == b.generator().conjugate()       # _
→ needs sage.symbolic
True
>>> c.field_element_value() == b.field_element_value()  # 
→ needs sage.symbolic
True

The parameter is ignored:

sage: (b.conjugate("random").generator() == c.generator())  # _
→ needs sage.symbolic
.... and b.conjugate("random").field_element_value() == c.field_element_value()  
→ value()
True

exactify()
Return an exact representation of self.

Since self is already exact, just return self.

EXAMPLES:

sage: x = polygen(ZZ, 'x')
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.exactify()  
sage: type(v)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: v.exactify() is v
True

field_element_value()
Return the underlying number field element.

EXAMPLES:
generator()

Return the `AlgebraicGenerator` object corresponding to `self`.

EXAMPLES:

```
sage: x = polygen(ZZ, 'x')
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._ descr.
    → exactify()
sage: v.generator()
Number Field in a with defining polynomial y^2 - y - 1 with a in 1.
  → 618039887469895?
```

handle_sage_input `(sib, coerce, is_qqbar)`

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always True for `ANExtensionElement`).

EXAMPLES:

```
sage: I = QQbar(I)
sage: sage_input(3+4*I, verify=True)
# Verified
QQbar(3 + 4*I)
sage: v = QQbar.zeta(3) + QQbar.zeta(5)
sage: v - v == 0
True
sage: sage_input(vector(QQbar, (4-3*I, QQbar.zeta(7))), verify=True)
# Verified
R.<y> = QQ[]
vector(QQbar, [4 - 3*I, QQbar.polynomial_root(AA.common_polynomial(y^6 + y^5
    → + y^4 + y^3 + y^2 + y + 1), CIF(RIF(RR(0.62348980185873348), RR(0.
    → 0.62348980185873359)), CIRF(RR(0.718314824680298), RR(0.718314824680298))))])
sage: sage_input(v, verify=True)
# Verified
R.<y> = QQ[]
```

(continues on next page)
v = QQbar.polynomial_root(AA.common_polynomial(y^8 - y^7 + y^5 - y^4 + y^3 -
-> y + 1), CIF(RIF(RR(0.91354545764260087), RR(0.91354545764260098)), RIF(RR(0.
-> 40673664307580015), RR(0.40673664307580021))))
v^5 + v^3

sage: v = QQbar(sqrt(AA(Integer(2))))
sage: v.exactify()
# Verified
R.<y> = QQ[]
QQbar(AA.polynomial_root(AA.common_polynomial(y^2 - 2), RIF(RR(1.
-> 4142135623730949), RR(1.4142135623730951))))

>>> from sage.all import *
>>> I = QQbar(I)
>>> sage_input(Integer(3)+Integer(4)*I, verify=True)
# Verified
QQbar(3 + 4*I)

>>> v = QQbar.zeta(Integer(3)) + QQbar.zeta(Integer(5))
>>> v - v == Integer(0)
True

>>> sage_input(vector(QQbar, (Integer(4)-Integer(3)*I, QQbar.
-> zeta(Integer(7)))), verify=True)
# Verified
R.<y> = QQ[]
vector(QQbar, [4 - 3*I, QQbar.polynomial_root(AA.common_polynomial(y^6 + y^5-
-> y^4 + y^3 + y^2 + y + 1), CIF(RIF(RR(0.62348980185873348), RR(0.
-> 62348980185873359)), RIF(RR(0.7818314824680298), RR(0.
-> 7813148246802991))))))

>>> sage_input(v, verify=True)
# Verified
R.<y> = QQ[]
v = QQbar.polynomial_root(AA.common_polynomial(y^8 - y^7 + y^5 - y^4 + y^3 -
-> y + 1), CIF(RIF(RR(0.91354545764260087), RR(0.91354545764260098)), RIF(RR(0.
-> 40673664307580015), RR(0.40673664307580021))))

invert(n)
Reciprocal of self.

EXAMPLES:

```
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: c = b.invert(None); c
# random (not uniquely represented)
-7/3*a^3 + 19/3*a^2 - 7*a - 9 where a^4 - 2*a^3 + a^2 + 6*a + 3 = 0
and a in 1.724744871391589? + 1.573132184970987?*I
sage: (c.generator() == b.generator() 
... and c.field_element_value() * b.field_element_value() == 1)
True
```

The parameter is ignored:

```
>>> from sage.all import *
>>> # needs sage.symbolic
>>> a = QQbar(sqrt(-Integer(2))) + QQbar(sqrt(-Integer(3)))
>>> a.exactify()
>>> b = a._descr
>>> type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
>>> c = b.invert(None); c
# random (not uniquely represented)
-7/3*a^3 + 19/3*a^2 - 7*a - 9 where a^4 - 2*a^3 + a^2 + 6*a + 3 = 0
and a in 1.724744871391589? + 1.573132184970987?*I
>>> (c.generator() == b.generator() 
... and c.field_element_value() * b.field_element_value() == Integer(1))
True
```

is_complex()

Return True if the number field that defines this element is not real.

This does not imply that the element itself is definitely non-real, as in the example below.

EXAMPLES:

```
sage: # needs sage.symbolic
sage: rt2 = QQbar(sqrt(2))
sage: rtm3 = QQbar(sqrt(-3))
sage: x = rtm3 + rt2 - rtm3
sage: x.exactify()
```
is_simple()  
Check whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

For ANExtensionElement elements, we check this by comparing the degree of the minimal polynomial to the degree of the field.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: rt2 = AA(sqrt(Integer(2)))
[...]
sage: rt2b._descr
a where a^2 - 2 = 0 and a in 1.414213562373095?

sage: rt2b._descr.is_simple()  # needs sage.symbolic
True

sage: rt2b._descr
a^3 - 3*a where a^4 - 4*a^2 + 1 = 0 and a in -0.5176380902050415?

sage: rt2b._descr.is_simple()  # needs sage.symbolic
False
```

(continues on next page)
minpoly()  
Compute the minimal polynomial of this algebraic number.

EXAMPLES:

```sage
sage: x = polygen(ZZ, 'x')
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr
˓→exactify()

sage: type(v)
<class sage.rings.qqbar.ANExtensionElement'>
sage: v.minpoly()
x^2 - x - 1
```

neg(n)  
Negation of self.

EXAMPLES:

```sage
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()

sage: b = a._descr

sage: type(b)
<class sage.rings.qqbar.ANExtensionElement'>
sage: c = b.neg(None); c  # random (not uniquely represented)
-1/3*a^3 + 1/3*a^2 - a - 1 where a^4 - 2*a^3 + a^2 + 6*a + 3 = 0
and a in 1.724744871391589? + 1.573132184970987?*I

sage: (c.generator() == b.generator() and c.field_element_value() + b.field_element_value() == 0)
True
```

from sage.all import *

# needs sage.symbolic

(continues on next page)
>>> a = QQbar(sqrt(-Integer(2))) + QQbar(sqrt(-Integer(3)))
>>> a.exactify()
>>> b = a._descr
>>> type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>

>>> c = b.neg(None); c
# random (not uniquely represented)
-1/3*a^3 + 1/3*a^2 - a - 1 where a^4 - 2*a^3 + a^2 + 6*a + 3 = 0
and a in 1.724744871391589? + 1.573132184970987?*I

>>> (c.generator() == b.generator() ... and
   c.field_element_value() + b.field_element_value() == Integer(0))
True

The parameter is ignored:

```
sage: (b.neg("random").generator() == c.generator())
# needs sage.symbolic
....: and b.neg("random").field_element_value() == c.field_element_value())
True
```

**norm**(n)

Norm of self (square of complex absolute value)

**EXAMPLES:**

```
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(-3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.norm(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

**rational_argument**(n)

If the argument of self is 2π times some rational number in [1/2, −1/2), return that rational; otherwise, return None.

**EXAMPLES:**

```
>>> from sage.all import *
>>> (b.neg("random").generator() == c.generator())
# needs sage.symbolic
....: and b.neg("random").field_element_value() == c.field_element_value())
True
```
sage: # needs sage.symbolic
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>

sage: b.rational_argument(a)  # is None
True

sage: x = polygen(QQ)
sage: a = (x**4 + 1).roots(QQbar, multiplicities=False)[0]
sage: a.exactify()
sage: b = a._descr

sage: b.rational_argument(a)
-3/8

simplify(n)

Compute an exact representation for this descriptor, in the smallest possible number field.

INPUT:

- n – The element of AA or QQbar corresponding to this descriptor.

EXAMPLES:
class sage.rings.qqbar.ANRational(x)

Bases: ANDescr

The subclass of ANDescr that represents an arbitrary rational. This class is private, and should not be used directly.

abs(n)

Absolute value of self.

EXAMPLES:

```python
sage: a = QQbar(3)
sage: b = a._descr
sage: b.abs(a)
3
```

angle()

Return a rational number $q \in (-1/2, 1/2]$ such that self is a rational multiple of $e^{2\pi i q}$. Always returns 0, since this element is rational.

EXAMPLES:

```python
sage: QQbar(3)._descr.angle()
0
sage: QQbar(-3)._descr.angle()
0
sage: QQbar(0)._descr.angle()
0
```

exactify()

Calculate self exactly. Since self is a rational number, return self.

EXAMPLES:
generator()

Return an AlgebraicGenerator object associated to this element. Returns the trivial generator, since self is rational.

EXAMPLES:

sage: QQbar(0)._descr.generator()
Trivial generator

handle_sage_input(sib, coerce, is_qqbar)

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always False, for rationals).

EXAMPLES:

sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: rat = ANRational(9/10)
sage: rat.handle_sage_input(sib, False, True)
(({call: {atomic:QQbar}({binop:/ {atomic:9} {atomic:10}})}, False)

>>> from sage.all import *
>>> sage_input(QQbar(Integer(22)/Integer(7)), verify=True)
# Verified
QQbar(22/7)
>>> sage_input(-AA(Integer(3))/Integer(5), verify=True)
# Verified
AA(-3/5)
>>> sage_input(vector(AA, (Integer(0), Integer(1)/Integer(2), Integer(1)/Integer(3))), verify=True)
# Verified
vector(AA, [0, 1/2, 1/3])

(continues on next page)
invert(n)

I/self.

EXAMPLES:

```sage
a = QQbar(3)
sage: b = a._descr
sage: b.invert(a)
1/3
```

is_complex()

Return False, since rational numbers are real

EXAMPLES:

```sage
QQbar(1/7)._descr.is_complex()
False
```

is_simple()

Checks whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

This is always true for rational numbers.

EXAMPLES:

```sage
AA(1/2)._descr.is_simple()
True
```

minpoly()

Return the min poly of self over \( \mathbb{Q} \).

EXAMPLES:

```sage
QQbar(7)._descr.minpoly()
x - 7
```
>>> from sage.all import *
>>> QQbar(Integer(7))._descr.minpoly()
x - 7

`neg(n)`

Negation of `self`.

**EXAMPLES:**

```python
sage: a = QQbar(3)
sage: b = a._descr
type(b)  # <class 'sage.rings.qqbar.ANRational'>
sage: b.neg(a)
-3
```

```python
>>> from sage.all import *
>>> a = QQbar(Integer(3))
>>> b = a._descr
type(b)  # <class 'sage.rings.qqbar.ANRational'>
>>> b.neg(a)
-3
```

`rational_argument(n)`

Return the argument of `self` divided by $2\pi$, or `None` if this element is 0.

**EXAMPLES:**

```python
sage: QQbar(3)._descr.rational_argument(None)
0
sage: QQbar(-3)._descr.rational_argument(None)
1/2
sage: QQbar(0)._descr.rational_argument(None) is None
True
```

```python
>>> from sage.all import *
>>> QQbar(Integer(3))._descr.rational_argument(None)
0
>>> QQbar(-Integer(3))._descr.rational_argument(None)
1/2
>>> QQbar(Integer(0))._descr.rational_argument(None) is None
True
```

`scale()`

Return a rational number $r$ such that `self` is equal to $re^{2\pi i q}$ for some $q \in (-1/2, 1/2]$. In other words, just return `self` as a rational number.

**EXAMPLES:**

```python
sage: QQbar(-3)._descr.scale()
-3
```

```python
>>> from sage.all import *
>>> QQbar(-Integer(3))._descr.scale()
-3
```
The subclass of \texttt{ANDescr} that represents a particular root of a polynomial with algebraic coefficients. This class is private, and should not be used directly.

**conjugate** \((n)\)

Complex conjugate of this \texttt{ANRoot} object.

**exactify** \((\text{)}\)

Return either an \texttt{ANRational} or an \texttt{ANExtensionElement} with the same value as this number.
handle_sage_input (sib, coerce, is_qqbar)

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always True for ANRoot).

EXAMPLES:

```python
sage: sage_input((AA(3)^(1/2))^(1/3), verify=True)
# Verified
R.<x> = AA[]
AA.polynomial_root(AA.common_polynomial(x^6 - AA.polynomial_root(AA.common_polynomial(x^2 - 3), RIF(RR(1.7320508075688772), RR(1.7320508075688774)))), RIF(RR(1.2009369551760025), RR(1.2009369551760027))]
```

These two examples are too big to verify quickly. (Verification would create a field of degree 28):

```python
sage: sage_input((sqrt(AA(3))^(5/7))^(9/4))
```

```python
sage: sage_input((sqrt(QQbar(-7))^(5/7))^(9/4))
```

```python
sage: sage_input((AA.polynomial_root(x^2-x-1, RIF(1, 2)), verify=True))
# Verified
R.<y> = QQ[]
AA.polynomial_root(AA.common_polynomial(y^2 - y - 1), RIF(RR(1.6051012265139511), RIF(RR(1.6051012265139511))))
```
5.1. Algebraic numbers

---

6180339887498947, RR(1.6180339887498949))

sage: sage_input(QQbar.polynomial_root(x^3-5, CIF(RIF(-3, 0), RIF(0, 3))),...
   verify=True)
# Verified
R.<y> = QQ[]

v1 = QQbar.polynomial_root(AA.common_polynomial(x^2 - 3), CIF(RIF(RR(1.7320508075688772), RR(1.7320508075688774))))
v2 = v1*v1
v3 = QQbar.polynomial_root(AA.common_polynomial(x^7 - v2*v2*v1), CIF(RIF(RR(0.8693488875796217), RR(0.86934888757962181)), RIF(RR(1.8052215661454434), RR(1.8052215661454436))))
v4 = v3*v3
v5 = v4*v4

---

from sage.all import *

>> sage_input((sqrt(AA(Integer(3)))**(Integer(5)/Integer(7)))**(Integer(9)/Integer(4)))
R.<x> = QQ[]
v1 = QQbar.polynomial_root(AA.common_polynomial(x^2 + 7), CIF(RIF(0),
   RIF(RR(2.6457513110645903), RR(2.6457513110645907))))
v2 = v1*v1
v3 = QQbar.polynomial_root(AA.common_polynomial(x^7 - v2*v2*v1), CIF(RIF(RR(0.8693488875796217), RR(0.86934888757962181)), RIF(RR(1.8052215661454434),
   RR(1.8052215661454436))))
v4 = v3*v3
v5 = v4*v4

---

>> x = polygen(QQ)
>> sage_input(AA.polynomial_root(x^2-x-Integer(1), RIF(Integer(1),...
   Integer(2))), verify=True)
# Verified
R.<y> = QQ[]

v1 = AA.polynomial_root(AA.common_polynomial(x^2 - y - 1), RIF(RR(1.6180339887498947), RR(1.6180339887498949)))

---

(continues on next page)
# Verified

R.<y> = QQ[]

QQbar.polynomial_root(AA.common_polynomial(y^3 - 5), CIF(RIF(-RR(0.85498797333834853), -RR(0.85498797333834842)), RIF(RR(1.4808826096823642), RR(1.4808826096823644))))

```python
from sage.rings.qqbar import *
from sage.misc.sage_input import SageInputBuilder

sib = SageInputBuilder()

rt = ANRoot(x**Integer(3) - Integer(2), RIF(Integer(0), Integer(4)))

rt.handle_sage_input(sib, False, True)
```

```python
({call: {getattr: {atomic:QQbar}.polynomial_root}({call: {getattr: {atomic:AA} → common_polynomial}({binop:- {binop:** {gen:y {constr_parent: {subscr: {atomic:QQ} {atomic:'y'}}} with gens: ('y',)} (atomic:3}) (atomic:2)}), →{call: {atomic:RIF}({call: {atomic:RR}({atomic:1.259921049894873})}, {call: {atomic:RR}({atomic:1.2599210498948732})})}), True})
```

**is_complex()**

Whether this is a root in $\overline{\mathbb{Q}}$ (rather than $\mathbb{A}$). Note that this may return True even if the root is actually real, as the second example shows; it does *not* trigger exact computation to see if the root is real.

**EXAMPLES:**

```python
sage: x = polygen(QQ)
sage: (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.is_complex() False
sage: (x^2 - x - 1).roots(ring=QQbar, multiplicities=False)[1]._descr.is_complex() True
```

**refine_interval** *(interval, prec)*

Takes an interval which is assumed to enclose exactly one root of the polynomial (or, with multiplicity=`k`, exactly one root of the $k-1$-st derivative); and a precision, in bits.

Tries to find a narrow interval enclosing the root using interval arithmetic of the given precision. (No particular number of resulting bits of precision is guaranteed.)

Uses a combination of Newton’s method (adapted for interval arithmetic) and bisection. The algorithm will converge very quickly if started with a sufficiently narrow interval.

**EXAMPLES:**

```python
>>> from sage.all import *
>>> x = polygen(QQ)
>>> (x**Integer(2) - x - Integer(1)).roots(ring=AA, ... multiplicities=False)[1]._descr.is_complex() False
>>> (x**Integer(2) - x - Integer(1)).roots(ring=QQbar, ... multiplicities=False)[1]._descr.is_complex() True
```
>>> from sage.all import *
>>> from sage.rings.qqbar import ANRoot
>>> x = polygen(AA)
>>> rt2 = ANRoot(x**Integer(2) - Integer(2), RIF(Integer(0), Integer(2)))
>>> rt2.refine_interval(RIF(Integer(0), Integer(2)), Integer(75))
1.4142135623730950488017?

class sage.rings.qqbar.ANUUnaryExpr(arg, op)

Bases: ANDescr

Initialize this ANUnaryExpr.

EXAMPLES:

sage: t = ~QQbar(sqrt(2)); type(t._descr)  # indirect doctest
\[needs sage.symbolic\]
<class 'sage.rings.qqbar.ANUUnaryExpr'>

>>> from sage.all import *
>>> t = ~QQbar(sqrt(Integer(2))); type(t._descr)  # indirect doctest
\[needs sage.symbolic\]
<class 'sage.rings.qqbar.ANUUnaryExpr'>

exactify()

Trigger exact computation of self.

EXAMPLES:

sage: v = (-QQbar(sqrt(AA(2)))._descr  # needs sage.symbolic
\[class 'sage.rings.qqbar.ANUUnaryExpr'\]
\[v.exactify()  # needs sage.symbolic\]

-s-a where a^2 - 2 = 0 and a in 1.414213562373095?

handle_sage_input(sib, coerce, is_qqbar)

Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always True for ANUnaryExpr).

EXAMPLES:

sage: sage_input(-sqrt(AA(2)), verify=True)
# Verified
R.<x> = AA[]
-AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949),
(continues on next page)
\[
\begin{align*}
\text{sage: } & \text{sage_input}(-\sqrt{\text{AA}(2)}, \ \text{verify=True}) \\
& \# \text{Verified} \\
R.<x> &= \text{AA[]} \\
& \text{-\text{AA}.polynomial_root(\text{AA}.common_polynomial(x^2 - 2), \ \text{RIF}(\text{RR}(1.4142135623730949), \ \rightarrow \text{RR}(1.4142135623730951)))} \\
\text{sage: } & \text{sage_input}(-\sqrt{\text{QQbar}(-3)}.\text{conjugate()}, \ \text{verify=True}) \\
& \# \text{Verified} \\
R.<x> &= \text{QQbar[]} \\
\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(x^2 + 3), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ \text{RIF}(\text{RR}(1.7320508075688772), \ \rightarrow \text{RR}(1.7320508075688774)))), \ \text{conjugate}() \\
\text{sage: } & \text{sage_input}(\text{QQbar.zeta}(3).\text{real}()), \ \text{verify=True}) \\
& \# \text{Verified} \\
R.<y> &= \text{QQ[]} \\
\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(y^2 + y + 1), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ \rightarrow \text{RR}(0.49999999999999994)), \ \text{RIF}(\text{RR}(0.8660254037844386), \ ightarrow \text{RR}(0.86602540378443871))) \text{.real}() \\
\text{sage: } & \text{sage_input}(\text{QQbar.zeta}(3).\text{imag}()), \ \text{verify=True}) \\
& \# \text{Verified} \\
R.<x> &= \text{QQ[]} \\
\text{abs}(\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(x^2 + 3), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ \rightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)\)).\text{norm}() \\
\text{sage: } & \text{sage_input}(\text{QQbar}(\text{QQbar}.zeta(3).\text{real}()), \ \text{verify=True}) \\
& \# \text{Verified} \\
R.<y> &= \text{QQ[]} \\
\text{QQbar}(\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(y^2 + y + 1), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ ightarrow \text{RR}(0.49999999999999994)), \ \text{RIF}(\text{RR}(0.8660254037844386), \ ightarrow \text{RR}(0.86602540378443871))) \text{.real}() \\
\text{from sage.rings.qqbar import *} \\
\text{from sage.misc.sage_input import SageInputBuilder} \\
\text{sage: } & \text{sage_input}(\text{sqrt}(\text{QQbar}(-3))), \ \text{verify=True}) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{sqrt}(\text{QQbar}(-3)))), \ \text{verify=True}) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{QQbar}(\text{QQbar}.zeta(3).\text{real}())), \ \text{verify=True}) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{QQbar}(\text{QQbar}.zeta(3).\text{imag}())), \ \text{verify=True}) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(x^2 + 3), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ \rightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ ightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))\)).\text{norm}() \\
\text{sage: } & \text{sage_input}(\text{QQbar}(\text{QQbar}.zeta(3).\text{real}()))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{QQbar}(\text{QQbar}.zeta(3).\text{imag}()))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{sqrt}(\text{QQbar}(-3)))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(x^2 + 3), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ \rightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ ightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))\)).\text{norm}() \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}(\text{QQbar}.zeta(3).\text{real}())) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}(\text{QQbar}.zeta(3).\text{imag}()))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{sqrt}(\text{QQbar}(-3)))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(x^2 + 3), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ \rightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ ightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))\)).\text{norm}() \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}(\text{QQbar}.zeta(3).\text{real}()))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}(\text{QQbar}.zeta(3).\text{imag}()))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{sqrt}(\text{QQbar}(-3)))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}.polynomial_root(\text{AA}.common_polynomial(x^2 + 3), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ \rightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))), \ \text{CIF}(\text{RIF}(\text{RR}(0)), \ ightarrow \text{RR}(1.7320508075688772), \ ightarrow \text{RR}(1.7320508075688774)))\)).\text{norm}() \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}(\text{QQbar}.zeta(3).\text{real}()))) \\
& \# \text{Verified} \\
\text{sage: } & \text{sage_input}(\text{abs}(\text{QQbar}(\text{QQbar}.zeta(3).\text{imag}()))) \\
& \# \text{Verified} \\
\end{align*}
\]
```python
>>> from sage.all import *

>>> sage_input(-sqrt(AA(Integer(2))), verify=True)
# Verified
R.<x> = AA[]
-~AA.polynomial_root(AA.common_polynomial(x**2 - 2), RIF(RR(1.4142135623730949),
     → RR(1.4142135623730951)))

>>> sage_input(~sqrt(AA(Integer(2))), verify=True)
# Verified
R.<x> = AA[]
~AA.polynomial_root(AA.common_polynomial(x**2 - 2), RIF(RR(1.4142135623730949),
     → RR(1.4142135623730951)))

>>> sage_input(sqrt(QQbar(-Integer(3))).conjugate(), verify=True)
# Verified
R.<x> = QQbar[]
QQbar.polynomial_root(AA.common_polynomial(x**2 + 3), CIF(RIF(RR(0)), RIF(RR(0.
     →7320508075688772), RR(1.7320508075688774))).conjugate()

>>> sage_input(QQbar.zeta(Integer(3)).real(), verify=True)
# Verified
R.<y> = QQ[]
QQbar.polynomial_root(AA.common_polynomial(y**2 + y + 1), CIF(RIF(-RR(0.
     →50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386),␣
     →RR(0.86602540378443871))).real()

>>> sage_input(QQbar.zeta(Integer(3)).imag(), verify=True)
# Verified
R.<y> = QQ[]
QQbar.polynomial_root(AA.common_polynomial(y**2 + y + 1), CIF(RIF(-RR(0.
     →50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386),␣
     →RR(0.86602540378443871))).imag()

>>> sage_input(abs(sqrt(QQbar(-Integer(3)))), verify=True)
# Verified
R.<x> = QQbar[]
abs(QQbar.polynomial_root(AA.common_polynomial(x**2 + 3), CIF(RIF(RR(0)),␣
     →RIF(RR(1.7320508075688772), RR(1.7320508075688774))))

>>> sage_input(sqrt(QQbar(-Integer(3))).norm(), verify=True)
# Verified
R.<x> = QQbar[]
QQbar.polynomial_root(AA.common_polynomial(x**2 + 3), CIF(RIF(RR(0)),␣
     →RIF(RR(1.7320508075688772), RR(1.7320508075688774))).norm()

>>> sage_input(QQbar(QQbar.zeta(Integer(3)).real()), verify=True)
# Verified
R.<y> = QQ[]
QQbar(QQbar.polynomial_root(AA.common_polynomial(y**2 + y + 1), CIF(RIF(-RR(0.
     →50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386),␣
     →RR(0.86602540378443871))).real())

>>> from sage.rings.qqbar import *
>>> from sage.misc.sage_input import SageInputBuilder

>>> sib = SageInputBuilder()
>>> unexp = ANUnaryExpr(sqrt(AA(Integer(2))), '~')
>>> unexp.handle_sage_input(sib, False, False)
{(unop:~ {call: {getattr: {atomic:AA}.polynomial_root}({call: {getattr:
     →{atomic:AA}.common_polynomial}({binop:- {binop:** {gen:x {constr_parent:␣
     →{subscr: {atomic:AA}{{atomic:'x'}}} with gens: ('x',)} {atomic:2}}
     →{atomic:2)}}), {call: {atomic:RIF}({call: {atomic:RR}{{atomic:1.
     →4142135623730949}}}, {call: {atomic:RR}{{atomic:1.4142135623730951}}}))}}}},
```

(continues on next page)
is_complex()

Return whether or not this element is complex. Note that this is a data type check, and triggers no computations – if it returns False, the element might still be real, it just doesn’t know it yet.

EXAMPLES:

```python
sage: # needs sage.symbolic
sage: t = AA(sqrt(2))
traceback:
  File "...", line 1, in <module>
    t = AA(sqrt(2))  # needs sage.symbolic
  File "<ipython-input-1-6d7800e77e76>", line 1, in <module>
NameError: name 'AA' is not defined

sage: s = (-t)._descr
sage: s.is_complex()
False
sage: QQbar(-sqrt(2))._descr.is_complex()
True
```

class sage.rings.qqbar.AlgebraicField

Bases: Singleton, AlgebraicField_common, AlgebraicField

The field of all algebraic complex numbers.

algebraic_closure()

Return the algebraic closure of this field.

As this field is already algebraically closed, just returns self.

EXAMPLES:

```python
sage: QQbar.algebraic_closure()
Algebraic Field
```

```python
>>> from sage.all import *
```

```python
>>> QQbar.algebraic_closure()
Algebraic Field
```
**completion** (*p*, *prec*, *extras=*{})

Return the completion of *self* at the place *p*.

Only implemented for *p* = ∞ at present.

**INPUT:**

- *p* – either a prime (not implemented at present) or Infinity
- *prec* – precision of approximate field to return
- *extras* – (optional) a dict of extra keyword arguments for the RealField constructor

**EXAMPLES:**

```python
sage: QQbar.completion(infinity, 500)
Complex Field with 500 bits of precision
sage: QQbar.completion(infinity, prec=53, extras=('type':'RDF'))
Complex Double Field
sage: QQbar.completion(infinity, 53) is CC
True
sage: QQbar.completion(3, 20)
Traceback (most recent call last):
... 
NotImplementedError
```

```python
>>> from sage.all import *
>>> QQbar.completion(infinity, Integer(500))
Complex Field with 500 bits of precision
>>> QQbar.completion(infinity, prec=Integer(53), extras=('type':'RDF'))
Complex Double Field
>>> QQbar.completion(infinity, Integer(53)) is CC
True
>>> QQbar.completion(Integer(3), Integer(20))
Traceback (most recent call last):
... 
NotImplementedError
```

**construction()**

Return a functor that constructs *self* (used by the coercion machinery).

**EXAMPLES:**

```python
sage: QQbar.construction()
(AlgebraicClosureFunctor, Rational Field)
```

```python
>>> from sage.all import *
>>> QQbar.construction()
(AlgebraicClosureFunctor, Rational Field)
```

**gen** (*n*=0)

Return the *n*-th element of the tuple returned by *gens()*.

**EXAMPLES:**

```python
sage: QQbar.gen(0)
I
sage: QQbar.gen(1)
Traceback (most recent call last):
```

(continues on next page)
>>> from sage.all import *
>>> QQbar.gen(Integer(0))
I
>>> QQbar.gen(Integer(1))
Traceback (most recent call last):
... IndexError: n must be 0

\textbf{gens}()

Return a set of generators for this field.

As this field is not finitely generated over its prime field, we opt for just returning I.

\textbf{EXAMPLES}:

\begin{verbatim}
sage: QQbar.gens()
(I,)
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> QQbar.gens()
(I,)
\end{verbatim}

\textbf{ngens}()

Return the size of the tuple returned by \textit{gens}().

\textbf{EXAMPLES}:

\begin{verbatim}
sage: QQbar.ngens()
1
\end{verbatim}

\begin{verbatim}
>>> from sage.all import *
>>> QQbar.ngens()
1
\end{verbatim}

\textbf{polynomial\_root} (\textit{poly}, \textit{interval}, \textit{multiplicity}=1)

Given a polynomial with algebraic coefficients and an interval enclosing exactly one root of the polynomial, constructs an algebraic real representation of that root.

The polynomial need not be irreducible, or even squarefree; but if the given root is a multiple root, its multiplicity must be specified. (IMPORTANT NOTE: Currently, multiplicity-\textit{k} roots are handled by taking the \textit{(k − 1)}-st derivative of the polynomial. This means that the interval must enclose exactly one root of this derivative.)

The conditions on the arguments (that the interval encloses exactly one root, and that multiple roots match the given multiplicity) are not checked; if they are not satisfied, an error may be thrown (possibly later, when the algebraic number is used), or wrong answers may result.

Note that if you are constructing multiple roots of a single polynomial, it is better to use \texttt{QQbar.common\_polynomial} to get a shared polynomial.

\textbf{EXAMPLES}:
Algebraic Numbers and Number Fields, Release 10.4

sage: x = polygen(QQbar)
sage: phi = QQbar.polynomial_root(x^2 - x - 1, RIF(0, 2)); phi
1.618033988749895?
sage: p = (x-1)^7 * (x-2)
sage: r = QQbar.polynomial_root(p, RIF(9/10, 11/10), multiplicity=7)
sage: r; r == 1
1
True
sage: p = (x-phi)*(x-sqrt(QQbar(2)))
sage: r = QQbar.polynomial_root(p, RIF(1, 3/2))
sage: r; r == sqrt(QQbar(2))
1.414213562373095?
True

>>> from sage.all import *
>>> x = polygen(QQbar)
>>> phi = QQbar.polynomial_root(x**Integer(2) - x - Integer(1),
˓→RIF(Integer(0), Integer(2))); phi
1.618033988749895?
>>> p = (x-Integer(1))**Integer(7) * (x-Integer(2))
>>> r = QQbar.polynomial_root(p, RIF(Integer(9)/Integer(10), Integer(11)/
˓→Integer(10)), multiplicity=Integer(7))
>>> r; r == Integer(1)
1
True
>>> p = (x-phi)*(x-sqrt(QQbar(Integer(2))))
>>> r = QQbar.polynomial_root(p, RIF(Integer(1), Integer(3)/Integer(2)))
>>> r; r == sqrt(QQbar(Integer(2)))
1.414213562373095?
True

random_element (poly_degree=2, *args, **kwds)

Returns a random algebraic number.

INPUT:

* poly_degree – default: 2; degree of the random polynomial over the integers of which the returned algebraic number is a root. This is not necessarily the degree of the minimal polynomial of the number. Increase this parameter to achieve a greater diversity of algebraic numbers, at a cost of greater computation time. You can also vary the distribution of the coefficients but that will not vary the degree of the extension containing the element.

* args, kwds – arguments and keywords passed to the random number generator for elements of ZZ, the integers. See random_element() for details, or see example below.

OUTPUT:

An element of QQbar, the field of algebraic numbers (see sage.rings.qqbar).

ALGORITHM:

A polynomial with degree between 1 and poly_degree, with random integer coefficients is created. A root of this polynomial is chosen at random. The default degree is 2 and the integer coefficients come from a distribution heavily weighted towards 0, ±1, ±2.

EXAMPLES:

sage: a = QQbar.random_element()
sage: a  # random

(continues on next page)
Parameters for the distribution of the integer coefficients of the polynomials can be passed on to the random element method for integers. For example, current default behavior of this method returns zero about 15% of the time; if we do not include zero as a possible coefficient, there will never be a zero constant term, and thus never a zero root.

```sage```
```
z = [QQbar.random_element(x=1, y=10) for _ in range(20)]
```
```
sage: QQbar(0) in z
False
```
```sage```
```
>>> from sage.all import *
>>> z = [QQbar.random_element(x=Integer(1), y=Integer(10)) for _ in range(Integer(20))]
>>> QQbar(Integer(0)) in z
False
```
```
If you just want real algebraic numbers you can filter them out. Using an odd degree for the polynomials will ensure some degree of success.

```sage```
```
r = []
sage: while len(r) < 3:
    x = QQbar.random_element(poly_degree=3)
    if x in AA:
        r.append(x)
sage: (len(r) == 3) and all(z in AA for z in r)
True
```
```sage```
```
>>> from sage.all import *
>>> r = []
>>> while len(r) < Integer(3):
...     x = QQbar.random_element(poly_degree=Integer(3))
...     if x in AA:
...         r.append(x)
```
zeta \( (n=4) \)

Return a primitive \( n \)’th root of unity, specifically \( \exp(2 \pi i/n) \).

**INPUT:**

- \( n \) (integer) – default 4

**EXAMPLES:**

```python
sage: QQbar.zeta(1)
1
sage: QQbar.zeta(2)
-1
sage: QQbar.zeta(3)
-0.500000000000000? + 0.866025403784439?*I
sage: QQbar.zeta(4)
I
sage: QQbar.zeta()
I
sage: QQbar.zeta(5)
0.309016994374947? + 0.9510565162951536?*I
sage: QQbar.zeta(3000)
0.999997806755380? + 0.002094393571219374?*I
```

```python
>>> from sage.all import *

>>> QQbar.zeta(Integer(1))
1
>>> QQbar.zeta(Integer(2))
-1
>>> QQbar.zeta(Integer(3))
-0.500000000000000? + 0.866025403784439?*I
>>> QQbar.zeta(Integer(4))
I
>>> QQbar.zeta()
I
>>> QQbar.zeta(Integer(5))
0.309016994374947? + 0.9510565162951536?*I
>>> QQbar.zeta(Integer(3000))
0.999997806755380? + 0.002094393571219374?*I
```

class sage.rings.qqbar.AlgebraicField_common

**Bases:** `AlgebraicField_common`

Common base class for the classes `AlgebraicRealField` and `AlgebraicField`.

**characteristic()**

Return the characteristic of this field.

Since this class is only used for fields of characteristic 0, this always returns 0.

**EXAMPLES:**

```python
sage: AA.characteristic()
0
```
common_polynomial (poly)

Given a polynomial with algebraic coefficients, return a wrapper that caches high-precision calculations and factorizations. This wrapper can be passed to polynomial_root() in place of the polynomial.

Using common_polynomial() makes no semantic difference, but will improve efficiency if you are dealing with multiple roots of a single polynomial.

EXAMPLES:

```python
sage: x = polygen(ZZ)
sage: p = AA.common_polynomial(x^2 - x - 1)
sage: phi = AA.polynomial_root(p, RIF(1, 2))
sage: tau = AA.polynomial_root(p, RIF(-1, 0))
sage: phi + tau == 1
True
sage: phi * tau == -1
True
```

```python
sage: # needs sage.symbolic
sage: x = polygen(SR)
sage: p = (x - sqrt(-5)) * (x - sqrt(3)); p
x^2 + (-sqrt(3) - sqrt(-5))*x + sqrt(3)*sqrt(-5)
sage: p = QQbar.common_polynomial(p)
sage: a = QQbar.polynomial_root(p, CIF(RIF(-0.1, 0.1), RIF(2, 3))); a
0.?e-18 + 2.236067977499790?*I
sage: b = QQbar.polynomial_root(p, RIF(1, 2)); b
1.732050807568878?
```

These “common polynomials” can be shared between real and complex roots:

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> p = AA.common_polynomial(x**Integer(2) - x - Integer(1))
>>> phi = AA.polynomial_root(p, RIF(Integer(1), Integer(2)))
>>> tau = AA.polynomial_root(p, RIF(-Integer(1), Integer(0)))
>>> phi + tau == Integer(1)
True
>>> phi * tau == -Integer(1)
True
```

```python
>>> # needs sage.symbolic
>>> x = polygen(SR)
>>> p = (x - sqrt(-Integer(5))) * (x - sqrt(Integer(3))); p
x^2 + (-sqrt(3) - sqrt(-5))*x + sqrt(3)*sqrt(-5)
>>> p = QQbar.common_polynomial(p)
>>> a = QQbar.polynomial_root(p, CIF(RIF(-RealNumber(0.1), RealNumber(0.1 ˓→)), RIF(Integer(2), Integer(3)))); a
0.7e-18 + 2.236067977499790?*I
>>> b = QQbar.polynomial_root(p, RIF(Integer(1), Integer(2))); b
1.732050807568878?
```
1.324717957244746?
sage: r2 = QQbar.polynomial_root(p, CIF(RIF(-0.7, -0.6), RIF(0.5, 0.6))); r2
-0.6623589786223730 + 0.5622795120623013*I

>>> from sage.all import *
>>> p = AA.common_polynomial(x**Integer(3) - x - Integer(1))
>>> r1 = AA.polynomial_root(p, RIF(RealNumber(1.3), RealNumber(1.4))); r1
1.324717957244746?
>>> r2 = QQbar.polynomial_root(p, CIF(RIF(-RealNumber(0.7), -RealNumber(0.6)), RIF(RealNumber(0.5), RealNumber(0.6)))); r2
-0.6623589786223730 + 0.5622795120623013*I

default_interval_prec()
Return the default interval precision used for root isolation.

EXAMPLES:
sage: AA.default_interval_prec()
64

>>> from sage.all import *
>>> AA.default_interval_prec()
64

options = Current options for AlgebraicField – display_format: decimal

order()
Return the cardinality of self.

Since this class is only used for fields of characteristic 0, always returns Infinity.

EXAMPLES:
sage: QQbar.order()
+Infinity

>>> from sage.all import *
>>> QQbar.order()
+Infinity

class sage.rings.qqbar.AlgebraicGenerator (field, root)

Bases: SageObject

An AlgebraicGenerator represents both an algebraic number \( \alpha \) and the number field \( \mathbb{Q}[\alpha] \). There is a single AlgebraicGenerator representing \( \mathbb{Q} \) (with \( \alpha = 0 \)).

The AlgebraicGenerator class is private, and should not be used directly.

conjugate()
If this generator is for the algebraic number \( \alpha \), return a generator for the complex conjugate of \( \alpha \).

EXAMPLES:
sage: from sage.rings.qqbar import AlgebraicGenerator
sage: x = polygen(QQ); f = x^4 + x + 17
sage: nf = NumberField(f, name='a')
sage: b = f.roots(QQbar)[0][0]
sage: root = b._descr
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.conjugate()
Number Field in a with defining polynomial \(x^4 + x + 17\) with a in \(-1\).
\(-43644997483091? + 1.374535713065812?\*I

>>> from sage.all import *
>>> from sage.rings.qqbar import AlgebraicGenerator
>>> x = polygen(QQ); f = x**Integer(4) + x + Integer(17)
>>> nf = NumberField(f, name='a')
>>> b = f.roots(QQbar)[Integer(0)][Integer(0)]
>>> root = b._descr
>>> gen = AlgebraicGenerator(nf, root)
>>> gen.conjugate()
Number Field in a with defining polynomial \(x^4 + x + 17\) with a in \(-1\).
\(-43644997483091? + 1.374535713065812?\*I

field()
Return the number field attached to self.

EXAMPLES:

sage: from sage.rings.qqbar import qq_generator
sage: qq_generator.field()
Rational Field

>>> from sage.all import *
>>> from sage.rings.qqbar import qq_generator
>>> qq_generator.field()
Rational Field

is_complex()
Return True if this is a generator for a non-real number field.

EXAMPLES:

sage: z7 = QQbar.zeta(Integer(7))
sage: g = z7._descr._generator
sage: g.is_complex()
True

sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator
sage: y = polygen(QQ, 'y')
sage: x = polygen(QQbar)
sage: nf = NumberField(y^2 - y - 1, name='a', check=False)
sage: root = ANRoot(x^2 - x - 1, RIF(1, 2))
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.is_complex()
False

>>> from sage.all import *
>>> z7 = QQbar.zeta(Integer(7))
>>> g = z7._descr._generator
>>> g.is_complex()
True

```python
>>> from sage.rings.qqbar import ANRoot, AlgebraicGenerator
>>> y = polygen(QQ, 'y')
>>> x = polygen(QQbar)
>>> nf = NumberField(y**Integer(2) - y - Integer(1), name='a', check=False)
>>> root = ANRoot(x**Integer(2) - x - Integer(1), RIF(Integer(1), Integer(2)))
>>> gen = AlgebraicGenerator(nf, root)
>>> gen.is_complex()
False
```

### is_trivial()

Return True iff this is the trivial generator (alpha == 1), which does not actually extend the rationals.

**EXAMPLES:**

```python
sage: from sage.rings.qqbar import qq_generator
sage: qq_generator.is_trivial()
True
```

```python
>>> from sage.all import *
>>> from sage.rings.qqbar import qq_generator

>>> qq_generator.is_trivial()
True
```

### pari_field()

Return the PARI field attached to this generator.

**EXAMPLES:**

```python
sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator, qq_generator
sage: y = polygen(QQ)
sage: x = polygen(QQbar)
sage: nf = NumberField(y**Integer(2) - y - Integer(1), name='a', check=False)
sage: root = ANRoot(x**Integer(2) - x - Integer(1), RIF(Integer(1), Integer(2)))
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.pari_field()  # doctest: +NORMALIZE_WHITESPACE
[[y^2 - y - 1, [2, 0], ...

```
root = ANRoot(x**Integer(2) - x - Integer(1), RIF(Integer(1), Integer(2)))
gen = AlgebraicGenerator(nf, root)
gen.pari_field()

[[(y**2 - y - 1, [2, 0], ...]]

root_as_algebraic()

Return the root attached to self as an algebraic number.

EXAMPLES:
sage: t = sage.rings.qqbar.qq_generator.root_as_algebraic(); t
1
sage: t.parent()
Algebraic Real Field

super_poly(super, checked=None)

Given a generator gen and another generator super, where super is the result of a tree of union() operations where one of the leaves is gen, gen.super_poly(super) returns a polynomial expressing the value of gen in terms of the value of super (except that if gen is qq_generator, super_poly() always returns None.)

EXAMPLES:
sage: from sage.rings.qqbar import AlgebraicGenerator, ANRoot, qq_generator
sage: _.<y> = QQ['y']
sage: x = polygen(QQbar)
sage: nf2 = NumberField(y**2 - 2, name='a', check=False)
sage: root2 = ANRoot(x**2 - 2, RIF(1, 2))
sage: gen2 = AlgebraicGenerator(nf2, root2)
sage: gen2
Number Field in a with defining polynomial y^2 - 2 with a in 1.˓→414213562373095?
sage: nf3 = NumberField(y**2 - 3, name='a', check=False)
sage: root3 = ANRoot(x**2 - 3, RIF(1, 2))
sage: gen3 = AlgebraicGenerator(nf3, root3)
sage: gen3
Number Field in a with defining polynomial y^2 - 3 with a in 1.˓→732050807568878?
sage: gen2_3 = gen2.union(gen3)
sage: gen2_3
Number Field in a with defining polynomial y^4 - 4*y^2 + 1 with a in -1.˓→931851652578137?
sage: qq_generator.super_poly(gen2) is None
True
sage: gen2.super_poly(gen2_3)
-a^3 + 3*a
sage: gen3.super_poly(gen2_3)
a^2 - 2
>>> from sage.all import *
>>> from sage.rings.qqbar import AlgebraicGenerator, ANRoot, qq_generator
>>> _ = QQ['y']; (y,) = _.first_ngens(1)
>>> x = polygen(QQbar)
>>> nf2 = NumberField(y**Integer(2) - Integer(2), name='a', check=False)
>>> root2 = ANRoot(x**Integer(2) - Integer(2), RIF(Integer(1), Integer(2)))
>>> gen2 = AlgebraicGenerator(nf2, root2)
>>> gen2
Number Field in a with defining polynomial y^2 - 2 with a in 1.
˓→414213562373095?

>>> nf3 = NumberField(y**Integer(2) - Integer(3), name='a', check=False)
>>> root3 = ANRoot(x**Integer(2) - Integer(3), RIF(Integer(1), Integer(2)))
>>> gen3 = AlgebraicGenerator(nf3, root3)
>>> gen3
Number Field in a with defining polynomial y^2 - 3 with a in 1.
˓→732050807568878?

>>> gen2_3 = gen2.union(gen3)
>>> gen2_3
Number Field in a with defining polynomial y^4 - 4*y^2 + 1 with a in -1.
˓→931851652578137?

>>> qq_generator.super_poly(gen2) is None
True

>>> gen2.super_poly(gen2_3)
-a^3 + 3*a

union (other, name='a')

Given generators self, α, and other, β, self.union(other) gives a generator for the number field Q[α][β].

INPUT:

• other – an algebraic number

• name – string (default: 'a')

a name for the primitive element

EXAMPLES:

sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator, qq_generator
sage: _.<y> = QQ['y']

sage: x = polygen(QQbar)

sage: nf2 = NumberField(y**Integer(2) - 2, name='a', check=False)

sage: root2 = ANRoot(x**Integer(2) - 2, RIF(1, 2))

sage: gen2 = AlgebraicGenerator(nf2, root2)

sage: gen2
Number Field in a with defining polynomial y^2 - 2 with a in 1.
˓→414213562373095?

sage: nf3 = NumberField(y**Integer(2) - 3, name='a', check=False)

sage: root3 = ANRoot(x**Integer(2) - 3, RIF(1, 2))

sage: gen3 = AlgebraicGenerator(nf3, root3)

sage: gen3
Number Field in a with defining polynomial y^2 - 3 with a in 1.
˓→732050807568878?

sage: gen2.union(qq_generator) is gen2
True

sage: gen2.union(qq_generator) is gen2
True

(continues on next page)
sage: gen2.union(gen3, name='b')
Number Field in b with defining polynomial y^4 - 4*y^2 + 1 with b in -1.
˓→931851652578137?

>>> from sage.all import *
>>> from sage.rings.qqbar import ANRoot, AlgebraicGenerator, qq_generator
>>> _ = QQ['y']; (y,) = _.first_ngens(1)
>>> x = polygen(QQbar)
>>> nf2 = NumberField(y**Integer(2) - Integer(2), name='a', check=False)
>>> root2 = ANRoot(x**Integer(2) - Integer(2), RIF(Integer(1), Integer(2)))
>>> gen2 = AlgebraicGenerator(nf2, root2)
>>> gen2
Number Field in a with defining polynomial y^2 - 2 with a in 1.
˓→414213562373095?
>>> nf3 = NumberField(y**Integer(2) - Integer(3), name='a', check=False)
>>> root3 = ANRoot(x**Integer(2) - Integer(3), RIF(Integer(1), Integer(2)))
>>> gen3 = AlgebraicGenerator(nf3, root3)
>>> gen3
Number Field in a with defining polynomial y^2 - 3 with a in 1.
˓→732050807568878?
>>> gen2.union(qq_generator) is gen2
True
>>> qq_generator.union(gen3) is gen3
True
>>> gen2.union(gen3, name='b')
Number Field in b with defining polynomial y^4 - 4*y^2 + 1 with b in -1.
˓→931851652578137?

class sage.rings.qqbar.AlgebraicGeneratorRelation(child1, child1_poly, child2, child2_poly, parent)

Bases: SageObject

A simple class for maintaining relations in the lattice of algebraic extensions.

class sage.rings.qqbar.AlgebraicNumber(x)

Bases: AlgebraicNumber_base

The class for algebraic numbers (complex numbers which are the roots of a polynomial with integer coefficients). Much of its functionality is inherited from AlgebraicNumber_base.

_\text{\_\texttt{richcmp}}\_ (other, op)

Compare two algebraic numbers, lexicographically. (That is, first compare the real components; if the real components are equal, compare the imaginary components.)

EXAMPLES:

sage: x = QQbar.zeta(3); x
-0.500000000000000? + 0.866025403784439?*I
sage: QQbar(-1) < x
True
sage: QQbar(-1/2) < x
True
sage: QQbar(0) > x
True
One problem with this lexicographic ordering is the fact that if two algebraic numbers have the same real component, that real component has to be compared for exact equality, which can be a costly operation. For the special case where both numbers have the same minimal polynomial, that cost can be avoided, though (see Issue #16964):

```python
sage: x = polygen(ZZ)
sage: p = 69721504*x^8 + 251777664*x^6 + 329532012*x^4 + 184429548*x^2 + Integer(37344321)
sage: sorted(p.roots(QQbar, False))
[-0.0221204634374361? - 1.090991904211621?*I,
 -0.0221204634374361? + 1.090991904211621?*I,
 -0.8088604911480535?*I,
 0.2e-182 - 0.7598602580415435?*I,
 0.2e-249 + 0.7598602580415435?*I,
 0.8088604911480535?*I,
 0.0221204634374361? - 1.090991904211621?*I,
 0.0221204634374361? + 1.090991904211621?*I]```

It also works for comparison of conjugate roots even in a degenerate situation where many roots have the same real part. In the following example, the polynomial \( p_2 \) is irreducible and all its roots have real part equal to 1:

```python
sage: p1 = x^8 + 74*x^7 + 2300*x^6 + 38928*x^5 + 388193*x^4 + 2295312*x^3 + 7613898*x^2 + 12066806*x + 5477001
sage: p2 = p1((x-1)^2)
sage: sum(1 for r in p2.roots(CC, False) if abs(r.real() - 1) < 0.0001)
16
sage: r1 = QQbar.polynomial_root(p2, CIF(1, (-4.1,-4.0)))
sage: r2 = QQbar.polynomial_root(p2, CIF(1, (4.0, 4.1)))
sage: all([r1<r2, r1==r1, r2==r2, r2>r1])
True
```
>>> from sage.all import *

>>> p1 = x**Integer(8) + Integer(74)*x**Integer(7) +
    Integer(2300)*x**Integer(6) + Integer(38928)*x**Integer(5) +
    Integer(388193)*x**Integer(4) + Integer(2295312)*x**Integer(3) +
    Integer(12066806)*x + Integer(5477001)

>>> p2 = p1((x-Integer(1))**Integer(2))

>>> sum(Integer(1) for r in p2.roots(CC,False) if abs(r.real() - Integer(1)) < RealNumber(0.0001))
16

>>> r1 = QQbar.polynomial_root(p2, CIF(Integer(1), (-RealNumber(4.1),-
    RealNumber(4.0))))

>>> r2 = QQbar.polynomial_root(p2, CIF(Integer(1), (RealNumber(4.0),
    RealNumber(4.1))))

>>> all([r1<r2, r1==r1, r2==r2, r2>r1])
True

Though, comparing roots which are not equal or conjugate is much slower because the algorithm needs to
check the equality of the real parts:

sage: sorted(p2.roots(QQbar,False))   # long time - 3 secs
[1.000000000000000? - 4.016778562562223?*I,
 1.000000000000000? - 3.850538755978243?*I,
 1.000000000000000? - 3.390564396412898?*I,
...]

complex_exact (field)

Given a ComplexField, return the best possible approximation of this number in that field. Note that if
either component is sufficiently close to the halfway point between two floating-point numbers in the corre-
sponding RealField, then this will trigger exact computation, which may be very slow.

EXAMPLES:

sage: a = QQbar.zeta(9) + QQbar(I) + QQbar.zeta(9).conjugate(); a
1.53208886237957? + 1.000000000000000?*I

sage: a.complex_exact(CIF)
1.53208886237957? + 1*I

>>> from sage.all import *

>>> a = QQbar.zeta(Integer(9)) + QQbar(I) + QQbar.zeta(Integer(9)).
    conjugate(); a
1.53208886237957? + 1.000000000000000?*I

>>> a.complex_exact(CIF)
1.53208886237957? + 1*I
complex_number(field)

Given the complex field field, compute an accurate approximation of this element in that field.

The approximation will be off by at most two ulp's in each component, except for components which are very close to zero, which will have an absolute error at most $2^{-\text{prec}+1}$ where prec is the precision of the field.

EXAMPLES:

```sage
sage: a = QQbar.zeta(5)
sage: a.complex_number(CC)
0.309016994374947 + 0.951056516295154*I

sage: b = QQbar(2).sqrt() + QQbar(3).sqrt() * QQbar.gen()
sage: b.complex_number(ComplexField(128))
1.4142135623730950488016887242096980786 + 1.
˓→7320508075688772935274463415058723669*I
```

conjugate()

Return the complex conjugate of self.

EXAMPLES:

```sage
sage: QQbar(3 + 4*I).conjugate()
3 - 4*I

sage: QQbar.zeta(7).conjugate()
0.6234898018587335? - 0.7818314824680299?*I

sage: QQbar.zeta(7) + QQbar.zeta(7).conjugate()
1.246979603717467? + 0.?e-18*I
```

imag()

Return the imaginary part of self.

EXAMPLES:

```sage
sage: QQbar.zeta(7).imag()
0.7818314824680299?

>>> from sage.all import *

>>> QQbar.zeta(Integer(7)).imag()
0.7818314824680299?
```
interval_exact(field)

Given a ComplexIntervalField, compute the best possible approximation of this number in that field. Note that if either the real or imaginary parts of this number are sufficiently close to some floating-point number (and, in particular, if either is exactly representable in floating-point), then this will trigger exact computation, which may be very slow.

EXAMPLES:

```
sage: a = QQbar(I).sqrt(); a
0.7071067811865475 + 0.7071067811865475*I
sage: a.interval_exact(CIF)
0.7071067811865475 + 0.7071067811865475*I
sage: b = QQbar((1+I)*sqrt(2)/2)
# needs sage.symbolic
sage: (a - b).interval(CIF)
# needs sage.symbolic
0.7e-19 + 0.2e-18*I
sage: (a - b).interval_exact(CIF)
# needs sage.symbolic
0
```

multiplicative_order()

Compute the multiplicative order of this algebraic number.

That is, find the smallest positive integer \( n \) such that \( x^n = 1 \). If there is no such \( n \), returns +Infinity.

We first check that \( \text{abs}(x) \) is very close to 1. If so, we compute \( x \) exactly and examine its argument.

EXAMPLES:

```
sage: QQbar(-sqrt(3)/2 - I/2).multiplicative_order()  # needs sage.symbolic
12
sage: QQbar(I).multiplicative_order()
1
sage: QQbar(-I).multiplicative_order()
4
sage: QQbar(707/1000 + 707/1000*I).multiplicative_order()
+Infinity
sage: QQbar(3/5 + 4/5*I).multiplicative_order()
+Infinity
```
norm()  
Return self * self.conjugate().  
This is the algebraic definition of norm, if we view QQbar as AA[I].  
EXAMPLES:

```
sage: QQbar(3 + 4*I).norm()
sage: type(QQbar(I).norm())
sage: QQbar.zeta(1007).norm()
```

rational_argument()  
Return the argument of self, divided by 2\pi, as long as this result is rational. Otherwise returns None. Always triggers exact computation.  
EXAMPLES:

```
sage: QQbar((1+I)*(sqrt(2)+sqrt(5))).rational_argument()
sage: QQbar(-1 + I*sqrt(3)).rational_argument()
sage: QQbar(-1 - I*sqrt(3)).rational_argument()
sage: QQbar(3+4*I).rational_argument()  
```

(continues on next page)
sage: (QQbar.zeta(3)^65536).rational_argument()
1/3

>>> from sage.all import *
>>> QQbar((Integer(1)+I)*(sqrt(Integer(2))+sqrt(Integer(5)))).rational_argument()  # needs sage.symbolic
1/8
>>> QQbar(-Integer(1) + I*sqrt(Integer(3))).rational_argument()  # needs sage.symbolic
1/3
>>> QQbar(-Integer(1) - I*sqrt(Integer(3))).rational_argument()  # needs sage.symbolic
-1/3
>>> QQbar(Integer(3)+Integer(4)*I).rational_argument() is None
True
>>> (QQbar(Integer(2))**(Integer(1)/Integer(5)) * QQbar.zeta(Integer(7))**Integer(2)).rational_argument()  # long time
2/7
>>> (QQbar.zeta(Integer(73))**Integer(5)).rational_argument()
5/73
>>> (QQbar.zeta(Integer(3))**Integer(65536)).rational_argument()
1/3

real()

Return the real part of self.

EXAMPLES:

sage: QQbar.zeta(5).real()
0.3090169943749474?

>>> from sage.all import *
>>> QQbar.zeta(Integer(5)).real()
0.3090169943749474?

class sage.rings.qqbar.AlgebraicNumberPowQQAction(G, S)

Bases: Action

Implement powering of an algebraic number (an element of QQbar or AA) by a rational.

This is always a right action.

INPUT:

- G – must be QQ
- S – the parent on which to act, either AA or QQbar.

Note: To compute \( x^{(a/b)} \), we take the \( b \)th root of \( x \); then we take that to the \( a \)th power. If \( x \) is a negative algebraic real and \( b \) is odd, take the real \( b \)th root; otherwise take the principal \( b \)th root.

EXAMPLES:

In QQbar:
Algebraic Numbers and Number Fields, Release 10.4

```
sage: QQbar(2)^(1/2)
1.414213562373095?

sage: QQbar(8)^(2/3)
4

sage: QQbar(8)^(2/3) == 4
True

sage: x = polygen(QQbar)
sage: phi = QQbar.polynomial_root(x^2 - x - 1, RIF(1, 2))
sage: tau = QQbar.polynomial_root(x^2 - x - 1, RIF(-1, 0))
sage: rt5 = QQbar(5)^(1/2)
sage: phi^10 / rt5
55.00363612324742?

sage: tau^10 / rt5
0.003636123247413266?

sage: (phi^10 - tau^10) / rt5
55.00000000000000?

sage: (phi^10 - tau^10) / rt5 == fibonacci(10)
True

sage: (phi^50 - tau^50) / rt5 == fibonacci(50)
True

sage: QQbar(-8)^(1/3)
1.000000000000000? + 1.732050807568878?*I

sage: (QQbar(-8)^(1/3))^3
-8

sage: QQbar(32)^(1/5)
2

sage: a = QQbar.zeta(7)^(1/3); a
0.9555728057861407? + 0.2947551744109043?*I

sage: a == QQbar.zeta(21)
True

sage: QQbar.zeta(7)^6
0.6234898018587335? - 0.7818314824680299?*I

sage: (QQbar.zeta(7)^6)^(1/3) * QQbar.zeta(21)
1.000000000000000? + 0.0e-17*I

>>> from sage.all import *

>>> QQbar(Integer(2))**(Integer(1)/Integer(2))
1.414213562373095?

>>> QQbar(Integer(8))**(Integer(2)/Integer(3))
4

>>> QQbar(Integer(8))**(Integer(2)/Integer(3)) == Integer(4)
True

>>> x = polygen(QQbar)

>>> phi = QQbar.polynomial_root(x**Integer(2) - x - Integer(1), RIF(Integer(1), -Integer(1)))

>>> tau = QQbar.polynomial_root(x**Integer(2) - x - Integer(1), RIF(-Integer(1), -Integer(0)))

>>> rt5 = QQbar(Integer(5))**(Integer(1)/Integer(2))

>>> phi**Integer(10) / rt5
55.00363612324742?

>>> tau**Integer(10) / rt5
0.003636123247413266?

>>> (phi**Integer(10) - tau**Integer(10)) / rt5
55.00000000000000?

>>> (phi**Integer(10) - tau**Integer(10)) / rt5 == fibonacci(Integer(10))
True

>>> (phi**Integer(50) - tau**Integer(50)) / rt5 == fibonacci(Integer(50))

(continues on next page)
True

```python
>>> QQbar(-Integer(8))**(Integer(1)/Integer(3))
1.000000000000000? + 1.732050807568878?*I
>>> (QQbar(-Integer(8))**(Integer(1)/Integer(3)))**(Integer(3))
-8
>>> QQbar(Integer(32))**(Integer(1)/Integer(5))
2
>>> a = QQbar.zeta(Integer(7))**(Integer(1)/Integer(3)); a
0.9555728057861407? + 0.2947551744109043?*I
>>> a == QQbar.zeta(Integer(21))
True
>>> QQbar.zeta(Integer(7))**Integer(6)
0.6234898018587335? - 0.7818314824680299?*I
>>> (QQbar.zeta(Integer(7))**Integer(6))**(Integer(1)/Integer(3)) * QQbar.zeta(Integer(21))
1.000000000000000? + 0.?e-17*I

In AA:

```python
sage: AA(2)^(1/2)
1.414213562373095?
sage: AA(8)^(2/3)
4
sage: AA(8)^(2/3) == Integer(4)
True
sage: x = polygen(AA)
sage: phi = AA.polynomial_root(x^2 - x - 1, RIF(0, 2))
sage: tau = AA.polynomial_root(x^2 - x - 1, RIF(-2, 0))
sage: rt5 = AA(Integer(5))^(1/2)
sage: phi^10 / rt5
55.00363612324742?
sage: tau^10 / rt5
0.003636123247413266?
sage: (phi^10 - tau^10) / rt5
55.00000000000000?
sage: (phi^10 - tau^10) / rt5 == fibonacci(10)
True
sage: (phi^50 - tau^50) / rt5 == fibonacci(50)
True
```

```python
>>> from sage.all import *
```

```python
>>> AA(Integer(2))**(Integer(1)/Integer(2))
1.414213562373095?
>>> AA(Integer(8))**(Integer(2)/Integer(3))
4
>>> AA(Integer(8))**(Integer(2)/Integer(3)) == Integer(4)
True
>>> x = polygen(AA)
>>> phi = AA.polynomial_root(x**Integer(2) - x - Integer(1), RIF(Integer(0), -Integer(2))
>>> tau = AA.polynomial_root(x**Integer(2) - x - Integer(1), RIF(-Integer(2), -Integer(0))
>>> rt5 = AA(Integer(5))**(Integer(1)/Integer(2))
>>> phi**Integer(10) / rt5
55.00363612324742?
>>> tau**Integer(10) / rt5
```

(continues on next page)
class sage.rings.qqbar.AlgebraicNumber_base (parent, x)

Bases: FieldElement

This is the common base class for algebraic numbers (complex numbers which are the zero of a polynomial in \( \mathbb{Z}[x] \)) and algebraic reals (algebraic numbers which happen to be real).

AlgebraicNumber objects can be created using QQbar (== AlgebraicNumberField()), and AlgebraicReal objects can be created using AA (== AlgebraicRealField()). They can be created either by coercing a rational or a symbolic expression, or by using the QQbar.polynomial_root() or AA.polynomial_root() method to construct a particular root of a polynomial with algebraic coefficients. Also, AlgebraicNumber and AlgebraicReal are closed under addition, subtraction, multiplication, division (except by 0), and rational powers (including roots), except that for a negative AlgebraicReal, taking a power with an even denominator returns an AlgebraicNumber instead of an AlgebraicReal.

AlgebraicNumber and AlgebraicReal objects can be approximated to any desired precision. They can be compared exactly; if the two numbers are very close, or are equal, this may require exact computation, which can be extremely slow.

As long as exact computation is not triggered, computation with algebraic numbers should not be too much slower than computation with intervals. As mentioned above, exact computation is triggered when comparing two algebraic numbers which are very close together. This can be an explicit comparison in user code, but the following list of actions (not necessarily complete) can also trigger exact computation:

- Dividing by an algebraic number which is very close to 0.
- Using an algebraic number which is very close to 0 as the leading coefficient in a polynomial.
- Taking a root of an algebraic number which is very close to 0.

The exact definition of “very close” is subject to change; currently, we compute our best approximation of the two numbers using 128-bit arithmetic, and see if that’s sufficient to decide the comparison. Note that comparing two algebraic numbers which are actually equal will always trigger exact computation, unless they are actually the same object.

EXAMPLES:

```
sage: sqrt(QQbar(2))
1.414213562373095?
sage: sqrt(QQbar(2))^2 == 2
True
sage: x = polygen(QQbar)
sage: phi = QQbar.polynomial_root(x^2 - x - 1, RIF(1, 2))
sage: phi
1.618033988749895?
sage: phi^2 == phi+1
True
sage: AA(sqrt(65537))  # needs sage.symbolic
256.0019531175495?
```
as_number_field_element (minimal=False, embedded=False, prec=53)

Return a number field containing this value, a representation of this value as an element of that number field, and a homomorphism from the number field back to AA or QQbar.

INPUT:

- minimal – Boolean (default: False). Whether to minimize the degree of the extension.
- embedded – Boolean (default: False). Whether to make the NumberField embedded.
- prec – integer (default: 53). The number of bit of precision to guarantee finding real roots.

This may not return the smallest such number field, unless minimal=True is specified.

To compute a single number field containing multiple algebraic numbers, use the function number_field_elements_from_algebraics instead.

EXAMPLES:

sage: QQbar(sqrt(8)).as_number_field_element()  # needs sage.symbolic
(Number Field in a with defining polynomial y^2 - 2, 2*a,
 Ring morphism:
    From: Number Field in a with defining polynomial y^2 - 2
    To:   Algebraic Real Field
    Defn: a |--> 1.414213562373095?)

sage: x = polygen(ZZ)
sage: p = x^3 + x^2 + x + 17
sage: (rt,) = p.roots(ring=AA, multiplicities=False); rt
-2.804642726932742?

sage: (nf, elt, hom) = rt.as_number_field_element()
sage: nf, elt, hom
(Number Field in a with defining polynomial y^3 - 2*y^2 - 31*y - 50, a^2 - 5*a - 19,
 Ring morphism:
    From: Number Field in a with defining polynomial y^3 - 2*y^2 - 31*y - 50
    To:   Algebraic Real Field
    Defn: a |--> 7.237653139801104?)

sage: elt == rt
False

sage: AA(elt)
256.0019531175495?
Creating an element of an embedded number field:

```
sage: (nf, elt, hom) = rt.as_number_field_element(embedded=True)
sage: nf.coerce_embedding()
Generic morphism:
    From: Number Field in a with defining polynomial y^3 - 2*y^2 - 31*y - 50
        with a = 7.237653139801104?
    To:      Algebraic Real Field
    Defn: a -> 7.237653139801104?
sage: elt
a^2 - 5*a - 19
sage: elt.parent() == nf
True
```

(continues on next page)
A complex algebraic number as an element of an embedded number field:

```python
sage: # needs sage.symbolic
sage: num = QQbar(sqrt(2) + 3**(1/3)*I)
sage: nf, elt, hom = num.as_number_field_element(embedded=True)
sage: hom(elt).parent() == QQbar
True
sage: nf.coerce_embedding() is not None
True
sage: QQbar(elt) == num == hom(elt)
True
```

We see an example where we do not get the minimal number field unless we specify `minimal=True`:
sage: # needs sage.symbolic
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt3b = rt2 + rt3 - rt2
sage: rt3b.as_number_field_element()
(Number Field in a with defining polynomial y^4 - 4*y^2 + 1, a^2 - 2,
   Ring morphism:
   From: Number Field in a with defining polynomial y^4 - 4*y^2 + 1
   To:    Algebraic Real Field
   Defn: a |--> -1.931851652578137?)
sage: rt3b.as_number_field_element(minimal=True)
(Number Field in a with defining polynomial y^2 - 3, a,
   Ring morphism:
   From: Number Field in a with defining polynomial y^2 - 3
   To:    Algebraic Real Field
   Defn: a |--> 1.732050807568878?)

>>> from sage.all import *

>>> # needs sage.symbolic

>>> rt2 = AA(sqrt(Integer(2)))
>>> rt3 = AA(sqrt(Integer(3)))
>>> rt3b = rt2 + rt3 - rt2
>>> rt3b.as_number_field_element()
(Number Field in a with defining polynomial y^4 - 4*y^2 + 1, a^2 - 2,
   Ring morphism:
   From: Number Field in a with defining polynomial y^4 - 4*y^2 + 1
   To:    Algebraic Real Field
   Defn: a |--> -1.931851652578137?)

>>> rt3b.as_number_field_element(minimal=True)
(Number Field in a with defining polynomial y^2 - 3, a,
   Ring morphism:
   From: Number Field in a with defining polynomial y^2 - 3
   To:    Algebraic Real Field
   Defn: a |--> 1.732050807568878?)

\textbf{degree}()

Return the degree of this algebraic number (the degree of its minimal polynomial, or equivalently, the degree of the smallest algebraic extension of the rationals containing this number).

\textbf{EXAMPLES:}

sage: QQbar(5/3).degree()
1
sage: sqrt(QQbar(2)).degree()
2
sage: QQbar(17).nth_root(5).degree()
5
sage: sqrt(3+sqrt(QQbar(8))).degree()
2

\texttt{(continues on next page)}
exactify()

Compute an exact representation for this number.

EXAMPLES:

```python
sage: two = QQbar(4).nth_root(4)**2
sage: two
2.000000000000000?
```

interval(field)

Given an interval (or ball) field (real or complex, as appropriate) of precision $p$, compute an interval representation of self with $\text{diameter()}$ at most $2^{-p}$; then round that representation into the given field. Here $\text{diameter()}$ is relative diameter for intervals not containing 0, and absolute diameter for intervals that do contain 0; thus, if the returned interval does not contain 0, it has at least $p - 1$ good bits.

EXAMPLES:

```python
sage: CIF64 = ComplexIntervalField(64)
sage: x = QQbar.zeta(11)
sage: x.interval(CIF64)
0.8412535328311811689? + 0.5406408174555975821*I
```

```python
>>> from sage.all import *
```

```python
>>> two = QQbar(Integer(4)).nth_root(Integer(4))**Integer(2)
>>> two
2.000000000000000?
```

```python
>>> from sage.all import *
```

```python
>>> RIF64 = RealIntervalField(Integer(64))
>>> x = AA(Integer(2)).sqrt()
>>> y = x*x
>>> y = Integer(1000) * y - Integer(999) * y
```

```python
>>> y.interval(RIF64)
2.000000000000000000?
```
2.000000000000000000?

```python
>>> CIF64 = ComplexIntervalField(Integer(64))
>>> x = QQbar.zeta(Integer(11))
>>> x.interval_fast(CIF64)
0.8412535328311811689? + 0.5406408174555975821?*I
```

```python
>>> x.interval(CIF64)
0.8412535328311811689? + 0.5406408174555975822?*I
```

```python
>>> x.interval(CBF) # abs tol 1e-16
[0.8412535328311812 +/- 3.12e-17] + [0.5406408174555976 +/- 1.79e-17]*I
```

The following implicitly use this method:

```python
sage: RIF(AA(5).sqrt())
2.236067797499790?
```

```python
sage: AA(-5).sqrt().interval(RIF)
```

Traceback (most recent call last):
  ...
TypeError: unable to convert 2.236067797499790?*I to real interval

```python
>>> from sage.all import *
```

```python
>>> RIF(AA(Integer(5)).sqrt())
2.236067797499790?
```

```python
>>> AA(-Integer(5)).sqrt().interval(RIF)
```

Traceback (most recent call last):
  ...
TypeError: unable to convert 2.236067797499790?*I to real interval

`interval_diameter(diam)`

Compute an interval representation of self with `diameter()` at most `diam`. The precision of the returned value is unpredictable.

**EXAMPLES:**

```python
sage: AA(2).sqrt().interval_diameter(1e-10)
1.4142135623730950488?
```

```python
sage: AA(2).sqrt().interval_diameter(1e-30)
1.41421356237309504880168872420969807857?
```

```python
sage: QQbar(2).sqrt().interval_diameter(1e-10)
1.4142135623730950488?
```

```python
sage: QQbar(2).sqrt().interval_diameter(1e-30)
1.41421356237309504880168872420969807857?
```

```python
>>> from sage.all import *
```

```python
>>> AA(Integer(2)).sqrt().interval_diameter(RealNumber('1e-10'))
1.4142135623730950488?
```

```python
>>> AA(Integer(2)).sqrt().interval_diameter(RealNumber('1e-30'))
1.41421356237309504880168872420969807857?
```

```python
>>> QQbar(Integer(2)).sqrt().interval_diameter(RealNumber('1e-10'))
1.4142135623730950488?
```

```python
>>> QQbar(Integer(2)).sqrt().interval_diameter(RealNumber('1e-30'))
1.41421356237309504880168872420969807857?
```

`interval_fast(field)`

Given a `RealIntervalField` or `ComplexIntervalField`, compute the value of this number using interval arithmetic of at least the precision of the field, and return the value in that field. (More precision may

5.1. Algebraic numbers 671
be used in the computation.) The returned interval may be arbitrarily imprecise, if this number is the result of a sufficiently long computation chain.

EXAMPLES:

```plaintext
sage: x = AA(2).sqrt()
sage: x.interval_fast(RIF)
1.414213562373095?
sage: x.interval_fast(RealIntervalField(200))
1.414213562373095048801688724209698078569671875376948073176680?
sage: x = QQbar(I).sqrt()
sage: x.interval_fast(CIF)
0.7071067811865475? + 0.7071067811865475?*I
sage: x.interval_fast(RIF)
Traceback (most recent call last):
... TypeError: unable to convert complex interval 0.7071067811865475244? + 0.
˓→7071067811865475244?*I to real interval
```

```plaintext
>>> from sage.all import *
>>> x = AA(Integer(2)).sqrt()
>>> x.interval_fast(RIF)
1.414213562373095?
>>> x.interval_fast(RealIntervalField(Integer(200)))
1.414213562373095048801688724209698078569671875376948073176680?
>>> x = QQbar(I).sqrt()
>>> x.interval_fast(CIF)
0.7071067811865475? + 0.7071067811865475?*I
>>> x.interval_fast(RIF)
Traceback (most recent call last):
... TypeError: unable to convert complex interval 0.7071067811865475244? + 0.
˓→7071067811865475244?*I to real interval
```

is_integer()  
Return True if this number is an integer.

EXAMPLES:

```plaintext
sage: QQbar(2).is_integer()
True
sage: QQbar(1/2).is_integer()
False
```

```plaintext
>>> from sage.all import *
>>> QQbar(Integer(2)).is_integer()
True
>>> QQbar(Integer(1)/Integer(2)).is_integer()
False
```

is_square()  
Return whether or not this number is square.

OUTPUT:

(boolean) True in all cases for elements of QQbar; True for non-negative elements of AA; otherwise False

EXAMPLES:
sage: AA(2).is_square()  
True  
sage: AA(-2).is_square()  
False  
sage: QQbar(-2).is_square()  
True  
sage: QQbar(I).is_square()  
True

>>> from sage.all import *  
>>> AA(Integer(2)).is_square()  
True  
>>> AA(-Integer(2)).is_square()  
False  
>>> QQbar(-Integer(2)).is_square()  
True  
>>> QQbar(I).is_square()  
True

minpoly()
Compute the minimal polynomial of this algebraic number. The minimal polynomial is the monic polynomial of least degree having this number as a root; it is unique.

EXAMPLES:

sage: QQbar(4).sqrt().minpoly()  
x - 2  
sage: ((QQbar(2).nth_root(4))^2).minpoly()  
x^2 - 2  
sage: v = sqrt(QQbar(2)) + sqrt(QQbar(3)); v  
3.146264369941973?  
sage: p = v.minpoly(); p  
x^4 - 10*x^2 + 1  
sage: p(RR(v.real()))  
1.31006316905768e-14

>>> from sage.all import *  
>>> QQbar(Integer(4)).sqrt().minpoly()  
x - 2  
>>> ((QQbar(Integer(2)).nth_root(Integer(4)))^2).minpoly()  
x^2 - 2  
>>> v = sqrt(QQbar(Integer(2))) + sqrt(QQbar(Integer(3))); v  
3.146264369941973?  
>>> p = v.minpoly(); p  
x^4 - 10*x^2 + 1  
>>> p(RR(v.real()))  
1.31006316905768e-14

nth_root (n, all=False)
Return the n-th root of this number.

INPUT:

• all – bool (default: False). If True, return a list of all n-th roots as complex algebraic numbers.
**Warning:** Note that for odd \( n \), all=`False` and negative real numbers, AlgebraicReal and AlgebraicNumber values give different answers: AlgebraicReal values prefer real results, and AlgebraicNumber values return the principal root.

**EXAMPLES:**

```sage
sage: AA(-8).nth_root(3)
-2
sage: QQbar(-8).nth_root(3)
1.000000000000000? + 1.732050807568878?*I
sage: QQbar.zeta(12).nth_root(15)
0.9993908270190957? + 0.03489949670250097?*I
```

```python
>>> from sage.all import *
```

```sage
>>> AA(-Integer(8)).nth_root(Integer(3))
-2
>>> QQbar(-Integer(8)).nth_root(Integer(3))
1.000000000000000? + 1.732050807568878?*I
>>> QQbar.zeta(Integer(12)).nth_root(Integer(15))
0.9993908270190957? + 0.03489949670250097?*I
```

You can get all \( n \)-th roots of algebraic numbers:

```sage
sage: AA(-8).nth_root(3, all=True)
[1.000000000000000? + 1.732050807568878?*I,
-2.000000000000000? + 0.?e-18*I,
1.000000000000000? - 1.732050807568878?*I]
```

```python
>>> from sage.all import *
```

```sage
>>> AA(-Integer(8)).nth_root(Integer(3), all=True)
[1.000000000000000? + 1.732050807568878?*I,
-2.000000000000000? + 0.?e-18*I,
1.000000000000000? - 1.732050807568878?*I]
```

```sage
>>> QQbar(I+I).nth_root(4, all=True)
[1.069553932363986? + 0.2127475047267431?*I,
-0.2127475047267431? + 1.069553932363986?*I,
-1.069553932363986? - 0.2127475047267431?*I,
0.2127475047267431? - 1.069553932363986?*I]
```

```python
>>> from sage.all import *
```

```sage
>>> AA(-Integer(8)).nth_root(Integer(3), all=True)
[1.000000000000000? + 1.732050807568878?*I,
-2.000000000000000? + 0.?e-18*I,
1.000000000000000? - 1.732050807568878?*I]
```

```sage
>>> QQbar(Integer(1)+I).nth_root(Integer(4), all=True)
[1.069553932363986? + 0.2127475047267431?*I,
-0.2127475047267431? + 1.069553932363986?*I,
-1.069553932363986? - 0.2127475047267431?*I,
0.2127475047267431? - 1.069553932363986?*I]
```

**radical_expression()**

Attempt to obtain a symbolic expression using radicals. If no exact symbolic expression can be found, the algebraic number will be returned without modification.

**EXAMPLES:**

```sage
sage: # needs sage.symbolic
sage: AA(1/sqrt(5)).radical_expression()
sqrt(1/5)
```

(continues on next page)
```sage
sage: AA(sqrt(5 + sqrt(5))).radical_expression()
sqrt(5 + sqrt(5))
sage: QQbar.zeta(5).radical_expression()
1/4*sqrt(5) + 1/2*sqrt(-1/2*sqrt(5) - 5/2) - 1/4
sage: x = polygen(QQ, 'x')
sage: a = (x^7 - x - 1).roots(AA, False)[0]
sage: a.radical_expression()
1.112775684278706?
sage: a.radical_expression().parent() == SR
False
sage: a = sorted((x^7-x-1).roots(QQbar, False), key=imag)[0]
sage: a.radical_expression()
-0.3636235193291805? - 0.9525611952610331?*I
sage: QQbar.zeta(5).imag().radical_expression()
1/2*sqrt(1/2*sqrt(5) + 5/2)
sage: AA(5/3).radical_expression()
5/3
sage: AA(5/3).radical_expression().parent() == SR
True
sage: QQbar(0).radical_expression()
0
```

```sage
>>> from sage.all import *
>>> # needs sage.symbolic
>>> AA(Integer(1)/sqrt(Integer(5))).radical_expression()
sqrt(1/5)
>>> AA(sqrt(Integer(5) + sqrt(Integer(5))).radical_expression()
sqrt(5 + sqrt(5))
>>> QQbar.zeta(Integer(5)).radical_expression()
1/4*sqrt(5) + 1/2*sqrt(-1/2*sqrt(5) - 5/2) - 1/4
>>> x = polygen(QQ, 'x')
>>> a = (x**Integer(7) - x - Integer(1)).roots(AA, False)[0]
>>> a.radical_expression()
1.112775684278706?
>>> a.radical_expression().parent() == SR
False
>>> a = sorted((x**Integer(7)-x-Integer(1)).roots(QQbar, False), key=imag)[0]
>>> a.radical_expression()
-0.3636235193291805? - 0.9525611952610331?*I
>>> QQbar.zeta(Integer(5)).imag().radical_expression()
1/2*sqrt(1/2*sqrt(5) + 5/2)
>>> AA(Integer(5)/Integer(3)).radical_expression()
5/3
>>> AA(Integer(5)/Integer(3)).radical_expression().parent() == SR
True
>>> QQbar(Integer(0)).radical_expression()
0
```

`simplify()`

Compute an exact representation for this number, in the smallest possible number field.

EXAMPLES:

```sage
sage: # needs sage.symbolic
sage: rt2 = AA(sqrt(2))
```

(continues on next page)
\texttt{sage}: rt3 = AA(sqrt(3))
\texttt{sage}: rt2b = rt3 + rt2 - rt3
\texttt{sage}: rt2b.exactify()
\texttt{sage}: rt2b._exact_value()
\begin{align*}
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415? \\
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415? \\
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415? \\
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415?
\end{align*}
\texttt{sage}: rt2b.simplify()
\texttt{sage}: rt2b._exact_value()
\begin{align*}
a & \text{ where } a^2 - 2 = 0 \text{ and } a \text{ in } 1.414213562373095? \\
a & \text{ where } a^2 - 2 = 0 \text{ and } a \text{ in } 1.414213562373095? \\
a & \text{ where } a^2 - 2 = 0 \text{ and } a \text{ in } 1.414213562373095?
\end{align*}

\texttt{from sage.all import *}
\texttt{# needs sage.symbolic}
\texttt{rt2 = AA(sqrt(Integer(2)))}
\texttt{rt3 = AA(sqrt(Integer(3)))}
\texttt{rt2b = rt3 + rt2 - rt3}
\texttt{rt2b.exactify()}
\begin{align*}
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415? \\
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415? \\
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415? \\
a^3 - 3*a & \text{ where } a^4 - 4*a^2 + 1 = 0 \text{ and } a \text{ in } -0.5176380902050415?
\end{align*}
\texttt{rt2b.simplify()}
\texttt{rt2b._exact_value()}
\begin{align*}
a & \text{ where } a^2 - 2 = 0 \text{ and } a \text{ in } 1.414213562373095? \\
a & \text{ where } a^2 - 2 = 0 \text{ and } a \text{ in } 1.414213562373095? \\
a & \text{ where } a^2 - 2 = 0 \text{ and } a \text{ in } 1.414213562373095?
\end{align*}

\texttt{sage}: AA(2).sqrt()
1.414213562373095?

\texttt{sage}: QQbar(I).sqrt()
0.7071067811865475? + 0.7071067811865475?*I
\texttt{sage}: QQbar(I).sqrt(\texttt{all=True})
[0.7071067811865475? + 0.7071067811865475?*I, -0.7071067811865475? - 0.7071067811865475?*I]

\texttt{sage}: a = QQbar(0)
\texttt{sage}: a.sqrt()
0
\texttt{sage}: a.sqrt(\texttt{all=True})
[0]

\texttt{sage}: a = AA(0)
\texttt{sage}: a.sqrt()
0
\texttt{sage}: a.sqrt(\texttt{all=True})
[0]
This second example just shows that the program does not care where 0 is defined, it gives the same answer regardless. After all, how many ways can you square-root zero?

```
sage: AA(-2).sqrt()
1.414213562373095*I
sage: AA(-2).sqrt(all=True)
[1.414213562373095*I, -1.414213562373095*I]
sage: AA(-2).sqrt(extend=False)
Traceback (most recent call last):
... ValueError: -2 is not a square in AA, being negative. Use extend = True for a... square root in QQbar.
```

```python
class sage.rings.qqbar.AlgebraicPolynomialTracker(poly)
    Bases: SageObject

    Keeps track of a polynomial used for algebraic numbers.
    If multiple algebraic numbers are created as roots of a single polynomial, this allows the polynomial and information about the polynomial to be shared. This reduces work if the polynomial must be recomputed at higher precision, or if it must be factored.
```
This class is private, and should only be constructed by \texttt{AA.common_polynomial()} or \texttt{QQbar.common_polynomial()}, and should only be used as an argument to \texttt{AA.polynomial_root()} or \texttt{QQbar.polynomial_root()}. (It does not matter whether you create the common polynomial with \texttt{AA.common_polynomial()} or \texttt{QQbar.common_polynomial()}).

**EXAMPLES:**

```python
sage: x = polygen(QQbar)
sage: P = QQbar.common_polynomial(x^2 - x - 1)
sage: P
x^2 - x - 1
sage: QQbar.polynomial_root(P, RIF(1, 2))
1.618033988749895?
```

```python
>>> from sage.all import *
>>> x = polygen(QQbar)
>>> P = QQbar.common_polynomial(x**Integer(2) - x - Integer(1))
>>> P
x^2 - x - 1
>>> QQbar.polynomial_root(P, RIF(Integer(1), Integer(2)))
1.618033988749895?
```

**complex_roots** (*prec, multiplicity*)

Find the roots of \texttt{self} in the complex field to precision \texttt{prec}.

**EXAMPLES:**

```python
sage: x = polygen(ZZ)
sage: cp = AA.common_polynomial(x^4 - 2)
```

```python
>>> from sage.all import *
>>> x = polygen(ZZ)
>>> cp = AA.common_polynomial(x**Integer(4) - Integer(2))
```

Note that the precision is not guaranteed to find the tightest possible interval since \texttt{complex_roots()} depends on the underlying BLAS implementation.

```python
sage: cp.complex_roots(30, 1)
[-1.18920711500272...?,
 1.189207115002721?,
-1.189207115002721?*I,
 1.189207115002721?*I]
```

```python
>>> from sage.all import *
>>> cp.complex_roots(Integer(30), Integer(1))
[-1.18920711500272...?,
 1.189207115002721?,
-1.189207115002721?*I,
 1.189207115002721?*I]
```

**exactify()**

Compute a common field that holds all of the algebraic coefficients of this polynomial, then factor the polynomial over that field. Store the factors for later use (ignoring multiplicity).

**EXAMPLES:**
sage: x = polygen(AA)
sage: p = sqrt(AA(2)) * x^2 - sqrt(AA(3))
sage: cp = AA.common_polynomial(p)
sage: cp._exact
False
sage: cp.exactify()

factors()

EXAMPLES:

sage: x = polygen(QQ)
sage: f = QQbar.common_polynomial(x^4 + 4)
sage: f.factors()
[y^2 - 2*y + 2, y^2 + 2*y + 2]

is_complex()

Return True if the coefficients of this polynomial are non-real.

EXAMPLES:
poly()

Return the underlying polynomial of self.

EXAMPLES:

```python
sage: x = polygen(QQ)
sage: f = x^3 - 7
sage: g = AA.common_polynomial(f)
sage: g.poly()
y^3 - 7
```

```
>>> from sage.all import *
>>> x = polygen(QQ)
>>> f = x**Integer(3) - Integer(7)
>>> g = AA.common_polynomial(f)
>>> g.poly()
y^3 - 7
```

class sage.rings.qqbar.AlgebraicReal(x)

Bases: AlgebraicNumber_base

A real algebraic number.

_richcmp_(other, op)

Compare two algebraic reals.

EXAMPLES:

```python
sage: AA(2).sqrt() < AA(3).sqrt()
True
sage: ((5+AA(5).sqrt())/2).sqrt() == 2*QQbar.zeta(5).imag()
True
sage: AA(3).sqrt() + AA(2).sqrt() < 3
False
```

```
>>> from sage.all import *
>>> AA(Integer(2)).sqrt() < AA(Integer(3)).sqrt()
True
>>> ((Integer(5)+AA(Integer(5)).sqrt())/Integer(2)).sqrt() == Integer(2)*QQbar.zeta(Integer(5)).imag()
True
```
ceiling()

Return the smallest integer not smaller than self.

EXAMPLES:

```python
sage: AA(sqrt(2)).ceil()  # needs sage.symbolic
2
sage: AA(-sqrt(2)).ceil()  # needs sage.symbolic
-1
sage: AA(42).ceil()
42
```

conjugate()

Return the complex conjugate of self, i.e. returns itself.

EXAMPLES:

```python
sage: a = AA(sqrt(2) + sqrt(3))  # needs sage.symbolic
sage: a.conjugate()  # needs sage.symbolic
3.146264369941973?
```

floor()

Return the largest integer not greater than self.

EXAMPLES:
imag()  
Return the imaginary part of this algebraic real.  
It always returns 0.  
EXAMPLES:

```sage
sage: a = AA(sqrt(2) + sqrt(3))  # needs sage.symbolic
sage: a.imag()  # needs sage.symbolic
0
```

interval_exact(field)  
Given a RealIntervalField, compute the best possible approximation of this number in that field.  
Note that if this number is sufficiently close to some floating-point number (and, in particular, if this number is exactly representable in floating-point), then this will trigger exact computation, which may be very slow.  
EXAMPLES:

```sage
sage: x = AA(2).sqrt()
sage: y = x*x
sage: x.interval(RIF)
1.414213562373095?
sage: x.interval_exact(RIF)
(continues on next page)
```
1.414213562373095?
\texttt{sage}: \ y.\ interval(\texttt{RIF})
2.000000000000000?
\texttt{sage}: \ y.\ interval\_exact(\texttt{RIF})
2
\texttt{sage}: \ z = 1 + \texttt{AA}(2)\ .\ sqrt() \ / \ 2^{200}
\texttt{sage}: \ z.\ interval(\texttt{RIF})
1.000000000000001?
\texttt{sage}: \ z.\ interval\_exact(\texttt{RIF})
1.000000000000001?

\begin{verbatim}
>>> from sage.all import *

>>> x = \texttt{AA}(\texttt{Integer}(2))\ .\ sqrt()

>>> y = x*x

>>> x.\ interval(\texttt{RIF})
1.414213562373095?

>>> x.\ interval\_exact(\texttt{RIF})
1.414213562373095?

>>> y.\ interval(\texttt{RIF})
2.000000000000000?

>>> y.\ interval\_exact(\texttt{RIF})
2

>>> z = \texttt{Integer}(1) + \texttt{AA}(\texttt{Integer}(2))\ .\ sqrt() \ / \ \texttt{Integer}(2)^{\texttt{Integer}(200)}

>>> z.\ interval(\texttt{RIF})
1.000000000000001?

>>> z.\ interval\_exact(\texttt{RIF})
1.000000000000001?
\end{verbatim}

**multiplicative\_order()**

Compute the multiplicative order of this real algebraic number. That is, find the smallest positive integer \( n \) such that \( x^n = 1 \). If there is no such \( n \), returns \(+\text{Infinity}\).

We first check that \( \text{abs}(x) \) is very close to 1. If so, we compute \( x \) exactly and compare it to 1 and \(-1\).

**EXAMPLES:**

\texttt{sage}: \ \texttt{AA}(1)\ .\ \texttt{multiplicative\_order()}
1

\texttt{sage}: \ \texttt{AA}(-1)\ .\ \texttt{multiplicative\_order()}
2

\texttt{sage}: \ \texttt{AA}(5)\ .\ \texttt{sqrt()}\ .\ \texttt{multiplicative\_order()}
+\text{Infinity}

\begin{verbatim}
>>> from sage.all import *

>>> \texttt{AA}(\texttt{Integer}(1))\ .\ \texttt{multiplicative\_order()}
1

>>> \texttt{AA}(-\texttt{Integer}(1))\ .\ \texttt{multiplicative\_order()}
2

>>> \texttt{AA}(\texttt{Integer}(5))\ .\ \texttt{sqrt()}\ .\ \texttt{multiplicative\_order()}
+\text{Infinity}
\end{verbatim}

**real()**

Return the real part of this algebraic real. It always returns \texttt{self}.

**EXAMPLES:**

5.1. Algebraic numbers
real_exact(field)

Given a `RealField`, compute the best possible approximation of this number in that field. Note that if this number is sufficiently close to the halfway point between two floating-point numbers in the field (for the default round-to-nearest mode) or if the number is sufficiently close to a floating-point number in the field (for directed rounding modes), then this will trigger exact computation, which may be very slow.

The rounding mode of the field is respected.

EXAMPLES:

```sage
def x = AA(2).sqrt()^2
sage: x
2.00000000000000
sage: x.real_exact(RR)
2.00000000000000
sage: x.real_exact(RealField(53, rnd='RNDD'))
2.00000000000000
sage: x.real_exact(RealField(53, rnd='RNDU'))
2.00000000000000
sage: x.real_exact(RealField(53, rnd='RNDZ'))
2.00000000000000

sage: (-x).real_exact(RR)
-2.00000000000000
sage: (-x).real_exact(RealField(53, rnd='RNDD'))
-2.00000000000000
sage: (-x).real_exact(RealField(53, rnd='RNDU'))
-2.00000000000000
sage: (-x).real_exact(RealField(53, rnd='RNDZ'))
-2.00000000000000

sage: y = (x-2).real_exact(RR).abs()
sage: y
0.0
sage: y == 0.0 or y == -0.0 # the sign of 0.0 is not significant in MPFI
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDD'))
2.00000000000000
sage: y == 0.0 or y == -0.0 # same as above
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDU'))
-2.00000000000000
sage: y == 0.0 or y == -0.0 # idem
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDZ'))
-2.00000000000000
sage: y == 0.0 or y == -0.0 # ibidem
```
Algebraic Numbers and Number Fields, Release 10.4

Real number (field)

Given a RealField, compute a good approximation to self in that field. The approximation will be off by at most two ulp's, except for numbers which are very close to 0, which will have an absolute error at most $2**(-\text{field.prec()-1})$. Also, the rounding mode of the field is respected.
EXAMPLES:

```
sage: x = AA(2).sqrt()^2
sage: x.real_number(RR)
2.00000000000000
sage: x.real_number(RealField(53, rnd='RNDD'))
1.99999999999999
sage: x.real_number(RealField(53, rnd='RNDU'))
2.00000000000001
sage: x.real_number(RealField(53, rnd='RNDZ'))
1.99999999999999
sage: (-x).real_number(RR)
-2.00000000000000
sage: (-x).real_number(RealField(53, rnd='RNDD'))
-2.00000000000001
sage: (-x).real_number(RealField(53, rnd='RNDU'))
-1.99999999999999
sage: (-x).real_number(RealField(53, rnd='RNDZ'))
-1.99999999999999
sage: (x-2).real_number(RR)
5.42101086242752e-20
sage: (x-2).real_number(RealField(53, rnd='RNDD'))
-1.08420217248551e-19
sage: (x-2).real_number(RealField(53, rnd='RNDU'))
2.16840434497101e-19
sage: (x-2).real_number(RealField(53, rnd='RNDZ'))
0.000000000000000
sage: y = AA(2).sqrt()
sage: y.real_number(RR)
1.41421356237309
sage: y.real_number(RealField(53, rnd='RNDD'))
1.41421356237309
sage: y.real_number(RealField(53, rnd='RNDU'))
1.41421356237310
sage: y.real_number(RealField(53, rnd='RNDZ'))
1.41421356237309
```

```
-1.08420217248551e-19
>>> (x=Integer(2)).real_number(RealField(Integer(53), rnd='RNDU'))
2.16840434497101e-19
>>> (x=Integer(2)).real_number(RealField(Integer(53), rnd='RNDZ'))
0.000000000000000
>>> y = AA(Integer(2)).sqrt()
>>> y.real_number(RR)
1.41421356237309
>>> y.real_number(RealField(Integer(53), rnd='RNDD'))
1.41421356237309
>>> y.real_number(RealField(Integer(53), rnd='RNDU'))
1.41421356237310
>>> y.real_number(RealField(Integer(53), rnd='RNDZ'))
1.41421356237309

round()
Round self to the nearest integer.

EXAMPLES:

```
sage: AA(sqrt(2)).round()  # needs sage.symbolic
1
sage: AA(1/2).round()
1
sage: AA(-1/2).round()
-1
```

```
>>> from sage.all import *
>>> AA(sqrt(Integer(2))).round()  # needs sage.symbolic
1
>>> AA(Integer(1)/Integer(2)).round()
1
>>> AA(-Integer(1)/Integer(2)).round()
-1
```

sign()
Compute the sign of this algebraic number (return −1 if negative, 0 if zero, or 1 if positive).

This computes an interval enclosing this number using 128-bit interval arithmetic; if this interval includes 0, then fall back to exact computation (which can be very slow).

EXAMPLES:

```
sage: AA(-5).nth_root(7).sign()
-1
sage: (AA(2).sqrt() - AA(2).sqrt()).sign()
0
```

```
sage: a = AA(2).sqrt() + AA(3).sqrt() - 58114382797550084497/1847091534626475921
sage: a.sign()
1
sage: b = AA(2).sqrt() + AA(3).sqrt() - 2602510228533039296408/82714681630786895911
sage: b.sign()
```

(continues on next page)
\begin{verbatim}
-1
sage: c = AA(5)**(1/3) - 1437624125539676934786/84072688792155114277
sage: c.sign()
1
sage: (((a+b)*(a+c)*(b+c))**9 / (a*b*c)).sign()
1
sage: (a-b).sign()
1
sage: (b-a).sign()
-1
sage: (a*b).sign()
-1
sage: ((a*b).abs() + a).sign()
1
sage: (a*b - b*a).sign()
0
sage: a = AA(sqrt(1/2))
# → needs sage.symbolic
sage: b = AA(-sqrt(1/2))
# → needs sage.symbolic
sage: (a + b).sign()
# → needs sage.symbolic
0

>>> from sage.all import *
>>> AA(-Integer(5)).nth_root(Integer(7)).sign()
-1
>>> (AA(Integer(2)).sqrt() - AA(Integer(2)).sqrt()).sign()
0

>>> a = AA(Integer(2)).sqrt() - AA(Integer(3)).sqrt() - Integer(58114382797550084497)/Integer(18470915334626475921)
>>> a.sign()
1

>>> b = AA(Integer(2)).sqrt() + AA(Integer(3)).sqrt() - Integer(2602510228533039296408)/Integer(827174681630786895911)
>>> b.sign()
-1

>>> c = AA(Integer(5))**(Integer(1)/Integer(3)) - Integer(1437624125539676934786)/Integer(84072688792155114277)
>>> c.sign()
1

>>> (((a+b)*(a+c)*(b+c))**Integer(9) / (a*b*c)).sign()
1
>>> (a-b).sign()
1
>>> (b-a).sign()
-1
>>> (a*b).sign()
-1
>>> ((a*b).abs() + a).sign()

\end{verbatim}
trunc()
Round self to the nearest integer toward zero.

EXAMPLES:

```
sage: AA(sqrt(2)).trunc()  # needs sage.symbolic
1
sage: AA(-sqrt(2)).trunc()  # needs sage.symbolic
-1
sage: AA(1).trunc()
1
sage: AA(-1).trunc()
-1
```

```
>>> from sage.all import *
```
completion\((p, \text{prec}, \text{extras}=/)\)

Return the completion of self at the place \(p\).

Only implemented for \(p = \infty\) at present.

INPUT:

- \(p\) – either a prime (not implemented at present) or Infinity
- \(\text{prec}\) – precision of approximate field to return
- \(\text{extras}\) – (optional) a dict of extra keyword arguments for the RealField constructor

EXAMPLES:

```
sage: AA.completion(infinity, 500)
Real Field with 500 bits of precision
sage: AA.completion(infinity, prec=53, extras={'type':'RDF'})
Real Double Field
sage: AA.completion(infinity, 53) is RR
True
sage: AA.completion(7, 10)
Traceback (most recent call last):
...NotImplementedError
```

\[\text{gen}(n=0)\]

Return the \(n\)-th element of the tuple returned by \textit{gens()}.

EXAMPLES:

```
sage: AA.gen(0)
1
sage: AA.gen(1)
Traceback (most recent call last):
...IndexError: n must be 0
```

```
>>> from sage.all import *
>>> AA.completion(infinity, Integer(500))
Real Field with 500 bits of precision
>>> AA.completion(infinity, prec=Integer(53), extras={'type':'RDF'})
Real Double Field
>>> AA.completion(infinity, Integer(53)) is RR
True
>>> AA.completion(Integer(7), Integer(10))
Traceback (most recent call last):
...NotImplementedError
```
**gens()**

Return a set of generators for this field.

As this field is not finitely generated, we opt for just returning 1.

**EXAMPLES:**

```sage
sage: AA.gens()
(1,)
```

```python
>>> from sage.all import *

>>> AA.gens()
(1,)
```

**ngens()**

Return the size of the tuple returned by `gens()`.

**EXAMPLES:**

```sage
sage: AA.ngens()
1
```

```python
>>> from sage.all import *

>>> AA.ngens()
1
```

**polynomial_root (poly, interval, multiplicity=1)**

Given a polynomial with algebraic coefficients and an interval enclosing exactly one root of the polynomial, constructs an algebraic real representation of that root.

The polynomial need not be irreducible, or even squarefree; but if the given root is a multiple root, its multiplicity must be specified. (IMPORTANT NOTE: Currently, multiplicity-\(k\) roots are handled by taking the \((k-1)\)-st derivative of the polynomial. This means that the interval must enclose exactly one root of this derivative.)

The conditions on the arguments (that the interval encloses exactly one root, and that multiple roots match the given multiplicity) are not checked; if they are not satisfied, an error may be thrown (possibly later, when the algebraic number is used), or wrong answers may result.

Note that if you are constructing multiple roots of a single polynomial, it is better to use `AA.common_polynomial` (or `QQbar.common_polynomial`; the two are equivalent) to get a shared polynomial.

**EXAMPLES:**

```sage
sage: x = polygen(AA)
sage: phi = AA.polynomial_root(x^2 - x - 1, RIF(1, 2)); phi
1.618033988749895?
sage: p = (x-1)^7 * (x-2)
sage: r = AA.polynomial_root(p, RIF(9/10, 11/10), multiplicity=7)
sage: r; r == 1
1.000000000000000?
True
sage: p = (x-phi)*(x-sqrt(AA(2)))
sage: r = AA.polynomial_root(p, RIF(1, 3/2))
sage: r; r == sqrt(AA(2))
1.414213562373095?
True
```

5.1. Algebraic numbers 691
We allow complex polynomials, as long as the particular root in question is real.

```python
sage: K.<im> = QQ[I]
sage: x = polygen(K)
sage: p = (im + Integer(1)) * (x**Integer(3) - Integer(2)); p
(I + 1)*x^3 - 2*I - 2
sage: r = AA.polynomial_root(p, RIF(Integer(1), Integer(2))); r**Integer(3)
2.000000000000000?
```

random_element (poly_degree=2, *args, **kwds)

Return a random algebraic real number.

**INPUT:**

- **poly_degree** – default: 2; degree of the random polynomial over the integers of which the returned algebraic real number is a (real part of a) root. This is not necessarily the degree of the minimal polynomial of the number. Increase this parameter to achieve a greater diversity of algebraic numbers, at a cost of greater computation time. You can also vary the distribution of the coefficients but that will not vary the degree of the extension containing the element.

- **args, kwds** – arguments and keywords passed to the random number generator for elements of \(\mathbb{Z}\), the integers. See `random_element()` for details, or see example below.

**OUTPUT:**

An element of \(\mathbb{AA}\), the field of algebraic real numbers (see `sage.rings.qqbar`).

**ALGORITHM:**

We pass all arguments to `AlgebraicField.random_element()`, and then take the real part of the result.

**EXAMPLES:**
Algebraic Numbers and Number Fields, Release 10.4

```
sage: a = AA.random_element()
sage: a in AA
True

>>> from sage.all import *

>>> a = AA.random_element()

>>> a in AA
True

sage: b = AA.random_element(poly_degree=5)
sage: b in AA
True

>>> from sage.all import *

>>> b = AA.random_element(poly_degree=Integer(5))

>>> b in AA
True
```

Parameters for the distribution of the integer coefficients of the polynomials can be passed on to the random element method for integers. For example, we can rule out zero as a coefficient (and therefore as a root) by requesting coefficients between 1 and 10:

```
sage: z = [AA.random_element(x=1, y=10) for _ in range(5)]
sage: AA(0) in z
False

>>> from sage.all import *

>>> z = [AA.random_element(x=Integer(1), y=Integer(10)) for _ in range(Integer(5))]

>>> AA(Integer(0)) in z
False
```

zeta \((n=2)\)

Return an \(n\)-th root of unity in this field. This will raise a ValueError if \(n \not\in \{1, 2\}\) since no such root exists.

INPUT:

- \(n\) (integer) – default 2

EXAMPLES:

```
sage: AA.zeta(1)
1
sage: AA.zeta(2)
-1
sage: AA.zeta()
-1
sage: AA.zeta(3)
Traceback (most recent call last):
... ValueError: no n-th root of unity in algebraic reals

>>> from sage.all import *

>>> AA.zeta(Integer(1))
1
```

(continues on next page)
Algebraic Numbers and Number Fields, Release 10.4

```
>>> AA.zeta(Integer(2))
-1
>>> AA.zeta()
-1
>>> AA.zeta(Integer(3))
Traceback (most recent call last):
...  
ValueError: no n-th root of unity in algebraic reals
```

Some silly inputs:

```
sage: AA.zeta(Mod(-5, 7))
-1
sage: AA.zeta(0)
Traceback (most recent call last):
...  
ValueError: no n-th root of unity in algebraic reals
```

```
>>> from sage.all import *

>>> AA.zeta(Mod(-Integer(5), Integer(7)))
-1
>>> AA.zeta(Integer(0))
Traceback (most recent call last):
...  
ValueError: no n-th root of unity in algebraic reals
```

`sage.rings.qqbar.an_binop_element(a, b, op)`

Add, subtract, multiply or divide two elements represented as elements of number fields.

**EXAMPLES:**

```
sage: sqtr2 = QQbar(2).sqrt()
sage: sqtr3 = QQbar(3).sqrt()
sage: sqtr5 = QQbar(5).sqrt()

sage: a = sqtr2 + sqtr3; a.exactify()
sage: b = sqtr3 + sqtr5; b.exactify()
sage: type(a._descr)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: from sage.rings.qqbar import an_binop_element
sage: an_binop_element(a, b, operator.add)
<sage.rings.qqbar.ANBinaryExpr object at ...>
sage: an_binop_element(a, b, operator.sub)
<sage.rings.qqbar.ANBinaryExpr object at ...>
sage: an_binop_element(a, b, operator.mul)
<sage.rings.qqbar.ANBinaryExpr object at ...>
sage: an_binop_element(a, b, operator.truediv)
<sage.rings.qqbar.ANBinaryExpr object at ...>
```

```
>>> from sage.all import *

>>> sqtr2 = QQbar(Integer(2)).sqrt()
>>> sqtr3 = QQbar(Integer(3)).sqrt()
>>> sqtr5 = QQbar(Integer(5)).sqrt()

>>> a = sqtr2 + sqtr3; a.exactify()
>>> b = sqtr3 + sqtr5; b.exactify()
```

(continues on next page)
The code tries to use existing unions of number fields:

```python
sage: sqrt17 = QQbar(17).sqrt()
sage: sqrt19 = QQbar(19).sqrt()
sage: a = sqrt17 + sqrt19
sage: b = sqrt17 * sqrt19 - sqrt17 + sqrt19 * (sqrt17 + 2)
sage: b, type(b._descr)
(40.53909377268655?, <class 'sage.rings.qqbar.ANBinaryExpr'>)
sage: a.exactify()
sage: b = sqrt17 * sqrt19 - sqrt17 + sqrt19 * (sqrt17 + Integer(2))
sage: b, type(b._descr)
(40.53909377268655?, <class 'sage.rings.qqbar.ANExtensionElement'>)
```

The code uses sage.rings.qqbar.an_binop_expr(a, b, op)

Add, subtract, multiply or divide algebraic numbers represented as binary expressions.

**INPUT:**

- a, b – two elements
- op – an operator

**EXAMPLES:**

```python
sage: from sage.rings.qqbar import an_binop_expr
sage: x = an_binop_expr(a, b, operator.add); x
```

### 5.1. Algebraic numbers

695
where $a^8 - 12a^6 + 23a^4 - 12a^2 + 1 = 0$ and $a$ in $-0.3199179336182997$?

```python
>>> from sage.all import *
>>> # needs sage.symbolic
>>> a = QQbar(sqrt(Integer(2))) + QQbar(sqrt(Integer(3)))
>>> b = QQbar(sqrt(Integer(3))) + QQbar(sqrt(Integer(5)))
>>> type(a._descr); type(b._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
<class 'sage.rings.qqbar.ANBinaryExpr'>
>>> from sage.rings.qqbar import an_binop_rational
>>> x = an_binop_rational(a, b, operator.mul); x
<sage.rings.qqbar.ANBinaryExpr object at ...>
>>> x.exactify()
2*a^7 - a^6 - 24*a^5 + 12*a^4 + 46*a^3 - 22*a^2 - 22*a + 9
where $a^8 - 12a^6 + 23a^4 - 12a^2 + 1 = 0$ and $a$ in $-0.3199179336182997$?
```
sage: type(f)
<class 'sage.rings.qqbar.ANRational'>

>>> from sage.all import *
>>> from sage.rings.qqbar import an_binop_rational
>>> f = an_binop_rational(QQbar(Integer(2)), QQbar(Integer(3)/Integer(7)), →
operator.add)
>>> f
17/7
>>> type(f)
<class 'sage.rings.qqbar.ANRational'>

>>> f = an_binop_rational(QQbar(Integer(2)), QQbar(Integer(3)/Integer(7)), →
operator.mul)
>>> f
6/7
>>> type(f)
<class 'sage.rings.qqbar.ANRational'>

sage.rings.qqbar.clear_denominators(poly)

Take a monic polynomial and rescale the variable to get a monic polynomial with “integral” coefficients.

This works on any univariate polynomial whose base ring has a denominator() method that returns integers; for example, the base ring might be \( \mathbb{Q} \) or a number field.

Returns the scale factor and the new polynomial.

(Inspired by pari:primitive_pol_to_monic.)

We assume that coefficient denominators are “small”; the algorithm factors the denominators, to give the smallest possible scale factor.

EXAMPLES:

sage: from sage.rings.qqbar import clear_denominators

sage: _.<x> = QQ['x']
sage: clear_denominators(x + 3/2)
(2, x + 3)
sage: clear_denominators(x^2 + x/2 + 1/4)
(2, x^2 + x + 1)

sage.rings.qqbar.cmp_elements_with_same_minpoly(a, b, p)

Compare the algebraic elements \( a \) and \( b \) knowing that they have the same minimal polynomial \( p \).

This is a helper function for comparison of algebraic elements (i.e. the methods \( \text{AlgebraicNumber.\_richcmp\_}() \) and \( \text{AlgebraicReal.\_richcmp\_}() \)).

INPUT:
• $a$ and $b$ – elements of the algebraic or the real algebraic field with same minimal polynomial
• $p$ – the minimal polynomial

OUTPUT:

$-1, 0, 1, None$ depending on whether $a < b$, $a = b$ or $a > b$ or the function did not succeed with the given precision of $a$ and $b$.

EXAMPLES:

```
sage: from sage.rings.qqbar import cmp_elements_with_same_minpoly
sage: x = polygen(ZZ)
sage: p = x^2 - 2
sage: a = AA.polynomial_root(p, RIF(1,2))
sage: b = AA.polynomial_root(p, RIF(-2,-1))
sage: cmp_elements_with_same_minpoly(a, b, p)
1
sage: cmp_elements_with_same_minpoly(-a, b, p)
0
```

`sage.rings.qqbar.conjugate_expand(v)`

If the interval $v$ (which may be real or complex) includes some purely real numbers, return $v'$ containing $v$ such that $v' = v'.conjugate()$. Otherwise return $v$ unchanged. (Note that if $v' = v'.conjugate()$, and $v'$ includes one non-real root of a real polynomial, then $v'$ also includes the conjugate of that root. Also note that the diameter of the return value is at most twice the diameter of the input.)

EXAMPLES:

```
sage: from sage.rings.qqbar import conjugate_expand
sage: conjugate_expand(CIF(RIF(0, 1), RIF(1, 2))).str(style='brackets')
'[0.000000000000000 .. 1.000000000000000] + [-1.000000000000000 .. 1.000000000000000]*I'
sage: conjugate_expand(CIF(RIF(0, 1), RIF(-2, -1))).str(style='brackets')
'[0.000000000000000 .. 1.000000000000000] + [2.000000000000000 .. -2.000000000000000]*I'
sage: conjugate_expand(RIF(1, 2)).str(style='brackets')
'[1.000000000000000 .. 2.000000000000000]'

>>> from sage.all import *
>>> from sage.rings.qqbar import conjugate_expand
>>>

(continues on next page)
After reading the previous page, we have:

```python
sage.rings.qqbar.conjugate_shrink(v)
```

If the interval \(v\) includes some purely real numbers, return a real interval containing only those real numbers. Otherwise return \(v\) unchanged.

If \(v\) includes exactly one root of a real polynomial, and \(v\) was returned by `conjugate_expand()`, then `conjugate_shrink(v)` still includes that root, and is a `RealIntervalFieldElement` if the root in question is real.

**EXAMPLES:**

```python
sage: from sage.rings.qqbar import conjugate_shrink
sage: conjugate_shrink(RIF(3, 4)).str(style='brackets')
'[3.0000000000000000 .. 4.0000000000000000]

sage: conjugate_shrink(CIF(RIF(1, 2), RIF(1, 2))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000] + [1.0000000000000000 .. 2.0000000000000000]*I

sage: conjugate_shrink(CIF(RIF(1, 2), RIF(0, 1))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000]

sage: conjugate_shrink(CIF(RIF(1, 2), RIF(-1, 2))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000]
```

```python
sage.rings.qqbar.do_polred(poly, threshold=32)
```

Find a polynomial of reasonably small discriminant that generates the same number field as \(poly\), using the PARI `polredbest` function.

**INPUT:**

- `poly` – a monic irreducible polynomial with integer coefficients
- `threshold` – an integer used to decide whether to run `polredbest`
OUTPUT:
A triple \((\text{elt}_\text{fwd}, \text{elt}_\text{back}, \text{new}_\text{poly})\), where:

- \(\text{new}_\text{poly}\) is the new polynomial defining the same number field,
- \(\text{elt}_\text{fwd}\) is a polynomial expression for a root of the new polynomial in terms of a root of the original polynomial,
- \(\text{elt}_\text{back}\) is a polynomial expression for a root of the original polynomial in terms of a root of the new polynomial.

EXAMPLES:

```
sage: from sage.rings.qqbar import do_polred
sage: R.<x> = QQ[x]
sage: oldpol = x^2 - 5
sage: fwd, back, newpol = do_polred(oldpol)
sage: newpol
x^2 - x - 1
sage: Kold.<a> = NumberField(oldpol)
sage: Knew.<b> = NumberField(newpol)
sage: newpol(fwd(a))
0
sage: oldpol(back(b))
0
sage: do_polred(x^2 - x - 11)
(1/3*x + 1/3, 3*x - 1, x^2 - x - 1)
sage: do_polred(x^3 + 123456)
(-1/4*x, -4*x, x^3 - 1929)
```

This shows that Issue #13054 has been fixed:

```
>>> from sage.all import *
>>> from sage.rings.qqbar import do_polred
>>> R = QQ['x']; (x,) = R._first_ngens(1)
>>> oldpol = x**Integer(2) - Integer(5)
>>> fwd, back, newpol = do_polred(oldpol)
>>> newpol
x^2 - x - 1
>>> Kold = NumberField(oldpol, names=('a',)); (a,) = Kold._first_ngens(1)
>>> Knew = NumberField(newpol, names=('b',)); (b,) = Knew._first_ngens(1)
>>> newpol(fwd(a))
0
>>> oldpol(back(b))
0
>>> do_polred(x**Integer(2) - x - Integer(11))
(1/3*x + 1/3, 3*x - 1, x^2 - x - 1)
>>> do_polred(x**Integer(3) + Integer(123456))
(-1/4*x, -4*x, x^3 - 1929)
```

This shows that Issue #13054 has been fixed:

```
sage: do_polred(x^4 - 4294967296*x^2 + 54265257667816538374400)
(1/4*x, 4*x, x^4 - 268435456*x^2 + 211973662764908353025)
```

This shows that Issue #13054 has been fixed:

```
>>> from sage.all import *
>>> do_polred(x**Integer(4) - Integer(4294967296)*x**Integer(2) + Integer(54265257667816538374400))
(1/4*x, 4*x, x^4 - 268435456*x^2 + 211973662764908353025)
```
sage.rings.qqbar.find_zero_result (fn, l)

l is a list of some sort. fn is a function which maps an element of l and a precision into an interval (either real or complex) of that precision, such that for sufficient precision, exactly one element of l results in an interval containing 0. Returns that one element of l.

EXAMPLES:

```python
sage: from sage.rings.qqbar import find_zero_result
sage: _.<x> = QQ['x']
sage: delta = 10^(-70)
sage: p1 = x - 1
sage: p2 = x - 1 - delta
sage: p3 = x - 1 + delta
sage: p2 == find_zero_result(lambda p, prec: p(RealIntervalField(prec)(1 + delta)), [p1, p2, p3])
True
```

sage.rings.qqbar.get_AA_golden_ratio()

Return the golden ratio as an element of the algebraic real field. Used by sage.symbolic.constants.golden_ratio._algebraic_().

EXAMPLES:

```python
sage: AA(golden_ratio)  # indirect doctest
1.618033988749895?
```

sage.rings.qqbar.is_AlgebraicField (F)

Check whether F is an AlgebraicField instance.

This function is deprecated. Use isinstance() with AlgebraicField instead.

EXAMPLES:

```python
sage: from sage.rings.qqbar import is_AlgebraicField
sage: [is_AlgebraicField(x) for x in [AA, QQbar, None, 0, "spam"]]
doctest:warning...
[False, True, False, False, False]
```

5.1. Algebraic numbers
sage.rings.qqbar.is_AlgebraicNumber(x)
Test if x is an instance of AlgebraicNumber. For internal use.

EXAMPLES:

sage: from sage.rings.qqbar import is_AlgebraicNumber
sage: is_AlgebraicNumber(AA(sqrt(2)))  # needs sage.symbolic
False
sage: is_AlgebraicNumber(QQbar(sqrt(2)))  # needs sage.symbolic
True
sage: is_AlgebraicNumber("spam")
False

sage.rings.qqbar.is_AlgebraicReal(x)
Test if x is an instance of AlgebraicReal. For internal use.

EXAMPLES:

sage: from sage.rings.qqbar import is_AlgebraicReal
sage: is_AlgebraicReal(AA(sqrt(2)))  # needs sage.symbolic
True
sage: is_AlgebraicReal(QQbar(sqrt(2)))  # needs sage.symbolic
True
sage: is_AlgebraicReal("spam")
False
sage.rings.qqbar.is_AlgebraicRealField(F)

Check whether \( F \) is an AlgebraicRealField instance. For internal use.

This function is deprecated. Use `isinstance()` with `AlgebraicRealField` instead.

EXAMPLES:

```python
sage: from sage.rings.qqbar import is_AlgebraicRealField
sage: [is_AlgebraicRealField(x) for x in [AA, QQbar, None, 0, "spam"]]
doctest:warning...
DeprecationWarning: `is_AlgebraicRealField` is deprecated; use `isinstance(..., sage.rings.abc.AlgebraicRealField)` instead.
See https://github.com/sagemath/sage/issues/32660 for details.
[True, False, False, False, False]
```

sage.rings.qqbar.isolating_interval(intv_fn, pol)

`intv_fn` is a function that takes a precision and returns an interval of that precision containing some particular root of `pol`. (It must return better approximations as the precision increases.) `pol` is an irreducible polynomial with rational coefficients.

Returns an interval containing at most one root of `pol`.

EXAMPLES:

```python
sage: from sage.rings.qqbar import isolating_interval
sage: _.<x> = QQ['x']
```

(continues on next page)
And an example that requires more precision:

```
sage: delta = 10^(-70)
sage: p = (x - 1) * (x - 1 - delta) * (x - 1 + delta)
sage: isolating_interval(lambda prec: RealIntervalField(prec)(1 + delta), p)
1.
˓→000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000?
˓→
```

The function also works with complex intervals and complex roots:

```
sage: p = x^2 - x + 13/36
sage: isolating_interval(lambda prec: ComplexIntervalField(prec)(1/2, 1/3), p)
0.500000000000000000000? + 0.3333333333333333334?*I
```

```
from sage.all import *
from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
>>> from sage.all import *
>>>
>>> _ = QQ['x']; (x,) = _.first_ngens(1)
>>> isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
>>> from sage.all import *
>>>
>>> _ = QQ['x']; (x,) = _.first_ngens(1)
>>> isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
>>> from sage.all import *

>>> from sage.rings.qqbar import isolating_interval

>>> _ = QQ['x']; (x,) = _.first_ngens(1)
>>> isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
>>> from sage.all import *

>>> from sage.rings.qqbar import isolating_interval

>>> _ = QQ['x']; (x,) = _.first_ngens(1)
>>> isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

The function also works with complex intervals and complex roots:

```
sage: p = x^2 - x + 13/36
sage: isolating_interval(lambda prec: ComplexIntervalField(prec)(1/2, 1/3), p)
0.500000000000000000000? + 0.3333333333333333334?*I
```

```
from sage.all import *
from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
from sage.all import *
from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
from sage.all import *

from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
from sage.all import *

from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

The function also works with complex intervals and complex roots:

```
sage: p = x^2 - x + 13/36
sage: isolating_interval(lambda prec: ComplexIntervalField(prec)(1/2, 1/3), p)
0.500000000000000000000? + 0.3333333333333333334?*I
```

```
from sage.all import *

from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
from sage.all import *

from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

The function also works with complex intervals and complex roots:

```
sage: p = x^2 - x + 13/36
sage: isolating_interval(lambda prec: ComplexIntervalField(prec)(1/2, 1/3), p)
0.500000000000000000000? + 0.3333333333333333334?*I
```

```
from sage.all import *

from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```
from sage.all import *

from sage.rings.qqbar import isolating_interval

_ = QQ['x']; (x,) = _.first_ngens(1)
isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(Integer(2))), x**Integer(2) - Integer(2))
1.4142135623730950488?
```

```sage:
from sage.rings.qqbar import number_field_elements_from_algebraics

sage.rings.qqbar.number_field_elements_from_algebraics(numbers, minimal=False, same_field=False, embedded=False, name='a', prec=53)
```

Given a sequence of elements of either AA or QQbar (or a mixture), computes a number field containing all of these elements, these elements as members of that number field, and a homomorphism from the number field back to AA or QQbar.

**INPUT:**

- **numbers** – a number or list of numbers
- **minimal** – Boolean (default: False). Whether to minimize the degree of the extension
- **same_field** – Boolean (default: False). See below

**Chapter 5. Algebraic Numbers**

704
• **embedded** – Boolean (default: `False`). Whether to make the NumberField embedded
• **name** – string (default: `'a'`); name of the primitive element
• **prec** – integer (default: 53). The number of bit of precision to guarantee finding real roots

**OUTPUT:**

A tuple with the NumberField, the numbers inside the NumberField, and a homomorphism from the number field back to `AA` or `QQbar`.

This may not return the smallest such number field, unless `minimal=True` is specified.

If `same_field=True` is specified, and all of the elements are from the same field (either `AA` or `QQbar`), the generated homomorphism will map back to that field. Otherwise, if all specified elements are real, the homomorphism might map back to `AA` (and will, if `minimal=True` is specified), even if the elements were in `QQbar`.

Also, a single number can be passed, rather than a sequence; and any values which are not elements of `AA` or `QQbar` will automatically be coerced to `QQbar`.

This function may be useful for efficiency reasons: doing exact computations in the corresponding number field will be faster than doing exact computations directly in `AA` or `QQbar`.

**EXAMPLES:**

We can use this to compute the splitting field of a polynomial. (Unfortunately this takes an unreasonably long time for non-toy examples.):

```sage
sage: x = polygen(QQ)
sage: p = x^3 + x^2 + x + 17
sage: rts = p.roots(ring=QQbar, multiplicities=False)
sage: splitting = number_field_elements_from_algebraics(rts, name='b')[0];
˓→splitting
Number Field in b with defining polynomial y^6 - 40*y^4 - 22*y^3 + 873*y^2 +
˓→1386*y + 594
sage: p.roots(ring=splitting)
[(361/29286*b^5 - 19/3254*b^4 - 14359/29286*b^3 + 401/29286*b^2 + 18183/1627*b +
˓→15930/1627, 1),
(49/117144*b^5 - 179/39048*b^4 - 3247/117144*b^3 + 22553/117144*b^2 + 1744/
˓→4881*b - 17195/6508, 1),
(-1493/117144*b^5 + 407/39048*b^4 + 60683/117144*b^3 - 24157/117144*b^2 - 56293/
˓→4881*b - 53033/6508, 1)]
```

```sage
sage: # needs sage.symbolic
sage: rt2 = AA(sqrt(2)); rt2
1.414213562373095?
sage: rt3 = AA(sqrt(3)); rt3
1.732050807568878?
sage: rt3a = QQbar(sqrt(3)); rt3a
1.732050807568878?
sage: qqI = QQbar.zeta(4); qqI
I
sage: z3 = QQbar.zeta(3); z3
-0.500000000000000? + 0.866025403784439?*I
sage: rt2b = rt3 + rt2 - rt3; rt2b
1.414213562373095?
sage: rt2c = z3 + rt2 - z3; rt2c
1.414213562373095? + 0.?e-19*I
sage: number_field_elements_from_algebraics(rt2)
(Number Field in a with defining polynomial y^2 - 2, a,
 Ring morphism:
(continues on next page)
```

5.1. Algebraic numbers 705
From: Number Field in a with defining polynomial $y^2 - 2$
To: Algebraic Real Field
Defn: a $\mapsto$ 1.414213562373095?

\texttt{sage: number_field_elements_from_algebraics((rt2,rt3))}
(\text{Number Field in a with defining polynomial $y^4 - 4*y^2 + 1$, \[-a^3 + 3*a, a^2 - 2\]},
Ring morphism:
From: Number Field in a with defining polynomial $y^4 - 4*y^2 + 1$
To: Algebraic Real Field
Defn: a $\mapsto$ -1.931851652578137?)

\begin{verbatim}
>>> from sage.all import *
>>> x = polygen(QQ)
>>> p = x**Integer(3) + x**Integer(2) + x + Integer(17)
>>> rts = p.roots(ring=QQbar, multiplicities=False)
>>> splitting = number_field_elements_from_algebraics(rts, name='b')[Integer(0)];
>>> splitting
Number Field in b with defining polynomial $y^6 - 40*y^4 - 22*y^3 + 873*y^2 + 1386*y + 594$
>>> p.roots(ring=splitting)
[(361/29286*b^5 - 19/3254*b^4 - 14359/29286*b^3 + 401/29286*b^2 + 18183/1627*b +
 15930/1627, 1),
 (49/117144*b^5 - 179/39048*b^4 - 3247/117144*b^3 + 22553/117144*b^2 + 1744/
 4881*b - 17195/6508, 1),
(-1493/117144*b^5 + 407/39048*b^4 + 60683/117144*b^3 - 24157/117144*b^2 - 56293/
 4881*b - 53033/6508, 1)]

>>> # needs sage.symbolic
>>> rt2 = AA(sqrt(Integer(2))); rt2
1.414213562373095?
>>> rt3 = AA(sqrt(Integer(3))); rt3
1.732050807568878?
>>> rt3a = QQbar(sqrt(Integer(3))); rt3a
1.732050807568878?
>>> qqI = QQbar.zeta(Integer(4)); qqI
I
>>> x3 = QQbar.zeta(Integer(3)); z3
-0.5000000000000000? + 0.866025403784439?*I
>>> rt2b = rt3 + rt2 - rt3; rt2b
1.414213562373095?
>>> rt2c = z3 + rt2 - z3; rt2c
1.414213562373095? + 0.2e-19*I

>>> number_field_elements_from_algebraics(rt2)
(\text{Number Field in a with defining polynomial $y^2 - 2$, a,}
Ring morphism:
From: Number Field in a with defining polynomial $y^2 - 2$
To: Algebraic Real Field
Defn: a $\mapsto$ 1.414213562373095?)

>>> number_field_elements_from_algebraics((rt2,rt3))
(\text{Number Field in a with defining polynomial $y^4 - 4*y^2 + 1$, \[-a^3 + 3*a, a^2 - 2\]},
Ring morphism:
From: Number Field in a with defining polynomial $y^4 - 4*y^2 + 1$
To: Algebraic Real Field
Defn: a $\mapsto$ -1.931851652578137?)
\end{verbatim}

rt3a is a real number in QQbar. Ordinarily, we'd get a homomorphism to AA (because all elements are real), but
if we specify same_field=True, we’ll get a homomorphism back to QQbar:

```python
sage: number_field_elements_from_algebraics(rt3a)  # ...

needs sage.symbolic
(Number Field in a with defining polynomial y^2 - 3, a,
 Ring morphism:
   From: Number Field in a with defining polynomial y^2 - 3
   To:   Algebraic Real Field
   Defn: a |--> 1.732050807568878?)

sage: number_field_elements_from_algebraics(rt3a, same_field=True)  # ...

needs sage.symbolic
(Number Field in a with defining polynomial y^2 - 3, a,
 Ring morphism:
   From: Number Field in a with defining polynomial y^2 - 3
   To:   Algebraic Field
   Defn: a |--> 1.732050807568878?)

We’ve created rt2b in such a way that sage does not initially know that it’s in a degree-2 extension of Q:

```python
>>> from sage.all import *

>>> number_field_elements_from_algebraics(rt3a)  # ...

needs sage.symbolic
(Number Field in a with defining polynomial y^2 - 3, a,
 Ring morphism:
   From: Number Field in a with defining polynomial y^2 - 3
   To:   Algebraic Real Field
   Defn: a |--> 1.732050807568878?)

We can specify minimal=True if we want the smallest number field:

```python
sage: number_field_elements_from_algebraics(rt2b, minimal=True)  # ...

(continues on next page)
Things work fine with rational numbers, too:

```python
sage: number_field_elements_from_algebraics((QQbar(1/2), AA(17)))
(Rational Field, [1/2, 17],
 Ring morphism:
   From: Rational Field
   To:   Algebraic Real Field
   Defn: 1 |--> 1)
```

Or we can just pass in symbolic expressions, as long as they can be coerced into QQbar:

```python
sage: number_field_elements_from_algebraics((sqrt(7), sqrt(9), sqrt(11)))
(Algebraic Field in a with defining polynomial y^4 - 9*y^2 + 1,
 [-a^3 + 8*a, 3, -a^3 + 10*a],
 Ring morphism:
   From: Algebraic Field in a with defining polynomial y^4 - 9*y^2 + 1
   To:   Algebraic Real Field
   Defn: a |--> 0.3354367396454047?)
```

Here we see an example of doing some computations with number field elements, and then mapping them back into QQbar:
sage: # needs sage.symbolic
sage: algebraics = (rt2, rt3, qqI, z3)
sage: fld, nums, hom = number_field_elements_from_algebraics(algebraics)
sage: fld, nums, hom  # random
(\text{Number Field in } a \text{ with defining polynomial } y^8 - y^4 + 1, \\
[-a^5 + a^3 + a, a^6 - 2a^2, a^6, -a^4], \\
\text{Ring morphism:} \\
\quad \text{From: Number Field in } a \text{ with defining polynomial } y^8 - y^4 + 1 \\
\quad \text{To: Algebraic Field} \\
\quad \text{Defn: } a \mapsto -0.2588190451025208\? - 0.9659258262890683\?*I)
sage: (nfrt2, nfrt3, nfI, nfz3) = nums
sage: hom(nfrt2)
1.414213562373095\? + 0.\?e-18*I
sage: nfrt2**Integer(2)
2
sage: nfrt3**Integer(2)
3
sage: nfz3 + nfz3**Integer(2)
-1
sage: nfI**Integer(2)
-1
sage: sum = nfrt2 + nfrt3 + nfI + nfz3; sum
a^5 + a^4 - a^3 + 2*a^2 - a - 1
sage: hom(sum)
2.646264369941973\? + 1.866025403784439\?*I
sage: hom(sum) == rt2 + rt3 + qqI + z3
True
sage: [hom(n) for n in nums] == [rt2, rt3, qqI, z3]  # (continues on next page)
True
>>> [hom(n) for n in nums] == [rt2, rt3, qqI, z3]
True

It is also possible to have an embedded Number Field:

```python
sage: x = polygen(ZZ)
sage: my_num = AA.polynomial_root(x^3 - 2, RIF(0,3))
sage: res = number_field_elements_from_algebraics(my_num, embedded=True)
sage: res[0].gen_embedding()
1.259921049894873?
```

```python
sage: res[2]
Ring morphism:
  From: Number Field in a with defining polynomial y^3 - 2 with a = 1.
  ---> 259921049894873?
  To:   Algebraic Real Field
  Defn: a |--> 1.259921049894873?
```

```python
>>> from sage.all import *
```
with $a = 0.6866813218928813$?

```python
>>> nums
[a^10 - 5*a^5 + 2, -a^8 + 4*a^3]
```

```python
>>> hom
Ring morphism:
  From: Number Field in a with defining polynomial $y^15 - 9*y^10 + 21*y^5 - 3$
  with $a = 0.6866813218928813$?
  To:  Algebraic Real Field
  Defn: a |--> 0.6866813218928813?
```

Complex embeddings are possible as well:

```python
sage: # needs sage.symbolic
sage: elems = [sqrt(5), 2^(1/3)+sqrt(3)*I, 3/4]
```

```python
sage: nf, nums, hom = number_field_elements_from_algebraics(elems, ....: embedded=True)
```

```python
sage: nf # random (polynomial and root not unique)
Number Field in a with defining polynomial $y^{24} - 6*y^{23} ...- 9*y^2 + 1$
  with $a = 0.2598679? + 0.0572892?*I$
```

```python
sage: list(map(QQbar, nums)) == elems == list(map(hom, nums))
True
```

```python
sage: # needs sage.symbolic
```

```python
from sage.all import *
```

```python
>>> elements = [sqrt(Integer(5)), Integer(2)**(Integer(1)/Integer(3))+sqrt(Integer(3))*I, Integer(3)/Integer(4)]
```

```python
>>> nf, nums, hom = number_field_elements_from_algebraics(elements, ....: embedded=True)
```

```python
>>> nf # random (polynomial and root not unique)
Number Field in a with defining polynomial $y^{24} - 6*y^{23} ...- 9*y^2 + 1$
  with $a = 0.2598679? + 0.0572892?*I$
```

```python
>>> nf.is_isomorphic(NumberField(...
  x**Integer(24) - Integer(9)*x**Integer(22) +
  Integer(135)*x**Integer(20) - Integer(720)*x**Integer(18) +
  Integer(1821)*x**Integer(16)
  ...
  - Integer(3015)*x**Integer(14) +
  Integer(3974)*x**Integer(12) - Integer(3015)*x**Integer(10) +
  Integer(1821)*x**Integer(8)
  ...
  - Integer(720)*x**Integer(6) +
  Integer(135)*x**Integer(4) - Integer(9)*x**Integer(2) + Integer(1), 'a'))
True
```

```python
>>> list(map(QQbar, nums)) == elements == list(map(hom, nums))
True
```

```python
sage.rings.qqbar.prec_seq()
```

Return a generator object which iterates over an infinite increasing sequence of precisions to be tried in various numerical computations.

Currently just returns powers of 2 starting at 64.

EXAMPLES:
sage: g = sage.rings.qqbar.prec_seq()
sage: [next(g), next(g), next(g)]
[64, 128, 256]

>>> from sage.all import *
>>> g = sage.rings.qqbar.prec_seq()
>>> [next(g), next(g), next(g)]
[64, 128, 256]

sage.rings.qqbar.rational_exact_root(r, d)

Check whether the rational $r$ is an exact $d$'th power.
If so, this returns the $d$'th root of $r$; otherwise, this returns None.

EXAMPLES:

sage: from sage.rings.qqbar import rational_exact_root
sage: rational_exact_root(16/81, 4)
2/3
sage: rational_exact_root(8/81, 3) is None
True

>>> from sage.all import *
>>> from sage.rings.qqbar import rational_exact_root
>>> rational_exact_root(Integer(16)/Integer(81), Integer(4))
2/3
>>> rational_exact_root(Integer(8)/Integer(81), Integer(3)) is None
True

sage.rings.qqbar.short_prec_seq()

Return a sequence of precisions to try in cases when an infinite-precision computation is possible: returns a couple of small powers of 2 and then None.

EXAMPLES:

sage: from sage.rings.qqbar import short_prec_seq
sage: short_prec_seq()
(64, 128, None)

>>> from sage.all import *
>>> from sage.rings.qqbar import short_prec_seq
>>> short_prec_seq()
(64, 128, None)

sage.rings.qqbar.t1
alias of ANBinaryExpr

sage.rings.qqbar.t2
alias of ANRoot

sage.rings.qqbar.tail_prec_seq()

A generator over precisions larger than those in short_prec_seq().

EXAMPLES:
5.2 Universal cyclotomic field

The universal cyclotomic field is the smallest subfield of the complex field containing all roots of unity. It is also the maximal abelian extension of the rational numbers.

EXAMPLES:

```sage
UCF = UniversalCyclotomicField(); UCF
Universal Cyclotomic Field
```

To generate cyclotomic elements:

```sage
UCF.gen(5)
E(5)

UCF.gen(5, 2)
E(5)^2

E = UCF.gen
```

Equality and inequality checks:

```sage
E(6, 2) == E(6)^2 == E(3)
True

E(6)^2 != E(3)
False
```

```sage
E(Integer(6), Integer(2)) == E(Integer(6))^2 == E(Integer(3))
True
```
Addition and multiplication:

```python
e(2) * e(3)  # -e(3)
e = e(2) + e(3);
e
2*e(3) + e(3)^2
```

Inverses:

```python
e^-1
1/3*e(3) + 2/3*e(3)^2
e.inverse()
1/3*e(3) + 2/3*e(3)^2
e * e.inverse()
1
```

Conjugation and Galois conjugates:

```python
e.conjugate()
e + 2*e(3)^2
e.galois_conjugates()
[2*e(3) + e(3)^2, e(3) + 2*e(3)^2]
e.norm_of_galois_extension()
3
```

One can create matrices and polynomials:
The implementation simply wraps GAP Cyclotomic. As mentioned in their documentation: arithmetical operations are quite expensive, so the use of internally represented cyclotomics is not recommended for doing arithmetic over number fields, such as calculations with matrices of cyclotomics.

5.2. Universal cyclotomic field

715
Note: There used to be a native Sage version of the universal cyclotomic field written by Christian Stump (see Issue #8327). It was slower on most operations and it was decided to use a version based on GAP instead (see Issue #18152). One main difference in the design choices is that GAP stores dense vectors whereas the native ones used Python dictionaries (storing only nonzero coefficients). Most operations are faster with GAP except some operation on very sparse elements. All details can be found in Issue #18152.

REFERENCES:

• [Bre1997]

AUTHORS:

• Christian Stump (2013): initial Sage version (see Issue #8327)
• Vincent Delecroix (2015): completed rewriting using libgap (see Issue #18152)
• Sebastian Oehms (2018): deleted the method is_finite since it returned the wrong result (see Issue #25686)
• Sebastian Oehms (2019): added _factor_univariate_polynomial() (see Issue #28631)

`sage.rings.universal_cyclotomic_field.E(n, k=1)`

Return the n-th root of unity as an element of the universal cyclotomic field.

EXAMPLES:

```python
sage: E(3)
E(3)
sage: E(3) + E(5)
-E(15)^2 - 2*E(15)^8 - E(15)^11 - E(15)^13 - E(15)^14

>>> from sage.all import *
>>> E(Integer(3))
E(3)
>>> E(Integer(3)) + E(Integer(5))
-E(15)^2 - 2*E(15)^8 - E(15)^11 - E(15)^13 - E(15)^14
```

`sage.rings.universal_cyclotomic_field.UCF_sqrt_int(N, UCF)`

Return the square root of the integer N.

EXAMPLES:

```python
sage: from sage.rings.universal_cyclotomic_field import UCF_sqrt_int
sage: UCF = UniversalCyclotomicField()

sage: UCF_sqrt_int(0, UCF)
0
sage: UCF_sqrt_int(1, UCF)
1
sage: UCF_sqrt_int(-1, UCF)
E(4)
sage: UCF_sqrt_int(2, UCF)
E(8) - E(8)^3
```

(continues on next page)
class sage.rings.universal_cyclotomic_field.UCFtoQQbar(UCF)

Bases: Morphism

Conversion to QQbar.

EXAMPLES:

```python
sage: UCF = UniversalCyclotomicField()
sage: QQbar(UCF.gen(Integer(3)))
-0.500000000000000? + 0.866025403784439?*I
sage: CC(UCF.gen(Integer(7), Integer(2)) + UCF.gen(Integer(7), Integer(6)))
0.400968867902419 + 0.193096429713793*I
sage: complex(E(Integer(7)) + E(Integer(7), Integer(2)))
(0.40096886790241915+1.7567593946498534j)
sage: complex(UCF.one()/Integer(2))
(0.5+0j)
```

class sage.rings.universal_cyclotomic_field.UniversalCyclotomicField(names=None)

Bases: UniqueRepresentation, UniversalCyclotomicField

The universal cyclotomic field.

The universal cyclotomic field is the infinite algebraic extension of \( \mathbb{Q} \) generated by the roots of unity. It is also the maximal Abelian extension of \( \mathbb{Q} \) in the sense that any Abelian Galois extension of \( \mathbb{Q} \) is also a subfield of the universal cyclotomic field.

**Element**

alias of UniversalCyclotomicFieldElement

**algebraic_closure()**

The algebraic closure.

**EXAMPLES:**

5.2. Universal cyclotomic field
sage: UniversalCyclotomicField().algebraic_closure()
Algebraic Field

>>> from sage.all import *
>>> UniversalCyclotomicField().algebraic_closure()
Algebraic Field

an_element()
Return an element.

EXAMPLES:

sage: UniversalCyclotomicField().an_element()
E(5) - 3*E(5)^2

>>> from sage.all import *
>>> UniversalCyclotomicField().an_element()
E(5) - 3*E(5)^2

complete()
Return the characteristic.

EXAMPLES:

sage: UniversalCyclotomicField().characteristic()
0

sage: parent(_)
Integer Ring

>>> from sage.all import *
>>> UniversalCyclotomicField().characteristic()
0

>>> parent(_)
Integer Ring

degree()
Return the degree of self as a field extension over the Rationals.

EXAMPLES:

sage: UCF = UniversalCyclotomicField()
sage: UCF.degree()
+Infinity

>>> from sage.all import *
>>> UCF = UniversalCyclotomicField()
>>> UCF.degree()
+Infinity

gen(n, k=1)
Return the standard primitive n-th root of unity.

If k is not None, return the k-th power of it.

EXAMPLES:
sage: UCF = UniversalCyclotomicField()
sage: UCF.gen(15)
E(15)
sage: UCF.gen(7,3)
E(7)^3
sage: UCF.gen(4,2)
-1

>>> from sage.all import *
>>> UCF = UniversalCyclotomicField()
>>> UCF.gen(Integer(15))
E(15)
>>> UCF.gen(Integer(7),Integer(3))
E(7)^3
>>> UCF.gen(Integer(4),Integer(2))
-1

There is an alias zeta also available:

sage: UCF.zeta(6)
-E(3)^2

>>> from sage.all import *
>>> UCF.zeta(Integer(6))
-E(3)^2

is_exact()

Return True as this is an exact ring (i.e. not numerical).

EXAMPLES:

sage: UniversalCyclotomicField().is_exact()
True

>>> from sage.all import *
>>> UniversalCyclotomicField().is_exact()
True

one()

Return one.

EXAMPLES:

sage: UCF = UniversalCyclotomicField()
sage: UCF.one()
1
sage: parent(_)
Universal Cyclotomic Field

>>> from sage.all import *
>>> UCF = UniversalCyclotomicField()
>>> UCF.one()
1
>>> parent(_)
Universal Cyclotomic Field
some_elements()  
Return a tuple of some elements in the universal cyclotomic field.  

EXAMPLES:  

```python  
sage: UniversalCyclotomicField().some_elements()  
(0, 1, -1, E(3), E(7) - 2/3*E(7)^2)  
sage: all(parent(x) is UniversalCyclotomicField() for x in _)  
True  
```  

```python  
>>> from sage.all import *  
>>> UniversalCyclotomicField().some_elements()  
(0, 1, -1, E(3), E(7) - 2/3*E(7)^2)  
>>> all(parent(x) is UniversalCyclotomicField() for x in _)  
True  
```  

zero()  
Return zero.  

EXAMPLES:  

```python  
sage: UCF = UniversalCyclotomicField()  
sage: UCF.zero()  
0  
sage: parent(_)
Universal Cyclotomic Field  
```  

```python  
>>> from sage.all import *  
>>> UCF = UniversalCyclotomicField()  
>>> UCF.zero()  
0  
>>> parent(_)
Universal Cyclotomic Field  
```  

zeta(n, k=1)  
Return the standard primitive \( n \)-th root of unity.  
If \( k \) is not None, return the \( k \)-th power of it.  

EXAMPLES:  

```python  
sage: UCF = UniversalCyclotomicField()  
sage: UCF.gen(15)  
E(15)  
sage: UCF.gen(7, 3)  
E(7)^3  
sage: UCF.gen(4, 2)  
-1  
```  

```python  
>>> from sage.all import *  
>>> UCF = UniversalCyclotomicField()  
>>> UCF.gen(Integer(15))  
E(15)  
>>> UCF.gen(Integer(7), Integer(3))  
E(7)^3  
>>> UCF.gen(Integer(4), Integer(2))  
-1  
```
There is an alias `zeta` also available:

```python
sage: UCF.zeta(6)
-E(3)^2
```

```python
>>> from sage.all import *

UCF.zeta(Integer(6))
-E(3)^2
```

class `sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement`

**Bases:** `FieldElement`

**INPUT:**

- `parent` – a universal cyclotomic field
- `obj` – a libgap element (either an integer, a rational or a cyclotomic)

**abs()**

Return the absolute value (or complex modulus) of `self`. The absolute value is returned as an algebraic real number.

**EXAMPLES:**

```python
sage: f = 5/2*E(3)+E(5)/7
sage: f.abs()
2.597760303873084?

sage: abs(f)
2.597760303873084?

sage: a = E(8)
sage: abs(a)
1

sage: v, w = vector([a]), vector([a, a])
sage: v.norm(), w.norm()
(1, 1.414213562373095?)

sage: v.norm().parent()
Algebraic Real Field
```

```python
>>> from sage.all import *

>>> f = Integer(5)/Integer(2)*E(Integer(3))+E(Integer(5))/Integer(7)

>>> f.abs()
2.597760303873084?

>>> abs(f)
2.597760303873084?

>>> a = E(Integer(8))

>>> abs(a)
1

>>> v, w = vector([a]), vector([a, a])

>>> v.norm(), w.norm()
(1, 1.414213562373095?)

>>> v.norm().parent()
Algebraic Real Field
```

**additive_order()**

Return the additive order.
EXAMPLES:

```python
sage: UCF = UniversalCyclotomicField()
sage: UCF.zero().additive_order()
0
sage: UCF.one().additive_order()
+Infinity
sage: UCF.gen(3).additive_order()
+Infinity
```

```python
>>> from sage.all import *
>>> UCF = UniversalCyclotomicField()
>>> UCF.zero().additive_order()
0
>>> UCF.one().additive_order()
+Infinity
>>> UCF.gen(Integer(3)).additive_order()
+Infinity
```

**conductor()**

Return the conductor of `self`.

EXAMPLES:

```python
sage: E(3).conductor()
3
sage: (E(5) + E(3)).conductor()
15
```

```python
>>> from sage.all import *
>>> E(Integer(3)).conductor()
3
>>> (E(Integer(5)) + E(Integer(3))).conductor()
15
```

**conjugate()**

Return the complex conjugate.

EXAMPLES:

```python
sage: (E(7) + 3*E(7,2) - 5 * E(7,3)).conjugate()
-5*E(7)^4 + 3*E(7)^5 + E(7)^6
```

```python
>>> from sage.all import *
>>> (E(Integer(7)) + Integer(3)*E(Integer(7),Integer(2)) - Integer(5) * Integer(7,3)).conjugate()
-5*E(7)^4 + 3*E(7)^5 + E(7)^6
```

**denominator()**

Return the denominator of this element.

See also:

`is_integral()`

EXAMPLES:
\begin{verbatim}
sage: a = E(5) + 1/2*E(5,2) + 1/3*E(5,3)
sage: a
E(5) + 1/2*E(5)^2 + 1/3*E(5)^3
sage: a.denominator()
6
sage: parent(_)
Integer Ring

>>> from sage.all import *

>>> a = E(Integer(5)) + Integer(1)/Integer(2)*E(Integer(5),Integer(2)) + Integer(1)/Integer(3)*E(Integer(5),Integer(3))

>>> a
E(5) + 1/2*E(5)^2 + 1/3*E(5)^3

>>> a.denominator()
6

>>> parent(_)
Integer Ring
\end{verbatim}

galois_conjugates \((n=None)\)

Return the Galois conjugates of \(\text{self} \).

INPUT:

- \(n\) – an optional integer. If provided, return the orbit of the Galois group of the \(n\)-th cyclotomic field over \(\mathbb{Q}\). Note that \(n\) must be such that this element belongs to the \(n\)-th cyclotomic field (in other words, it must be a multiple of the conductor).

EXAMPLES:

\begin{verbatim}
sage: E(6).galois_conjugates()
[-E(3)^2, -E(3)]
sage: E(6).galois_conjugates()
[-E(3)^2, -E(3)]
sage: (E(9,2) - E(9,4)).galois_conjugates()
[E(9)^2 - E(9)^4,
 E(9)^2 + E(9)^4 + E(9)^5,
 -E(9)^2 - E(9)^5 - E(9)^7,
 -E(9)^2 - E(9)^4 - E(9)^7,
 E(9)^4 + E(9)^5 + E(9)^7,
 -E(9)^5 + E(9)^7]
sage: zeta = E(5)
sage: zeta.galois_conjugates(5)
sage: zeta.galois_conjugates(10)
sage: zeta.galois_conjugates(15)
sage: zeta.galois_conjugates(17)
Traceback (most recent call last):
...
ValueError: n = 17 must be a multiple of the conductor (5)
\end{verbatim}

5.2. Universal cyclotomic field
>>> from sage.all import *
>>> E(Integer(6)).galois_conjugates()
[-E(3)^2, -E(3)]

>>> E(Integer(6)).galois_conjugates()
[-E(3)^2, -E(3)]

>>> (E(Integer(9), Integer(2)) - E(Integer(9), Integer(4))).galois_conjugates()
[E(9)^2 - E(9)^4,
 E(9)^2 + E(9)^4 + E(9)^5,
 -E(9)^2 - E(9)^5 - E(9)^7,
 -E(9)^2 - E(9)^4 - E(9)^7,
 E(9)^4 + E(9)^5 + E(9)^7,
 -E(9)^5 + E(9)^7]

>>> zeta = E(Integer(5))
>>> zeta.galois_conjugates(Integer(5))

>>> zeta.galois_conjugates(Integer(10))

>>> zeta.galois_conjugates(Integer(15))

>>> zeta.galois_conjugates(Integer(17))
Traceback (most recent call last):
  ... ValueError: n = 17 must be a multiple of the conductor (5)

imag()

Return the imaginary part of this element.

EXAMPLES:

sage: E(3).imag()
-1/2*E(12)^7 + 1/2*E(12)^11

sage: E(5).imag()
1/2*E(20) - 1/2*E(20)^9

sage: a = E(5) - 2*E(3)

sage: AA(a.imag()) == QQbar(a).imag()
True

imag_part()

Return the imaginary part of this element.

EXAMPLES:
sage: E(3).imag()
-1/2*E(12)^7 + 1/2*E(12)^11
sage: E(5).imag()
1/2*E(20) - 1/2*E(20)^9
sage: a = E(5) - 2*E(3)
sage: AA(a.imag()) == QQbar(a).imag()
True

>>> from sage.all import *
>>> E(Integer(3)).imag()
-1/2*E(12)^7 + 1/2*E(12)^11
>>> E(Integer(5)).imag()
1/2*E(20) - 1/2*E(20)^9
>>> a = E(Integer(5)) - Integer(2)*E(Integer(3))
>>> AA(a.imag()) == QQbar(a).imag()
True

inverse()

is_integral()

Return whether self is an algebraic integer.

This just wraps IsIntegralCyclotomic from GAP.

See also:

denominator()

EXAMPLES:

sage: E(6).is_integral()
True
sage: (E(4)/2).is_integral()
False

>>> from sage.all import *
>>> E(Integer(6)).is_integral()
True
>>> (E(Integer(4))/Integer(2)).is_integral()
False

is_rational()

Test whether this element is a rational number.

EXAMPLES:

sage: E(3).is_rational()
False
sage: (E(3) + E(3,2)).is_rational()
True

>>> from sage.all import *
>>> E(Integer(3)).is_rational()
False
>>> (E(Integer(3)) + E(Integer(3),Integer(2))).is_rational()
True

5.2. Universal cyclotomic field
**is_real()**

Test whether this element is real.

**EXAMPLES:**

```python
sage: E(3).is_real()
False
sage: (E(3) + E(3,2)).is_real()
True
sage: a = E(3) - 2*E(7)
sage: a.real_part().is_real()
True
sage: a.imag_part().is_real()
True
```

```python
>>> from sage.all import *
>>> E(Integer(3)).is_real()
False
>>> (E(Integer(3)) + E(Integer(3),Integer(2))).is_real()
True
>>> a = E(Integer(3)) - Integer(2)*E(Integer(7))
>>> a.real_part().is_real()
True
>>> a.imag_part().is_real()
True
```

**is_square()**

**EXAMPLES:**

```python
sage: UCF = UniversalCyclotomicField()
sage: UCF(5/2).is_square()
True
sage: UCF.zeta(7,3).is_square()
True
sage: (2 + UCF.zeta(3)).is_square()
Traceback (most recent call last):
... Not Implemented Error: is_square() not fully implemented for elements of Universal Cyclotomic Field
```

```python
>>> from sage.all import *
>>> UCF = UniversalCyclotomicField()
>>> UCF(Integer(5)/Integer(2)).is_square()
True
>>> UCF.zeta(Integer(7),Integer(3)).is_square()
True
>>> (Integer(2) + UCF.zeta(Integer(3))).is_square()
Traceback (most recent call last):
... Not Implemented Error: is_square() not fully implemented for elements of Universal Cyclotomic Field
```
minpoly\((var='x')\)

The minimal polynomial of self element over $\mathbb{Q}$.

**INPUT:**

- $var$ – (default: 'x') the name of the variable to use.

**EXAMPLES:**

```sage
sage: UCF.<E> = UniversalCyclotomicField()

sage: UCF(4).minpoly()
x - 4

sage: UCF(4).minpoly(var='y')
y - 4

sage: E(3).minpoly()
x^2 + x + 1

sage: E(3).minpoly(var='y')
y^2 + y + 1
```

**Todo:** Polynomials with libgap currently does not implement a .sage() method (see Issue #18266). It would be faster/safer to not use string to construct the polynomial.

**multiplicative_order()**

Return the multiplicative order.

**EXAMPLES:**

```sage
sage: E(5).multiplicative_order()
5

sage: (E(5) + E(12)).multiplicative_order()
+Infinity

sage: UniversalCyclotomicField().zero().multiplicative_order()
Traceback (most recent call last):
...
GAPError: Error, argument must be nonzero
```
norm_of_galois_extension()

Return the norm as a Galois extension of \( \mathbb{Q} \), which is given by the product of all galois_conjugates.

EXAMPLES:

```python
sage: E(3).norm_of_galois_extension()
1
sage: E(6).norm_of_galois_extension()
1
sage: (E(2) + E(3)).norm_of_galois_extension()
3
sage: parent(_)
Integer Ring
```

real()

Return the real part of this element.

EXAMPLES:

```python
sage: E(3).real()
-1/2
sage: E(5).real()
1/2*E(5) + 1/2*E(5)^4
sage: a = E(5) - 2*E(3)
sage: AA(a.real()) == QQbar(a).real()
True
```
**real_part()**

Return the real part of this element.

**EXAMPLES:**

```python
sage: E(3).real()
-1/2
sage: E(5).real()
1/2*E(5) + 1/2*E(5)^4
sage: a = E(5) - 2*E(3)
sage: AA(a.real()) == QQbar(a).real()
True
```

```python
>>> from sage.all import *
>>> E(Integer(3)).real()
-1/2
>>> E(Integer(5)).real()
1/2*E(5) + 1/2*E(5)^4
>>> a = E(Integer(5)) - Integer(2)*E(Integer(3))
>>> AA(a.real()) == QQbar(a).real()
True
```

**sqrt(extend=True, all=False)**

Return a square root of self.

With default options, the output is an element of the universal cyclotomic field when this element is expressed via a single root of unity (including rational numbers). Otherwise, return an algebraic number.

**INPUT:**

- `extend` – bool (default: True); if True, might return a square root in the algebraic closure of the rationals. If false, return a square root in the universal cyclotomic field or raises an error.
- `all` – bool (default: False); if True, return a list of all square roots.

**EXAMPLES:**

```python
sage: UCF = UniversalCyclotomicField()
sage: UCF(3).sqrt()
E(12)^7 - E(12)^11
sage: (UCF(3).sqrt())**2
3
sage: r = UCF(-1400/143).sqrt()
sage: r**2
-1400/143
sage: E(33).sqrt()
-E(33)^17
sage: E(33).sqrt() ** 2
E(33)
sage: (3 * E(5)).sqrt()
-E(60)^11 + E(60)^31
sage: (3 * E(5)).sqrt() ** 2
3*E(5)
```

5.2. Universal cyclotomic field
>>> from sage.all import *
>>> UCF = UniversalCyclotomicField()
>>> UCF(Integer(3)).sqrt()
E(12)^7 - E(12)^11
>>> (UCF(Integer(3)).sqrt())**Integer(2)
3

>>> r = UCF(-Integer(1400) / Integer(143)).sqrt()
>>> r**Integer(2)
-1400/143

>>> E(Integer(33)).sqrt()
-E(33)^17
>>> E(Integer(33)).sqrt() ** Integer(2)
E(33)

>>> (Integer(3) * E(Integer(5))).sqrt()
-E(60)^11 + E(60)^31
>>> (Integer(3) * E(Integer(5))).sqrt() ** Integer(2)
3*E(5)

Setting all=True you obtain the two square roots in a list:

```
sage: UCF(3).sqrt(all=True)
[E(12)^7 - E(12)^11, -E(12)^7 + E(12)^11]
sage: (1 + UCF.zeta(Integer(5))).sqrt(all=True)
[1.209762576525833? + 0.3930756888787117?*I,
 -1.209762576525833? - 0.3930756888787117?*I]
```

In the following situation, Sage is not (yet) able to compute a square root within the universal cyclotomic field:

```
sage: (E(5) + E(5, 2)).sqrt()
0.7476743906106103? + 1.029085513635746?*I
sage: (E(5) + E(5, 2)).sqrt(extend=False)
Traceback (most recent call last):
  ... 
NotImplementedError: sqrt() not fully implemented for elements of Universal\n→Cyclotomic Field
```

```
>> from sage.all import *
>> (E(Integer(5)) + E(Integer(5), Integer(2))).sqrt()
0.7476743906106103? + 1.029085513635746?*I
>> (E(Integer(5)) + E(Integer(5), Integer(2))).sqrt(extend=False)
Traceback (most recent call last):
  ... 
NotImplementedError: sqrt() not fully implemented for elements of Universal\n→Cyclotomic Field
```

```
\textbf{to\_\text{cyclotomic\_field}}(R=None)

\text{Return this element as an element of a cyclotomic field.}
```
EXAMPIES:

```python
sage: UCF = UniversalCyclotomicField()
sage: UCF.gen(3).to_cyclotomic_field()
zeta3
sage: UCF.gen(3,2).to_cyclotomic_field()
-zeta3 - 1
sage: CF = CyclotomicField(5)
sage: CF(E(5)) # indirect doctest
zeta5
sage: CF = CyclotomicField(7)
sage: CF(E(5)) # indirect doctest
Traceback (most recent call last):
  ...
TypeError: cannot coerce zeta5 into Cyclotomic Field of order 7 and
degree 6
sage: CF = CyclotomicField(10)
sage: CF(E(5)) # indirect doctest
zeta10^2
```

>>> from sage.all import *
>>> UCF = UniversalCyclotomicField()

```python
>>> UCF.gen(Integer(3)).to_cyclotomic_field()
zeta3
>>> UCF.gen(Integer(3),Integer(2)).to_cyclotomic_field()
-zeta3 - 1
```

```python
>>> CF = CyclotomicField(Integer(5))
>>> CF(E(Integer(5))) # indirect doctest
zeta5
```

```python
>>> CF = CyclotomicField(Integer(7))
>>> CF(E(Integer(5))) # indirect doctest
Traceback (most recent call last):
  ...
TypeError: cannot coerce zeta5 into Cyclotomic Field of order 7 and
degree 6
```

```python
>>> CF = CyclotomicField(Integer(10))
>>> CF(E(Integer(5))) # indirect doctest
zeta10^2
```

Matrices are correctly dealt with:

```python
[ E(3) E(4)]
[ E(5) -E(3)^2]
sage: Matrix(CyclotomicField(60),M) # indirect doctest
[zeta60^10 - 1  zeta60^15]
[ zeta60^12  zeta60^10]
```

5.2. Universal cyclotomic field
>>> from sage.all import *
>>> M = Matrix(UCF,Integer(2),[E(Integer(3)),E(Integer(4)),E(Integer(5)),
˓→E(Integer(6))]); M
[ E(3) E(4)]
[ E(5) -E(3)^2]

>>> Matrix(CyclotomicField(Integer(60)),M)  # indirect doctest
[zeta60^10 - 1 zeta60^15]
[ zeta60^12 zeta60^10]

Using a non-standard embedding:

sage: # needs sage.symbolic
sage: CF = CyclotomicField(5, embedding=CC(exp(4*pi*i/5)))
sage: x = E(5)
sage: CC(x)
0.309016994374947 + 0.951056516295154*I
sage: CC(CF(x))
0.309016994374947 + 0.951056516295154*I

>>> from sage.all import *
>>> # needs sage.symbolic
>>> CF = CyclotomicField(Integer(5), embedding=CC(exp(Integer(4)*pi*i/
˓→Integer(5))))
>>> x = E(Integer(5))
>>> CC(x)
0.309016994374947 + 0.951056516295154*I
>>> CC(CF(x))
0.309016994374947 + 0.951056516295154*I

Test that the bug reported in Issue #19912 has been fixed:

sage: a = 1+E(4); a
1 + E(4)
sage: a.to_cyclotomic_field()
zeta4 + 1

>>> from sage.all import *
>>> a = Integer(1)+E(Integer(4)); a
1 + E(4)
>>> a.to_cyclotomic_field()
zeta4 + 1

sage.rings.universal_cyclotomic_field.late_import()  
This function avoids importing libgap on startup. It is called once through the constructor of UniversalCyclotomicField.

EXAMPLES:

sage: import sage.rings.universal_cyclotomic_field as ucf
sage: _ = UniversalCyclotomicField()  # indirect doctest
sage: ucf.libgap is None  # indirect doctest
False
5.2. Universal cyclotomic field

>>> _ = UniversalCyclotomicField()  # indirect doctest
>>> ucf.libgap is None  # indirect doctest
False
6.1 Enumeration of primitive totally real fields

This module contains functions for enumerating all primitive totally real number fields of given degree and small discriminant. Here a number field is called \textit{primitive} if it contains no proper subfields except \( \mathbb{Q} \).

See also \textit{sage.rings.number_field.totallyreal_rel}, which handles the non-primitive case using relative extensions.

\textbf{ALGORITHM:}

We use Hunter’s algorithm ([Coh2000], Section 9.3) with modifications due to Takeuchi [Tak1999] and the author [Voi2008].

We enumerate polynomials \( f(x) = x^n + a_{n-1}x^{n-1} + \ldots + a_0 \). Hunter’s theorem gives bounds on \( a_{n-1} \) and \( a_{n-2} \); then given \( a_{n-1} \) and \( a_{n-2} \), one can recursively compute bounds on \( a_{n-3}, \ldots, a_0 \), using the fact that the polynomial is totally real by looking at the zeros of successive derivatives and applying Rolle’s theorem. See [Tak1999] for more details.

\textbf{EXAMPLES:}

In this first simple example, we compute the totally real quadratic fields of discriminant \( \leq 50 \).

\begin{verbatim}
 sage: enumerate_totallyreal_fields_prim(2,50)
 [[5, x^2 - x - 1],
  [8, x^2 - 2],
  [12, x^2 - 3],
  [13, x^2 - x - 3],
  [17, x^2 - x - 4],
  [21, x^2 - x - 5],
  [24, x^2 - 6],
  [28, x^2 - 7],
  [29, x^2 - x - 7],
  [33, x^2 - x - 8],
  [37, x^2 - x - 9],
  [40, x^2 - 10],
  [41, x^2 - x - 10],
  [44, x^2 - 11]]
 sage: [d for d in range(5,50) if (is_squarefree(d) and d%4 == 1) or (d%4 == 0 and is_squarefree(d/4))]
 [5, 8, 12, 13, 17, 20, 21, 24, 28, 29, 33, 37, 40, 41, 44]

 >>> from sage.all import *
 >>> enumerate_totallyreal_fields_prim(Integer(2),Integer(50))
 [[5, x^2 - x - 1],
  [8, x^2 - 2],
  [12, x^2 - 3],
  [13, x^2 - x - 3],
  [17, x^2 - x - 4],
  [21, x^2 - x - 5],
  [24, x^2 - 6],
  [28, x^2 - 7],
  [29, x^2 - x - 7],
  [33, x^2 - x - 8],
  [37, x^2 - x - 9],
  [40, x^2 - 10],
  [41, x^2 - x - 10],
  [44, x^2 - 11]]

 (continues on next page)
\end{verbatim}
Next, we compute all totally real quintic fields of discriminant \( \leq 10^5 \):

```python
sage: ls = enumerate_totallyreal_fields_prim(5,10^5) ; ls
[[14641, x^5 - x^4 - 4*x^3 + 3*x^2 + 3*x - 1],
 [24217, x^5 - 5*x^3 - x^2 + 3*x + 1],
 [36497, x^5 - 2*x^4 - 3*x^3 + 5*x^2 + x - 1],
 [38569, x^5 - 5*x^3 + 4*x - 1],
 [65657, x^5 - x^4 - 5*x^3 + 2*x^2 + 5*x + 1],
 [70601, x^5 - x^4 - 5*x^3 + 2*x^2 + 3*x - 1],
 [81509, x^5 - x^4 - 5*x^3 + 3*x^2 + 5*x - 2],
 [81589, x^5 - 6*x^3 + 8*x - 1],
 [89417, x^5 - 6*x^3 - x^2 + 8*x + 3]]
sage: len(ls)
9
```

We see that there are 9 such fields (up to isomorphism!).

See also [Mar1980].

AUTHORS:

- John Voight (2007-09-01): initial version; various optimization tweaks
- John Voight (2007-10-09): added DSage module; added pari functions to avoid recomputations; separated DSage component
- Craig Citro and John Voight (2007-11-04): additional doctests and type checking
Enumerate primitive totally real fields of degree \( n > 1 \) with discriminant \( d \leq B \); optionally one can specify the first few coefficients, where the sequence \( a \) corresponds to

\[
a[d]*x^n + \ldots + a[0]*x^{(n-d)}
\]

where \( \text{length}(a) = d+1 \), so in particular always \( a[d] = 1 \).

**Note:** This is guaranteed to give all primitives such fields, and seems in practice to give many imprimitive ones.

**INPUT:**
- \( n \) – (integer) the degree
- \( B \) – (integer) the discriminant bound
- \( a \) – (list, default: \([\]\)) the coefficient list to begin with
- \( \text{verbose} \) – (integer or string, default: 0) if \( \text{verbose} = 1 \) (or 2), then print to the screen (really) verbosely; if \( \text{verbose} \) is a string, then print verbosely to the file specified by \( \text{verbose} \).
- \( \text{return_seqs} \) – (boolean, default \( \text{False} \)) If \( \text{True} \), then return the polynomials as sequences (for easier exporting to a file).
- \( \text{phc} \) – boolean or integer (default: \( \text{False} \))
- \( \text{keep_fields} \) – (boolean or integer, default: \( \text{False} \)) If \( \text{keep_fields} \) is \( \text{True} \), then keep fields up to \( B \log(B) \); if \( \text{keep_fields} \) is an integer, then keep fields up to that integer.
- \( t_2 \) – (boolean or integer, default: \( \text{False} \)) If \( t_2 = T \), then keep only polynomials with \( t_2 \) norm \( \geq T \).
- \( \text{just_print} \) – (boolean, default: \( \text{False} \)) if \( \text{just_print} \) is not \( \text{False} \), instead of creating a sorted list of totally real number fields, we simply write each totally real field we find to the file whose filename is given by \( \text{just_print} \). In this case, we don’t return anything.
- \( \text{return_pari_objects} \) – (boolean, default: \( \text{True} \)) if both \( \text{return_seqs} \) and \( \text{return_pari_objects} \) are \( \text{False} \) then it returns the elements as Sage objects; otherwise it returns PARI objects.

**OUTPUT:**
the list of fields with entries \([d, f]\), where \( d \) is the discriminant and \( f \) is a defining polynomial, sorted by discriminant.

**AUTHORS:**
- John Voight (2007-09-03)
- Craig Citro (2008-09-19): moved to Cython for speed improvement
sage.rings.number_field.totallyreal.odlyzko_bound_totallyreal($n$)

This function returns the unconditional Odlyzko bound for the root discriminant of a totally real number field of degree $n$.

**Note:** The bounds for $n > 50$ are not necessarily optimal.

**INPUT:**
- $n$ – (integer) the degree

**OUTPUT:**
a lower bound on the root discriminant (as a real number)

**EXAMPLES:**

```python
sage: from sage.rings.number_field.totallyreal import odlyzko_bound_totallyreal
sage: [odlyzko_bound_totallyreal(n) for n in range(1, 5)]
[1.0, 2.223, 3.61, 5.067]
```

**AUTHORS:**
- John Voight (2007-09-03)

**Note:** The values are calculated by Martinet [Mar1980].

sage.rings.number_field.totallyreal.weed_fields($S, lenS=0$)

Function used internally by the `enumerate_totallyreal_fields_primal()` routine. (Weeds the fields listed by `[discriminant, polynomial]` for isomorphism classes.) Returns the size of the resulting list.

**EXAMPLES:**

```python
sage: ls = [[5,pari('x^2-3*x+1')], [5,pari('x^2-5')]]
sage: sage.rings.number_field.totallyreal.weed_fields(ls)
1
sage: ls
[[5, x^2 - 3*x + 1]]
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.totallyreal import odlyzko_bound_totallyreal
>>> [odlyzko_bound_totallyreal(n) for n in range(Integer(1), Integer(5))]
[1.0, 2.223, 3.61, 5.067]
```
6.2 Enumeration of totally real fields: relative extensions

This module contains functions to enumerate primitive extensions $L/K$, where $K$ is a given totally real number field, with given degree and small root discriminant. This is a relative analogue of the problem described in `sage.rings.number_field.totallyreal`, and we use a similar approach based on a relative version of Hunter’s theorem.

In this first simple example, we compute the totally real quadratic fields of $F = \mathbb{Q}(\sqrt{2})$ of discriminant $\leq 2000$.

```
sage: ZZx.<x> = ZZ[]
sage: F.<t> = NumberField(x^2 - 2)
sage: enumerate_totallyreal_fields_rel(F, 2, 2000)
[[1600, x^4 - 6*x^2 + 4, xF^2 + xF - 1]]
```

There is indeed only one such extension, given by $F(\sqrt{5})$.

Next, we list all totally real quadratic extensions of $\mathbb{Q}(\sqrt{5})$ with root discriminant $\leq 10$.

```
sage: F.<t> = NumberField(x^2 - 5)
sage: ls = enumerate_totallyreal_fields_rel(F, 2, 10^4)
sage: ls
# random (the second factor is platform-dependent)
[[725, x^4 - x^3 - 3*x^2 + x + 1, xF^2 + (-1/2*t - 3/2)*xF + 1],
 [1125, x^4 - x^3 - 4*x^2 + 4*x + 1, xF^2 + (-1/2*t - 7/2)*xF + 1/2*t + 3/2],
 [1600, x^4 - 6*x^2 + 4, xF^2 - 2],
 [2000, x^4 - 5*x^2 + 5, xF^2 - 1/2*t - 5/2],
 [2225, x^4 - x^3 - 5*x^2 + 2*x + 4, xF^2 + (-1/2*t + 1/2)*xF - 3/2*t - 7/2],
 [2525, x^4 - 2*x^3 - 4*x^2 + 5*x + 5, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 5/2],
 [3600, x^4 - 2*x^3 - 7*x^2 + 8*x + 1, xF^2 - 3],
 [4225, x^4 - 9*x^2 + 4, xF^2 + (-1/2*t - 1/2)*xF - 3/2*t - 9/2],
 [4400, x^4 - 7*x^2 + 11, xF^2 - 1/2*t - 7/2],
 [4525, x^4 - x^3 - 7*x^2 + 3*x + 9, xF^2 + (-1/2*t + 1/2)*xF - 3],
 [5125, x^4 - x^3 - 6*x^2 + 7*x + 11, xF^2 + (-1/2*t - 1/2)*xF - t - 4],
 [5225, x^4 - x^3 - 8*x^2 + x + 11, xF^2 + (-1/2*t + 1/2)*xF - 1/2*t - 7/2],
 [5725, x^4 - x^3 - 8*x^2 + 6*x + 11, xF^2 + (-1/2*t + 1/2)*xF - 1/2*t - 7/2],
 [6125, x^4 - x^3 - 9*x^2 + 9*x + 11, xF^2 + (-1/2*t + 1/2)*xF - t - 4],
 [7225, x^4 - 11*x^2 + 9, xF^2 + (-1)*xF - 4],
 [7600, x^4 - 9*x^2 + 19, xF^2 - 1/2*t - 9/2],
 [7625, x^4 - 11*x^2 + 19, xF^2 - 1/2*t - 9/2],
 [7650, x^4 - 11*x^2 + 3, xF^2 + (-1)*xF - 4],
 [7675, x^4 - 11*x^2 + 11, xF^2 - 1/2*t - 9/2],
 [7725, x^4 - 9*x^2 + 20, xF^2 - t - 5],
 [8255, x^4 - 2*x^3 - 8*x^2 + 9*x + 19, xF^2 + (-1)*xF - 1/2*t - 9/2],
 [8725, x^4 - x^3 - 10*x^2 + 2*x + 19, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 9/2],
 [9225, x^4 - x^3 - 10*x^2 + 7*x + 19, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 9/2]]
sage: [ f[0] for f in ls ]
[725, 1125, 1600, 2000, 2225, 2525, 3600, 4225, 4400, 4525, 5125, 5225, 5725, 6125, 7225, 7600, 7625, 7650, 8255, 8725, 9225]
sage: [NumberField(ZZx(x[1]), 't').is_galois() for x in ls] # needs sage.groups
[False, True, True, True, False, False, True, True, False, False, False, False, True, False, False, False, False, False, False]
Eight out of 21 such fields are Galois (with Galois group $C_4$ or $C_2 \times C_2$); the others have Galois closure of degree 8 (with Galois group $D_8$).

Finally, we compute the cubic extensions of $Q(\zeta_7)^+$ with discriminant $\leq 17 \times 10^9$.

```
sage: F.<t> = NumberField(ZZx([1,-4,3,1]))
sage: F.disc()
49
```

```
[16240385609L, x^9 - x^8 - 9*x^7 + 4*x^6 + 26*x^5 - 2*x^4 - 25*x^3 - x^2 + 7*x + 1, xF^3 + (-t^2 - 4*t + 1)*xF^2 + (t^2 + 3*t - 5)*xF + 3*t^2 + 11*t - 5]
```

```
(continues on next page)
```
AUTHORS:

• John Voight (2007-11-03): initial version

sage.rings.number_field.totallyreal_rel.enumerate_totallyreal_fields_all(n, B, verbose=0, return_seqs=False, return_pari_objects=True)

Enumerate all totally real fields of degree \( n \) with discriminant at most \( B \), primitive or otherwise.

INPUT:

• \( n \) – integer, the degree

• \( B \) – integer, the discriminant bound

• \( verbose \) – boolean or nonnegative integer or string (default: 0) give a verbose description of the computations being performed. If \( verbose \) is set to 2 or more, it outputs some extra information. If \( verbose \) is a string, it outputs to a file specified by \( verbose \)

• \( return\_seqs \) – (boolean, default False) If True, then return the polynomials as sequences (for easier exporting to a file). This also returns a list of four numbers, as explained in the OUTPUT section below.

• \( return\_pari\_objects \) – (boolean, default: True) if both \( return\_seqs \) and \( return\_pari\_objects \) are False then it returns the elements as Sage objects; otherwise it returns PARI objects.

EXAMPLES:

```sage
sage: enumerate_totallyreal_fields_all(4, 2000)
[[725, x^4 - x^3 - 3*x^2 + x + 1],
 [1125, x^4 - x^3 - 4*x^2 + 4*x + 1],
 [1600, x^4 - 6*x^2 + 4],
 [1957, x^4 - 4*x^2 - x + 1],
 [2000, x^4 - 5*x^2 + 5]]
sage: enumerate_totallyreal_fields_all(1, 10)
[[1, x - 1]]
```
This function enumerates (primitive) totally real field extensions of degree \( m > 1 \) of the totally real field \( F \) with discriminant \( d \leq B \); optionally one can specify the first few coefficients, where the sequence \( a \) corresponds to a polynomial by

\[
a[d] \cdot x^n + \ldots + a[0] \cdot x^{(n-d)}
\]

if \( \text{length}(a) = d+1 \), so in particular always \( a[d] = 1 \).

**Note:** This is guaranteed to give all primitive such fields, and seems in practice to give many imprimitive ones.

**INPUT:**

- \( F \) – number field, the base field
- \( m \) – integer, the degree
- \( B \) – integer, the discriminant bound
- \( a \) – list (default: \([\ ]\)), the coefficient list to begin with
- \( \text{verbose} \) – boolean or nonnegative integer or string (default: 0) give a verbose description of the computations being performed. If \( \text{verbose} \) is set to 2 or more then it outputs some extra information. If \( \text{verbose} \) is a string then it outputs to a file specified by \( \text{verbose} \)
- \( \text{return_seqs} \) – (boolean, default False) If True, then return the polynomials as sequences (for easier exporting to a file). This also returns a list of four numbers, as explained in the OUTPUT section below.
- \( \text{return_pari_objects} \) – (boolean, default True) if both \( \text{return_seqs} \) and \( \text{return_pari_objects} \) are False then it returns the elements as Sage objects; otherwise it returns PARI objects.

**OUTPUT:**

- the list of fields with entries \([d, \text{fabs}, f]\), where \( d \) is the discriminant, \( \text{fabs} \) is an absolute defining polynomial, and \( f \) is a defining polynomial relative to \( F \), sorted by discriminant.
- if \( \text{return_seqs} \) is True, then the first field of the list is a list containing the count of four items as explained below
  - the first entry gives the number of polynomials tested
  - the second entry gives the number of polynomials with its discriminant having a large enough square divisor
  - the third entry is the number of irreducible polynomials
  - the fourth entry is the number of irreducible polynomials with discriminant at most \( B \)

**EXAMPLES:**
AUTHORS:

• John Voight (2007-11-01)

sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)

Return all integral elements of the totally real field $K$ whose embeddings lie numerically within the bounds specified by the list $C$. The output is architecture dependent, and one may want to expand the bounds that define $C$ by some epsilon.

INPUT:

• $K$ – a totally real number field

• $C$ – a list [[lower, upper], ...] of lower and upper bounds, for each embedding

EXAMPLES:

sage: x = polygen(QQ)
sage: K.<alpha> = NumberField(x**Integer(2) - Integer(2), names=('alpha',)); (alpha,) = K._first_ngens(1)
```python
>>> eps = RealNumber('10e-6')
>>> C = [[Integer(0)-eps, Integer(5)+eps], [Integer(0)-eps, Integer(10)+eps]]
>>> ls = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
>>> sorted(a.trace() for a in ls)
[0, 2, 4, 4, 4, 6, 6, 6, 6, 8, 8, 8, 10, 10, 10, 10, 12, 12, 14]
>>> len(ls)
19

>>> v = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
>>> sorted(v)
[-alpha + 2, 1, -alpha + 3, 2, 3, alpha + 2, 4, alpha + 3, 5, alpha + 4,
 2*alpha + 3, alpha + 5, 2*alpha + 4, alpha + 6, 2*alpha + 5, 2*alpha + 6,
 3*alpha + 5, 2*alpha + 7]
```

A cubic field:

```python
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3 - 16*x +16)
sage: eps = 10e-6
sage: C = [[[0-eps,5+eps]]]*3
sage: v = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
```

Note that the output is platform dependent (sometimes a 5 is listed below, and sometimes it isn't):

```python
sage: sorted(v)
[-1/2*a + 2, 1/4*a^2 + 1/2*a, 0, 1, 2, 3, 4,...-1/4*a^2 - 1/2*a + 5,
 1/2*a + 3, -1/4*a^2 + 5]
```

```python
>>> from sage.all import *
```

```python
>>> sorted(v)
[-1/2*a + 2, 1/4*a^2 + 1/2*a, 0, 1, 2, 3, 4,...-1/4*a^2 - 1/2*a + 5,
 1/2*a + 3, -1/4*a^2 + 5]
```

```python
class sage.rings.number_field.totallyreal_rel.tr_data_rel(F, m, B, a=None)

Bases: object

This class encodes the data used in the enumeration of totally real fields for relative extensions.

We do not give a complete description here. For more information, see the attached functions; all of these are used internally by the functions in totallyreal_rel.py, so see that file for examples and further documentation.

```
```
INPUT:

- `f_out` – an integer sequence, to be written with the coefficients of the next polynomial
- `verbose` – boolean or nonnegative integer (default: False) print verbosely computational details. It prints extra information if `verbose` is set to 2 or more
- `haltk` – integer, the level at which to halt the inductive coefficient bounds

OUTPUT:

the successor polynomial as a coefficient list.

### 6.3 Enumeration of totally real fields: data

AUTHORS:

- John Voight (2007-09-01): Initial version
- John Voight (2007-09-19): various optimization tweaks
- Craig Citro and John Voight (2007-11-04): type checking and other polishing

sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(f)

Used solely for testing easy_is_irreducible.

EXAMPLES:

```python
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2+1'))
1
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2-1'))
0
```

```python
>>> from sage.all import *

>>> sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2+1'))
1
>>> sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2-1'))
0
```

sage.rings.number_field.totallyreal_data.hermite_constant(n)

Return the $n$-th Hermite constant.

The $n$-th Hermite constant (typically denoted $\gamma_n$), is defined to be

$$\max_L \min_{0 \neq x \in L} \|x\|^2$$

where $L$ runs over all lattices of dimension $n$ and determinant 1.

For $n \leq 8$ it returns the exact value of $\gamma_n$, and for $n > 9$ it returns an upper bound on $\gamma_n$.

INPUT:

- `n` – integer

OUTPUT:

(an upper bound for) the Hermite constant $\gamma_n$
EXAMPLES:

```python
sage: hermite_constant(1) # trivial one-dimensional lattice
1.0
sage: hermite_constant(2) # Eisenstein lattice
1.1547005383792515
sage: 2/sqrt(3.)
1.15470053837925
sage: hermite_constant(8) # E_8
2.0
```

>>> from sage.all import *
>>> hermite_constant(Integer(1)) # trivial one-dimensional lattice
1.0
>>> hermite_constant(Integer(2)) # Eisenstein lattice
1.1547005383792515
>>> Integer(2)/sqrt(RealNumber('3.'))
1.15470053837925
>>> hermite_constant(Integer(8)) # E_8
2.0

Note: The upper bounds used can be found in [CS1999] and [CE2003].

AUTHORS:

• John Voight (2007-09-03)

sage.rings.number_field.totallyreal_data.int_has_small_square_divisor(d)

Return the largest $a$ such that $a^2$ divides $d$ and $a$ has prime divisors $< 200$.

EXAMPLES:

```python
sage: from sage.rings.number_field.totallyreal_data import int_has_small_square_divisor
sage: int_has_small_square_divisor(500)
100
sage: is_prime(691)
True
sage: int_has_small_square_divisor(691)
1
sage: int_has_small_square_divisor(691^2)
1
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.totallyreal_data import int_has_small_square_divisor
>>> int_has_small_square_divisor(Integer(500))
100
>>> is_prime(Integer(691))
True
>>> int_has_small_square_divisor(Integer(691))
1
>>> int_has_small_square_divisor(Integer(691)^Integer(2))
1
```

sage.rings.number_field.totallyreal_data.lagrange_degree_3(n, an1, an2, an3)

Private function. Solves the equations which arise in the Lagrange multiplier for degree 3: for each $1 \leq r \leq n - 2$, 

Chapter 6. Enumeration of Totally Real Fields
we solve
\[ r \cdot x^i + (n - 1 - r) \cdot y^i + z^i = s_i \quad (i = 1, 2, 3) \]

where the \( s_i \) are the power sums determined by the coefficients \( a \). We output the largest value of \( z \) which occurs.

We use a precomputed elimination ideal.

EXEMPLARY:

```sage
sage: ls = sage.rings.number_field.totallyreal_data.lagrange_degree_3(3,0,1,2)
sage: [RealField(10)(x) for x in ls]
[-1.0, -1.0]
```

```sage
sage: sage.rings.number_field.totallyreal_data.lagrange_degree_3(3,6,1,2) # random
[-5.8878, -5.8878]
```

```python
>>> from sage.all import *
>>> ls = sage.rings.number_field.totallyreal_data.lagrange_degree_3(Integer(3), Integer(0), Integer(1), Integer(2))
>>> [RealField(Integer(10))(x) for x in ls]
[-1.0, -1.0]
```

```python
>>> sage.rings.number_field.totallyreal_data.lagrange_degree_3(Integer(3), Integer(6), Integer(1), Integer(2)) # random
[-5.8878, -5.8878]
```

**class** `sage.rings.number_field.totallyreal_data.tr_data`

**Bases:** `object`

This class encodes the data used in the enumeration of totally real fields.

We do not give a complete description here. For more information, see the attached functions; all of these are used internally by the functions in `totallyreal`, so see that file for examples and further documentation.

**increment** *(verbose=False, haltk=0, phc=False)*

‘Increment’ the totally real data to the next value which satisfies the bounds essentially given by Rolle’s theorem, and return the next polynomial as a sequence of integers.

The default or usual case just increments the constant coefficient; then inductively, if this is outside of the bounds we increment the next higher coefficient, and so on.

If there are no more coefficients to be had, returns the zero polynomial.

**INPUT:**

- `verbose` – boolean to print verbosely computational details
- `haltk` – integer, the level at which to halt the inductive coefficient bounds
- `phc` – boolean, if PHCPACK is available, use it when \( k = n - 5 \) to compute an improved Lagrange multiplier bound

**OUTPUT:**

The next polynomial, as a sequence of integers

**EXAMPLES:**

```sage
sage: T = sage.rings.number_field.totallyreal_data.tr_data(2,100)
sage: T.increment()
[-24, -1, 1]
sage: for i in range(19): _ = T.increment()
sage: T.increment()
```
(continues on next page)
>>> from sage.all import *
>>> T = sage.rings.number_field.totallyreal_data.tr_data(Integer(2), → Integer(100))
>>> T.increment()
[-24, -1, 1]
>>> for i in range(Integer(19)): _ = T.increment()
>>> T.increment()
[-3, -1, 1]
>>> T.increment()
[-25, 0, 1]

printa()

Print relevant data for self.

EXAMPLES:

sage: T = sage.rings.number_field.totallyreal_data.tr_data(3,2**10)
sage: T.printa()
k = 1
a = [0, 0, -1, 1]
amax = [0, 0, 0, 1]
beta = [...]  
gnk = [...]  

6.4 Enumeration of totally real fields: PHC interface

AUTHORS:

• John Voight (2007-09-19): initial version

sage.rings.number_field.totallyreal_phc.coefficients_to_power_sums(n, m, a)

Take the list a, representing a list of initial coefficients of a (monic) polynomial of degree n, and return the power sums of the roots of f up to (m – 1)-th powers.

INPUT:

• n – integer, the degree
• a – list of integers, the coefficients

OUTPUT:
list of integers.

**Note:** This uses Newton’s relations, which are classical.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.totallyreal_phc import coefficients_to_power_sums
sage: coefficients_to_power_sums(3, 2, [1, 5, 7])
[3, -7, 39]
sage: coefficients_to_power_sums(5, 4, [1, 5, 7, 9, 8])
[5, -8, 46, -317, 2158]
```

```python
>>> from sage.all import *
>>> from sage.rings.number_field.totallyreal_phc import coefficients_to_power_sums
>>> coefficients_to_power_sums(Integer(3), Integer(2), [Integer(1), Integer(5), Integer(7)])
[3, -7, 39]
>>> coefficients_to_power_sums(Integer(5), Integer(4), [Integer(1), Integer(5), Integer(7), Integer(9), Integer(8)])
[5, -8, 46, -317, 2158]
```
CHAPTER
SEVEN

INDICES AND TABLES

• Index
• Module Index
• Search Page
sage.rings.number_field.bdd_height, 362
sage.rings.number_field.class_group, 526
sage.rings.number_field.galois_group, 250
sage.rings.number_field.homset, 373
sage.rings.number_field.maps, 387
sage.rings.number_field.morphism, 371
sage.rings.number_field.number_field, 1
sage.rings.number_field.number_field_base, 191
sage.rings.number_field.number_field_element, 269
sage.rings.number_field.number_field_element_quadratic, 339
sage.rings.number_field.number_field_ideal, 447
sage.rings.number_field.number_field_ideal_rel, 498
sage.rings.number_field.number_field_morphisms, 380
sage.rings.number_field.number_field_rel, 197
sage.rings.number_field.order, 401
sage.rings.number_field.order_ideal, 517
sage.rings.number_field.S_unit_solver, 548
sage.rings.number_field.selmer_group, 593
sage.rings.number_field.small_primes_of_degree_one, 590
sage.rings.number_field.splitting_field, 240
sage.rings.number_field.structure, 397
sage.rings.number_field.totallyreal, 735
sage.rings.number_field.totallyreal_data, 745
sage.rings.number_field.totallyreal_phc, 748
sage.rings.number_field.totallyreal_rel, 739
sage.rings.number_field.unit_group, 537
sage.rings.qqbar, 601
sage.rings.universal_cyclotomic_field, 713
Non-alphabetical

_ richcmp_() (sage.rings.qqbar.AlgebraicNumber method), 656
_ richcmp_() (sage.rings.qqbar.AlgebraicReal method), 680
A
abs() (sage.rings.number_field.number_field_element.NumberFieldElement method), 270
abs() (sage.rings.qqbar.ANDescr method), 620
abs() (sage.rings.qqbar.ANExtensionElement method), 623
abs() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 633
abs_hom() (sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs method), 721
abs_non_arch() (sage.rings.number_field.number_field_element.NumberFieldElement method), 272
abs_val() (sage.rings.number_field.number_field_element.NumberFieldElement method), 18
absolute_base_field() (sage.rings.number_field.number_field_element.NumberFieldElement method), 19
absolute_charpoly() (sage.rings.number_field.number_field_element.NumberFieldElement_absolute method), 324
absolute_charpoly() (sage.rings.number_field.number_field_element.NumberFieldElement_relative method), 329
absolute_charpoly() (sage.rings.number_field.number_field_element.OrderElement_relative method), 335
absolute_degree() (sage.rings.number_field.number_field_element.NumberFieldElement method), 19
absolute_degree() (sage.rings.number_field_number_field.Generic method), 73
absolute_degree() (sage.rings.number_field_order.Order method), 406
absolute_different() (sage.rings.number_field.number_field_element.NumberFieldElement method), 273
absolute_different() (sage.rings.number_field.number_field_element.NumberFieldElement method), 201
absolute_discriminant() (sage.rings.number_field.number_field_element.NumberField_element.Absolute method), 20
absolute_discriminant() (sage.rings.number_field.number_field_element.NumberField_element.Absolute method), 20
absolute_field() (sage.rings.number_field.number_field_element.NumberField_element.Absolute method), 74
absolute_generator() (sage.rings.number_field_number_field.Generic method), 74
absolute_generator() (sage.rings.number_field_number_field_element.NumberFieldElement_absolute method), 203
absolute_ideal() (sage.rings.number_field_number_field_element.NumberField_element.Absolute method), 20
absolute_minpoly() (sage.rings.number_field_number_field_element.NumberField_element.Absolute method), 20
absolute_minpoly() (sage.rings.number_field_number_field_element.NumberField_element.Absolute method), 20
artin_symbol() (in module sage.rings.number_field.galois_group.GaloisGroup_v2 method), 259
artin_symbol() (sage.rings.number_field.number_field._ideal.NumberFieldIdeal method), 474
as_hom() (sage.rings.number_field.galois_group.GaloisGroupElement method), 251
as_number_field_element() (sage.rings.qqbar.AlgebraicNumber_base method), 666
automorphisms() (sage.rings.number_field.number_field_rel.NumberField_relative method), 206
automorphisms() (sage.rings.number_field.number_field.NumberField_absolute method), 21

B
bach_bound() (sage.rings.number_field.number_field_base.NumberField method), 191
base_field() (sage.rings.number_field.number_field_rel.NumberField_relative method), 208
base_field() (sage.rings.number_field.number_field.AbsoluteNumberField method), 23
base_ring() (sage.rings.number_field.number_field_rel.NumberField_relative method), 209
basis() (sage.rings.number_field.number_field._ideal.NumberFieldIdeal method), 475
basis() (sage.rings.number_field.order.Order method), 407
basis() (sage.rings.number_field.order.Order_absolute method), 429
basis() (sage.rings.number_field.order.Order_relative method), 436
basis_for_p_cokernel() (in module sage.rings.number_field.selmer_group), 593
basis_to_module() (in module sage.rings.number_field.number_field._ideal), 494
bdd_height() (in module sage.rings.number_field.bdd_height), 362
bdd_height_iq() (in module sage.rings.number_field.bdd_height), 364
bdd_norm_pr_gens_iq() (in module sage.rings.number_field.bdd_height), 365
bdd_norm_pr_ideal_gens() (in module sage.rings.number_field.bdd_height), 367
beta_k() (in module sage.rings.number_field.S_unit_solver), 555

C
c3_func() (in module sage.rings.number_field.S_unit_solver), 557
c4_func() (in module sage.rings.number_field.S_unit_solver), 558
c11_func() (in module sage.rings.number_field.S_unit_solver), 555
c13_func() (in module sage.rings.number_field.S_unit_solver), 556
cardinality() (sage.rings.number_field.homset.NumberFieldHomset method), 374
ceil() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 342
ceil() (sage.rings.number_field.number_field_element.NumberFieldElement method), 275
ceil() (sage.rings.qqbar.AlgebraicReal method), 681
change_generator() (sage.rings.number_field.number_field.NumberField_generic method), 75
change_names() (sage.rings.number_field.number_field_rel.NumberField_relative method), 208
change_names() (sage.rings.number_field.number_field.NumberField_absolute method), 23
change_names() (sage.rings.number_field.order.Order_absolute method), 430
characteristic() (sage.rings.number_field.number_field.NumberField_generic method), 76
characteristic() (sage.rings.qqbar.AlgebraicField_common method), 649
characteristic() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 718
charpoly() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 342
charpoly() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 357
charpoly() (sage.rings.number_field.number_field_element.NumberFieldElement method), 276
charpoly() (sage.rings.number_field.number_field_element.NumberFieldElementAbsolute method), 326
charpoly() (sage.rings.number_field.number_field_element.NumberFieldElementRelative method), 331
charpoly() (sage.rings.number_field.number_field_element.OrderElementRelative method), 336
class_group() (sage.rings.number_field.number_field.NumberFieldGeneric method), 77
class_group() (sage.rings.number_field.order.Order method), 408
class_number() (sage.rings.number_field.number_field.NumberFieldGeneric method), 79
class_number() (sage.rings.number_field.NumberField.method), 75

Index 757
degree() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 718
denominator() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 344
discriminant() (sage.rings.number_field.order.Order.absolute method), 431
divides() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 448
do_polred() (in module sage.rings.qqbar), 699
drop_vector() (in module sage.rings.number_field.number_field.S_unit_solver), 571

each_is_integral() (in module sage.rings.number_field.order), 445
easy_is_irreducible_py() (in module sage.rings.number_field.totallyreal_data), 745
easy_order() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 261

elements_of_norm() (sage.rings.number_field.number_field_rel.NumberFieldElement, 282

elements_of_bounded_height() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField, 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.number_field.number_field_element.NumberFieldElement method), 280
discriminant() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 358
discriminant() (sage.rings.universal_cyclotomic_field.NumberField method), 176
discriminant() (sage.rings.number_field.ber_field.NumberField_quadratic method), 176
disc() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 356
disc() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
disc() (sage.rings.number_field.number_field_element.NumberFieldElement method), 281
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722
discriminant() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 722

ideal() (sage.rings.number_field.class_group.FractionalIdealClass method), 531
ideal() (sage.rings.number_field_number_field.NumberField_generic method), 103
ideal() (sage.rings.number_field.order.Order method), 414
ideal_below() (sage.rings.number_field.number_field.NumberFieldIdeal_rel method), 505
ideal_class_log() (sage.rings.number_field.number_field.NumberFieldIdeal method), 481
idealchinese() (sage.rings.number_field.number_field.NumberField_generic method), 104
idealco prime() (sage.rings.number_field.number_field.NumberFieldFractionalIdeal method), 451
ideallog() (sage.rings.number_field.number_field.NumberFieldFractionalIdeal method), 452
ideals_of_bdd_norm() (sage.rings.number_field.number_field.NumberField_generic method), 105
idealstar() (sage.rings.number_field.number_field.NumberFieldFractionalIdeal method), 454
im_gens() (sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs method), 372
imag() (sage.rings.number_field_number_field_element.quadratic.NumberFieldElement_gaussian method), 340
imag() (sage.rings.number_field_number_field_element.quadratic.NumberFieldElement_quadratic method), 345
imag() (sage.rings.qqbar.AlgebraicNumber method), 659
imag() (sage.rings.qqbar.AlgebraicReal method), 682
imag() (sage.rings.qqbar.ANDescr method), 621
imag() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 724
imag_part() (sage.rings.number_field_number_field_element.quadratic.NumberFieldElement_gaussian method), 340
imag_part() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 724
incr() (sage.rings.number_field.totallyreal_rel.tr_data_rel method), 744
increment() (sage.rings.number_field.totallyreal_data.tr_data method), 747
index_in() (sage.rings.number_field.order.Order_absolute method), 431
index_in() (sage.rings.number_field.order.Order_relative method), 437
inertia_group() (sage.rings.number_field.galois.GaloisGroup_v2 method), 262
inertia_group() (sage.rings.number_field.number_field.NumberFieldIdeal method), 482
int_has_small_square_divisor() (in module sage.rings.number_field.totallyreal_data), 746
integer_points_in_polytope() (in module sage.rings.number_field.bdd_height), 368
integral_basis() (sage.rings.number_field.number_field.NumberFieldFractionalIdeal_rel method), 508
integral_basis() (sage.rings.number_field.number_field_NumberFieldIdeal method), 483
integral_basis() (sage.rings.number_field.number_field_NumberFieldIdeal method), 106
integral_closure() (sage.rings.number_field.order.Order method), 415
integral_elements_in_box() (in module sage.rings.number_field.totallyreal_rel), 743
integral_split() (sage.rings.number_field.number_field_NumberFieldIdeal method), 508
integral_split() (sage.rings.number_field.number_field_NumberFieldIdeal method), 483
intersection() (sage.rings.number_field_number_field_NumberFieldIdeal method), 484
intersection() (sage.rings.number_field.order.Order_absolute method), 432
interval() (sage.rings.qqbar.AlgebraicNumber_base method), 670
interval_diameter() (sage.rings.qqbar.AlgebraicNumber_base method), 671
interval_exact() (sage.rings.qqbar.AlgebraicNumber method), 659
interval_exact() (sage.rings.qqbar.AlgebraicReal method), 682
interval_fast() (sage.rings.qqbar.AlgebraicNumber_base method), 671
inverse() (sage.rings.number_field.class_group.FractionalIdealClass method), 531
inverse() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 725
inverse_mod() (sage.rings.number_field_number_field_element_quadratic.OrderElement_quadratic method), 358
is_abelian() (sage.rings.number_field_number_field_element.NumberFieldElement method), 292
inverse_mod() (sage.rings.number_field_number_field_element.OrderElement_absolute method), 334
invertible_residues() (sage.rings.number_field_number_field_element.OrderElement_relative method), 337
invertible_residues_mod() (sage.rings.number_field_number_field_element.RelativeCyclotomicFieldElement method), 462
is_abelian() (sage.rings.number_field_number_field_cyclotomic method), 61
is_abelian() (sage.rings.number_field_number_field_cyclotomic_field method), 110
is_absolute() (sage.rings.number_field_number_field_base.NumberField method), 193
is_absolute() (sage.rings.number_field_number_field_rel.NumberField_relative method), 218
is_absolute() (sage.rings.number_field_number_field_number_field_absolute method), 41
is absolute() (sage.rings.number_field_number_field_number_field_generic method), 110
is_ABSOLUTE() (sage.rings.number_field_number_field_quadratic method), 183
is_AlgebraicField() (sage.rings.qqbar), 701
is_AlgebraicNumber() (sage.rings.qqbar), 702
is_AlgebraicReal() (sage.rings.qqbar), 702
is_AlgebraicRealField() (sage.rings.qqbar), 703
is_CM() (sage.rings.number_field_number_field_NumberField_generic method), 107
is_CM_extension() (sage.rings.number_field_number_field_number_field_rel.NumberField_relative method), 217
is_complex() (sage.rings.qqbar.AlgebraicGenerator method), 652
is_complex() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 679
is_complex() (sage.rings.qqbar.ANBinaryExpr method), 619
is_complex() (sage.rings.qqbar.ANExtensionElement method), 628
is_complex() (sage.rings.qqbar.ANRoot method), 635
is_complex() (sage.rings.qqbar.ANRational method), 640
is_complex() (sage.rings.qqbar.ANUnderlyingExpr method), 635
is_complex() (sage.rings.qqbar.ANUnderlyingPolynomial method), 644
is_coprim() (sage.rings.number_field_number_field_ideal.NumberFieldFractionalIdeal method), 459
is_coprime() (sage.rings.number_field_order.OrderIdeal_quadratic method), 522
is_exact() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 719
is_field() (sage.rings.number_field_number_field_number_field_generic method), 111
is_field() (sage.rings.number_field_number_field_order.Order method), 416
is_free() (sage.rings.number_field_number_field_rel.NumberField_relative method), 219
is_fundamental_discriminant() (in module sage.rings.number_field_number_field), 184
is_galois() (sage.rings.number_field_galois_group.GaloisGroup_v2 method), 263
is_galois() (sage.rings.number_field_number_field_number_field_relative method), 219
is_galois() (sage.rings.number_field_number_field_number_field_relative method), 61
is_galois() (sage.rings.number_field_number_field_number_field_cyclotomic method), 179
is_galois() (sage.rings.number_field_number_field_number_field_quadratic method), 179
is_galois() (sage.rings.number_field_number_field_number_field_relative method), 219
is_galois_relative() (sage.rings.number_field_number_field_number_field_relative method), 220
is_injective() (sage.rings.number_field.maps.NumberFieldIsomorphism method), 396
is_integer() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 346
is_integer() (sage.rings.number_field.number_field_element.NumberFieldElement method), 293
is_integer() (sage.rings.qqbar.AlgebraicNumber_base method), 672
is_integral() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 347
is_integral() (sage.rings.number_field.number_field_element.NumberFieldElement method), 294
is_integral() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 509
is_integral() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 485
is_integral() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField_element method), 725
is_integra1y_closed() (sage.rings.number_field.number_field_element.Order method), 416
is_isomorphic() (sage.rings.number_field.number_field_element.NumberField_element_cyclotomic method), 61
is_isomorphic() (sage.rings.number_field.number_field_element_number_field_element_method), 112
is_isomorphic_relative() (sage.rings.number_field.number_field_rel.NumberField_relative method), 221
is_maximal() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal method), 460
is_maximal() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 485
is_maximal() (sage.rings.number_field.number_field_order.Order_absolute method), 433
is_maximal() (sage.rings.number_field.number_field_order.Order_relative method), 437
is_noetherian() (sage.rings.number_field.number_field_order.Order method), 417
is_norm() (sage.rings.number_field.number_field_element.NumberFieldElement method), 295
is_nth_power() (sage.rings.number_field.number_field_element.NumberFieldElement method), 297
is_NumberField() (in module sage.rings.number_field.number_field_base), 196
is_NumberFieldElement() (in module sage.rings.number_field.number_field_element), 338
is_NumberFieldFractionalIdeal() (in module sage.rings.number_field.number_field_ideal), 495
is_NumberFieldFractionalIdeal_rel() (in module sage.rings.number_field.number_field_ideal_rel), 516
is_NumberFieldHomsetCodomain() (in module sage.rings.number_field.number_field), 183
is_NumberFieldIdeal() (in module sage.rings.number_field.number_field_ideal), 496
is_NumberFieldOrder() (in module sage.rings.number_field.order), 445
is_one() (sage.rings.number_field.number_field_element_number_field_element_quadratic.NumberFieldElement_quadratic method), 347
is_one() (sage.rings.number_field.number_field_element.NumberFieldElement method), 298
is_padic_square() (sage.rings.number_field.number_field_element.NumberFieldElement method), 298
is_prime() (sage.rings.number_field.number_field_element.NumberFieldElement method), 299
is_prime() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 509
is_prime() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 485
is_principal() (sage.rings.number_field.number_field_class_group.FractionalIdeal_class method), 532
is_principal() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 510
is_principal() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 486
is_principal() (sage.rings.number_field.number_field_order.Order_ideal.NumberFieldOrderIdeal_quadratic method), 524
is_rational() (sage.rings.number_field.number_field_element_number_field_element_quadratic.NumberFieldElement_quadratic method), 348
is_rational() (sage.rings.number_field.number_field_element.NumberFieldElement method), 299
is_rational() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField_element method), 725
is_real() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField_element method), 725
is_real_place() (in module sage.rings.num-


```plaintext
ber_field.number_field), 185

is_real_positive() (sage.rings.number_field.number_field_element.NumberFieldElement_absolute method), 326

is_relative() (sage.rings.number_field.number_field_number_field.Generic method), 113

is_RelativeNumberField() (in module sage.rings.number_field.number_field_rel), 239

is_S_integral() (sage.rings.number_field.number_field_number_field_Ideal.FractionalIdeal method), 458

is_S_unit() (sage.rings.number_field.number_field_number_field_Ideal.FractionalIdeal method), 458

is_simple() (sage.rings.qqbar.ANDescr method), 629

is_simple() (sage.rings.qqbar.ANExtensionElement method), 629

is_simple() (sage.rings.qqbar.ANRational method), 635

is_sqrt_disc() (in module sage.rings.number_field.number_field_element_quadratic), 361

is_square() (sage.rings.number_field.number_field_element.NumberFieldElement method), 300

is_square() (sage.rings.qqbar.AlgebraicNumber_base method), 672

is_square() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 726

is_suborder() (sage.rings.number_field.order.Order method), 417

is_suborder() (sage.rings.number_field.order.Order_abs method), 438

is_surjective() (sage.rings.number_field.maps.NumberFieldIsomorphism method), 396

is_totally_imaginary() (sage.rings.number_field.number_field_number_field_generic method), 113

is_totally_positive() (sage.rings.number_field.number_field_element.NumberFieldElement method), 301

is_totally_real() (sage.rings.number_field.number_field_element.NumberFieldElement method), 301

is_trivial() (sage.rings.number_field.number_field_Ideal.FractionalIdeal method), 461

is_trivial() (sage.rings.qqbar.AlgebraicGenerator method), 653

is_unit() (sage.rings.number_field.number_field_element.NumberFieldElement method), 301

is_zero() (sage.rings.number_field.number_field_Ideal.NumericalFieldElement, NumberFieldFractionalIdeal_rel method), 510

is_zero() (sage.rings.number_field.number_field_Ideal.NumericalFieldIdeal method), 487

isolating_interval() (in module sage.rings.qqbar), 703

K

K0_func() (in module sage.rings.number_field.number_field.S_unit_solver), 549

K1_func() (in module sage.rings.number_field.number_field.S_unit_solver), 550

key() (sage.rings.number_field.splitting_field.SplittingData method), 240

krull_dimension() (sage.rings.number_field.order.Order method), 418

lagrange_degree_3() (in module sage.rings.number_field.totallyreal_data), 746

late_import() (in module sage.rings.universal_cyclotomic_field), 732

lift() (sage.rings.number_field.number_field_element.NumberFieldElement_absolute method), 328

lift() (sage.rings.number_field.number_field_element.NumberFieldElement_relative method), 332

lift_to_base() (sage.rings.number_field.number_field_Ideal.NumericalField_rel method), 223

LiftMap (class in sage.rings.number_field.number_field_number_field, 447

list() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 263

list() (sage.rings.number_field.homset.CyclotomicFieldHomset method), 373

list() (sage.rings.number_field.homset.RelativeNumberFieldHomset method), 378

list() (sage.rings.number_field.homset.RelativeNumberFieldElement method), 378

list() (sage.rings.number_field.homset.RelativeNumberFieldElement method), 302

list() (sage.rings.number_field.number_field_element.NumberFieldElement_absolute method), 328

list() (sage.rings.number_field.number_field_element.NumberFieldElement_relative method), 332

lmfdb_page() (sage.rings.number_field.number_field.generic, 114

local_height() (sage.rings.number_field.number_field_element.NumberFieldElement method), 114

Index

766
```
method), 302
local_height_arch() (sage.rings.number_field.number_field_element.NumberFieldElement method), 303
log() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian method), 341
log() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 359
log() (sage.rings.number_field.number_field_element.NumberFieldElement absolute method), 307
log() (sage.rings.number_field.number_field_element.NumberFieldElement absolute method), 328
log() (sage.rings.number_field.number_field_element.OrderElement relative method), 337
log() (sage.rings.qqbar.AlgebraicNumber base method), 673
log() (sage.rings.qqbar.ANExtensionElement method), 630
log() (sage.rings.qqbar.ANRational method), 635
log() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 726

M
MapAbsoluteToRelativeNumberField (class in sage.rings.number_field.maps), 388
MapNumberFieldToVectorSpace (class in sage.rings.number_field.maps), 388
MapRelativeNumberFieldToRelativeVectorSpace (class in sage.rings.number_field/maps), 388
MapRelativeNumberFieldToVectorSpace (class in sage.rings.number_field/maps), 389
MapRelativeToAbsoluteNumberField (class in sage.rings.number_field/maps), 390
MapRelativeVectorSpaceToRelativeNumberField (class in sage.rings.number_field/maps), 392
MapVectorSpaceToNumberField (class in sage.rings.number_field/maps), 393
MapVectorSpaceToRelativeNumberField (class in sage.rings.number_field/maps), 395
matching_root() (in module sage.rings.number_field_number_field_element_morphisms), 385
matrix() (sage.rings.number_field.number_field_element.NumberFieldElement method), 305
maximal_order() (sage.rings.number_field.number_field_element.NumberFieldElement method), 193
maximal_order() (sage.rings.number_field.number_field_number_field_generic method), 115
maximal_totally_real_subfield() (sage.rings.number_field.number_field_number_field_generic method), 117
minimal_vector() (in module sage.rings.number_field.S_unit_solver), 577
minkowski_bound() (sage.rings.number_field.number_field_base.NumberField method), 194
minkowski_embedding() (sage.rings.number_field.number_field_number_field_number_field_element_absolute method), 42
minpoly() (sage.rings.number_field.number_field_element_number_field_element_quadratic method), 348
minpoly() (sage.rings.number_field.number_field_element_number_field_element_quadratic.OrderElement_quadratic method), 359
minpoly() (sage.rings.number_field.number_field_element.NumberFieldElement absolute method), 307
minpoly() (sage.rings.number_field.number_field_element.OrderElement relative method), 337
minpoly() (sage.rings.qqbar.AlgebraicNumber base method), 673
minpoly() (sage.rings.qqbar.ANExtensionElement method), 630
minpoly() (sage.rings.qqbar.ANRational method), 635
minpoly() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 726

module
sage.rings.number_field.bdd_height, 362
sage.rings.number_field.class_group, 526
sage.rings.number_field.galois_group, 250
sage.rings.number_field.homset, 373
sage.rings.number_field.maps, 387
sage.rings.number_field.morphism, 371
sage.rings.number_field.number_field, 1
sage.rings.number_field.number_field_base, 191
sage.rings.number_field.number_field_element, 269
sage.rings.number_field.number_field_element_quadratic, 339
sage.rings.number_field.number_field_ideal, 447
sage.rings.number_field.number_field_ideal_rel, 498
sage.rings.number_field.number_field_morphisms, 380
sage.rings.number_field.number_field_rel, 197
sage.rings.number_field.order, 401
sage.rings.number_field.order_ideal, 517
sage.rings.number_field.S_unit_solver, 548
sage.rings.number_field.selmer_group, 593
sage.rings.number_field.small_primes_of_degree_one, 590
sage.rings.number_field.splitting_field, 240
sage.rings.number_field.structure, 397
sage.rings.number_field.totallyreal, 735
sage.rings.number_field.totallyreal_data, 745
sage.rings.number_field.totallyreal_pbc, 748
sage.rings.number_field.totallyreal_rel, 739
sage.rings.number_field.unit_group, 537
sage.rings.qqbar, 601
sage.rings.universal_cyclo-

tomic_field, 713

module() (sage.rings.number_field.order.Order_absolute method), 434
multiplicative_order() (sage.rings.number_field.number_field_element.NumberFieldElement method), 308
multiplicative_order() (sage.rings.qqbar.AlgebraicNumber method), 660
multiplicative_order() (sage.rings.qqbar.AlgebraicReal method), 683
multiplicative_order() (sage.rings.universal_cyclo-
tomic_field.UniversalCyclotomicFieldElement method), 727
mus() (in module sage.rings.number_field.S_unit_solver), 578

N
NameChange (class in sage.rings.number_field.structure), 398
NameChangeMap (class in sage.rings.number_field.maps), 395
narrow_class_group() (sage.rings.number_field_number_field_generic method), 120
neg() (sage.rings.qqbar.ANDescr method), 622
neg() (sage.rings.qqbar.ANElement method), 630
neg() (sage.rings.qqbar.ANRational method), 636
next() (sage.rings.number_field_small_primes_of_degree_one_small_primes_of_degree_one_iter method), 592
next_split_prime() (sage.rings.number_field_number_field_cyclotomic method), 62
ngens() (sage.rings.number_field_number_field_rel.NumberField_relative method), 224
ngens() (sage.rings.number_field_number_field_NumberField_generic method), 121
ngens() (sage.rings.number_field_number_field.Order method), 419
ngens() (sage.rings.qqbar.AlgebraicField method), 646
ngens() (sage.rings.qqbar.AlgebraicRealField method), 691
norm() (sage.rings.number_field_number_field_element_quadratic_number_field_element_quadratic method), 349
norm() (sage.rings.number_field_number_field_element_quadratic.OrderElement_quadratic method), 359
norm() (sage.rings.number_field_number_field_element.NumberFieldElement method), 309
norm() (sage.rings.number_field_number_field_ideal NUMBER_REAL_ideal method), 510
norm() (sage.rings.number_field_number_field_ideal_number_field_ideal method), 487
norm() (sage.rings.number_field_number_field_ideal_number_field_ideal_generic method), 520
norm() (sage.rings.qqbar.AlgebraicNumber method), 661
norm() (sage.rings.qqbar.ANElement method), 631
norm() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 728
nth_root() (sage.rings.number_field_number_field_element.NumberFieldElement method), 311
nth_root() (sage.rings.qqbar.AlgebraicNumber_base method), 673
number_field() (sage.rings.number_field_class_group.ClassGroup method), 529
number_field() (sage.rings.number_field_galois_group.GaloisGroup_v1 method), 256
number_field() (sage.rings.number_field_galois_group.GaloisGroup_v2 method), 263
number_field() (sage.rings.number_field_number_field_ideal.NumberFieldIdeal method), 488
number_field() (sage.rings.number_field_order.Order method), 419
number_field() (sage.rings.number_field_unit_group.UnitGroup method), 544
Algebraic Numbers and Number Fields, Release 10.4

optimized_subfields() (sage.rings.number_field.number_field.NumberField_absolute method), 44

options (sage.rings.qqbar.AlgebraicField_common attribute), 651

ord() (sage.rings.number_field.number_field_element.NumberFieldElement method), 312

Order (class in sage.rings.number_field.order), 405

order() (sage.rings.number_field.galois_group.GaloisGroup_v1 method), 257

order() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 264

order() (sage.rings.number_field_homset.NumberFieldHomset method), 376

order() (sage.rings.number_field_number_field.NumberField_relative method), 225

order() (sage.rings.number_field_number_field.NumberField_absolute method), 46

order() (sage.rings.number_field_number_field.NumberField.generic method), 122

order() (sage.rings.qqbar.AlgebraicField_common method), 651

Order_absolute (class in sage.rings.number_field.order), 428

order_of_conductor() (sage.rings.number_field_number_field.NumberField_quadratic method), 179

Order_relative (class in sage.rings.number_field.order), 435

OrderElement_absolute (class in sage.rings.number_field_number_field_number_field_element), 333

OrderElement_quadratic (class in sage.rings.number_field_number_field_number_field_element_quadratic), 357

OrderElement_relative (class in sage.rings.number_field_number_field_number_field_element_quadratic), 334

OrderFactory (class in sage.rings.number_field_order), 427

P

p_adicLLL_bound() (in module sage.rings.number_field.S_unit_solv), 579

p_adicLLL_bound_one_prime() (in module sage.rings.number_field.S_unit_solv), 579

pari_absolute_base_polynomial() (sage.rings.number_field_number_field_number_field_relative method), 227

pari_bnf() (sage.rings.number_field_number_field_NumberField_generic method), 122

pari_field() (sage.rings.qqbar.AlgebraicGenerator method), 653

pari_hnf() (sage.rings.number_field_number_field_number_field_ideal.NumberFieldIdeal method), 488

pari_label() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 264

pari_nf() (sage.rings.number_field_number_field_NumberField_generic method), 123

pari_polynomial() (sage.rings.number_field_number_field_NumberField_generic method), 125

pari_prime() (sage.rings.number_field_number_field_number_field_ideal.NumberFieldIdeal method), 488

pari_relative_polynomial() (sage.rings.number_field_number_field_number_field_rel.NumberField_relative method), 228

pari_rhnf() (sage.rings.number_field_number_field_number_field_rel.NumberField_relative method), 228

pari_rnf() (sage.rings.number_field_number_field_number_field_fracallIdeal_rel method), 511

pari_rnf() (sage.rings.number_field_number_field_number_field_relative method), 228

pari_rnfnorm_data() (sage.rings.number_field_number_field_NumberField_relative method), 488

pari_zk() (sage.rings.number_field_number_field_NumberField_generic method), 127

parts() (sage.rings.number_field_number_field_element_quadratic.NumberFieldElement_quadratic method), 351

places() (sage.rings.number_field_number_field_number_field_relative method), 228

places() (sage.rings.number_field_number_field_number_field_NumberField_absolute method), 47

poldegree() (sage.rings.number_field.splitting_field.SplittingData method), 241

poly() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 680

polynomial() (sage.rings.number_field_number_field_number_field_element.NumberFieldElement method), 313

polynomial() (sage.rings.number_field_number_field_number_field_relative method), 230

polynomial() (sage.rings.number_field_number_field_NumberField_generic method), 127

polynomial_NT1() (sage.rings.number_field_number_field_NumberField_generic method), 128

polynomial_quotient_ring() (sage.rings.number_field_number_field_NumberField_generic method), 128

polynomial_ring() (sage.rings.number_field_number_field_NumberField_generic method), 128

polynomial_root() (sage.rings.qqbar.AlgebraicField method), 646

polynomial_root() (sage.rings.qqbar.AlgebraicRe-
### Q

- Q_to_quadratic_field_element (class in sage.rings.number_field.number_field_element_quadratic), 360
- quadratic_defect () (sage.rings.number_field.number_field.NumberField_generic method), 141
- quadratic_form () (sage.rings.number_field.orderIdeal.NumberFieldOrderIdeal_quadratic method), 524
- quadratic_order_class_number () (in module sage.rings.number_field.order), 446
- QuadraticField () (in module sage.rings.number_field.number_field), 180
- quotient_char_p () (in module sage.rings.number_field.number_field.number_field_ideal), 496
- QuotientMap (class in sage.rings.number_field.number_field_number_field_ideal), 494

### R

- radical_expression () (sage.rings.qqbar.AlgebraicNumber_base method), 674
- ramification_breaks () (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 264
- ramification_degree () (sage.rings.number_field_galois_group.GaloisGroupElement method), 251
- ramification_group () (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 265
- ramification_group () (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 489
- ramification_index () (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal_rel method), 511
- ramification_index () (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 464
- random_element () (sage.rings.number_field.number_field_number_field_ideal.NumberFieldIdeal method), 489
- random_element () (sage.rings.number_field.number_field_number_field_ideal.NumberFieldIdealRel method), 489
- random_element () (sage.rings.number_field.number_field_order.Order method), 420
- random_element () (sage.rings.qqbar.AlgebraicField method), 647
- random_element () (sage.rings.qqbar.AlgebraicRealField method), 692
- rank () (sage.rings.number_field_order.Order method), 422
- rank () (sage.rings.number_field.unit_group.UnitGroup method), 545
rational_argument() (sage.rings.qqbar.AlgebraicNumber method), 661
rational_argument() (sage.rings.qqbar.ANExtensionElement method), 631
rational_argument() (sage.rings.qqbar.ANRealNumber method), 636
rational_exact_root() (sage.rings.qqbar), 712
ray_class_number() (sage.rings.number_field.number_field_number_field_ideal.NumberFieldFractionalIdeal method), 465
real() (sage.rings.number_field_number_field_element_quadratic.NumberFieldElement_gaussian method), 341
real() (sage.rings.number_field_number_field_element_quadratic.NumberFieldElement_quadratic method), 351
real() (sage.rings.qqbar.AlgebraicNumber method), 662
real() (sage.rings.qqbar.AlgebraicReal method), 683
real() (sage.rings.qqbar.ANDescr method), 623
real() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 728
real_embeddings() (sage.rings.number_field_number_field_number_field_cyclotomic.UniversalCyclotomicFieldElement method), 63
real_embeddings() (sage.rings.number_field_number_field_number_field_cyclotomic.UniversalCyclotomicFieldElement method), 728
real_exact() (sage.rings.qqbar.AlgebraicReal method), 684
real_number() (sage.rings.qqbar.AlgebraicReal method), 685
real_part() (sage.rings.number_field_number_field_element_quadratic.NumberFieldElement_gaussian method), 341
real_part() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 728
real_places() (sage.rings.number_field_number_field_number_field_cyclotomic.UniversalCyclotomicFieldElement method), 49
reduce() (sage.rings.number_field.class_group.FractionalIdealClass method), 532
reduce() (sage.rings.number_field.number_field_number_field_ideal.NumberFieldFractionalIdeal method), 465
reduce_data() (sage.rings.number_field.order.AbsoluteOrderFactory method), 402
reduce_equiv() (sage.rings.number_field.number_field_number_field_ideal.NumberFieldIdeal method), 490
reduced_basis() (sage.rings.number_field_number_field_number_field_generic method), 144
reduced_gram_matrix() (sage.rings.number_field_number_field_number_field_generic method), 145
reduction_step_complex_case() (in module sage.rings.number_field.SUnitSolver), 582
refine_embedding() (in module sage.rings.number_field_number_field_number_field), 187
refine_interval() (sage.rings.qqbar.ANRoot method), 640
regulator() (sage.rings.number_field_number_field_NumberField_generic method), 147
relative_degree() (sage.rings.number_field_number_field_number_field_rel.NumberField_relative method), 230
relative_degree() (sage.rings.number_field_number_field_number_field_number_field_absolute method), 50
relative_different() (sage.rings.number_field_number_field_number_field_number_field_rel.NumberField_relative method), 230
relative_different() (sage.rings.number_field_number_field_number_field_number_field_absolute method), 50
relative_discriminant() (sage.rings.number_field_number_field_number_field_number_field_rel.NumberField_relative method), 231
relative_discriminant() (sage.rings.number_field_number_field_number_field_number_field_number_field_absolute method), 50
relative_norm() (sage.rings.number_field_number_field_number_field_element.NumberFieldElement method), 314
relative_norm() (sage.rings.number_field_number_field_number_field_number_field_number_field_rel.NumberField_relative method), 512
relative_order_from_ring_generators() (in module sage.rings.number_field.order), 446
relative_polynomial() (sage.rings.number_field_number_field_number_field_number_field_rel.NumberField_relative method), 231
relative_polynomial() (sage.rings.number_field_number_field_number_field_number_field_number_field_absolute method), 51
relative_ramification_index() (sage.rings.number_field_number_field_number_field_number_field_number_field_rel.NumberFieldFractionalIdeal method), 513
relative_ramification_index() (sage.rings.number_field_number_field_number_field_number_field_number_field_absolute method), 491
relative_vector_space() (sage.rings.number_field_number_field_number_field_number_field_rel.NumberField_relative method), 232
relative_vector_space() (sage.rings.number_field_number_field_number_field_number_field_number_field_absolute method), 51
relative_prime() (sage.rings.number_field.class_group.FractionalIdealClass method), 533
residue_class_degree() (sage.rings.number_field.number_field идеал rel.NumberFieldFractionalIdeal rel method), 514
residue_class_degree() (sage.rings.number_field.number_field идеал.NumberFieldFractionalIdeal method), 467
residue_field() (sage.rings.number_field.number_field идеал.NumberFieldFractionalIdeal method), 467
residue_field() (sage.rings.number_field.number_field generic method), 148
residue_field() (sage.rings.number_field.order.Order method), 422
residue_symbol() (sage.rings.number_field.number_field элемент.NumberFieldElement method), 315
residue_symbol() (sage.rings.number_field.number_field идеал.NumberFieldIdeal method), 492
residues() (sage.rings.number_field.number_field идеал rel.NumberFieldFractionalIdeal rel method), 514
residues() (sage.rings.number_field.number_field идеал.NumberFieldFractionalIdeal method), 470
ring_generators() (sage.rings.number_field.order.Order method), 423
ring_of_integers() (sage.rings.number_field.number_field base.NumberField method), 196
root_as_algebraic() (sage.rings.qqbar.AlgebraicGenerator method), 654
root_from_approx() (in module sage.rings.number_field.number_field морфизм), 386
roots_of_unity() (sage.rings.number_field.number_field rel.NumberField_relative method), 235
roots_of_unity() (sage.rings.number_field_number_field_cyclotomic method), 63
roots_of_unity() (sage.rings.number_field_number_field_generic method), 148
roots_of_unity() (sage.rings.number_field_number_field_unit_group.UnitGroup method), 546
round() (sage.rings.number_field.number_field_element quadratic.NumberFieldElement_quadratic method), 352
round() (sage.rings.number_field.number_field_element.NumberFieldElement method), 317
round() (sage.rings.qqbar.AlgebraicReal method), 687
S() (sage.rings.number_field.class_group.SClassGroup method), 534
S_class_group() (sage.rings.number_field_number_field_generic method), 67
S_ideal_class_log() (sage.rings.number_field_number_field идеал.NumberFieldIdeal method), 473
S_unit_group() (sage.rings.number_field_number_field_generic method), 68
S_unit_solutions() (sage.rings.number_field_number_field_generic method), 71
S_units() (sage.rings.number_field_number_field_generic method), 72
sage.rings.number_field.bdd_height module, 362
sage.rings.number_field.class_group module, 526
sage.rings.number_field.galois_group module, 250
sage.rings.number_field.homset module, 373
sage.rings.number_field.maps module, 387
sage.rings.number_field.morphism module, 371
sage.rings.number_field.number_field module, 1
sage.rings.number_field_number_field_base module, 191
sage.rings.number_field_number_field_element module, 269
scale() (sage.rings.qqbar.ANRational method), 636
SClassGroup (class in sage.rings.number_field.class_group), 534
section() (sage.rings.number_field.number_field_morphisms.CyclotomicFieldEmbedding method), 380
section() (sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldMorphism method), 382
selmer_generators() (sage.rings.number_field.number_field.NumberField_generic method), 149
selmer_group_iterator() (sage.rings.number_field.number_field.NumberField_generic method), 151
selmer_space() (sage.rings.number_field.number_field.NumberField_generic method), 152
SFractionalIdealClass (class in sage.rings.number_field.number_field_class_group), 535
short_prec_seq() (in module sage.rings.qqbar), 712
sieve_below_bound() (in module sage.rings.number_field.S_unit_solver), 583
sieve_ordering() (in module sage.rings.number_field.S_unit_solver), 584
sign() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 353
sign() (sage.rings.number_field.number_field_element.NumberFieldElement method), 318
sign() (sage.rings.qqbar.AlgebraicReal method), 687
degature() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 265
signature() (sage.rings.number_field.number_field_base.NumberField method), 196
signature() (sage.rings.number_field_number_field_number_field_cyclotomic method), 63
signature() (sage.rings.number_field_number_field_number_field_numberfieldmethod), 154
simplify() (sage.rings.qqbar.AlgebraicNumber_base method), 632
simplify() (sage.rings.qqbar.ANExtensionElement method), 632
Small_primes_of_degree_one_iter (class in sage.rings.number_field.small_primes_of_degree_one), 592
small_residue() (sage.rings.number_field_number_field_number_field_cyclotomic method), 471
smallest_integer() (sage.rings.number_field_number_field_number_field_fractionalideal method), 471
smallest_integer() (sage.rings.number_field_number_field_number_field_ideal_renumberfieldfractionalidealrel method), 471
solutions_fromSystems() (in module sage.rings.number_field.S_unit_solver), 585
solve_CRT() (sage.rings.number_field_number_field_number_field_generic method), 155
solve_S_unit_equation() (in module sage.rings.number_field.S_unit_solver), 586
some_elements() (sage.rings.number_field_num-
Algebraic Numbers and Number Fields, Release 10.4

some_elements() (sage.rings.number_field.order.Order method), 424
some_elements() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 719
specified_complex_embedding() (sage.rings.number_field.NumberField_generic method), 156
split_primes_large_lcm() (in module sage.rings.number_field.S_unit_solver), 588
splitting_field() (in module sage.rings.number_field.splitting_field), 242
SplittingData (class in sage.rings.number_field_splitting_field), 240
SplittingFieldAbort, 241
sqrt() (sage.rings.number_field_element.NumberFieldElement method), 319
sqrt() (sage.rings.qqbar.AlgebraicNumber_base method), 676
sqrt() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 729
structure() (sage.rings.number_field.NumberField_generic method), 158
subfield() (sage.rings.number_field.NumberField_generic method), 159
subfield_from_elements() (sage.rings.number_field.NumberField_generic method), 162
subfields() (sage.rings.number_field.NumberField_relative method), 236
subfields() (sage.rings.number_field.NumberField_absolute method), 54
Subgroup (sage.rings.number_field.galois_group.GaloisGroup_v2 attribute), 259
super_poly() (sage.rings.qqbar.AlgebraicGenerator method), 654
support() (sage.rings.number_field_number_field_element.NumberFieldElement method), 321
support() (sage.rings.number_field.NumberField_fractional_ideal.NumberFieldFractionalIdeal method), 472

T

tr_data (class in sage.rings.number_field.totallyreal_data), 747
tr_data_rel (class in sage.rings.number_field.totallyreal_rel), 744
trace() (sage.rings.number_field_number_field_element_quadratic.NumberFieldElement_quadratic method), 355
trace() (sage.rings.number_field_number_field_element_quadratic.OrderElement_quadratic method), 360
trace() (sage.rings.number_field_number_field_element.NumberFieldElement method), 322
trace_dual_basis() (sage.rings.number_field_number_field_NumberField_generic method), 163
trace_pairing() (sage.rings.number_field_number_field_NumberField_generic method), 164
transitive_number() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 266
tru() (sage.rings.qqbar.AlgebraicReal method), 689

U

UCF_sqrnt_int() (in module sage.rings.universal_cyclotomic_field), 716
UCFtoQQbar (class in sage.rings.universal_cyclotomic_field), 717
uniformizer() (sage.rings.number_field_number_field_NumberField_relative method), 238
uniformizer() (sage.rings.number_field_number_field_NumberField_generic method), 164
union() (sage.rings.qqbar.AlgebraicGenerator method), 655
unit_group() (sage.rings.number_field_number_field_NumberField_generic method), 166
UnitGroup (class in sage.rings.number_field_number_field_unit_group), 542
units() (sage.rings.number_field_number_field_NumberField_generic method), 168
UniversalCyclotomicField (class in sage.rings.universal_cyclotomic_field), 717
UniversalCyclotomicFieldElement (class in sage.rings.universal_cyclotomic_field), 721
unrank() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 267

V

valuation() (sage.rings.number_field_number_field_element.NumberFieldElement method), 322
valuation() (sage.rings.number_field_number_field_element_NumberFieldElement_relative method), 332
valuation()
(sage.rings.number_field.number_field_element.NumberFieldElement method), 172
valuation()
(sage.rings.number_field.number_field_element.Order method), 424
valuation()
(sage.rings.number_field.number_field_element.NumberFieldElement method), 323
valuation()
(sage.rings.number_field.number_field_element.NumberFieldRel method), 238
valuation()
(sage.rings.number_field.number_field.Element method), 493
valuation()
(sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 493
valuation()
(sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 515
zeta_order()
(sage.rings.number_field.number_field.Order method), 426
zeta_order()
(sage.rings.number_field.number_field_element.UniversalCyclotomicField method), 720
zeta_order()
(sage.rings.number_field.number_field.NumberField_cyclotomic method), 66
zeta_order()
(sage.rings.number_field.number_field.NumberField generic method), 174
zeta_order()
(sage.rings.number_field.number_field.UnitGroup method), 548
zero()
(sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 720
zeta()
(sage.rings.number_field.number_field.NumberField_cyclotomic method), 64
zeta()
(sage.rings.number_field.number_field.NumberField_generic method), 172
zeta()
(sage.rings.number_field.number_field.order.Order method), 426
zeta()
(sage.rings.number_field.number_field_unit_group.UnitGroup method), 546
zeta()
(sage.rings.qqbar.AlgebraicField method), 649
zeta()
(sage.rings.qqbar.AlgebraicRealField method), 693
zeta()
(sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 720
zeta_coefficients()
(sage.rings.number_field.number_field_element.NumberField_element_quadratic method), 361
zeta_order()
(sage.rings.number_field.number_field_unit_group.UnitGroup method), 546
zeta()
(sage.rings.qqbar.AlgebraicField method), 649
zeta()
(sage.rings.qqbar.AlgebraicRealField method), 693
zeta()
(sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 720
zeta_coefficients()
(sage.rings.number_field.number_field_number_field_unit_group.UnitGroup method), 174