## CONTENTS

1 Algebraic Number Fields ........................................... 1
2 Morphisms .................................................................. 185
3 Orders, Ideals, Ideal Classes ...................................... 201
4 Totally Real Fields .................................................... 299
5 Algebraic Numbers .................................................... 311
6 Indices and Tables ..................................................... 381

Bibliography .................................................................. 383

Python Module Index .................................................. 385

Index ........................................................................... 387
1.1 Number Fields

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).

This example follows one in the Magma reference manual:

```sage
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
(continues on next page)
```
Number Field in \(a\) with defining polynomial \(y^2 + y + 1\) over its base field

```
sage: KL.<a> = NumberField([x^4 - 420*x^2 + 40000, x^2 + x + 1]); KL
```
Number Field in \(b_0\) 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*sqrt2
```

```
sage: (sqrt2 + cuberoot3)^5
```
```
(20*cuberoot3^2 + 15*cuberoot3 + 4)*sqrt2 + 3*cuberoot3^2 + 20*cuberoot3 + 60
```

```
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: a = S(cuberoot2); a
```
```
cuberoot2
```

```
sage: 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).

**class** `sage.rings.number_field.number_field.CyclotomicFieldFactory`

**Bases:** `sage.structure.factory.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 `UniversalCyclotomicField`.

**INPUT:**

- \(n\) - a nonnegative integer, default: 0
- `names` - name of generator (optional - defaults to `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 **universal cyclotomic field**:

```
sage: CyclotomicField()
Universal Cyclotomic Field
```
We create the 7th cyclotomic field $\mathbb{Q}(\zeta_7)$ with the default generator name.

\begin{verbatim}
sage: k = CyclotomicField(7); k
Cyclotomic Field of order 7 and degree 6
sage: k.gen()
zeta7
\end{verbatim}

The default embedding sends the generator to the complex primitive $n^{th}$ root of unity of least argument.

\begin{verbatim}
sage: CC(k.gen())
0.623489801858734 + 0.781831482468030*I
\end{verbatim}

Cyclotomic fields are of a special type.

\begin{verbatim}
sage: type(k)
<class 'sage.rings.number_field.number_field.NumberField_cyclotomic_with_category'>
\end{verbatim}

We can specify a different generator name as follows.

\begin{verbatim}
sage: k.<z7> = CyclotomicField(7); k
Cyclotomic Field of order 7 and degree 6
sage: k.gen()
z7
\end{verbatim}

The $n$ must be an integer.

\begin{verbatim}
sage: CyclotomicField(3/2)
Traceback (most recent call last):
  ...TypeError: no conversion of this rational to integer
\end{verbatim}

The degree must be nonnegative.

\begin{verbatim}
sage: CyclotomicField(-1)
Traceback (most recent call last):
  ...ValueError: n (=1) must be a positive integer
\end{verbatim}

The special case $n = 1$ does not return the rational numbers:

\begin{verbatim}
sage: CyclotomicField(1)
Cyclotomic Field of order 1 and degree 1
\end{verbatim}

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

\begin{verbatim}
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
\end{verbatim}
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

create_key\((n=0, names=None, embedding=True)\)
Create the unique key for the cyclotomic field specified by the parameters.

create_object\((version, key, **extra_args)\)
Create the unique cyclotomic field defined by key.

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.
- **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_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 AlgebraicExtensionFunctor but we ignore them as they are not used.

**EXAMPLES:**

sage: z = QQ['z'].0
sage: K = NumberField(z^2 - 2,'s'); K

(continues on next page)
Constructing a relative number field:

```python
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
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
Number Field in a with defining polynomial x^2 + 3
sage: m
Number Field in a with defining polynomial 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:

```python
sage: K.<b> = NumberField(x^2 + 5, 'a'); K
Number Field in b with defining polynomial x^2 + 5
```

1.1. Number Fields
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.

```
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
  To:   Real Field with 53 bits of precision
    Defn: Generic morphism:
      From: Number Field in a with defining polynomial x^3 - 2
      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
```

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: 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^\rightarrow15 + 3*5^16 + 5^17 + 5^18 + 2*5^19 + O(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
  To:   Number Field in b with defining polynomial x^6 - x^2 + 1/10
  Defn: a -> b^2
```
The `QuadraticField` and `CyclotomicField` constructors create an embedding by default unless otherwise specified:

```sage
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
```

Comparison depends on the (real) embedding specified (or the one selected by default). Note that the codomain of the embedding must be $\mathbb{Q}\bar{a}$ or $\mathbb{A}$ for this to work (see trac ticket #20184):

```sage
sage: N.<g> = NumberField(x^3+2,embedding=1)
sage: 1 < g
False
sage: g > 1
False
sage: RR(g)
-1.25992104989487
```

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

```sage
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
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
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:

```sage
sage: L.<b> = NumberField(3*x^2 - 1/5)
sage: L.pari_polynomial()
x^2 - 15
```

(continues on next page)
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:

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

sage: L = NumberField(y^3 + y + 3, 'a'); L
Number Field in a with defining polynomial y^3 + y + 3
sage: L.defining_polynomial().parent()
Univariate Polynomial Ring in y over Rational Field

sage: W1 = NumberField(x^2+1,'a')
sage: K.<x> = CyclotomicField(5)[x]
sage: W.<a> = NumberField(x^2 + 1); W
Number Field in a with defining polynomial x^2 + 1 over its base field

The following has been fixed in trac ticket #8800:

sage: P.<x> = QQ[]
sage: K.<a> = NumberField(x^3-5,embedding=0)
sage: L.<b> = K.extension(x^2+a)
sage: F, R = L.construction()
sage: F(R) == L  # indirect doctest
True

Check that trac ticket #11670 has been fixed:

sage: K.<a> = NumberField(x^2 - x - 1)
sage: loads(dumps(K)) is K
True
sage: K.<a> = NumberField(x^3 - x - 1)
sage: loads(dumps(K)) is K
True
sage: K.<a> = CyclotomicField(7)

Another problem that was found while working on trac ticket #11670, maximize_at_primes and assume_disc_small were lost when pickling:

sage: K.<a> = NumberField(x^3-2, assume_disc_small=True, maximize_at_primes=[2], latex_name='\alpha', embedding=2^(1/3))
sage: L = loads(dumps(K))
sage: L.assume_disc_small
True
sage: L.maximize_at_primes
(2,)

It is an error not to specify the generator:

```python
sage: K = NumberField(x^2-2)
Traceback (most recent call last):
  ...  
TypeError: You must specify the name of the generator.
```

Check that we can construct morphisms to matrix space (trac ticket #23418):

```python
sage: t = polygen(QQ)
sage: K = NumberField(t^4 - 2, 'a')
sage: K.hom([K.gen().matrix()])
Ring morphism:
  From: Number Field in a with defining polynomial x^4 - 2
  To:   Full MatrixSpace of 4 by 4 dense matrices over Rational Field
  Defn: a |--> [0 1 0 0]
       [0 0 1 0]
       [0 0 0 1]
       [2 0 0 0]
```

```python
class sage.rings.number_field.number_field.NumberFieldFactory
Bases: sage.structure.factory.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 field 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:**

Returns 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: 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
```

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

```
sage: k.galois_group(type="pari")
Galois group PARI group [8, 1, 3, "E(8)=2[x]2[x]2"] of degree 8 of the NumberField in a with defining polynomial x^2 + 1 over its base field
```

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

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

A bigger tower of quadratic fields:

```python
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
```

The function can also be called by name:

```python
sage: NumberFieldTower([x^2 + 1, x^2 + 2], ['a','b'])
Number Field in a with defining polynomial x^2 + 1 over its base field
```

```python
class sage.rings.number_field.number_field.NumberField_absolute (polynomial, name, latex_name=None, check=True, embedding=None, assume_disc_small=False, maximize_at_primes=None, structure=None)
```

Bases: `sage.rings.number_field.number_field.NumberField_generic`

Function to initialize an absolute number field.

**EXAMPLES:**

1.1. Number Fields
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()

Minkowski_embedding(*args, **kwds)
Deprecated: Use minkowski_embedding() instead. See trac ticket #23685 for details.

abs_val(v, iota, prec=None)
Return the value $|\iota v|$.

INPUT:
- v – a place of K, finite (a fractional ideal) or infinite (element of K.places(prec))
- iota – an element of K
- prec – (default: None) the precision of the real field

OUTPUT:
The absolute value as a real number

EXAMPLES:

```python
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: 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
```

absolute_degree()
A synonym for degree.

EXAMPLES:

```python
sage: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_degree()
2
```

absolute_different()
A synonym for different.

EXAMPLES:

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

absolute_discriminant()
A synonym for discriminant.

EXAMPLES:
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: K.<i> = NumberField(x^2 + 1)
sage: K.absolute_discriminant()
-4
```

absolute_polynomial()

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

EXAMPLES:

```python
sage: K.<a> = NumberField(x^2 - 17)
sage: K.absolute_generator()
a
```

absolute_vector_space()

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 `self.vector_space()`.

EXAMPLES:

```python
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)
```

automorphisms()

Compute all Galois automorphisms of self.

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

EXAMPLES:

```python
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
]
```
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
```

$L$ 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')
sage: len(L.automorphisms())
24
```

Number fields defined by non-monic and non-integral polynomials are supported (trac ticket #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
```

```python
base_field()
Returns the base field of self, which is always QQ

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

```python
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.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.<z> = NumberField(x^2 + 3); K
Number Field in z with defining polynomial x^2 + 3
```

(continues on next page)
sage: L.<ww> = K.change_names()
sage: L
Number Field in ww with defining polynomial x^2 + 3

sage: L.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.structure()[0](ww + 5/3)
z + 5/3

\texttt{elements\_of\_bounded\_height(**kwds)}

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

This algorithm computes 2 lists: \texttt{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 [Doyle-Krumm]. Algorithm 5 is used for imaginary quadratic fields.

**INPUT:**

\texttt{kwds}:

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

**OUTPUT:**

- an iterator of number field elements

**EXAMPLES:**

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

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:

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]

sage: K.<a> = CyclotomicField(20)
sage: len(list(K.elements_of_bounded_height(bound=1)))
21

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

sage: K.<a> = NumberField(x^6 + 2)
sage: L = K.elements_of_bounded_height(bound=5)

(continues on next page)
sage: for t in L:
....: exp(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

sage: K.<a> = NumberField(x^2 + 17)

sage: L = K.elements_of_bounded_height(bound=120)

sage: len(list(L))
9047

sage: K.<a> = NumberField(x^4 - 5)

sage: L = K.elements_of_bounded_height(bound=50)

sage: len(list(L))  # long time (2 s)
2163

sage: K.<a> = CyclotomicField(13)

sage: L = K.elements_of_bounded_height(bound=2)

sage: len(list(L))  # long time (3 s)
27

sage: K.<a> = NumberField(x^6 + 2)

sage: L = K.elements_of_bounded_height(bound=60, precision=100)

sage: len(list(L))  # long time (5 s)
1899

sage: K.<a> = NumberField(x^4 - x^3 - 3*x^2 + x + 1)

sage: L = K.elements_of_bounded_height(bound=10, tolerance=0.1)

sage: len(list(L))
99

AUTHORS:

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

`embeddings(K)`

Compute all field embeddings of self 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 self into $K$ is put first in the list.

**INPUT:**

- $K$ - a number field

**EXAMPLES:**

```python
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<a1> = K.galois_closure(); L
Number Field in a1 with defining polynomial x^6 + 108
sage: K.embeddings(L)[0]
Ring morphism:
    From: Number Field in a with defining polynomial x^3 - 2
    To:   Number Field in a1 with defining polynomial x^6 + 108
    Defn: a |--> 1/18*a1^4
sage: K.embeddings(L)
is K.embeddings(L)
True
```

We embed a quadratic field into a cyclotomic field:

```python
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
    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
    To:   Cyclotomic Field of order 7 and degree 6
    Defn: a |--> -2*zeta7^4 - 2*zeta7^2 - 2*zeta7 - 1]
```

We embed a cubic field in the complex numbers:

```python
sage: K.<a> = NumberField(x^3 - 2)
sage: K.embeddings(CC)
[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,
    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,
    Ring morphism:
    From: Number Field in a with defining polynomial x^3 - 2
    To:   Complex Field with 53 bits of precision
    Defn: a |--> 1.25992104989487]
```

Test that trac ticket #15053 is fixed:

```python
sage: K = NumberField(x^3 - 2, 'a')
sage: K.embeddings(GF(3))
[]
```
galois_closure(names=None, map=False)

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

INPUT:

- names - variable name for Galois closure
- map - (default: False) also return an embedding of self into \( K \)

EXAMPLES:

```python
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
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
```

A cyclotomic field is already Galois:

```python
sage: K.<a> = NumberField(cyclotomic_polynomial(23))
sage: L.<z> = K.galois_closure()
sage: L
Number Field in z with defining polynomial x^22 + x^21 + x^20 + x^19 + x^18 + \ldots + 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:

```python
sage: F.<a> = NumberField(x^2-x-1)
sage: F.hilbert_conductor(2*a,F(-1))
```
AUTHOR:

• Aly Deines

hilbert_symbol(a, b, P=None)

Returns 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 \(P\) is 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:

\[
\begin{align*}
\text{sage: } & K.<a> = NumberField(x^2 - 23) \\
\text{sage: } & K.hilbert_symbol(0, a+5) \\
& 0 \\
\text{sage: } & K.hilbert_symbol(a, 0) \\
& 0 \\
\text{sage: } & K.hilbert_symbol(-a, a+1) \\
& 1 \\
\text{sage: } & K.hilbert_symbol(-a, a+2) \\
& -1 \\
\text{sage: } & K.hilbert_symbol(a, a+5) \\
& -1
\end{align*}
\]

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:

\[
\begin{align*}
\text{sage: } & K.hilbert_symbol(-a, a+2, a+2) \\
& -1 \\
\text{sage: } & K.hilbert_symbol(a, a+5, K.ideal(23).factor()[0][0]) \\
& -1
\end{align*}
\]

More local examples:

\[
\begin{align*}
\text{sage: } & K.hilbert_symbol(a, 0, K.ideal(5)) \\
& 0
\end{align*}
\]
sage: K.hilbert_symbol(a, a+5, K.ideal(5))
1
sage: K.hilbert_symbol(a+1, 13, (a+6)*K.maximal_order())
-1
sage: [emb1, emb2] = K.embeddings(AA)
sage: K.hilbert_symbol(a, -1, emb1)
-1
sage: K.hilbert_symbol(a, -1, emb2)
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
sage: rho = 2*a^4 + 3*a^3 + 4*a^2 + 15*a + 11
sage: K.hilbert_symbol(a, a+5, rho)
1

This also works for non-principal ideals:

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:

sage: K.<a> = NumberField(x^5 - 23)
sage: O = K.maximal_order()
sage: p = [p[0] for p in (2*O).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

Various real fields are allowed:

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:
Traceback (most recent call last):
...
ValueError: Possibly real place (=Ring morphism:
  From: Number Field in a with defining polynomial x^2 - 5
  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
  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
a and b do not have to be integral or coprime:

sage: K.<i> = QuadraticField(-1)
sage: O = K.maximal_order()
sage: K.hilbert_symbol(1/2, 1/6, 3*O)
1
sage: p = 1+i
sage: K.hilbert_symbol(p, p, p)
1
sage: K.hilbert_symbol(p, 3*p, p)
1
sage: K.hilbert_symbol(3, p, p)
1
sage: K.hilbert_symbol(1/3, 1/5, 1+i)
1
sage: L = QuadraticField(5, 'a')
sage: L.hilbert_symbol(-3, -1/2, 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^5-23)
sage: P = K.ideal(-1105*a^4 + 1541*a^3 - 795*a^2 - 2993*a + 11853)
sage: K.hilbert_symbol(a, a+5, P)
1
sage: K.hilbert_symbol(a, 2, P)
1
sage: K.hilbert_symbol(a+5, 2, P)
-1
sage: K.<a> = NumberField(x^3 - 4*x + 2)
sage: K.hilbert_symbol(2, -2, K.primes_above(2)[0])
1

Check that the bug reported at trac ticket #16043 has been fixed:

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

AUTHOR:

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

is_absolute()
Returns True since self is an absolute field.

EXAMPLES:

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

maximal_order(v=None)
Return the maximal order, i.e., the ring of integers, associated to this number field.

INPUT:

• v - (default: None) None, a prime, or a list of primes.
  – if v is None, return the maximal order.
  – if v is a prime, return an order that is $p$-maximal.
  – if v is a list, return an order that is maximal at each prime in the list v.

EXAMPLES:

In this example, the maximal order cannot be generated by a single element:

sage: k.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: o = k.maximal_order()
sage: o
Maximal Order 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); q = next_prime(10^23)
Sage: K.<a> = NumberField(x^3 - p*q)
Sage: K.maximal_order([3]).basis()
  [1/3*a^2 + 1/3*a + 1/3, a, a^2]
Sage: K.maximal_order([2]).basis()
  [1/3*a^2 + 1/3*a + 1/3, a, a^2]
Sage: K.maximal_order([p]).basis()
  [1/3*a^2 + 1/3*a + 1/3, a, a^2]
Sage: K.maximal_order([q]).basis()
  [1/3*a^2 + 1/3*a + 1/3, a, a^2]
Sage: K.maximal_order([p,3]).basis()
  [1/3*a^2 + 1/3*a + 1/3, a, a^2]

An example with bigger discriminant:
Sage: p = next_prime(10^97); q = next_prime(10^99)
Sage: K.<a> = NumberField(x^3 - p*q)
Sage: K.maximal_order(prime_range(10000)).basis()
  [1, a, a^2]

minkowski_embedding (B=None, prec=None)
Return an nxn 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 prec is not None, use RealField(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

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

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: F.<alpha> = NumberField(x^3+2)
Sage: F.minkowski_embedding()
  [ 1.00000000000000 -1.25992104989487  1.58740105196820]
  [ 1.41421356237309 0.890898718131508 -1.22462048382677]
  [0.00000000000000 1.54308184421110  1.94416129723666]
Sage: F.minkowski_embedding([1, alpha+2, alpha^2-alpha])
  [ 1.00000000000000 0.740078950105127  2.01336076644328]
  [ 1.41421356237309 3.71932584285714 -2.01336076644328]
  [0.00000000000000 1.54308184421110  0.40107945302014]
Sage: F.minkowski_embedding() * (alpha + 2).vector().column()
  [0.740078950105127]
  [ 3.71932584285714]
  [ 1.54308184421110]

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.

1.1. Number Fields 23
EXAMPLES: We construct a compositum of 3 quadratic fields, then find an optimized representation and transform elements back and forth.

```
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 +
\rightarrow 576
sage: L, from_L, to_L = K.optimized_representation()
sage: L
# your answer may different, since algorithm is random
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
\rightarrow 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/
\rightarrow 24*b - 3/4
```

The transformation maps are mutually inverse isomorphisms.

```
sage: from_L(to_L(K.0)) == K.0
True
sage: to_L(from_L(L.0)) == L.0
True
```

Number fields defined by non-monic and non-integral polynomials are supported (trac ticket #252):

```
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
    \rightarrow 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
    \rightarrow 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:

```
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 +
\rightarrow 576
sage: L = K.optimized_subfields(name='b')
sage: L[0][0]
Number Field in b0 with defining polynomial x
sage: L[1][0]
Number Field in b1 with defining polynomial x^2 - 3*x + 3
sage: [z[0] for z in L]
# random -- since algorithm is random
[Number Field in b0 with defining polynomial x - 1,
 Number Field in b1 with defining polynomial x^2 - x + 1,
 Number Field in b2 with defining polynomial x^4 - 5*x^2 + 25,
```

(continues on next page)
We examine one of the optimized subfields in more detail:

```
sage: M, from_M, to_M = L[2]
sage: M
Number Field in b2 with defining polynomial x^4 - 5*x^2 + 25
sage: from_M
Ring morphism:
  From: Number Field in b2 with defining polynomial x^4 - 5*x^2 + 25
  To:   Number Field in a1 with defining polynomial x^8 + 40*x^6 + 352*x^4 + 960*x^2 + 576
  Defn: b2 |--> -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
```

The to_M map is None, since there is no map from K to M:

```
sage: to_M
```

We apply the from_M map to the generator of M, which gives a rather large element of \( K \):

```
sage: from_M(M.0)
-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:

```
sage: from_M(M.0).minpoly()
```

```
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:**

```
sage: k.<i> = NumberField(x^2 + 1)
sage: k.order(2*i)
Order in Number Field in i with defining polynomial x^2 + 1
sage: k.order(10*i)
Order in Number Field in i with defining polynomial x^2 + 1
sage: k.order(3)
Traceback (most recent call last):
```

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

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

places \((all\_complex=False, prec=None)\)

Return the collection of all infinite places of self.

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

On the other hand, if \( \text{prec} \) is not None, we simply return places into RealField(\( \text{prec} \)) and ComplexField(\( \text{prec} \)) (or RDF, CDF if \( \text{prec}=53 \)). One can also use \( \text{prec}={} \text{infinity} \), which returns embeddings into the field \( \mathbb{Q} \) 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 CIF instead of RIF.

EXAMPLES:

```
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
  To: Real Field with 106 bits of precision
  Defn: alpha |--> -10.00499625499181184573367219280,
  Ring morphism:
  From: Number Field in alpha with defining polynomial x^3 - 100*x + 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 - 100*x + 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 |--> -1.912931182772389101199116839549,
  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 + 1.6566469997230*I]
sage: F.<alpha> = NumberField(x^3+7) ; 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 |--> 0.956465591386195 + 1.6566469997230*I]
```

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 + 1.6566469997230*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)

\[\text{real_places} \ (\text{prec} = \text{None})\]
Return all real places of self as homomorphisms into RIF.

EXAMPLES:

```
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 |--> -1.626576561697785743211232345494,
Ring morphism:
From: Number Field in alpha with defining polynomial \(x^4 - 7\)
To: Real Field with 106 bits of precision
Defn: alpha |--> 1.626576561697785743211232345494]

\[\text{relative_degree}()\]
A synonym for degree.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_degree()
```

2

\[\text{relative_different}()\]
A synonym for different.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_different()
```

Fractional ideal (2)

\[\text{relative_discriminant}()\]
A synonym for discriminant.

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_discriminant()
```

-4
relative_polynomial()  
A synonym for polynomial.

EXAMPLES:
```python
sage: K.<i> = NumberField(x^2 + 1)
sage: K.relative_polynomial()
x^2 + 1
```

relative_vector_space()  
A synonym for vector_space.

EXAMPLES:
```python
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:
- \( \alpha \) - an element of self or an embedding of a subfield into self
- \( \text{names} \) - 2-tuple of names of generator for output field \( K \) and the subfield \( \mathbb{Q}(\alpha) \) \( \text{names}[0] \) generators \( K \) and \( \text{names}[1] \) \( \mathbb{Q}(\alpha) \).
- \( \text{structure} \) – an instance of `structure.NumberFieldStructure` or None (default: None), if None, then the resulting field’s `structure()` will return isomorphisms from and to this field. Otherwise, the field will be equipped with `structure`.

OUTPUT:
\( K \) – relative number field

Also, \( K\_\text{structure}() \) 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.<a> = NumberField(x^10 - 2)
sage: L.<c,d> = K.relativize(a^4 + a^2 + 2); L
Number Field in c with defining polynomial x^2 - 1/5*d^4 + 8/5*d^3 - 23/5*d^2 - 18/5 over its base field
sage: c.absolute_minpoly()
x^10 - 2
sage: d.absolute_minpoly()
x^5 - 10*x^4 + 40*x^3 - 90*x^2 + 110*x - 58
sage: (a^4 + a^2 + 2).minpoly()
x^5 - 10*x^4 + 40*x^3 - 90*x^2 + 110*x - 58
sage: from_L, to_L = L.structure()
sage: to_L(a)
```

(continues on next page)
The following demonstrates distinct embeddings of a subfield into a larger field:

```python
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:

```python
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
```

**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: K.<a> = NumberField([x^3 - 2, x^2 + x + 1])
sage: K = K.absolute_field('b')
sage: S = K.subfields()
(continues on next page)```
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

vector_space()

Return a vector space V and isomorphisms self \rightarrow V and V \rightarrow self.

OUTPUT:

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

EXAMPLES:

sage: k.<a> = NumberField(x^3 + 2)
sage: V, from_V, to_V = k.vector_space()
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(2/3*a - 5/8))
2/3*a - 5/8
sage: to_V(from_V(V([0,-1/7,0])))
(0, -1/7, 0)

sage.rings.number_field.number_field.NumberField_absolute_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
```

```

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

Bases: sage.rings.number_field.number_field.NumberField_absolute

Create a cyclotomic extension of the rational field.

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

```python
sage: cf12 = CyclotomicField(12)
sage: z12 = cf12.0
sage: cf6 = CyclotomicField(6)
sage: z6 = cf6.0
sage: FF = Frac( cf12['x'] )
sage: x = FF.0
sage: z6*x^3/(z6 + x)
zeta12^2*x^3/(x + zeta12^2)
```

```python
sage: cf6 = CyclotomicField(6) ; z6 = cf6.gen(0)
sage: cf3 = CyclotomicField(3) ; z3 = cf3.gen(0)
sage: cf3(z6)
zeta3 + 1
sage: cf6(z3)
zeta6 - 1
sage: type(cf6(z3))
<type 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic'>
```

continues on next page
sage: cf1 = CyclotomicField(1) ; z1 = cf1.0
sage: cf3(z1)
1
sage: type(cf3(z1))
<type 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic'>

complex_embedding(\texttt{\textit{prec}=53})

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

EXAMPLES:

sage: C = CyclotomicField(4)
sage: C.complex_embedding()
Ring morphism:
  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 \texttt{zeta} is computed (using \texttt{sin} and \texttt{cosine} in \texttt{MPFR}) means that only the \texttt{prec} bits of the number after the decimal point are valid.

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

complex_embeddings(\texttt{\textit{prec}=53})

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

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

EXAMPLES:

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,
(continues on next page)
construction()  
Return data defining a functorial construction of self.

EXAMPLES:

```python
sage: 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  
[0.309016994374948? + 0.951056516295154?*I]
sage: F.structures  
[None]
```

different()  
Returns the different ideal of the cyclotomic field self.

EXAMPLES:

```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
```

discriminant (v=None)  
Returns 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 element 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
```

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

EXAMPLES:
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: CyclotomicField(3).is_isomorphic(NumberField(x^2 + x +1, 'a'))
True
sage: CyclotomicField(18).is_isomorphic(CyclotomicField(9))
True
sage: CyclotomicField(10).is_isomorphic(NumberField(x^4 - x^3 + x^2 - x + 1, 'b'))
True
```

Check trac ticket #14300:

```
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
```

next_split_prime(p=2)
Return the next prime integer \( p \) that splits completely in this cyclotomic field (and does not ramify).

EXAMPLES:

```
sage: K.<z> = CyclotomicField(3)
sage: K.next_split_prime(7)
13
```

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
```

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: CyclotomicField(4).real_embeddings()
[]
sage: CyclotomicField(2).real_embeddings()
[Ring morphism:
```
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]
```

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:

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

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:

```
sage: k = CyclotomicField(4)
sage: k.zeta()
zeta4
sage: k.zeta(2)
-1
sage: k.zeta().multiplicative_order()
4
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()
6
sage: k.zeta(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)

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]
sage: [b^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

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
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: sage.misc.fast_methods.WithEqualityById, sage.rings.number_field.number_field_base.NumberField

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

EXAMPLES:

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

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

Returns the S-class group of this number field over its base field.

INPUT:

- $S$ - a set of primes of the base field
- \textit{proof} - if False, assume the GRH in computing the class group. Default is True. Call \texttt{number_field\_proof} to change this default globally.
- \textit{names} - names of the generators of this class group.

OUTPUT:

The S-class group of this number field.

EXAMPLES:

A well known example:

```
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
```

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

```
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
```

\textbf{S\_unit\_group} (\textit{proof}=\textit{None}, \textit{S}=\textit{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 + 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))
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))
```

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

```python
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
```

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

```python
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))
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 x Z of Number Field in a with defining polynomial x^3 + 3 with S = (Fractional ideal (-a^2 + a - 1), Fractional ideal (a))
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))
```

(continues on next page)
The exp and log methods 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
-6*a^2 + 18*a - 54
sage: u.norm().factor()
-1 * 2^9 * 3^5
sage: U.log(u)
(1, 1, 4, 1, 5)
```

The `S_unit_solutions` function can be used to find all solutions to the $S$-unit equation $x + y = 1$ over $K$.

```
S_unit_solutions(S=[], prec=106, include_exponents=False, include_bound=False, proof=None)
```

Return all solutions to the $S$-unit equation $x + y = 1$ over $K$.

**INPUT:**

- $S$ – a list of finite primes in this number field
- `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)
- `proof` – if False, assume the GRH in computing the class group. Default is True.

**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 $(sols, bound)$ where `sols` is as above and `bound` is the bound used for the entries in the exponent vectors.

**EXAMPLES:**
```python
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, xi + 1), (-xi + 1, xi), (xi + 2, -xi - 1), (1/3*xi + 2/3, -1/3*xi + 1/3)]
```

You can get the exponent vectors:

```python
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:

```python
sage: solutions, bound = K.S_unit_solutions(S, prec=100, include_bound=True)
sage: bound
2
```

**`S_units(S, proof=True)`**

Returns 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 `S_unit_group()` function.

**EXAMPLES:**

```python
sage: K.<a> = QuadraticField(-3)
sage: K.unit_group()
Unit group with structure C6 of Number Field in a with defining polynomial x^2 + 3
sage: K.S_units([]) # random
[1/2*a + 1/2]
sage: K.S_units([])[0].multiplicative_order()
6
```

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

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

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

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

**absolute_degree()**

Return the degree of self over \( \mathbb{Q} \).

**EXAMPLES:**

```python
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
```

**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: NumberField(x^2 + (2/3)*x - 9/17,'a').absolute_polynomial_ntl()
([[-27 34 51], 51])
```

**algebraic_closure()**

Return the algebraic closure of self (which is \( \mathbb{Q}\bar{\mathbb{Q}} \)).

**EXAMPLES:**

```python
sage: K.<i> = QuadraticField(-1)
sage: K.algebraic_closure()
Algebraic Field
sage: K.<a> = NumberField(x^3-2)
sage: K.algebraic_closure()
Algebraic Field
sage: K = CyclotomicField(23)
```

(continues on next page)
change_generator\( (\alpha, \text{name}=\text{None}, \text{names}=\text{None}) \)

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
sage: 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
sage: from_K
Ring morphism:
  From: Number Field in i0 with defining polynomial x^2 - 6*x + 37/4
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i0 |--> 1/2*i + 3
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
  Defn: i |--> 2*i0 - 6
```

We can also do

```python
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
```

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

```python
sage: to_K(i)
2*c - 6
```

Note that the image is indeed a square root of \(-1\).

```python
sage: to_K(i)^2
-1
sage: from_K(to_K(i))
i
sage: to_K(from_K(c))
c
```

**characteristic()**

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

**EXAMPLES:**

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

**class_group\( (\text{proof}=\text{None}, \text{names}=\text{'}c\text{'}\)\)**

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: 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),)
sage: G.number_field()
Number Field in a with defining polynomial x^2 + 23
sage: G is K.class_group()
True
sage: G is K.class_group(proof=False)
False
sage: G.gens()
(Fractional ideal class (2, 1/2*a - 1/2),)
```

There can be multiple generators:

```
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
sage: G.0 # random
Fractional ideal class (41, a + 10)
sage: G.0^38
Trivial principal fractional ideal class
sage: G.1 # random
Fractional ideal class (2, -1/2*a)
sage: G.1^2
Trivial principal fractional ideal class
```

Class groups of Hecke polynomials tend to be very small:

```
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]
```

**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:**

1.1. Number Fields 43
```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
```

**completely_split_primes** (\(B=200\))

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

**completion** (\(p,\) **prec**, **extras**=\{\})

Returns the completion of self at \(p\) to the specified precision. Only implemented at archimedean places, and then only if an embedding has been fixed.

**EXAMPLES:**

```sage```
K.<a> = QuadraticField(2)
sage: K.completion(infinity, 100)
Real Field with 100 bits of precision
sage: K.<zeta> = CyclotomicField(12)
sage: K.completion(infinity, 53, extras={'type': 'RDF'})
Complex Double Field
sage: zeta + 1.5
2.36602540378444 + 0.500000000000000*I
```

**complex_conjugation**()

Return the complex conjugation of self.

This is only well-defined for fields contained in CM fields (i.e., 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```
QuadraticField(-1, 'I').complex_conjugation()
Ring endomorphism of Number Field in I with defining polynomial x^2 + 1
  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
sage: F = NumberField(x^4 + x^3 - 3*x^2 - x + 1, 'a')
sage: F.is_totally_real()
True
```

(continues on next page)
sage: F.complex_conjugation()
Identity endomorphism of Number Field in a with defining polynomial \(x^4 + x^3 - 3x^2 - x + 1\)

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

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:

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
]

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 \((F, \text{selfight}_\text{into}_F, \text{otheright}_\text{into}_F, k)\) such that \(\text{selfight}_\text{into}_F\) is an embedding of \(\text{self}\) in \(F\), \(\text{otheright}_\text{into}_F\) is an embedding of \(\text{other}\) in \(F\), and \(k\) is one of the following:

• an integer such that \(F\text{.gen()}\) equals \(\text{otheright}_\text{into}_F(\text{other\right}_\text{gen()}) + k\times\text{selfight}_\text{into}_F(\text{self\right}_\text{gen()})\);

• Infinity, in which case \(F\text{.gen()}\) equals \(\text{selfight}_\text{into}_F(\text{self\right}_\text{gen()})\);

• None (when \(\text{other}\) is a relative number field).

If both \(\text{self}\) and \(\text{other}\) have embeddings into an ambient field, then each \(F\) will have an embedding with respect to which both \(\text{selfight}_\text{into}_F\) and \(\text{otheright}_\text{into}_F\) will be compatible with the ambient embeddings.

If preserve_embedding is True and if \(\text{self}\) and \(\text{other}\) 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:

```python
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]
```

A particular compositum is selected, together with compatible maps into the compositum, if the fields are endowed with a real or complex embedding:

```python
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, both_maps=True); F
Number Field in a with defining polynomial x^4 - 2
sage: f(K1.0), g(K2.0)
(a, -a)
```

With preserve_embedding set to False, the embeddings are ignored:

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

Changing the embedding selects a different compositum:

```python
sage: K3 = NumberField(x^4 - 2, 'a', embedding=CC(2^(1/4)*I))
sage: [F, f, g, k] = K1.composite_fields(K3, both_maps=True); F
Number Field in a0 with defining polynomial x^8 + 28*x^4 + 2500
sage: f(K1.0), g(K3.0)
(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

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: B.<a> = NumberField(x^9 - 7, embedding=QQbar.polynomial_root(x^9 - 7, RIF(1.2, 1.3)))
sage: 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 \( \mathbb{Q} \), the output is not meaningful.

**INPUT:**

- check_abelian - a boolean (default: True); check to see that this is an abelian extension of \( \mathbb{Q} \)

**OUTPUT:**

Integer which is the conductor of the field.

**EXAMPLES:**

sage: K = CyclotomicField(27)
sage: k = K.subfields(9)[0][0]
sage: k.conductor()
27
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

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.

collection (
Construction of self

EXAMPLES:

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

The construction functor respects distinguished embeddings:

sage: F(R) is K
True
sage: F.embeddings
[0.2327856159383841? + 0.7925519925154479?*I]

sage: P.<x> = QQ[]
sage: K.<a> = NumberField(x^3-5, embedding=0)
sage: L.<b> = K.extension(x^2+a)
sage: a*b
a*b
defining_polynomial ()
Return the defining polynomial of this number field.

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

EXAMPLES:

sage: k5.<z> = CyclotomicField(5)
sage: k5.defining_polynomial()
x^4 + x^3 + x^2 + x + 1
sage: y = polygen(QQ, 'y')
sage: k.<a> = NumberField(y^9 - 3*y + 5); k
Number Field in a with defining polynomial y^9 - 3*y + 5
sage: k.defining_polynomial()
y^9 - 3*y + 5
degree ()
Return the degree of this number field.

EXAMPLES:
**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:**

```python
sage: k.<a> = NumberField(x^2 + 23)
sage: d = k.different()
sage: d
Fractional ideal (-a)
sage: d.norm()
23
sage: k.disc()
-23
```

The different is cached:

```python
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
```

**disc(v=None)**

Shortcut for self.discriminant.

**EXAMPLES:**

```python
sage: k.<b> = NumberField(x^2 - 123)
sage: k.disc()
492
```

**discriminant(v=None)**

Returns 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: K.<t> = NumberField(x^3 + x^2 - 2*x + 8)
sage: K.disc()
-503
sage: K.disc([1, t, t^2])
-2012
sage: K.disc([1/7, (1/5)*t, (1/3)*t^2])
-2012/11025
sage: (5*7*3)^2
11025
sage: NumberField(x^2 - 1/2, 'a').discriminant()
8
```

**elements_of_norm\((n, proof=None)\)**
Return a list of elements of norm \( n \).

**INPUT:**
- \( n \) – integer in this number field
- \( proof \) – boolean (default: True, unless you called number_field_proof and set it otherwise)

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

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

**extension\((poly, name=None, names=None, *args, **kwds)\)**
Return the relative extension of this field by a given polynomial.

**EXAMPLES:**
```
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 (trac ticket #20791):

```
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 \( x^2 + 1 \):

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

Here are the factors:

```
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:

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

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

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

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

```
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 (-3*a - 2))
```

An example over a relative number field:

```
sage: pari('setrand(2)')
sage: L.<b> = K.extension(x^2 - 7)
sage: f = L.factor(a + 1); f
(Fractional ideal (1/2*a*b - a + 1/2)) * (Fractional ideal (-1/2*a*b - a + 1/2))
```

(continues on next page)
It doesn’t make sense to factor the ideal (0), so this raises an error:

```sage
sage: L.factor(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 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
sage: K.<a> = NumberField(x^3 - 2)
sage: K.fractional_ideal([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
sage: K.fractional_ideal(K.ideal(a))
Fractional ideal (a)
sage: L.<b> = K.extension(x^2 - 3, x^2 + 1)
sage: M.<c> = L.extension(x^2 + 1)
sage: L.ideal(K.ideal(2, a))
Fractional ideal (a)
sage: M.ideal(K.ideal(2, a)) == M.ideal(a*(b - c)/2)
True
```

The zero ideal is not a fractional ideal!

```sage
sage: K.fractional_ideal(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)**

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

**INPUT:**

- `type` - none, gap, or pari. If None (the default), return an explicit group of automorphisms of self as a `GaloisGroup_v2` object. Otherwise, return a `GaloisGroup_v1` wrapper object based on a PARI or Gap transitive group object, which is quicker to compute, but rather less useful (in particular, it can’t be made to act on self).
• *algorithm* - ‘pari’, ‘kash’, ‘magma’. (default: ‘pari’, except when the degree is \( \geq 12 \) when ‘kash’ is tried.)

• *name* - a string giving a name for the generator of the Galois closure of self, when self is not Galois. This is ignored if type is not None.

Note that computing Galois groups as abstract groups is often much faster than computing them as explicit automorphism groups (but of course you get less information out!) For more (important!) documentation, so the documentation for Galois groups of polynomials over \( \mathbb{Q} \), e.g., by typing \( K.polynomial() \). galois_group?, where \( K \) is a number field.

To obtain actual field homomorphisms from the number field to its splitting field, use type=None.

**EXAMPLES:**

With type None:

```python
sage: k.<b> = NumberField(x^2 - 14) # a Galois extension
sage: G = k.galois_group(); G
Galois group of Number Field in b with defining polynomial 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)
```

```python
sage: k.<b> = NumberField(x^3 - x + 1) # not Galois
sage: G = k.galois_group(names='c'); G
Galois group of Galois closure in c of Number Field in b with defining polynomial x^3 - x + 1
sage: G.gen(0)
(1,2,3)(4,5,6)
```

With type 'pari':

```python
sage: NumberField(x^3-2, 'a').galois_group(type="pari")
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the Number Field in a
with defining polynomial x^3 - 2
```

```python
sage: NumberField(x-1, 'a').galois_group(type="gap")
Galois group Transitive group number 1 of degree 1 of the Number Field in a
with defining polynomial x - 1
```

```python
sage: NumberField(x^2+2, 'a').galois_group(type="gap")
Galois group Transitive group number 1 of degree 2 of the Number Field in a
with defining polynomial x^2 + 2
```

```python
sage: NumberField(x^3-2, 'a').galois_group(type="gap")
Galois group Transitive group number 2 of degree 3 of the Number Field in a
with defining polynomial x^3 - 2
```

```python
sage: x = polygen(QQ)
sage: NumberField(x^3 + 2*x + 1, 'a').galois_group(type='gap')
Galois group Transitive group number 2 of degree 3 of the Number Field in a
with defining polynomial x^3 + 2*x + 1
```

```python
sage: NumberField(x^3 + 2*x + 1, 'a').galois_group(algorithm='magma')  # optional - magma
Galois group Transitive group number 2 of degree 3 of the Number Field in a
with defining polynomial x^3 + 2*x + 1
```

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.

```sage
K.<a> = NumberField(x^3 - 2)
sage: L.<b1> = K.galois_closure(); L
Number Field in b1 with defining polynomial x^6 + 108
galois group
```

**gen (n=0)**

Return the generator for this number field.

**INPUT:**

- *n* - must be 0 (the default), or an exception is raised.

**EXAMPLES:**

```sage
k.<theta> = NumberField(x^14 + 2); k
sage: k.gen()
theta
sage: k.gen(1)
Traceback (most recent call last):
... IndexError: Only one generator.
```

**gen_embedding ()**

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

**EXAMPLES:**

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

**ideal (**gens**, **kws**)**

K.ideal() returns a fractional ideal of the field, except for the zero ideal which is not a fractional ideal.

**EXAMPLES:**

```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
```
ideals_of_bdd_norm(bound)
All integral ideals of bounded norm.

INPUT:
  • bound - a positive integer

OUTPUT: A dict of all integral ideals I such that Norm(I) <= bound, keyed by norm.

EXAMPLES:

```
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 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 (4, 1/2*a + 3/2)
Fractional ideal (2)
Fractional ideal (4, 1/2*a + 5/2)
5
Fractional ideal (1/2*a - 1/2)
Fractional ideal (6, 1/2*a + 5/2)
Fractional ideal (6, 1/2*a + 7/2)
Fractional ideal (1/2*a + 1/2)
7
Fractional ideal (1/2*a + 3/2)
Fractional ideal (4, a - 1)
Fractional ideal (4, a + 1)
Fractional ideal (1/2*a - 3/2)
9
Fractional ideal (9, 1/2*a + 11/2)
Fractional ideal (3)
Fractional ideal (9, 1/2*a + 7/2)
10
```

integral_basis(v=None)
Returns 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 self.maximal_order.

EXAMPLES:

```
sage: K.<a> = NumberField(x^5 + 10*x + 1)
sage: 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.

```
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]
```

ALGORITHM: Uses the pari library (via `_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:**

```
sage: Q.<a> = NumberField(x - 1)
sage: Q.is_CM()
False
sage: K.<i> = NumberField(x^2 + 1)
sage: K.is_CM()
True
sage: L.<zeta20> = CyclotomicField(20)
sage: L.is_CM()
True
sage: K.<omega> = QuadraticField(-3)
sage: K.is_CM()
True
sage: L.<sqrt5> = QuadraticField(5)
sage: L.is_CM()
False
sage: F.<a> = NumberField(x^3 - 2)
sage: F.is_CM()
False
sage: F.<a> = NumberField(x^4-x^3-3*x^2+x+1)
sage: F.is_CM()
False
```

The following are non-CM totally imaginary fields.

```
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
```

The following is a non-cyclotomic CM field.

```
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).
Finally, a CM field that is given as an extension that is not CM.

```
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_extension()
False
```

**is_abelian()**

Return True if this number field is an abelian Galois extension of \( \mathbb{Q} \).

**EXAMPLES:**

```
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()**

Returns True if self is an absolute field.

This function will be implemented in the derived classes.

**EXAMPLES:**

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

**is_field**(proof=True)

Return True since a number field is a field.

**EXAMPLES:**

```
sage: NumberField(x^5 + x + 3, 'c').is_field()
True
```

**is_galois()**

Return True if this number field is a Galois extension of \( \mathbb{Q} \).

**EXAMPLES:**
is_galois()
Return True if self is a Galois number field.

EXAMPLES:

sage: NumberField(x^2 + 1, 'i').is_galois()
True
sage: NumberField(x^3 + 2, 'a').is_galois()
False
sage: NumberField(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 - 10, 'a').is_galois()
False

is_isomorphic (other, isomorphism_maps=False)
Return True if self is isomorphic as a number field to other.

EXAMPLES:

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 = NumberField(x^2 - x - 1, 'b')
sage: ky.<y> = k[]
sage: l = NumberField(y, 'a')
sage: k.is_isomorphic(l, True)
(True, [-x, x + 1])

is_relative()
EXAMPLES:

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 con-
tain the real numbers.

EXAMPLES:

```python
sage: NumberField(x^2+2, 'alpha').is_totally_imaginary()
True
sage: NumberField(x^2-2, 'alpha').is_totally_imaginary()
False
sage: NumberField(x^4-2, 'alpha').is_totally_imaginary()
False
```

**is_totally_real()**

Return True if self is totally real, and False otherwise.

Totally real means that every isomorphic embedding of self into the complex numbers has image contained in the real numbers.

EXAMPLES:

```python
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
```

**latex_variable_name**(name=None)

Return the latex representation of the variable name for this number field.

EXAMPLES:

```python
sage: NumberField(x^2 + 3, 'a').latex_variable_name() 'a'
sage: NumberField(x^3 + 3, 'theta3').latex_variable_name() '\theta_{3}'
sage: CyclotomicField(5).latex_variable_name() '\zeta_{5}'
```

**maximal_totally_real_subfield()**

Return the maximal totally real subfield of self together with an embedding of it into self.

EXAMPLES:

```python
sage: F.<a> = QuadraticField(11)
sage: F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^2 - 11, Identity endomorphism of Number Field in a with defining polynomial x^2 - 11]
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]
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, 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
  To:  Number Field in a with defining polynomial x^2 + 15]
```

(continues on next page)
To: Cyclotomic Field of order 29 and degree 28
Defn: \( a_0 \mapsto -a^{27} - a^{26} - a^{25} - a^{24} - a^{23} - a^{22} - a^{21} - a^{20} - a^{19} \)
\( \cdots a^{18} - a^{17} - a^{16} - a^{15} - a^{14} - a^{13} - a^{12} - a^{11} - a^{10} - a^9 - a^8 - a^7 - a^6 - a^5 - a^4 - a^3 - a^2 - 1 \)
sage: 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 - x^3 + 2*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 - 2*x^3 + x^2 + 6*x + 3
        Defn: a1 \mapsto -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 \mapsto 1]

An example of a relative extension where the base field is not the maximal totally real subfield.

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^4 - 2*x^3 + x^2 + 6*x + 3
        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 \mapsto -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: 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:

```python
sage: NumberField(x^2 + 17,'a').ngens()
1
sage: NumberField(x + 3,'a').ngens()
1
sage: k.<a> = NumberField(x + 3)
sage: k.ngens()
1
sage: k.0
-3
```

**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:

```python
sage: F.<alpha> = NumberField(x**22+3)
sage: F.zeta_order()
6
sage: F.<alpha> = NumberField(x**2-7)
sage: F.zeta_order()
2
```

**order()**

Return the order of this number field (always +infinity).

OUTPUT: always positive infinity

EXAMPLES:

```python
sage: NumberField(x^2 + 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: 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), [...]]

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)
pari_nf(important=True)
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: 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]
```

With important=False, we simply bail out if we cannot easily factor the discriminant:
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:

\begin{verbatim}
sage: K.<a> = NumberField(x^2 - p*q, maximize_at_primes=[p])
sage: K.pari_nf()
y^2 - 100000000000000000000...
\end{verbatim}

Since the discriminant is square-free, this also works:

\begin{verbatim}
sage: K.<a> = NumberField(x^2 - p*q, assume_disc_small=True)
sage: K.pari_nf()
y^2 - 100000000000000000000...
\end{verbatim}

\texttt{pari\_polynomial}(name='x')

Return the PARI polynomial corresponding to this number field.

\textbf{INPUT:}

- \texttt{name} = variable name (default: 'x')

\textbf{OUTPUT:}

A monic polynomial with integral coefficients (PARI \texttt{t\_POL}) defining the PARI number field corresponding to \texttt{self}.

\textbf{Warning:} This is \emph{not} the same as simply converting the defining polynomial to PARI.

\textbf{EXAMPLES:}

\begin{verbatim}
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.pari_polynomial('a')
a^2 - a + 40
\end{verbatim}

Some examples with relative number fields:

\begin{verbatim}
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
sage: k.absolute_polynomial()
x^4 + 8*x^2 + 4
sage: k.relative_polynomial()
\end{verbatim}
\begin{verbatim}
x^2 + 3
sage: k.<a, c> = NumberField([x^2 + 1/3, x^2 + 1/4])
sage: k.pari_polynomial()
x^4 - x^2 + 1
sage: k.absolute_polynomial()
x^4 - x^2 + 1

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
\end{verbatim}

\textbf{pari_rnfnorm_data}\hspace{1em}(L, proof=True)

Return the PARI \texttt{pari:rnfisnorminit} data corresponding to the extension \texttt{L/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}

\textbf{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: 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]
\end{verbatim}

\textbf{polynomial()}

Return the defining polynomial of this number field.

This is exactly the same as \texttt{self.defining_polynomial().}

\textbf{EXAMPLES:}
sage: NumberField(x^2 + (2/3)*x - 9/17,'a').polynomial()

x^2 + 2/3*x - 9/17

**polynomial_n tl ()**

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: NumberField(x^2 + (2/3)*x - 9/17,'a').polynomial_n tl ()

([-27 34 51], 51)

**polynomial_quotient_ring ()**

Return the polynomial quotient ring isomorphic to this number field.

**EXAMPLES:**

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

**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: 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

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

**power_b a sis ()**

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: K.<a> = NumberField(x^5 + 10*x + 1)
sage: K.power_basis()

[1, a, a^2, a^3, a^4]

sage: L.<b> = K.extension(x^2 - 2)
sage: L.power_basis()

[1, b]

(continues on next page)
prime_above \( (x, \text{degree} = \text{None}) \)

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 \( x = 0 \).
- \( \text{degree} \) (default: \text{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 \text{ValueError}.

EXAMPLES:

- \( x = \mathbb{Z}['x'].\text{gen()} \)
- \( F.<t> = \text{NumberField}(x^3 - 2) \)

\begin{verbatim}
sage: P2 = F.prime_above(2)
sage: P2 # random
Fractional ideal (-t)
sage: 2 in P2
True
sage: P2.is_prime()
True
sage: P2.norm()
2

sage: P3 = F.prime_above(3)
sage: P3 # random
Fractional ideal (t + 1)
sage: 3 in P3
True
sage: P3.is_prime()
True
sage: P3.norm()
3
\end{verbatim}

The ideal \( (3) \) is totally ramified in \( F \), so there is no degree 2 prime above \( 3 \):

\begin{verbatim}
sage: F.prime_above(3, \text{degree}=2)
Traceback (most recent call last):
...
ValueError: No prime of degree 2 above Fractional ideal (3)
sage: [id.residue_class_degree() \text{ for id, _ in } F.ideal(3).factor() ]
[1]
\end{verbatim}

Asking for a specific degree works:
Relative number fields are okay:

```python
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):

```python
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:**

```python
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, degree=None)**  
Return prime ideals 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 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 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)
```
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
```

Works in relative extensions too:

```
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=1) == [P]
True
sage: K.primes_above(I, degree=4) == [Q]
True
```

It doesn’t make sense to factor the ideal $(0)$, so this raises an error:
sage: F.prime_above(0)
Traceback (most recent call last):
...
AttributeError: 'NumberFieldIdeal' object has no attribute 'prime_factors'

primes_of_bounded_norm(B)
Returns 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: 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]
```

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:
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)]
sage: list(K.primes_of_bounded_norm_iter(1))
[]

\texttt{primes\_of\_degree\_one\_iter} \ (\text{num\_integer\_primes}=10000, \text{max\_iterations}=100)

Return an iterator yielding 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 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:}

\begin{itemize}
  \item \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.
  \item \texttt{max\_iterations} (default: 100) - an integer. We test \texttt{max\_iterations} integers to find small primes before raising StopIteration.
\end{itemize}

\textbf{EXAMPLES:}

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),
  Fractional ideal (z^3 - 3*z^2 + 2) ]
sage: [ P.norm() for P in Ps ] # random
[11, 31, 41]
sage: [ P.residue_class_degree() for P in Ps ]
[1, 1, 1]

\texttt{primes\_of\_degree\_one\_list} \ (n, \text{num\_integer\_primes}=10000, \text{max\_iterations}=100)

Return a list of \texttt{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 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:}

\begin{itemize}
  \item \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.
  \item \texttt{max\_iterations} (default: 100) - an integer. We test \texttt{max\_iterations} integers to find small primes before raising StopIteration.
\end{itemize}
EXAMPLES:

```sage
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 (2*z^3 - 2*z^2 + 2*z - 3), Fractional ideal (2*z^3 - 3*z^2 + z - 2)]
sage: [ P.norm() for P in Ps ]
[11, 31, 41]
sage: [ 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:

```sage
sage: K.<a> = NumberField(x^3 + 2)
sage: K.primitive_element()
a
sage: K.<a,b,c> = NumberField([x^2-2,x^2-3,x^2-5])
sage: K.primitive_element()
a - b + c
sage: alpha = K.primitive_element(); alpha
a - b + c
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
```

**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:

```sage
sage: K.<i> = NumberField(x^2+1)
sage: z = K.primitive_root_of_unity(); z
i
sage: z.multiplicative_order()
4
sage: K.<a> = NumberField(x^2+x+1)
sage: z = K.primitive_root_of_unity(); z
a + 1
sage: z.multiplicative_order()
6
sage: x = polygen(QQ)
sage: F.<a,b> = NumberField([x^2 - 2, x^2 - 3])
sage: y = polygen(F)
sage: K.<c> = F.extension(y^2 - (1 + a)*(a + b)*a+b)
```
We do not special-case cyclotomic fields, so we do not always get the most obvious primitive root of unity:

```python
sage: K.<a> = CyclotomicField(3)
sage: z = K.primitive_root_of_unity(); z
a + 1
sage: 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:**

```python
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
```

### random_element \((\text{num\_bound}=\text{None}, \ \text{den\_bound}=\text{None}, \ \text{integral\_coefficients}=\text{False}, \ \text{distribution}=\text{None})\)
Return a random element of this number field.

**INPUT:**

- \(\text{num\_bound}\) - Bound on numerator of the coefficients of the resulting element
- \(\text{den\_bound}\) - Bound on denominators of the coefficients of the resulting element
- \(\text{integral\_coefficients}\) (default: False) - If True, then the resulting element will have integral coefficients. This option overrides any value of \(\text{den\_bound}\).
- \(\text{distribution}\) - Distribution to use for the coefficients of the resulting element
OUTPUT:

- Element of this number field

EXAMPLES:

```python
sage: K.<j> = NumberField(x^8+1)
sage: K.random_element()
1/2*j^7 - j^6 - 12*j^5 + 1/2*j^4 - 1/95*j^3 - 1/2*j^2 - 4

sage: K.<a,b,c> = NumberField([x^2-2,x^2-3,x^2-5])
sage: K.random_element()
((6136*c - 7489/3)*b + 5825/3*c - 71422/3)*a + (-4849/3*c + 58918/3)*b -
˓→45718/3*c + 75409/12

sage: K.<a> = NumberField(x^5-2)
sage: K.random_element(integral_coefficients=True)
a^3 + a^2 - 3*a - 1
```

**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:

```python
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 |--> -1.259921049894873
]

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 |--> -1.2599210498948731647672106073
]
```

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
```

**reduced\_basis\(\text{(prec=}\text{None})\)**

This function returns an LLL-reduced basis for the Minkowski-embedding of the maximal order of a number field.

**INPUT:**

- **self** - number field, the base field
- **prec** (default: 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.

**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. The matrix returned will always be integral; however, it may only be only “almost” LLL-reduced when the precision is not sufficiently high.

**EXAMPLES:**

```python
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, 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]
```

**reduced\_gram\_matrix\(\text{(prec=}\text{None})\)**

This function returns the Gram matrix of an LLL-reduced basis for the Minkowski embedding of the maximal order of a number field.

**INPUT:**

- **self** - number field, the base field
- **prec** (default: None) - the precision with which to calculate the Minkowski embedding.

(See NOTE below.)

**OUTPUT:** The Gram matrix \(\langle x_i, x_j \rangle\) of an LLL reduced basis for the maximal order of self, where the integral basis for self is given by \(\{x_0, \ldots, x_{n-1}\}\). Here \(\langle , \rangle\) is the usual inner product on \(\mathbb{R}^n\), and self is embedded in \(\mathbb{R}^n\) by the Minkowski embedding. See the docstring for `NumberField_absolute.minkowski_embedding()` for more information.
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 flag above.

If the following run-time error occurs: “PariError: not a definite matrix in lllgram (42)” try increasing the prec parameter.

EXAMPLES:

```
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 6 16 -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]
sage: x = polygen(QQ)
sage: F.<alpha> = NumberField(x^4+x^2+712312*x+131001238)
sage: F.reduced_gram_matrix(prec=128)
[ 4.0000000000000000000000000000000000000 0.
˓→0.0000000000000000000000000000000000000 -1.0000000000000000000000000000000000000 0.0000000000000000000000000000000000000 -1.0000000000000000000000000000000000000]
[ 0.0000000000000000000000000000000000000 4672.1539331563218381658483353092335550 -1148.839100265517242751227497036149666768 -418.9212718083977141198754424579680468382]
[ -1.9999999999999999999999999999999999999 -1148.839100265517242751227497036149666768 5.068151310500611768713076521847709187e8 1.4179092271494070050433368847682152174e8]
[ -0.9999999999999999999999999999999999999 -418.9212718083977141198754424579680468382 1.4179092271494070050433368847682152174e8 1.3658972679198113788411201405279175e12]
```

**regulator** *(proof=\text{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:**

1.1. Number Fields 75

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
sage: K.<i> = NumberField(x^2 + 1)
sage: K.residue_field(1+i)
Residue field of Fractional ideal (i + 1)
```

```python
sage: L.residue_field(L.prime_above(5)^2)
Traceback (most recent call last):
  ... ValueError: Fractional ideal (5) is not a prime ideal
```

roots_of_unity()
Return all the roots of unity in this field, primitive or not.

EXAMPLES:

```python
sage: K.<b> = NumberField(x^2+1)
sage: zs = K.roots_of_unity(); zs
[b, -1, -b, 1]
sage: [ z**K.number_of_roots_of_unity() for z in zs ]
[1, 1, 1, 1]
```

selmer_group (S, m, proof=True, orders=False)
Compute 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 $K^×/(K^×)^m$. This is the subgroup of $K^×/(K^×)^m$ 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:

$$1 \rightarrow O_{K,S}^×/(O_{K,S}^×)^m \rightarrow K(S,m) \rightarrow Cl_{K,S}[m] \rightarrow 0.$$ 

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 $S$.

EXAMPLES:

```
sage: K.<a> = QuadraticField(-5)
sage: K.selmer_group((,), 2)
[-1, 2]
```

The previous example shows that the group generated by the output may be strictly larger than the ‘true’ Selmer group of elements giving extensions unramified outside $S$, since that has order just 2, generated by $-1$:

```
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_group((,), 2, orders=True)
([-1, 2], [2, 2])
sage: K.selmer_group((,), 4, orders=True)
([-1, 4], [2, 2])
sage: K.selmer_group([P2, 2]
[2, -1]
sage: K.selmer_group([P2, P3], 4)
[2, -a - 1, -1]
sage: K.selmer_group([P2, P3], 4, orders=True)
([2, -a - 1, -1], [4, 4, 2])
sage: K.selmer_group([P2], 3)
[2]
sage: K.selmer_group([P2, P3], 3)
[2, -a - 1]
sage: K.selmer_group([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_group([],5)
[]
sage: K.selmer_group([K.prime_above(p) for p in [2,3,5]],2)
[2, 3, 5, -1]
sage: K.selmer_group([K.prime_above(p) for p in [2,3,5]],6, orders=True)
([2, 3, 5, -1], [6, 6, 6, 2])
```

1.1. Number Fields
**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_group()` for more information.

**EXAMPLES:**

```python
sage: K.<a> = 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
sage: K.<a> = NumberField(polygen(QQ))
sage: list(K.selmer_group_iterator([], 5))
[1]
sage: list(K.selmer_group_iterator([], 4))
[1, -1]
sage: list(K.selmer_group_iterator(([K.prime_above(p) for p in [11,13]],2)))
[1, -1, 13, -13, 11, -11, 143, -143]
```

**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:**

```python
sage: NumberField(x^2+1, 'a').signature()
(0, 1)
sage: NumberField(x^3-2, 'a').signature()
(1, 1)
```

**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 \(x-reslist[i]\) is in \(Ilist[i]\) for all \(i\).
Note: The current implementation requires the ideals to be pairwise coprime. A more general version would be possible.

**EXAMPLES:**

```python
sage: 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
sage: Ilist[0]+Ilist[1]
Fractional ideal (2, a)
```

**some_elements()**

Return a list of elements in the given number field.

**EXAMPLES:**

```python
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[]
sage: M.<b> = K.extension(t^3 - 5); M
Number Field in b with defining polynomial t^3 - 5 over its base field
sage: 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()**

Returns 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:**

```python
sage: QuadraticField(-1, 'I').specified_complex_embedding()
Generic morphism:
  From: Number Field in I with defining polynomial x^2 + 1
  To:   Complex Lazy Field
  Defn: I -> 1*I
sage: QuadraticField(3, 'a').specified_complex_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^2 - 3
  To:   Real Lazy Field
  Defn: a -> 1.732050807568878?
```
sage: CyclotomicField(13).specified_complex_embedding()
Generic morphism:
  From: Cyclotomic Field of order 13 and degree 12
  To:   Complex Lazy Field
  Defn: zeta13 -> 0.885456025653210? + 0.464723172043769?*I

Most fields don’t implicitly have embeddings unless explicitly specified:

sage: NumberField(x^2-2, 'a').specified_complex_embedding() is None
True
sage: NumberField(x^3-x+5, 'a').specified_complex_embedding() is None
True
sage: NumberField(x^3-x+5, 'a', embedding=2).specified_complex_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 5
  To:   Real Lazy Field
  Defn: a -> -1.904160859134921?
sage: NumberField(x^3-x+5, 'a', embedding=CDF.0).specified_complex_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^3 - x + 5
  To:   Complex Lazy Field
  Defn: a -> 0.952080429567461? + 1.311248044077123?*I

This function only returns complex embeddings:

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 + 6*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
  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 + 6*7^11 + 7^12 + 7^13 + 2*7^15 + 7^16 + 7^17 + 4*7^18 + 6*7^19 + O(7^20)

structure()  
Return fixed isomorphism or embedding structure on self.

This is used to record various isomorphisms or embeddings that arise naturally in other constructions.

EXAMPLES:

sage: K.<a> = 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

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:

```sage
sage: K.<a> = NumberField(x^4 - 3); K
Number Field in a with defining polynomial x^4 - 3
sage: H.<b>, from_H = K.subfield(a^2)
```

A relative example. Note that the result returned is the subfield generated by \(\alpha\) over self. \(\text{base_field}()\), not over \(\mathbb{Q}\) (see trac ticket #5392):

```sage
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:

```sage
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
sage: CLF_from_K = K.coerce_embedding(); CLF_from_K
Generic morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Lazy Field
  Defn: z |--> 0.309016994374948? + 0.951056516295154?*I
sage: CLF_from_L = L.coerce_embedding(); CLF_from_L
Generic morphism:
  From: Number Field in z0 with defining polynomial x^2 - 5
  To:   Complex Field with 53 bits of precision
  Defn: z0 |--> 0.309016994374948? + 0.951056516295154?*I
```

(continues on next page)
To: Complex Lazy Field
Defn: z0 -> 2.236067977499790?

Check transitivity:

```
sage: CLF_from_L(L.gen())
2.236067977499790?
sage: CLF_from_K(K_from_L(L.gen()))
2.236067977499790? + 0.?e-14*I
```

If `self` has no specified embedding, then `K` comes with an embedding in `self`:

```
sage: K.<a> = NumberField(x^6 - 6*x^4 + 8*x^2 - 1)
sage: L.<b>, from_L = K.subfield(a^2)
sage: L
Number Field in b with defining polynomial x^3 - 6*x^2 + 8*x - 1
sage: L.gen_embedding()
a^2
```

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 `self.change_generator`, which gives isomorphisms in both directions.

### `trace_dual_basis(b)`

Compute the dual basis of a basis of `self` with respect to the trace pairing.

**EXAMPLES:**

```
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]
```

### `trace_pairing(v)`

Return the matrix of the trace pairing on the elements of the list `v`.

**EXAMPLES:**

```
sage: K.<zeta3> = NumberField(x^2 + 3)
sage: K.trace_pairing([1, zeta3])
[ 2  0]
[ 0 -6]
```

### `uniformizer(P, 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`
- `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`.  

---

82 Chapter 1. Algebraic Number Fields
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: 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))
```

```python
sage: K = CyclotomicField(9)
sage: Plist = K.ideal(17).prime_factors()
sage: pilist = [K.uniformizer(P) for P in Plist]
[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]
```

```
```

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:bnfunit command.

INPUT:

• proof (bool, default True) flag passed to pari.

Note: The group is cached.
**EXAMPLES:**

```
sage: x = QQ['x'].0
sage: A = x^4 - 10*x^3 + 20*5*x^2 - 15*5^2*x + 11*5^3
sage: K = NumberField(A, 'a')
sage: 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
sage: U.gens()
(u0, u1)
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]
sage: U.invariants()
(10, 0)
sage: [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^15 - a^12 + a^10 - a^9 - 2*a^8 + 3*a^7 + a^6 - 3*a^5 + a^...
```

```
units(proof=None)
```

Return generators for the unit group modulo torsion.

**ALGORITHM:** Uses PARI’s pari:bnfunit command.

**INPUT:**

- `proof` (bool, default True) flag passed to pari.

**Note:** For more functionality see the `unit_group()` function.

See also:
unit_group() S_unit_group() S_units()

EXAMPLES:

```
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()
(8/275*a^3 - 12/55*a^2 + 15/11*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^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,
a^16 - a^15 + a^14 - a^12 + a^11 - a^10 - a^9 + 2*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,
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^16 - a^15 - 3*a^14 - 4*a^13 - 4*a^12 - 3*a^11 - a^10 + 2*a^9 + 4*a^8 + 5*a^7 - 7 + 4*a^6 + 2*a^5 - 2*a^4 - 6*a^3 - 9*a^2 - 9*a - 7,
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,
5*a^16 - 6*a^14 + a^13 + 7*a^12 - 2*a^11 - 7*a^10 + 4*a^9 + 7*a^8 - 6*a^7 - 7*a^6 + 8*a^5 + 6*a^4 - 11*a^3 - 5*a^2 + 13*a + 4)
```

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: K.<a> = NumberField(x^2 + 1)
sage: K valuation(2)
2-adic valuation
```

It can also be unramified in R:

```
sage: K valuation(3)
3-adic valuation
```

A prime that factors into pairwise distinct factors, results in an error:

```
sage: K 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
```

(continues on next page)
The valuation can also be selected by giving a valuation on the base ring that extends uniquely:

```
sage: CyclotomicField(5).valuation(ZZ.valuation(5))
5-adic valuation
```

When the extension is not unique, this does not work:

```
sage: K.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
```

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:

```
sage: R.<x> = QQ[]
sage: v = K.valuation(GaussValuation(R, QQ.valuation(5)).augmentation(x + 2, infinity))
sage: w = K.valuation(GaussValuation(R, QQ.valuation(5)).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:

```
sage: ww = K.valuation(GaussValuation(R, QQ.valuation(5)).augmentation(x + 3, infinity))
sage: w is ww
True
```

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:

```
sage: v = GaussValuation(R, QQ.valuation(5)).augmentation(x + 3, infinity)
sage: 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:

```
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
• \(all\) - bool. 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 `self.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
code
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):
  ... 
ValueError: There are no 4th roots of unity in self.
```

```python
code
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.zeta(3, all=True)
[]
```

Number fields defined by non-monic and non-integral polynomials are supported (trac ticket #252):

```python
code
sage: K.<a> = NumberField(1/2*x^2 + 1/6)
sage: K.zeta(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:**
**zeta_function** \( (\text{prec}=53, \text{max\_imaginary\_part}=0, \text{max\_asymp\_coeffs}=40) \)

Return the Zeta function of this number field.

This actually returns an interface to Tim Dokchitser’s program for computing with the Dedekind zeta function \( \zeta_F(s) \) of the number field \( F \).

**INPUT:**

- \text{prec} - integer (bits precision)
- \text{max\_imaginary\_part} - real number
- \text{max\_asymp\_coeffs} - integer

**OUTPUT:** The zeta function of this number field.

**EXAMPLES:**

```python
sage: K.<a> = NumberField(ZZ['x'].0^2+ZZ['x'].0-1)
sage: Z = K.zeta_function()
sage: Z
Zeta function associated to Number Field in a with defining polynomial x^2 + x - 1
sage: Z(-1)
0.0333333333333333
sage: L.<a, b, c> = NumberField([x^2 - 5, x^2 + 3, x^2 + 1])
sage: Z = L.zeta_function()
sage: Z(5)
1.00199015670185
```

**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:**

```python
sage: F.<alpha> = NumberField(x**22+3)
sage: F.zeta_order()
6
sage: F.<alpha> = NumberField(x**2-7)
sage: F.zeta_order()
2
```

sage.rings.number_field.number_field.NumberField_generic_v1\( (\text{poly}, \text{name}, \text{latex\_name}, \text{canonical\_embedding}=\text{None}) \)

Used for unpickling old pickles.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.number_field import NumberField_absolute_v1
sage: R.<x> = QQ[]
```
Create a quadratic extension of the rational field.

The command `QuadraticField(a)` creates the field $\mathbb{Q}(\sqrt{a})$.

**EXAMPLES:**

```sage
sage: QuadraticField(3, 'a')
Number Field in a with defining polynomial x^2 - 3
sage: QuadraticField(-4, 'b')
Number Field in b with defining polynomial x^2 + 4
```

**class_number**(proof=None)

Return the size of the class group of self.

If `proof = 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:**

```sage
sage: QuadraticField(-23,'a').class_number() 3
```

These are all the primes so that the class number of $\mathbb{Q}(\sqrt{-p})$ is 1:

```sage
sage: [d for d in prime_range(2,300) 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:

```
sage: len([d for d in range(2,200) if not is_square(d) and QuadraticField(d,'a →').class_number() == 1])
121
```

discriminant\(^\text{(v=None)}\)

Returns 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 element of this number field

**OUTPUT:** Integer if \(v\) is omitted, and Rational otherwise.

**EXAMPLES:**

```
sage: K.<i> = NumberField(x^2+1)
sage: K.discriminant()
-4
sage: K.<a> = NumberField(x^2+5)
sage: K.discriminant()
-20
sage: K.<a> = NumberField(x^2-5)
sage: K.discriminant()
5
```

hilbert_class_field\(^\text{(names)}\)

Returns 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 hilbert_class_field_defining_polynomial command, which is vastly faster than this command, since it doesn’t construct a relative extension.

**EXAMPLES:**

```
sage: K.<a> = NumberField(x^2 + 23)
sage: L = K.hilbert_class_field('b'); L
Number Field in b with defining polynomial x^3 - x^2 + 1 over its base field
sage: L.absolute_field('c')
Number Field in c with defining polynomial x^6 - 2*x^5 + 70*x^4 - 90*x^3 + 1631*x^2 - 1196*x + 12743
sage: K.hilbert_class_field_defining_polynomial()
x^3 - x^2 + 1
```

hilbert_class_field_defining_polynomial\(^\text{(name='x')}\)

Returns 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
```
Note that this polynomial is not the actual Hilbert class polynomial: see `hilbert_class_polynomial`:

```
sage: K.hilbert_class_polynomial()
x^3 + 3491750*x^2 - 5151296875*x + 12771880859375
```

```
sage: K.<a> = QuadraticField(-431)
sage: K.class_number()
21
sage: K.hilbert_class_field_defining_polynomial(name='z')
z^21 + 6*z^20 + 9*z^19 - 4*z^18 + 33*z^17 + 140*z^16 + 220*z^15 + 243*z^14 +
 -> 297*z^13 + 461*z^12 + 658*z^11 + 743*z^10 + 722*z^9 + 681*z^8 + 619*z^7 +
 -> 522*z^6 + 405*z^5 + 261*z^4 + 119*z^3 + 35*z^2 + 7*z + 1
```

`hilbert_class_polynomial(name='x')`

Compute the Hilbert class polynomial of this quadratic field.

Right now, this is only implemented for imaginary quadratic fields.

**EXAMPLES:**

```
sage: K.<a> = QuadraticField(-3)
sage: K.hilbert_class_polynomial()
x
sage: K.<a> = QuadraticField(-31)
sage: K.hilbert_class_polynomial(name='z')
z^3 + 39491307*z^2 - 58682638134*z + 1566028350940383
```

`is_galois()`

Return True since all quadratic fields are automatically Galois.

**EXAMPLES:**

```
sage: QuadraticField(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:**

```
sage: QF = QuadraticField
sage: [QF(d).number_of_roots_of_unity() for d in range(-7, -2)]
[2, 2, 2, 4, 6]
```

`sage.rings.number_field.number_field.NumberField_quadratic_v1(poly, name, canonical_embedding=None)`

Used for unpickling old pickles.

**EXAMPLES:**

```
sage: from sage.rings.number_field.number_field import NumberField_quadratic_v1
sage: R.<x> = QQ[]
sage: NumberField_quadratic_v1(x^2 - 2, 'd')
Number Field in d with defining polynomial x^2 - 2
```
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
sage: K.<theta> = QuadraticField(3); K
Number Field in theta with defining polynomial x^2 - 3
sage: RR(theta)
1.73205080756888
sage: QuadraticField(9, 'a')
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
```

Quadratic number fields derive from general number fields.

```python
sage: from sage.rings.number_field.number_field import is_NumberField
sage: type(K)
<class 'sage.rings.number_field.number_field.NumberField_quadratic_with_category'>
sage: is_NumberField(K)
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}
sage: K.latex_variable_name()
'\sqrt{-7}'
```

92 Chapter 1. Algebraic Number Fields
We can provide our own name as well:

```
sage: K.<a> = QuadraticField(next_prime(10^10), latex_name=r'\sqrt{D}')
sage: 1+a
a + 1
sage: latex(1+a)
\sqrt{D} + 1
sage: latex(QuadraticField(-1, 'a', latex_name=None).gen())
a
```

The name of the generator does not interfere with Sage preparser, see trac ticket #1135:

```
sage: K1 = QuadraticField(5, 'x')
sage: K2.<x> = QuadraticField(5)
sage: K3.<x> = QuadraticField(5, 'x')
sage: K1 is K2
True
sage: K1 is K3
True
sage: K1
Number Field in x with defining polynomial x^2 - 5
```

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
sage: 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: is_AbsoluteNumberField(NumberField(x^2 + 1,'a'))
True
sage: is_AbsoluteNumberField(NumberField([x^3 + 17, x^2+1],'a'))
False
```

The rationals are a number field, but they’re not of the absolute number field class.

```
sage: is_AbsoluteNumberField(QQ)
False
```

```
sage.rings.number_field.number_field.is_CyclotomicField(x)
Return True if x is a cyclotomic field, i.e., of the special cyclotomic field class. This function does not return True for a number field that just happens to be isomorphic to a cyclotomic field.

EXAMPLES:
sage: from sage.rings.number_field.number_field import is_CyclotomicField
sage: is_CyclotomicField(NumberField(x^2 + 1,'zeta4'))
False
sage: is_CyclotomicField(CyclotomicField(4))
True
sage: is_CyclotomicField(CyclotomicField(1))
True
```


```sage
sage: is_CyclotomicField(QQ)
False
sage: is_CyclotomicField(7)
False
```

```sage
sage.rings.number_field.number_field.is_NumberFieldHomsetCodomain(codomain)
```

Returns 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.morphism.NumberFieldHomset`.

**EXAMPLES:**

This currently accepts any parent (CC, RR,...) in `Fields`:

```sage
sage: from sage.rings.number_field.number_field import is_
 sage: is_NumberFieldHomsetCodomain(QQ)
True
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
```

**Question:** should, for example, QQ-algebras be accepted as well?

**Caveat:** Gap objects are not (yet) in `Fields`, and therefore not accepted as number field homset codomains:

```sage
sage: is_NumberFieldHomsetCodomain(gap.Rationals)
False
```

```sage
sage.rings.number_field.number_field.is_QuadraticField(x)
```

Return True if `x` is of the quadratic number field type.

**EXAMPLES:**

```sage
sage: from sage.rings.number_field.number_field import is_QuadraticField
sage: is_QuadraticField(QuadraticField(5,'a'))
True
sage: is_QuadraticField(NumberField(x^2 - 5, 'b'))
True
sage: is_QuadraticField(NumberField(x^3 - 5, 'b'))
False
```

A quadratic field specially refers to a number field, not a finite field:

```sage
sage: is_QuadraticField(GF(9,'a'))
False
```

```sage
sage.rings.number_field.number_field.is_fundamental_discriminant(D)
```

Return True if the integer `D` is a fundamental discriminant, i.e., if $D \equiv 0, 1 \pmod{4}$, 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.
EXAMPLES:

```python
sage: [D for D in range(-15,15) if is_fundamental_discriminant(D)]
[-15, -11, -8, -7, -4, -3, 5, 8, 12, 13]
sage: [D for D in range(-15,15) if not is_square(D) and QuadraticField(D,'a').disc() == D]
[-15, -11, -8, -7, -4, -3, 5, 8, 12, 13]
```

```
sage.rings.number_field.number_field.is_real_place(v)
Return True if v is real, False if v is complex

INPUT:

- v – an infinite place of K

OUTPUT:

A boolean indicating whether a place is real (True) or complex (False).

EXAMPLES:

```python
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
```

It is an error to put in a finite place

```python
sage: 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.

Returns t if t is not None, otherwise returns 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'
```

```
sage.rings.number_field.number_field.put_natural_embedding_first(v)
Helper function for embeddings() functions for number fields.

INPUT: a list of embeddings of a number field
```

1.1. Number Fields 95
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
sage: permuted_embs = list(embs); permuted_embs.remove(id); permuted_embs.
  stuff removed
  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
```

```python
sage.rings.number_field.number_field.refine_embedding(e, prec=None)
```

Given an embedding from a number field to either \( \mathbb{R} \) or \( \mathbb{C} \), returns an equivalent embedding with higher precision.

**INPUT:**

- \( e \) - an embedding of a number field into either \( \mathbb{R} \) or \( \mathbb{C} \) (with some precision)
- \( \text{prec} \) - (default None) the desired precision; if None, current precision is doubled; if Infinity, the equivalent embedding into either \( \mathbb{Q} \bar{\mathbb{Q}} \) or \( \mathbb{A} \) is returned.

**EXAMPLES:**

```python
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, Infinity); e25
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: e25.codomain().precision()
25
```

An example where we extend a real embedding into \( \mathbb{A} \):

```python
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:
```
Now we can obtain arbitrary precision values with no trouble:

```
sage: RealField(150)(e(a))
1.2599210498948731647672106072782283505702515
sage: _^3
2.000000000000000000000000000000000000000000
sage: RealField(200)(e(a^2-3*a+7))
4.80763790235799804500738174376232086807389337953290695624
```

Complex embeddings can be extended into `QQbar`:

```
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 |--> -0.6299605249474365... - 1.0911236597172*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 |--> -0.6299605249474365? - 1.09112365971722?*I
sage: ComplexField(200)(e(a))
-0.629960524947436582386053063911417528512573235075399004099 - 1.09112365971721403560072614898088132587338740309407036*I
sage: e(a)^3
2
```

Embeddings into lazy fields work:

```
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: zeta7 |--> 0.623489801858734? + 0.781831482468030?*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: zeta7 |--> 0.6234898018587335305250048840042398106322747308964021053655494390968536524564872845759425074575942507...
+ 0.
7818314824680298087084454266740577502323345187086875289806349580450917316339364412700868007*I
sage: refine_embedding(x, infinity)
Ring morphism:
From: Cyclotomic Field of order 7 and degree 6
To:    Algebraic Field
Defn: zeta7 |--> 0.6234898018587335305250048840042398106322747308964021053655494390968536524564872845759425074575942507...
+ 0.
7818314824680298087084454266740577502323345187086875289806349580450917316339364412700868007*I
```

When the old embedding is into the real lazy field, then only real embeddings should be considered. See trac ticket #17495:
1.2 Base class for all number fields

class sage.rings.number_field.number_field_base.NumberField
Bases: sage.rings.ring.Field

Base class for all number fields.

OK(*args, **kwds)

Synonym for self.maximal_order(...).

EXAMPLES:

sage: NumberField(x^3 - 2, 'a').OK()
Maximal Order in Number Field in a with defining polynomial x^3 - 2

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: K = QQ[sqrt(5)]
sage: K.minkowski_bound()
1/2*sqrt(5)
sage: K.minkowski_bound().n()
1.11803398874989
sage: K.bach_bound()
12*log(5)^2
sage: K.bach_bound().n()
31.0834847277628
We compute both the Minkowski and Bach bounds for a bigger degree field, where the Bach bound is much better:

```sage
sage: K = CyclotomicField(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:

```sage
sage: QQ.minkowski_bound() 1
```

degree()
Return the degree of this number field.

EXAMPLES:

```sage
sage: NumberField(x^3 + 9, 'a').degree()
3
```

discriminant()
Return the discriminant of this number field.

EXAMPLES:

```sage
sage: NumberField(x^3 + 9, 'a').discriminant()
-243
```

is_absolute()
Return True if self is viewed as a single extension over Q.

EXAMPLES:

```sage
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:

```sage
sage: NumberField(x^3 - 2,'b').maximal_order()  
Maximal Order 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()`
symbolic expression or Rational

EXAMPLES:

The Minkowski bound for \( \mathbb{Q}[i] \) tells us that the class number is 1:

```python
sage: K = QQ[I]
sage: B = K.minkowski_bound(); B
4/pi
sage: B.n()
1.27323954473516
```

We compute the Minkowski bound for \( \mathbb{Q}[\sqrt{2}] \):

```python
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: 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 \( \mathbb{Q}[^2 + \sqrt{3}] \):

```python
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:

```python
sage: QQ.minkowski_bound()
1
```

```python
ring_of_integers(*args, **kwds)
Synonym for self.maximal_order(...).
```

EXAMPLES:

```python
sage: K.<a> = NumberField(x^2 + 1)
sage: K.ring_of_integers()
Gaussian Integers in Number Field in a with defining polynomial x^2 + 1
```
signature()
Return (r1, r2), where r1 and r2 are the number of real embeddings and pairs of complex embeddings of
this field, respectively.

EXAMPLES:

```python
sage: NumberField(x^3 - 2, 'a').signature()
(1, 1)
```

sage.rings.number_field.number_field_base.is_NumberField(x)
Return True if x is of number field type.

EXAMPLES:

```python
sage: from sage.rings.number_field.number_field_base import is_NumberField
sage: is_NumberField(NumberField(x^2+1,'a'))
True
sage: is_NumberField(QuadraticField(-97,'theta'))
True
sage: is_NumberField(CyclotomicField(97))
True
```

Note that the rational numbers QQ are a number field::

```python
sage: is_NumberField(QQ)
True
sage: is_NumberField(ZZ)
False
```

## 1.3 Relative Number Fields

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): modernize coercion implementation
- Robert Harron (2012-08): added is_CM_extension
- Julian Rueth (2014-04-03): absolute number fields are unique parents

This example follows one in the Magma reference manual:

```python
sage: K.<y> = NumberField(x^4 - 420*x^2 + 40000)
sage: z = y^5/11; z
420/11*y^3 - 40000/11*y
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:

```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: cuberoot2 + cuberoot3
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
```

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).

```
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
```

```python
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: sage.rings.number_field.number_field.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 \texttt{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: 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
```

**absolute_base_field()**

Return the base field of this relative extension, but viewed as an absolute field over \( \mathbb{Q} \).

**EXAMPLES:**

```python
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
sage: K.base_field().absolute_field('z')
Number Field in z 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:**

```python
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K
Number Field in a with defining polynomial x^2 - 17 over its base field
sage: 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 \( L\.relative\_different() \).

**EXAMPLES:**

```python
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)
```

**absolute_discriminant \( v=\text{None} \)**

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:

```python
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
```

`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 \( \text{from}_K \) and \( \text{to}_K \), where \( \text{from}_K \) is an isomorphism from \( K \) to \( self \) and \( \text{to}_K \) is an isomorphism from \( self \) to \( K \).

EXAMPLES:

```python
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
```
absolute_generator()
Return the chosen generator over \( \mathbb{Q} \) 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()
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:

```
sage: 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
sage: 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:

```
sage: k.<a, c> = NumberField([x^2 + 1/3, x^2 + 1/4])
sage: k.absolute_polynomial()
x^4 - x^2 + 1
sage: k.pari_polynomial()
x^4 - x^2 + 1
sage: k.base_field().absolute_polynomial()
x^2 + 1/4
sage: k.pari_absolute_base_polynomial()
y^2 + 1
sage: k.relative_polynomial()
x^2 + 1/3
sage: k.pari_relative_polynomial()
x^2 + Mod(y, y^2 + 1)*x - 1
```

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:

```
sage: NumberField(x^2 + (2/3)*x - 9/17,'a').absolute_polynomial_ntl()
([-27 34 51], 51)
```

absolute_vector_space()
Return vector space over \( \mathbb{Q} \) of self and isomorphisms from the vector space to self and in the other
direction.

EXAMPLES:

```
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)
sage: from_V(to_V(c))
7*a^2 - 10*a - 29
sage: from_V(3*to_V(b))
3*b
```

`automorphisms()`

Compute all Galois automorphisms of self over the base field. This is different than computing the embeddings of self into self; there, automorphisms that do not fix the base field are considered.

EXAMPLES:

```
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
(continues on next page)
```
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]

base_field()
Return the base field of this relative number field.

EXAMPLES:

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 a1 with defining polynomial x^3 + x + 1
sage: L.base_field() is k
True

This is very useful because the print representation of a relative field doesn’t describe the base field.:  
sage: L
Number Field in b with defining polynomial z^3 + a over its base field

base_ring()
This is exactly the same as base_field.

EXAMPLES:

sage: k.<a> = NumberField([x^2 + 1, x^3 + x + 1])
sage: k.base_ring()
Number Field in a1 with defining polynomial x^3 + x + 1
sage: k.base_field()
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.

1.3. Relative Number Fields 107
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:

```
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()
```

```
sage: L
Number Field in c with defining polynomial x^4 + 3 over its base field
```

```
sage: L.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)
sage: L.<m, n, r> = K.change_names(); 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)
```

```
composite_fields(other, names=None, both_maps=False, preserve_embedding=True)
```

List of all possible composite number fields formed from self and other, together with (optionally) embeddings into the compositum; see the documentation for both_maps below.

Since relative fields do not have ambient embeddings, preserve_embedding has no effect. In every case all possible composite number fields are returned.

INPUT:
• other - a number field
• names - generator name for composite fields
• both_maps - (default: False) if True, return quadruples \( (F, \text{self}
\text{into}_F, \text{other}
\text{into}_F, k) \) such that \( \text{self}
\text{into}_F \) maps \( \text{self} \) into \( F \), \( \text{other}
\text{into}_F \) maps \( \text{other} \) into \( F \). For relative number fields \( k \) is always None.
• preserve_embedding - (default: True) has no effect, but is kept for compatibility with the absolute version of this function. In every case the list of all possible compositums is returned.

OUTPUT:
• list - list of the composite fields, possibly with maps.

EXAMPLES:

```python
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,
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,
Relative number field morphism:
From: Number Field in c 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: c |---> -9/66560*e^7 + 11/4160*e^5 - 241/4160*e^3 - 101/104*e
d |---> -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
```

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: 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

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:

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_
→ different or absolute_different 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:

sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.discrim() 
Traceback (most recent call last):
...
NotImplementedError: For a relative number field you must use relative_
→ discriminant or absolute_d discriminant as appropriate

discriminant() 
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:

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_d discriminant as appropriate

embeddings(K) 
Compute all field embeddings of the relative number field self 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 self into \( K \) is put first in the list.

INPUT:
• \( K \) – a field

EXAMPLES:
sage: K.<a,b> = NumberField([x^3 - 2, x^2+1])
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

galois_closure(names=None)

Return the absolute number field $K$ that is the Galois closure of this relative number field.

EXAMPLES:

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')
Number Field in c with defining polynomial x^16 + 16*x^14 + 28*x^12 + 784*x^10 + 10 + 19846*x^8 - 595280*x^6 + 2744476*x^4 + 3212848*x^2 + 29953729

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

Return the Galois group of the Galois closure of this number field as an abstract group. Note that even though this is an extension $L/K$, the group will be computed as if it were $L/Q$.

INPUT:

- type: 'pari' or 'gap': type of object to return – a wrapper around a Pari or Gap transitive group object.
- algorithm: ‘pari’, ‘kash’, ‘magma’ (default: ‘pari’, except when the degree is >= 12 when ‘kash’ is tried)

At present much less functionality is available for Galois groups of relative extensions than absolute ones, so try the galois_group method of the corresponding absolute field.

EXAMPLES:

sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^2 + 1)
sage: R.<t> = PolynomialRing(K)
sage: L = K.extension(t^5-t+a, 'b')
sage: L.galois_group(type='pari')
Galois group PARI group [240, -1, 22, "S(5)[x]2"] of degree 10 of the Number Field in b with defining polynomial t^5 - t + a over its base field

gen(n=0)

Return the $n$’th generator of this relative number field.

1.3. Relative Number Fields 111
EXAMPLES:

```python
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
```

gens()
Return the generators of this relative number field.

EXAMPLES:

```python
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)
```

is_CM_extension()
Return True is this is a CM extension, i.e. a totally imaginary quadratic extension of a totally real field.

EXAMPLES:

```python
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()
Returns False, since this is not an absolute field.

EXAMPLES:

```python
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
```

is_free(proof=None)
Determine whether or not $L/K$ is free (i.e. if $\mathcal{O}_L$ is a free $\mathcal{O}_K$-module).
INPUT:

- proof – default: True

EXAMPLES:

```python
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
```

**is_galois()**

For a relative number field, *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 *is_galois_absolute()* or *is_galois_relative()* respectively.

EXAMPLES:

```python
sage: k.<a> = NumberField([x^3 - 2, x^2 + x + 1])
sage: k.is_galois()  # return False
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
```

**is_galois_absolute()**

Return True if for this relative extension $L/K$, $L$ is a Galois extension of $\mathbb{Q}$.

EXAMPLES:

```python
sage: K.<a> = NumberField(x^3 - 2)
sage: y = polygen(K); L.<b> = K.extension(y^2 - a)
sage: L.is_galois_absolute()
False
```

**is_galois_relative()**

Return True if for this relative extension $L/K$, $L$ is a Galois extension of $K$.

EXAMPLES:

```python
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
sage: M.<c> = K.extension(y^3 - a)
sage: M.is_galois_relative()
False
```

The next example previously gave a wrong result; see trac ticket #9390:

```python
sage: F.<a, b> = NumberField([x^2 - 2, x^2 - 3])
sage: F.is_galois_relative()
True
```

**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, \( \text{other} \) can be a relative extension \( M/K' \) with \( \text{base_isom} \) an isomorphism from \( K \) to \( K' \).

**EXAMPLES:**

```python
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')
sage: G = K.galois_group(); gamma = G.gen()
sage: m2 = (gamma^2)(m1)
sage: L2 = K.extension(z^2 - m2, 'b2')
sage: L1.is_isomorphic_relative(L2)
False
sage: L1.is_isomorphic(L2)
True
sage: L3 = K.extension(z^4 - m1, 'b3')
sage: L1.is_isomorphic_relative(L3)
False
```

If we have two extensions over different, but isomorphic, bases, we can compare them by letting \( \text{base_isom} \) be an isomorphism from self’s base field to other’s base field:

```python
cyc.<zeta9> = CyclotomicField(9)
sage: Rcyc.<zcyc> = PolynomialRing(Kcyc)
sage: phi1 = K.hom([zeta9])
sage: m1cyc = phi1(m1)
sage: L1cyc = Kcyc.extension(zcyc^2 - m1cyc, 'b1cyc')
sage: L1.is_isomorphic_relative(L1cyc, base_isom=phi1)
True
sage: L2.is_isomorphic_relative(L1cyc, base_isom=phi1)
False
```

Omitting \( \text{base_isom} \) raises a ValueError when the base fields are not identical:

```python
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.
```

The parameter \( \text{base_isom} \) can also be used to check if the relative extensions are Galois conjugate:

```python
for g in G:
....: if L1.is_isomorphic_relative(L2, g.as_hom()):
....: print(g.as_hom())
```

**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
```

### maximal_order (v=None)

Return the maximal order, i.e., the ring of integers of this number field.

**INPUT:**

- `v` - (default: None) None, a prime, or a list of primes.
  - if `v` is None, return the maximal order.
  - if `v` is a prime, return an order that is p-maximal.
  - if `v` is a list, return an order that is maximal at each prime in the list `v`.

**EXAMPLES:**

```
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.2)
x^2 - x + 1
sage: O2 = K.order([3*a, 2*b])
sage: O2.index_in(OK)
144
```

The following was previously “ridiculously slow”; see trac ticket #4738:

```
sage: K.<a,b> = NumberField([x^4 + 1, x^4 - 3])
sage: K.maximal_order()
Maximal Relative Order in Number Field in a with defining polynomial x^4 + 1
˓→over its base field
```

An example with nontrivial `v`:

```
sage: L.<a,b> = NumberField([x^2 - 1000003, x^2 - 5*1000099^2])
sage: O3 = L.maximal_order([3])
sage: O3.absolute_discriminant()
400160824478095086350656915693814563600
sage: O3.is_maximal()
False
```

### ngens ()

Return the number of generators of this relative number field.

---

### 1.3. Relative Number Fields 115
**EXAMPLES:**

```python
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
```

**number_of_roots_of_unity()**
Return number of roots of unity in this relative field.

**EXAMPLES:**

```python
sage: K.<a, b> = NumberField( [x^2 + x + 1, x^4 + 1] )
sage: 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 (oo) 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:**

```python
sage: P.<a,b,c> = QQ[2^(1/2), 2^(1/3), 3^(1/2)]
sage: R = P.order([a,b,c]); R
Relative Order 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 \( \mathbb{Z} \):.

```python
sage: R.base_ring()
Integer Ring
```

One must give enough generators to generate a ring of finite index in the maximal order:

```python
sage: P.order([a,b])
Traceback (most recent call last):
  ...
ValueError: the rank of the span of gens is wrong
```

**pari_absolute_base_polynomial()**
Return the PARI polynomial defining the absolute base field, in \( y \).

**EXAMPLES:**

```python
```
sage: x = polygen(ZZ)
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())
<type 'cypari2.gen.Gen'>
sage: z = ZZ['z'].0
sage: K.<a, b, c> = NumberField([z^2 + 2, z^2 + 3, z^2 + 5]); 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())
<type 'cypari2.gen.Gen'>

\texttt{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 \texttt{rnfinit} function. Notice that if this is a relative extension of a relative extension, the base field is the absolute base field.

EXAMPLES:

\begin{verbatim}
  sage: k.<i> = NumberField(x^2 + 1)
sage: m.<z> = k.extension(k['w'](i,0,1))
sage: m
Number Field in z with defining polynomial w^2 + i over its base field
sage: m.pari_relative_polynomial()
Mod(1, y^2 + 1)*x^2 + Mod(y, y^2 + 1)
sage: l.<t> = m.extension(m['t'].0^2 + z)
sage: l.pari_relative_polynomial()
Mod(1, y^4 + 1)*x^2 + Mod(y, y^4 + 1)
\end{verbatim}

\texttt{pari_rnf()}

Return the PARI relative number field object associated to this relative extension.

EXAMPLES:

\begin{verbatim}
  sage: k.<a> = NumberField([x^4 + 3, x^2 + 2])
sage: k.pari_rnf()
[x^4 + 3, [364, -10*x^7 - 87*x^5 - 370*x^3 - 41*x], [108, 3], ...]
\end{verbatim}

\texttt{places(\texttt{all\_complex=False, prec=None})}

Return the collection of all infinite places of self.

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

On the other hand, if \texttt{prec} is not None, we simply return places into \texttt{RealField(prec)} and \texttt{ComplexField(prec)} (or RDF, CDF if \texttt{prec}=53).

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

EXAMPLES:

\begin{verbatim}
1.3. Relative Number Fields 117
\end{verbatim}
sage: L.<b, c> = NumberFieldTower([x^2 - 5, x^3 + x + 3])
sage: L.places()
[Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base field
  To:   Real Field with 106 bits of precision
  Defn: b |--> -2.236067977499789696409173668937
  c |--> -1.21341166276229634132131377426,
  Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base field
  To:   Real Field with 106 bits of precision
  Defn: b |--> 2.236067977499789696411548005367
  c |--> -1.21341166276229634130492421800,
  Relative number field morphism:
  From: Number Field in b with defining polynomial x^2 - 5 over its base field
  To:   Complex Field with 53 bits of precision
  Defn: b |--> -2.23606797749979...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 field
  To:   Complex Field with 53 bits of precision
  Defn: b |--> 2.23606797749979 - 4.44089209850063e-16*I
  c |--> 0.606705831381115 - 1.45061224918844*I]

polynomial()
For a relative number field, polynomial() is deliberately not implemented. Either
relative_polynomial() or absolute_polynomial() must be used.

EXAMPLES:
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

relative_degree()
Returns the relative degree of this relative number field.

EXAMPLES:
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: K.relative_degree()
2

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 L.absolute_different().

EXAMPLES:
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 $L/\mathbb{Q}$, use e.g. $L.absolute_discriminant()$.

**EXAMPLES:**

```python
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
```

**relative_polynomial()**

Return the defining polynomial of this relative number field over its base field.

**EXAMPLES:**

```python
sage: K.<a> = NumberFieldTower([x^2 + x + 1, x^3 + x + 1])
sage: K.relative_polynomial()
x^2 + x + 1
```

Use absolute polynomial for a polynomial that defines the absolute extension:

```python
sage: K.absolute_polynomial()
x^6 + 3*x^5 + 8*x^4 + 9*x^3 + 7*x^2 + 6*x + 3
```

**relative_vector_space()**

Return vector space over the base field of self and isomorphisms from the vector space to self and in the other direction.

**EXAMPLES:**

```python
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()
sage: from_V(V.0)
1
sage: to_V(K.0)
(0, 1)
sage: from_V(to_V(K.0))
a
sage: to_V(from_V(V.0))
(1, 0)
sage: to_V(from_V(V.1))
(0, 1)
```

The underlying vector space and maps is cached:

```python
sage: W, from_V, to_V = K.relative_vector_space()
sage: V is W
True
```

**relativize**(alpha, names)

Given an element in self or an embedding of a subfield into self, return a relative number field $K$ isomor-
The Sage Reference Manual: Algebraic Numbers and Number Fields, Release 8.7

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:**
- `alpha` – an element of self, or an embedding of a subfield into self
- `names` – name of generator for output field $K$.

**OUTPUT:** $K$ – a relative number 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
test = 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)
|z^2 = z^2
|w^2 = w^2
|z^3 = -3
|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
```

Now suppose we have $K$ below $L$ below $M$:

```sage
test = 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()
|L_over_K = L.relativize(K_into_L, 'c'); L_over_K
|L_over_K_to_L = L.relativize(K_into_L, 'c'); L_over_K_to_L
|L_to_L_over_K = L.relativize(K_into_L, 'c'); L_to_L_over_K
|L_over_K = Number Field in c with defining polynomial x^2 + a0_0 over its base field
|L_to_L_over_K = L.relativize(K_into_L, 'c'); L_to_L_over_K
|L_over_K = Number Field in d with defining polynomial x^2 + c over its base field
|L_over_K = Number Field in d with defining polynomial x^2 + c over its base field
|is L_over_K = True
```

Test relativizing a degree 6 field over its degree 2 and degree 3 subfields, using both an explicit element:

```sage
test = K.<a> = NumberField(x^6 + 2); K
Number Field in a with defining polynomial x^6 + 2
sage: K2, K2_into_K, _ = K.subfields(2)[0]; K2
Number Field in a0 with defining polynomial x^2 + 2
sage: K3, K3_into_K, _ = K.subfields(3)[0]; K3
Number Field in a0 with defining polynomial x^3 - 2
```

Here we explicitly relativize over an element of $K_2$ (not the generator):
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()
Number Field in c1 with defining polynomial x^2 - 2*x + 3
```

```
roots_of_unity()
```

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

**EXAMPLES:**

```python
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 + 72, Ring morphism:
  From: Number Field in c0 with defining polynomial x^2 - 24*x + 72
  To:  Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b -
  Defn: c0 |--> -6*a + 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 -
  Defn: c1 |--> 2*b*a + 12, None),
(Number Field in c2 with defining polynomial x^2 - 24*x + 96, Ring morphism:
  From: Number Field in c2 with defining polynomial x^2 - 24*x + 96
  To:  Number Field in c with defining polynomial Y^2 + (-2*b - 3)*a - 2*b -
  Defn: c2 |--> -4*b + 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 w0 with defining polynomial x^8 - 12*x^6 + 36*x^4 - 36*x^2 + 9
   Defn: c |--> -1/3*w0^7 + 4*w0^5 - 12*w0^3 + 11*w0
   a |--> 1/3*w0^6 - 10/3*w0^4 + 5*w0^2
   b |--> -2/3*w0^6 + 7*w0^4 - 14*w0^2)]
sage: K.subfields(3)
[]

uniformizer (P, 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

• 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:

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

vector_space()
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: 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

sage.rings.number_field.number_field_rel.NumberField_relative_v1(base_field,
poly, name, latex_name, canoni-
ocal_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.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: is_RelativeNumberField(NumberField(x^2+1,'a'))
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
sage: is_RelativeNumberField(QQ)
False
```

1.4 Number Field Elements

AUTHORS:

- William Stein: version before it got Cython’d
- Joel B. Mohler (2007-03-09): First reimplementat‌‌‌ion in Cython
- Robert Bradshaw (2007-09-15): specialized classes for relative and absolute elements
- Robert Harron (2012-08): conjugate() now works for all fields contained in CM fields

```
class sage.rings.number_field.number_field_element.CoordinateFunction(alpha, W,
i0_V)

Bases: object

This class provides a callable object which expresses elements in terms of powers of a fixed field generator α.
```

EXAMPLES:
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

alpha()

EXAMPLES:

sage: k.<a> = NumberField(x^3 + 2)
sage: (a + 2).coordinates_in_terms_of_powers().alpha()
a + 2

class sage.rings.number_field.number_field_element.NumberFieldElement
Bases: sage.structure.element.FieldElement

An element of a number field.

EXAMPLES:

sage: k.<a> = NumberField(x^3 + x + 1)
sage: a^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 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 prec bits of precision is used, defaulting to 53 bits of precision if prec is not provided. The result is in the corresponding real field.

INPUT:

• 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: K.<a> = NumberField(x^3+17)
sage: abs(a)
2.57128159065824
sage: abs(a, prec=100)
2.571281590658235354531872087
sage: abs(a, prec=100, i=1)
(continues on next page)
2.5712815906582353554531872087
\texttt{sage: a.abs(100, 2)}
2.5712815906582353554531872087

Here’s one where the absolute value depends on the embedding:

\begin{verbatim}
\texttt{sage: K.<b> = NumberField(x^2-2)}
\texttt{sage: a = 1 + b}
\texttt{sage: a.abs(i=0)}
0.414213562373095
\texttt{sage: a.abs(i=1)}
2.41421356237309
\end{verbatim}

Check that trac ticket #16147 is fixed:

\begin{verbatim}
\texttt{sage: x = polygen(ZZ)}
\texttt{sage: f = x^3 - x - 1}
\texttt{sage: beta = f.complex_roots()[0]; beta}
1.32471795724475
\texttt{sage: K.<b> = NumberField(f, embedding=beta)}
\texttt{sage: b.abs()}
1.32471795724475
\end{verbatim}

Check that for fields with real coercion embeddings, absolute values are in the same field (trac ticket #21105):

\begin{verbatim}
\texttt{sage: x = polygen(ZZ)}
\texttt{sage: f = x^3 - x - 1}
\texttt{sage: K.<b> = NumberField(f, embedding=1.3)}
\texttt{sage: b.abs()}
\texttt{b}
\end{verbatim}

However, if a specific embedding is requested, the behavior reverts to that of number fields without a coercion embedding into $\mathbb{R}$:

\begin{verbatim}
\texttt{sage: b.abs(i=2)}
1.32471795724475
\end{verbatim}

Also, if a precision is requested explicitly, the behavior reverts to that of number fields without a coercion embedding into $\mathbb{R}$:

\begin{verbatim}
\texttt{sage: b.abs(prec=53)}
1.32471795724475
\end{verbatim}

\texttt{abs\_non\_arch (P, prec=None)}

Return the non-archimedean absolute value of this element with respect to the prime $P$, to the given precision.

\textbf{INPUT:}

- $P$ - a prime ideal of the parent of self
- \texttt{prec} (int) – desired floating point precision (default: default RealField precision).

\textbf{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 $1/p$. 

1.4. Number Field Elements
EXAMPLES:

```python
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:

```python
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_norm()  
Return the absolute norm of this number field element.

EXAMPLES:

```python
sage: K1.<a1> = CyclotomicField(11)
```
```
sage: K2.<a2> = K1.extension(x^2 - 3)
```
```
sage: K3.<a3> = K2.extension(x^2 + 1)
```
```
sage: (a1 + a2 + a3).absolute_norm()
1353244757701
```
```
sage: QQ(7/5).absolute_norm()
7/5
```

additive_order()  
Return the additive order of this element (i.e. infinity if self != 0, 1 if self == 0)

EXAMPLES:

```python
sage: K.<u> = NumberField(x^4 - 3*x^2 + 3)
```
```
sage: u.additive_order()
+Infinity
```
```
sage: K(0).additive_order()
1
```
```
sage: K.ring_of_integers().characteristic() # implicit doctest
0
```

ceil()  
Return the ceiling 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)
4.1344473767055
```
```
sage: b.ceil()
5
```

This function always succeeds even if a tremendous precision is needed:
If the number field is not embedded, this function is valid only if the element is rational:

```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
```

`charpoly` \((\text{var}='x')\)
Return the characteristic polynomial of this number field element.

**EXAMPLES:**

```sage
K.<a> = NumberField(x^3 + 7)
sage: a.charpoly()
x^3 + 7
sage: K(1).charpoly()
x^3 - 3*x^2 + 3*x - 1
```

`complex_embedding` \((\text{prec}=53, i=0)\)
Return the i-th embedding of self in the complex numbers, to the given precision.

**EXAMPLES:**

```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: a.complex_embedding(100)
-0.62996052494743658238360530364 - 1.0911236359717214035600726142*I
sage: a.complex_embedding(20, 1)
-0.62996 + 1.0911*I
sage: a.complex_embedding(20, 2)
1.2599
```

`complex_embeddings` \((\text{prec}=53)\)
Return the images of this element in the floating point complex numbers, to the given bits of precision.

**INPUT:**

- \(\text{prec} - \text{integer (default: 53)}\) bits of precision

**EXAMPLES:**

```sage
k.<a> = NumberField(x^3 - 2)
sage: a.complex_embeddings()
[-0.629960524947437 - 1.09112363597172*I, -0.629960524947437 + 1.09112363597172*I, 1.25992104989487]
(continues on next page)```
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
```

The following example now works.

```
sage: F.<b> = NumberField(x^2 - 2)
sage: K.<j> = F.extension(x^2 + 1)
sage: 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.
```

An example of a non-quadratic totally real field.

```
sage: F.<a> = NumberField(x^4 + x^3 - 3*x^2 - x + 1)
sage: a.conjugate()
```

An example of a non-cyclotomic CM field.

```
sage: K.<a> = NumberField(x^4 - x^3 + 2*x^2 + x + 1)
sage: a.conjugate()
-a^3 - 2*a^2 - 2
```

coordinates_in_terms_of_powers()

Let $\alpha$ be self. Return a callable object (of type CoordinateFunction) that takes any element of the parent of self in $\mathbb{Q}(\alpha)$ and writes it in terms of the powers of $\alpha$: $1, \alpha, \alpha^2, \ldots$. 
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.

```
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]
```

This function works even if self only generates a subfield of this number field.

```
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:

```
sage: K.<z> = CyclotomicField(3)
sage: a = 1/3 + (1/5)*z
sage: a.denominator()
15
```

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: 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)

\textbf{descend\_mod\_power (K='QQ', d=2)}

Return a list of elements of the subfield $K$ equal to \texttt{self} modulo $d$'th powers.

\textbf{INPUT:}

\begin{itemize}
  \item $K$ (number field, default \texttt{QQ}) – a subfield of the parent number field $L$ of \texttt{self}
  \item $d$ (positive integer, default 2) – an integer at least 2
\end{itemize}

\textbf{OUTPUT:}

A list, possibly empty, of elements of $K$ equal to \texttt{self} modulo $d$'th powers, i.e. the preimages of \texttt{self} under the map $K^*/(K^*)^d \rightarrow L^*/(L^*)^d$ where $L$ is the parent of \texttt{self}. A \texttt{ValueError} is raised if $K$ does not embed into $L$.

\textbf{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 \texttt{self} is not divisible by $d$.

\textbf{EXAMPLES:}

A relative example:

\begin{verbatim}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)
[-3*i - 2, -2*i + 3]
\end{verbatim}

An absolute example:

\begin{verbatim}sage: K.<zeta> = CyclotomicField(8)
sage: K(1).descend_mod_power(QQ,2)
[1, 2, 2, 2]
sage: a = 17*K.random_element()^2
sage: a.descend_mod_power(QQ,2)
[17, 34, 34, 34]
\end{verbatim}

\textbf{factor ()}

Return factorization of this element into prime elements and a unit.

\textbf{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:

```python
sage: K.<i> = NumberField(x^2+1)
sage: (6+i + 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:

```python
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
```

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
```

`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(125/7).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)
```

(continues on next page)
sage: K(2/3).floor()
0
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(Qbar/Q)-conjugates of this number field element in the field K.

EXAMPLES:
In the first example the conjugates are obvious:
sage: K.<a> = NumberField(x^2 - 2)
sage: a.galois_conjugates(K)
[a, -a]
sage: K(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.
sage: K.<a> = NumberField(x^3 - 2)
sage: c = a.galois_conjugates(K); c
[a]
sage: K.<a> = NumberField(x^3 - 2)
sage: 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]
sage: c[0]^3
2
sage: parent(c[0])
Number Field in a1 with defining polynomial x^6 + 108
sage: parent(c[0]).is_galois()
True

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]

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(1).gcd(0)
1
sage: K(0).gcd(0)
0
sage: R = K.maximal_order()
sage: R(i+1).gcd(2)
i + 1

Non-maximal orders are not supported:

sage: R = K.order(2*i)
sage: R(1).gcd(R(4*i))
Traceback (most recent call last):
...  
NotImplementedError: gcd() for Order in Number Field in i with defining polynomial x^2 + 1 is not implemented

The following field has class number 3, but if the ideal (self, other) happens to be principal, this still works:

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

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()
0.78978069900813892060267152032141577237037181070060784564457

The global height of an algebraic number is absolute, i.e. it does not depend on the parent field:

```
sage: QQ(6).global_height()
sage: K(6).global_height()
sage: L.<b> = NumberField((a^2).minpoly())
sage: L.degree()
sage: b.global_height()  # element of L (degree 2 field)
sage: (a^2).global_height()  # element of K (degree 4 field)
```

And of course every element has the same height as its inverse:

```
sage: K.<s> = QuadraticField(2)
sage: s.global_height()
sage: (1/s).global_height()  # make sure that 11758 is fixed
```

---

### global_height_arch(prec=None)

Returns the total archimedean component of the height of self.

**INPUT:**

- `prec` (int) – desired floating point precision (default: default RealField precision).

**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.

**EXAMPLES:**

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: b = a/2
sage: b.global_height_arch()
```

---

### global_height_non_arch(prec=None)

Returns 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:**
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: [b.local_height(P, weighted=True) for P in b.support()]
[0.000000000000000, 2.77258872223978, 2.19722457733622, 2.19722457733622]
sage: sum([b.local_height(P,weighted=True) for P in b.support()])
7.16703787691222
```

A relative example:

```python
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) != (1).

**NOTE:** It’s not implemented yet for non-integral elements.

**EXAMPLES:**

```python
sage: k.<a> = NumberField(x^2 + 23)
sage: N = k.ideal(3)
sage: d = 3*a + 1
sage: d.inverse_mod(N)
1
```
```

```python
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])
7
sage: 7*b - 1 in K.ideal(37, a - b)
True
sage: b.inverse_mod([37, a - b]).parent() == K
True
```

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:

```python
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()
True
sage: K(0).is_integer()
True
sage: K(1/2).is_integer()
False
```

`is_integral()`
Determine if a number is in the ring of integers of this number field.

EXAMPLES:

```python
sage: K.<a> = NumberField(x^2 + 23)
sage: a.is_integral()
True
sage: t = (1+a)/2
sage: t.is_integral()
True
sage: t.minpoly()
x^2 - x + 6
sage: t = a/2
sage: t.is_integral()
False
sage: t.minpoly()
x^2 + 23/4
```

An example in a relative extension:

```python
sage: K.<a,b> = NumberField([x^2+1, x^2+3])
sage: (a+b).is_integral()
True
sage: ((a-b)/2).is_integral()
False
```

`is_norm(L, element=False, 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=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 False, 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 \( \text{self} == \text{x.norm(K)} \). Otherwise, \( x \) is None.

**ALGORITHM:**

Uses PARI’s \texttt{rnfisnorm}. See \texttt{self._rnfisnorm()}.  

**EXAMPLES:**

```python
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
```

With a relative base field:

```python
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
```

Non-Galois number fields:

```python
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)
Traceback (most recent call last):
  ...  
NotImplementedError: is_norm is not implemented unconditionally for norms \( \rightarrow \) from non-Galois number fields
sage: (a/2).is_norm(L, proof=False)
False
sage: K.<a> = NumberField(x^3 + x + 1)
sage: Q.<X> = K[]
sage: L.<b> = NumberField(X^4 + a)
sage: t = (-a).is_norm(L, element=True); t
(True, b^3 + 1)
sage: t[1].norm(K)
-a
```

**AUTHORS:**

- Craig Citro (2008-04-05)  
- Marco Streng (2010-12-03)

**is_nth_power** \((n)\)

Return True if \( \text{self} \) is an \( n \)’th power in its parent \( K \).

**EXAMPLES:**

1.4. Number Field Elements 137
is_one()  
Test whether this number field element is 1.

EXAMPLES:

```python
sage: K.<a> = NumberField(x^3 + 3)
sage: K(1).is_one()  # True
sage: K(0).is_one()  # False
sage: K(-1).is_one()  # False
sage: K(1/2).is_one()  # False
sage: 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
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:

```python
sage: K.<cbrt3> = NumberField(x^3 - 3)
sage: cbrt3.is_rational()  # False
```
\begin{itemize}
\item \texttt{sage: (cbrt3**2 - cbrt3 + 1/2).is_rational()}
False
\item \texttt{sage: K(-12).is_rational()}
True
\item \texttt{sage: K(0).is_rational()}
True
\item \texttt{sage: K(1/2).is_rational()}
True
\end{itemize}

\textbf{is\_square (root=False)}

Return True if self is a square in its parent number field and otherwise return False.

**INPUT:**

- \texttt{root} - if True, also return a square root (or None if self is not a perfect square)

**EXAMPLES:**

\begin{itemize}
\item \texttt{sage: m.<b> = NumberField(x^4 - 1789)}
\item \texttt{sage: b.is_square()}
False
\item \texttt{sage: c = (2/3*b + 5)^2; c}
4/9*b^2 + 20/3*b + 25
\item \texttt{sage: c.is_square()}
True
\item \texttt{sage: c.is_square(True)}
(True, 2/3*b + 5)
\end{itemize}

We also test the functional notation.

\begin{itemize}
\item \texttt{sage: is_square(c, True)}
(True, 2/3*b + 5)
\item \texttt{sage: is_square(c)}
True
\item \texttt{sage: is_square(c+1)}
False
\end{itemize}

\textbf{is\_totally\_positive ()}

Returns True if self is positive for all real embeddings of its parent number field. We do nothing at complex places, so e.g. any element of a totally complex number field will return True.

**EXAMPLES:**

\begin{itemize}
\item \texttt{sage: F.<b> = NumberField(x^3-3*x-1)}
\item \texttt{sage: b.is_totally_positive()}
False
\item \texttt{sage: (b^2).is_totally_positive()}
True
\end{itemize}

\textbf{is\_unit ()}

Return True if self is a unit in the ring where it is defined.

**EXAMPLES:**

\begin{itemize}
\item \texttt{sage: K.<a> = NumberField(x^2 - x - 1)}
\item \texttt{sage: OK = K.ring_of_integers()}
\item \texttt{sage: OK(a).is_unit()}  
True
\end{itemize}
It also works for relative fields and orders:

```
sage: K.<a,b> = NumberField([x^2 - 3, x^4 + x^3 + x^2 + x + 1])
sage: OK = K.ring_of_integers()
sage: OK(b).is_unit()  # True
sage: OK(a).is_unit()  # False  
sage: a.is_unit()  # True
```

### list()

Return the list of coefficients of self written in terms of a power basis.

**EXAMPLES:**

```
sage: K.<a> = NumberField(x^3 - x + 2); ((a + 1)/(a + 2)).list()
[1/4, 1/2, -1/4]
sage: K.<a, b> = NumberField([x^3 - x + 2, x^2 + 23]); ((a + b)/(a + 2)).list()
[3/4*b - 1/2, -1/2*b + 1, 1/4*b - 1/2]
```

### local_height($P$, prec=None, weighted=False)

Returns the local height of self at a given prime ideal $P$.

**INPUT:**

- $P$ - a prime ideal of the parent of self
- prec (int) – desired floating point precision (default: default RealField precision).
- weighted (bool, default False) – if True, apply local degree weighting.

**OUTPUT:**

(real) The local height of this number field element at the place $P$. If $weighted$ is True, this is multiplied by the local degree (as required for global heights).

**EXAMPLES:**

```
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)
4.1108738641733112487513891034256147463156817430812610629374
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
```

### local_height_arch

*(i, prec=None, 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).
- **`prec` (int)** – desired floating point precision (default: default RealField precision).
- **`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 `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
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: [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]
sage: [a.local_height_arch(i) for i in range(3)]
[0.5301924545717755083366563897519,
 0.5301924545717755083366563897519,
 0.886414217456333]
sage: [a.local_height_arch(i, weighted=True) for i in range(3)]
[0.5301924545717755083366563897519,
 0.5301924545717755083366563897519,
 1.77282843491267]
```

A relative example:

```python
sage: L.<b, c> = NumberFieldTower([x^2 - 5, x^3 + x + 3])
sage: [(b + c).local_height_arch(i) for i in range(4)]
[1.23822339075788491842206617439,
 0.02240347229957857807697491391,
 0.7800289614914618,
 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
sage: L.<b> = K.extension(K['x'].0^2 - 2)
sage: M = b.matrix(); M
[0 1]
[2 0]
sage: M.base_ring() is K
True
```

Absolute number field:

```python
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
```

More complicated relative number field:

```python
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
```

An example where we explicitly give the subfield or the embedding:

```python
sage: K.<a> = NumberField(x^4 + 1); L.<a2> = NumberField(x^2 + 1)
sage: a.matrix(L)
[ 0 1]
[a2 0]
```

Notice that if we compute all embeddings and choose a different one, then the matrix is changed as it should be:
```python
sage: v = L.embeddings(K)
sage: a.matrix(v[1])
[ 0 1]
[-a2 0]
```

The norm is also changed:

```python
sage: a.norm(v[1])
a2
sage: a.norm(v[0])
-a2
```

### minpoly (var='x')

Return the minimal polynomial of this number field element.

**EXAMPLES:**

```python
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
```

### 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
sage: x = polygen(QQ)
sage: K.<a>=NumberField(x^40 - x^20 + 4)
sage: u = 1/4*a^30 + 1/4*a^10 + 1/2
sage: u.multiplicative_order()
6
sage: a.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
```
\textbf{\texttt{\texttt{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 norm from \( L \) to \( K \). In all other cases, the norm is the absolute norm down to \( \mathbb{Q} \).

\textbf{EXAMPLES:}

\begin{verbatim}
sage: K.<a> = NumberField(x^3 + x^2 + x - 132/7); K
Number Field in a with defining polynomial x^3 + x^2 + x - 132/7
sage: a.norm()
132/7
sage: factor(a.norm())
2^2 * 3 * 7^-1 * 11
sage: K(0).norm()
0

Some complicated relatives norms in a tower of number fields.

sage: K.<a,b,c> = NumberField([x^2 + 1, x^2 + 3, x^2 + 5])
sage: L = K.base_field(); M = L.base_field()
sage: a.norm()
1
sage: a.norm(L)
1
sage: a.norm(M)
1
sage: a
a
sage: (a+b+c).norm()
121
sage: (a+b+c).norm(L)
2*c*b - 7
sage: (a+b+c).norm(M)
-11

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
\end{verbatim}
\texttt{nth\_root}(n, all=False)

Return an \(n\)‘th root of \texttt{self} in its parent \(K\).

**EXAMPLES:**

\begin{verbatim}
sage: K.<a> = NumberField(x^4-7)
sage: K(7).nth_root(2)
a^2
sage: K((a-3)^5).nth_root(5)
a - 3
\end{verbatim}

**ALGORITHM:** Use PARI to factor \(x^n - \texttt{self}\) in \(K\).

\texttt{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:**

\texttt{denominator\_ideal}()

**EXAMPLES:**

\begin{verbatim}
sage: K.<a> = NumberField(x^2+5)
sage: b = (1+a)/2
sage: b.norm()
3/2
sage: N = b.numerator_ideal(); N
Fractional ideal (3, a + 1)
sage: N.norm()
3
sage: (1/b).numerator_ideal()
Fractional ideal (2, a + 1)
sage: K(0).numerator_ideal()
Ideal (0) of Number Field in a with defining polynomial x^2 + 5
\end{verbatim}

\texttt{ord}(P)

Returns the valuation of \texttt{self} at a given prime ideal \(P\).

**INPUT:**

\begin{itemize}
  \item \(P\) - a prime ideal of the parent of \texttt{self}
\end{itemize}

**Note:** The function \texttt{ord()} is an alias for \texttt{valuation()}.

**EXAMPLES:**

\begin{verbatim}
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))
<type 'sage.rings.integer.Integer'>
\end{verbatim}
The function can be applied to elements in relative number fields:

```
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]
```

**polynomial (var='x')**

Return the underlying polynomial corresponding to this number field element.

The resulting polynomial is currently *not* cached.

**EXAMPLES:**

```
sage: K.<a> = NumberField(x^5 - x - 1)
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
sage: g = f.polynomial(); g
1/81*x^4 - 8/81*x^3 + 8/27*x^2 - 32/81*x + 16/81
sage: parent(g)
Univariate Polynomial Ring in x over Rational Field
```

Note that the result of this function is not cached (should this be changed?):

```
sage: g is f.polynomial()
False
```

**relative_norm ()**

Return the relative norm of this number field element over the next field down in some tower of number fields.

**EXAMPLES:**

```
sage: K1.<a1> = CyclotomicField(11)
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
```

**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```
```python
K.<a> = NumberField(x - 1)
K(11).residue_symbol(K.ideal(17),2)
-1
KroneckerSymbol(11,17)
-1
```

The result depends on the number field of the ideal:

```sage```
```python
K.<a> = NumberField(x - 1)
L.<b> = K.extension(x^2 + 1)
K(7).residue_symbol(K.ideal(11),2)
-1
K(7).residue_symbol(L.ideal(11),2)
1
```

Cubic Residue:

```sage```
```python
K.<w> = NumberField(x^2 - x + 1)
K(w^2 + 3).residue_symbol(K.ideal(17),3)
-w
```

The field must contain the m-th roots of unity:

```sage```
```python
K.<w> = NumberField(x^2 - x + 1)
K(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
```

round() Return the round (nearest integer) of this number field element.

EXAMPLES:

```sage```
```python
x = polygen(ZZ)
p = x^7 - 5*x^2 + x + 1
```

(continues on next page)
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:

sage: c = b - 5678322907931/1225243417356 + 3
sage: c.round()
3
sage: RIF(c).unique_round()
Traceback (most recent call last):
... ValueError: interval does not have a unique round (nearest integer)

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(k/3).round() for k in range(-3,4)]
[-1, -1, 0, 0, 0, 1, 1]

The sign of this algebraic number (if a real embedding is well defined)

EXAMPLES:

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

If the field is not embedded in real numbers, this method will only work for rational elements:
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

sqrt (all=False)
Returns the square root of this number in the given number field.

EXAMPLES:

sage: K.<a> = NumberField(x^2 - 3)
sage: K(3).sqrt()
a
sage: K(3).sqrt(all=True)
[a, -a]
sage: K(a^10).sqrt()
9*a
sage: K(49).sqrt()
7
sage: K(I+a).sqrt()
Traceback (most recent call last):
  ...
ValueError: a + 1 not a square in Number Field in a with defining polynomial
  x^2 - 3
sage: K(0).sqrt()
0
sage: K((7+a)^2).sqrt(all=True)
[a + 7, -a - 7]

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 primes 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)
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
sage: [ P5.norm() for P5 in P5s ]
[5, 25]

trace(K=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 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 QQ.

EXAMPLES:

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:

sage: K.<zeta> = CyclotomicField(17)
sage: R = K.ring_of_integers()
sage: R(zeta).trace().parent()
Integer Ring

valuation(P)

Returns the valuation of self at a given prime ideal P.

INPUT:

• P - a prime ideal of the parent of self

Note: The function ord() is an alias for valuation().

EXAMPLES:

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))
<type 'sage.rings.integer.Integer'>

The function can be applied to elements in relative number fields:
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]

vector()
Return vector representation of self in terms of the basis for the ambient number field.

EXAMPLES:

sage: K.<a> = NumberField(x^2 + 1)
sage: (2/3*a - 5/6).vector()
(-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)

class sage.rings.number_field.number_field_element.NumberFieldElement_absolute
Bases: sage.rings.number_field.number_field_element.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
sage: 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:
```python
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^10 + 10*x^9 + 25*x^8 - 80*x^7 - 438*x^6 + 80*x^5 + 2950*x^4 + 1520*x^3 - 10439*x^2 - 5130*x + 18225
sage: b.absolute_minpoly()
x^5 + 5*x^4 - 40*x^3 - 19*x + 135
sage: b.absolute_minpoly(algorithm='pari') == b.absolute_minpoly(algorithm='sage')
True
```

`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 `self.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 trac ticket #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: 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
sage: (a-a).is_real_positive()
False

lift(var='x')
Return an element of QQ[x], where this number field element lives in QQ[x]/(f(x)).

EXAMPLES:

sage: K.<a> = QuadraticField(-3)
sage: a.lift()
x

list()
Return the list of coefficients of self written in terms of a power basis.

EXAMPLES:

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]

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.

sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3-2)
sage: a.minpoly('x')
x^3 - 2
class sage.rings.number_field.number_field_element.NumberFieldElement_relative
Bases: sage.rings.number_field.number_field_element.NumberFieldElement

The current relative number field element implementation does everything in terms of absolute polynomials. All conversions from relative polynomials, lists, vectors, etc should happen in the parent.

absolute_charpoly(var='x', algorithm=None)
The characteristic polynomial of this element over \( \mathbb{Q} \).

We construct a relative extension and find the characteristic polynomial over \( \mathbb{Q} \).

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 trac ticket #5213 for a tuning script.

EXAMPLES:

```python
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3-2)
sage: S.<X> = K[]
sage: L.<b> = NumberField(X^3 + 17); L
Number Field in b with defining polynomial X^3 + 17 over its base field
sage: b.absolute_charpoly()
x^9 + 51*x^6 + 867*x^3 + 4913
sage: b.charpoly()(b)
0
sage: a = L.0; a
b
sage: a.absolute_charpoly('x')
x^9 + 51*x^6 + 867*x^3 + 4913
sage: a.absolute_charpoly('y')
y^9 + 51*y^6 + 867*y^3 + 4913
sage: a.absolute_charpoly(algorithm='pari') == a.absolute_charpoly(algorithm='sage')
True
```

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 \texttt{absolute_charpoly()}.

EXAMPLES:

```python
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.absolute_charpoly()
x^8 - 1996*x^6 + 996006*x^4 + 1997996*x^2 + 1
sage: c.absolute_minpoly()
x^8 - 1996*x^6 + 996006*x^4 + 1997996*x^2 + 1
sage: L(a).absolute_charpoly()
x^8 + 8*x^6 + 24*x^4 + 32*x^2 + 16
```
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
```

lift (var='x')

Return an element of K[x], where this number field element lives in the relative number field K[x]/(f(x)).

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
```

list ()

Return the list of coefficients of self written in terms of a power basis.

EXAMPLES:

```
sage: K.<a,b> = NumberField([x^3+2, x^2+1])
sage: a.list()
(continues on next page)
```
valuation($P$)
Returns the valuation of self at a given prime ideal $P$.

INPUT:

- $P$ - a prime ideal of relative number field which is the parent of self

EXAMPLES:

```
sage: K.<a, b, c> = NumberField([x^2 - 2, x^2 - 3, x^2 - 5])
sage: P = K.prime_factors(5)[0]
sage: (2*a + b - c).valuation(P)
1
```
sage: w.inverse_mod(13).parent() == OE
True

sage: w.inverse_mod(2*OE)
Traceback (most recent call last):
...
ZeroDivisionError: w is not invertible modulo Fractional ideal (2)

```python
class sage.rings.number_field.number_field_element.OrderElement_relative
    Bases: sage.rings.number_field.number_field_element.NumberFieldElement_relative

Element of an order in a relative number field.

EXAMPLES:

sage: O = EquationOrder([x^2 + x + 1, x^3 - 2],'a,b')
sage: c = O.1; c
(-2*b^2 - 2)*a - 2*b^2 - b
sage: type(c)
<type 'sage.rings.number_field.number_field_element.OrderElement_relative'>

absolute_charpoly(var='x')
The absolute characteristic polynomial of this order element over ZZ.

EXAMPLES:

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

absolute_minpoly(var='x')
The absolute minimal polynomial of this order element over ZZ.

EXAMPLES:

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

charpoly(var='x')
The characteristic polynomial of this order element over its base ring.
This special implementation works around bug #4738. At this time the base ring of relative order elements is ZZ; it should be the ring of integers of the base field.

**EXAMPLES:**

```python
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
```

```python
c 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
```

```python
sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order in Number Field in b with defining polynomial x^2 - 3
```

```python
sage: [ charpoly(t) for t in OK.basis() ]
[x^2 - 2*x + 1, x^2 + b*x + 1, x^2 - x + 1, x^2 + 1]
sage: 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: E.<a,b> = NumberField([x^2 - x + 2, x^2 + 1])
```

```python
c sage: OE = E.ring_of_integers()
c sage: t = OE(b - a).inverse_mod(17*b)
c sage: t*(b - a) - 1 in E.ideal(17*b)
True
c sage: t.parent() == OE
True
```

```python
sage: minpoly('x')
```

The minimal polynomial of this order element over its base ring.

This special implementation works around bug #4738. At this time the base ring of relative order elements is ZZ; it should be the ring of integers of the base field.

**EXAMPLES:**

```python
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
```

```python
c 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
```

```python
sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order in Number Field in b with defining polynomial x^2 - 3
```

```python
sage: _, u, _, v = OK.basis()
c sage: t = 2*u - v; t
-b
c sage: t.charpoly()
x^2 + 2*b*x + 3
c sage: t.minpoly()
x + b
```
```python
sage: t.absolute_charpoly()
x^4 - 6*x^2 + 9
sage: 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:
```
```python
sage: from sage.rings.number_field.number_field_element import is_
    -NumberFieldElement
sage: is_NumberFieldElement(2)
False
sage: k.<a> = NumberField(x^7 + 17*x + 1)
sage: is_NumberFieldElement(a+1)
True
```

### 1.5 Optimized Quadratic Number Field Elements

This file defines a Cython class `NumberFieldElement_quadratic` to speed up computations in quadratic extensions of $\mathbb{Q}$.

**AUTHORS:**

- David Harvey (2007-10): fix up a few bugs, polish around the edges
- David Loeffler (2009-05): add more documentation and tests
- Vincent Delecroix (2012-07): comparisons for quadratic number fields (trac ticket #13213), abs, floor and ceil functions (trac ticket #13256)

**Todo:** The `__new__` method should be overridden in this class to copy the $D$ and `standard_embedding` attributes

```python
class sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic:
    Bases: sage.rings.number_field.number_field_element.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()`
    Returns the ceil.

    EXAMPLES:
```
```python
sage: K.<sqrt7> = QuadraticField(7, name='sqrt7')
sage: sqrt7.ceil()
3
```
charpoly (var='x')

The characteristic polynomial of this element over \( \mathbb{Q} \).

EXAMPLES:

```
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
```

continued_fraction()

Return the (finite or ultimately periodic) continued fraction of self.

EXAMPLES:

```
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, 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: K.<sqrt2> = QuadraticField(2)
sage: sqrt2.continued_fraction_list()
((1,), (2,))
sage: (1/2+sqrt2/3).continued_fraction_list()
((0, 1, 33), (1, 32))
```

For rational entries a pair of tuples is also returned but the second one is empty:

```
sage: K(123/567).continued_fraction_list()
((0, 4, 1, 1, 1, 3, 2), ())
```

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: 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
```

**floor()**

Returns the floor of x.

**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
```

**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: 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
```

**imag()**

Returns 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
```

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
sage: K(0).is_one()
False
sage: K(1/2).is_one()
False
sage: 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:

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

minpoly(var='x')
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 $x$.

EXAMPLES:

sage: K.<a> = NumberField(x^2+13)
sage: a.minpoly()
1
sage: a.minpoly('T')
T^2 + 13
sage: (a+1/2-a).minpoly()
x - 1/2

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: K.<a> = NumberField(x^2-x+3)
sage: a.norm()
The norm is multiplicative:

```
sage: K.<a> = NumberField(x^2-3)
sage: a.norm()
-3
sage: K(3).norm()
9
sage: (3*a).norm()
-27
```

We test that the optional argument is handled sensibly:

```
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
```

**numerator()**

Return self*self.denominator().

**EXAMPLES:**

```
sage: K.<a> = NumberField(x^2+x+41)
sage: b = (2*a+1)/6
sage: b.denominator()
6
sage: b.numerator()
2*a + 1
```

**parts()**

This function returns a pair of rationals $a$ and $b$ such that $self = a + b\sqrt{D}$.

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:**

```
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: K.<a> = NumberField(x^2-x+7)
sage: a.parts()
(1/2, 3/2)
sage: a._coefficients()
[0, 1]

**real()**
Returns the real part of self, which is either self (if self lives in a totally real field) or a rational number.

**EXAMPLES:**

```python
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: 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
```

**round()**
Returns the round (nearest integer).

**EXAMPLES:**

```python
sage: K.<sqrt7> = QuadraticField(7, name='sqrt7')
sage: sqrt7.round()
3
sage: (-sqrt7).round()
-3
sage: (12/313*sqrt7 - 1745917/2902921).round()
0
sage: (12/313*sqrt7 - 1745918/2902921).round()
-1
```

**sign()**
Returns the sign of self (0 if zero, +1 if positive and -1 if negative).

**EXAMPLES:**

```python
sage: K.<sqrt2> = QuadraticField(2, name='sqrt2')
sage: K(0).sign()
0
sage: sqrt2.sign()
```

(continued on next page)
trace()

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+x+41)
sage: a.trace()
-1

sage: a.matrix()
[ 0 1]
[-41 -1]
```

The trace is additive:
class sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic

Bases: sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic

Element of an order in a quadratic field.

EXAMPLES:

```python
sage: K.<a> = NumberField(x^2 + 1)
sage: O2 = K.order(2*a)
sage: w = O2.1; w
2*a
sage: parent(w)
Order in Number Field in a with defining polynomial x^2 + 1
```

charpoly(var='x')
The characteristic polynomial of this element, which is over \( \mathbb{Z} \) because this element is an algebraic integer.

EXAMPLES:

```python
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
sage: f(b)
0
```

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: 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)
(continues on next page)
```
minpoly (var='x')

The minimal polynomial of this element over \( \mathbb{Z} \).

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 + 163)
sage: R = K.ring_of_integers()
sage: f = R(a).minpoly('x'); f
x^2 + 163
```

norm()

The norm of an element of the ring of integers is an Integer.

EXAMPLES:

```
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
```

trace()

The trace of an element of the ring of integers is an Integer.

EXAMPLES:

```
sage: K.<a> = NumberField(x^2 - 5)
sage: R = K.ring_of_integers()
sage: b = R((1+a)/2)
sage: b.trace()
1
```

class sage.rings.number_field.number_field_element_quadratic.Q_to_quadratic_field_element

Bases: sage.categories.morphism.Morphism

Morphism that coerces from rationals to elements of a quadratic number field \( K \).

EXAMPLES:

```
sage: K.<a> = QuadraticField(-3)
sage: f = K.coerce_map_from(QQ); f
Natural morphism:
  From: Rational Field
  To:   Number Field in a with defining polynomial x^2 + 3
sage: f(3/1)
```

(continues on next page)
class sage.rings.number_field.number_field_element_quadratic.Z_to_quadratic_field_element

Bases: sage.categories.morphism.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
sage: phi(4)
4
sage: phi(5).parent() is K
True
```

1.6 Splitting fields of polynomials over number fields

AUTHORS:

- Jeroen Demeyer (2014-01-02): initial version for trac ticket #2217
- Jeroen Demeyer (2014-01-03): add abort_degree argument, trac ticket #15626

class sage.rings.number_field.splitting_field.SplittingData(_pol, _dm)

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), SplittingData(x^3 + 1, 2)]
sage: [x.key() for x in L]
[(1, 2, 16), (1, 3, 729), (2, 2, 784), (2, 3, 256), (2, 3, 729)]
```

poldegree()

Return the degree of self.pol

1.6. Splitting fields of polynomials over number fields 169
EXAMPLES:

```python
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)`

Bases: `exceptions.Exception`

Special exception class to indicate an early abort of `splitting_field()`.

EXAMPLES:

```python
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
```

`sage.rings.number_field.splitting_field.splitting_field(poly, name, 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 intermediate number fields using PARI's `polred()`. 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:**
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.

```sage
domain: (x^4 - x + 1).splitting_field('a', simplify=False)
Number Field in a with defining polynomial x^24 - 2780*x^22 + 2*x^2 + 3527512*x^19 - 20 - 28766*x^18 - 2701391985*x^17 + 945948*x^16 + 1390511639677*x^15 + 73675420*x^14 - 506816498313560*x^13 + 134120588299548463*x^12 + 362240696528256*x^11 - 2596458236688039486*x^10 - 91743672243419990*x^9 + 3649424973447308439427*x^8 + 1431032927134072336*x^7 - 363192569823568746892571*x^6 - 1353403793640477725898*x^5 + 24293393281774694677256*x^4 + 706738184999341257628*x^3 + 98062147508959243128437933*x^2 - 153984144061780545432660*x + 1806591401201350260245656599
```

Reducible polynomials also work:

```sage
domain: 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
```

Relative situation:

```sage
domain: 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:

```sage
domain: L.<b>, 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
sage: (x^4 - x + 1).splitting_field('a', simplify_all=True, map=True)[1]
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
```
From: Rational Field
To: Number Field in a with defining polynomial x^24 - 3*x^23 + 2*x^22 - x^20 -
\[4*x^19 + 32*x^18 - 35*x^17 - 92*x^16 + 49*x^15 + 163*x^14 - 15*x^13 - 194*x^12 -
15*x^11 + 163*x^10 + 49*x^9 - 92*x^8 - 35*x^7 + 32*x^6 + 4*x^5 - x^4 + 2*x^3 - 3*x + 1\]
Defn: 1 |--> 1
We can enable verbose messages:

```
sage: set_verbose(2)
sage: K.<a> = (x^3 - x + 1).splitting_field()  
verbose 1 (...: splitting_field.py, splitting_field) Starting field: y
verbose 2 (...: splitting_field.py, splitting_field) Factors:
[[3, 0], [3, 3]]
verbose 2 (...: splitting_field.py, splitting_field) Done factoring (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) New field before simplifying: x^2 + 23 (time = ...)
verbose 2 (...: splitting_field.py, splitting_field) New field: y^2 - y + 6 (time = ...)
```

Try all Galois groups in degree 4. We use a quadratic base field such that `polgalois()` cannot be used:

```
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 -
\[3 - 18*x^2 - 27*x + 81\]
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 +
\[82816*x^6 + 41608*x^5 - 25586*x^4 - 60048*x^3 - 16628*x^2 + 12008*x + 34961\]
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 + 26899472*x^16 -
106792132*x^15 + 371066538*x^14 - 1127792656*x^13 + 2991524876*x^12 -
\[688328152*x^11 + 15355560064*x^10 - 230007836336*x^9 + 32544206281792*x^8 -
36347834476*x^7 + 30850889884*x^6 - 16707053128*x^5 + 105693942792*x^4 -
\[4832907784*x^3 - 3038258802*x^2 + 200383596*x + 593179173\]```
```python
sage: S4pol = x^4 + x + 1
sage: S4pol.splitting_field('x')
Number Field in x with defining polynomial x^48 ...
```

Some bigger examples:

```python
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
sage: pol48 = x^6 - 4*x^4 + 12*x^2 - 12
sage: pol48.splitting_field('a')
Number Field in a with defining polynomial x^48 ...
```

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:

```python
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:

```python
sage: pol48.splitting_field('a', degree_multiple=20)
Traceback (most recent call last):
...  
ValueError: inconsistent degree_multiple in splitting_field()
```

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: D10pol = x^5 - x^4 - 5*x^3 + 4*x^2 + 3*x - 1
sage: D10pol.splitting_field('x')
Number Field in x with defining polynomial x^10 - 28*x^8 + 216*x^6 - 681*x^4 + 902*x^2 - 401
sage: AGL_1_5pol = x^5 - 2
sage: AGL_1_5pol.splitting_field('x')
Number Field in x with defining polynomial x^20 + 10*x^19 + 55*x^18 + 210*x^17 + 595*x^16 + 1300*x^15 + 2250*x^14 + 3130*x^13 + 3855*x^12 + 3500*x^11 + 2965*x^10 + 2250*x^9 + 1625*x^8 + 1150*x^7 + 750*x^6 + 400*x^5 + 275*x^4 + 100*x^3 + 75*x^2 + 25
```

(continues on next page)
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):

```python
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:

```python
sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: try:
....:   (x^8+x+1).splitting_field('b', abort_degree=60, simplify=False)
....: except SplittingFieldAbort as e:
....:   print(e.degree_divisor)
....:   print(e.degree_multiple)
120
1440
```

### 1.7 Galois Groups of Number Fields

AUTHORS:

- David Loeffler (2009): rewrite to give explicit homomorphism groups

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:

```python
sage: K.<w> = QuadraticField(-7); G = K.galois_group()
sage: G[1]
(1,2)
sage: G[1](w + 2)
-w + 2
sage: L.<v> = NumberField(x^3 - 2); G = L.galois_group(names='y')
sage: G[4]
```
(continued from previous page)

\[(1,5) (2,4) (3,6)\]
\[
\text{sage: } G[4](v) \\
1/18*y^4
\]
\[
\text{sage: } G[4](G[4](v)) \\
-1/36*y^4 - 1/2*y
\]
\[
\text{sage: } G[4](G[4](G[4](v))) \\
1/18*y^4
\]

\text{as_hom}()\]

Return the homomorphism \(L \rightarrow L\) corresponding to self, where \(L\) is the Galois closure of the ambient number field.

\text{EXAMPLES:}\]

\[
\text{sage: } G = \text{QuadraticField(-7,'w').galois_group()} \\
\text{sage: } G[1].as_hom() \\
\text{Ring endomorphism of Number Field in w with defining polynomial x^2 + 7} \\
\text{Defn: } w \mapsto -w
\]

\text{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 \(P\).

\text{EXAMPLES:}\]

\[
\text{sage: } K.<b> = \text{NumberField(x^3 - 3, 'a').galois_closure()} \\
\text{sage: } G = K.galois_group() \\
\text{sage: } P = K.primes_above(3)[0] \\
\text{sage: } s = \text{hom(K, K, 1/18*b^4 - 1/2*b)} \\
\text{sage: } G(s).ramification_degree(P) \\
4
\]

class sage.rings.number_field.galois_group.GaloisGroup_subgroup(ambient, elts)\]

Bases: sage.rings.number_field.galois_group.GaloisGroup_v2

A subgroup of a Galois group, as returned by functions such as \text{decomposition_group}.

\text{fixed_field}()\]

Return the fixed field of this subgroup (as a subfield of the Galois closure of the number field associated to the ambient Galois group).

\text{EXAMPLES:}\]

\[
\text{sage: } L.<a> = \text{NumberField(x^4 + 1)} \\
\text{sage: } G = L.galois_group() \\
\text{sage: } H = G.decomposition_group(L.primes_above(3)[0]) \\
\text{sage: } H.fixed_field() \\
(\text{Number Field in a0 with defining polynomial x^2 + 2, Ring morphism:} \\
\text{From: Number Field in a0 with defining polynomial x^2 + 2} \\
\text{To: Number Field in a with defining polynomial x^4 + 1} \\
\text{Defn: a0 |--> a^3 + a})
\]

class sage.rings.number_field.galois_group.GaloisGroup_v1(group, number_field)\]

Bases: sage.structure.sage_object.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: K = QQ[2^(1/3)]
sage: G = K.galois_group(type="pari"); G
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the Number Field in a
˓→with defining polynomial x^3 - 2
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
```

group()

Return the underlying abstract group.

EXAMPLES:

```
sage: G = NumberField(x^3 + 2*x + 2, 'theta').galois_group(type="pari")
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)]
```

number_field()

Return the number field of which this is the Galois group.

EXAMPLES:

```
sage: G = NumberField(x^6 + 2, 't').galois_group(type="pari"); G
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
```

order()

Return the order of this Galois group.

EXAMPLES:

```
sage: G = NumberField(x^5 + 2, 'theta_1').galois_group(type="pari"); G
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
```

class sage.rings.number_field.galois_group.GaloisGroup_v2(number_field, names=None)

Bases: sage.groups.perm_gps.permgroup.PermutationGroup_generic

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.

**Element alias of** *GaloisGroupElement*

**artin_symbol(P)**

Return the Artin symbol \( \left( \frac{K/Q}{P} \right) \), where K is the number field of self, and \( \mathfrak{P} \) is an unramified prime ideal. This is the unique element \( s \) of the decomposition group of \( \mathfrak{P} \) such that \( s(x) = x^p \mod \mathfrak{P} \), where \( p \) is the residue characteristic of \( \mathfrak{P} \).

**EXAMPLES:**

```python
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)]
[(1,5)(2,6)(3,7)(4,8), (1,5)(2,6)(3,7)(4,8), (1,4)(2,3)(5,8)(6,7), (1,4)(2,
  3)(5,8)(6,7)]
sage: G.artin_symbol(17)
Traceback (most recent call last):
  ... ValueError: Fractional ideal (17) is not prime
```

**complex_conjugation**(\(P=None\))

Return the unique element of 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:**

```python
sage: L.<z> = CyclotomicField(7)
sage: G = L.galois_group()
sage: conj = G.complex_conjugation(); conj
(1,4)(2,5)(3,6)
sage: conj(z)
-z^5 - z^4 - z^3 - z^2 - z - 1
```

An example where the field is not CM, so complex conjugation really depends on the choice of embedding:

```python
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:

```sage
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 [((), (1,8)(2,7)(3,6)(4,5)] of Galois group of Number Field in a with defining polynomial 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 prime
sage: G.decomposition_group(17)
Traceback (most recent call last):
  ...  
ValueError: Fractional ideal (17) is not prime
```

An example with an infinite place:

```sage
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
```

**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
sage: K.<b> = NumberField(x^2 - 3,'a')
sage: G = K.galois_group()
sage: G.inertia_group(K.primes_above(2)[0])
Galois group of Number Field in b with defining polynomial x^2 - 3
sage: G.inertia_group(K.primes_above(5)[0])
Subgroup [(()) of Galois group of Number Field in b with defining polynomial x^2 - 3
```

**is_galois()**

Return True if the underlying number field of self is actually Galois.

EXAMPLES:

```sage
sage: NumberField(x^3 - x + 1,'a').galois_group(names='b').is_galois()
False
sage: NumberField(x^2 - x + 1,'a').galois_group().is_galois()
True
```

**list()**

List of the elements of self.

EXAMPLES:

```sage
sage: NumberField(x^3 - 3*x + 1,'a').galois_group().list()
[(), (1,2,3), (1,3,2)]
```

**ngens()**

Number of generators of self.

EXAMPLES:
number_field()

The ambient number field.

EXAMPLES:

```python
sage: K = NumberField(x^3 - x + 1, 'a')
sage: K.galois_group(names='b').number_field() is K
True
```

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:

```python
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
```

ramification_group(P, v)

Return the vth 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:

```python
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 [(), (1,2,4)(3,5,6), (1,4,2)(3,6,5)] of Galois group of Number Field in b with defining polynomial x^6 + 243
sage: G.ramification_group(P, 5)
Subgroup [()] of Galois group of Number Field in b with defining polynomial x^6 + 243
```

splitting_field()

The Galois closure of the ambient number field.

EXAMPLES:

```python
sage: K = NumberField(x^3 - x + 1, 'a')
sage: K.galois_group(names='b').splitting_field()
Number Field in b with defining polynomial x^6 - 6*x^4 + 9*x^2 + 23
sage: L = QuadraticField(-23, 'c'); L.galois_group().splitting_field() is L
True
```

subgroup(els)

Return the subgroup of self with the given elements. Mostly for internal use.

EXAMPLES:
Sage functions to list all elements of a given number field with height less than a specified bound.

AUTHORS:

- John Doyle (2013): initial version
- David Krumm (2013): initial version
- TJ Combs (2018): added Doyle-Krumm algorithm - 4
- Raghukul Raman (2018): added Doyle-Krumm algorithm - 4

REFERENCES:

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 $\text{height\_bound}$. 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 [Doyle-Krumm].

INPUT:

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

OUTPUT:

- an iterator of number field elements
EXAMPLES:

There are no elements of negative height:

```
sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = NumberField(x^5 - x + 7)
sage: list(bdd_height(K,-3))
[]
```

The only nonzero elements of height 1 are the roots of unity:

```
sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = QuadraticField(3)
sage: list(bdd_height(K,1))
[0, -1, 1]
```

```
sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = QuadraticField(36865)
sage: len(list(bdd_height(K,101)))  # long time (4 s)
131
```

```
sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = NumberField(x^6 + 2)
sage: len(list(bdd_height(K,60)))  # long time (5 s)
1899
```

```
sage: from sage.rings.number_field.bdd_height import bdd_height
sage: K.<g> = NumberField(x^4 - x^3 - 3*x^2 + x + 1)
sage: len(list(bdd_height(K,10)))
99
```

```
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 \( \text{height\_bound} \).

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 [Doyle-Krumm].

INPUT:

- \( K \) – an imaginary quadratic number field
- \( \text{height\_bound} \) – a real number

OUTPUT:

- an iterator of number field elements

EXAMPLES:

```
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + 191)
sage: for t in bdd_height_iq(K,8):
    ....:     print(exp(2*t.global_height()))
1.00000000000000
```

(continues on next page)
There are 175 elements of height at most 10 in \( \mathbb{Q}(\sqrt{-3}) \):

```python
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:

```python
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:

```python
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))
[]
```

sage.rings.number_field.bdd_height.bdd_norm_pr_gens_iq(K, norm_list)
Compute generators for all principal ideals in an imaginary quadratic field \( K \) whose norms are in norm_list.
The only keys for the output dictionary are integers \( n \) appearing in 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
- 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:
Sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens_iq
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: bdd_pr_ideals[5]
[-g + 2, -g + 2]
Sage: bdd_pr_ideals[7]
[]

There are no ideals in the ring of integers with negative norm:

Sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_gens_iq
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: [], -4: [], -3: [], -2: []}

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_iq
Sage: K.<g> = NumberField(x^2 + 20)
Sage: L = range(100)
Sage: bdd_pr_ideals = bdd_norm_pr_gens_iq(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 norm_list.

INPUT:
  • K – a number field
  • 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.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]}
sage: from sage.rings.number_field.bdd_height import bdd_norm_pr_ideal gens
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]

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 matrix and a real number interval_radius, this function finds all integer lattice points in the polytope obtained by transforming the cube [-interval_radius,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:

sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([2])
sage: r = 1
sage: integer_points_in_polytope(m,r)
[(-2), (-1), (0), (1), (2)]

Integer points inside a parallelogram:

sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([[1, 2],[3, 4]])
sage: r = RealField()(1.3)

sage: integer_points_in_polytope(m,r)
[(-3, -7), (-2, -5), (-2, -4), (-1, -3), (-1, -2), (-1, -1), (0, -1), (0, 0), (0, 1), (1, 1), (1, 2), (1, 3), (2, 4), (2, 5), (3, 7)]

Integer points inside a parallelepiped:

sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([[1.2,3.7,0.2],[-5.3,-.43,3],[1.2,4.7,-2.1]])
sage: r = 2.2

sage: L = integer_points_in_polytope(m,r)
sage: len(L)
4143

If interval_radius is 0, the output should include only the zero tuple:

sage: from sage.rings.number_field.bdd_height import integer_points_in_polytope
sage: m = matrix([[1,2,3,7],[4,5,6,2],[7,8,9,3],[0,3,4,5]])
sage: integer_points_in_polytope(m,0)
[(0, 0, 0, 0)]
CHAPTER TWO

MORPHISMS

2.1 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: sage.rings.number_field.morphism.NumberFieldHomomorphism_im_gens

class sage.rings.number_field.morphism.CyclotomicFieldHomset(R, S, category=None):
    Bases: sage.rings.number_field.morphism.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

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: 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))
[]

class sage.rings.number_field.morphism.NumberFieldHomomorphism_im_gens:
    Bases: sage.rings.morphism.RingHomomorphism_im_gens
```
preimage(y)
Computes a preimage of \( y \) in the domain, provided one exists. Raises a 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 ValueError if \( y \) has no preimage.

EXAMPLES:

```sage
K.<a> = NumberField(x^2 - 7)
L.<b> = NumberField(x^4 - 7)
f = K.embeddings(L)[0]
f.preimage(3*b^2 - 12/7)
3*a - 12/7
f.preimage(b)
Traceback (most recent call last):
... ValueError: Element 'b' is not in the image of this homomorphism.
```

```sage
F.<b> = QuadraticField(23)
G.<a> = F.extension(x^3+5)
f = F.embeddings(G)[0]
f.preimage(a^3+2*b+3)
2*b - 2
```

```sage
K.<a> = NumberField(x^2 + 1)
End(k)
Automorphism group of Number Field in a with defining polynomial x^2 + 1
End(k).order()
2
k.<a> = NumberField(x^3 + 2)
End(k).order()
1
K.<a> = NumberField([x^3 + 2, x^2 + x + 1 ])
End(K).order()
6
```

list()
Return a list of all the elements of self.

EXAMPLES:

```sage
K.<a> = NumberField(x^3 - 3*x + 1)
End(K).list()
[ ]
```
Ring endomorphism of Number Field in a with defining polynomial $x^3 - 3*x + 1$
Defn: $a \mapsto a$,

Ring endomorphism of Number Field in a with defining polynomial $x^3 - 3*x + 1$
Defn: $a \mapsto a^2 - 2$,

Ring endomorphism of Number Field in a with defining polynomial $x^3 - 3*x + 1$
Defn: $a \mapsto -a^2 - a + 2$

```
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:

```
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 |--> 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 |--> -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 |--> 1/2*a*b - 1/2*a
]
```

```
sage: order() # indirect doctest

Return the order of this set of field homomorphism.

EXAMPLES:

```
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
```

```
class sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs

Bases: sage.rings.morphism.RingHomomorphism

A homomorphism from a relative number field to some other ring, stored as a homomorphism from the corresponding absolute field.

2.1. Morphisms between number fields

187
abs_hom()

Return the corresponding homomorphism from the absolute number field.

EXAMPLES:

```
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
```

im_gens()

Return the images of the generators under this map.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^3 + 2, x^2 + x + 1])
sage: K.hom(a, K).im_gens()
[a, b]
```

class `<module>`

```
class sage.rings.number_field.morphism.RelativeNumberFieldHomset(R, S, category=None)

Bases: sage.rings.number_field.morphism.NumberFieldHomset

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:

```
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:

```
sage: phi(phi(cuberoot2 + zeta3))
(-zeta3 - 1)*cuberoot2 + zeta3
sage: phi(phi(phi(cuberoot2 + zeta3)))
cuberoot2 + zeta3
```

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: 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 |--> 1/70*c^3 - 1/35*c^2 - 1/70*c + 1/35
```
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: 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:

```
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 |--> z^6 - 2*z^2
 b |--> -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 |--> -z^6 + 2*z^2
 b |--> z^5 + z^3 - z]
```

## 2.2 Embeddings into ambient fields

This module provides classes to handle embeddings of number fields into ambient fields (generally R or C).

```
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.

class sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldConversion
    Bases: sage.categories.map.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 C).
```

### 2.2. Embeddings into ambient fields 189
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: sage.rings.number_field.number_field_morphisms.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;
- C.

EXAMPLES:

```
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:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i -> -i
sage: EmbeddedNumberFieldMorphism(K,L,QQbar)
Generic morphism:
  From: Number Field in i with defining polynomial x^2 + 1
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i -> -i
```

ambient_field

section()

EXAMPLES:

```
sage: from sage.rings.number_field.number_field_morphisms import EmbeddedNumberFieldMorphism
sage: K.<a> = NumberField(x^2-700, embedding=25)
sage: L.<b> = NumberField(x^6-700, embedding=3)
sage: f = EmbeddedNumberFieldMorphism(K, L)
sage: f(2*a-1)
2*b^3 - 1
sage: g = f.section()
sage: g(2*b^3-1)
2*a - 1
```

class sage.rings.number_field.number_field_morphisms.NumberFieldEmbedding

Bases: sage.categories.morphism.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
```

(continues on next page)
sage: K.<a> = NumberField(x^3-2)
sage: f = NumberFieldEmbedding(K, RLF, 1)
sage: f(a)^3
2.00000000000000
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

gen_image()
Returns the image of the generator under this embedding.

EXAMPLES:

sage: f = QuadraticField(7, 'a', embedding=2).coerce_embedding()
sage: f.gen_image()
2.645751311064591?

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 \texttt{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 \* \texttt{abs}(target-\(x\)) < \texttt{abs}(target-\(y\)).

sage.rings.number_field.number_field_morphisms.create_embedding_from_approx(K, gen_image)
Return an embedding of \(K\) determined by \texttt{gen_image}.

The codomain of the embedding is the parent of \texttt{gen_image} or, if \texttt{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 \texttt{gen_image}.

EXAMPLES:

We can define embeddings from one number field to another:

2.2. Embeddings into ambient fields
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:

sage: create_embedding_from_approx(K, b)
Traceback (most recent call last):
...  
ValueError: b is not a root of x^3 - x + 1/10

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, this function chooses 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 code{abs} function) in the ambient field to the target, and return the root of poly (if any) that approximates it best.

EXAMPLES:

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: matching_root(x^2-2, .000000001)
1.41421356237310
sage: matching_root(x^3-3-1, CDF.0)
-0.5000000000000000 + 0.866025403784439*I
sage: matching_root(x^3-x, 2, ambient_field=RR)
1.00000000000000

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:

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
The `root_from_approx` function in Sage allows approximating roots of polynomials.

```sage
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 - 2, sqrt(2))
sqrt(2)
sage: root_from_approx(x^2 - 2, sqrt(3))
Traceback (most recent call last):
  ... ValueError: sqrt(3) is not a root of x^2 - 2
```

### 2.3 Structure maps for number fields

Provides isomorphisms between relative and absolute presentations, to and from vector spaces, name changing maps, etc.

**EXAMPLES:**

```sage
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*a^5 + 6*a^4 - 11*a^3 + 12*a^2 + 3*a + 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*a^5 + 6*a^4 - 11*a^3 + 12*a^2 + 3*a + 1
```

**class** `sage.rings.number_field.maps.MapAbsoluteToRelativeNumberField(A, R)`

Bases: `sage.rings.number_field.maps.NumberFieldIsomorphism`

See `MapRelativeToAbsoluteNumberField` for examples.

**class** `sage.rings.number_field.maps.MapNumberFieldToVectorSpace(K, V)`

Bases: `sage.categories.map.Map`

A class for the isomorphism from an absolute number field to its underlying \(\mathbb{Q}\)-vector space.

**EXAMPLES:**

```sage
sage: L.<a> = NumberField(x^3 - x + 1)
sage: V, fr, to = L.vector_space()
```

(continues on next page)
class sage.rings.number_field.maps.MapRelativeNumberFieldToRelativeVectorSpace(K, V)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

EXAMPLES:

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'>

class sage.rings.number_field.maps.MapRelativeNumberFieldToVectorSpace(L, V, to_K, to_V)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

The isomorphism from a relative number field to its underlying \( \mathbb{Q} \)-vector space. Compare MapRelativeNumberFieldToRelativeVectorSpace.

EXAMPLES:

sage: K.<a> = NumberField(x^8 + 100*x^6 + x^2 + 5)

sage: L = K.relativize(K.subfields(4)[0][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: 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)

sage: to(L.gen()), fr(to(L.gen())) == L.gen()
((0, 1, 0, 0, 0, 0, 0, 0), True)

class sage.rings.number_field.maps.MapRelativeToAbsoluteNumberField(R, A)

Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

EXAMPLES:

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
Relative number field morphism:
From: Number Field in b with defining polynomial $x^2 + a_0$ over its base field
To: Number Field in a with defining polynomial $x^6 + 4x^2 + 200$
Defn: $b \mapsto a$
$a_0 \mapsto -a^2$

Ring morphism:
From: Number Field in a with defining polynomial $x^6 + 4x^2 + 200$
To: Number Field in b with defining polynomial $x^2 + a_0$ over its base field
Defn: $a \mapsto b$

class sage.rings.number_field.maps.MapRelativeVectorSpaceToRelativeNumberField($V$, $K$)
Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

EXAMPLES:

```python
sage: L.<b> = NumberField(x^4 + 3x^2 + 1)
sage: K = L.relativize(L.subfields(2)[0][1], 'a'); K
Number Field in a with defining polynomial $x^2 - b_0x + 1$ 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: 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
cpysage: (to + fr)(V([1, 2])) == V([1, 2])
True

class sage.rings.number_field.maps.MapVectorSpaceToNumberField(V, K)
Bases: sage.rings.number_field.maps.NumberFieldIsomorphism

The map to an absolute number field from its underlying \(\mathbb{Q}\)-vector space.

EXAMPLES:

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
 \longrightarrow 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
 \longrightarrow defining polynomial x^4 + 3*x + 1)
sage: fr * to
Composite map:
  From: Number Field in a with defining polynomial x^4 + 3*x + 1
  To:   Number Field in a with defining polynomial x^4 + 3*x + 1
  Defn: Isomorphism map:
    From: Number Field in a with defining polynomial x^4 + 3*x + 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 + 3*x + 1
sage: to * fr
Composite map:
  From: Vector space of dimension 4 over Rational Field
  To:   Number Field in a with defining polynomial x^4 + 3*x + 1
(continues on next page)
To: Vector space of dimension 4 over Rational Field
Defn: Isomorphism map:
  From: Vector space of dimension 4 over Rational Field
  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

\[
\text{sage: } \text{to}(a), \text{to}(a + 1)
\]
\[
(0, 1, 0, 0), (1, 1, 0, 0)
\]
\[
\text{sage: } \text{fr} \left( \text{to}(a) \right), \text{fr}(V([0, 1, 2, 3]))
\]
\[
(a, 3a^3 + 2a^2 + a)
\]

**class** sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField(*V, L, from_V, from_K*)

**Bases:** sage.rings.number_field.maps.NumberFieldIsomorphism

The isomorphism to a relative number field from its underlying \( \mathbb{Q} \)-vector space. Compare MapRelativeVectorSpaceToRelativeNumberField.

**EXAMPLES:**

\[
\text{sage: } L.<a, b> = \text{NumberField}([x^2 + 3, x^2 + 5])
\]
\[
\text{sage: } V, \text{fr}, \text{to} = L.\text{absolute_vector_space()}
\]
\[
\text{sage: } \text{type}(\text{fr})
\]
\[
<\text{class 'sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField'>>
\]

**class** sage.rings.number_field.maps.NameChangeMap(*K, L*)

**Bases:** sage.rings.number_field.maps.NumberFieldIsomorphism

A map between two isomorphic number fields with the same defining polynomial but different variable names.

**EXAMPLES:**

\[
\text{sage: } K.<a> = \text{NumberField}(x^2 - 3)
\]
\[
\text{sage: } L.<b> = K.\text{change_names()}
\]
\[
\text{sage: } \text{from}_L, \text{to}_L = L.\text{structure()}
\]
\[
\text{sage: } \text{from}_L.
\]

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 \( x^2 - 3 \)
\[
\text{sage: } \text{to}_L.
\]

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 \( x^2 - 3 \)
\[
\text{sage: } \text{type}(\text{from}_L), \text{type}(\text{to}_L)
\]
\[
(<\text{class 'sage.rings.number_field.maps.NameChangeMap'>, <\text{class 'sage.rings.number_field.maps.NameChangeMap'>})
\]

**class** sage.rings.number_field.maps.NumberFieldIsomorphism

**Bases:** sage.categories.map.Map

A base class for various isomorphisms between number fields and vector spaces.

**EXAMPLES:**

2.3. Structure maps for number fields
is_injective()

EXAMPLES:

```sage
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_injective()
```

is_surjective()

EXAMPLES:

```sage
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_surjective()
```

### 2.4 Helper classes for structural embeddings and isomorphisms of number fields

AUTHORS:

- Julian Rueth (2014-04-03): initial version

Consider the following fields \( L \) and \( M \):

```sage
sage: L.<a> = QuadraticField(2)
sage: M.<a> = L.absolute_field()
```

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,
 Identity endomorphism of Number Field in a with defining polynomial x^2 - 2)
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,
  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)
```

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:
class sage.rings.number_field.structure.AbsoluteFromRelative(other)
    Bases: sage.rings.number_field.structure.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: sage.rings.number_field.structure.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.

class sage.rings.number_field.structure.NumberFieldStructure(other)
    Bases: sage.structure.unique_representation.UniqueRepresentation

    Abstract base class encapsulating information about a number fields relation to other number fields.

    create_structure(field)
    Return a tuple encoding structural information about field.

    OUTPUT:
    Typically, the output is a pair of morphisms. The first one from field to a field from which field has been constructed and the second one its inverse. In this case, these morphisms are used as conversion maps between the two fields.

class sage.rings.number_field.structure.RelativeFromAbsolute(other, gen)
    Bases: sage.rings.number_field.structure.NumberFieldStructure

    Structure for a relative number field created from an absolute number field.

    INPUT:
    • other – the (absolute) number field from which this field has been created.
    • gen – the generator of the intermediate field

create_structure(field)
    Return a pair of isomorphisms which go from field to other and vice versa.

    INPUT:
    • field – a relative number field

class sage.rings.number_field.structure.RelativeFromRelative(other)
    Bases: sage.rings.number_field.structure.NumberFieldStructure

2.4. Helper classes for structural embeddings and isomorphisms of number fields 199
Structure for a relative number field created from another relative number field.

INPUT:

• *other* – the relative number field used in the construction, see `create_structure()`; there this field will be called `field_`.

`create_structure(field)`

Return a pair of isomorphisms which go from `field` to the relative number field (called `other` below) from which `field` has been created and vice versa.

The isomorphism is created via the relative number field `field_` which is identical to `field` but is equipped with an isomorphism to an absolute field which was used in the construction of `field`.

INPUT:

• *field* – a relative number field
3.1 Orders in Number Fields

AUTHORS:


EXAMPLES:

We define an absolute order:

```python
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:

```python
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3]); 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:

```python
sage: K.<a> = NumberField((x+1)^10 + 17)
sage: K.maximal_order()
Maximal Order 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
```

We compute a suborder, which has index a power of 17 in the maximal order:

```python
sage: O = K.order(17*a); O
Order 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
```

```python
class sage.rings.number_field.order.AbsoluteOrder(K, module_rep, is_maximal=None, check=True)
    Bases: sage.rings.number_field.order.Order

EXAMPLES:
```
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
Order 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.
sage: loads(dumps(O)) == O
True

Quadradic elements have a special optimized type:

**absolute_discriminant()**

Return the discriminant of this order.

EXAMPLES:

```sage
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
```

**absolute_order()**

Return the absolute order associated to this order, which is just this order again since this is an absolute order.

EXAMPLES:

```sage
sage: K.<a> = NumberField(x^3 + 2)
sage: O1 = K.order(a); O1
Order 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
sage: k.<c> = NumberField(x^3 + x^2 + 1)
sage: O = k.maximal_order(); O
Maximal Order in Number Field in c with defining polynomial x^3 + x^2 + 1
sage: O.basis()
[1, c, c^2]
```
The basis is an immutable sequence:

```
sage: type(O.basis())
<class 'sage.structure.sequence.Sequence_generic'>
```

The generator functionality uses the basis method:

```
sage: O.0
1
sage: O.1
c
sage: O.basis()
[1, c, c^2]
sage: O.ngens()
3
```

`change_names(names)`

Return a new order isomorphic to this one in the number field with given variable names.

**EXAMPLES:**

```
sage: R = EquationOrder(x^3 + x + 1, 'alpha'); R
Order 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 in Number Field in gamma with defining polynomial x^3 + x + 1
sage: S.basis()
[1, gamma, gamma^2]
```

`discriminant()`

Return the discriminant of this order.

**EXAMPLES:**

```
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
```

`index_in(other)`

Return the index of self in other. This is a lattice index, so it is a rational number if self isn’t contained in other.

**INPUT:**

- `other` – another absolute order with the same ambient number field.

**OUTPUT:**

a rational number

**EXAMPLES:**

### 3.1. Orders in Number Fields
intersection (other)

Return the intersection of this order with another order.

EXAMPLES:

```python
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]
```

module()

Returns the underlying free module corresponding to this order, embedded in the vector space corresponding to the ambient number field.

EXAMPLES:

```python
sage: k.<a> = NumberField(x^3 + x + 3)
sage: m = k.order(3*a); m
Order in Number Field in a with defining polynomial x^3 + x + 3
sage: m.module()
Free module of degree 3 and rank 3 over Integer Ring
Echelon basis matrix:
[1 0 0]
[0 3 0]
[0 0 9]
```
Return the ring of Eisenstein integers, that is all complex numbers of the form $a + b\omega$ with $a$ and $b$ integers and $\omega = (-1 + \sqrt{-3})/2$.

**EXAMPLES:**

```python
sage: R.<omega> = EisensteinIntegers()
sage: R
Eisenstein Integers in Number Field in omega with defining polynomial x^2 + x + 1
sage: factor(3 + omega)
(omega) * (-3*omega - 2)
sage: CC(omega)
-0.500000000000000 + 0.866025403784439*I
sage: omega.minpoly()
x^2 + x + 1
sage: EisensteinIntegers().basis()
[1, omega]
```

Return the equation order generated by a root of the irreducible polynomial $f$ or list of polynomials $f$ (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:**

```python
sage: O.<a,b> = EquationOrder([x^2+1, x^2+2])
sage: O
Relative Order 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
```

Of course the input polynomial must be integral:

```python
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
```

Return the ring of Gaussian integers, that is all complex numbers of the form $a + bI$ with $a$ and $b$ integers and $I = \sqrt{-1}$.

**EXAMPLES:**

```python
sage: ZZI.<I> = GaussianIntegers()
sage: ZZI
Gaussian Integers in Number Field in I with defining polynomial x^2 + 1
sage: factor(3 + I)
(-I) * (I + 1) * (2*I + 1)
```

(continues on next page)
sage: CC(I)
1.00000000000000*I
sage: I.minpoly()
x^2 + 1
sage: GaussianIntegers().basis()
[1, I]

class sage.rings.number_field.order.Order($K$, is_maximal)

Bases: sage.rings.ring.IntegralDomain

An order in a number field.

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:

sage: K.<theta> = NumberField(x^4 + x + 17)
sage: K.maximal_order()
Maximal Order in Number Field in theta with defining polynomial x^4 + x + 17
sage: R = K.order(17*theta); R
Order in Number Field in theta with defining polynomial x^4 + x + 17
sage: R.basis()
[1, 17*theta, 289*theta^2, 4913*theta^3]
sage: R = K.order(17*theta, 13*theta); R
Order in Number Field in theta with defining polynomial x^4 + x + 17
sage: R.basis()
[1, theta, theta^2, theta^3]
sage: R = K.order([34*theta, 17*theta + 17]); R
Order in Number Field in theta with defining polynomial x^4 + x + 17
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()

Returns the absolute degree of this order, ie the degree of this order over \(\mathbb{Z}\).

EXAMPLES:

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 self.number_field() and self.fraction_field()?

EXAMPLES:

sage: k.<z> = NumberField(x^2 - 389)
sage: o = k.order(389*z + 1)
sage: o
Order in Number Field in \( z \) with defining polynomial \( x^2 - 389 \)
\[
\text{sage: } o.basis() \\
[1, 389 \times z]
\]
\[
\text{sage: } o.ambient() \\
\text{Number Field in } z \text{ with defining polynomial } x^2 - 389
\]

\textbf{basis()}

Return a basis over \( \mathbb{Z} \) of this order.

\textbf{EXAMPLES:}

\[
\text{sage: } K.<a> = NumberField(x^3 + x^2 - 16 \times x + 16) \\
\text{sage: } O = K.maximal_order(); O \\
\text{Maximal Order in Number Field in } a \text{ with defining polynomial } x^3 + x^2 - 16 \times x + 16
\]
\[
\text{sage: } O.basis() \\
[1, 1/4 \times a^2 + 1/4 \times a, a^2]
\]

\textbf{class_group}(\textit{proof}=None, \textit{names}='c')

Return the class group of this order.

(Currently only implemented for the maximal order.)

\textbf{EXAMPLES:}

\[
\text{sage: } k.<a> = NumberField(x^2 + 5077) \\
\text{sage: } O = k.maximal_order(); O \\
\text{Maximal Order in Number Field in } a \text{ with defining polynomial } x^2 + 5077
\]
\[
\text{sage: } O.class_group() \\
\text{Class group of order } 22 \text{ with structure } C_{22} \text{ of Number Field in } a \text{ with defining polynomial } x^2 + 5077
\]

\textbf{class_number}(\textit{proof}=None)

Return the class number of this order.

\textbf{EXAMPLES:}

\[
\text{sage: } \mathbb{Z}[2^{(1/3)}].class_number() \\
1
\]
\[
\text{sage: } \mathbb{Q}[\sqrt{-23}].maximal_order().class_number() \\
3
\]
\[
\text{sage: } \mathbb{Z}[120 \times \sqrt{-23}].class_number() \\
288
\]

Note that non-maximal orders are only supported in quadratic fields:

\[
\text{sage: } \mathbb{Z}[120 \times \sqrt{-23}].class_number() \\
288
\]
\[
\text{sage: } \mathbb{Z}[100 \times \sqrt{3}].class_number() \\
4
\]
\[
\text{sage: } \mathbb{Z}[11 \times 2^{(1/3)}].class_number() \\
\text{Traceback (most recent call last):} \\
... \\
\text{NotImplementedError: computation of class numbers of non-maximal orders not in quadratic fields is not implemented}
\]

\textbf{coordinates}(\textit{x})

Return the coordinate vector of \( x \) with respect to this order.

\textbf{3.1. Orders in Number Fields}
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
\((23, -14)\)
```

```python
sage: sum([OK_basis[j]*acoords[j] for j in range(2)]) == a
True
```

```python
sage: OK.coordinates((120+340*i)/8)
\((15, 85/2)\)
```

```python
sage: O = K.order(3*i)
sage: O.is_maximal()
False
sage: O.index_in(OK)
3
```

```python
sage: acoords = O.coordinates(a); acoords
\((23, -14/3)\)
```

```python
sage: sum([O.basis()[j]*acoords[j] for j in range(2)]) == a
True
```

\textbf{degree()}

Return the degree of this order, which is the rank of this order as a \( \mathbb{Z} \)-module.

EXAMPLES:

```python
sage: k.<c> = NumberField(x^3 + x^2 - 2*x + 8)
sage: o = k.maximal_order()
sage: o.degree()
3
```

```python
sage: o.rank()
3
```

\textbf{fraction_field()}

Return the fraction field of this order, which is the ambient number field.

EXAMPLES:

```python
sage: K.<b> = NumberField(x^4 + 17*x^2 + 17)
sage: O = K.order(17*b); O
Order in Number Field in b with defining polynomial x^4 + 17*x^2 + 17
```
sage: O.fraction_field()
Number Field in b with defining polynomial x^4 + 17*x^2 + 17

fractional_ideal(*args, **kwds)
Return the fractional ideal of the maximal order with given generators.

EXAMPLES:

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 Z-module contained in the vector space associated to the ambient number field, that corresponds to this ideal.

EXAMPLES:

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]

An example in a relative extension. Notice that the module is a Z-module in the absolute_field associated to the relative field:

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]

gen(i)
Return i'th module generator of this order.

EXAMPLES:

sage: K.<c> = NumberField(x^3 + 2*x + 17)
sage: O = K.maximal_order(); O
Maximal Order 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

(continues on next page)
ideal(*args,**kwds)
Return the integral ideal with given generators.

EXAMPLES:

```
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):
  ... IndexError: 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 = K.order(4*a)
sage: R.ideal(8)
Traceback (most recent call last):
  ... NotImplementedError: ideals of non-maximal orders not yet supported.
```

This function is called implicitly below:

```
sage: R = EquationOrder(x^2 + 2, 'a'); R
Order in Number Field in a with defining polynomial x^2 + 2
sage: (3,15)*R
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
```

integral_closure()
Return the integral closure of this order.

EXAMPLES:

```
sage: K.<a> = QuadraticField(5)
sage: O2 = K.order(2*a); O2
Order in Number Field in a with defining polynomial x^2 - 5
sage: O2.integral_closure()
Maximal Order in Number Field in a with defining polynomial x^2 - 5
sage: OK = K.maximal_order()
sage: OK is OK.integral_closure()
True
```

is_field(proof=True)
Return False (because an order is never a field).
### EXAMPLES:

```python
sage: L.<alpha> = NumberField(x**4 - x**2 + 7)
sage: O = L.maximal_order() ; O.is_field()
False
sage: CyclotomicField(12).ring_of_integers().is_field()
False
```

#### `is_integrally_closed()`
Return True if this ring is integrally closed, i.e., is equal to the maximal order.

**EXAMPLES:**

```python
sage: K.<a> = NumberField(x^2 + 189*x + 394)
sage: R = K.order(2*a)
sage: R.is_integrally_closed()
False
sage: R
Order in Number Field in a with defining polynomial x^2 + 189*x + 394
sage: S = K.maximal_order(); S
Maximal Order in Number Field in a with defining polynomial x^2 + 189*x + 394
sage: S.is_integrally_closed()
True
```

#### `is_maximal()`
Return True if this is the maximal order.

**EXAMPLES:**

```python
sage: k.<i> = NumberField(x^2 + 1)
sage: O3 = k.order(3*i); O5 = k.order(5*i); Ok = k.maximal_order(); Osum = ...
```

An example involving a relative order::

```python
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3]); O = K.order([3*a,2*b]); O.is_noetherian()
False
```

#### `is_noetherian()`
Return True (because orders are always Noetherian)

**EXAMPLES:**

```python
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)
```

(continues on next page)
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:**

```sage
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)
sage: O15.is_suborder(O5)
True
sage: O5.is_suborder(O15)
False
sage: O10.is_suborder(O15)
False
```

We create another isomorphic but different field:

```sage
sage: W2.<j> = NumberField(x^2 + 1)
sage: P5 = W2.order(5*j)
```

This is False because the ambient number fields are not equal:

```sage
sage: O5.is_suborder(P5)
False
```

We create a field that contains (in no natural way!) W, and of course again is_suborder returns False:

```sage
sage: K.<z> = NumberField(x^4 + 1)
sage: M = K.order(5*z)
sage: O5.is_suborder(M)
False
```

**krull_dimension** ()

Return the Krull dimension of this order, which is 1.

**EXAMPLES:**

```sage
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
```

**ngens** ()

Return the number of module generators of this order.

**EXAMPLES:**

```sage
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: K.<b> = NumberField(x^4 + x^2 + 2)
sage: O = K.order(2*b); O
Order 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 $\mathbb{Z}$-linear combination of the basis.

EXAMPLES:

```python
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
sage: K.<z> = CyclotomicField(17)
sage: OK = K.ring_of_integers()
sage: OK.random_element()  # random output
z^15 - z^11 - z^10 - 4*z^9 + z^8 + 2*z^7 + z^6 - 2*z^5 - z^4 - 445*z^3 - 2*z^2 - 15*z - 2
sage: OK.random_element().is_integral()  # random output
True
sage: OK.random_element().parent() is OK  # random output
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
```
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
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: k.<c> = NumberField(x^5 + x^2 + 1)
sage: o = k.maximal_order(); o
Maximal Order in Number Field in c with defining polynomial x^5 + x^2 + 1
sage: o.rank()
5
```

```python
residue_field(prime, names=None, check=False)
```

Return the residue field of this order at a given prime, ie \( O/pO \).

**INPUT:**

- `prime` – a prime ideal of the maximal order in this number field.
- `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: OK = K.maximal_order()
sage: OK.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)
sage: Fp.<b> = OK.residue_field(P)
sage: Fp
Residue field in b of Fractional ideal (61, a^2 + 30)
```

```python
ring_generators()
```

Return generators for self as a ring.

**EXAMPLES:**
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()
[1/2*a^2 + 1/2*a, a^2]
```

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()
sage: O.ring_generators()
[(-5/3*b^2 + 3*b - 2)*a - 7/3*b^2 + b + 3, (-5*b^2 - 9)*a - 5*b^2 - b, (-6*b^2 - 11)*a - 6*b^2 - b]
```

### some_elements()

Return a list of elements of the given order.

**EXAMPLES:**

```python
sage: G = GaussianIntegers(); G
Gaussian Integers in Number Field in I with defining polynomial x^2 + 1
sage: G.some_elements()
[1, I, 2*I, -1, 0, -I, 2, 4*I, -2, -2*I, -4]
sage: R.<t> = QQ[]
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 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:**

The valuation can be specified with an integer prime that is completely ramified or unramified:

```python
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
sage: GaussianIntegers().valuation(3)
3-adic valuation
```
A prime \( p \) that factors into pairwise distinct factors, results in an error:

```sage
GaussianIntegers().valuation(5)
```

```
Traceback (most recent call last):
...
ValueError: The valuation Gauss valuation induced by 5-adic valuation does
\[\not\approx\]
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).ring_of_integers().valuation(ZZ.valuation(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
\[\not\approx\]
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:

```sage
R.<x> = QQ[]
v = GaussianIntegers().valuation(GaussValuation(R, QQ.valuation(5)).
\[\not\approx\]
augmentation(x + 2, infinity))
w = GaussianIntegers().valuation(GaussValuation(R, QQ.valuation(5)).
\[\not\approx\]
augmentation(x + 1/2, infinity))
v == w
```

```
False
```

See also:

- `NumberField_generic.valuation()`, `pAdicGeneric.valuation()

\texttt{zeta}(n=2, all=False)

Return a primitive n-th root of unity in this order, if it contains one. If all is True, return all of them.

EXAMPLES:

```sage
F.<alpha> = NumberField(x**2+3)
F.ring_of_integers().zeta(6)
```

```
1/2*alpha + 1/2
```

```sage
O = F.order([3*alpha])
O.zeta(3)
```

```
Traceback (most recent call last):
...
ArithmeticError: There are no 3rd roots of unity in self.
```

class \texttt{sage.rings.number_field.order.RelativeOrder}(\( K, \) \texttt{absolute_order},
\texttt{is_maximal=None}, \texttt{check=True})

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:

```python
sage: R = EquationOrder([x^2 + 1, x^3 + 2], '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: R = EquationOrder([x^2 + 1, x^2 - 5], 'i,g'); R
Relative Order 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 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]
```

We compute a relative order in alpha0, alpha1, then make 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 in Number Field in alpha0 with defining polynomial x^2 + 2 over its base field
sage: R.absolute_order('gamma')
Order 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]
```

basis()

Return a basis for this order as \(Z\)-module.

EXAMPLES:

```python
sage: K.<a,b> = NumberField([x^2+1, x^2+3])
sage: O = K.order([a,b])
sage: O.basis()
```
\[1, -2a + b, -b*a - 2, -5*a + 3*b\]

```
sage: z = O.1; z
-2*a + b
sage: z.absolute_minpoly()
x^4 + 14*x^2 + 1
```

**index_in** *(other)*

Return the index of self in other. This is a lattice index, so it is a rational number if self isn’t contained in other.

**INPUT:**

- other – another order with the same ambient absolute number field.

**OUTPUT:**

a rational number

**EXAMPLES:**

```
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
```

**is_suborder** *(other)*

Returns true if self is a subset of the order other.

**EXAMPLES:**

```
sage: K.<a,b> = NumberField([x^2 + 1, x^3 + 2])
sage: R1 = K.order([a,b])
sage: R2 = K.order([2*a,b])
sage: R3 = K.order([a + b, b + 2*a])
sage: R1.is_suborder(R2)
False
sage: R2.is_suborder(R1)
True
sage: R3.is_suborder(R1)
True
sage: R1.is_suborder(R3)
True
sage: R1 == R3
True
```

```
sage.rings.number_field.order.absolute_order_from_module_generators(gens, check_integral=True, check_rank=True, check_is_ring=True, is_maximal=None, al-
low_subfield=False)
```

**INPUT:**

- gens – list of elements of an absolute number field that generates an order in that number field as a \( \mathbb{Z} \) module.
• check_integral – check that each gen is integral
• check_rank – check that the gens span a module of the correct rank
• check_is_ring – check that the module is closed under multiplication (this is very expensive)
• is_maximal – bool (or None); set if maximality of the generated order is known

OUTPUT:
an absolute order

EXAMPLES:
We have to explicitly import the function, since it isn’t meant for regular usage:

```
sage: from sage.rings.number_field.order import absolute_order_from_module_generators

sage: K.<a> = NumberField(x^4 - 5)
sage: O = K.maximal_order(); O
Maximal Order 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]
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)
Order 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)
Order in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_module_generators(g, check_rank=False)
Order in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_module_generators(g, check_integral=False)
Order in Number Field in a with defining polynomial x^4 - 5
```

Next we illustrate constructing “fake” orders to illustrate turning off various check flags:

```
sage: k.<i> = NumberField(x^2 + 1)
sage: R = absolute_order_from_module_generators([2, 2*i], check_is_ring=False); R
Order 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 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 don’t check that, we’ll find that the rank is wrong or that the order isn’t closed under multiplication:

3.1. Orders in Number Fields 219
We turn off all check flags and make a really messed up order:

```python
sage: R = absolute_order_from_module_generators([1/2, i], check_is_ring=False, check_integral=False, check_rank=False); R
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1/2, i]
```

An order that lives in a subfield:

```python
sage: F.<alpha> = NumberField(x**4 + 3)
sage: F.order([alpha**2], allow_subfield=True)
Order in Number Field in alpha with defining polynomial x^4 + 3
```

```python
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:**

```python
sage: K.<a> = NumberField(x^4 - 5)
sage: K.order(a)
Order 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:

```python
sage: from sage.rings.number_field.order import absolute_order_from_ring_generators
sage: absolute_order_from_ring_generators([a])
Order in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_ring_generators([3*a, 2, 6*a+1])
Order 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

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

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 in Number Field in a with defining polynomial x^4 - 5
sage: absolute_order_from_ring_generators([a^2], check_rank=False)
Order in Number Field in a with defining polynomial x^4 - 5

sage.rings.number_field.order.each_is_integral(v)
Return True if each element of the list v of elements of a number field is integral.

EXAMPLES:

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 Z of integers.

EXAMPLES:

sage: from sage.rings.number_field.order import is_NumberFieldOrder
sage: is_NumberFieldOrder(NumberField(x^2+1,'a').maximal_order())
True
sage: is_NumberFieldOrder(ZZ)
True
sage: is_NumberFieldOrder(QQ)
False
sage: is_NumberFieldOrder(45)
False

sage.rings.number_field.order.relative_order_from_ring_generators(gens,  
    check_is_integral=check_is_integral,  
    check_rank=check_rank,  
    is_maximal=is_maximal,  
    allow_subfield=allow_subfield)

INPUT:

- gens – list of integral elements of an absolute order.
- check_is_integral – bool (default: True), whether to check that each generator is integral.

3.1. Orders in Number Fields
• 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

EXAMPLES:
We have to explicitly import this function, since it isn’t meant for regular usage:

```
sage: from sage.rings.number_field.order import relative_order_from_ring_generators
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 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:
sage: S.basis()
[1, 7*i - 2*a, -a*i + 8, 25*i - 7*a]
```

3.2 Number Field Ideals

AUTHORS:
• Steven Sivek (2005-05-16)
• 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

We test that pickling works:

```
sage: K.<a> = NumberField(x^2 - 5)
sage: I = K.ideal(2/(5+a))
sage: I == loads(dumps(I))
True
```

```
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.

class sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal(field, gens, coerce=True)
    A fractional ideal in a number field.
```

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^3 - 2)
sage: I = K.ideal(2/(5+a))
sage: J = I^2
```
\begin{verbatim}
sage: Jinv = I^(-2)
sage: J*Jinv
Fractional ideal (1)
\end{verbatim}

**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:**

\begin{verbatim}
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
\end{verbatim}

**divides(other)**

Returns True if this ideal divides other and False otherwise.

**EXAMPLES:**

\begin{verbatim}
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
\end{verbatim}

**element_1_mod(other)**

Returns an element \( r \) in this ideal such that \( 1 - r \) is in other

An error is raised if either ideal is not integral of if they are not coprime.

**INPUT:**

- other – another ideal of the same field, or generators of an ideal.

**OUTPUT:**

An element \( r \) of the ideal self such that \( 1 - r \) is in the ideal other

**AUTHOR:** Maite Aranes (modified to use PARI’s pari:idealaddtoone by Francis Clarke)

**EXAMPLES:**

\begin{verbatim}
sage: K.<a> = NumberField(x^3-2)
sage: A = K.ideal(a+1); A; A.norm()
\end{verbatim}

(continues on next page)
Fractional ideal \((a + 1)\)
3
\[\text{sage: } B = K.ideal(a^2-4*a+2); B; B.norm()\]
Fractional ideal \((a^2 - 4*a + 2)\)
68
\[\text{sage: } r = A.element_1_mod(B); r\]
-33
\[\text{sage: } r \text{ in } A\]
True
\[\text{sage: } 1-r \text{ in } B\]
True

**euler_phi()**

Returns 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:**

\[\text{sage: } K.<i>=NumberField(x^2+1)\]
\[\text{sage: } I = K.ideal(2+i)\]
\[\text{sage: } [r \text{ for } r \text{ in } I.residues() \text{ if } I.is_coprime(r)]\]
\[-2*i, -i, i, 2*i\]
\[\text{sage: } I.euler_phi()\]
4
\[\text{sage: } J = I^3\]
\[\text{sage: } J.euler_phi()\]
100
\[\text{sage: } len([r \text{ for } r \text{ in } J.residues() \text{ if } J.is_coprime(r)])\]
100
\[\text{sage: } J = K.ideal(3-2*i)\]
\[\text{sage: } I.is_coprime(J)\]
True
\[\text{sage: } I.euler_phi() * J.euler_phi() == (I*J).euler_phi()\]
True
\[\text{sage: } L.<b> = K.extension(x^2 - 7)\]
\[\text{sage: } L.ideal(3).euler_phi()\]
64

**factor()**

Factorization of this ideal in terms of prime ideals.

**EXAMPLES:**

\[\text{sage: } K.<a> = NumberField(x^4 + 23); K\]
Number Field in a with defining polynomial \(x^4 + 23\)
\[\text{sage: } I = K.ideal(19); I\]
Fractional ideal \((19)\)
\[\text{sage: } F = I.factor(); F\]
(Fractional ideal \((19, 1/2*a^2 + a - 17/2)) \times \text{(Fractional ideal } (19, 1/2*a^2 - a - 17/2))\)
\[\text{sage: } type(F)\]
<class 'sage.structure.factorization.Factorization'>
\[\text{sage: } list(F)\]
[(Fractional ideal \((19, 1/2*a^2 + a - 17/2)), 1), \text{(Fractional ideal } (19, 1/2*a^2 - a - 17/2)), 1)]
idealcoprime \( J \)

Returns \( l \) such that \( l \cdot \text{self} \) is coprime to \( J \).

**INPUT:**

- \( J \) - another integral ideal of the same field as \text{self}, which must also be integral.

**OUTPUT:**

- \( l \) - an element such that \( l \cdot \text{self} \) is coprime to the ideal \( J \)

**TODO:** Extend the implementation to non-integral ideals.

**EXAMPLES:**

```python
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); lam  # -1/6*a + 1/6
sage: (lam*A).is_coprime(B)  # True
```

**ALGORITHM:** Uses Pari function \texttt{pari:idealcoprime}.

ideallog \( (x, \ gens=None, \ check=True) \)

Returns 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 \( \gens \) 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}.
- \( \gens \) - 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 \texttt{idealstar}().
- \( \check \) - if True, do a consistency check on the results. Ignored if \( \gens \) is None.

**OUTPUT:**

- \( l \) - a list of non-negative integers \( (x_i) \) such that \( x = \prod_i g_i^{x_i} \) in \((R/I)^*\), where \( x_i \) 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 \texttt{idealstar}(), this just means that \( 0 \leq x_i < d_i \).

A \texttt{ValueError} will be raised if the elements specified in \( \gens \) do not in fact generate the unit group (even if the element \( x \) is in the subgroup they generate).

**EXAMPLES:**

```python
sage: k.<a> = NumberField(x^3 - 11)
sage: A = k.ideal(a^2 +3)
sage: r = G(l).value()  # (continues on next page)
```
Examples with custom generators:

```python
sage: K.<a> = NumberField(x^2 - 7)
sage: I = K.ideal(17)
sage: I.ideallog(a + 7, [1+a, 2])
[10, 3]
sage: I.ideallog(a + 7, [2, 1+a])
[0, 118]
sage: L.<b> = NumberField(x^4 - x^3 - 7*x^2 + 3*x + 2)
sage: J = L.ideal(-b^3 - b^2 - 2)
sage: u = -14*b^3 + 21*b^2 + b - 1
sage: v = 4*b^2 + 2*b - 1
sage: J.ideallog(5+2*b, [u, v], check=True)
[4, 13]
```

A non-example:

```python
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`  
Returns the finite abelian group \((O_K/I)^*\), where I is the ideal self of the number field \(K\), and \(O_K\) is the ring of integers of \(K\).

INPUT:

- `flag` (int default 1) – when `flag`=2, it also computes the generators of the group \((O_K/I)^*\), 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 \((O_K/I)^*\) (with generators) and not the special `bid` structure.

OUTPUT:

The finite abelian group \((O_K/I)^*\).

**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:**

```python
sage: k.<a> = NumberField(x^3 - 11)
sage: A = k.ideal(5)
sage: G = A.idealstar(); G
(continues on next page)```

(continued from previous page)
Multiplicative Abelian group isomorphic to $C_{24} \times C_{4}$

```sage
sage: G.gens()
(f0, f1)
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
```

**ALGORITHM:** Uses Pari function `pari:idealstar` `invertible_residues` 

`invertible_residues (reduce=True)`

Returns a 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
sage: K.<i> = NumberField(x^2+1)
sage: ires = K.ideal(2).invertible_residues(); ires
xmrange_iter([[0, 1]], <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
sage: K.<a> = NumberField(x^3-10)
sage: I = K.ideal(a-1)
sage: len(list(I.invertible_residues())) == I.euler_phi()
True
sage: K.<z> = CyclotomicField(10)
sage: len(list(K.primes_above(3)[0].invertible_residues()))
80
```

**AUTHOR:** John Cremona
invertible_residues_mod(subgp_gens=[], reduce=True)

Returns 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 \( \mathcal{O} \) representing the elements of \((\mathcal{O}/I)^*/U\), where \( I \) is this ideal and \( U \) is the subgroup of \((\mathcal{O}/I)^*/\) generated by subgp_gens.

EXAMPLES:

```
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
```
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
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
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
```

**is_coprime**(other)
Returns True if this ideal is coprime to the other, else False.

INPUT:
- other – another ideal of the same field, or generators of an ideal.

OUTPUT:
True if self and other are coprime, else False.

**Note:** This function works for fractional ideals as well as integral ideals.

AUTHOR: John Cremona

EXAMPLES:
```python
sage: K.<i>=NumberField(x^2+1)
sage: I = K.ideal(2+i)
sage: J = K.ideal(2-i)
sage: I.is_coprime(J)
True
sage: (I^-1).is_coprime(J^3)
True
sage: I.is_coprime(5)
False
sage: I.is_coprime(6+i)
True

See trac ticket #4536:
```
```python
sage: E.<a> = NumberField(x^5 + 7*x^4 + 18*x^2 + x - 3)
sage: OE = E.ring_of_integers()
sage: i,j,k = [u[0] for u in factor(3*OE)]
sage: (i/j).is_coprime(j/k)
False
sage: (j/k).is_coprime(j/k)
False
sage: F.<a, b> = NumberField([x^2 - 2, x^2 - 3])
sage: F.ideal(3 - a*b).is_coprime(F.ideal(3))
False
```

**is_maximal()**

Return True if this ideal is maximal. This is equivalent to self being prime, since it is nonzero.

**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_maximal()
False
sage: K.ideal(7).is_maximal()
True
```

**is_trivial**(proof=None)

Returns True if this is a trivial ideal.

**EXAMPLES:**
```python
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
```

**numerator()**

Return the numerator ideal of this fractional ideal.

Each fractional ideal has a unique expression as $N/D$ where $N$, $D$ are coprime integral ideals. The numerator is $N$.  

230  Chapter 3. Orders, Ideals, Ideal Classes
EXAMPLES:

```python
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
```

`prime_factors()`

Return a list of the prime ideal factors of self.

**OUTPUT:** list – list of prime ideals (a new list is returned each time this function is called)

**EXAMPLES:**

```python
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)]
```

`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: 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)
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)]
```

(continues on next page)
### prime_to_idealM_part \((M)\)

Version for integral ideals of the \texttt{prime_to_m_part} function over \(\mathbb{Z}\). Returns the largest divisor of 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 self that is coprime to \(M\).

**AUTHOR:** Maite Aranes

**EXAMPLES:**

```python
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
```

### 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.

**EXAMPLES:**

```python
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
```

### ray_class_number()

Return the order of the ray class group modulo this ideal. This is a wrapper around Pari’s \texttt{bnrclassno} function.

**EXAMPLES:**

```python
```
reduce \( f \)
Return the canonical reduction of the element of \( f \) 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 \( f + I \) (\( I \) =self) as described in the residues function, namely an integral element 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 self.

Note: The reduced element \( g \) is not necessarily small. To get a small \( g \) use the method small_residue.

EXAMPLES:
```
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
```

The reduced element is in the list of canonical representatives returned by the residues method:
```
sage: I.reduce(c) in list(I.residues())
True
```

The reduced element does not necessarily have smaller norm (use small_residue for that)
```
sage: c.norm()
25
sage: (I.reduce(c)).norm()
209
sage: (I.small_residue(c)).norm()
10
```

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: 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: K.<a> = NumberField(x^3-7)
sage: P = K.ideal(29).factor()[0][0]
sage: P.residue_field()  # 390
Residue field in abar of Fractional ideal (2*a^2 + 3*a - 10)
sage: P.residue_field('z')
Residue field in z of Fractional ideal (2*a^2 + 3*a - 10)
```

Another example:

```
sage: K.<a> = NumberField(x^3-7)
sage: P = K.ideal(389).factor()[0][0]; P  # 390
Fractional ideal (389, a^2 - 44*a - 9)
sage: P.residue_class_degree()
2
sage: P.residue_field()  # 390
Residue field in abar of Fractional ideal (389, a^2 - 44*a - 9)
sage: P.residue_field('z')
Residue field in z of Fractional ideal (389, a^2 - 44*a - 9)
sage: FF.<w> = P.residue_field()
sage: FF  # 390
Residue field in w of Fractional ideal (389, a^2 - 44*a - 9)
sage: FF((a+1)^390)
36
sage: FF(a)  # 390
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 trac ticket #1951 has been fixed:
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

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)
2*abar

We verify that trac ticket #8721 is fixed:

sage: L.<a, b> = NumberField([x^2 - 3, x^2 - 5])
sage: L.ideal(a).residue_field()
Residue field in abar of Fractional ideal (a)

residues()
Return a 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:
Given an element $f$ of the ambient number field, returns 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:**

```python
class sage.rings.number_field.number_field_ideal.NumberFieldIdeal(field, gens, coerce=True):

    Bases: sage.rings.ideal.Ideal_generic

    An ideal of a number field.

    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]
sage: I.S_ideal_class_log([])
[3]
```

**absolute_norm()**
A synonym for norm.

EXAMPLES:

```python
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + 2*i).absolute_norm()
5
```

**absolute_ramification_index()**
A synonym for ramification_index.

EXAMPLES:

```python
sage: K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + i).absolute_ramification_index()
2
```

**artin_symbol()**
Return the Artin symbol \( (K/\mathbb{Q}, P) \), where \( K \) is the number field of \( P = \text{self} \). This is the unique element \( s \) of the decomposition group of \( P \) such that \( s(x) = x^p \mod P \) where \( p \) is the residue characteristic of \( P \). (Here \( P(\text{self}) \) should be prime and unramified.)

See the `artin_symbol` method of the `GaloisGroup_v2` class for further documentation and examples.

EXAMPLES:

```python
sage: QuadraticField(-23, 'w').primes_above(7)[0].artin_symbol()
(1,2)
```

**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:

```python
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 (trac ticket #252):

sage: K.<a> = NumberField(2*x^2 - 1/3)
sage: K.ideal(a).basis()
[1, a]

coordinates (x)

Returns 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:**

sage: K.<i> = QuadraticField(-1)
sage: I = K.ideal(7+3*i)
sage: Ibasis = I.integral_basis(); Ibasis
[58, i + 41]
sage: a = 23-14*i
sage: acoords = I.coordinates(a); acoords
(597/58, -14)
sage: sum([Ibasis[j]*acoords[j] for j in range(2)]) == a
True
sage: b = 123+456*i
sage: bcoords = I.coordinates(b); bcoords
(-18573/58, 456)
sage: sum([Ibasis[j]*bcoords[j] for j in range(2)]) == b
True
sage: J = K.ideal(0)
sage: J.coordinates(0)
()  
sage: J.coordinates(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 decomposition_group method of the GaloisGroup_v2 class for further examples and doctests.

**EXAMPLES:**
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 (2*Z[x].gen()^4 + 2*Z[x].gen()^2 - 2*Z[x].gen() + 1)
sage: A = I.free_module()
sage: A
Free module of degree 6 and rank 6 over Integer Ring
User basis matrix:
[1 0 0 0 0 0]
[0 1 0 0 0 0]
[0 0 1 0 0 0]
[10 4 5 1 0 0]
[5 1 1 0 1 0]
[5 6 2 1 1 1]
```

However, the actual \( \mathbb{Z} \)-module is not at all random:

```
sage: A.basis_matrix().change_ring(ZZ).echelon_form()
[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]
```

The ideal doesn’t have to be integral:

```
sage: J = I^(-1)
sage: B = J.free_module()
sage: B.echelonized_basis_matrix()
[ 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]
```

This also works for relative extensions:

```
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()
```

(continues on next page)
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
\[
\begin{bmatrix}
\frac{1}{4} & 0 & 0 & 0 \\
-\frac{3}{16} & \frac{7}{16} & -\frac{1}{16} & \frac{1}{16} \\
\frac{3}{16} & \frac{7}{16} & \frac{1}{16} & \frac{1}{16} \\
0 & -\frac{5}{8} & 0 & -\frac{1}{8}
\end{bmatrix}
\]

An example of intersecting ideals by intersecting free modules:

\begin{verbatim}
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:
\[
\begin{bmatrix}
1 & 1/2 & 1/2 \\
0 & 1 & 1 \\
0 & 0 & 2
\end{bmatrix}
\]
sage: N.index_in(p1.free_module()).abs()
2
\end{verbatim}

gens_reduced (\textit{proof}=None)
Express this ideal in terms of at most two generators, and one if possible.
This function indirectly uses \texttt{bnfisprincipal}, so set \textit{proof}=	exttt{True} if you want to prove correctness (which is the default).

\textbf{EXAMPLES:}

\begin{verbatim}
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 5)
sage: K.ideal([a+2, 9]).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)
\end{verbatim}

gens_two ()
Express this ideal using exactly two generators, the first of which is a generator for the intersection of the ideal with \(Q\).

\textbf{ALGORITHM:} uses \texttt{PARI}'s \texttt{pari:idealtwoelt} function, which runs in randomized polynomial time and is very fast in practice.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 5)
sage: J = K.ideal([a+2, 9])
sage: J.gens()
(a + 2, 9)
sage: J.gens_two()
(9, a + 2)
\end{verbatim}
The second generator is zero if and only if the ideal is generated by a rational, in contrast to the PARI function pari:idealtwoelt:

```python
sage: I = K.ideal(12)
sage: pari(K).idealtwoelt(I)  # Note that second element is not zero
[12, [0, 12]]
sage: I.gens_two()
(12, 0)
```

### 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:

```python
sage: K.<a> = QuadraticField(-163)
sage: J = K.primes_above(random_prime(10^6))[0]
sage: J.ideal_class_log()
[]
```

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: 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]
```

### 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 inertia_group method of the GaloisGroup_v2 class for further examples and doctests.

**EXAMPLES:**

```python
sage: QuadraticField(-23, 'w').primes_above(23)[0].inertia_group()
Galois group of Number Field in w with defining polynomial x^2 + 23
```

3.2. Number Field Ideals
integral_basis()
Return a list of generators for this ideal as a \( \mathbb{Z} \)-module.

EXAMPLES:
```
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:
```
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
```

intersection(other)
Return the intersection of self and other.

EXAMPLES:
```
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

An example with non-principal ideals:
```
sage: L.<a> = NumberField(x^3 - 7)
sage: p = L.ideal(a^2 + a + 1, 2)
sage: q = L.ideal(a+1)
sage: p.intersection(q) == L.ideal(8, 2*a + 2)
True
```

A relative example:
```
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
```
**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()  # Example 1
True

sage: (K.ideal(1) / (3*a+1)).is_integral()  # Example 2
False
```

**is_maximal()**
Return True if this ideal is maximal. This is equivalent to self being prime and nonzero.

**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_maximal()  # Example 1
False

sage: K.ideal(7).is_maximal()  # Example 2
True
```

**is_prime()**
Return True if this ideal is prime.

**EXAMPLES:**
```python
sage: K.<a> = NumberField(x^2 - 17); K
Number Field in a with defining polynomial x^2 - 17
sage: K.ideal(5).is_prime()  # inert prime
True

sage: K.ideal(13).is_prime()  # split
False

sage: K.ideal(17).is_prime()  # ramified
False
```

**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()  # proof=None
False

sage: I = P^5
sage: I.is_principal()  # proof=True
True

sage: I # random
Fractional ideal (-1/2*a + 3/2)
sage: P = K.ideal([2]).factor()[1][0]
sage: I = P^5
sage: I.is_principal()  # proof=False
True
```
is_zero()  
Return True iff self is the zero ideal  
Note that \((0)\) is a \texttt{NumberFieldIdeal}, not a \texttt{NumberFieldFractionalIdeal}.

**EXAMPLES:**

```python
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
```

norm()  
Return the norm of this fractional ideal as a rational number.

**EXAMPLES:**

```python
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
```

number_field()  
Return the number field that this is a fractional ideal in.

**EXAMPLES:**

```python
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: 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()  
Returns a PARI prime ideal corresponding to the ideal \texttt{self}.

**INPUT:**

- \texttt{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
```

```
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 ramification_group method of the GaloisGroup class for further examples and doctests.

EXAMPLES:

```python
sage: QuadraticField(-23, 'w').primes_above(23)[0].ramification_group(0)
Galois group of Number Field in w with defining polynomial x^2 + 23
sage: QuadraticField(-23, 'w').primes_above(23)[0].ramification_group(1)
Subgroup [()] of Galois group of Number Field in w with defining polynomial x^2 + 23
```

```
random_element(*args, **kwds)
```

Return a random element of this order.

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.

EXAMPLES:

```python
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
```

A relative example:

```python
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: I = K.ideal(1-a)
sage: I.random_element() # random output
```

(continues on next page)
reduce_equiv()

Return a small ideal that is equivalent to self in the group of fractional ideals modulo principal ideals. Very often (but not always) if self is principal then this function returns the unit ideal.

ALGORITHM: Calls pari:idealred function.

EXAMPLES:

```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)
```

relative_norm()

A synonym for norm.

EXAMPLES:

```sage
K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + 2*i).relative_norm()
5
```

relative_ramification_index()

A synonym for ramification_index.

EXAMPLES:

```sage
K.<i> = NumberField(x^2 + 1)
sage: K.ideal(1 + i).relative_ramification_index()
2
```

residue_symbol(e, m, check=True)

The m-th power residue symbol for an element e and the proper ideal.

\[
\left( \frac{\alpha}{P} \right) \equiv \alpha^{\frac{N(P)-1}{m}} \mod P
\]

Note: accepts m=1, in which case returns 1

Note: can also be called for an element from sage.rings.number_field_element.residue_symbol

Note: e is coerced into the number field of self
Note: if $m=2$, $e$ is an integer, and self.number_field() has absolute degree 1 (i.e. it is a copy of the rationals), then this calls kronecker_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: K.<a> = NumberField(x - 1)
sage: K.ideal(11).residue_symbol(7, 2)
-1
```

Cubic Residue:

```
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:

```
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:

```
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:
```python
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
```

**Class** `sage.rings.number_field.number_field_ideal.QuotientMap(K, M_OK_change, Q, I)`

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:**

```python
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:
[0 1 0 0]
[3 0 1 0]
```

**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: k.<a> = NumberField(x^2 + 2)
sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
sage: is_NumberFieldFractionalIdeal(I)
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
```

**sage.rings.number_field.number_field_ideal.is_NumberFieldIdeal(x)**

Return True if $x$ is an ideal of a number field.
EXAMPLES:

```python
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: k.<a> = NumberField(x^2 + 2)
sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
sage: is_NumberFieldIdeal(I)
True
sage: Z = k.ideal(0); Z
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
sage: is_NumberFieldIdeal(Z)
True
```

`sage.rings.number_field.number_field_ideal.quotient_char_p(I, p)`

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: 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 (-3*i - 2)
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]
```

### 3.3 Relative Number Field Ideals

**AUTHORS:**

- Steven Sivek (2005-05-16)
EXAMPLES:

```
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 (2*b*a + 1)
sage: K.fractional_ideal(G).absolute_norm().factor()
7^2
```

```python
class sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel(field, gens, coerce=True):
    Bases: sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal

    An ideal of a relative number field.

    EXAMPLES:

    ```
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: (g / (a0 + 1)).is_integral()
    True
    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:

    ```
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()
```

An example of an inert ideal:
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
```

Now pass ‘m’ as the name for the generator of the absolute field:

```python
sage: 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)
sage: J.ideal_below()
Fractional ideal (b)
sage: J.ideal_below().norm()
2
```

### absolute_norm()

Compute the absolute norm of this fractional ideal in a relative number field, returning a positive integer.

**EXAMPLES:**

```python
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:

```
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.absolute_ramification_index()
4
sage: I.smallest_integer()
3
sage: K.ideal(3) == I^4
True
```

**element_1_mod** *(other)*

Returns an element \( r \) in this ideal such that \( 1 - r \) is in other.

An error is raised if either ideal is not integral of if they are not coprime.

**INPUT:**

- *other* – another ideal of the same field, or generators of an ideal.

**OUTPUT:**

an element \( r \) of the ideal self such that \( 1 - r \) is in the ideal other.

**EXAMPLES:**

```
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:**

```
sage: K.<a, b> = QQ.extension([x^2 + 11, x^2 - 5])
sage: K.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)]
```

(continues on next page)
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
sage: [p.is_prime() for p in [P, Q]]
[True, True]

free_module()
Return this ideal as a \( \mathbb{Z} \)-submodule of the \( \mathbb{Q} \)-vector space corresponding to the ambient number field.

EXAMPLES:

```sage
sage: K.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
sage: I = K.ideal(a*b - 1)
sage: I.free_module()
Free module of degree 6 and rank 6 over Integer Ring
User basis matrix:
...
sage: I.free_module().is_submodule(K.maximal_order().free_module())
True
```

gens_reduced()
Return a small set of generators for this ideal. This will always return a single generator if one exists (i.e. if the ideal is principal), and otherwise two generators.

EXAMPLES:

```sage
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 2])
sage: I = K.ideal((a + 1)*b/2 + 1)
sage: I.gens_reduced()
(1/2*b*a + 1/2*b + 1,)
```

ideal_below()
Compute the ideal of \( K \) below this ideal of \( L \).

EXAMPLES:

```sage
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:

### 3.3. Relative Number Field Ideals
This example concerns an ideal that splits in the quadratic field but each factor ideal remains inert in the extension:

```
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)
```

The choice of embedding of quadratic field into quartic field matters:

```
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^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^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: J = I.ideal_below(); J
Fractional ideal (-b)
```
Number fields defined by non-monic and non-integral polynomials are supported (trac ticket #252):

```
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)
```

**integral_basis()**

Return a basis for self as a Z-module.

**EXAMPLES:**

```
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
[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:

```
sage: V, _, phi = K.absolute_vector_space()
sage: 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:**

```
sage: K.<a, b> = NumberFieldTower([x^2 - 23, x^2 + 1])
sage: I = K.ideal([a + b/3])
sage: J, d = I.integral_split()
sage: J.is_integral()
True
sage: J == d*I
True
```

**is_integral()**

Return True if this ideal is integral.

**EXAMPLES:**

```
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:**
```python
sage: K.<a, b> = NumberField([x^2 - 17, x^3 - 2])
sage: K理想的(a + b).is_prime()
True
sage: K理想的(13).is_prime()
False
```

### is_principal(proof=None)

Return True if this ideal is principal. If so, set self.__reduced_genersators, with length one.

**EXAMPLES:**

```python
sage: K.<a, b> = NumberField([x^2 - 23, x^2 + 1])
sage: I = K理想([7, (-1/2*b - 3/2)*a + 3/2*b + 9/2])
sage: I.is_principal()
True
sage: I # random
Fractional ideal ((1/2*b + 1/2)*a - 3/2*b - 3/2)
```

### is_zero()

Return True if this is the zero ideal.

**EXAMPLES:**

```python
sage: K.<a, b> = NumberField([x^2 + 3, x^3 + 4])
sage: K理想(17).is_zero()
False
sage: K理想(0).is_zero()
True
```

### 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:**

```python
sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 2])
sage: K理想的(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: K.<a, b> = NumberField([x^2 + 23, x^2 - 7])
sage: I = K理想的(2, (a + 2*b + 3)/2)
sage: 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:**
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

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
sage: N.ring()
Number Field in a with defining polynomial x^2 + 6
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 (trac ticket #252):

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:

sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberFieldTower([X^2 - 2, X^2 - 3])
sage: PF.<Y> = F[]

(continues on next page)
```python
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
```

**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: K.<a, w> = NumberFieldTower([x^2 - 3, x^2 + x + 1])
sage: I = K.ideal(6, -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]
```

**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: 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]
```

**valuation**(p)

Return the valuation of this fractional ideal at p.

**INPUT:**

- p – a prime ideal p of this relative number field.

**OUTPUT:**
The valuation of this fractional ideal at the prime \( p \). If \( p \) is not prime, raise a ValueError.

EXAMPLES:

```sage
K.<a, b> = NumberField([x^2 - 17, x^3 - 2])
A = K.ideal(a + b)
A.is_prime()
True
(A*K.ideal(3)).valuation(A)
1
K.ideal(25).valuation(5)

sage: 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:

```sage
from sage.rings.number_field.number_field_ideal_rel import is_NumberFieldFractionalIdeal_rel
from sage.rings.number_field.number_field_ideal import is_NumberFieldFractionalIdeal
from sage.rings.number_field.number_field_ideal import is_NumberFieldFractionalIdeal

k.<a> = NumberField(x^2 + 2)
I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
is_NumberFieldFractionalIdeal_rel(I)
False
R.<x> = QQ[
K.<a> = NumberField(x^2+6)
L.<b> = K.extension(K['x'].gen()^4 + a)
I = L.ideal(b); I
Fractional ideal (6, b)
is_NumberFieldFractionalIdeal_rel(I)
True
N = I.relative_norm(); N
Fractional ideal (-a)
is_NumberFieldFractionalIdeal_rel(N)
False
```

3.4 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:
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: O = K.OK(); O
Maximal Order in Number Field in a with defining polynomial x^2 + 23
sage: O*(2, 1/2*a + 1/2)
Fractional ideal (2, 1/2*a - 1/2)
sage: (O*(2, 1/2*a + 1/2)).is_principal()
False
sage: (O*(2, 1/2*a + 1/2))^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: sage.groups.abelian_gps.values.AbelianGroupWithValues_class

The class group of a number field.

EXAMPLES:

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

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

(continues on next page)
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: 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))

type

number_field()

Return the number field that this (S-)class group is attached to.

EXAMPLES:

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

class sage.rings.number_field.class_group.FractionalIdealClass(parent, element, ideal=None)

Bases: sage.groups.abelian_gps.values.AbelianGroupWithValuesElement

A fractional ideal class in a number field.

EXAMPLES:

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
sage: I = G.0; I
Fractional ideal class (2, 1/2*a - 1/2)
sage: I.ideal()
Fractional ideal \( (2, 1/2*a - 1/2) \)

**EXAMPLES:**

```python
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*OK).factor()]
sage: c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.gens()
(2, 1/2*w - 1/2)
```

**gens**

Return generators for a representative ideal in this (S-)ideal class.

**EXAMPLES:**

```python
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*OK).factor()]
sage: c = C(P2a); c
Fractional ideal class (2, 1/2*w - 1/2)
sage: c.gens()
(2, 1/2*w - 1/2)
```

**ideal**

Return a representative ideal in this ideal class.

**EXAMPLES:**

```python
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*OK).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:**

```python
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)
```

**is_principal**

Returns True iff this ideal class is the trivial (principal) 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*OK).factor()]
sage: c=C(P2a)
sage: c.is_principal()  
False
sage: (c^2).is_principal()  
False
sage: (c^3).is_principal()  
True

\texttt{reduce()} 
Return representative for this ideal class that has been reduced using PARI’s \texttt{idealred}.

\textbf{EXAMPLES:}

sage: k.<a> = NumberField(x^2 + 20072); 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
sage: I = (G.0)^11; I
Fractional ideal class (41, 1/2*a + 5)
sage: J = G(I.ideal()^5); J
Fractional ideal class (115856201, 1/2*a + 40407883)
sage: J.reduce()
Fractional ideal class (57, 1/2*a + 44)
sage: J == I^5
True

\texttt{representative_prime(\texttt{norm\_bound}=1000)}  
Return a prime ideal in this ideal class.

\textbf{INPUT:}

\texttt{norm\_bound} (positive integer) – upper bound on the norm of primes tested.

\textbf{EXAMPLES:}

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)]

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 + 1/2),
 Fractional ideal (17, 1/2*a + 27/2),
 Fractional ideal (2, 1/2*a - 1/2)]
class sage.rings.number_field.class_group.SClassGroup(gens_orders, names, number_field, gens, S, proof=True)

Bases: sage.rings.number_field.class_group.ClassGroup

The S-class group of a number field.

EXAMPLES:

```python
sage: K.<a> = QuadraticField(-14)
sage: S = K.primes_above(2)
sage: K.S_class_group(S).gens()  # random gens (platform dependent)
(Fractional S-ideal class (3, a + 2),)

sage: K.<a> = QuadraticField(-974)
sage: CS = K.S_class_group(K.primes_above(2)); CS
S-class group of order 18 with structure C6 x C3 of Number Field in a with defining polynomial x^2 + 974
sage: CS.gen(0)  # random
Fractional S-ideal class (3, a + 2)
sage: CS.gen(1)  # random
Fractional S-ideal class (31, a + 24)
```

Element

alias of SFractionalIdealClass

S()

Return the set (or rather tuple) of primes used to define this class group.

EXAMPLES:

```python
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
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
sage: CS.S()
(Fractional ideal (2, a),)
sage: CT.S()
()```

class sage.rings.number_field.class_group.SFractionalIdealClass(parent, element, ideal=None)

Bases: sage.rings.number_field.class_group.FractionalIdealClass

An S-fractional ideal class in a number field for a tuple of primes S.

EXAMPLES:

```python
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)
```
### 3.5 Unit and S-unit groups of Number Fields

**EXAMPLES:**

```python
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)
```

The first generator is a primitive root of unity in the field:

```python
sage: UK = UnitGroup(K); UK
Unit group with structure C4 x Z of Number Field in a with defining polynomial x^4 - 14
sage: UK.gens()
(u0, u1)
sage: UK.gens_values()
[-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)  # u0
```

(continues on next page)
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, u0^3, u0]
```

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: all([UK.log(u^k) == (0,k) for k in range(10)])
True
```

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)]
```
A relative number field example:

\begin{verbatim}
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, b^2*a, b^3, a + 1, -b*a, -b^2, b^3*a + b^3, a, b, -b^2*a - b^2, b^3*a, -1, b*a + b, -b^2*a, -b^3, -a - 1, b*a, b^2, -b^3*a - b^3, -a, -b, b^2*a + b^2, -b^3*a, 1]
\end{verbatim}

A relative extension example, which worked thanks to the code review by F.W.Clarke:

\begin{verbatim}
sage: PQ.<X> = QQ[]
sage: F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
\end{verbatim}
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

AUTHOR:

• John Cremona

class sage.rings.number_field.unit_group.UnitGroup(number_field, proof=True, S=None)
Bases: sage.groups.abelian_gps.values.AbelianGroupWithValues_class

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:

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); unit
-8732*z^11 + 15496*z^10 + 51840*z^9 + 68804*z^8 + 51840*z^7 + 15496*z^6 -
 \rightarrow 8732*z^5 + 34216*z^3 + 64312*z^2 + 64312*z + 34216
sage: SUK.log(unit)
(3, 1, 4, 1, 5, 9, 2)
sage: SUK.log(unit) == vec
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 -
 \rightarrow 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
fundamental_units()
Return generators for the free part of the unit group, as a list.

```
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.

```
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); unit  # random
-253576*z^11 + 7003*z^10 - 395532*z^9 - 35275*z^8 - 500326*z^7 - 35275*z^6 -
 395532*z^5 + 7003*z^4 - 253576*z^3 - 59925*z - 59925
sage: UK.log(unit)
(13, 6, 7, 8, 9, 10)
```

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 -
 395532*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
```

number_field()
Return the number field associated with this unit group.

**EXAMPLES:**
```
sage: U = UnitGroup(QuadraticField(-23, 'w')); U
Unit group with structure C2 of Number Field in w with defining polynomial x^2 + 23
```
sage: U.number_field()
Number Field in w with defining polynomial x^2 + 23

primes()

Return the (possibly empty) list of primes associated with this S-unit group.

EXAMPLES:

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 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: K.<z> = CyclotomicField(13)
sage: UnitGroup(K).rank()
5
sage: SUK = UnitGroup(K,S=2); SUK.rank()
6

roots_of_unity()

Return all the roots of unity in this unit group, primitive or not.

EXAMPLES:

sage: x = polygen(QQ)
sage: K.<b> = NumberField(x^2+1)
sage: U = UnitGroup(K)
sage: zs = U.roots_of_unity(); zs
[b, -1, -b, 1]
sage: [ z**U.zeta_order() for z in zs ]
[1, 1, 1, 1]

torsion_generator()

Return a generator for 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.torsion_generator()
u0
sage: U.torsion_generator().value() # random
-1/4*a^3 - 1/4*a + 1/2

zeta (n=2, all=False)

Return one, or a list of all, primitive n-th root of unity in this unit group.

EXAMPLES:
3.6 Solve S-unit equation $x + y = 1$

Inspired by work of Tzanakis–de Weger, Baker–Wustholz and Smart, we use the LLL methods in Sage to implement an algorithm that returns all S-unit solutions to the equation $x + y = 1$.

REFERENCES:

- [MR2016]
- [Sma1995]
- [Sma1998]

AUTHORS:
EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import solve_S_unit_equation, eq_up_to_order
sage: K.<xi> = NumberField(x^2+x+1)
sage: S = K.primes_above(3)
sage: expected = [
    ((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)
]
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

```python
sage.rings.number_field.S_unit_solver.K0_func(SUK, A, prec=106)
```

Return the constant $K_0$ from Smart’s TCDF paper, [Sma1995]

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:

```python
sage: from sage.rings.number_field.S_unit_solver import K0_func
sage: K.<xi> = NumberField(x^3-3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: A = K.roots_of_unity()
sage: K0_func(SUK, A)  # abs tol 1e-29
9.475576673109275443280257946929e17
```

REFERENCES:

- [Sma1995] p. 824

```python
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:

```python
sage: from sage.rings.number_field.S_unit_solver import K1_func
sage: K.<xi> = NumberField(x^3-3)
```

```python
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
```

```python
sage: phi_real = K.places()[0]
```

```python
sage: phi_complex = K.places()[1]
```

```python
sage: A = K.roots_of_unity()
```

```python
sage: K1_func(SUK, phi_real, A)
4.396386097852707394927181864635e16
```

```python
sage: K1_func(SUK, phi_complex, A)
2.034870098399844430207420286581e17
```

REFERENCES:

• [Sma1995] p. 825

---

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 `betas_and_ns`

INPUT:

• `betas_and_ns` – a list of pairs $[\beta, |\beta|_v]$ outputted from the function where $\beta$ is an element of `SUK.fundamental_units()`

OUTPUT:

The pair $[\beta_k, |\beta_k|_v]$, where $\beta_k$ is an element of $K$ and $|\beta_k|_v$ is an integer

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import beta_k
sage: K.<xi> = NumberField(x^3-3)
```

```python
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
```

```python
sage: v_fin = tuple(K.primes_above(3))[0]
```

```python
sage: betas = [ [beta, beta.valuation(v_fin)] for beta in SUK.fundamental_units() ]
```

```python
sage: beta_k(betas)
[xi, 1]
```

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:

• `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 $K$
• $\text{prec}$ – the precision of the real field (default: 106)

**OUTPUT:**
The constant $c_{11}$, a real number

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import c11_func
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, \text{prec}=106)`

Return the constant $c_{13}$ from Smart’s TCDF paper, [Sma1995]

**INPUT:**
• $\text{SUK}$ – a group of $S$-units
• $v$ – an infinite place of $K$ (element of $\text{SUK.number_field().places(\text{prec})}$)
• $\text{prec}$ – 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: 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: \sage: c13_func(SUK, phi_real) \# abs tol 1e-29
0.425785913479803474619732728672
\sage: c13_func(SUK, phi_complex) \# abs tol 1e-29
0.2128929567399017373098663643363
```

It is an error to input a finite place.

```python
sage: phi_finite = K.primes_above(3)[0]
\sage: c13_func(SUK, phi_finite)
Traceback (most recent call last):
  ...
TypeError: Place must be infinite
```
REFERENCES:

- [Sma1995] p. 825

sage.rings.number_field.S_unit_solver.c3_func(SUK, prec=106)

Return the constant $c_3$ from Smart's 1995 TCDF paper, [Sma1995]

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: K.<xi> = NumberField(x^3-3)sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: c3_func(SUK)  # abs tol 1e-29
doctest:...: DeprecationWarning: function s_unit_solver.c3_func is deprecated; use S_unit_solver.c3_func instead

0.42578591347980347461973286726
```

Note: The numerator should be as close to 1 as possible, especially as the rank of the $S$-units grows large

REFERENCES:

- [Sma1995] p. 823

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 $\text{SUK.number_field().places(prec)}$)
- $A$ – 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: 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: phi_fin = tuple(K.primes_above(3))[0]
sage: A = K.roots_of_unity()
sage: c4_func(SUK, phi_real, A)
```

(continues on next page)
REFERENCES:

- [Sma1995] p. 824

\[\text{sage.rings.number_field.S_unit_solver.c8_c9_func}(\text{SU}K, \nu, A, \text{prec}=106)\]

Return the constants \(c_8\) and \(c_9\) from Smart's TCDF paper, [Sma1995]

INPUT:

- \(\text{SU}K\) – a group of \(S\)-units
- \(\nu\) – a finite place of \(K\) (a fractional ideal)
- \(A\) – the set of the product of the coefficients of the \(S\)-unit equation with each root of unity of \(K\)
- \(\text{prec}\) – the precision of the real field

OUTPUT:

The constants \(c_8\) and \(c_9\), as real numbers

EXAMPLES:

\[\text{sage: from sage.rings.number_field.S_unit_solver import c8_c9_func}\\\text{sage: K.<xi> = NumberField(x^3-3)}\\\text{sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))}\\\text{sage: v_fin = K.primes_above(3)[0]}\\\text{sage: A = K.roots_of_unity()}\\\text{sage: c8_c9_func(SUK, v_fin, A)}\]

# abs tol 1e-29

\((4.524941291354698258804956696127\times10^{15}, 1.621521281297160786545580368612\times10^{16})\)

REFERENCES:

- [Sma1995] p. 825
- [Sma1998] p. 226, Theorem A.2 for the local constants

\[\text{sage.rings.number_field.S_unit_solver.clean_rfv_dict}(rfv\_dictionary)\]

Given a residue field vector dictionary, removes 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 the entries of 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:

```python
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]}
```

```python
sage: len(rfv_dict)
7
sage: clean_rfv_dict(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.

```python
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 $K$
- `prec` – the precision of the real field (default: 106)

**OUTPUT:**

The log vector as a list of real numbers

**EXAMPLES:**
sage: from sage.rings.number_field.S_unit_solver import column_Log
sage: K.<xi> = NumberField(x^3-3)
sage: S = tuple(K.primes_above(3))
sage: SUK = UnitGroup(K, S=S)
sage: phi_complex = K.places()[1]
sage: v_fin = S[0]
sage: U = [phi_complex, v_fin]
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(compatible_system, split_primes_list)

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 \([q_0, q_1, \ldots, q_k]\)

OUTPUT:

A pair of vectors \([v, w]\) satisfying:

1. \(v[0] == v_i[0]\) for all \(i\)
2. \(w[0] == w_i[0]\) for all \(i\)
3. \(v[j] == v_i[j]\) modulo \(q_i - 1\) for all \(i\) and all \(j > 0\)
4. \(w[j] == w_i[j]\) 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)

sage.rings.number_field.S_unit_solver.compatible_systems(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:
  - \(\text{``q}_j = \text{split_prime_list}[j]``\)
  - \(\text{``v}_j``\) and \(\text{``w}_j``\) are complementary exponent vectors modulo \(\text{``q}_j - 1``\)
  - 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 \(\text{``v``}\) and \(\text{``w``}\) is bounded in absolute value by \(\text{``H``}\)
  - for any \(\text{``q}_j``\), \(\text{``v``}\) and \(\text{``v}_j``\) agree modulo \(\text{``(q}_j - 1``\)
  - for any \(\text{``q}_j``\), \(\text{``w``}\) and \(\text{``w}_j``\) agree modulo \(\text{``(q}_j - 1``\)

EXAMPLES:

```python
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, ml, g)

Given an exponent vector a modulo m0, returns an iterator over the exponent vectors for the modulus ml, such that a lift to the lcm modulus exists.

INPUT:

- a – an exponent vector for the modulus m0
- m0 – a positive integer (specifying the modulus for a)
- ml – a positive integer (specifying the alternate modulus)
- g – the gcd of m0 and ml

OUTPUT:

A list of exponent vectors modulo ml which are compatible with a.

Note:

- Exponent vectors must agree exactly in the 0th position in order to be compatible.

EXAMPLES:

```python
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),
(continues on next page)]
```
The order of the moduli matters.

\begin{verbatim}
sage: len(list(compatible_vectors(a, 18, 12, gcd(18,12))))
8
sage: len(list(compatible_vectors(a, 12, 18, gcd(18,12))))
27
\end{verbatim}

`sage.rings.number_field.S_unit_solver.compatable_vectors_check(a0, a1, g, l)`

Given exponent vectors with respect to two moduli, determines if they are compatible.

**INPUT:**
- `a0` – an exponent vector modulo $m_0$
- `a1` – an exponent vector modulo $m_1$ (must have the same length as `a0`)
- `g` – the gcd of $m_0$ and $m_1$
- `l` – the length of `a0` and of `a1`

**OUTPUT:**
True if there is an integer exponent vector $a$ satisfying

\[
\begin{align*}
a[0] &\equiv a0[0] \equiv a1[0] \\
\text{a[1 :]} &\equiv \text{a0[1 :]} \mod m_0 \\
\text{a[1 :]} &\equiv \text{a1[1 :]} \mod m_1
\end{align*}
\]

and False otherwise.

**Note:**
- Exponent vectors must agree exactly in the first coordinate.
- If exponent vectors are different lengths, an error is raised.

**EXAMPLES:**

\begin{verbatim}
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
\end{verbatim}

`sage.rings.number_field.S_unit_solver.construct_comp_exp_vec(tfv_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): [(3, 4)], (2, 2): [(0, 4)], (3, 5): [(1, 0)], (6, 4): [(1, 1)], (3, 2): [(3, 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)]}
```

```python
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, 2): [(2, 0)], (1, 3): [(1, 2)], (1, 4): [(0, 1)], (2, 0): [(1, 1)], (2, 1): [(4, 0)], (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)]}
```

**sage.rings.number_field.S_unit_solver.construct_complement_dictionaries**

A function to 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,...,1].

Note:

- The data of comp_exp_vec will later be lifted to 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: f = x^2 + 5
sage: H = 10
sage: K.<xi> = NumberField(f)
\...
<table>
<thead>
<tr>
<th>str</th>
<th>str</th>
<th>str</th>
</tr>
</thead>
</table>
sage: split_primes_list = [3, 7]
sage: actual = construct_complement_dictionaries(split_primes_list, SUK)
sage: expected = {3: {(0, 1, 0): [(1, 0, 0), (0, 1, 0)], (1, 0, 0): [(0, 5, 4), (0, 3, 2), (0, 1, 0)], \..., (1, 0, 4): [(1, 2, 4), (1, 4, 0), (1, 0, 2)], \..., (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 \\\-
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>str</td>
<td>str</td>
<td>str</td>
</tr>
</tbody>
</table>
\for vec in \\|expected[p])\n```

sage: all(set(actual[p][vec]) == set(expected[p][vec]) \for p in [3,7] \for vec in \\~expected[p])
```
True
```

sage.rings.number_field.S_unit_solver.construct_rfv_to_ev(rfv_dictionary, q, d, verbose=False)

Returns a reverse lookup dictionary, to find the exponent vectors associated to a given residue field vector.

INPUTS:

- **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 \(q\)
• \(\text{verbose}\) – 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 \(\text{rfv}_\text{dictionary}[\text{e0}] = r0\), 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)], (5, 6): [(2, 1)], (1, 2): [(1, 2)]}
```

**sage.rings.number_field.S_unit_solver.cx_LLL_bound** (\(SUK, A, \text{prec}=106\))

Return the maximum of all of the \(K_j\)'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 \(K\)
• \(\text{prec}\) – 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: K.<xi> = NumberField(x^3-3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: A = K.roots_of_unity()
sage: cx_LLL_bound(SUK, A) # long time
22
```

**sage.rings.number_field.S_unit_solver.defining_polynomial_for_Kp** (\(\text{prime}, \text{prec}=106\))

**INPUT:**
• \(\text{prime}\) – a prime ideal of a number field \(K\)
• \(\text{prec}\) – a positive natural number (default: 106)

---

3.6. Solve \(S\)-unit equation \(x + y = 1\)
OUTPUT:
A polynomial with integer coefficients that is equivalent \( \text{mod } p^\text{prec} \) to a defining polynomial for the completion of \( K \) associated to the specified prime.

**Note:** \( K \) has to be an absolute extension

**EXAMPLES:**

```python
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
```

```python
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
```

```python
sage: p5 = K.prime_above(5); p5
Fractional ideal (5, a + 2)
sage: defining_polynomial_for_Kp(p5, 100)
x + 340832191958133385114942613351834100964285496304040728906961917542037
```

\[ \text{sage.rings.number_field.S_unit_solver.drop_vector} \( (ev, p, q, \text{complement\_ev\_dict}) \) \]

Determines if the exponent vector, \( ev \), may be removed from the complement dictionary during construction. This will occur if \( ev \) is not compatible with an exponent vector mod \( q-1 \).

**INPUT:**
- \( ev \) – an exponent vector modulo \( p - 1 \)
- \( p \) – the prime such that \( ev \) is an exponent vector modulo \( p-1 \)
- \( q \) – a prime, distinct from \( p \), that is a key in the \( \text{complement\_ev\_dict} \)
- \( \text{complement\_ev\_dict} \) – a dictionary of dictionaries, whose keys are primes \( \text{complement\_ev\_dict}[q] \) 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
dsage: drop_vector((1, 2, 5), 7, 11, {11: {11: {(1, 1, 3): [(1, 1, 3), (2, 3, 4)]}}})
True
```
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)], (0, 5, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)], (0, 3, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)], (1, 0, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)], (1, 2, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)], (1, 4, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)], (1, 2, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)]}

sage: drop_vector((0,1,0),3,7,P)
False

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 $\text{prime} \in \mathbb{Z}$ and $e,f \in \mathbb{N}$ are the ramification index and residue degree, respectively.

Note: $K$ has to be an absolute number field

EXAMPLES:

```
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
```

```
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)
-1261985118949117462968282807202378
```

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 $[b_0,b_1,b_2,b_3]$, checks 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:

```
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: L2 = [(2,1,4,3),(6,5,8,7)]
```

3.6. Solve $S$-unit equation $x + y = 1$ 285
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^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:

sage: from sage.rings.number_field.S_unit_solver import log_p
sage: K.<a> = NumberField(x^2+14)
\sage: pl = K.primes_above(3)[0]
\sage: pl
Fractional ideal (3, a + 1)
\sage: log_p(a+2, pl, 20)
8255385638/3*a + 15567609440/3

sage: K.<a> = NumberField(x^4+14)
\sage: pl = K.primes_above(5)[0]
\sage: pl
Fractional ideal (5, a + 1)
\sage: log_p(1/(a^2-4), pl, 30)
-42392683853751591352946/25*a^3 - 113099841599709611260219/25*a^2 - 8496494127064033599196/5*a - 18774052619501226990432/25

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: 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

sage: K.<a> = NumberField(x^4+14)

sage: p1 = K.primes_above(5)[0]

sage: p1
Fractional ideal (5, a + 1)

sage: log_p_series_part(1/(a^2-4), p1, 30)
56289408832104653692224688048459896543498793204839654215019548006062122195901510657655581925236618

sage: B = matrix(ZZ, 2, [1,1,1,0])

sage: y = vector(ZZ, [2,1])

sage: minimal_vector(B, y)
1/2

sage: B = random_matrix(ZZ, 3)

sage: y = vector(ZZ, [2,1])

sage: minimal_vector(B, y)
15/28
```

Sage.rings.number_field.S_unit_solver.minimal_vector\((A, y, prec=106)\)

**INPUT:**
- \(A\) : a square \(n\) by \(n\) non-singular integer matrix whose rows generate a lattice \(\mathcal{L}\)
- \(y\) : a row \((1 \times n)\) vector with integer coordinates
- \(prec\) : precision of real field (default: 106)

**OUTPUT:**
A lower bound for the square of

\[
\ell(\mathcal{L}, \vec{y}) = \begin{cases} 
\min_{\vec{x} \in \mathcal{L}} \| \vec{x} - \vec{y} \|, & \vec{y} \notin \mathcal{L} \\
\min_{\vec{y} \notin \mathcal{L}} \| \vec{y} \|, & \vec{y} \in \mathcal{L}.
\end{cases}
\]

**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: B = matrix(ZZ, 2, [1,1,1,0])

sage: y = vector(ZZ, [2,1])

sage: minimal_vector(B, y)
1/2

sage: B = random_matrix(ZZ, 3)

sage: y = vector(ZZ, [2,1])

sage: minimal_vector(B, y)
15/28
```

Sage.rings.number_field.S_unit_solver.mus\((\text{SUK}, v)\)

Return a list \([\mu]\), for \(\mu\) defined on pp. 824-825 of TCDF, [Sma1995]

**INPUT:**

3.6. Solve S-unit equation \(x + y = 1\) 287
• 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: K.<xi> = NumberField(x^3-3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: v_fin = tuple(K.primes_above(3))[0]
sage: mus(SUK, v_fin)
[xi^2 - 2]
```

REFERENCES:
• [Sma1995] pp. 824-825

sage.rings.number_field.S_unit_solver.p_adicLLL_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_adicLLL_bound
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_adicLLL_bound(SUK, A, prec)
89
```

sage.rings.number_field.S_unit_solver.p_adicLLL_bound_one_prime(prime, B0, M, M_logp, m0, c3, prec=106)

INPUT:
• prime – a prime ideal of a number field $K$
• $B_0$ – 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 $M$
• $m_0$ – an element of $K$, this is $\mu_0$ from Lemma IX.3 of [Sma1998]
• $c_3$ – a positive real constant
• prec – the precision of the calculations (default: 106)

OUTPUT:

A pair 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 $c_5$ 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: 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]
```

```python
sage: Log_p_Mus = [185056824593551109742400*a^2 + 1389583284398773572269676*a + 71789798769185258770249]
```

```python
sage: mu0 = K(-1)
```

```python
sage: c3_value = 0.42578591347980
```

```python
sage: m0_Kv_new, increase_precision = p_adic_LLL_bound_one_prime(v, K0_old, Mus, Log_p_Mus, mu0, c3_value, prec)
```

```python
sage: m0_Kv_new
```

```python
0
```

```python
sage: increase_precision
```

```python
True
```

And now we increase the precision to make it all work:

```python
sage: prec = 106
sage: K0_old = 9.4755766731092754280257946930e17
sage: Log_p_Mus = [1029563604390986737334686387890424583658678662701816*a^2 + 661450700156368458475507052066889190195530948403866*a]
```

```python
sage: c3_value = 0.4257859134798034746197327286726
```

```python
sage: m0_Kv_new, increase_precision = p_adic_LLL_bound_one_prime(v, K0_old, Mus, Log_p_Mus, mu0, c3_value, prec)
```

```python
sage: m0_Kv_new
```

```python
476
```

```python
sage: increase_precision
```

```python
False
```

sage.rings.number_field.S_unit_solver.possible_mu0s(SUK, v)

Return a list $[\mu_0]$ of all possible $\mu_0$ values defined on pp. 824-825 of TCDF, [Sma1995]

INPUT:

• SUK – a group of $S$-units
• v – a finite place of $K$
A list \([\mu_0s]\) where each \(\mu_0\) is an element of \(K\)

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import possible_mu0s
sage: K.<xi> = NumberField(x^3-3)
sage: S = tuple(K.primes_above(3))
sage: SUK = UnitGroup(K, S=S)
sage: v_fin = S[0]
sage: 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:**

- [Sma1995] pp. 824-825, but we modify the definition of \(\sigma\) (\(\sigma_{\tilde{t}}\)) to make it easier to code

```python
sage.rings.number_field.S_unit_solver.reduction_step_complex_case(place, B0, G, g0, c7)
```

**INPUT:**

- \(place\) – (ring morphism) a complex place of a number field \(K\)
- \(B0\) – the initial bound
- \(G\) – a set of generators of the free part of the group
- \(g0\) – an element of the torsion part of the group
- \(c7\) – 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 \(c7\) in the reference page 138

**REFERENCES:**

See [Sma1998].

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import reduction_step_complex_case
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: pl = K.places(prec=100)[1]
```
sage: reduction_step_complex_case(p1, 10^5, G, -1, 2)
(17, False)

sage: reduction_step_real_case(place, B0, G, c7)

INPUT:

• place – (ring morphism) a real place of a number field \( K \)
• B0 – the initial bound
• G – a set of generators of the free part of the group
• c7 – 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 \( c7 \) in the reference page 137

REFERENCES:

• [Sma1998]

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import reduction_step_real_case
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.real_places(prec=300)[0]

sage: reduction_step_real_case(p1, 10**10, G, 2)
(58, False)
```

sage: 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 \( (p_i-1) \) for primes \( p_i \) greater than bound (default: [])

3.6. Solve S-unit equation \( x + y = 1 \)
• `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: 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)
sage: expected = 
    ...: ((5, -1), (4, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
    ...: ((2, 1), (4, 0), xi + 2, -xi - 1),
    ...: ((2, 0), (1, 1), xi, -xi + 1),
    ...: ((5, 0), (1, 0), -xi, xi + 1)
sage: eq_up_to_order(sols, expected)
True
```

```python
sage.rings.number_field.S_unit_solver.sieve_ordering(SUK, q)
```

Returns ordered data for running sieve on the primes in \(SU K\) 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, \([\text{ideals_over}_q, \text{residue_fields}, \text{rho_images}, \text{product_rho_orders}]\), where

1. `ideals_over_q` is a list of the \(d = [K : \mathbb{Q}]\) 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: K.<xi> = NumberField(x^3 - 3*x + 1)
sage: SUK = K.S_unit_group(S=3)
sage: sieve_data = list(sieve_ordering(SUK, 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, 9, 16, 8], [18, 7, 10, 4], [18, 3, 12, 10])
sage: sieve_data[3]
(972, 972, 3888)
```

```python
sage.rings.number_field.S_unit_solver.solutions_from_systems(SUK, bound, cs_list, split_primes_list)
```

Lifts compatible systems to the integers and returns the S-unit equation solutions 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 $q$
- `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_exp`
4. the S-unit corresponding to `cv`, `iota_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_exp + iota_comp == 1`

**EXAMPLES:**

Given a single compatible system, a solution can be found.

### 3.6. Solve S-unit equation $x + y = 1$
sage: from sage.rings.number_field.S_unit_solver import solutions_from_systems
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, 5], [0, 5]], [[0, 15], [0, 15]]]
sage: solutions_from_systems(SUK, 20, a_compatible_system, split_primes_list)
[((0, -1), (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 \( K \)
- \( \text{prec} \) – precision used for computations in real, complex, and p-adic fields (default: 106)
- \( \text{include_exponents} \) – whether to include the exponent vectors in the returned value (default: True).
- \( \text{include_bound} \) – whether to return the final computed bound (default: False)
- \( \text{verbose} \) – whether to print information during the sieving step (default: False)

OUTPUT:

A list of tuples \( [(A_1, B_1, x_1, y_1), (A_2, B_2, x_2, y_2), \ldots] \) 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 \( K \) 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 \( \text{include_bound} \), 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:

sage: from sage.rings.number_field.S_unit_solver import solve_S_unit_equation, eq_up_to_order
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 = [(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]]
sage: eq_up_to_order(sols, expected)
True

In order to see the bound as well use the optional parameter \( \text{include_bound} \):
sage: solutions, bound = solve_S_unit_equation(K, S, 100, include_bound=True)
sage: bound
2

You can omit the exponent vectors:

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

It is an error to use values in S that are not primes in K:

sage: solve_S_unit_equation(K, [3], 200)
Traceback (most recent call last):
  ... ValueError: S must consist only of prime ideals, or a single element from which a prime ideal can be constructed.

We check the case that the rank is 0:

sage: K.<xi> = NumberField(x^2+x+1)
sage: solve_S_unit_equation(K, [])
[((1,), (5,), xi + 1, -xi)]

sage.rings.number_field.S_unit_solver.split_primes_large_lcm(SUK, bound)
Return a list L of rational primes q which split completely in K and which have desirable properties (see NOTE).

INPUT:

• 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 \( \text{lcm} \{ q-1 : q \text{ in L} \} \) 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 bound.

• The ordering of this set is not very intelligent for the purposes of the later sieving processes.

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import split_primes_large_lcm
sage: K.<xi> = NumberField(x^3 - 3*x + 1)
sage: S = K.primes_above(3)
sage: SUK = UnitGroup(K,S=tuple(S))
sage: split_primes_large_lcm(SUK, 200)
[17, 19, 37, 53]
With a tiny bound, SAGE may ask you to increase the bound.

```
sage: from sage.rings.number_field.S_unit_solver import split_primes_large_lcm
sage: K.<xi> = NumberField(x^2 + 163)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(23)))
sage: split_primes_large_lcm(SUK, 8)
Traceback (most recent call last):
  ...  
ValueError: Not enough split primes found. Increase bound.
```

### 3.7 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 [C]). 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 [C]), 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.

**To do:**

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:**

```
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
```

The next two examples are for relative number fields.:
AUTHORS:

- Nick Alexander (2008)
- David Loeffler (2009): fixed a bug with relative fields
- Maarten Derickx (2017): fixed a bug with number fields not generated by an integral element

class sage.rings.number_field.small_primes_of_degree_one.Small_primes_of_degree_one_iter(field, num_integer_primes=10000, max_iterations=100)

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:

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)]

See also:
sage.rings.finite_rings.residue_field
4.1 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 primitive if it contains no proper subfields except \( \mathbb{Q} \).

See also \texttt{sage.rings.number_field.totallyreal_rel}, which handles the non-primitive case using relative extensions.

4.1.1 Algorithm

We use Hunter’s algorithm ([Cohen2000], Section 9.3) with modifications due to Takeuchi ([Takeuchi1999] and the author ([Voight2008]).

We enumerate polynomials \( f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_0 \). 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 [Takeuchi1999] for more details.

4.1.2 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]
\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!).

### 4.1.3 References

### 4.1.4 Authors

- John Voight (2007-10-17): Added pari functions to avoid recomputations.
- Craig Citro and John Voight (2007-11-04): Additional doctests and type checking.
- Craig Citro and John Voight (2008-02-10): Final modifications for submission.

This function enumerates 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 `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:

- \( n \) – (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 False) If True, then return the polynomials as sequences (for easier exporting to a file).
- \( \text{phc} \) – boolean or integer (default: False)
- \( \text{keep\_fields} \) – (boolean or integer, default: False) If \( \text{keep\_fields} \) is True, then keep fields up to \( B \cdot \log(B) \); if \( \text{keep\_fields} \) is an integer, then keep fields up to that integer.
- \( t_2 \) – (boolean or integer, default: False) If \( t_2 = T \), then keep only polynomials with \( t_2 \) norm \( \geq T \).
- \( \text{just\_print} \) – (boolean, default: False): if \( \text{just\_print} \) is not 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: 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 \([d, f]\), 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

\[\text{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 \( n \).

\textbf{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:

\begin{verbatim}
sage: [sage.rings.number_field.totallyreal.odlyzko_bound_totallyreal(n) for n in \[range(1,5)\]]
[1.0, 2.223, 3.61, 5.067]
\end{verbatim}

AUTHORS:

- John Voight (2007-09-03)
NOTES: The values are calculated by Martinet [Martinet1980].

sage.rings.number_field.totallyreal.weed_fields(S, lenS=0)
Function used internally by the enumerate_totallyreal_fields_prim() 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]]
```

### 4.2 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 \( \leq 2000 \).

```python
sage: ZZx = ZZ['x']
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 \( \leq 10 \).

```python
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 - 7/2)*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 - 2*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 - x^3 - 9*x^2 + 4*x + 16, xF^2 + (-1/2*t - 1/2)*xF - 4],
 [8000, x^4 - 10*x^2 + 20, xF^2 - t - 5],
 [8525, 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]]
```

```python
sage: [ f[0] for f in ls ]
```

(continues on next page)
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 $\mathbb{Q}(\zeta_7)^+$ with discriminant $\leq 17 \times 10^9$.

```python
sage: F.<t> = NumberField(ZZx([1,-4,3,1]))
sage: F.disc()
49
```

AUTHORS:

Enumerates 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 then it outputs some extra information. If `verbose` is a string then 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:

```python
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],

```
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:

```
sage: ZZx = ZZ['x']
sage: F.<t> = NumberField(x^2-2)
sage: enumerate_totallyreal_fields_rel(F, 1, 2000)
[[1, [-2, 0, 1], xF - 1]]
sage: enumerate_totallyreal_fields_rel(F, 2, 2000)
[[1600, x^4 - 6*x^2 + 4, xF^2 + xF - 1]]
sage: enumerate_totallyreal_fields_rel(F, 2, 2000, return_seqs=True)
[[9, 6, 5, 0], [[1600, [4, 0, -6, 0, 1], [-1, 1, 1]]]]
```

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 $\langle$lower, upper$\rangle, \ldots$ of lower and upper bounds, for each embedding

EXAMPLES:

```
sage: x = polygen(QQ)
sage: K.<alpha> = NumberField(x^2-2)
sage: eps = 10e-6
sage: C = [[0-eps,5+eps],[0-eps,10+eps]]
sage: ls = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
sage: sorted([ a.trace() for a in ls ])
[0, 2, 4, 4, 4, 6, 6, 6, 8, 8, 8, 10, 10, 10, 12, 12, 14]
sage: len(ls)
19

sage: v = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
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]
```

A cubic field:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3 - 16*x +16)
sage: eps = 10e-6
sage: C = [[0-eps,3*eps]]*3
sage: v = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
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]
```

Note that the output is platform dependent (sometimes a 5 is listed below, and sometimes it isn’t):
class sage.rings.number_field.totallyreal_rel.tr_data_rel(F, m, B, a=None)

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.

incr(f_out, verbose=False, haltk=0)

This function ‘increments’ the totally real data to the next value which satisfies the bounds essentially given by Rolle’s theorem, and returns the next polynomial in the sequence f_out.

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:
• 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.

4.3 Enumeration of Totally Real Fields

AUTHORS:
• 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:

\begin{verbatim}
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2+1'))
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2-1'))
\end{verbatim}

sage.rings.number_field.totallyreal_data.hermite_constant(n)

This function returns the nth Hermite constant

The nth 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 $\gamma_n$. 

306 Chapter 4. Totally Real Fields
INPUT:

• \( n \) – integer

OUTPUT:

• (an upper bound for) the Hermite constant \( \gamma_n \)

EXAMPLES:

```
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
```

Note: The upper bounds used can be found in [CS] and [CE].

REFERENCES:

AUTHORS:

• John Voight (2007-09-03)

```
sage.rings.number_field.totallyreal_data.int_has_small_square_divisor(d)
Returns the largest \( a \) such that \( a^2 \) divides \( d \) and \( a \) has prime divisors < 200.
```

EXAMPLES:

```
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
```

```
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 \( 1 \leq r \leq n-2 \), we solve
\[
r^i x^i + (n-1-r)^i y^i + z^i = s_i \quad (i = 1, 2, 3)
\]
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.
```

EXAMPLES:

```
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.rings.number_field.totallyreal_data.lagrange_degree_3(3,6,1,2) # random
[-5.8878, -5.8878]
```

4.3. Enumeration of Totally Real Fields
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.py, so see that file for examples and further documentation.

increment (verbose=False, haltk=0, phc=False)

This function ‘increments’ the totally real data to the next value which satisfies the bounds essentially given by Rolle’s theorem, and returns 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: 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()  
[ -3, -1, 1]
sage: 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 = [...]  
```

4.4 Enumeration of Totally Real Fields: PHC interface

AUTHORS:

– John Voight (2007-10-10):

• Zeroth attempt.
sage.rings.number_field.totallyreal_phc.coefficients_to_power_sums\( (n, m, a) \)

Takes the list \( a \), representing a list of initial coefficients of a (monic) polynomial of degree \( n \), and returns 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.

**NOTES:**

Uses Newton’s relations, which are classical.

**AUTHORS:**

- John Voight (2007-09-19)

**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]
```
5.1 Field of Algebraic Numbers

This is an implementation of 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 \( \mathbb{Q} \) is available with abbreviation \( \mathbb{QQbar} \); the field of algebraic reals has abbreviation \( \mathbb{AA} \).

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 \( \mathbf{RealIntervalFieldElement} \)) 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 \( \mathbf{NumberFieldElement} \).

An algebraic number can be coerced into \( \mathbf{ComplexIntervalField} \) (or \( \mathbf{RealIntervalField} \), 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
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
```

We can convert from symbolic expressions:

```
sage: QQbar(sqrt(-5))
2.236067977499790?*I
sage: AA(sqrt(2) + sqrt(3))
3.146264369941973?
sage: QQbar(I)
I
sage: AA(I)
Traceback (most recent call last):
  ... Value Error: Cannot coerce algebraic number with non-zero imaginary part to algebraic
```
sage: QQbar((-8)^(1/3))
1.000000000000000? + 1.732050807568878?*I
sage: AA((-8)^(1/3))
-2
sage: QQbar((-4)^(1/4))
1 + 1*I
sage: AA((-4)^(1/4))
Traceback (most recent call last):
  ...
ValueError: Cannot coerce algebraic number with non-zero imaginary part to algebraic,
→real

The coercion, however, goes in the other direction, since not all symbolic expressions are algebraic numbers:

sage: QQbar(sqrt(2)) + sqrt(3)
sqrt(3) + 1.414213562373095?
sage: QQbar(sqrt(2) + QQbar(sqrt(3)))
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:

sage: AA(-1)^(1/3)
-1
sage: QQbar(-1)^(1/3)
0.500000000000000? + 0.866025403784439?*I

We can explicitly coerce from \( \mathbb{Q}[I] \). (Technically, this is not quite kosher, since \( \mathbb{Q}[I] \) does not come with an embedding; 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: K.<im> = QQ[I]
sage: pythag = QQbar(3/5 + 4*im/5); pythag
4/5*I + 3/5
sage: pythag.abs() == 1
True

However, implicit coercion from \( \mathbb{Q}[I] \) is not allowed:

sage: QQbar(1) + im
Traceback (most recent call last):
  ...
TypeError: unsupported operand parent(s) for +: 'Algebraic Field' and 'Number Field
→in I with defining polynomial x^2 + 1'

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. Field of Algebraic Numbers
```
sage: phi = (1 + sqrt(AA(5))) / 2
sage: phi^2 == phi + 1
    True
sage: tau = (1 - sqrt(AA(5))) / 2
sage: tau^2 == tau + 1
    True
sage: phi + tau == 1
    True
sage: tau < 0
    True

sage: rt23 = sqrt(AA(2/3))
sage: rt35 = sqrt(AA(3/5))
sage: rt25 = sqrt(AA(2/5))
sage: rt23 * rt35 == rt25
    True
```

The Sage rings `AA` and `QQbar` can decide equalities between radical expressions (over the reals and complex numbers respectively):
```
sage: a = AA((2/(3*sqrt(3)) + 10/27)^(1/3) - 2/(9*(2/(3*sqrt(3)) + 10/27)^(1/3)) + 1/
˓→3)
sage: a
    1.000000000000000?
```

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
sage: r.real()
    0.1812324444698754?
```

(continues on next page)
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 \( \mathbb{A} \mathbb{I} \).):

```python
sage: R.<x> = QQ[]
sage: r = (x^3 + 8).roots(QQbar, multiplicities=False)[2]; r
1.000000000000000? + 1.732050807568878?*I
sage: r.abs() == 2
True
sage: r.norm() == 4
True
sage: (r+QQbar(I)).norm().minpoly()
\( x^2 - 10*x + 13 \)
```

We can compute the multiplicative order of an algebraic number:

```python
sage: QQbar(-1/2 + I*sqrt(3)/2).multiplicative_order()
3
sage: QQbar(-sqrt(3)/2 + I/2).multiplicative_order()
12
sage: (QQbar.zeta(23)**5).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^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 - rhs
0
sage: lhs._exact_value()
-106486994025108862293341329896926906002223831*a^9 +
-231745624910286133718183712802529035435800*a^8 +
-2725979062544225260555847346959458901265*a^7 +
-214646949004652376912957054411004410158065*a^6 +
-145430828640187180554510898657837637140321*a^5 +
-64580500008796664339372667222902512216589785*a^4 +
-2052219053800078449122081871454923124998263*a^3 +
-14238966128623353681821644902045640915516176*a^2 +
-6189329204392161001*a +
-64195816997811495783724756588012516273410 where a^10 - a^9 + a^7 - a^6 + a^5 - a^4 + a^3 - a + 0 and a in -1.1762808182599187
```

(continues on next page)
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:

```
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[
  v1 = 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:

```
sage: n == 109*rt2 + 89*rt3
True
sage: sage_input(n)
R.<x> = AA[
  v = AA.polynomial_root(AA.common_polynomial(x^4 - 4*x^2 + 1), CIF(RIF(-RR(0.51763809020504148), RR(0.51763809020504159))),
                     → -109*v^3 - 89*v^2 + 327*v + 178
```

We can also see that some computations (basically, those which are easy to perform exactly) are performed directly, instead of storing the computation tree:

```
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.<x> = AA[
  3*QQbar.polynomial_root(AA.common_polynomial(x^2 + x + 1), CIF(RIF(-RR(0.50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386), RR(0.86602540378443871)*QQbar(4*I)*QQbar.polynomial_root(AA.common_polynomial(x^4 + x^3 + x^2 + x + 1), CIF(RIF(RR(0.3090169943749474), RR(0.3090169943749474))),
                     → -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:

```
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), RIF(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 trac ticket #10255:

```python
sage: rt2 = sqrt(AA(2))
sage: rt3 = sqrt(QQbar(3))
sage: x = polygen(QQbar)
sage: nrt3 = AA.polynomial_root((x-rt2)*(x+rt3), RIF(-2, -1))
sage: one = AA.polynomial_root((x-rt2)*(x-rt3)*(x-nrt3)*(x-1-rt3-nrt3), RIF(0.9, 1.1))
sage: one
1.000000000000000?
```

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
```

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: t = QQbar(sqrt(2)); type(t._descr)
<class 'sage.rings.qqbar.ANRoot'>
sage: loads(dumps(t)) == QQbar(sqrt(2))
True
```

```python
sage: t.exactify(); type(t._descr)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: loads(dumps(t)) == QQbar(sqrt(2))
```

(continues on next page)
We can convert elements of \( \mathbb{QQbar} \) and \( AA \) into the following types: \( \text{float} \), \( \text{complex} \), \( \text{RDF} \), \( \text{CDF} \), \( \text{RR} \), \( \text{CC} \), \( \text{RIF} \), \( \text{CIF} \), \( \mathbb{ZZ} \), and \( \mathbb{QQ} \), with a few exceptions. (For the arbitrary-precision types, \( \text{RR} \), \( \text{CC} \), \( \text{RIF} \), and \( \text{CIF} \), it can convert into a field of arbitrary precision.)

Converting from \( \mathbb{QQbar} \) to a real type (\( \text{float} \), \( \text{RDF} \), \( \text{RIF} \), \( \mathbb{ZZ} \), or \( \mathbb{QQ} \)) succeeds only if the \( \mathbb{QQbar} \) is actually real (has an imaginary component of exactly zero). Converting from either \( AA \) or \( \mathbb{QQbar} \) to \( \mathbb{ZZ} \) or \( \mathbb{QQ} \) 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:

```
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, 3.1428571428571432, 1.61803398874989895, -13.0, 1.6181818181818182, -2.6457513110645907, None]
sage: convert_test_all(complex)
[(42+0j), (3.1428571428571432+0j), (1.618033988749895+0j), (-13+0j), (1.6181818181818182+0j), (-2.6457513110645907+0j), (0.30901699437494745+0.9510565162951536*I)]
sage: convert_test_all(RDF)
[42.0, 3.142857142857143?, 1.618033988749895?, -13.0, 1.618181818181819?, -2.64575131106459?, None]
sage: convert_test_all(CDF)
[42.0000000000000, 3.14285714285714, 1.61803398874989, -13.0000000000000, 1.61818181818182, -2.64575131106459, 0.30901699437494745 + 0.9510565162951536*I]
sage: convert_test_all(RR)
[42.00000000000000000, 3.1428571428571428535, 1.6180339887498948482, -13.00000000000000000, 1.6181818181818181818, -2.6457513110645911111, None]
sage: convert_test_all(CC)
[42.00000000000000000, 3.1428571428571428535, 1.6180339887498948482, -13.00000000000000000, 1.6181818181818181818, -2.6457513110645911111, None]
sage: convert_test_all(RIF)
[42.00000000000000000, 3.1428571428571428535, 1.6180339887498948482, -13.00000000000000000, 1.6181818181818181818, -2.6457513110645911111, None]
sage: convert_test_all(CIF)
[42.00000000000000000, 3.1428571428571428535, 1.6180339887498948482, -13.00000000000000000, 1.6181818181818181818, -2.6457513110645911111, None]
sage: convert_test_all(ZZ)
```
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))
sage: y2 = (1-x2**2).sqrt()
sage: x - x2
0.?e-18
sage: y - y2
0.?e-17
```

Ideally, in the above example we should be able to test $x \approx x_2$ and $y \approx y_2$ but this is currently infinitely long.

``` sage
sage.rings.qqbar.AA = Algebraic Real Field
```

```
class sage.rings.qqbar.ANBinaryExpr(left, right, op)
Bases: sage.rings.qqbar.ANDescr

Initialize this ANBinaryExpr.

EXAMPLES:

```
sage: t = QQbar(sqrt(2)) + QQbar(sqrt(3)); type(t._descr) # indirect doctest
<class 'sage.rings.qqbar.ANBinaryExpr'>
```

```
extaxify()

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 ANBinaryExpr).

EXAMPLES:

```
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: sage_input(sqrt(AA(2)) + 2, verify=True)
# Verified
R.<x> = AA[]
AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951))) + 2
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: 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: 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)
1 + AA(3)
sage: rt3 = QQbar(sqrt(3))
sage: one = rt3/rt3
sage: n = sqrt(AA(2)) + one
sage: sage_input(n)
1 + AA(3)
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))
sage: one = rt3/rt3
sage: n = sqrt(AA(2)) + one
sage: sage_input(n)
1 + AA(3)
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 *
sage: 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()
Whether this element is complex. Does not trigger exact computation, so may return True even if the
element is real.

EXAMPLES:

```
sage: x = (QQbar(sqrt(-2)) / QQbar(sqrt(-5)))._descr
sage: x.is_complex()
True
```

class sage.rings.qqbar.ANDescr

Bases: sage.structure.sage_object.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 sub-
classes are for internal use, and should not be used directly.

abs(n)
Absolute value of self.

EXAMPLES:

```
sage: a = QQbar(sqrt(2))
sage: b = a._descr
sage: b.abs(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

conjugate(n)
Complex conjugate of self.

EXAMPLES:

```
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.conjugate(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

imag(n)
Imaginary part of self.

EXAMPLES:

```
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.imag(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

invert(n)
1/self.

EXAMPLES:

```
sage: a = QQbar(sqrt(2))
sage: b = a._descr
sage: b.invert(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```
**is_simple()**

Check whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

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: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2b.exactify()
sage: rt2b._descr.is_simple()
True
sage: rt2b._descr.is_simple()
False
sage: rt2b.simplify()
sage: rt2b._descr.is_simple()
True
```

**neg**

Negation of self.

**EXAMPLES:**

```python
sage: a = QQbar(sqrt(2))
sage: b = a._descr
sage: b.neg(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

**norm**

Field norm of self from \( \overline{Q} \) to its real subfield \( A \), i.e.~the square of the usual complex absolute value.

**EXAMPLES:**

```python
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.norm(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

**real**

Real part of self.

**EXAMPLES:**

```python
sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.real(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>
```

---

**class** sage.rings.qqbar.ANExtensionElement (generator, value)

**Bases:** sage.rings.qqbar.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.
abs \((n)\)

Return the absolute value of self (square root of the norm).

EXAMPLES:

```
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)
```

conjugate \((n)\)

Negation of self.

EXAMPLES:

```
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.conjugate(a)
-1/3*a^3 + 2/3*a^2 - 4/3*a + 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.7247448713915890? + 1.573132184970987?*I
sage: b.conjugate("ham spam and eggs")
-1/3*a^3 + 2/3*a^2 - 4/3*a + 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -0.7247448713915890? + 1.573132184970987?*I
```

exactify

Return an exact representation of self. Since self is already exact, just return self.

EXAMPLES:

```
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:

```
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.
˓→exactify()
sage: type(v)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: v.field_element_value()
a
```

generator

Return the AlgebraicGenerator object corresponding to self.

EXAMPLES:

```
```
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.

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.<x> = AA[]
vector(QQbar, [4 - 3*I, QQbar.polynomial_root(AA.common_polynomial(x^6 + x^5 + x^4 + x^3 + x + 1), CIF(RIF(RR(0.62348980185873348), RR(0.62348980185873359)), RIF(RR(0.7818314824680298), RR(0.78183148246802991))))])
sage: sage_input(v, verify=True)
# Verified
R.<x> = AA[]
v = QQbar.polynomial_root(AA.common_polynomial(x^8 - x^7 + x^5 - x^4 + x^3 - x + 1), CIF(RIF(RR(0.91354545764260087), RR(0.91354545764260098)), RIF(RR(0.4067364307580015), RR(0.4067364307580021))))
v^5 + v^3
sage: v = QQbar(sqrt(AA(2)))
sage: v.exactify()
sage: sage_input(v, verify=True)
# Verified
R.<x> = AA[]
QQbar(AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951))))
sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: extel = ANExtensionElement(QQbar_I_generator, QQbar_I_generator.field()).gen() + 1
sage: extel.handle_sage_input(sib, False, True)
{(call: {atomic:QQbar}({binop:+ {atomic:1} {atomic:I}})}, True)

invert (n)

1/self.

EXAMPLES:

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.invert(a)
7/3*a^3 - 2/3*a^2 + 4/3*a - 12 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in 
˓→-0.7247448713915890? - 1.573132184970987?*I
sage: b.invert("ham spam and eggs")
7/3*a^3 - 2/3*a^2 + 4/3*a - 12 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in 
˓→-0.7247448713915890? - 1.573132184970987?*I

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:

```python
sage: rt2 = QQbar(sqrt(2))
sage: rtm3 = QQbar(sqrt(-3))
sage: x = rtm3 + rt2 - rtm3
sage: x.exactify()
sage: y = x._descr
sage: type(y)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: y.is_complex()
True
sage: x.imag() == 0
True
```

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: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2.exactify()
sage: rt2._descr
a where a^2 - 2 = 0 and a in 1.414213562373095?
sage: rt2._descr.is_simple()
True
sage: rt2b.exactify()
sage: rt2b._descr
a^3 - 3*a where a^4 - 4*a^2 + 1 = 0 and a in 1.931851652578137?
sage: rt2b._descr.is_simple()
False
```

minpoly()
Compute the minimal polynomial of this algebraic number.
EXAMPLES:

```python
sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.
sage: v.exactify()
sage: type(v)
```

(continues on next page)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: v.minpoly()
x^2 - x - 1

neg(n)
Negation of self.

EXAMPLES:

```
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.neg(a)
1/3*a^3 - 2/3*a^2 + 4/3*a - 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -→ 0.7247448713915890? - 1.573132184970987?*I
sage: b.neg("ham spam and eggs")
1/3*a^3 - 2/3*a^2 + 4/3*a - 2 where a^4 - 2*a^3 + a^2 - 6*a + 9 = 0 and a in -→ 0.7247448713915890? - 1.573132184970987?*I
```

norm(n)
Norm of self (square of complex absolute value)

EXAMPLES:

```
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:

```
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 \( \text{AA} \) or \( \text{QQbar} \) corresponding to this descriptor.

**EXAMPLES:**

```python
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2b.exactify()
sage: rt2b._descr
a^3 - 3*a where a^4 - 4*a^2 + 1 = 0 and a in 1.931851652578137?
sage: rt2b._descr.simplify(rt2b)
a where a^2 - 2 = 0 and a in 1.414213562373095?
```

**class** `sage.rings.qqbar.ANRational(x)`

**Bases:** `sage.rings.qqbar.ANDescr`

The subclass of \( \text{ANDescr} \) 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 \( \text{self} \) 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:**

```python
sage: a = QQbar(1/3)._descr
sage: a.exactify() is a
True
```

**generator** ()

Return an \( \text{AlgebraicGenerator} \) object associated to this element. Returns the trivial generator, since self is rational.

**EXAMPLES:**

```python
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).

```python
sage: sage_input(QQbar(22/7), verify=True)
# Verified
QQbar(22/7)
sage: sage_input(-AA(3)/5, verify=True)
# Verified
AA(-3/5)
sage: sage_input(vector(AA, (0, 1/2, 1/3)), verify=True)
# Verified
vector(AA, [0, 1/2, 1/3])
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)
```

**invert** *(n)*

1/self.

**EXAMPLES:**

```python
sage: a = QQbar(3)
sage: b = a._descr
sage: b.invert(a)
1/3
```

**is_complex()**

Return False, since rational numbers are real

**EXAMPLES:**

```python
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:**

```python
sage: AA(1/2)._descr.is_simple()
True
```

**minpoly()**

Return the min poly of self over Q.

**EXAMPLES:**

```python
sage: QQbar(7)._descr.minpoly()
x - 7
```
**neg** \( (n) \)
Negation of self.

**EXAMPLES:**
```
sage: a = QQbar(3)
sage: b = a._descr
type(b) <class 'sage.rings.qqbar.ANRational'>
sage: b.neg(a)
-3
```

**rational_argument** \( (n) \)
Return the argument of self divided by \( 2\pi \), or None if this element is 0.

**EXAMPLES:**
```
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
```

**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:**
```
sage: QQbar(-3)._descr.scale()
-3
```

**class** sage.rings.qqbar.ANRoot \( (poly, interval, multiplicity=1) \)
**Bases:** sage.rings.qqbar.ANDescr

The subclass of 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 ANRoot object.

**EXAMPLES:**
```
sage: a = (x^2 + 23).roots(ring=QQbar, multiplicities=False)[0]
sage: b = a._descr
type(b) <class 'sage.rings.qqbar.ANRoot'>
sage: c = b.conjugate(a); c
<sage.rings.qqbar.ANUnaryExpr object at ...>
sage: c.exactify()
-2*a + 1 where a^2 - a + 6 = 0 and a in 0.50000000000000000? - 2.
\rightarrow397915761656360?*I
```

**exactify**
Return either an ANRational or an ANExtensionElement with the same value as this number.

**EXAMPLES:**
```sage
from sage.rings.qqbar import ANRoot
x = polygen(QQbar)
two = ANRoot((x-2)*(x-sqrt(QQbar(2))), RIF(1.9, 2.1))
two.exactify()
2
strange = ANRoot(x^2 + sqrt(QQbar(3))*x - sqrt(QQbar(2)), RIF(-0, 1))
strange.exactify()
a where a^8 - 6*a^6 + 5*a^4 - 12*a^2 + 4 = 0 and a in 0.6051012265139511?
```

**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:**

```sage
sage: sage_input((AA(3)^(1/2))^(1/3), verify=True)
# Verified
R.<x> = AA[]
AA.polynomial_root(AA.common_polynomial(x^3 - 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.):

```sage
sage: sage_input((sqrt(AA(3))^(5/7))^(9/4))
R.<x> = AA[]
v1 = AA.polynomial_root(AA.common_polynomial(x^2 + 7), CIF(RIF(-RR(3.
   →8954086044650791), -RR(3.8954086044650786)), RIF(RR(2.7639398015408925),
   →RR(2.7639398015408929))))
v2 = v1*v1
v3 = QQbar.polynomial_root(AA.common_polynomial(x^7 - v2*v2*v1), CIF(RIF(0.
   →8693488875796217), RR(0.86934888757962181)), RIF(RR(1.8052215661454434),
   RR(1.8052215661454436)))
v4 = v3*v3
v5 = v4*v4
```

```sage
sage: sage_input(AA.polynomial_root(x^2-x-1, RIF(1, 2)), verify=True)
# Verified
R.<x> = AA[]
AA.polynomial_root(AA.common_polynomial(x^2 - x - 1), RIF(RR(1.6180339887498947),
   RR(1.6180339887498949)))
v1 = AA.polynomial_root(AA.common_polynomial(x^2 - x - 1), RIF(1, 2), verify=True)
# Verified
R.<x> = AA[]
AA.polynomial_root(AA.common_polynomial(x^2 - x - 1), RIF(1, 2), verify=True)
```

(continues on next page)
**Sage Reference Manual: Algebraic Numbers and Number Fields, Release 8.7**

(continued from previous page)

```python
R.<x> = AA[]
QQbar.polynomial_root(AA.common_polynomial(x^3 - 5), CIF(RIF(-RR(0.
˓→8549879733838483), -RR(0.85498797338384842)), RIF(RR(1.4808826096823642),
˓→RR(1.4808826096823644))))
sage: from sage.rings.qqbar import *
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: rt = ANRoot(x^3 - 2, RIF(0, 4))
sage: rt.handle_sage_input(sib, False, True)
{{call: {getattr: {atomic:QQbar}.polynomial_root}({call: {getattr: {atomic:AA}
˓→→.common_polynomial}({binop:- {binop:** {gen:x {constr_parent: {subscr:
˓→→(atomic:AA)[{atomic:'x']}} with gens: ('x',)} {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 Q (rather than 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()  # indirect doctest
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
sage: from sage.rings.qqbar import ANRoot
sage: x = polygen(AA)
sage: rt2 = ANRoot(x^2 - 2, RIF(0, 2))
sage: rt2.refine_interval(RIF(0, 2), 75)
1.4142135623730950488017?
```

**class** `sage.rings.qqbar.ANUnaryExpr (arg, op)`

**Bases:** `sage.rings.qqbar.ANDescr`

Initialize this ANUnaryExpr.

**EXAMPLES:**

```python
sage: t = ~QQbar(sqrt(2)); type(t._descr)  # indirect doctest
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

**exactify()**

Trigger exact computation of self.

---

5.1. Field of Algebraic Numbers 331
EXAMPLES:

```python
sage: v = (-QQbar(sqrt(2)))._descr
sage: type(v)
<class 'sage.rings.qqbar.ANUnaryExpr'>
sage: v.exactify()
-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 \texttt{ANUnaryExpr}).

EXAMPLES:

```python
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),
    → RR(1.4142135623730951)))
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),
    → RR(1.4142135623730951)))
sage: sage_input(sqrt(QQbar(-3)).conjugate(), 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))))).conjugate()
sage: sage_input(QQbar.zeta(3).real(), verify=True)
# Verified
R.<x> = AA[]
QQbar.polynomial_root(AA.common_polynomial(x^2 + x + 1), CIF(RIF(-RR(0.50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386), RR(0.86602540378443871))).real()
sage: sage_input(QQbar.zeta(3).imag(), verify=True)
# Verified
R.<x> = AA[]
QQbar.polynomial_root(AA.common_polynomial(x^2 + x + 1), CIF(RIF(-RR(0.50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386), RR(0.86602540378443871))).imag()
sage: sage_input(abs(sqrt(QQbar(-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: sage_input(sqrt(QQbar(-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: sage_input(QQbar(QQbar.zeta(3).real()), verify=True)
# Verified
R.<x> = AA[]
QQbar(QQbar.polynomial_root(AA.common_polynomial(x^2 + x + 1), CIF(RIF(-RR(0.50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386), RR(0.86602540378443871))).real())
sage: from sage.rings.qqbar import *
```

(continues on next page)
```
sage: from sage.misc.sage_input import SageInputBuilder
sage: sib = SageInputBuilder()
sage: unexp = ANUnaryExpr(sqrt(AA(2)), '~~')
sage: unexp.handle_sage_input(sib, False, False)

{unop:~ {call: {getattr: {atomic:AA}.polynomial_root}({call: {getattr:
  \rightarrow{atomic:AA}.common_polynomial}({binop:- {binop:** {gen:x {constr_parent:
  \rightarrow{subscr: {atomic:AA}[[atomic:'x']] with gens: ('x',)]} {atomic:2}}
  \rightarrow{atomic:2}]}}, {call: {atomic:RIF}({call: {atomic:RR}({atomic:1.
  \rightarrow4142135623730949})}, {call: {atomic:RR}({atomic:1.4142135623730951})})}}},
  True)

sage: unexp.handle_sage_input(sib, False, True)

{call: {atomic:QQbar}({unop:~ {call: {getattr: {atomic:AA}.polynomial_root}(
  \rightarrow{call: {getattr: {atomic:AA}.common_polynomial}({binop:- {binop:** {gen:x
  \rightarrow{constr_parent: {subscr: {atomic:AA}[[atomic:'x']] with gens: ('x',)]} {atomic:2}}
  \rightarrow{atomic:2}]}}, {call: {atomic:RIF}({call: {atomic:RR}({atomic:1.
  \rightarrow4142135623730949})}, {call: {atomic:RR}({atomic:1.4142135623730951})})}})}}
  \rightarrow, True)

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:
```
sage: t = AA(sqrt(2))
sage: s = (-t)._descr
sage: s
<sage.rings.qqbar.ANUnaryExpr object at ...>
sage: s.is_complex()
False
sage: QQbar(-sqrt(2))._descr.is_complex()
True
```

class sage.rings.qqbar.AlgebraicField

Bases: sage.misc.fast_methods.Singleton, sage.rings.qqbar.AlgebraicField_common

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:
```
sage: QQbar.algebraic_closure()
Algebraic Field
```

completion (p, prec, 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
- $prec$ – precision of approximate field to return
- $extras$ – (optional) a dict of extra keyword arguments for the RealField constructor

EXAMPLES:
```
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
```

**construction()**

Return a functor that constructs self (used by the coercion machinery).

**EXAMPLES:**

```
sage: QQbar.construction()
(AlgebraicClosureFunctor, Rational Field)
```

**gen** (n=0)

Return the \(n\)-th element of the tuple returned by `gens()`.

**EXAMPLES:**

```
sage: QQbar.gen(0)
I
sage: QQbar.gen(1)
Traceback (most recent call last):
... 
IndexError: n must be 0
```

**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.

**EXAMPLES:**

```
sage: QQbar.gens()
(I,)
```

**ngens()**

Return the size of the tuple returned by `gens()`.

**EXAMPLES:**

```
sage: QQbar.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 \texttt{QQbar}.
\texttt{common_polynomial} to get a shared polynomial.

**EXAMPLES:**

```python
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
```

```
random_element (poly_degree=2, *args, **kwds)
```

Return a random algebraic number.

**INPUT:**

- \texttt{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.
- \texttt{args, kwds} - arguments and keywords passed to the random number generator for elements of \texttt{ZZ}, the integers. See \texttt{random_element()} for details, or see example below.

**OUTPUT:**

An element of \texttt{QQbar}, the field of algebraic numbers (see \texttt{sage.rings.qqbar}).

**ALGORITHM:**

A polynomial with degree between 1 and \texttt{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, \pm 1, \pm 2.

**EXAMPLES:**

```python
sage: a = QQbar.random_element()
sage: a
# random
0.2626138748742799? + 0.8769062830975992?*I
sage: a in QQbar
True

sage: b = QQbar.random_element(poly_degree=20)
```

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.

```python
sage: z = [QQbar.random_element(x=1, y=10) for _ in range(20)]
sage: QQbar(0) in z
False
```

If you just want real algebraic numbers you can filter them out. Using an odd degree for the polynomials will insure some degree of success. ::

```python
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
```

**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()     # default 4
I
sage: QQbar.zeta(5)
0.3090169943749474? + 0.9510565162951536?*I
sage: QQbar.zeta(3000)
0.999997806755380? + 0.002094393571219374?*I
```

class sage.rings.qqbar.AlgebraicField_common

Bases: sage.rings.ring.Field

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, always returns 0.

**EXAMPLES:**

```python
sage: AA.characteristic()
0
```

**common_polynomial**(poly)

Given a polynomial with algebraic coefficients, returns 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

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
0.2e-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
sage: p = AA.common_polynomial(x^3 - x - 1)
sage: r1 = AA.polynomial_root(p, RIF(1.3, 1.4)); r1
1.324717957244746?
sage: r2 = QQbar.polynomial_root(p, CIF(RIF(-0.7, -0.6), RIF(0.5, 0.6)));
```

**default_interval_prec()**

Return the default interval precision used for root isolation.

**EXAMPLES:**

```python
sage: AA.default_interval_prec()
64
```

**options(*get_value, **set_value)**

**OPTIONS:**

- `display_format` (default: decimal)
  - `decimal` – Always display a decimal approximation
  - `radical` – Display using radicals (if possible)

See `GlobalOptions` for more features of these options.

**order()**

Return the cardinality of self. Since this class is only used for fields of characteristic 0, always returns Infinity.

**EXAMPLES:**

```python
sage: QQbar.order()
+Infinity
```

**class sage.rings.qqbar.AlgebraicGenerator(field, root)**

**Bases:** `sage.structure.sage_object.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:**

```python
sage: from sage.rings.qqbar import AlgebraicGenerator
sage: x = polygen(QQ); f = x^4 + x + 17
sage: nf = NumberField(f, name='a')
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 -1.436449974830917? + 1.374535713065812?*I
```

**field()**
Return the number field attached to self.

**EXAMPLES:**

```python
sage: from sage.rings.qqbar import qq_generator
sage: qq_generator.field()
Rational Field
```

**is_complex()**
Return True if this is a generator for a non-real number field.

**EXAMPLES:**

```python
sage: z7 = QQbar.zeta(7)
sage: g = z7._descr._generator
sage: g.is_complex()
True
```

**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 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_trivial()
False
```

**pari_field()**
Return the PARI field attached to this generator.

**EXAMPLES:**
```
sage: from sage.rings.qqbar import qq_generator
sage: qq_generator.pari_field()
Traceback (most recent call last):
...
ValueError: No PARI field attached to trivial generator
sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator, qq_generator
sage: y = polygen(QQ)
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.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 0. →5176380902050415?
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
```

---

5.1. Field of Algebraic Numbers 339
Given generators $\alpha$ and $\beta$, $\alpha . \text{union}(\beta)$ gives a generator for the number field $\mathbb{Q}[\alpha][\beta]$.

**EXAMPLES:**

```python
sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator, 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: 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.union(qq_generator) is gen2
True

sage: gen3.union(qq_generator) is gen3
True

sage: gen2.union(gen3)
Number Field in a with defining polynomial y^4 - 4*y^2 + 1 with a in 0.5176380902050415?
```

**class** `sage.rings.qqbar.AlgebraicGeneratorRelation`

A simple class for maintaining relations in the lattice of algebraic extensions.

```python
class sage.rings.qqbar.AlgebraicNumber(x)

Bases: sage.rings.qqbar.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`.

```python
x = QQbar.zeta(3); x
-0.500000000000000? + 0.866025403784439?*I
```

**EXAMPLES:**

```python
sage: x = QQbar(-1); x
-1

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 trac ticket #16964):
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:

\[
\begin{align*}
sage: & p1 = x^8 + 74*x^7 + 2300*x^6 + 38928*x^5 + 388193*x^4 + 2295312*x^3 + 7613898*x^2 + 12066806*x + 5477001 \\
sage: & p2 = p1((x-1)^2) \\
sage: & sum(1 for r in p2.roots(CC,False) if abs(r.real() - 1) < 0.0001) \\
16 \\
sage: & r1 = QQbar.polynomial_root(p2, CIF(1, (-4.1,-4.0))) \\
sage: & r2 = QQbar.polynomial_root(p2, CIF(1, (4.0, 4.1))) \\
sage: & all([r1<r2, r1==r1, r2==r2, r2>r1]) \\
True
\end{align*}
\]

Though, comparing roots which are not equal or conjugate is much slower because the algorithm needs to check the equality of the real parts:

\[
\begin{align*}
sage: & sorted(p2.roots(QQbar,False)) # long time - 3 secs \\
& \{1.000000000000000? - 4.016778562562223?*I, \\
& 1.000000000000000? - 3.850538755978243?*I, \\
& 1.000000000000000? - 3.390564396412898?*I, \\
& \ldots \\
& 1.000000000000000? + 3.390564396412898?*I, \\
& 1.000000000000000? + 3.850538755978243?*I, \\
& 1.000000000000000? + 4.016778562562223?*I\}
\end{align*}
\]

**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 corresponding RealField, then this will trigger exact computation, which may be very slow.

**EXAMPLES:**

\[
\begin{align*}
sage: & a = QQbar.zeta(9) + QQbar(I) + QQbar.zeta(9).conjugate(); a \\
& 1.532088802277797? + 1.000000000000000?*I \\
sage: & a.complex_exact(CIF) \\
& 1.53208886237957? + 1*I
\end{align*}
\]

**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 \( \text{prec} \) is the precision of the field.
EXAMPLES:

```python
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:

```python
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:

```python
sage: QQbar.zeta(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:

```python
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)
sage: (a - b).interval(CIF)
0.?e-19 + 0.?e-18*I
sage: (a - b).interval_exact(CIF)
0
```

multiplicative_order()

Compute the multiplicative order of this algebraic real 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 \( \abs{x} \) is very close to 1. If so, we compute \( x \) exactly and examine its argument.

EXAMPLES:

```python
sage: QQbar(-sqrt(3)/2 - I/2).multiplicative_order()
12
(continues on next page)
```
norm()

Return \( \textbf{self} \times \text{self.conjugate()} \).

This is the algebraic definition of norm, if we view \( \text{QQbar} \) as \( \text{AA}[I] \).

EXAMPLES:

\begin{verbatim}
sage: QQbar(3 + 4*I).norm() 25
sage: type(QQbar(I).norm())
<class 'sage.rings.qqbar.AlgebraicReal'>
sage: QQbar.zeta(1007).norm()
1.000000000000000?
\end{verbatim}

rational_argument()

Return the argument of \( \textbf{self} \), divided by \( 2\pi \), as long as this result is rational. Otherwise returns \textbf{None}. Always triggers exact computation.

EXAMPLES:

\begin{verbatim}
sage: QQbar((1+I)*(sqrt(2)+sqrt(5))).rational_argument() 1/8
sage: QQbar(-1 + I*sqrt(3)).rational_argument() 1/3
sage: QQbar(-1 - I*sqrt(3)).rational_argument() -1/3
sage: QQbar(3+4*I).rational_argument() is None True
sage: (QQbar(2)**(1/5) * QQbar.zeta(7)**2).rational_argument() 2/7
sage: (QQbar.zeta(73)**5).rational_argument() 5/73
sage: (QQbar.zeta(3)**65536).rational_argument() 1/3
\end{verbatim}

real()

Return the real part of \( \textbf{self} \).

EXAMPLES:

\begin{verbatim}
sage: QQbar.zeta(5).real() 0.3090169943749474?
\end{verbatim}

class \texttt{sage.rings.qqbar.AlgebraicNumberPowQQAction}(G, S)

\textbf{Bases}: \texttt{sage.categories.action.Action}

Implement powering of an algebraic number (an element of \texttt{QQbar} or \texttt{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:

```python
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 == fibonacci(10)
True
sage: QQbar(-8)^(1/3)
1.000000000000000? + 1.732050807568878?*I
sage: (QQbar(-8)^(1/3))^3
-8
sage: QQbar.zeta(7)^(1/3) * QQbar.zeta(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) == 4
True
sage: x = polygen(AA)
sage: phi = AA.polynomial_root(x^2 - x - 1, RIF(0, 2))
```

(continues on next page)
sage: tau = AA.polynomial_root(x^2 - x - 1, RIF(-2, 0))
sage: rt5 = AA(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

class sage.rings.qqbar.AlgebraicNumber_base(parent, x)

Bases: sage.structure.element.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
as_number_field_element (minimal=False)

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.

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
sage: QQbar(sqrt(8)).as_number_field_element()
(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
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: hom(elt) == rt
True
```

We see an example where we do not get the minimal number field unless we specify minimal=True:

```sage
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 |--> 0.5176380902050415?)
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 |--> 1.7320508075688878?)
```

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).

EXAMPLES:
The Sage Reference Manual: Algebraic Numbers and Number Fields, Release 8.7

```python
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
```

**exactify()**

Compute an exact representation for this number.

**EXAMPLES:**

```python
sage: two = QQbar(4).nth_root(4)^2
sage: two
2.000000000000000?
```

```python
sage: two.exactify()
```

```python
sage: two
2
```

**interval (field)**

Given an interval 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: RIF64 = RealIntervalField(64)
sage: x = AA(2).sqrt()
sage: y = x*x
sage: y = 1000 * y - 999 * y
```

```python
sage: y.interval_fast(RIF64)
2.000000000000000?
```

```python
sage: y.interval(RIF64)
2.000000000000000000?
```

```python
sage: CIF64 = ComplexIntervalField(64)
sage: x = QQbar.zeta(11)
sage: x.interval_fast(CIF64)
0.8412535328311811689? + 0.5406408174555975821?*I
```

```python
sage: x.interval(CIF64)
0.8412535328311811689? + 0.5406408174555975822?*I
```

The following implicitly use this method:

```python
sage: RIF(AA(5).sqrt())
2.236067977499790?
```

```python
sage: AA(-5).sqrt().interval(RIF)
Traceback (most recent call last):
  ...
TypeError: unable to convert 2.236067977499790?*I to real interval
```

**interval_diameter (diam)**

Compute an interval representation of self with \( \text{diameter}() \) at most \( \text{diam} \). The precision of the returned value is unpredictable.

**EXAMPLES:**
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 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:

```
sage: x = AA(2).sqrt()
sage: x.interval_fast(RIF)
1.414213562373095?
sage: x.interval_fast(RealIntervalField(200))
1.4142135623730950488016872420969807857?
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
```

is_integer()

Return True if this number is a integer

EXAMPLES:

```
sage: QQbar(2).is_integer()
True
sage: QQbar(1/2).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()
False
```
minpoly()
Compute the minimal polynomial of this algebraic number. The minimal polynomial is the monic polyno-
mial 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
```

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: 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
```

You can get all n-th roots of algebraic numbers:
```
sage: AA(-8).nth_root(3, all=True)
sage: QQbar(1+I).nth_root(4, all=True)
[1.0695539323639867? + 0.21274750472674317?*I, -0.21274750472674317? + 1.0695539323639867?*I, -1.0695539323639867? - 0.21274750472674317?*I, 0.21274750472674317? - 1.0695539323639867?*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:
Compute an exact representation for this number, in the smallest possible number field.

EXAMPLES:

```python
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
sage: rt2b = rt3 + rt2 - rt3
sage: rt2b.exactify()
sage: rt2b._exact_value()
a^3 - 3*a where a^4 - 4*a^2 + 1 = 0 and a in 1.931851652578137?
sage: rt2b.simplify()
sage: rt2b._exact_value()
a where a^2 - 2 = 0 and a in 1.414213562373095?
```

`simplify()`
Compute an exact representation for this number, in the smallest possible number field.

INPUT:

- `extend` - bool (default: True); ignored if self is in QQbar, or positive in AA. If self is negative in AA, do the following: if True, return a square root of self in QQbar, otherwise raise a ValueError.
- `all` - bool (default: False); if True, return a list of all square roots. If False, return just one square root, or raise an ValueError if self is a negative element of AA and extend=False.

OUTPUT:

```python
sage: sqrt(2) + 10^25
sage: p = a.minpoly()
sage: v = a._value
sage: f = ComplexIntervalField(v.prec())
sage: [f(b.rhs()).overlaps(f(v)) for b in SR(p).solve(x)]
[True, True]
sage: a.radical_expression()
sqrt(2) + 1000000000000000000000000
```
Either the principal square root of self, or a list of its square roots (with the principal one first).

EXAMPLES:

```python
sage: AA(2).sqrt()
1.414213562373095?
sage: QQbar(I).sqrt()
0.7071067811865475? + 0.7071067811865475?*I
sage: QQbar(I).sqrt(all=True)
[0.7071067811865475? + 0.7071067811865475?*I, -0.7071067811865475? - 0.
  →7071067811865475?*I]
sage: a = QQbar(0)
sage: a.sqrt()
0
sage: a.sqrt(all=True)
[0]
sage: a = AA(0)
sage: a.sqrt()
0
sage: a.sqrt(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?

```python
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: sage.structure.sage_object.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 AA.common_polynomial() or QQbar.
common_polynomial(), and should only be used as an argument to AA.polynomial_root() or QQbar.polynomial_root(). (It does not matter whether you create the common polynomial with AA.
common_polynomial() or 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
sage: QQbar.polynomial_root(P, RIF(1, 2))
sage: QQbar.polynomial_root(P, RIF(1, 2))
1.618033988749895?
```

**complex_roots** *(prec, multiplicity)*

Find the roots of self in the complex field to precision prec.

**EXAMPLES:**

```sage
sage: x = polygen(ZZ)
sage: cp = AA.common_polynomial(x^4 - 2)
```

Note that the precision is not guaranteed to find the tightest possible interval since complex_roots() depends on the underlying BLAS implementation.

```sage
sage: cp.complex_roots(30, 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
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()
sage: cp._exact
True
```

**factors**

**EXAMPLES:**

```sage
sage: x=polygen(QQ); f=QQbar.common_polynomial(x^4 + 4)
sage: f.factors()
[y^2 - 2*y + 2, y^2 + 2*y + 2]
```

**generator**

Return an `AlgebraicGenerator` for a number field containing all the coefficients of self.

**EXAMPLES:**

```sage
sage: x = polygen(AA)
sage: p = sqrt(AA(2)) * x^2 - sqrt(AA(3))
sage: cp = AA.common_polynomial(p)
sage: cp.generator()
Number Field in a with defining polynomial y^4 - 4*y^2 + 1 with a in 1.
```

**is_complex**

Return True if the coefficients of this polynomial are non-real.
EXAMPLES:

```sage
tax = polygen(QQ); f = x^3 - 7
g = AA.common_polynomial(f)
g.is_complex()
False
QQbar.common_polynomial(x^3 - QQbar(I)).is_complex()
True
```

`poly()`
Return the underlying polynomial of self.

EXAMPLES:

```sage
tax = polygen(QQ); f = x^3 - 7
g = AA.common_polynomial(f)
g.poly() == f
True
```

```sage
class sage.rings.qqbar.AlgebraicReal(x)
    Bases: sage.rings.qqbar.AlgebraicNumber_base

A real algebraic number.

_richcmp_(other, op)
Compare two algebraic reals.

EXAMPLES:

```sage
AA(2).sqrt() < AA(3).sqrt()
True
((5+AA(5).sqrt())/2).sqrt() == 2*QQbar.zeta(5).imag()
True
AA(3).sqrt() + AA(2).sqrt() < 3
False
```

`ceil()`
Return the smallest integer not smaller than self.

EXAMPLES:

```sage
AA(sqrt(2)).ceil()
2
AA(-sqrt(2)).ceil()
-1
AA(42).ceil()
42
```

`conjugate()`
Return the complex conjugate of self, i.e. returns itself.

EXAMPLES:

```sage
a = AA(sqrt(2) + sqrt(3))
a.conjugate()
3.146264369941973?
a.conjugate() is a
True
```
floor()  
Return the largest integer not greater than self.

EXAMPLES:

```
sage: AA(sqrt(2)).floor()
sage: AA(-sqrt(2)).floor()
sage: AA(42).floor()
```

imag()  
Return the imaginary part of this algebraic real.

It always returns 0.

EXAMPLES:

```
sage: a = AA(sqrt(2) + sqrt(3))
sage: a.imag()
sage: parent(a.imag())
```

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: x = AA(2).sqrt()
sage: y = x*x
sage: x.interval(RIF)
sage: x.interval_exact(RIF)
sage: y.interval(RIF)
sage: y.interval_exact(RIF)
sage: z = 1 + AA(2).sqrt() / 2^200
sage: z.interval(RIF)
sage: z.interval_exact(RIF)
```

real()  
Return the real part of this algebraic real.

It always returns self.

EXAMPLES:

```
sage: a = AA(sqrt(2) + sqrt(3))
sage: a.real()
```

(continues on next page)
sage: a.real() is a
True

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: x = AA(2).sqrt()^2
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 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'))
0.00000000000000
sage: y == 0.0 or y == -0.0 # same as above
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDU'))
0.00000000000000
sage: y == 0.0 or y == -0.0 # idem
True
sage: y = (x-2).real_exact(RealField(53, rnd='RNDZ'))
0.00000000000000
sage: y == 0.0 or y == -0.0 # ibidem
True
sage: y = AA(2).sqrt()
sage: y.real_exact(RR)
1.41421356237309
sage: y.real_exact(RealField(53, rnd='RNDD'))
1.41421356237309
sage: y.real_exact(RealField(53, rnd='RNDU'))
1.41421356237309
sage: y.real_exact(RealField(53, rnd='RNDZ'))
1.41421356237309

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 \(2 \times 2^{-(\text{field}.\text{prec}()-1)}\). Also, the rounding mode of the field is respected.

EXAMPLES:
round()

Round self to the nearest integer.

EXAMPLES:

```
sage: AA(sqrt(2)).round()
1
sage: AA(1/2).round()
1
sage: AA(-1/2).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()
```

(continues on next page)
0

\begin{verbatim}
sage: a = AA(2).sqrt() + AA(3).sqrt() - 58114382797550084497/18470915334626475921
sage: a.sign()
1
sage: b = AA(2).sqrt() + AA(3).sqrt() - 2602510228533039296408/827174681630786895911
sage: b.sign()
-1
sage: c = AA(5)**(1/3) - 1437624125539676934786/840727688792155114277
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
\end{verbatim}

\texttt{trunc()} 
Round \texttt{self} to the nearest integer toward zero.

EXAMPLES:

\begin{verbatim}
sage: AA(sqrt(2)).trunc()
1
sage: AA(-sqrt(2)).trunc()
-1
sage: AA(1).trunc()
1
sage: AA(-1).trunc()
-1
\end{verbatim}

\textbf{class sage.rings.qqbar.AlgebraicRealField}

Bases: \texttt{sage.misc.fast_methods.Singleton}, \texttt{sage.rings.qqbar.AlgebraicField_common}

The field of algebraic reals.

\textbf{algebraic_closure}()

Return the algebraic closure of this field, which is the field $\mathbb{Q}$ of algebraic numbers.

EXAMPLES:

\begin{verbatim}
sage: AA.algebraic_closure()
Algebraic Field
\end{verbatim}

\textbf{completion}(p, prec, extras=())

Return the completion of \texttt{self} at the place \texttt{p}. Only implemented for \texttt{p = \infty} at present.

5.1. Field of Algebraic Numbers
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 \texttt{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 \( \text{gens}() \).

EXAMPLES:

```
sage: AA.gen(0)
1
sage: AA.gen(1)
Traceback (most recent call last):
  ...
IndexError: n must be 0
```

\( \text{gens}() \)

Return a set of generators for this field. As this field is not finitely generated, we opt for just returning 1.

EXAMPLES:

```
sage: AA.gens()
(1,)
```

\( \text{ngens}() \)

Return the size of the tuple returned by \( \text{gens}() \).

EXAMPLES:

```
sage: AA.ngens()
1
```

\( \text{polynomial_root}(\text{poly}, \text{interval}, \text{multiplicity}=1) \)

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: 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
```

We allow complex polynomials, as long as the particular root in question is real.

```
sage: K.<im> = QQ[I]
sage: x = polygen(K)
sage: p = (im + 1) * (x^3 - 2); p
(I + 1)*x^3 - 2*I - 2
sage: r = AA.polynomial_root(p, RIF(1, 2)); r^3
2.000000000000000?
```

\textbf{zeta} (n=2)

Return an \( n \)-th root of unity in this field. This will raise a \texttt{ValueError} if \( n \not\in \{1, 2\} \) since no such root exists.

\textbf{INPUT:}

- \( n \) (integer) – default 2

\textbf{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
```

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
```

\texttt{sage.rings.qqbar.QQbar} = \texttt{Algebraic Field}
sage.rings.qqbar.an_binop_element \((a, b, op)\)
Add, subtract, multiply or divide two elements represented as elements of number fields.

**EXAMPLES:**

```python
sage: sqrt2 = QQbar(2).sqrt()
sage: sqrt3 = QQbar(3).sqrt()
sage: sqrt5 = QQbar(5).sqrt()
sage: a = sqrt2 + sqrt3; a.exactify()
sage: b = sqrt3 + sqrt5; 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 ...>
```

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.ANExtensionElement'>)
sage: a.exactify()
sage: b = sqrt17 * sqrt19 - sqrt17 + sqrt19 * (sqrt17 + 2)
sage: b, type(b._descr)
(40.53909377268655?, <class 'sage.rings.qqbar.ANExtensionElement'>)
```

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: a = QQbar(sqrt(2)) + QQbar(sqrt(3))
sage: b = QQbar(sqrt(3)) + QQbar(sqrt(5))
sage: type(a._descr); type(b._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: from sage.rings.qqbar import an_binop_expr
sage: x = an_binop_expr(a, b, operator.add); x
<sage.rings.qqbar.ANBinaryExpr object at ...>
sage: x.exactify()
-6/7*a^7 + 2/7*a^6 + 71/7*a^5 - 26/7*a^4 - 125/7*a^3 + 72/7*a^2 + 43/7*a - 47/7
where a^8 - 12*a^6 + 23*a^4 - 12*a^2 + 1 = 0 and a in 3.12580...?
sage: a = QQbar(sqrt(2)) + QQbar(sqrt(3))
```

(continues on next page)
sage: b = QQbar(sqrt(3)) + QQbar(sqrt(5))
sage: type(a._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: x = an_binop_expr(a, b, operator.mul); x
<sage.rings.qqbar.ANBinaryExpr object at ...>
sage: 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 - 12*a^6 + 23*a^4 - 12*a^2 + 1 = 0 and a in 3.1258...?

sage.rings.qqbar.an_binop_rational(a, b, op)
Used to add, subtract, multiply or divide algebraic numbers.

EXAMPLES:

sage: from sage.rings.qqbar import an_binop_rational
sage: f = an_binop_rational(QQbar(2), QQbar(3/7), operator.add)
sage: f
17/7
sage: type(f)
<class 'sage.rings.qqbar.ANRational'>
sage: f = an_binop_rational(QQbar(2), QQbar(3/7), operator.mul)
sage: f
6/7
sage: 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 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 an helper function for comparison of algebraic elements (i.e. the methods AlgebraicNumber._richcmp_() and 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:

```python
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:

```python
sage: from sage.rings.qqbar import conjugate_expand
sage: conjugate_expand(CIF(RIF(0, 1), RIF(1, 2))).str(style='brackets')
'[0.0000000000000000 .. 1.0000000000000000] + [1.0000000000000000 .. 2.0000000000000000]*I'
sage: conjugate_expand(CIF(RIF(0, 1), RIF(0, 1))).str(style='brackets')
'[0.0000000000000000 .. 1.0000000000000000] + [-1.0000000000000000 .. 1.0000000000000000]*I'
sage: conjugate_expand(CIF(RIF(0, 1), RIF(-2, 1))).str(style='brackets')
'[0.0000000000000000 .. 1.0000000000000000] + [-2.0000000000000000 .. 2.0000000000000000]*I'
sage: conjugate_expand(RIF(1, 2)).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000]'
```

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 iff 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')
'[3.000000000000000 .. 4.000000000000000]'
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]'
```

(continues on next page)
\begin{verbatim}
sage: conjugate_shrink(CIF(RIF(1, 2), RIF(-1, 2))).str(style='brackets') '[1.0000000000000000 .. 2.0000000000000000]
\end{verbatim}

\texttt{sage.rings.qqbar.do_polred \textit{(poly, threshold=32)}}

Find a polynomial of reasonably small discriminant that generates the same number field as \textit{poly}, using the \texttt{PARI polredbest} function.

**INPUT:**

- poly - a monic irreducible polynomial with integer coefficients
- threshold - an integer used to decide whether to run \texttt{polredbest}

**OUTPUT:**

A triple (\textit{elt_fwd}, \textit{elt_back}, \textit{new_poly}), where:

- \textit{new_poly} is the new polynomial defining the same number field,
- \textit{elt_fwd} is a polynomial expression for a root of the new polynomial in terms of a root of the original polynomial,
- \textit{elt_back} is a polynomial expression for a root of the original polynomial in terms of a root of the new polynomial.

**EXAMPLES:**

\begin{verbatim}
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)
\end{verbatim}

This shows that \texttt{trac ticket \#13054} has been fixed:

\begin{verbatim}
sage: do_polred(x^4 - 4294967296*x^2 + 54265257667816538374400)
(1/4*x, 4*x, x^4 - 268435456*x^2 + 211973662764908353025)
\end{verbatim}

\texttt{sage.rings.qqbar.find_zero_result \textit{(fn, l)}}

\textit{l} is a list of some sort. \textit{fn} is a function which maps an element of \textit{l} and a precision into an interval (either real or complex) of that precision, such that for sufficient precision, exactly one element of \textit{l} results in an interval containing 0. Returns that one element of \textit{l}.

**EXAMPLES:**

\begin{verbatim}
sage: from sage.rings.qqbar import find_zero_result
sage: _.<x> = QQ['x']
sage: delta = 10^(-70)
\end{verbatim}

\section{Field of Algebraic Numbers}

363
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:

sage: AA(golden_ratio)  # indirect doctest
1.618033988749895?

sage.rings.qqbar.is_AlgebraicField(F)
Check whether F is an AlgebraicField instance.

EXAMPLES:

sage: from sage.rings.qqbar import is_AlgebraicField
sage: [is_AlgebraicField(x) for x in [AA, QQbar, None, 0, "spam"]]
[False, True, False, False, False]

sage.rings.qqbar.is_AlgebraicField_common(F)
Check whether F is an AlgebraicField_common instance.

EXAMPLES:

sage: from sage.rings.qqbar import is_AlgebraicField_common
sage: [is_AlgebraicField_common(x) for x in [AA, QQbar, None, 0, "spam"]]
[True, True, False, False, False]

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)))
False
sage: is_AlgebraicNumber(QQbar(sqrt(2)))
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)))
True
sage: is_AlgebraicReal(QQbar(sqrt(2)))
False

(continued from previous page)

sage: is_AlgebraicReal("spam")
False

sage.rings.qqbar.is_AlgebraicRealField(F)
Check whether F is an AlgebraicRealField instance. For internal use.
EXAMPLES:
sage: from sage.rings.qqbar import is_AlgebraicRealField
sage: [is_AlgebraicRealField(x) for x in [AA, QQbar, None, 0, "spam"]]
[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:
sage: from sage.rings.qqbar import isolating_interval
sage: _.<x> = QQ['x']
sage: isolating_interval(lambda prec: sqrt(RealIntervalField(prec)(2)), x^2 - 2)
1.4142135623730950488?

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.
˓→0000000000000000000000000000000000000000000000000000000000000000000001000000000000000000000000
˓→

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

sage.rings.qqbar.number_field_elements_from_algebraics(numbers, minimal=False,
same_field=False)
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.
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

5.1. Field of Algebraic Numbers

365


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.):

```python
sage: x = polygen(QQ)
sage: p = x^3 + x^2 + x + 17
sage: rts = p.roots(ring=QQbar, multiplicities=False)
```

```python
sage: splitting = number_field_elements_from_algebraics(rts)[0]; splitting
Number Field in a with defining polynomial y^6 - 40*y^4 - 22*y^3 + 873*y^2 + 1386*y + 594
```

```python
sage: p.roots(ring=splitting)
[(361/29286*a^5 - 19/3254*a^4 - 14359/29286*a^3 + 401/29286*a^2 + 18183/1627*a + 15930/1627, 1), (49/117144*a^5 - 179/39048*a^4 - 3247/117144*a^3 + 22553/117144*a^2 + 60683/117144*a^3 - 24157/117144*a^2 - 56293/4881*a - 53033/6508, 1), (-1493/117144*a^5 + 407/39048*a^4 + 60683/117144*a^3 - 24157/117144*a^2 - 56293/4881*a - 53033/6508, 1)]
```

```python
sage: rt2 = AA(sqrt(2)); rt2
1.414213562373095?
```

```python
sage: rt3 = AA(sqrt(3)); rt3
1.732050807568878?
```

```python
sage: rt3a = QQbar(sqrt(3)); rt3a
1.732050807568878?
```

```python
sage: qqI = QQbar.zeta(4); qqI
I
```

```python
sage: z3 = QQbar.zeta(3); z3
-0.500000000000000? + 0.866025403784439?*I
```

```python
sage: rt2b = rt3 + rt2 - rt3; rt2b
1.414213562373095?
```

```python
sage: rt2c = z3 + rt2 - z3; rt2c
1.414213562373095? + 0.?e-19*I
```

```python
sage: number_field_elements_from_algebraics(rt2)
(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 |--> 1.414213562373095?)
```

```python
sage: number_field_elements_from_algebraics((rt2,rt3))
(Number Field in a with defining polynomial y^4 - 4*y^2 + 1, [-a^3 + 3*a, -a^2 + a - 2], Ring morphism:
  From: Number Field in a with defining polynomial y^4 - 4*y^2 + 1
  To:   Algebraic Real Field
  Defn: a |--> 0.5176380902050415?)
```

```python
sage: number_field_elements_from_algebraics(rt3a)
(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?)
```

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, same_field=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?)
```

(continues on next page)
We’ve created \( rt2b \) in such a way that sage does not initially know that it’s in a degree-2 extension of \( \mathbb{Q} \):

```
sage: number_field_elements_from_algebraics(rt2b)
(Number Field in a with defining polynomial y^4 - 4*y^2 + 1, -a^3 + 3*a, Ring morphism:
    From: Number Field in a with defining polynomial y^4 - 4*y^2 + 1
    To:  Algebraic Real Field
    Defn: a |--> 0.5176380902050415?)
```

We can specify minimal=True if we want the smallest number field:

```
sage: number_field_elements_from_algebraics(rt2b, minimal=True)
(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 |--> 1.414213562373095?)
```

Things work fine with rational numbers, too:

```
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 \( \mathbb{Q} \bar{\mathbb{Q}} \):

```
sage: number_field_elements_from_algebraics((sqrt(7), sqrt(9), sqrt(11)))
(Number Field in a with defining polynomial y^4 - 9*y^2 + 1, \([-a^3 + 8*a, 3, -a^3_\rightarrow 2*a^2, a^6, -a^4]\), Ring morphism:
    From: Number 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 \( \mathbb{Q} \bar{\mathbb{Q}} \):

```
sage: (fld,nums,hom) = number_field_elements_from_algebraics((rt2, rt3, qqI, z3))
sage: fld,nums,hom
# random
(Number Field in a with defining polynomial y^8 - y^4 + 1, \([-a^5 + a^3 + a, a^6 \rightarrow 2*a^2, a^6, -a^4]\), Ring morphism:
    From: Number Field in a with defining polynomial y^8 - y^4 + 1
    To:  Algebraic Field
    Defn: a |--> -0.2588190451025208? - 0.9659258262890683?*I)
sage: (nfrt2, nfrt3, nfI, nfz3) = nums
sage: hom(nfrt2)
1.414213562373095? + 0.?e-18*I
sage: nfrt2^2
2
sage: nfrt3^2
3
sage: nfz3 + nfz3^2
```

(continues on next page)
-1
sage: nfrt2 + nfrt3 + nfI + nfz3; sum
2*a^6 + a^5 - a^4 - a^3 - 2*a^2 - a
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]
True

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]

sage.rings.qqbar.rational_exact_root(r, d)
Checks whether the rational \( r \) is an exact \( d \)'th power. If so, returns the \( d \)'th root of \( r \); otherwise, 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

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)

sage.rings.qqbar.t1
alias of sage.rings.qqbar.ANBinaryExpr
sage.rings.qqbar.t2
alias of sage.rings.qqbar.ANRoot
sage.rings.qqbar.tail_prec_seq()
A generator over precisions larger than those in short_prec_seq().
EXAMPLES:

sage: from sage.rings.qqbar import tail_prec_seq
sage: g = tail_prec_seq()
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 Galois Abelian extension of the rational numbers.

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.

**Note:** There used to be a native Sage version of the universal cyclotomic field written by Christian Stump (see trac ticket #8327). It was slower on most operations and it was decided to use a version based on GAP instead (see trac ticket #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 trac ticket #18152.

**REFERENCES:**

**EXAMPLES:**

```python
sage: UCF = UniversalCyclotomicField(); UCF
Universal Cyclotomic Field
```

To generate cyclotomic elements:

```python
sage: UCF.gen(5)
E(5)
sage: UCF.gen(5,2)
E(5)^2
sage: E = UCF.gen
```

Equality and inequality checks:

```python
sage: E(6,2) == E(6)^2 == E(3)
True
sage: E(6)^2 != E(3)
False
```

Addition and multiplication:

```python
sage: E(2) * E(3)
-E(3)
sage: f = E(2) + E(3); f
2*E(3) + E(3)^2
```

Inverses:
```python
sage: f^-1
1/3*E(3) + 2/3*E(3)^2
sage: f.inverse()
1/3*E(3) + 2/3*E(3)^2
sage: f * f.inverse()
1
```

Conjugation and Galois conjugates:

```python
sage: f.conjugate()
E(3) + 2*E(3)^2
sage: f.galois_conjugates()
[2*E(3) + E(3)^2, E(3) + 2*E(3)^2]
sage: f.norm_of_galois_extension()
3
```

One can create matrices and polynomials:

```python
sage: m = matrix(2, [E(3), 1, 1, E(4)]); m
[ E(3) 1]
[ 1 E(4)]
sage: m.parent()
Full MatrixSpace of 2 by 2 dense matrices over Universal Cyclotomic Field
sage: m**2
[ -E(3) E(12)^4 - E(12)^7 - E(12)^11
  E(12)^4 - E(12)^7 - E(12)^11 0]
sage: m.charpoly()
x^2 + (-E(12)^4 + E(12)^7 + E(12)^11)*x + E(12)^4 + E(12)^7 + E(12)^8
sage: m.echelon_form()
[1 0]
[0 1]
sage: m.pivots()
(0, 1)
sage: m.rank()
2
```

AUTHORS:
- Christian Stump (2013): initial Sage version (see trac ticket #8327)
- Vincent Delecroix (2015): complete rewriting using libgap (see trac ticket #18152)
- Sebastian Oehms (2018): deleting the method is_finite since it returned the wrong result (see trac ticket #25686)
class sage.rings.universal_cyclotomic_field.UCFtoQQbar(UCF)
    Bases: sage.categories.morphism.Morphism

Conversion to QQbar.

EXAMPLES:

sage: UCF = UniversalCyclotomicField()
sage: QQbar(UCF.gen(3))
-0.500000000000000? + 0.866025403784439?*I
sage: CC(UCF.gen(7,2) + UCF.gen(7,6))
0.400968867902419 + 0.193096429713794*I
sage: complex(E(7)+E(7,2))
(0.40096886790241915+1.7567593946498534j)
sage: complex(UCF.one()/2)
(0.5+0j)

class sage.rings.universal_cyclotomic_field.UniversalCyclotomicField(names=None)
    Bases: sage.structure.unique_representation.UniqueRepresentation, sage.rings.
    ring.Field

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:

sage: UniversalCyclotomicField().algebraic_closure()
Algebraic Field

an_element()
    Return an element.

EXAMPLES:

sage: UniversalCyclotomicField().an_element()
E(5) - 3*E(5)^2

characteristic()
    Return the characteristic.

EXAMPLES:
sage: UniversalCyclotomicField().characteristic()
0
sage: parent(_)
Integer Ring

degree()
Return the degree of self as a field extension over the Rationals.

EXAMPLES:

sage: UCF = UniversalCyclotomicField()
sage: 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

There is an alias zeta also available:

sage: UCF.zeta(6)
-E(3)^2

is_exact()
Return True as this is an exact ring (i.e. not numerical).

EXAMPLES:

sage: UniversalCyclotomicField().is_exact()
True

one()
Return one.

EXAMPLES:

sage: UCF = UniversalCyclotomicField()
sage: UCF.one()
1
sage: parent(_)
Universal Cyclotomic Field

some_elements()
Return a tuple of some elements in the universal cyclotomic field.

EXAMPLES:
**zero()**

Return zero.

**EXAMPLES:**

```
sage: UCF = UniversalCyclotomicField()
sage: UCF.zero()
0
sage: 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:**

```
sage: UCF = UniversalCyclotomicField()
sage: UCF.gen(15)
E(15)
sage: UCF.gen(7,3)
E(7)^3
sage: UCF.gen(4,2)
-1
```

There is an alias `zeta` also available:

```
sage: UCF.zeta(6)
-E(3)^2
```

```python
class sage.ringsuniversal_cyclotomic_field.UniversalCyclotomicFieldElement

Bases: sage.structure.element.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:**

```
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

additive_order()
Return the additive order.

EXAMPLES:

sage: UCF = UniversalCyclotomicField()
sage: UCF.zero().additive_order()
0
sage: UCF.one().additive_order()
+Infinity
sage: UCF.gen(3).additive_order()
+Infinity

conductor()
Return the conductor of self.

EXAMPLES:

sage: E(3).conductor()
3
sage: (E(5) + E(3)).conductor()
15

conjugate()
Return the complex conjugate.

EXAMPLES:

sage: (E(7) + 3*E(7,2) - 5 * E(7,3)).conjugate()
-5*E(7)^4 + 3*E(7)^5 + E(7)^6

denominator()
Return the denominator of this element.

See also:

\texttt{is\_integral()}

EXAMPLES:

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

galois_conjugates
\texttt{(n=None)}
Return the Galois conjugates of 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:

```python
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)
```

`imag()`

Return the imaginary part of this element.

EXAMPLES:

```python
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:

```python
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)
```

(continues on next page)
inverse()  

is_integral()  
Return whether self is an algebraic integer. 

This just wraps IsIntegralCyclotomic from GAP.  

See also:  

denominator()  

EXAMPLES:  

```python  
sage: E(6).is_integral()  
True  
sage: (E(4)/2).is_integral()  
False  
```

is_rational()  
Test whether this element is a rational number.  

EXAMPLES:  

```python  
sage: E(3).is_rational()  
False  
sage: (E(3) + E(3,2)).is_rational()  
True  
```

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  
```

minpoly(var='x')  
The minimal polynomial of self element over Q.  

INPUT:  

• var – (optional, default ‘x’) the name of the variable to use.  

EXAMPLES:  

```python  
sage: UCF.<E> = UniversalCyclotomicField()  
sage: UCF(4).minpoly()  
x - 4  
```
Todo: Polynomials with libgap currently does not implement a `.sage()` method (see trac ticket #18266). It would be faster/safer to not use string to construct the polynomial.

**multiplicative_order()**
Return the multiplicative order.

**EXAMPLES:**

```python
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
```

**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
```

5.2. Universal cyclotomic field
real_part()
Return the real part of this element.

EXAMPLES:

```
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
```

sqrt()
Return a square root of self as an algebraic number.

EXAMPLES:

```
sage: f = E(33)
sage: f.sqrt()
0.9954719225730846? + 0.0950560433041827?*I
sage: f.sqrt()**2 == f
True
```

to_cyclotomic_field(R=None)
Return this element as an element of a cyclotomic field.

EXAMPLES:

```
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
```

Matrices are correctly dealt with:

```
[ E(3) E(4)]
[ E(5) -E(3)^2]
sage: Matrix(CyclotomicField(60),M)  # indirect doctest
```
Using a non-standard embedding:

```python
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
```

Test that the bug reported in trac ticket #19912 has been fixed:

```python
sage: a = 1+E(4); a
1 + E(4)
sage: 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:**

```python
sage: import sage.rings.universal_cyclotomic_field as ucf
sage: _ = UniversalCyclotomicField()  # indirect doctest
sage: ucf.libgap is None  # indirect doctest
False
```

See also:

`sage.rings.algebraic_closure_finite_field`
CHAPTER

SIX

INDICES AND TABLES

• Index
• Module Index
• Search Page


<table>
<thead>
<tr>
<th>Module Name</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sage.rings.number_field.bdd_height</td>
<td>180</td>
</tr>
<tr>
<td>sage.rings.number_field.class_group</td>
<td>259</td>
</tr>
<tr>
<td>sage.rings.number_field.galois_group</td>
<td>174</td>
</tr>
<tr>
<td>sage.rings.number_field.maps</td>
<td>193</td>
</tr>
<tr>
<td>sage.rings.number_field.morphism</td>
<td>185</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field</td>
<td>1</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field_base</td>
<td>98</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field_element</td>
<td>123</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field_element_quadratic</td>
<td>159</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field_ideal</td>
<td>222</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field_ideal_rel</td>
<td>249</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field_morphisms</td>
<td>189</td>
</tr>
<tr>
<td>sage.rings.number_field.number_field_rel</td>
<td>101</td>
</tr>
<tr>
<td>sage.rings.number_field.order</td>
<td>201</td>
</tr>
<tr>
<td>sage.rings.number_field.S_unit_solver</td>
<td>271</td>
</tr>
<tr>
<td>sage.rings.number_field.small_primes_of_degree_one</td>
<td>296</td>
</tr>
<tr>
<td>sage.rings.number_field.splitting_field</td>
<td>169</td>
</tr>
<tr>
<td>sage.rings.number_field.structure</td>
<td>198</td>
</tr>
<tr>
<td>sage.rings.number_field.totallyreal</td>
<td>299</td>
</tr>
<tr>
<td>sage.rings.number_field.totallyreal_data</td>
<td>306</td>
</tr>
<tr>
<td>sage.rings.number_field.totallyreal_phc</td>
<td>308</td>
</tr>
<tr>
<td>sage.rings.number_field.totallyreal_rel</td>
<td>302</td>
</tr>
<tr>
<td>sage.rings.number_field.unit_group</td>
<td>265</td>
</tr>
<tr>
<td>sage.rings.qqbar</td>
<td>311</td>
</tr>
<tr>
<td>sage.rings.universal_cyclotomic_field</td>
<td>369</td>
</tr>
</tbody>
</table>
## Symbols

_\_richcmp\_() (sage.rings.qqbar.AlgebraicNumber method), 340
_\_richcmp\_() (sage.rings.qqbar.AlgebraicReal method), 353

A

AA (in module sage.rings.qqbar), 319
abs() (sage.rings.number_field.number_field_element.NumberFieldElement method), 124
abs() (sage.rings.qqbar.ANDescr method), 321
abs() (sage.rings.qqbar.ANExtensionElement method), 322
abs() (sage.rings.qqbar.ANRational method), 327
abs() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 373
abs_hom() (sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs method), 187
abs_non_arch() (sage.rings.number_field.number_field_element.NumberFieldElement method), 125
abs_val() (sage.rings.number_field.number_field.NumberField_absolute method), 12
absolute_base_field() (sage.rings.number_field.number_field_RelativeNumberField method), 103
absolute_charpoly() (sage.rings.number_field.number_field_element.NumberFieldElement_absolute method), 151
absolute_charpoly() (sage.rings.number_field.number_field_element.NumberFieldElement_relative method), 154
absolute_charpoly() (sage.rings.number_field.number_field_element.OrderElement_relative method), 157
absolute_degree() (sage.rings.number_field.number_field.NumberField_absolute method), 12
absolute_degree() (sage.rings.number_field.number_field_NumberField_generic method), 41
absolute_degree() (sage.rings.number_field.number_field_RelativeNumberField method), 103
absolute_degree() (sage.rings.number_field.number_field_RelativeNumberField method), 206
absolute_different() (sage.rings.number_field.number_field.NumberField_absolute method), 12
absolute_different() (sage.rings.number_field.number_field_RelativeNumberField method), 103
absolute_discriminant() (sage.rings.number_field.number_field.NumberField_absolute method), 12
absolute_discriminant() (sage.rings.number_field.number_field_RelativeNumberField method), 103
absolute_discriminant() (sage.rings.number_field.number_field_RelativeNumberField method), 202
absolute_discriminant() (sage.rings.number_field.number_field_RelativeNumberField method), 216
absolute_field() (sage.rings.number_field.number_field_NumberField_generic method), 41
absolute_field() (sage.rings.number_field.number_field_NumberField_relative method), 104
absolute_generator() (sage.rings.number_field.number_field_NumberField_absolute method), 13
absolute_generator() (sage.rings.number_field.number_field_NumberField_relative method), 104
absolute_ideal() (sage.rings.number_field.number_field_ideal_NumberFieldFractionalIdeal method), 250
absolute_minpoly() (sage.rings.number_field.number_field_element.NumberFieldElement absolute method), 151
absolute_minpoly() (sage.rings.number_field.number_field_element.NumberFieldElement relative method), 154
absolute_minpoly() (sage.rings.number_field.number_field_element.OrderElement relative method), 157
absolute_norm() (sage.rings.number_field.number_field_element.NumberFieldElement method), 126
absolute_norm() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 237
absolute_norm() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 251
absolute_order() (sage.rings.number_field.order.AbsoluteOrder method), 202
absolute_order() (sage.rings.number_field.order.RelativeOrder method), 217
absolute_order_from_module_generators() (in module sage.rings.number_field.order), 218
absolute_order_from_ring_generators() (in module sage.rings.number_field.order), 220
absolute_polynomial() (sage.rings.number_field.number_field.NumberField_absolute method), 13
absolute_polynomial() (sage.rings.number_field.number_field_rel.NumberField_relative method), 105
absolute_polynomial_ntl() (sage.rings.number_field.number_field.NumberField_generic method), 41
absolute_polynomial_ntl() (sage.rings.number_field.number_field_rel.NumberField_relative method), 105
absolute_ramification_index() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 237
absolute_ramification_index() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 251
absolute_vector_space() (sage.rings.number_field.number_field.NumberField_absolute method), 13
absolute_vector_space() (sage.rings.number_field.number_field_rel.NumberField_relative method), 105
AbsoluteFromRelative (class in sage.rings.number_field.structure), 199
AbsoluteOrder (class in sage.rings.number_field.order), 201
additive_order() (sage.rings.number_field.number_field_element.NumberFieldElement method), 126
additive_order() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 374
algebraic_closure() (sage.rings.number_field.number_field.NumberField_generic method), 41
algebraic_closure() (sage.rings.number_field.number_field.NumberField_generic method), 374
algebraic_closure() (sage.rings.qqbar.AlgebraicField method), 333
algebraic_closure() (sage.rings.qqbar.AlgebraicComplexField method), 512
algebraic_closure() (sage.rings.qqbar.AlgebraicRealField method), 357
algebraic_closure() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 371
AlgebraicField (class in sage.rings.qqbar), 333
AlgebraicField_common (class in sage.rings.qqbar), 336
AlgebraicField_common.options() (in module sage.rings.qqbar), 337
AlgebraicGenerator (class in sage.rings.qqbar), 337
AlgebraicGenerator_Relation (class in sage.rings.qqbar), 340
AlgebraicNumber (class in sage.rings.qqbar), 340
AlgebraicNumber_base (class in sage.rings.qqbar), 345
AlgebraicNumber_PowQQAction (class in sage.rings.qqbar), 343
AlgebraicPolynomialTracker (class in sage.rings.qqbar), 351
AlgebraicReal (class in sage.rings.qqbar), 353
AlgebraicRealField (class in sage.rings.qqbar), 357
alpha() (sage.rings.number_field.number_field_element.CoordinateFunction method), 124
ambient() (sage.rings.number_field.number_field.order.Order method), 206
ambient_field (sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldConversion attribute), 190
ambient_field (sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldMorphism attribute), 190
an_binop_element() (in module sage.rings.qqbar), 360
an_binop_expr() (in module sage.rings.qqbar), 360
an_binop_rational() (in module sage.rings.qqbar), 361
an_element() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 371
ANBinaryExpr (class in sage.rings.qqbar), 319
ANDescr (class in sage.rings.qqbar), 321
ANEExtensionElement (class in sage.rings.qqbar), 322
angle() (sage.rings.qqbar.ANRational method), 327
ANRational (class in sage.rings.qqbar), 327
ANRoot (class in sage.rings.qqbar), 329
ANUnaryExpr (class in sage.rings.qqbar), 331
artin_symbol() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 177
artin_symbol() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 237
as_hom() (sage.rings.number_field.galois_group.GaloisGroupElement method), 175
as_number_field_element() (sage.rings.qqbar.AlgebraicNumber_base method), 346
automorphisms() (sage.rings.number_field.number_field.NumberField_absolute method), 13
automorphisms() (sage.rings.number_field.number_field_rel.NumberField_relative method), 106

B

bach_bound() (sage.rings.number_field.number_field_base.NumberField method), 98
base_field() (sage.rings.number_field.number_field.AbsoluteFieldAbsolute method), 14
base_field() (sage.rings.number_field.number_field_RelNumberField_relative method), 107
base_ring() (sage.rings.number_field.number_field_RelNumberField_relative method), 107
basis() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 237
basis() (sage.rings.number_field.number_field_order.AbsoluteOrder method), 202
basis() (sage.rings.number_field.number_field_order.Order method), 207
basis() (sage.rings.number_field.number_field_order.RelativeOrder method), 217
basis_to_module() (in module sage.rings.number_field.number_field_ideal), 248
bdd_height() (in module sage.rings.number_field.bdd_height), 180
bdd_height_iq() (in module sage.rings.number_field.bdd_height), 181
bdd_norm_pr_gens_iq() (in module sage.rings.number_field.bdd_height), 182
bdd_norm_pr_ideal_gens() (in module sage.rings.number_field.bdd_height), 183
beta_k() (in module sage.rings.number_field.S_unit_solver), 273

C
c11_func() (in module sage.rings.number_field.S_unit_solver), 273
c13_func() (in module sage.rings.number_field.S_unit_solver), 274
c3_func() (in module sage.rings.number_field.S_unit_solver), 275
c4_func() (in module sage.rings.number_field.S_unit_solver), 275
c8_c9_func() (in module sage.rings.number_field.S_unit_solver), 276
cardinality() (sage.rings.number_field.morphism.NumberFieldHomset method), 186
ceil() (sage.rings.number_field.number_field_element.NumberFieldElement method), 126
ceil() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 159
ceil() (sage.rings.qqbar.AlgebraicReal method), 353
change_generator() (sage.rings.number_field.number_field.GenericNumberField method), 42
change_names() (sage.rings.number_field.number_field.GenericNumberField method), 14
change_names() (sage.rings.number_field.number_field_RelNumberField_relative method), 107
change_names() (sage.rings.number_field.number_field_order.AbsoluteOrder method), 203
characteristic() (sage.rings.number_field.number_field.GenericNumberField method), 42
characteristic() (sage.rings.number_field.number_field_RelNumberField_relative method), 336
characteristic() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 371
det() (sage.rings.number_field.number_field_element.NumberFieldElement method), 127
det() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 152
det() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 155
det() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 157
det() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 160
det() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 167
class_group() (sage.rings.number_field.number_field.GenericNumberField method), 42
class_group() (sage.rings.number_field.number_field.Order method), 207
class_number() (sage.rings.number_field.generic_number_field.GenericNumberField method), 43
class_number() (sage.rings.number_field.number_field.GenericNumberField method), 89
class_number() (sage.rings.number_field.number_field.Order method), 207

Index 389
ClassGroup (class in sage.rings.number_field.class_group), 260
clean_rfv_dict() (in module sage.rings.number_field.S_unit_solver), 276
clean_sfs() (in module sage.rings.number_field.S_unit_solver), 277
clear_denominators() (in module sage.rings.qqbar), 361
closest() (in module sage.rings.number_field.number_field_morphisms), 191
cmp_elements_with_same_minpoly() (in module sage.rings.qqbar), 361
coefficients_to_power_sums() (in module sage.rings.qqbar), 361
column_Log() (in module sage.rings.number_field.S_unit_solver), 277
common_polynomial() (sage.rings.qqbar.AlgebraicField_common method), 336
compatible_system_lift() (in module sage.rings.number_field.S_unit_solver), 278
compatible_systems() (in module sage.rings.number_field.S_unit_solver), 278
compatible_vectors() (in module sage.rings.number_field.S_unit_solver), 279
compatible_vectors_check() (in module sage.rings.number_field.S_unit_solver), 280
completely_split_primes() (sage.rings.number_field.number_field.NumberField_generic method), 44
completion() (sage.rings.number_field.number_field_NumberField_generic method), 44
completion() (sage.rings.qqbar.AlgebraicField method), 333
completion() (sage.rings.qqbar.AlgebraicRealField method), 357
complex_conjugation() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 177
complex_conjugation() (sage.rings.number_field.number_field.NumberField_generic method), 44
complex_embedding() (sage.rings.number_field.number_field_number_field_cyclotomic method), 32
complex_embedding() (sage.rings.number_field.number_field_number_field_element.NumberFieldElement method), 127
complex_embeddings() (sage.rings.number_field.number_field_number_field_cyclotomic method), 32
complex_embeddings() (sage.rings.number_field.number_field_number_field generic method), 45
complex_embeddings() (sage.rings.number_field.number_field_number_field_element.NumberFieldElement method), 127
complex_exact() (sage.rings.qqbar.AlgebraicNumber method), 341
complex_number() (sage.rings.qqbar.AlgebraicNumber method), 341
complex_roots() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 352
composite_fields() (sage.rings.number_field.number_field.NumberField_generic method), 45
composite_fields() (sage.rings.number_field.number_field_number_field_rel.NumberField_relative method), 108
conductor() (sage.rings.number_field.number_field.NumberField_generic method), 47
conductor() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField_element method), 374
conjugate() (sage.rings.number_field.number_field_element.NumberFieldElement method), 128
conjugate() (sage.rings.qqbar.AlgebraicGenerator method), 338
conjugate() (sage.rings.qqbar.AlgebraicNumber method), 342
conjugate() (sage.rings.qqbar.AlgebraicReal method), 353
conjugate() (sage.rings.qqbar.ANDescr method), 321
conjugate() (sage.rings.qqbar.ANExtensionElement method), 323
conjugate() (sage.rings.qqbar.ANRoot method), 329
conjugate() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField_element method), 374
conjugate() (sage.rings.qqbar), 362
conjugate() (sage.rings.qqbar), 362
continue_comp_exp_vec() (in module sage.rings.number_field.S_unit_solver), 280
continue_complement_dictionaries() (in module sage.rings.number_field.S_unit_solver), 281
continue_rfv_to_ev() (in module sage.rings.number_field.S_unit_solver), 282
construction() (sage.rings.number_field.number_field_number_field_cyclotomic method), 33
construction() (sage.rings.number_field.number_field_number_field_generic method), 48
construction() (sage.rings.qqbar.AlgebraicField method), 334
continued_fraction() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 160
continued_fraction_list() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 160
CoordinateFunction (class in `sage.rings.number_field.number_field_element`), 123
coordinates() (`sage.rings.number_field.number_field_ideal.NumberFieldIdeal` method), 238
coordinates() (`sage.rings.number_field.order.Order` method), 207
coordinates_in_terms_of_powers() (`sage.rings.number_field.number_field_element.NumberFieldElement` method), 128
create_embedding_from_approx() (in module `sage.rings.number_field.number_field_morphisms`), 191
create_key() (`sage.rings.number_field.number_field.CyclotomicFieldFactory` method), 4
create_key_and_extra_args() (`sage.rings.number_field.number_field.NumberFieldFactory` method), 9
create_object() (`sage.rings.number_field.number_field.CyclotomicFieldFactory` method), 4
create_object() (`sage.rings.number_field.number_field.NumberFieldFactory` method), 9
create_structure() (`sage.rings.number_field.structure.AbsoluteFromRelative` method), 199
create_structure() (`sage.rings.number_field.structure.NameChange` method), 199
create_structure() (`sage.rings.number_field.structure.NumberFieldStructure` method), 199
create_structure() (`sage.rings.number_field.structure.RelativeFromAbsolute` method), 199
create_structure() (`sage.rings.number_field.structure.RelativeFromRelative` method), 200

cx_LLL_bound() (in module `sage.rings.number_field.S_unit_solver`), 283
CyclotomicFieldEmbedding (class in `sage.rings.number_field.number_field_morphisms`), 189
CyclotomicFieldFactory (class in `sage.rings.number_field.number_field`), 2
CyclotomicFieldHomomorphism_im_gens (class in `sage.rings.number_field.morphism`), 185
CyclotomicFieldHomset (class in `sage.rings.number_field.morphism`), 185

decomposition_group() (`sage.rings.number_field.galois_group.GaloisGroup_v2` method), 177
decomposition_group() (`sage.rings.number_field.number_field_ideal.NumberFieldIdeal` method), 238
default_base_hom() (`sage.rings.number_field.morphism.RelativeNumberFieldHomset` method), 188
default_interval_prec() (`sage.rings.qqbar.AlgebraicField_common` method), 337
defining_polynomial() (`sage.rings.number_field.number_field.NumberField_generic` method), 48
defining_polynomial() (`sage.rings.number_field.number_field_rel.NumberField_relative` method), 109
defining_polynomial_for_Kp() (in module `sage.rings.number_field.S_unit_solver`), 283
degree() (`sage.rings.number_field.number_field_number_field-generic` method), 48
degree() (`sage.rings.number_field.number_field_number_field-base.NumberField_method), 99
degree() (`sage.rings.number_field.number_field_number_field_rel.NumberField-relative-method), 109
degree() (`sage.rings.number_field.order.Order` method), 208
degree() (`sage.rings.qqbar.AlgebraicNumber_base` method), 346
degree() (`sage.rings.universal_cyclotomic_field.UniversalCyclotomicField` method), 372
denominator() (`sage.rings.number_field.number_field_element.NumberFieldElement` method), 129
denominator() (`sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic` method), 160
denominator() (`sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal` method), 223
denominator() (`sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement` method), 374
denominator_ideal() (`sage.rings.number_field.number_field_element.NumberFieldElement` method), 129
descend_mod_power() (`sage.rings.number_field.number_field_element.NumberFieldElement` method), 130
difference() (`sage.rings.number_field.number_field_number_field-generic` method), 49
difference() (`sage.rings.number_field.number_field_number_field-cyclotomic` method), 33
difference() (`sage.rings.number_field.number_field_number_field-generic` method), 49
disc() (`sage.rings.number_field.number_field_number_field-generic` method), 49
disc() (`sage.rings.number_field.number_field_number_field-generic` method), 49
discriminant() (`sage.rings.number_field.number_field_number_field-cyclotomic` method), 33
discriminant() (`sage.rings.number_field.number_field_number_field-generic` method), 49

Index 391
discriminant() (sage.rings.number_field.number_field.NumberField_quadratic method), 90
discriminant() (sage.rings.number_field.number_field_base.NumberField method), 99
discriminant() (sage.rings.number_field.number_field_rel.NumberField_relative method), 110
discriminant() (sage.rings.number_field.order.AbsoluteOrder method), 203
divides() (sage.rings.number_field.number_field_fractional_ideal.NumberFieldFractionalIdeal method), 223
do_polred() (in module sage.rings.qqbar), 363
drop_vector() (in module sage.rings.number_field.S_unit_solver), 284

e
E() (in module sage.rings.universal_cyclotomic_field), 370
each_is_integral() (in module sage.rings.number_field.order), 221
easy_is_irreducible_py() (in module sage.rings.number_field.totallyreal_data), 306
EisensteinIntegers() (in module sage.rings.number_field.order), 204
Element (sage.rings.number_field.class_group.ClassGroup attribute), 261
Element (sage.rings.number_field.class_group.SClassGroup attribute), 264
Element (sage.rings.number_field.galois_group.GaloisGroup_v2 attribute), 177
Element (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField attribute), 371
element_1_mod() (sage.rings.number_field.number_field_fractional_ideal.NumberFieldFractionalIdeal method), 223
element_1_mod() (sage.rings.number_field.number_field_fractions.NumberFieldFractionalIdeal ideal method), 252
elements_of_bounded_height() (sage.rings.number_field.number_field.NumberField_absolute method), 15
elements_of_norm() (sage.rings.number_field.number_field_number_field.Generic method), 50
EmbeddedNumberFieldConversion (class in sage.rings.number_field.number_field_number_field.morphisms), 189
EmbeddedNumberFieldMorphism (class in sage.rings.number_field.number_field_number_field.morphisms), 190
embedding_to_Kp() (in module sage.rings.number_field.S_unit_solver), 285
embeddings() (sage.rings.number_field.number_field.NumberField_absolute method), 16
embeddings() (sage.rings.number_field.number_field_number_field_rel.NumberField_relative method), 110
enumerate_totallyreal_fields_all() (in module sage.rings.number_field.totallyreal_rel), 303
enumerate_totallyreal_fields_prim() (in module sage.rings.number_field.number_field), 300
enumerate_totallyreal_fields_rel() (in module sage.rings.number_field.totallyreal_rel), 304
eq_up_to_order() (in module sage.rings.number_field.S_unit_solver), 285
EquationOrder() (in module sage.rings.number_field.order), 205
euler_phi() (sage.rings.number_field.number_field_number_field_fractions.NumberFieldFractionalIdeal method), 224
exactify() (sage.rings.qqbar.AlgebraicNumber_base method), 347
exactify() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 352
exactify() (sage.rings.qqbar.ANBinaryExpr method), 319
exactify() (sage.rings.qqbar.ANExtensionElement method), 323
exactify() (sage.rings.qqbar.ANRational method), 327
exactify() (sage.rings.qqbar.ANRoot method), 329
exactify() (sage.rings.qqbar.ANUnaryExpr method), 331
expt() (sage.rings.number_field.unit_group.UnitGroup method), 268
extension() (sage.rings.number_field.number_field.NumberField_generic method), 50

F
factor() (sage.rings.number_field.number_field_number_field_generic_number_field_Generic method), 51
factor() (sage.rings.number_field.number_field_element.NumberFieldElement method), 130
factor() (sage.rings.number_field.number_field_fractions.NumberFieldFractionalIdeal method), 224
factor() (sage.rings.number_field.number_field_fractions.NumberFieldFractionalIdeal method), 252
factors() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 352
field() (sage.rings.qqbar.AlgebraicGenerator method), 338
field_element_value() (sage.rings.qqbar.ANExtensionElement method), 323
find_zero_result() (in module sage.rings.qqbar), 363
fixed_field() (sage.rings.number_field.galois_group.GaloisGroup_subgroup method), 175
floor() (sage.rings.number_field.number_field_element.NumberFieldElement method), 131
floor() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 161
floor() (sage.rings.qqbar.AlgebraicReal method), 353
fraction_field() (sage.rings.number_field.order.Order method), 208
fractional_ideal() (sage.rings.number_field_number_field.Generic method), 52
FractionalIdealClass (class in sage.rings.number_field.class_group), 261
free_module() (sage.rings.number_field.ideal.ideal.NumberFieldIdeal method), 239
free_module() (sage.rings.number_field.ideal.ideal_NumberFieldFractionalIdeal_Rel method), 253
fractional_ideal() (sage.rings.number_field.order.Order method), 209
fundamental_units() (sage.rings.number_field.unit_group.UnitGroup method), 268

g

galois_closure() (sage.rings.number_field.number_field.NumberField_absolute method), 17
galois_closure() (sage.rings.number_field.number_field_rel.NumberField_relative method), 111
galois_conjugate() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 161
galois_conjugates() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 374
galois_group() (sage.rings.number_field.number_field.Generic method), 52
GaloisGroup (in module sage.rings.number_field.galois_group), 174
GaloisGroup_subgroup (class in sage.rings.number_field.galois_group), 175
GaloisGroup_v1 (class in sage.rings.number_field.galois_group), 175
GaloisGroup_v2 (class in sage.rings.number_field.galois_group), 176
GaloisGroupElement (class in sage.rings.number_field.galois_group), 174
GaussianIntegers() (in module sage.rings.order.Order), 205
gcd() (sage.rings.number_field.number_field_element.NumberFieldElement method), 132
gen() (sage.rings.number_field.number_field_Generic method), 54
gen() (sage.rings.number_field.number_field_Relative method), 111
gen() (sage.rings.number_field.order.Order method), 209
gen() (sage.rings.qqbar.AlgebraicField method), 334
gen() (sage.rings.qqbar.AlgebraicRealField method), 358
gen() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 372
gen_embedding() (sage.rings.number_field.number_field.Generic method), 54
gen_image() (sage.rings.number_field_number_field_morphisms.NumberFieldEmbedding method), 191
generator() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 352
generator() (sage.rings.qqbar.ANExtensionElement method), 323
generator() (sage.rings.qqbar.ANRational method), 327
gens() (sage.rings.number_field.class_group.FractionalIdealClass method), 262
gens() (sage.rings.number_field.number_field_rel.NumberField_relative method), 112
gens() (sage.rings.qqbar.AlgebraicField method), 334
gens() (sage.rings.qqbar.AlgebraicRealField method), 358
gens_ideals() (sage.rings.number_field.class_group.ClassGroup method), 261
gens_reduced() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 240
gens_reduced() (sage.rings.number_field.number_field_ideal_Relative method), 253
gens_two() (sage.rings.number_field.number_field_ideal_NumberFieldIdeal method), 240
get_AA_golden_ratio() (in module sage.rings.qqbar), 364
global_height() (sage.rings.number_field.number_field_element.NumberFieldElement method), 133
global_height_arch() (sage.rings.number_field.number_field_element.NumberFieldElement method), 134
global_height_non_arch() (sage.rings.number_field.number_field_element.NumberFieldElement method), 134
group() (sage.rings.number_field.galois_group.GaloisGroup_v1 method), 176

H
handle_sage_input() (sage.rings.qqbar.ANBinaryExpr method), 319
handle_sage_input() (sage.rings.qqbar.ANExtensionElement method), 324
handle_sage_input() (sage.rings.qqbar.ANRational method), 327
handle_sage_input() (sage.rings.qqbar.ANRoot method), 330
handle_sage_input() (sage.rings.qqbar.ANUnaryExpr method), 332
hermite_constant() (in module sage.rings.number_field.totallyreal_data), 306
hilbert_class_field() (sage.rings.number_field.number_field.NumberField_quadratic method), 90
hilbert_class_field_defining_polynomial() (sage.rings.number_field.number_field.NumberField_quadratic method), 90
hilbert_class_polynomial() (sage.rings.number_field.number_field.NumberField_quadratic method), 91
hilbert_conductor() (sage.rings.number_field.number_field.NumberField_absolute method), 18
hilbert_symbol() (sage.rings.number_field.number_field.NumberField_absolute method), 19

I
ideal() (sage.rings.number_field.class_group.FractionalIdealClass method), 262
ideal() (sage.rings.number_field.number_field.NumberField_generic method), 54
ideal() (sage.rings.number_field.order.Order method), 210
ideal_below() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 253
ideal_class_log() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 241
idealcoprime() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 225
ideallog() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 225
ideals_of_bdd_norm() (sage.rings.number_field.number_field.NumberField_generic method), 54
idealstar() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 226
im_gens() (sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs method), 188
imag() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 161
imag() (sage.rings.qqbar.AlgebraicNumber method), 342
imag() (sage.rings.qqbar.AlgebraicReal method), 354
imag() (sage.rings.qqbar.ANDescr method), 321
imag() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 375
imag_part() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 375
incr() (sage.rings.number_field.totallyreal_rel.tr_data_rel method), 306
increment() (sage.rings.number_field.totallyreal_data_tr_data method), 308
index_in() (sage.rings.number_field.order.AbsoluteOrder method), 203
index_in() (sage.rings.number_field.order.RelativeOrder method), 218
inertia_group() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 178
inertia_group() (sage.rings.number_field_number_field.Element method), 241
int_has_small_square_divisor() (in module sage.rings.number_field.totallyreal_data), 307
integer_points_in_polytope() (in module sage.rings.number_field.bdd_height), 184
integral_basis() (sage.rings.number_field.number_field.NumberField_generic method), 55
integral_basis() (sage.rings.number_field.number_field_number_field.Element method), 241
integral_basis() (sage.rings.number_field.number_field_number_field_ideal.NumberFieldIdeal method), 241
integral_closure() (sage.rings.number_field.order.Order method), 210
integral_elements_in_box() (in module sage.rings.number_field.totallyreal_rel), 305
integral_split() (sage.rings.number_field.number_field_number_field.Element method), 242
integral_split() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 255
intersection() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 242
intersection() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal_rel method), 204
intersection() (sage.rings.qqbar.AlgebraicNumber_base method), 347
interval_diameter() (sage.rings.qqbar.AlgebraicNumber_base method), 347
interval_exact() (sage.rings.qqbar.AlgebraicNumber method), 342
interval_exact() (sage.rings.qqbar.AlgebraicReal method), 354
interval_fast() (sage.rings.qqbar.AlgebraicNumber_base method), 348
inverse() (sage.rings.number_field.class_group.FractionalIdealClass method), 262
inverse() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 376
inverse_mod() (sage.rings.number_field.number_field_element.NumberFieldElement method), 135
inverse_mod() (sage.rings.number_field.number_field_element.OrderElement_absolute method), 156
inverse_mod() (sage.rings.number_field.number_field_element.OrderElement_relative method), 158
inverse_mod() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 167
invert() (sage.rings.qqbar.ANDescr method), 321
invert() (sage.rings.qqbar.ANExtensionElement method), 324
invert() (sage.rings.qqbar.ANRational method), 328
invertible_residues() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 227
invertible_residues_mod() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 227
is_abelian() (sage.rings.number_field.number_field.NumberField_generic method), 57
is_abs() (sage.rings.number_field.number_field.NumberField_absolute method), 22
is_abs() (sage.rings.number_field.number_field.NumberField_absolutmethod, 22
is_abs() (sage.rings.number_field.number_field.NumberField_relative method), 112
is_AbsoluteNumberField() (in module sage.rings.number_field.number_field), 93
is_AlgebraicField() (in module sage.rings.qqbar), 364
is_AlgebraicField_common() (in module sage.rings.qqbar), 364
is_AlgebraicNumber() (in module sage.rings.qqbar), 364
is_AlgebraicReal() (in module sage.rings.qqbar), 364
is_AlgebraicRealField() (in module sage.rings.qqbar), 365
is_CM() (sage.rings.number_field.number_field.NumberField_generic method), 56
is_CM_extension() (sage.rings.number_field.number_field_rel.NumberField_relative method), 112
is_complex() (sage.rings.qqbar.AlgebraicGenerator method), 338
is_complex() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 352
is_complex() (sage.rings.qqbar.ANBinaryExpr method), 321
is_complex() (sage.rings.qqbar.ANExtensionElement method), 325
is_complex() (sage.rings.qqbar.ANRational method), 328
is_complex() (sage.rings.qqbar.ANRoot method), 331
is_complex() (sage.rings.qqbar.ANUUnaryExpr method), 333
is_coprime() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 229
is_CyclotomicField() (in module sage.rings.number_field.number_field), 93
is_exact() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 372
is_field() (sage.rings.number_field.number_field.NumberField_generic method), 57
is_field() (sage.rings.number_field.number_field.Order.Order method), 210
is_free() (sage.rings.number_field.number_field_rel.NumberField_relative method), 112
is_fundamental_discriminant() (in module sage.rings.number_field.number_field), 94
is_galois() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 178
is_galois() (sage.rings.number_field.number_field.NumberField_order_cyclotomic method), 33
is_galois() (sage.rings.number_field.number_field.NumberField_order_cyclomethod, 57
is_galois() (sage.rings.number_field.number_field.NumberField_quadratic method), 91
is_galois() (sage.rings.number_field.number_field_rel.NumberField_relative method), 113
is_galois_absolute() (sage.rings.number_field.number_field_rel.NumberField_relative method), 113
is_galois_relative() (sage.rings.number_field.number_field_rel.NumberField_relative method), 113
is_injective() (sage.rings.number_field.maps.NumberFieldIsomorphism method), 198
is_integer() (sage.rings.number_field.number_field_element.NumberFieldElement method), 135
is_integer() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 135
is_integer() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 162
is_integer() (sage.rings.qqbar.AlgebraicNumber_base method), 348
is_integral() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 135
is_integral() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 162
is_integral() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 242
is_integral() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 255
is_integral() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 376
is_integrally_closed() (sage.rings.number_field.order.Order method), 211
is_isomorphic() (sage.rings.number_field.number_field_cyclotomic.NumberField_cyclotomic method), 34
is_isomorphic() (sage.rings.number_field.number_field_generic.NumberField_generic method), 58
is_isomorphic_relative() (sage.rings.number_field.number_field_RelativeNumberField.RelativeNumberField method), 113
is_maximal() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 230
is_maximal() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 243
is_maximal() (sage.rings.number_field.order.Order method), 211
is_noetherian() (sage.rings.number_field.order.Order method), 211
is_norm() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 135
is_nth_power() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 135
is_NumberField() (in module sage.rings.number_field.number_field_base), 101
is_NumberFieldElement() (in module sage.rings.number_field.number_field_elements), 159
is_NumberFieldFractionalIdeal() (in module sage.rings.number_field.number_field_ideal), 248
is_NumberFieldFractionalIdeal_rel() (in module sage.rings.number_field.number_field_ideal_rel), 259
is_NumberFieldHomsetCodomain() (in module sage.rings.number_field.number_field_homset), 94
is_NumberFieldIdeal() (in module sage.rings.number_field.number_field_ideal), 248
is_NumberFieldIdeal() (in module sage.rings.number_field.order.Order method), 221
is_one() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 135
is_one() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 162
is_padic_square() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 135
is_prime() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 243
is_prime() (sage.rings.number_field.number_field_element_number_field_element.Element_number_field_element method), 243
is_principal() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 243
is_principal() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 255
is_principal() (sage.rings.number_field.number_field_ideal_class_group.NumberFieldIdeal_class_group method), 262
is_real() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 376
is_real() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 376
is_real_positive() (sage.rings.number_field.number_field_element_number_field.Element_number_field method), 152
is_relative() (sage.rings.number_field_number_field.Element_number_field method), 152
is_RelativeNumberField() (in module sage.rings.number_field.number_field_RelativeNumberField.RelativeNumberField method), 113
is_S_integral() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 228
is_S_unit() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 229
is_simple() (sage.rings.qqbar.ANDescr method), 321
is_simple() (sage.rings.qqbar.ANExtensionElement method), 325
is_simple() (sage.rings.qqbar.ANRational method), 328
is_square() (sage.rings.number_field.number_field_element.NumberFieldElement method), 139
is_square() (sage.rings.qqbar.AlgebraicNumber_base method), 348
is_suborder() (sage.rings.number_field.order.Order method), 212
is_suborder() (sage.rings.number_field.order.RelativeOrder method), 218
is_surjective() (sage.rings.number_field.maps.NumberFieldIsomorphism method), 198
is_totally_imaginary() (sage.rings.number_field.number_field:NumberField_generic method), 58
is_totally_positive() (sage.rings.number_field.number_field_element.NumberFieldElement method), 139
is_totally_real() (sage.rings.number_field.number_field.NumberField_generic method), 59
is_trivial() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 230
is_trivial() (sage.rings.qqbar.AlgebraicGenerator method), 338
is_unit() (sage.rings.number_field.number_field_element.NumberFieldElement method), 139
is_zero() (sage.rings.number_field.number_field_element.NumberFieldElement method), 243
is_zero() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 256
isolating_interval() (in module sage.rings.qqbar), 365
K
K0_func() (in module sage.rings.number_field.S_unit_solver), 272
K1_func() (in module sage.rings.number_field.S_unit_solver), 272
key() (sage.rings.number_field.splitting_field.SplittingData method), 169
krull_dimension() (sage.rings.number_field.order.Order method), 212
L
lagrange_degree_3() (in module sage.rings.number_field.totallyreal_data), 307
late_import() (in module sage.rings.universal_cyclotomic_field), 379
latex_variable_name() (sage.rings.number_field.number_field.NumberField_generic method), 59
lift() (sage.rings.number_field.number_field_element.NumberFieldElement_absolute method), 153
lift() (sage.rings.number_field.number_field_element.NumberFieldElement_relative method), 155
lift_to_base() (sage.rings.number_field.number_field_rel.NumberField_relative method), 114
LiftMap (class in sage.rings.number_field.number_field_ideal), 222
list() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 178
list() (sage.rings.number_field.morphism.CyclotomicFieldHomset method), 185
list() (sage.rings.number_field.morphism.NumberFieldHomset method), 186
list() (sage.rings.number_field.morphism.RelativeNumberFieldHomset method), 189
list() (sage.rings.number_field.morphism.RelativeNumberFieldHomset method), 189
list() (sage.rings.number_field.morphism.RelativeNumberFieldHomset method), 140
list() (sage.rings.number_field.number_field_element.NumberFieldElement method), 153
list() (sage.rings.number_field.number_field_element.NumberFieldElement method), 153
local_height() (sage.rings.number_field.number_field_element.NumberFieldElement relative method), 140
local_height_arch() (sage.rings.number_field.number_field_element.NumberFieldElement method), 141
log() (sage.rings.number_field.unit_group.UnitGroup method), 269
log_p() (in module sage.rings.number_field.S_unit_solver), 286
log_p_series_part() (in module sage.rings.number_field.S_unit_solver), 286
M
MapAbsoluteToRelativeNumberField (class in sage.rings.number_field.maps), 193
MapNumberFieldToVectorSpace (class in sage.rings.number_field.maps), 193
MapRelativeNumberFieldToRelativeVectorSpace (class in sage.rings.number_field.maps), 194
MapRelativeNumberFieldToVectorSpace (class in sage.rings.number_field.maps), 194
Index 397
norm() (sage.rings.qqbar.ANDescr method), 322
norm() (sage.rings.qqbar.ANExtensionElement method), 326
norm_of_galois_extension() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 377
nth_root() (sage.rings.number_field.number_field_element.NumberFieldElement method), 144
nth_root() (sage.rings.qqbar.AlgebraicNumber_base method), 349
number_field() (sage.rings.number_field.class_group.ClassGroup method), 261
number_field() (sage.rings.number_field.galois_group.GaloisGroup_v1 method), 176
number_field() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 179
number_field() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 244
number_field() (sage.rings.number_field.order.Order method), 212
number_field() (sage.rings.number_field.unit_group.UnitGroup method), 269
number_field_elements_from_algebraics() (in module sage.rings.qqbar), 365
number_of_roots_of_unity() (sage.rings.number_field.number_field.NumberField_cyclotomic method), 34
number_of_roots_of_unity() (sage.rings.number_field.number_field.NumberField_generic method), 61
number_of_roots_of_unity() (sage.rings.number_field.number_field.NumberField_quadratic method), 91
number_of_roots_of_unity() (sage.rings.number_field.number_field_rel.NumberField_relative method), 116
NumberField (class in sage.rings.number_field.number_field_base), 98
NumberField() (in module sage.rings.number_field.number_field), 4
NumberField_absolute (class in sage.rings.number_field.number_field), 11
NumberField_absolute_v1() (in module sage.rings.number_field.number_field), 30
NumberField_cyclotomic (class in sage.rings.number_field.number_field), 31
NumberField_cyclotomic_v1() (in module sage.rings.number_field.number_field), 36
NumberField_extension_v1() (in module sage.rings.number_field.number_field_rel), 102
NumberField_generic (class in sage.rings.number_field.number_field), 37
NumberField_generic_v1() (in module sage.rings.number_field.number_field), 88
NumberField_quadratic (class in sage.rings.number_field.number_field), 89
NumberField_quadratic_v1() (in module sage.rings.number_field.number_field), 91
NumberField_relative (class in sage.rings.number_field.number_field_rel), 102
NumberField_relative_v1() (in module sage.rings.number_field.number_field_rel), 123
NumberFieldElement (class in sage.rings.number_field.number_field_element), 124
NumberFieldElement_absolute (class in sage.rings.number_field.number_field_element), 151
NumberFieldElement_quadratic (class in sage.rings.number_field.number_field_element_quadratic), 159
NumberFieldElement_relative (class in sage.rings.number_field.number_field_element), 154
NumberFieldEmbedding (class in sage.rings.number_field.number_field_morphisms), 190
NumberFieldFactory (class in sage.rings.number_field.number_field), 9
NumberFieldFractionalIdeal (class in sage.rings.number_field.number_field_ideal), 222
NumberFieldFractionalIdeal_rel (class in sage.rings.number_field.number_field_ideal_rel), 250
NumberFieldHomomorphism_im_gens (class in sage.rings.number_field.morphism), 185
NumberFieldHomset (class in sage.rings.number_field.morphism), 186
NumberFieldIdeal (class in sage.rings.number_field.number_field_ideal), 236
NumberFieldIsomorphism (class in sage.rings.number_field.morphism), 197
NumberFieldStructure (class in sage.rings.number_field.structure), 199
NumberFieldTower() (in module sage.rings.number_field.number_field), 9
numerator() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 164
numerator() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 230
numeratorIdeal() (sage.rings.number_field.number_field_element.NumberFieldElement method), 145

Index

odlyzko_bound_totallyreal() (in module sage.rings.number_field.totallyreal), 301
OK() (sage.rings.number_field.number_field_base.NumberField method), 98
one()(sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 372
optimized_representation()(sage.rings.number_field.number_field.NumberField_absolute method), 23
optimized_subfields()(sage.rings.number_field.number_field.NumberField_absolute method), 24
ord() (sage.rings.number_field.number_field_element.NumberFieldElement method), 145
Order (class in sage.rings.number_field.order), 206
order() (sage.rings.number_field.galois_group.GaloisGroup_v1 method), 176
order() (sage.rings.number_field.morphism.NumberFieldHomset method), 187
order() (sage.rings.number_field.number_field.NumberField_absolute method), 25
order() (sage.rings.number_field.number_field.NumberField_generic method), 61
order() (sage.rings.number_field.number_field_number_field_relative.NumberField_relative method), 116
order() (sage.rings.qqbar.AlgebraicField_common method), 337
OrderElement_absolute (class in sage.rings.number_field.number_field_element), 156
OrderElement_quadratic (class in sage.rings.number_field.number_field_element_quadratic), 167
OrderElement_relative (class in sage.rings.number_field.number_field_element), 157

P

p_adicLLL_bound() (in module sage.rings.number_field.S_unit_solver), 288
p_adicLLL_bound_one_prime() (in module sage.rings.number_field.S_unit_solver), 288
pari_absolute_base_polynomial() (sage.rings.number_field.number_field_rel.NumberField_relative method), 116
pari_bnf() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 61
pari_field() (sage.rings.qqbar.AlgebraicGenerator method), 338
pari_hnf() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 244
pari_nf() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 62
pari_polynomial() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 63
pari_prime() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 244
pari_relative_polynomial() (sage.rings.number_field.number_field_rel.NumberField_relative method), 117
pari_rhnf() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 256
pari_rnf() (sage.rings.number_field.number_field_rel.NumberField_relative method), 117
pari_rnfnorm_data() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 64
pari_zk() (sage.rings.number_field.number_field_element.NumberFieldElement method), 64
parts() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 164
place() (sage.rings.number_field.number_field_element.AbsoluteNumberField method), 26
place() (sage.rings.number_field.number_field_number_field_rel.AbsoluteNumberField method), 117
poldegree() (sage.rings.number_field.splitting_field.SplittingData method), 169
poly() (sage.rings.qqbar.AlgebraicPolynomialTracker method), 353
polynomial() (sage.rings.number_field.number_field_element.AbsoluteNumberField method), 64
polynomial() (sage.rings.number_field_number_field_element.ElementNumberField method), 146
polynomial() (sage.rings.number_field.number_field_element_number_field_RelativeNumberFieldRel method), 118
polynomial_nfl() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 65
polynomial_quotient_ring() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 65
polynomial_ring() (sage.rings.number_field.nf_field.nf_field.AbsoluteNumberField method), 65
polynomial_root() (sage.rings.qqbar.AlgebraicField method), 334
polynomial_root() (sage.rings.qqbar.AlgebraicRealField method), 358
possible_mus() (in module sage.rings.number_field.S_unit_solver), 289
power_basis() (sage.rings.number_field.number_field_element.AbsoluteNumberField method), 65
prec_seq() (in module sage.rings.qqbar), 368
preimage() (sage.rings.number_field.morphism.NumberFieldHomomorphism_im_gens method), 185
prime_above() (sage.rings.number_field.number_field_element.AbsoluteNumberField method), 66
prime_factors() (sage.rings.number_field.number_field_element.AbsoluteNumberField method), 67
prime_factors() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 231
prime_to_idealM_part() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 232
prime_to_S_part() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 231
primes() (sage.rings.number_field.unit_group.UnitGroup method), 270
primes_above() (sage.rings.number_field_number_field_NumberField_generic method), 67
primes_of_bounded_norm() (sage.rings.number_field_number_field_NumberField_generic method), 69
primes_of_bounded_norm_iter() (sage.rings.number_field_number_field_NumberField_generic method), 69
primes_of_degree_one_iter() (sage.rings.number_field_number_field_NumberField_generic method), 70
primes_of_degree_one_list() (sage.rings.number_field_number_field_NumberField_generic method), 70
primitive_element() (sage.rings.number_field_number_field_NumberField_generic method), 71
primitive_root_of_unity() (sage.rings.number_field_number_field_NumberField_generic method), 71
printa() (sage.rings.number_field_totallyreal_data_tr_data method), 308
proof_flag() (in module sage.rings.number_field_number_field), 95
put_natural_embedding_first() (in module sage.rings.number_field_number_field), 95
Q
Q_to_quadratic_field_element (class in sage.rings.number_field_number_field_element_quadratic), 168
QQbar (in module sage.rings.qqbar), 359
quadratic_defect() (sage.rings.number_field_number_field_NumberField_generic method), 72
QuadraticField() (in module sage.rings.number_field_number_field), 91
quotient_char_p() (in module sage.rings.number_field_number_field_number_field_ideal), 249
QuotientMap (class in sage.rings.number_field_number_field_ideal), 248
R
radical_expression() (sage.rings.qqbar.AlgebraicNumber_base method), 349
ramification_breaks() (sage.rings.number_field_galois_group.GaloisGroup_v2 method), 179
ramification_degree() (sage.rings.number_field_galois_group.GaloisGroupElement method), 175
ramification_group() (sage.rings.number_field_galois_group.GaloisGroup_v2 method), 179
ramification_index() (sage.rings.number_field_number_field_number_field_ideal_NumberFieldFractionalIdeal method), 232
ramification_index() (sage.rings.number_field_number_field_number_field_ideal_rel_NumberFieldFractionalIdeal_rel method), 256
random_element() (sage.rings.number_field_number_field_number_field_NumberField_generic method), 72
random_element() (sage.rings.number_field_number_field_number_field_NumberField_ideal method), 245
random_element() (sage.rings.number_field_number_field_order.Order method), 213
random_element() (sage.rings.qqbar.AlgebraicField method), 335
rank() (sage.rings.number_field_order.Order method), 214
rank() (sage.rings.number_field_unit_group.UnitGroup method), 270
rational_argument() (sage.rings.qqbar.AlgebraicNumber method), 343
rational_argument() (sage.rings.qqbar.ANExtensionElement method), 326
rational_argument() (sage.rings.qqbar.ANRational method), 329
rational_exact_root() (in module sage.rings.qqbar), 368
ray_class_number() (sage.rings.number_field_number_field_ideal_NumberFieldFractionalIdeal method), 232
real() (sage.rings.number_field_number_field_element_quadratic.NumberFieldElement_quadratic method), 165
real() (sage.rings.qqbar.AlgebraicNumber method), 343
real() (sage.rings.qqbar.AlgebraicReal method), 354
real() (sage.rings.qqbar.ANDescr method), 322
real() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method), 377
real_embeddings() (sage.rings.number_field_number_field_NumberField_cyclotomic method), 34
real_embeddings() (sage.rings.number_field_number_field_NumberField_generic method), 73
real_exact() (sage.rings.qqbar.AlgebraicReal method), 355
root_as_algebraic() (sage.rings.qqbar.AlgebraicGenerator method), 339
root_from_approx() (in module sage.rings.number_field.number_field_morphisms), 192
roots_of_unity() (sage.rings.number_field.number_field.NumberField_cyclotomic method), 35
roots_of_unity() (sage.rings.number_field.number_field.NumberField_generic method), 76
roots_of_unity() (sage.rings.number_field.unit_group.UnitGroup method), 270
round() (sage.rings.number_field.number_field_element.NumberFieldElement method), 147
round() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 165
S
S() (sage.rings.number_field.class_group.SClassGroup method), 264
S_class_group() (sage.rings.number_field.number_field.NumberField_generic method), 37
S_ideal_class_log() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 236
S_unit_group() (sage.rings.number_field.number_field.NumberField_generic method), 37
S_unit_solutions() (sage.rings.number_field.number_field.NumberField_generic method), 39
S_units() (sage.rings.number_field.number_field.NumberField_generic method), 40
sage.rings.number_field.bdd_height (module), 180
sage.rings.number_field.class_group (module), 259
sage.rings.number_field.galois_group (module), 174
sage.rings.number_field.maps (module), 193
sage.rings.number_field.morphism (module), 185
sage.rings.number_field.number_field (module), 1
sage.rings.number_field.number_field_base (module), 98
sage.rings.number_field.number_field_element (module), 123
sage.rings.number_field.number_field_element_quadratic (module), 159
sage.rings.number_field.number_field_ideal (module), 222
sage.rings.number_field.number_field_ideal_rel (module), 249
sage.rings.number_field.number_field_morphisms (module), 189
sage.rings.number_field.number_field_rel (module), 101
sage.rings.number_field.order (module), 201
sage.rings.number_field.S_unit_solver (module), 271
sage.rings.number_field.small_primes_of_degree_one (module), 296
sage.rings.number_field.splitting_field (module), 169
sage.rings.number_field.structure (module), 198
sage.rings.number_field.totallyreal (module), 299
sage.rings.number_field.totallyreal_data (module), 306
sage.rings.number_field.totallyreal_phc (module), 308
sage.rings.number_field.totallyreal_rel (module), 302
sage.rings.number_field.unit_group (module), 265
sage.rings.qqbar (module), 311
sage.rings.universal_cyclotomic_field (module), 369
scale() (sage.rings.qqbar.ANRational method), 329
SClassGroup (class in sage.rings.number_field.class_group), 263
section() (sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldMorphism method), 190
selmer_group() (sage.rings.number_field.number_field.NumberField_generic method), 76
selmer_group_iterator() (sage.rings.number_field.number_field.NumberField_generic method), 77
SFractionalIdealClass (class in sage.rings.number_field.class_group), 264
short_prec_seq() (in module sage.rings.qqbar), 368
sieve_below_bound() (in module sage.rings.number_field.S_unit_solver), 291
sieve_ordering() (in module sage.rings.number_field.S_unit_solver), 292
sign() (sage.rings.number_field.number_field_element.NumberFieldElement method), 148
sign() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 165
sign() (sage.rings.qqbar.AlgebraicReal method), 356
signature() (sage.rings.number_field.number_field_cyclotomic method), 35
signature() (sage.rings.number_field.number_field.NumberField_generic method), 78
signature() (sage.rings.number_field.number_field_base.NumberField method), 100
simplify() (sage.rings.qqbar.AlgebraicNumber_base method), 350
simplify() (sage.rings.qqbar.ANElement method), 326
Small_primes_of_degree_one_iter (class in sage.rings.number_field.small_primes_of_degree_one), 297
small_residue() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 236
smallest_integer() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 247
smallest_integer() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 258
solutions_from_systems() (in module sage.rings.number_field.S_unit_solver), 293
solve_CRT() (sage.rings.number_field.number_field.NumberField_generic method), 78
solve_S_unit_equation() (in module sage.rings.number_field.S_unit_solver), 294
some_elements() (sage.rings.number_field.number_field_number_field_number_field_order.Order method), 215
some_elements() (sage.rings.number_field_universal_cyclotomic.UniversalCyclotomicField method), 372
specification_cubed_embedding() (sage.rings.number_field.number_field.NumberField_generic method), 79
split_primes_large_lcm() (in module sage.rings.number_field.S_unit_solver), 295
splitting_field() (in module sage.rings.number_field.splitting_field), 170
splitting_field() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 179
SplittingData (class in sage.rings.number_field.splitting_field), 169
SplittingFieldAbort, 170
sqrt() (sage.rings.number_field.number_field_element.NumberFieldElement method), 149
sqrt() (sage.rings.qqbar.AlgebraicNumber_base method), 350
sqrt() (sage.rings.number_field_universal_cyclotomic_universal_cyclotomic.UniversalCyclotomicFieldElement method), 378
structure() (sage.rings.number_field.number_field.NumberField_generic method), 80
subfield() (sage.rings.number_field.number_field.NumberField Generic method), 81
subfield() (sage.rings.number_field.number_field.NumberField abs method), 29
subfields() (sage.rings.number_field.number_field_rel.NumberField relative method), 121
subgroup() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 179
super_poly() (sage.rings.qqbar.AlgebraicGenerator method), 339
support() (sage.rings.number_field.number_field_element.NumberFieldElement method), 149

T
t1 (in module sage.rings.qqbar), 368
t2 (in module sage.rings.qqbar), 368
tail_prec_seq() (in module sage.rings.qqbar), 368
to_cyclotomic_field() (sage.rings.number_field_universal_cyclotomic.UniversalCyclotomicFieldElement method), 378
torsion_generator() (sage.rings.number_field_unit_group.UnitGroup method), 270
tr_data (class in sage.rings.number_field.totallyreal_data), 307
tr_data_rel (class in sage.rings.number_field.totallyreal_rel), 305
trace() (sage.rings.number_field.number_field_element.NumberFieldElement method), 150
trace() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 166
trace() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 168
trace_dual_basis() (sage.rings.number_field.number_field.NumberField_generic method), 82
trace_pairing() (sage.rings.number_field_number_field_number_field.NumberField Generic method), 82
trunc() (sage.rings.qqbar.AlgebraicReal method), 357
U

UCFtoQQbar (class in sage.rings.universal_cyclotomic_field), 371
uniformizer() (sage.rings.number_field.number_field.NumberField_generic method), 82
uniformizer() (sage.rings.number_field.number_field_rel.NumberField_relative method), 122
union() (sage.rings.qqbar.AlgebraicGenerator method), 339
unit_group() (sage.rings.number_field.number_field.NumberField_generic method), 83
UnitGroup (class in sage.rings.number_field.unit_group), 268
units() (sage.rings.number_field.number_field.NumberField_generic method), 84
UniversalCyclotomicField (class in sage.rings.universal_cyclotomic_field), 371
UniversalCyclotomicFieldElement (class in sage.rings.universal_cyclotomic_field), 373
unrank() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 180

V

valuation() (sage.rings.number_field.number_field.NumberField_generic method), 85
valuation() (sage.rings.number_field.number_field_element.NumberFieldElement method), 150
valuation() (sage.rings.number_field.number_field_element.NumberFieldElement_relative method), 156
valuation() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal method), 247
valuation() (sage.rings.number_field.number_field_ideal_rel.NumberFieldFractionalIdeal_rel method), 258
valuation() (sage.rings.number_field.order.Order method), 215
vector() (sage.rings.number_field.number_field_element.NumberFieldElement method), 151
vector_space() (sage.rings.number_field.number_field.NumberField_absolute method), 30
vector_space() (sage.rings.number_field.number_field_rel.NumberField_relative method), 122

W

weed_fields() (in module sage.rings.number_field.totallyreal), 302

Z

Z_to_quadratic_field_element (class in sage.rings.number_field.number_field_element_quadratic), 169
zero() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 373
zeta() (sage.rings.number_field.number_field.NumberField_cyclotomic method), 35
zeta() (sage.rings.number_field.number_field.Element method), 86
zeta() (sage.rings.number_field.order.Order method), 216
zeta() (sage.rings.number_field.unit_group.UnitGroup method), 270
zeta() (sage.rings.qqbar.AlgebraicField method), 336
zeta() (sage.rings.qqbar.AlgebraicRealField method), 359
zeta() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 373
zeta_coefficients() (sage.rings.number_field.number_field.Element method), 87
zeta_function() (sage.rings.number_field.number_field.Element method), 88
zeta_order() (sage.rings.number_field.number_field.Element method), 36
zeta_order() (sage.rings.number_field.number_field.NumberField_cyclotomic method), 36
zeta_order() (sage.rings.number_field.number_field.NumberField_generic method), 88
zeta_order() (sage.rings.number_field.unit_group.UnitGroup method), 271