Algebraic Numbers and Number Fields

Release 9.6

The Sage Development Team

May 16, 2022
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Algebraic Number Fields</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Morphisms</td>
<td>209</td>
</tr>
<tr>
<td>3</td>
<td>Orders, Ideals, Ideal Classes</td>
<td>227</td>
</tr>
<tr>
<td>4</td>
<td>Totally Real Fields</td>
<td>339</td>
</tr>
<tr>
<td>5</td>
<td>Algebraic Numbers</td>
<td>351</td>
</tr>
<tr>
<td>6</td>
<td>Indices and Tables</td>
<td>429</td>
</tr>
<tr>
<td></td>
<td>Python Module Index</td>
<td>431</td>
</tr>
<tr>
<td></td>
<td>Index</td>
<td>433</td>
</tr>
</tbody>
</table>
1.1 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
- Simon King (2010-05): Improve coercion from GAP
- Jeroen Demeyer (2010-07, 2011-04): Upgrade PARI (trac ticket #9343, trac ticket #10430, trac ticket #11130)
- Robert Harron (2012-08): added is_CM(), complex_conjugation(), and maximal_totally_real_subfield()
- Christian Stump (2012-11): added conversion to universal cyclotomic field
- Julian Rueth (2014-04-03): absolute number fields are unique parents
- Vincent Delecroix (2015-02): comparisons/floor/ceil using embeddings
- Kiran Kedlaya (2016-05): relative number fields hash based on relative polynomials
- Peter Bruin (2016-06): make number fields fully satisfy unique representation
- John Jones (2017-07): improve check for is_galois(), add is_abelian(), building on work in patch by Chris Wuthrich
- Anna Haensch (2018-03): added quadratic_defect()
- Michael Daub, Chris Wuthrich (2020-09-01): adding Dirichlet characters for abelian 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: 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)
```
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
```

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

```sage
sage: k = CyclotomicField(7); k
Cyclotomic Field of order 7 and degree 6
sage: k.gen()
\zeta_7
```

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

```sage
sage: CC(k.gen())
0.623489801858734 + 0.781831482468030*I
```

Cyclotomic fields are of a special type.

```sage
sage: type(k)
<class 'sage.rings.number_field.number_field.NumberField_cyclotomic_with_category'>
```

We can specify a different generator name as follows.

```sage
sage: k.<z7> = CyclotomicField(7); k
Cyclotomic Field of order 7 and degree 6
sage: k.gen()
z7
```

The \( n \) must be an integer.

```sage
sage: CyclotomicField(3/2)
Traceback (most recent call last):
...
TypeError: no conversion of this rational to integer
```

The degree must be nonnegative.

```sage
sage: CyclotomicField(-1)
Traceback (most recent call last):
...
ValueError: n (-1) must be a positive integer
```

The special case \( n = 1 \) does not return the rational numbers:

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

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

```sage
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
```
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.

sage.rings.number_field.number_field.GaussianField()
The field \(\mathbb{Q}[i]\).

sage.rings.number_field.number_field.NumberField(polynomial, name, check=None, names=True, embedding=None, latex_name=None, assume_disc_small=None, maximize_at_primes=False, structure=None, latex_names=None, **kwds)
Return the number field (or tower of number fields) defined by the irreducible polynomial.

**INPUT:**

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

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

EXAMPLES:

```python
sage: z = QQ['z'].0
sage: K = NumberField(z^2 - 2, 's'); K
Number Field in s with defining polynomial z^2 - 2
sage: s = K.0; s
s
sage: s*s
2
sage: s^2
2
```

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*b
sage: L(a)
a
sage: L.lift_to_base(b^3 + b)
-a
```

Constructing another number field:

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

Number fields are globally unique:

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

(continues on next page)
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:

```
sage: K.<b> = NumberField(x^2 + 5, 'a'); K
```

```
Number Field in b with defining polynomial $x^2 + 5$
```

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 $x^3 - 2$ with a = 1.
    → 259921049894873?
  To:  Real Field with 53 bits of precision
    Defn:  Generic morphism:
      From: Number Field in a with defining polynomial $x^3 - 2$ with a = 1.
        → 259921049894873?
      To:  Real Lazy Field
        Defn:  a -> 1.259921049894873?
        then
          Conversion via _mpfr_ method map:
            From: Real Lazy Field
            To:  Real Field with 53 bits of precision
```

```
sage: RR(a)
1.259992104989487
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)
1.25992104989487316476721060672782283505702514647015079800820
```

One can embed into any other field:
The QuadraticField and CyclotomicField constructors create an embedding by default unless otherwise specified:

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

Comparison depends on the (real) embedding specified (or the one selected by default). Note that the codomain of the embedding must be QQbar or AA for this to work (see trac ticket #20184):

```python
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:

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

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

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

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

```
sage: pari(a)
Mod(-1/2*y^2 + 1/2*y, y^3 - y^2 - 2)
sage: K(pari(a))
a
sage: I = K.ideal(a); I
Fractional ideal (a)
sage: I.pari_hnf()
[1, 0, 0; 0, 1, 0; 0, 0, 1/2]
sage: K.ideal(I.pari_hnf())
Fractional ideal (a)
```

Here is an example where the field has non-trivial class group:

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

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

```
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
```

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:**

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

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

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

```sage
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:

```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:

```sage
k.<y> = QQ[]
```

```sage
m = NumberField([y^3 - 3, x^2 + x + 1, y^3 + 2], 'beta')
```

```sage
m.base_field()
Number Field in beta0 with defining polynomial y^3 - 3 over its base field
```

A tower of quadratic fields:

```sage
K.<a> = NumberField([x^2 + 3, x^2 + 2, x^2 + 1])
```

```sage
K.base_field()
Number Field in a0 with defining polynomial x^2 + 3 over its base field
```

```sage
K.base_field().base_field()
Number Field in a1 with defining polynomial x^2 + 2 over its base field
```

```sage
K.base_field().base_field().base_field()
Number Field in a2 with defining polynomial x^2 + 1
```

LaTeX versions of generator names can be specified either as:

```sage
K = NumberField([x^3 - 2, x^3 - 3, x^3 - 5], names=['a', 'b', 'c'], latex_names=[r'\alpha', r'\beta', r'\gamma'])
```

```sage
K.inject_variables(overwrite=True)
```

```sage
latex(a + b + c)
\alpha + \beta + \gamma
```

or as:
A bigger tower of quadratic fields:

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

The function can also be called by name:

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

### 1.1. Number Fields

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

The function can also be called by name:
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

Check that trac ticket #28345 is fixed:

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

**absolute_degree()**

A synonym for degree.

**EXAMPLES:**

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

**absolute_different()**

A synonym for different.

**EXAMPLES:**

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

**absolute_discriminant()**

A synonym for discriminant.

**EXAMPLES:**

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

**absolute_generator()**

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

**EXAMPLES:**
sage: K.<a> = NumberField(x^2 - 17)
sage: K.absolute_generator()
a
**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 + 1)
sage: K.absolute_polynomial ()
x^2 + 1
```

**absolute_vector_space(*args, **kwds)**

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

For an absolute extension this is identical to `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 \( K \):
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
```

**base_field()**

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

**EXAMPLES:**

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

**change_names(names)**

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

**INPUT:**

- names - should be exactly one variable name.

Also, K.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
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
```

(continues on next page)
To: Number Field in z with defining polynomial x^2 + 3
sage: L.structure()[0](ww + 5/3)
z + 5/3

**elements_of_bounded_height(**kwds**)**

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

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

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

**ALGORITHM:**

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

**INPUT:**

kwds:

• bound - a real number
• tolerance - (default: 0.01) a rational number in (0,1]
• 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)
sage: for t in L:
    ....: exp(6^t.global_height())
1.0
1.0
1.0
1.0
```

(continues on next page)
 AUTHORS:

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

**embeddings**(*K*)

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

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

INPUT:
• \( K \) - a field

EXAMPLES:

```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 with a = 2.
  To:   Cyclotomic Field of order 7 and degree 6
        Defn: a |--> 2*zeta7^4 + 2*zeta7^2 + 2*zeta7 + 1,
        Ring morphism:
  From: Number Field in a with defining polynomial x^2 + 7 with a = 2.
  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:
sage: K = NumberField(x^3 - 2, 'a')
sage: K.embeddings(GF(3))
[]

free_module(base=None, basis=None, map=True)
Return a vector space V and isomorphisms self –> V and V –> self.

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

OUTPUT:
• V - a vector space over the rational numbers
• from_V - an isomorphism from V to self (if requested)
• to_V - an isomorphism from self to V (if requested)

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

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:
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

(continues on next page)
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
sage: L, phi = K.galois_closure('b', map=True)
sage: L
Number Field in b with defining polynomial x^8 + 28*x^4 + 2500
sage: phi
Ring morphism:
  From: Number Field in a with defining polynomial x^4 - 2
  To:   Number Field in b with defining polynomial x^8 + 28*x^4 + 2500
  Defn: a |--> 1/240*b^5 - 41/120*b

A cyclotomic field is already Galois:

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 + x^17 + x^16 + x^15 + x^14 + x^13 + x^12 + x^11 + x^10 + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1

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

INPUT:
• a, b – elements of the number field self

OUTPUT:
• squarefree ideal of the ring of integers of self

EXAMPLES:

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

AUTHOR:
• Aly Deines
hilbert_symbol\( (a, b, P=None) \)
Return the Hilbert symbol \( (a, b)_P \) for a prime \( P \) of self and non-zero elements \( a \) and \( b \) of self.

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

**INPUT:**

- \( a, b \) – elements of self
- \( P \) – (default: None) If \( P \) 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:

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

That the latter two are unsolvable should be visible in local obstructions. For the first, this is a prime ideal above 19. For the second, the ramified prime above 23:

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

More local examples:

```
sage: K.hilbert_symbol(a, 0, K.ideal(5))
0
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:

```python
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:

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

Primes above 2:

```python
sage: K.<a> = NumberField(x^5 - 23)
sage: 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:

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

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

```python
sage: K.<a> = QuadraticField(5)
sage: K.hilbert_symbol(-1, -1, K.embeddings(CC)[0])
Traceback (most recent call last):
  ...
ValueError: Possibly real place (=Ring morphism:  
  From: Number Field in a with defining polynomial x^2 - 5 with a = 2. 
  -->2360679774997969762
  To:   Complex Field with 53 bits of precision
  Defn: a |--> -2.2360679774997969762 given as complex embedding in hilbert_symbol.
  ...Is it real or complex?
```

(continues on next page)
sage: K.hilbert_symbol(-1, -1, K.embeddings(QQbar)[0])
Traceback (most recent call last):
...
ValueError: Possibly real place (=Ring morphism:
  From: Number Field in a with defining polynomial x^2 - 5 with a = 2.
  -->236067977499790?
  To:  Algebraic Field
  Defn: a |--> -2.236067977499790?) given as complex embedding in hilbert_
  symbol. Is it real or complex?
sage: K.<b> = QuadraticField(-5)
sage: K.hilbert_symbol(-1, -1, K.embeddings(CDF)[0])
1
sage: K.hilbert_symbol(-1, -1, K.embeddings(QQbar)[0])
1

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
\[\text{sage: K.hilbert_symbol(a+5, 2, P)}\]
\[-1\]
\[\text{sage: K.<a> = NumberField(x^3 - 4*x + 2)}\]
\[\text{sage: K.hilbert_symbol(2, -2, K.primes_above(2)[0])}\]
\[1\]

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

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

AUTHOR:

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

hilbert_symbol_negative_at_S(S, b, check=True)

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

INPUT:

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

OUTPUT:

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

ALGORITHM:

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

EXAMPLES:

\[\text{sage: K.<a> = NumberField(x^2 + 20072)}\]
\[\text{sage: S = [K.primes_above(3)[0], K.primes_above(23)[0]]}\]
\[\text{sage: b = K.hilbert_symbol_negative_at_S(S, a + 1)}\]
\[\text{sage: [K.hilbert_symbol(b, a + 1, p) for p in S]}\]
\[-1, -1\]
\[\text{sage: K.<d> = CyclotomicField(11)}\]
\[\text{sage: S = [K.primes_above(2)[0], K.primes_above(11)[0]]}\]
\[\text{sage: b = d + 5}\]
\[\text{sage: a = K.hilbert_symbol_negative_at_S(S, b)}\]
\[\text{sage: [K.hilbert_symbol(a,b,p) for p in S]}\]
Note that the closely related hilbert conductor takes only the finite places into account:

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

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

**is_absolute()**
Return True since self is an absolute field.

**EXAMPLES:**

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

**minkowski_embedding**

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 \(\text{RealField}(prec)\) instead of \(\text{RDF}\).

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:**

```python
sage: F.<alpha> = NumberField(x^3+2)
sage: F.minkowski_embedding()
[ 1.00000000000000  1.25992104989487  1.58740105196820]
[ 0.740078950105127  0.890898718108442  2.84732210186307]
```

(continues on next page)
optimized_representation(name=None, both_maps=True)

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

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

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

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()  # representation varies, not tested
(Number Field in a1 with defining polynomial x^3 - 7*a - 7,
 Ring morphism:
  From: Number Field in a1 with defining polynomial x^3 - 7*a - 7
  To:   Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
     Defn: a1 |--> 7/225*a^2 - 7/75*a - 42/25,
 Ring morphism:
  From: Number Field in a with defining polynomial 7/9*x^3 + 7/3*x^2 - 56*x + 123
  To:   Number Field in a1 with defining polynomial x^3 - 7*a - 7
     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 + 576
sage: L = K.optimized_subfields(name='b')[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,
 Number Field in b3 with defining polynomial x^4 - 2*x^2 + 4,
 Number Field in b4 with defining polynomial x^8 + 4*x^6 + 7*x^4 + 36*x^2 + 81]
```

We examine one of the optimized subfields in more detail:

```
sage: M, from_M, to_M = L[2]
Number Field in b2 with defining polynomial x^4 - 5*x^2 + 25
sage: from_M
# may be slightly random
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
sage: to_M
```

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

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

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

`order(*args, **kwds)`

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

**INPUT:**

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

**EXAMPLES:**

```python
sage: 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):
...
ValueError: the rank of the span of gens is wrong
sage: k.order(i/2)
Traceback (most recent call last):
...
ValueError: each generator must be integral
```

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

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

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

Return the collection of all infinite places of 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 prec is not None, we simply return places into RealField(prec) and ComplexField(prec) (or RDF, CDF if prec=53). One can also use prec=\texttt{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 all_complex, which defaults to False. If all_complex is True, then the real embeddings are returned as embeddings into CIF instead of RIF.

**EXAMPLES:**

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

real_places(prec=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 |--> -1.62657656169778543211232345494,
Ring morphism:
From: Number Field in alpha with defining polynomial x^4 - 7
To: Real Field with 106 bits of precision
Defn: alpha |--> 1.62657656169778543211232345494]

relative_degree()
A synonym for degree.

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

relative_different()
A synonym for different.
EXAMPLES:

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

**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:

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

**relative_vector_space(*args, **kwds)**
A synonym for vector_space.

EXAMPLES:

```
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=None, 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
- `names` - 2-tuple of names of generator for output field \( K \) and the subfield \( \mathbb{Q}(\alpha) \) names[0] generators \( K \) and names[1] \( \mathbb{Q}(\alpha) \).
- `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, \( \text{K.structure()} \) 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> = 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 + 1/57*d - 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)
c
sage: to_L(a^4 + a^2 + 2)
d
sage: from_L(to_L(a^4 + a^2 + 2))
a^4 + a^2 + 2
```

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
```

(continues on next page)
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:

sage: K.<a> = NumberField(\[x^3 - 2, x^2 + x + 1\] )
sage: K = K.absolute_field('b')
sage: S = K.subfields()
sage: len(S)
6
sage: [k[0].polynomial() for k in S]
[x - 3, 
x^2 - 3*x + 9, 
x^3 - 3*x^2 + 3*x + 1, 
x^3 - 3*x^2 + 3*x + 1, 
x^3 - 3*x^2 + 3*x - 17, 
x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 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

sage.rings.number_field.number_field.NumberField_absolute_v1\(poly, name, latex_name,
canonical_embedding=None\)

Used for unpickling old pickles.

EXAMPLES:

sage: from sage.rings.number_field.number_field import NumberField_absolute_v1
sage: R.<x> = QQ[]
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)  

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:

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

sage: K = CyclotomicField(197)  
sage: loads(K.dumps()) == K  
True

sage: loads((z^2).dumps()) == z^2  
True

sage: cf12 = CyclotomicField(12)  
sage: z12 = cf12.0

cf6 = CyclotomicField(6)

cf3 = CyclotomicField(3)

sage: FF = Frac(cf12[x])  
sage: x = FF.0

ez6 = cf6.0

cf3(z6)  
zeta3 + 1

cf6(z3)  
zeta6 - 1

sage: type(cf6(z3))  
<class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic'>

cf1 = CyclotomicField(1) ; z1 = cf1.0  
sage: cf3(z1)  
1

sage: K = CyclotomicField(197)  
sage: loads(K.dumps()) == K  
True

sage: loads((z^2).dumps()) == z^2  
True

sage: cf6 = CyclotomicField(6) ; z6 = cf6.gen(0)

cf3 = CyclotomicField(3) ; z3 = cf3.gen(0)

cf3(z6)  
zeta3 + 1

cf6(z3)  
zeta6 - 1

sage: type(cf6(z3))  
<class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic'>

sage: cf1 = CyclotomicField(1) ; z1 = cf1.0

cf3(z1)  
1

Chapter 1. Algebraic Number Fields
sage: type(cf3(z1))
<class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic'>

complex_embedding(prec=53)

Return the embedding of this cyclotomic field into the approximate complex field with precision 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 zeta is computed (using sin and cosine in MPFR) means that only the 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(prec=53)

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

If you want 53-bit double precision, which is faster but less reliable, then do 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 + 0.951056516292473*I,
  Ring morphism:
    From: Cyclotomic Field of order 5 and degree 4
    To:   Complex Field with 53 bits of precision
    Defn: zeta5 |--> -0.809016994374947 + 0.5877852522473*I,
  Ring morphism:
    From: Cyclotomic Field of order 5 and degree 4
    To:   Complex Field with 53 bits of precision
    Defn: zeta5 |--> -0.809016994374947 - 0.5877852522473*I,
  Ring morphism:
construction()

Return data defining a functorial construction of self.

EXAMPLES:

```
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()

Return the different ideal of the cyclotomic field self.

EXAMPLES:

```
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)

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

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

INPUT:

- v (optional) - list of element of this number field

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

EXAMPLES:

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

embeddings(K)

Compute all field embeddings of this field into the field K.
INPUT:

• K – a field

EXAMPLES:

```sage
sage: CyclotomicField(5).embeddings(ComplexField(53))[1]
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   Complex Field with 53 bits of precision
  Defn: zeta5 |--> -0.809016994374947 + 0.587785252292473*I
sage: CyclotomicField(5).embeddings(Qp(11, 4, print_mode='digits'))[1]
Ring morphism:
  From: Cyclotomic Field of order 5 and degree 4
  To:   11-adic Field with capped relative precision 4
  Defn: zeta5 |--> ...1525
```

**is_abelian()**

Return True since all cyclotomic fields are automatically abelian.

EXAMPLES:

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

**is_galois()**

Return True since all cyclotomic fields are automatically Galois.

EXAMPLES:

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

**is_isomorphic**(other)

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

EXAMPLES:

```sage
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 \rightarrow '))
True
```

Check trac ticket #14300:

```sage
sage: K = CyclotomicField(4)
sage: N = K.extension(x^2-5, 'z')
sage: K.is_isomorphic(N)
False
```

(continues on next page)
```python
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:**

```python
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:**

```python
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 \( \text{prec} \).

Mostly, of course, there are no such embeddings.

**EXAMPLES:**

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

**roots_of_unity**

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

**EXAMPLES:**

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

```sage
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
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)
```

1.1. Number Fields
```python
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:**

```python
sage: CyclotomicField(1).zeta_order()
2
sage: CyclotomicField(4).zeta_order()
4
sage: CyclotomicField(5).zeta_order()
10
```

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

Used for unpickling old pickles.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.number_field import NumberField_cyclotomic_v1
sage: NumberField_cyclotomic_v1(5,'a')
Cyclotomic Field of order 5 and degree 4
sage: NumberField_cyclotomic_v1(5,'a').variable_name()
'a'
```

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

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

**EXAMPLES:**

```python
sage: K.<a> = NumberField(x^3 - 2); K
Number Field in a with defining polynomial x^3 - 2
sage: TestSuite(K).run()
```
**S_class_group**($S$, proof=None, names=None)

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

**INPUT:**

- $S$ - a set of primes of the base field
- proof - if False, assume the GRH in computing the class group. Default is True. Call `number_field_proof` to change this default globally.
- 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 with a = 2.236067977499790?*I
```

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 with a = 2.236067977499790?*I
```

**S_unit_group**(proof=None, $S=None$)

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

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

**INPUT:**

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

**Note:** The group is cached.

**EXAMPLES:**

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

(continues on next page)
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:

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:

sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=K.ideal(6).prime_factors()); U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial x^3 + 3 with S = (Fractional ideal (-a^2 + a - 1), Fractional ideal (a + 1), Fractional ideal (a))
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=K.ideal(6)); U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial x^3 + 3 with S = (Fractional ideal (-a^2 + a - 1), Fractional ideal (a + 1), Fractional ideal (a))
sage: K.<a> = NumberField(x^3 + 3)
sage: U = K.S_unit_group(proof=False, S=6); U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial x^3 + 3 with S = (Fractional ideal (-a^2 + a - 1), Fractional ideal (a + 1), Fractional ideal (a))

sage: U
S-unit group with structure C2 x Z x Z x Z x Z of Number Field in a with defining polynomial x^3 + 3 with S = (Fractional ideal (-a^2 + a - 1), Fractional ideal (a + 1), Fractional ideal (a))
sage: U.primes()
(Fractional ideal (-a^2 + a - 1), Fractional ideal (a + 1), Fractional ideal (a))
sage: U.gens()
(u0, u1, u2, u3, u4)
sage: U.gens_values()
[-1, a^2 - 2, -a^2 + a - 1, a + 1, a]

The 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
\(-6a^2 + 18a - 54\)

\[
sage: u.norm().factor()
-1 * 2^9 * 3^5
\]

\[
sage: U.log(u)
(1, 1, 4, 1, 5)
\]

**S_unit_solutions**\((S=\emptyset, \text{prec}=106, \text{include_exponents}=\text{False}, \text{include_bound}=\text{False}, \text{proof}=\text{None})\)

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

**INPUT:**
- \(S\) – a list of finite primes in this number field
- \(\text{prec}\) – 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)
- \(\text{proof}\) – if False, assume the GRH in computing the class group. Default is True.

**OUTPUT:**
A list of tuples \(\left( (A_1, B_1, x_1, y_1), (A_2, B_2, x_2, y_2), \ldots, (A_n, B_n, x_n, y_n) \right)\) 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:**

\[
sage: K.<xi> = NumberField(x^2+x+1)
sage: S = K.primes_above(3)
sage: K.S_unit_solutions(S) \# random, due to ordering
[[(xi + 2, -xi - 1), (1/3*xi + 2/3, -1/3*xi + 1/3), (-xi, xi + 1), (-xi + 1, xi)]
\]

You can get the exponent vectors:

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

And the computed bound:

\[
sage: solutions, bound = K.S_unit_solutions(S, \text{prec}=100, \text{include_bound}=\text{True})
sage: bound
7
\]

\(S_units(S, \text{proof}=\text{True})\)

1.1. Number Fields 41
Return a list of generators of the S-units.

**INPUT:**
- S – a set of primes of the base field
- proof – if False, assume the GRH in computing the class group

**OUTPUT:**
A list of generators of the unit group.

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

**EXAMPLES:**

```sage
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 with a = 1.732050807568878?I
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):

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

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

```sage
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 Q.

**EXAMPLES:**

```sage
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 QQbar).

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)
sage: K.algebraic_closure()
Algebraic Field
```

**change_generator(alpha, name=None, names=None)**

Given the number field self, construct another isomorphic number field K generated by the element alpha of self, along with isomorphisms from K to self and from self to K.

EXAMPLES:

```python
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 with i0 = 1/2*i + 3
sage: from_K
Ring morphism:
  From: Number Field in i0 with defining polynomial x^2 - 6*x + 37/4 with i0 = 1/2*i + 3
  To:   Number Field in i with defining polynomial x^2 + 1
  Defn: i0 |--> 1/2*i + 3
sage: to_K
```
Ring morphism:
From: Number Field in i with defining polynomial \( x^2 + 1 \)
To: Number Field in i0 with defining polynomial \( x^2 - 6x + 37/4 \) with i0 = \( \frac{1}{2}i + 3 \)
Defn: i \( \mapsto 2i0 - 6 \)

We can also do

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

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

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

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

```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:**

```sage```
k.<a> = NumberField(x^99 + 2); k
Number Field in a with defining polynomial \( x^{99} + 2 \)
```

```sage```
k.characteristic()
0
```

**class_group**(proof=\text{None}, names='c')
Return the class group of the ring of integers of this number field.

**INPUT:**

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

**OUTPUT:** The class group of this number field.

**EXAMPLES:**

```sage```
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:

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

EXAMPLES:

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)

Return a list of rational primes which split completely in the number field K.
INPUT:

- B – a positive integer bound (default: 200)

OUTPUT:

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

EXAMPLES:

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

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

Return 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
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. 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
sage: QuadraticField(-1, 'I').complex_conjugation()
Ring endomorphism of Number Field in I with defining polynomial x^2 + 1 with I ↦ 1*I
  Defn: I |--> -I
sage: CyclotomicField(8).complex_conjugation()
Ring endomorphism of Cyclotomic Field of order 8 and degree 4
  Defn: zeta8 |--> -zeta8^3
sage: QuadraticField(5, 'a').complex_conjugation()
Identity endomorphism of Number Field in a with defining polynomial x^2 - 5 with a = 2.236067977499790?
```

(continues on next page)
Relative number field endomorphism of Number Field in a with defining polynomial $x^2 + 1$ over its base field
- Defn: $a \mapsto -a$
- $b \mapsto b$

```sage
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
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 $(F, \text{self_into}_F, \text{other_into}_F, k)$ such that
\text{self_into}_F is an embedding of \text{self} in \text{F}, \text{other_into}_F is an embedding of \text{other} in \text{F}, and \text{k} is one of the following:

• an integer such that \text{F.gen()} equals \text{other_into}_F(\text{other.gen()}) + \text{k*self_into}_F(\text{self.gen()});
• Infinity, in which case \text{F.gen()} equals \text{self_into}_F(\text{self.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 \text{F} will have an embedding with
respect to which both \text{self_into}_F and \text{other_into}_F will be compatible with the ambient embeddings.

If \text{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)
[Number Field in a with defining polynomial x^4 - 2 with a = 1.189207115002722?, Number Field in a0 with defining polynomial x^8 + 28*x^4 + 2500]
sage: [F, f, g, k], = K1.composite_fields(K2, both_maps=True); F
Number Field in a with defining polynomial x^4 - 2 with a = 1.189207115002722?
sage: f(K1.0), g(K2.0)
(a, -a)
```

With \text{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 with a = 1.189207115002722?, 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 with a0 = -2.˓→378414230005443? + 1.189207115002722?*I
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
\textbf{conductor}(\texttt{check_abelian=True})

Computes the conductor of the abelian field \(K\). If \texttt{check_abelian} is set to false and the field is not an abelian extension of \(\mathbb{Q}\), the output is not meaningful.

\textbf{INPUT:}

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

\textbf{OUTPUT:}

Integer which is the conductor of the field.

\textbf{EXAMPLES:}

\begin{verbatim}
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()
\end{verbatim}
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.

construction()

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]

decomposition_type(p)

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

INPUT:

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

OUTPUT:

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

• \(e\) is the ramification index,

• \(f\) is the residue class degree,

• \(g\) is the number of primes above \(p\) with given \(e\) and \(f\)

EXAMPLES:

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

```python
sage: K.discriminant().factor()
2^20 * 3^10 * 53^10
```

```python
sage: K.decomposition_type(2)
[(2, 5, 2)]
```

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

```python
sage: K.decomposition_type(53)
[(2, 2, 5)]
```

This example is only ramified at 11:

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

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

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

Computing the decomposition type is feasible even in large degree:

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

```python
sage: K.discriminant().factor(limit=100000)
2^144 * 3^288 * 7^18 * 11^17 * 31^18 * 157^18 * 2153^18 * 13907^18 * ...
```

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

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

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

It also works for relative extensions:

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

```python
sage: M.<c> = K.extension(x^10 - 6*x^8 + (a + 12)*x^6 + (-7/2*a - 89/2)*x^4 +
→(13/2*a - 77/2)*x^2 + 25)
```

There is a unique prime above 11 and above 13 in $K$, each of which is unramified in $M$:

```python
sage: M.decomposition_type(11)
[(1, 2, 5)]
```

```python
sage: M.primes_above(11)[0]
```

```python
sage: len(M.primes_above(P11))
5
```

```python
sage: M.decomposition_type(13)
[(1, 1, 10)]
```

```python
sage: M.primes_above(13)[0]
```

```python
sage: len(M.primes_above(P13))
10
```

There are two primes above 2, each of which ramifies in $M$:
sage: Q0, Q1 = K.primes_above(2)
sage: M.decomposition_type(Q0)
[(2, 5, 1)]
sage: q0, = M.primes_above(Q0)
sage: q0.residue_class_degree()
5
sage: q0.relative_ramification_index()
2
sage: M.decomposition_type(Q1)
[(2, 5, 1)]

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:

sage: NumberField(x^3 + x^2 + 997*x + 1, 'a').degree()
3
sage: NumberField(x + 1, 'a').degree()
1
sage: NumberField(x^997 + 17*x + 3, 'a', check=False).degree()
997

different()

Compute the different fractional ideal of this number field.

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

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

See Wikipedia article Different_ideal

EXAMPLES:

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

(continues on next page)
The different is cached:

```python
sage: d = k.different(); d
Fractional ideal (2*b)
```

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
```

dirichlet_group()

Given an abelian field $K$, this computes and returns the set of all Dirichlet characters corresponding to the characters of the Galois group of $K/Q$.

The output is random if the field is not abelian

OUTPUT:

• a list of Dirichlet characters

EXAMPLES:

```python
sage: K.<t> = NumberField(x^3+x^2-36*x-4)
sage: K.conductor()
109
sage: K.dirichlet_group()
[Dirichlet character modulo 109 of conductor 1 mapping 6 |--> 1,
Dirichlet character modulo 109 of conductor 109 mapping 6 |--> zeta3,
Dirichlet character modulo 109 of conductor 109 mapping 6 |--> -zeta3 - 1]
```

```python
sage: K = CyclotomicField(44)
sage: L = K.subfields(5)[0][0]
sage: X = L.dirichlet_group()
sage: X
[Dirichlet character modulo 11 of conductor 1 mapping 2 |--> 1,
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5,
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5^2,
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5^3,
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> -zeta5^3 - zeta5^2
      \rightarrow zeta5 - 1]
sage: X[4]^2
Dirichlet character modulo 11 of conductor 11 mapping 2 |--> zeta5^3
sage: X[4]^2 in X
True
```

disc($v=\text{None}$)

Shortcut for self.discriminant.
EXAMPLES:

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

discriminant(v=None)

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

INPUT:

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

OUTPUT:

Integer if $v$ is omitted, and Rational otherwise.

EXAMPLES:

```
sage: 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
  • $proof$ – boolean (default: True, unless you called proof.number_field() and set it otherwise)

OUTPUT:

A complete system of integral elements of norm $n$, modulo units of positive norm.

EXAMPLES:

```
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, latex_name=None, latex_names=None, *args, **kwds)

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

EXAMPLES:

```
sage: K.<a> = NumberField(x^2+1)
sage: R.<t> = K[]
```

We create another extension:

```python
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 $y^2 + 2$:

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

A relative extension of a relative extension:

```python
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):

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

**factor**($n$)

Ideal factorization of the principal ideal generated by $n$.

**EXAMPLES:**

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

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

Here are the factors:

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

Now we extract the reduced form of the generators:

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

We recover the integer that was factored in $\mathbb{Z}[I]$ (up to a unit):
One can also factor elements or ideals of the number field:

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

An example over a relative number field:

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

```sage
sage: f
# representation varies, not tested
(Fractional ideal (1/2*a*b - a + 1/2)) * (Fractional ideal (-1/2*a*b - a + 1/2))
sage: f.value() == a+1
True
```

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'
```
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: 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, gc_numbering=None)

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

INPUT:

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

The resulting group will only compute with automorphisms when necessary, so certain functions (such as `sage.rings.number_field.galois_group.GaloisGroup_v2.order()`) will still be fast. For more (important!) documentation, see the documentation for Galois groups of polynomials over Q, e.g., by typing `K.polynomial().galois_group?`, where K is a number field.

EXAMPLES:

sage: k.<b> = NumberField(x^2 - 14)  # a Galois extension
sage: G = k.galois_group(); G
Galois group 2T1 (S2) with order 2 of x^2 - 14
sage: G.gen(0)
(1,2)
sage: G.gen(0)(b)
-b
sage: G.artin_symbol(k.primes_above(3)[0])
(1,2)

sage: k.<b> = NumberField(x^3 - x + 1)  # not Galois
sage: G = k.galois_group(names='c'); G
Galois group 3T2 (S3) with order 6 of x^3 - x + 1
sage: G.gen(0)
(1,2,3)(4,5,6)
EXPLICIT GALOIS GROUP: We compute the Galois group as an explicit group of automorphisms of the Galois closure of a field.

```sage```
sage: K.<a> = NumberField(x^3 - 2)
sage: L.<b1> = K.galois_closure(); L
Number Field in b1 with defining polynomial x^6 + 108
sage: G = End(L); G
Automorphism group of Number Field in b1 with defining polynomial x^6 + 108
sage: G.list()
[Ring endomorphism of Number Field in b1 with defining polynomial x^6 + 108
  Defn: b1 |--> b1,
  ...
Ring endomorphism of Number Field in b1 with defining polynomial x^6 + 108
  Defn: b1 |--> -1/12*b1^4 - 1/2*b1]
sage: G[2](b1)
1/12*b1^4 + 1/2*b1
```

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

If $L/K$ is a relative number field, this method will currently return $Gal(L/Q)$. This behavior will change in the future, so it's better to explicitly call `absolute_field()` if that is the desired behavior:

```sage```
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()
...DeprecationWarning: Use absolute_field().galois_group() if you want the_
  Galois group of the absolute field
See https://trac.sagemath.org/28782 for details.
Galois group 10T22 (S(5)[x]2) with order 240 of t^5 - t + a
```

`gen(n=0)`
Return the generator for this number field.

**INPUT:**

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

**EXAMPLES:**

```sage```
sage: k.<theta> = NumberField(x^14 + 2); k
Number Field in theta with defining polynomial x^14 + 2
sage: k.gen()
theta
sage: k.gen(1)
```

(continues on next page)
gen_embedding()

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

EXAMPLES:

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

ideal(*gens, **kwds)

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

EXAMPLES:

```sage
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
```

idealchinese(ideals, residues)

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

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

INPUT:

• ideals - a list of ideals of the number field.
• residues - a list of elements of the number field.

OUTPUT:

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

See also:

• crt()

EXAMPLES:

This is the example from the pari page on idealchinese:

```sage
sage: K.<sqrt2> = NumberField(sqrt(2).minpoly())
sage: ideals = [K.ideal(4),K.ideal(3)]
sage: residues = [sqrt2,1]
sage: r = K.idealchinese(ideals,residues); r
-3*sqrt2 + 4
sage: all((r - a) in I for I,a in zip(ideals,residues))
True
```
The result may be non-integral if the results are non-integral:

```python
sage: K.<sqrt2> = NumberField(sqrt(2).minpoly())
sage: ideals = [K.ideal(4),K.ideal(21)]
sage: residues = [1/sqrt2,1]
sage: r = K.idealchinese(ideals,residues); r
-r63/2*sqrt2 - 20
sage: all(
    ....:     (r-a).valuation(P) >= k
    ....:     for I,a in zip(ideals,residues)
    ....:     for P,k in I.factor())
True
```

`ideals_of_bdd_norm(bound)`

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:**

```python
sage: K.<a> = NumberField(x^2 + 23)
sage: d = K.ideals_of_bdd_norm(10)
sage: for n in d:
    ....:     print(n)
    ....:     for I in sorted(d[n]):
    ....:         print(I)
1
Fractional ideal (1)
2
Fractional ideal (2, 1/2*a - 1/2)
Fractional ideal (2, 1/2*a + 1/2)
3
Fractional ideal (3, 1/2*a - 1/2)
Fractional ideal (3, 1/2*a + 1/2)
4
Fractional ideal (2)
Fractional ideal (4, 1/2*a + 3/2)
Fractional ideal (4, 1/2*a + 5/2)
5
6
Fractional ideal (1/2*a - 1/2)
Fractional ideal (1/2*a + 1/2)
Fractional ideal (6, 1/2*a + 5/2)
Fractional ideal (6, 1/2*a + 7/2)
7
8
Fractional ideal (4, a - 1)
Fractional ideal (4, a + 1)
Fractional ideal (1/2*a + 3/2)
Fractional ideal (1/2*a - 3/2)
9
(continues on next page)
Fractional ideal (3)
Fractional ideal (9, 1/2*a + 7/2)
Fractional ideal (9, 1/2*a + 11/2)

\texttt{integral\_basis}(v=None)

Return a list containing a ZZ-basis for the full ring of integers of this number field.

\textbf{INPUT:}

- \texttt{v} - None, a prime, or a list of primes. See the documentation for \texttt{self.maximal\_order}.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: K.<a> = NumberField(x^5 + 10*x + 1)
sage: K.integral\_basis()
[1, a, a^2, a^3, a^4]
\end{verbatim}

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.

\begin{verbatim}
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]
\end{verbatim}

\textbf{ALGORITHM:} Uses the pari library (via \texttt{_pari\_integral\_basis}).

\texttt{is\_CM}()

Return True if self is a CM field (i.e. a totally imaginary quadratic extension of a totally real field).

\textbf{EXAMPLES:}

\begin{verbatim}
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
\end{verbatim}

The following are non-CM totally imaginary fields.
```python
sage: F.<a> = NumberField(x^4 + x^3 - x^2 - x + 1)
sage: F.is_totally_imaginary()
True
sage: F.is_CM()
False
sage: F2.<a> = NumberField(x^12 - 5*x^11 + 8*x^10 - 5*x^9 - 
                          x^8 + 9*x^7 + 7*x^6 - 3*x^5 + 5*x^4 + 
                          7*x^3 - 4*x^2 - 7*x + 7)
sage: F2.is_totally_imaginary()
True
sage: F2.is_CM()
False
```

The following is a non-cyclotomic CM field.

```python
sage: M.<a> = NumberField(x^4 - x^3 - x^2 - 2*x + 4)
sage: M.is_CM()
True
```

Now, we construct a totally imaginary quadratic extension of a totally real field (which is not cyclotomic).

```python
sage: E_0.<a> = NumberField(x^7 - 4*x^6 - 4*x^5 + 10*x^4 + 4*x^3 - 
                          6*x^2 - x + 1)
sage: E_0.is_totally_real()
True
sage: E.<b> = E_0.extension(x^2 + 1)
sage: E.is_CM()
True
```

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

```python
sage: E_0.<a> = NumberField(x^2 - 4*x + 16)
sage: y = polygen(E_0)
sage: E.<z> = E_0.extension(y^2 - E_0.gen() / 2)
sage: E.is_CM()
True
sage: E.is_CM_extension()
False
```

### is_abelian()

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

**EXAMPLES:**

```python
sage: NumberField(x^2 + 1, 'i').is_abelian()
True
sage: NumberField(x^3 + 2, 'a').is_abelian()
False
sage: NumberField(x^3 + x^2 - 2*x - 1, 'a').is_abelian()
True
sage: NumberField(x^6 + 40*x^3 + 1372, 'a').is_abelian()
False
sage: NumberField(x^6 + x^5 - 5*x^4 - 4*x^3 + 6*x^2 + 3*x - 1, 'a').is_abelian()
True
```
is_absolute()
Return True if self is an absolute field.
This function will be implemented in the derived classes.

EXAMPLES:

```
sage: 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:

```
sage: NumberField(x^2 + 1, 'i').is_galois()
True
sage: NumberField(x^3 + 2, 'a').is_galois()
False
sage: NumberField(x^2 + 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').is_galois()
True
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'))
(continues on next page)
```

1.1. Number Fields 63
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 contained in the real numbers.

EXAMPLES:

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:

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

```python
lmfdb_page()
Open the LMFDB web page of the number field in a browser.
See https://www.lmfdb.org
```

EXAMPLES:

```python
sage: E = QuadraticField(-1)
sage: E.lmfdb_page()  # optional -- webbrowser
```

Even if the variable name is different it works:

```python
sage: R.<y>= PolynomialRing(QQ, "y")
sage: K = NumberField(y^2 + 1 , "i")
sage: K.lmfdb_page()  # optional -- webbrowser
```

```python
maximal_order(v=None, assume_maximal='non-maximal-non-unique')
Return the maximal order, i.e., the ring of integers, associated to this number field.
```

INPUT:

- \(v\) - \(\text{None}, \text{a prime}, \text{or a list of integer primes} \) (default: \(\text{None}\))
  - if \(\text{None}\), return the maximal order.
  - if a prime \(p\), return an order that is \(p\)-maximal.
  - if a list, return an order that is maximal at each prime of these primes
- \(\text{assume_maximal} \) - \(\text{True}, \text{False}, \text{None}, \text{or "non-maximal-non-unique"} \) (default: \"non-maximal-non-unique\") ignored when \(v\) is \(\text{None}\); otherwise, controls whether we assume that the order \(\text{order.is_maximal()}\) outside of \(v\).
  - if \(\text{True}\), the order is assumed to be maximal at all primes.
  - if \(\text{False}\), the order is assumed to be non-maximal at some prime not in \(v\).
  - if \(\text{None}\), no assumptions are made about primes not in \(v\).
  - if \"non-maximal-non-unique\" (deprecated), like \(\text{False}\), however, the order is not a unique parent, so creating the same order later does typically not poison caches with the information that the order is not maximal.

EXAMPLES:

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

```python
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)
K.<a> = NumberField(x^3 - p*q)
K.maximal_order([3], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]
K.maximal_order([2], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]
K.maximal_order([p], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]
K.maximal_order([q], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]
K.maximal_order([p, 3], assume_maximal=None).basis()
[1/3*a^2 + 1/3*a + 1/3, a, a^2]
```

An example with bigger discriminant:

```sage
p = next_prime(10^97)
q = next_prime(10^99)
K.<a> = NumberField(x^3 - p*q)
K.maximal_order(prime_range(10000), assume_maximal=None).basis()
[1, a, a^2]
```

An example in a relative number field:

```sage
K.<a, b> = NumberField([x^2 + 1, x^2 - 3])
OK = K.maximal_order()
OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
charpoly(OK.1)
x^2 + b*x + 1
charpoly(OK.2)
x^2 - x + 1
O2 = K.order([3*a, 2*b])
O2.index_in(OK)
144
```

An order that is maximal at a prime. We happen to know that it is actually maximal and mark it as such:

```sage
K.<i> = NumberField(x^2 + 1)
K.maximal_order(v=2, assume_maximal=True)
Gaussian Integers in Number Field in i with defining polynomial x^2 + 1
```

It is an error to create a maximal order and declare it non-maximal, however, such mistakes are only caught automatically if they evidently contradict previous results in this session:
sage: K.maximal_order(v=2, assume_maximal=False)
Traceback (most recent call last):
... 
ValueError: cannot assume this order to be non-maximal because we already found it to be a maximal order

maximal_totally_real_subfield()

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

EXAMPLES:

sage: F.<a> = QuadraticField(11)
sage: F.maximal_totally_real_subfield()
[Number Field in a with defining polynomial x^2 - 11 with a = 3.316624790355400?, Identity endomorphism of Number Field in a with defining polynomial x^2 - 11 with a = 3.316624790355400?]
sage: F.<a> = QuadraticField(-15)
sage: F.maximal_totally_real_subfield()
[Rational Field, Natural morphism:
  From: Rational Field
  To: Number Field in a with defining polynomial x^2 + 15 with a = 3.
  872983346207417*I]
sage: F.<a> = CyclotomicField(29)
sage: F.maximal_totally_real_subfield()
(Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 - 12*x^11 + 66*x^10 + 55*x^9 - 165*x^8 - 120*x^7 + 210*x^6 + 126*x^5 - 126*x^4 - 56*x^3 + 28*x^2 + 7*x - 1 with a0 = 1.953241111420174?, Ring morphism:
  From: Number Field in a0 with defining polynomial x^14 + x^13 - 13*x^12 - 12*x^11 + 66*x^10 + 55*x^9 - 165*x^8 - 120*x^7 + 210*x^6 + 126*x^5 - 126*x^4 - 56*x^3 + 28*x^2 + 7*x - 1 with a0 = 1.953241111420174?
  To: Cyclotomic Field of order 29 and degree 28
  Defn: a0 |--> -a^27 - a^26 - a^25 - a^24 - a^23 - a^22 - a^21 - a^20 - a^19 - a^18 - a^17 - a^16 - a^15 - a^14 - a^13 - a^12 - a^11 - a^10 - a^9 - a^8 - a^7 - a^6 - a^5 - a^4 - a^3 - a^2 - 1)
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 - x^3 + 2*x^2 + x + 1
  Defn: a1 |--> -1/2*a^3 - 1/2]
sage: F.<a> = NumberField(x^4-4*x^2-x+1)
An example of a relative extension where the base field is not the maximal totally real subfield.

```python
sage: E_0.<a> = NumberField(x^2 - 4*x + 16)
sage: y = polygen(E_0)
sage: E.<z> = E_0.extension(y^2 - E_0.gen() / 2)
sage: E.maximal_totally_real_subfield()
[Number Field in z1 with defining polynomial x^2 - 2*x - 5, Composite map:
  From: Number Field in z1 with defining polynomial x^2 - 2*x - 5
  To: Number Field in z with defining polynomial x^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 |--> -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
gens()
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:**

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

order()

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

OUTPUT: always positive infinity

**EXAMPLES:**

```
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:**

```
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: len(k.pari_nf())
9
sage: k.<a> = NumberField(x^7 + 7); k
(continues on next page)
```
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:**

```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```
p = next_prime(10^40); q = next_prime(10^41)
K.<a> = NumberField(x^2 - p*q)
K.pari_nf(important=False)
Traceback (most recent call last):
... RuntimeWarning: Unable to factor discriminant with trial division
```
```python
sage: K.<a> = NumberField(x^2 - p*q, maximize_at_primes=[p])
sage: K.pari_nf()
[y^2 - 100000000000000000000...]
```

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

```python
sage: K.<a> = NumberField(x^2 - p*q, assume_disc_small=True)
sage: K.pari_nf()
[y^2 - 100000000000000000000...]
```

### pari_polynomial

Return the PARI polynomial corresponding to this number field.

**INPUT:**

- `name` - variable name (default: `'x'`)

**OUTPUT:**

A monic polynomial with integral coefficients (PARI `t_POL`) defining the PARI number field corresponding to `self`.

**Warning:** This is not the same as simply converting the defining polynomial to PARI.

**EXAMPLES:**

```python
sage: y = polygen(QQ)
sage: k.<a> = NumberField(y^2 - 3/2*y + 5/3)
sage: k.pari_polynomial()
x^2 - x + 40
sage: k.polynomial().__pari__()
x^2 - 3/2*x + 5/3
sage: k.pari_polynomial('a')
a^2 - a + 40
```

Some examples with relative number fields:

```python
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()
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:
```python
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
```

```python
pari_rnfnorm_data(L, proof=True)
Return the PARI pari:rnfisnorminit data corresponding to the extension L/self.
EXAMPLES:

```python
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
```

```python
pari_zk()
Integral basis of the PARI number field corresponding to this field.
This is the same as pari_nf().getattr('zk'), but much faster.
EXAMPLES:

```python
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]
```

```python
polynomial()
Return the defining polynomial of this number field.
This is exactly the same as self.defining_polynomial().
EXAMPLES:

```python
sage: NumberField(x^2 + (2/3)*x - 9/17, 'a').polynomial()
x^2 + 2/3*x - 9/17
```

```python
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').polynomial_ntl()
([-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_basis()**

Return a power basis for this number field over its base field.

If this number field is represented as \( k[t]/f(t) \), then the basis returned is \( 1, t, t^2, \ldots, t^{d-1} \) where \( d \) is the degree of this number field over its base field.

EXAMPLES:

```
sage: 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]
```

```
sage: L.absolute_field('c').power_basis()
[1, c, c^2, c^3, c^4, c^5, c^6, c^7, c^8, c^9]
```

```
sage: M = CyclotomicField(15)
sage: M.power_basis()
[1, zeta15, zeta15^2, zeta15^3, zeta15^4, zeta15^5, zeta15^6, zeta15^7]
```

### 1.1. Number Fields
**prime_above**(*x, degree=None*)

Return a prime ideal of self lying over *x*.

**INPUT:**

- **x**: usually an element or ideal of self. It should be such that self.ideal(*x*) is sensible. This excludes *x=0*.

- **degree** (default: **None**): **None** or an integer. If one, find a prime above *x* of any degree. If an integer, find a prime above *x* such that the resulting residue field has exactly this degree.

**OUTPUT:** A prime ideal of self lying over *x*. If degree is specified and no such ideal exists, raises a ValueError.

**EXAMPLES:**

```sage
x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)
sage: P2 = F.prime_above(2)
sage: P2
Fractional ideal (-t)
sage: 2 in P2
True
sage: P2.is_prime()
True
sage: P2.norm()
2
```

```sage
sage: P3 = F.prime_above(3)
sage: P3
Fractional ideal (t + 1)
sage: 3 in P3
True
sage: P3.is_prime()
True
sage: P3.norm()
3
```

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

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

Asking for a specific degree works:

```sage
sage: P5_1 = F.prime_above(5, degree=1)
sage: P5_1
Fractional ideal (-t^2 - 1)
sage: P5_1.residue_class_degree()
1
```
Algebraic Numbers and Number Fields, Release 9.6

```
sage: P5_2 = F.prime_above(5, degree=2)
sage: P5_2  # random
Fractional ideal (t^2 - 2*t - 1)
sage: P5_2.residue_class_degree()
2
```

Relative number fields are ok:

```
sage: G = F.extension(x^2 - 11, 'b')
sage: G.prime_above(7)
Fractional ideal (b + 2)
```

It doesn’t make sense to factor the ideal (0):

```
sage: F.prime_above(0)
Traceback (most recent call last):
  ...  
AttributeError: 'NumberFieldIdeal' object has no attribute 'prime_factors'
```

**prime_factors(x)**

Return a list of the prime ideals of self which divide the ideal generated by \( x \).

**OUTPUT:** list of prime ideals (a new list is returned each time this function is called)

**EXAMPLES:**

```
sage: 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:**

```
sage: x = ZZ['x'].gen()
sage: F.<t> = NumberField(x^3 - 2)

sage: P2s = F.primes_above(2)
sage: P2s  # random
[Fractional ideal (-t)]
sage: all(2 in P2 for P2 in P2s)
True
sage: all(P2.is_prime() for P2 in P2s)
```

(continues on next page)
The ideal (3) is totally ramified in F, so there is no degree 2 prime above 3:

\[
\text{sage: } \text{F.primes_above(3), degree=2} \\
[]
\]
\[
\text{sage: } [ \text{id.residue_class_degree() for id, _ in F.ideal(3).factor()} ] \\
[1]
\]

Asking for a specific degree works:

\[
\text{sage: } \text{F.primes_above(5, degree=1)} \\
\text{sage: } \text{F.primes_above(5, degree=2)} \\
[\text{Fractional ideal } (-t^2 - 1)]
\]
\[
\text{sage: } \text{P5_1 = P5_1s[0]; P5_1.residue_class_degree()} \\
1
\]
\[
\text{sage: } \text{P5_2 = P5_2s[0]; P5_2.residue_class_degree()} \\
2
\]

Works in relative extensions too:

\[
\text{sage: } \text{PQ.<X> = QQ[]} \\
\text{sage: } \text{F.<a, b> = NumberField([X^2 - 2, X^2 - 3])} \\
\text{sage: } \text{PF.<Y> = F[]} \\
\text{sage: } \text{K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)} \\
\text{sage: } \text{I = F.ideal(a + 2*b)} \\
\text{sage: } \text{P, Q = K.primes_above(I)} \\
\text{sage: } \text{K.ideal(I) == P^4*Q} \\
\text{True}
\]
\[
\text{sage: } \text{K.primes_above(I, degree=1) == [P]} \\
\text{True}
\]
\[
\text{sage: } \text{K.primes_above(I, degree=4) == [Q]} \\
\text{True}
\]

It doesn’t make sense to factor the ideal (0), so this raises an error:
**primes_of_bounded_norm** \((B)\)

Return a sorted list of all prime ideals with norm at most \(B\).

**INPUT:**

- \(B\) – a positive integer or real; upper bound on the norms of the primes generated.

**OUTPUT:**

A list of all prime ideals of this number field of norm at most \(B\), sorted by norm. Primes of the same norm are sorted using the comparison function for ideals, which is based on the Hermite Normal Form.

**Note:** See also `primes_of_bounded_norm_iter()` for an iterator version of this, but note that the iterator sorts the primes in order of underlying rational prime, not by norm.

**EXAMPLES:**

```python
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)]
```

```python
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)]

primes_of_degree_one_iter(num_integer_primes=10000, max_iterations=100)

Return an iterator yielding prime ideals of absolute degree one and small norm.

Warning: It is possible that there are no primes of $K$ of absolute degree one of small prime norm, and it possible that this algorithm will not find any primes of small norm.

See module sage.rings.number_field.small_primes_of_degree_one for details.

INPUT:

• num_integer_primes (default: 10000) - an integer. We try to find primes of absolute norm no greater than the num_integer_primes-th prime number. For example, if num_integer_primes is 2, the largest norm found will be 3, since the second prime is 3.

• max_iterations (default: 100) - an integer. We test max_iterations integers to find small primes before raising StopIteration.

EXAMPLES:

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 (2*z^3 - 3*z^2 + z - 2)]

sage: [ P.norm() for P in Ps ] # random
[11, 31, 41]

sage: [ P.residue_class_degree() for P in Ps ]
[1, 1, 1]

primes_of_degree_one_list(n, num_integer_primes=10000, max_iterations=100)

Return a list of $n$ prime ideals of absolute degree one and small norm.

Warning: It is possible that there are no primes of $K$ of absolute degree one of small prime norm, and it possible that this algorithm will not find any primes of small norm.

See module sage.rings.number_field.small_primes_of_degree_one for details.

INPUT:

• num_integer_primes (default: 10000) - an integer. We try to find primes of absolute norm no greater than the num_integer_primes-th prime number. For example, if num_integer_primes is 2, the largest norm found will be 3, since the second prime is 3.
max_iterations (default: 100) - an integer. We test max_iterations integers to find small primes before raising StopIteration.

EXAMPLES:

```
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: 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: 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()
```

(continues on next page)
We do not special-case cyclotomic fields, so we do not always get the most obvious primitive root of unity:

```
sage: K.<a> = CyclotomicField(3)
sage: z = K.primitive_root_of_unity(); z
 a + 1
sage: z.multiplicative_order()
 6
```

### Quadratic Defect

**quadratic_defect**(a, p, check=True)

Return the valuation of the quadratic defect of 𝑎 at 𝑝.

**INPUT:**

- 𝑎 – an element of self
- 𝑝 – a prime ideal
- check – (default: True); check if 𝑝 is prime

**ALGORITHM:**

This is an implementation of Algorithm 3.1.3 from [Kir2016]

**EXAMPLES:**

```
sage: 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**(num_bound=None, den_bound=None, integral_coefficients=False, distribution=None)

Return a random element of this number field.

**INPUT:**
• **num_bound** - Bound on numerator of the coefficients of the resulting element

• **den_bound** - Bound on denominators of the coefficients of the resulting element

• **integral_coefficients (default: False)** - If True, then the resulting element will have integral coefficients. This option overrides any value of den_bound.

• **distribution** - Distribution to use for the coefficients of the resulting element

**OUTPUT:**

• Element of this number field

**EXAMPLES:**

```python
sage: K.<j> = NumberField(x^8+1)
sage: K.random_element().parent() is K
True

sage: while K.random_element().list()[0] != 0:
    ....:    pass
sage: while K.random_element().list()[0] == 0:
    ....:    pass
sage: while not K.random_element().is_prime():
    ....:    pass
sage: while not K.random_element().is_prime(): # long time
    ....:    pass

sage: K.<a,b,c> = NumberField([x^2-2,x^2-3,x^2-5])
sage: K.random_element().parent() is K
True

sage: while K.random_element().is_prime():
    ....:    pass
sage: while not K.random_element().is_prime(): # long time
    ....:    pass

sage: K.<a> = NumberField(x^5-2)
sage: p = K.random_element(integral_coefficients=True)
sage: p.is_integral()
True

sage: while K.random_element().is_integral():
    ....:    pass
```

**real_embeddings** *(prec=53)*

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

If prec is 53 (the default), then the real double field is used; otherwise the arbitrary precision (but slow) real field is used. If you want embeddings into the 53-bit double precision, which is faster, use self.embeddings(RDF).

**Note:** This function uses finite precision real numbers. In functions that should output proven results, one could use self.embeddings(AA) instead.

**EXAMPLES:**
sage: K.<a> = NumberField(x^3 + 2)
sage: K.real_embeddings()
[  
    Ring morphism:
    From: Number Field in a with defining polynomial x^3 + 2
    To:   Real Field with 53 bits of precision
    Defn: a |--> -1.25992104989487
]
sage: K.real_embeddings(16)
[  
    Ring morphism:
    From: Number Field in a with defining polynomial x^3 + 2
    To:   Real Field with 16 bits of precision
    Defn: a |--> -1.260
]
sage: K.real_embeddings(100)
[  
    Ring morphism:
    From: Number Field in a with defining polynomial x^3 + 2
    To:   Real Field with 100 bits of precision
    Defn: a |--> -1.2599210498948731647672106073
]

As this is a numerical function, the number of embeddings may be incorrect if the precision is too low:

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(prec=None)
Return an LLL-reduced basis for the Minkowski-embedding of the maximal order of a number field.

INPUT:

  * 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:
sage: F.<t> = NumberField(x^6-7*x^4-x^3+11*x^2+x-1)
sage: F.maximal_order().basis()
[1/2*t^5 + 1/2*t^4 + 1/2*t^2 + 1/2, t, t^2, t^3, t^4, t^5]
sage: F.reduced_basis()
[-1, -1/2*t^5 + 1/2*t^4 + 3*t^3 - 3/2*t^2 - 4*t - 1/2, t, 1/2*t^5 + 1/2*t^4 - 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 - 5/2*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(prec=None)

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

INPUT:

- 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()])

sage: x = polygen(QQ)
sage: F.<alpha> = NumberField(x^4+x^2+712312*x+131001238)

(continues on next page)
\textbf{regulator}(\texttt{proof=\texttt{None}})

Return the regulator of this number field.

Note that PARI computes the regulator to higher precision than the Sage default.

INPUT:

\begin{itemize}
  \item \texttt{proof} - default: \texttt{True}, unless you set it otherwise.
\end{itemize}

EXAMPLES:

\begin{verbatim}
sage: NumberField(x^2-2, 'a').regulator()
0.881373587019543
sage: NumberField(x^4+x^3+x^2+x+1, 'a').regulator()
0.962423650119207
\end{verbatim}

\textbf{residue_field}(\texttt{prime}, \texttt{names=None}, \texttt{check=True})

Return the residue field of this number field at a given prime, ie $\mathcal{O}_K/\mathfrak{p}\mathcal{O}_K$.

INPUT:

\begin{itemize}
  \item \texttt{prime} - a prime ideal of the maximal order in this number field, or an element of the field which generates a principal prime ideal.
  \item \texttt{names} - the name of the variable in the residue field
  \item \texttt{check} - whether or not to check the primality of prime.
\end{itemize}

OUTPUT: The residue field at this prime.

EXAMPLES:

\begin{verbatim}
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^3+2*x^2+1, 'a')
sage: P = K.ideal(61).factor()[0][0]
sage: K.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)
sage: K.<i> = NumberField(x^2 + 1)
sage: K.residue_field(1+i)
Residue field of Fractional ideal (i + 1)
\end{verbatim}
roots_of_unity()

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

EXAMPLES:

```
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_generators(S, m, proof=True, orders=False)

Compute generators of the 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
- 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 $S$.

See also:

`NumberField_generic.selmer_space()`, which gives additional output when $m = p$ is prime: as well as generators, it gives an abstract vector space over $GF(p)$ isomorphic to $K(S, p)$ and maps implementing the isomorphism between this space and $K(S, p)$ as a subgroup of $K^*/(K^*)^p$.

EXAMPLES:

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

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

```
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 $m$:
```python
sage: K.<a> = QuadraticField(-5)
sage: P2 = K.ideal(2, -a+1)
sage: P3 = K.ideal(3, a+1)
sage: K.selmer_generators((,), 2, orders=True)
([-1, 2], [2, 2])
sage: K.selmer_generators((,), 4, orders=True)
([-1, 4], [2, 2])
sage: K.selmer_generators([P2], 2)
[2, -1]
sage: K.selmer_generators((P2,P3), 4)
[2, -a - 1, -1]
sage: K.selmer_generators((P2,P3), 4, orders=True)
([2, -a - 1, -1], [4, 4, 2])
sage: K.selmer_generators([P2], 3)
[2]
sage: K.selmer_generators([P2, P3], 3)
[2, -a - 1]
sage: K.selmer_generators([P2, P3, K.ideal(a)], 3)  # random signs
[2, a + 1, a]
sage: K.<a> = NumberField(polygen(QQ))
sage: K.selmer_generators([],5)
[]
sage: K.selmer_generators([K.prime_above(p) for p in [2,3,5]],2)
[2, 3, 5, -1]
sage: K.selmer_generators([K.prime_above(p) for p in [2,3,5]],6, orders=True)
([2, 3, 5, -1], [6, 6, 6, 2])
```

**selmer_group**

Deprecated: Use **selmer_generators** instead. See trac ticket #31345 for details.

**selmer_group_iterator**

Return an iterator through elements of the finite group \( K(S, m) \).

**INPUT:**

- \( S \) – a set of primes of \( self \)
- \( m \) – a positive integer
- \( proof \) – if False, assume the GRH in computing the class group

**OUTPUT:**

An iterator yielding the distinct elements of \( K(S, m) \). See the docstring for **NumberField_generic.selmer_generators** for more information.

**EXAMPLES:**

```python
sage: K.<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))
```
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]
```

The `selmer_group_iterator` function returns a list of elements of the Selmer group of a number field with respect to a given set of primes and a given prime number. The elements are represented as tuples, with each tuple containing a generator of the Selmer group.

The `selmer_space` function computes the Selmer space of a number field with respect to a set of primes and a prime number.

```python
def selmer_space(S, p, proof=None):
    """Compute the group \( K(S, p) \) as a vector space with maps to and from \( K^* \).
    """
    if proof:
        return Kselmer_group_iterator(S, p)
    else:
        return Kselmer_group_iterator(S, p, proof)
```

The input parameters of the `selmer_space` function are:

- **S**: A list of prime ideals of the ring of integers of \( K \).
- **p**: A prime number.
- **proof**: If `proof=True`, use the standard proof of the GRH.

The output of the `selmer_space` function is a tuple containing:

- `KSp`:
  - A vector space over \( GF(p) \) isomorphic to \( K(S, p) \).
- `KSp_gens`:
  - A list of generators of \( K(S, p) \).
- `from_KSp`:
  - A function from \( KSp \) to \( K^* \) implementing the isomorphism from the abstract vector space to \( K(S, p) \) as a subgroup of \( K^*/(K^*)^p \).
- `to_KSp`:
  - A partial function from \( K^* \) to \( KSp \), defined on elements \( a \) whose image in \( K^*/(K^*)^p \) lies in \( K(S, p) \), mapping them via the inverse isomorphism to the abstract vector space \( KSp \).

The Selmer space is the finite subgroup of \( K^*/(K^*)^p \) consisting of elements whose valuation at all primes not in \( S \) is a multiple of \( p \). It contains the subgroup of those \( a \in K^* \) such that \( K(\sqrt[p]{a})/K \) is unramified at all primes of \( K \) outside of \( S \), but may contain it properly when not all primes dividing \( p \) are in \( S \).

**Examples:**

A real quadratic field with class number 2, where the fundamental unit is a generator, and the class group provides another generator when \( p = 2 \):

```python
sage: K.<a> = QuadraticField(-5)
sage: K.class_number()
2
sage: P2 = K.ideal(2, -a+1)
sage: P3 = K.ideal(3, a+1)
sage: P5 = K.ideal(a)
sage: KS2, gens, fromKS2, toKS2 = K.selmer_space([P2, P3, P5], 2)
sage: KS2
Vector space of dimension 4 over Finite Field of size 2
sage: gens
[a + 1, a, 2, -1]
```
Each generator must have even valuation at primes not in \( S \):

```
sage: [K.ideal(g).factor() for g in gens]
[(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1)),
 Fractional ideal (-a),
 (Fractional ideal (2, a + 1))^2,
 1]
```

```
sage: toKS2(10)
(0, 0, 1, 1)
sage: fromKS2([0,0,1,1])
-2
```

```
sage: K(10/(-2)).is_square()
True
```

```
sage: KS3, gens, fromKS3, toKS3 = K.selmer_space([P2, P3, P5], 3)
sage: KS3
Vector space of dimension 3 over Finite Field of size 3
sage: gens
[1/2, 1/4*a + 1/4, a]
```

An example to show that the group \( K(S, 2) \) may be strictly larger than the group of elements giving extensions unramified outside \( S \). In this case, with \( K \) of class number 2 and \( S \) empty, there is only one quadratic extension of \( K \) unramified outside \( S \), the Hilbert Class Field \( K(\sqrt{-1}) \):

```
sage: K.<a> = QuadraticField(-5)
sage: KS2, gens, fromKS2, toKS2 = K.selmer_space([], 2)
sage: KS2
Vector space of dimension 2 over Finite Field of size 2
sage: gens
[2, -1]
sage: for v in KS2:
    ....:     if not v:
    ....:       continue
    ....:     a = fromKS2(v)
    ....:     print((a,K.extension(x^2-a, 'roota').relative_discriminant().˓
    ....:          factor()))
(2, (Fractional ideal (2, a + 1))^4)
(-1, 1)
(-2, (Fractional ideal (2, a + 1))^4)
```

```
sage: K.hilbert_class_field('b')
Number Field in b with defining polynomial x^2 + 1 over its base field
```

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

EXAMPLES:

```
sage: 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
```

**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()**
Return the embedding of this field into the complex numbers which has been specified.

Fields created with the QuadraticField or CyclotomicField constructors come with an implicit embedding. To get one of these fields without the embedding, use the generic NumberField constructor.

**EXAMPLES:**
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 with a = -1.
  → 0.9041608591349217
  To:  Real Lazy Field
  Defn: a -> -1.9041608591349217
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 with a = 0.
  → 0.9520804295674617 + 1.311248044077123?*I
  To:  Complex Lazy Field
  Defn: a -> 0.9520804295674617 + 1.311248044077123?*I
```

This function only returns complex embeddings:

```
sage: K.<a> = NumberField(x^2-2, embedding=Qp(7)(2).sqrt())
```

```mermaid
mermaid
diagram
```
sage: K.specified_complex_embedding() is None
True
sage: K.coerce_embedding()
Generic morphism:
  From: Number Field in a with defining polynomial x^2 - 2 with a = 3 + 7 + 2*7^2 + 6*7^3 + 7^4 + 2*7^5 + 7^6 + 2*7^7 + 4*7^8 + 6*7^9 + 6*7^10 +...
  → 2*7^11 + 7^12 + 7^13 + 2*7^15 + 7^16 + 7^17 + 4*7^18 + 6*7^19 + O(7^20)
  To:  7-adic Field with capped relative precision 20
  Defn: a -> 3 + 7 + 2*7^2 + 6*7^3 + 7^4 + 2*7^5 + 7^6 + 2*7^7 + 4*7^8 + 6*7^9 + 6*7^10 + 2*7^11 + 7^12 + 7^13 + 2*7^15 + 7^16 + 7^17 + 4*7^18 + 6*7^19 + 0(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.<z> = NumberField(x^2 + 3)
sage: L.<a> = K.absolute_field(); L
Number Field in a with defining polynomial x^2 + 3
sage: L.structure()
(Isomorphism given by variable name change map:
  From: Number Field in a with defining polynomial x^2 + 3
  To: Number Field in z with defining polynomial x^2 + 3,
Isomorphism given by variable name change map:
  From: Number Field in z with defining polynomial x^2 + 3
  To: Number Field in a with defining polynomial x^2 + 3)
```

```
sage: K.<a> = QuadraticField(-3)
sage: R.<y> = K[]
sage: D.<x0> = K.extension(y)
sage: D_abs.<y0> = D.absolute_field()
sage: D_abs.structure()[0](y0)
-a
```

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: 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)
```

```
sage: H
Number Field in b with defining polynomial x^2 - 3 with b = a^2
sage: from_H(b)
a^2
```

```
sage: from_H
Ring morphism:
  From: Number Field in b with defining polynomial x^2 - 3 with b = a^2
  To: Number Field in a with defining polynomial x^4 - 3
  Defn: b |--> a^2
```

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

```python
sage: K.<z> = CyclotomicField(5)
sage: L, K_from_L = K.subfield(z-z^2-z^3+z^4)
sage: L
Number Field in z0 with defining polynomial x^4 - 5 with z0 = 2.236067977499790?
```

```python
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

clf_from_L = L.coerce_embedding(); clf_from_L
Generic morphism:
  From: Number Field in z0 with defining polynomial x^4 - 5 with z0 = 2.236067977499790?
  To:   Complex Lazy Field
  Defn: z0 -> 2.236067977499790?
```

Check transitivity:

```python
clf_from_L(L.gen())
2.236067977499790?
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`:

```python
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 with b = a^2
```

```python
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.

```
subfield_from_elements(alpha, name=None, polred=True, threshold=None)
Return the subfield generated by the elements alpha.
```

If the generated subfield by the elements `alpha` is either the rational field or the complete number field, the field returned is respectively `QQ` or `self`.

INPUT:

- `alpha` - list of elements in this number field
- `name` - a name for the generator of the new number field
- `polred` (boolean, default True) - whether to optimize the generator of the newly created field
• threshold (positive number, default None) - threshold to be passed to the do_polred function

OUTPUT: a triple (field, beta, hom) where
• field - a subfield of this number field
• beta - a list of elements of field corresponding to alpha
• hom - inclusion homomorphism from field to self

EXAMPLES:

```python
sage: x = polygen(QQ)
sage: poly = x^4 - 4*x^2 + 1
sage: emb = AA.polynomial_root(poly, RIF(0.51, 0.52))
sage: K.<a> = NumberField(poly, embedding=emb)
sage: sqrt2 = -a^3 + 3*a
sage: sqrt3 = -a^2 + 2
sage: assert sqrt2 ** 2 == 2 and sqrt3 ** 2 == 3
sage: L, elts, phi = K.subfield_from_elements([sqrt2, 1 - sqrt2/3])
sage: L
Number Field in a0 with defining polynomial x^2 - 2 with a0 = 1.414213562373095?
sage: elts
[a0, -1/3*a0 + 1]
sage: phi
Ring morphism:
  From: Number Field in a0 with defining polynomial x^2 - 2 with a0 = 1.414213562373095?
  To:   Number Field in a with defining polynomial x^4 - 4*x^2 + 1 with a = 0.5176380902050415?
  Defn: a0 |--> -a^3 + 3*a
sage: assert phi(elts[0]) == sqrt2
sage: assert phi(elts[1]) == 1 - sqrt2/3

sage: L, elts, phi = K.subfield_from_elements([1, sqrt3])
sage: assert phi(elts[0]) == 1
sage: assert phi(elts[1]) == sqrt3

sage: L, elts, phi = K.subfield_from_elements([sqrt2, sqrt3])
sage: phi
Identity endomorphism of Number Field in a with defining polynomial x^4 - 4*x^2 + 1 with a = 0.5176380902050415?
```

`trace_dual_basis(b)`
Compute the dual basis of a basis of self with respect to the trace pairing.

EXAMPLES:

```python
sage: K.<a> = NumberField(x^3 + x + 1)
sage: b = [1, 2*a, 3*a^2]
sage: T = K.trace_dual_basis(b); T
sage: [(b[i]*T[j]).trace() for i in range(3) for j in range(3)]
[1, 0, 0, 1, 0, 0, 0, 1]
```

1.1. Number Fields 93
**trace_pairing**($\nu$)
Return the matrix of the trace pairing on the elements of the list $\nu$.

**EXAMPLES:**

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

**uniformizer**($P$, **others**='positive')
Return 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> = 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: K.uniformizer(P, 'negative')
1/2*a + 1/2
sage: K.ideal(pi).factor()  
(Fractional ideal (2, a + 1))^-1 * (Fractional ideal (3, a + 1))
```

```
sage: K = CyclotomicField(9)
sage: Plist=K.ideal(17).prime_factors()
sage: pilist = [K.uniformizer(P) for P in Plist]
[True, True, True]
sage: [pi.is_integral() for pi in pilist]
[True, True, True]
sage: [pi.valuation(P) for pi,P in zip(pilist,Plist)]
[1, 1, 1]
```

```
sage: K.<t> = NumberField(x^4 - x^3 - 3*x^2 - x + 1)
sage: [K.uniformizer(P) for P,e in factor(K.ideal(2))]
(continues on next page)
\begin{verbatim}

sage: [K.uniformizer(P) for P, e in factor(K.ideal(3))]
[t - 1]
sage: [K.uniformizer(P) for P, e in factor(K.ideal(5))]
[t^2 - t + 1, t + 2, t - 2]
sage: [K.uniformizer(P) for P, e in factor(K.ideal(7))]
# representation varies, not tested
[t^2 + 3*t + 1]
sage: [K.uniformizer(P) for P, e in factor(K.ideal(67))]
[t + 23, t + 26, t - 32, t - 18]

ALGORITHM:

Use PARI. More precisely, use the second component of \texttt{pari:idealprimedec} in the “positive” case.
Use \texttt{pari:idealappr} with exponent of -1 and invert the result in the “negative” case.

\textbf{unit\_group}(\textit{proof}=\texttt{None})

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

ALGORITHM: Uses PARI’s \texttt{pari:bnfinit} command.

INPUT:

- \textit{proof} (bool, default True) flag passed to \texttt{pari}.

\textbf{Note:} The group is cached.

See also:

\texttt{units()} \texttt{S\_unit\_group()} \texttt{S\_units()}

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)

\end{verbatim}
units(proof=None)
Return generators for the unit group modulo torsion.

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

**INPUT:**
- proof (bool, default True) flag passed to pari.

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

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

**EXAMPLES:**

```python
sage: x = polygen(QQ)
sage: A = x^4 - 10*x^3 + 20*5*x^2 - 15*5^2*x + 11*5^3
sage: K = NumberField(A, 'a')
sage: K.units()
(-1/275*a^3 - 4/55*a^2 + 5/11*a - 3,)
```

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):

```python
sage: K = NumberField(x^17 + 3, 'a')
sage: K.units(proof=True) # takes forever, not tested...
sage: K.units(proof=False) # result not independently verified
(-a^9 - a + 1, -a^16 + a^15 - a^14 + a^12 - a^11 + a^10 - a^9 + 2*a^6, a^15 + a^14 + 2*a^10 - 2*a^12 + 2*a^9 - 3*a^8 + 3*a^7 + 3*a^6 + 3*a^5 + 3*a^4 + 3*a^3 + 2*a^2 + 2*a - 1, 2*a^14 + a^13 + 3*a^12 - 2*a^10 + a^9, -3*a^8 + 2*a^7 - a^5 + 2*a^3 - a^2 - 3*a + 1, -a^16 - a^15 - a^14 + a^13 - a^12 + a^11 - a^10 + a^9 + a^8 - a^7 + 2*a^6 - a^4 + 3*a^3 - 2*a^2 + 2*a - 1, -a^14 + 2*a^10 + a^8 - 2*a^7 - 2*a^6 + 2*a^3 - a^2 + 2*a - 2)
```
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
```

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))
```
Note that you get the same valuation, even if you write down the pseudo-valuation differently:

```python
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:

```python
sage: v = GaussValuation(R, QQ.valuation(5)).augmentation(x + 3, 1)
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:

```python
sage: K.valuation(K.fractional_ideal(a + 1))
2-adic valuation
```

See also:

`Order.valuation()`, `pAdicGeneric.valuation()`

**zeta**(n=2, all=False)

Return one, or a list of all, primitive $n$-th root of unity in this field.

**INPUT:**

- n – positive integer
- all – boolean. If `False` (default), return a primitive $n$-th root of unity in this field, or raise a `ValueError` exception if there are none. If `True`, return a list of all primitive $n$-th roots of unity in this field (possibly empty).

**Note:** To obtain the maximal order of a root of unity in this field, use `number_of_roots_of_unity()`.

**Note:** We do not create the full unit group since that can be expensive, but we do use it if it is already known.

**EXAMPLES:**

```python
sage: K.<z> = NumberField(x^2 + 3)
sage: K.zeta(1)
1
sage: K.zeta(2)
-1
sage: K.zeta(2, all=True)
[-1]
```
Number fields defined by non-monic and non-integral polynomials are supported (trac ticket #252):

```sage
sage: r.<x> = QQ[]
sage: K.<a> = NumberField(x^2+1)
sage: K.<b> = NumberField(1/2*x^2 + 1/6)
sage: K.<a> = NumberField(1/2*x^2 + 1/6)
sage: K.<b> = NumberField(1/2*x^2 + 1/6)
```

**zeta_coefficients**

Compute the first \(n\) coefficients of the Dedekind zeta function of this field as a Dirichlet series.

**EXAMPLES:**

```sage
sage: x = QQ['x'].0
sage: NumberField(x^2+1, 'a').zeta_coefficients(10)
[1, 1, 0, 1, 2, 0, 0, 1, 1, 2]
```

**zeta_order**

Return the number of roots of unity in this field.

**Note:** We do not create the full unit group since that can be expensive, but we do use it if it is already known.

**EXAMPLES:**

```sage
sage: F.<alpha> = NumberField(x**22+3)
sage: F.<alpha> = NumberField(x**22+3)
sage: F.<alpha> = NumberField(x**2-7)
sage: F.<alpha> = NumberField(x**2-7)
sage: F.zeta_order()
sage: F.zeta_order()
sage: F.zeta_order()
sage: F.zeta_order()
6
2
```
sage.rings.number_field.number_field.NumberField_generic_v1(poly, name, latex_name, canonical_embedding=None)

Used for unpickling old pickles.

EXAMPLES:

```
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_quadratic(polynomial, name=None, 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_absolute, sage.rings.abc.
NumberField_quadratic

Create a quadratic extension of the rational field.

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

EXAMPLES:

```
sage: QuadraticField(3, 'a')
Number Field in a with defining polynomial x^2 - 3 with a = 1.732050807568878?
sage: QuadraticField(-4, 'b')
Number Field in b with defining polynomial x^2 + 4 with b = 2*I
```

class_number(proof=None)

Return the size of the class group of self.

INPUT:

- proof – boolean (default: True, unless you called proof.number_field() and set it otherwise). If proof is False (not the default!), and the discriminant of the field is negative, then the following warning from the PARI manual applies:

```
Warning: For $D < 0$, this function may give incorrect results when the class group has a low exponent (has many cyclic factors), because implementing Shank’s method in full generality slows it down immensely.
```

EXAMPLES:

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

INPUT:
• v (optional) - list of 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(names)
Return the Hilbert class field of this quadratic field as a relative extension of this field.

Note: For the polynomial that defines this field as a relative extension, see the
hilbert_class_field_defining_polynomial command, which is vastly faster than this com-
mand, 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(name=x)
Return a polynomial over Q whose roots generate the Hilbert class field of this quadratic field as an exten-
sion 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*b^2 - 5151296875*b + 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: 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

sage.rings.number_field.number_field.QuadraticField($D$, name='a', check=True, embedding=True, latex_name='sqrt', **args)

Return a quadratic field obtained by adjoining a square root of $D$ to the rational numbers, where $D$ is not a perfect square.

INPUT:

- $D$ - a rational number
- name - variable name (default: 'a')
- check - bool (default: True)
- embedding - bool or square root of $D$ in an ambient field (default: True)
- latex_name - latex variable name (default: sqrt($D$))

OUTPUT: A number field defined by a quadratic polynomial. Unless otherwise specified, it has an embedding into $\mathbb{R}$ or $\mathbb{C}$ by sending the generator to the positive or upper-half-plane root.

EXAMPLES:

```
sage: QuadraticField(3, 'a')
Number Field in a with defining polynomial x^2 - 3 with a = 1.732050807568878?
sage: K.<theta> = QuadraticField(3); K
Number Field in theta with defining polynomial x^2 - 3 with theta = 1.732050807568878?

sage: RR(theta)
1.73205080756888
sage: QuadraticField(9, 'a')
Traceback (most recent call last):
... Value Error: D must not be a perfect square.
sage: QuadraticField(9, 'a', check=False)
Number Field in a with defining polynomial x^2 - 9 with a = 3
```

Quadratic number fields derive from general number fields.

```
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:

```
sage: QuadraticField(-11, 'a') is QuadraticField(-11, 'a')
True
```

By default, quadratic fields come with a nice latex representation:

1.1. Number Fields
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: list(K.latex_variable_names())
['\sqrt{-7}']

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 == K2
True
sage: K1 == K3
True
sage: K1
Number Field in x with defining polynomial x^2 - 5 with x = 2.236067977499790?

Note that, in presence of two different names for the generator, the name given by the parser takes precedence:

sage: K4.<y> = QuadraticField(5, 'x'); K4
Number Field in y with defining polynomial x^2 - 5 with y = 2.236067977499790?
sage: K1 == K4
False

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
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.

This function is deprecated. Use `isinstance()` with `NumberField_cyclotomic` instead.

**EXAMPLES:**

```
sage: from sage.rings.number_field.number_field import is_CyclotomicField
sage: is_CyclotomicField(NumberField(x^2 + 1, 'zeta4'))
doctest:warning...
DeprecationWarning: is_CyclotomicField is deprecated; use isinstance(..., sage.rings.abc.NumberField_cyclotomic instead
See https://trac.sagemath.org/32660 for details.
False
sage: is_CyclotomicField(CyclotomicField(4))
True
sage: is_CyclotomicField(CyclotomicField(1))
True
sage: is_CyclotomicField(QQ)
False
sage: is_CyclotomicField(7)
False
```

Return whether `codomain` is a valid codomain for a number field homset. This is used by `NumberField._Hom_` to determine whether the created homsets should be a `sage.rings.number_field.homset.NumberFieldHomset`.

**EXAMPLES:**

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

```
sage: from sage.rings.number_field.number_field import is_NumberFieldHomsetCodomain
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: is_NumberFieldHomsetCodomain(gap.Rationals)
False
```

Return True if \( x \) is of the quadratic number field type.

This function is deprecated. Use `isinstance()` with `NumberField_quadratic` instead.

1.1. Number Fields
EXAMPLES:

```python
sage: from sage.rings.number_field.number_field import is_QuadraticField
sage: is_QuadraticField(QuadraticField(5, 'a'))
doctest:warning...
DeprecationWarning: is_QuadraticField is deprecated; use isinstance(..., sage.rings.abc.NumberField_quadratic instead
See https://trac.sagemath.org/32660 for details.
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:

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

`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
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.

Return t if t is not None, otherwise return the system-wide proof-flag for number fields (default: True).

EXAMPLES:

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")
'bananap'

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

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:

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.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

sage.rings.number_field.number_field.refine_embedding(e, prec=None)
Given an embedding from a number field to either R or C, returns an equivalent embedding with higher precision.

INPUT:

* e - an embedding of a number field into either RR or CC (with some precision)
- **prec** - (default None) the desired precision; if None, current precision is doubled; if Infinity, the equivalent embedding into either QQbar or AA is returned.

**EXAMPLES:**

```python
define_embedding()
```
sage: K = CyclotomicField(3)
sage: e10 = K.complex_embedding(10)
sage: e10 codomain().precision()
10
sage: e25 = refine_embedding(e10, prec=25)
sage: e25 codomain().precision()
25

An example where we extend a real embedding into AA:

```python
define_embedding()
```
sage: K.<a> = NumberField(x^3 - 2)
sage: K signature()
(1, 1)
sage: e = K.embeddings(RR)[0]; e
Ring morphism:
From: Number Field in a with defining polynomial x^3 - 2
To: Real Field with 53 bits of precision
Defn: a |--> 1.25992104989487
sage: e = refine_embedding(e, Infinity); e
Ring morphism:
From: Number Field in a with defining polynomial x^3 - 2
To: Algebraic Real Field
Defn: a |--> 1.259921049894873?

Now we can obtain arbitrary precision values with no trouble:

```python
define_embedding()
```
sage: RealField(150)(e(a))
1.2599210498948731647672106072782283505702515
sage: ^3
2.00000000000000000000000000000000000000000000
sage: RealField(200)(e(a^2 - 3*a + 7))
4.8076379022835799804500738174376232086807389337953290695624

Complex embeddings can be extended into QQbar:

```python
define_embedding()
```
sage: e = K.embeddings(CC)[0]; e
Ring morphism:
From: Number Field in a with defining polynomial x^3 - 2
To: Complex Field with 53 bits of precision
Defn: a |--> -0.62996052494743... - 1.0911236359717214*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? - 1.0911236359717227?*I
sage: ComplexField(200)(e(a))
-0.6299605249474365828360530639114752851257323507539904099 - 1.
-0911236359717214035600726141898088813258733387403009407036*I
sage: e(a)^3
```

(continues on next page)
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 |--> 0.62348980185873353052500488400423981063227473089640210536554943909685365245648728457942507...
      + 0.78183148246802980870844526674057750232334518708687528980634958045091731633936441700868007*I
sage: refine_embedding(x, infinity)
Ring morphism:
  From: Cyclotomic Field of order 7 and degree 6
  To: Algebraic Field
  Defn: zeta7 |--> 0.6234898018587335? + 0.7818314824680299?*I
```

When the old embedding is into the real lazy field, then only real embeddings should be considered. See trac ticket #17495:

```
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^3 + x - 1, embedding=0.68)
sage: from sage.rings.number_field.number_field import refine_embedding
sage: refine_embedding(K.specified_complex_embedding(), 100)
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + x - 1 with a = 0.
      + 0.623278038280193?
      + 0.623278038280193?
  To: Real Field with 100 bits of precision
  Defn: a |---> 0.6232780382801937948373971
sage: refine_embedding(K.specified_complex_embedding(), Infinity)
Ring morphism:
  From: Number Field in a with defining polynomial x^3 + x - 1 with a = 0.
      + 0.623278038280193?
      + 0.623278038280193?
  To: Algebraic Real Field
  Defn: a |---> 0.6232780382801937948373971
```
1.2 Base class for all number fields

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

degree()
    Return the degree of this number field.

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

discriminant()
Return the discriminant of this number field.

EXAMPLES:

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

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

EXAMPLES:

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: 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()

OUTPUT:

symbolic expression or Rational

EXAMPLES:

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

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

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

1.2. Base class for all number fields
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 \( \mathbb{Q}[\sqrt{2} + \sqrt{3}] \):

```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
```

### `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 \((r_1, r_2)\), 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^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:

```
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:

```
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 Rüth (2014-04): absolute number fields are unique parents

This example follows one in the Magma reference manual:

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

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

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

(continues on next page)
sage: (sqrt2 + cuberoot3)^5
(20*cuberoot3^2 + 15*cuberoot3 + 4)*sqrt2 + 3*cuberoot3^2 + 20*cuberoot3 + 60
sage: cuberoot2 + cuberoot3
cuberoot3 + cuberoot2
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).

sage.rings.number_field.number_field_rel.NumberField_extension_v1(base_field, polynomial, name, latex_name, canonical_embedding=None)

Used for unpickling old pickles.

EXAMPLES:

sage: from sage.rings.number_field.number_field_rel import NumberField_relative_v1
sage: R.<x> = CyclotomicField(3)[]

sage: NumberField_relative_v1(CyclotomicField(3), x^2 + 7, 'a', 'a')
Number Field in a with defining polynomial x^2 + 7 over its base field

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 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:
absolute_base_field()

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

EXAMPLES:

```
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:

```
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.base_field()
Number Field in a1 with defining polynomial x^3 - 2 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 \( L\text{.relative}\_different() \).

EXAMPLES:

```
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=None)

Return the absolute discriminant of this relative number field or if \( v \) is specified, the determinant of the trace pairing on the elements of the list \( v \).

INPUT:

- \( v \) (optional) – list of element of this relative number field.

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

EXAMPLES:

```
sage: K.<i> = NumberField(x^2 + 1)
sage: t = K['t'].gen()
sage: L.<b> = K.extension(t^4 - i)
sage: L.absolute_discriminant()
```

(continues on next page)
absolute_field(names)
Return self as an absolute number field.

INPUT:

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

OUTPUT:
An absolute number field $K$ that is isomorphic to this field.
Also, K.structure() returns from_K and to_K, where from_K is an isomorphism from $K$ to self and to_K is an isomorphism from self to $K$.

EXAMPLES:

sage: 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])
```python
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:**

```python
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:

```python
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:**

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

### absolute_vector_space(base=None, *args, **kwds)  
Return vector space over \( \mathbb{Q} \) of self and isomorphisms from the vector space to self and in the other direction.

**EXAMPLES:**
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
] (continues on next page)
Relative number field endomorphism of Number Field in \( b \) with defining polynomial \( x^2 + x + 50 \) over its base field
Defn: \( b \mapsto -b - 1 \)
\( a \mapsto a \)

```
sage: rho, tau = L.automorphisms()
sage: tau(a) == a
True
sage: tau(b)
-b - 1
```

```
PQ.<X> = QQ[]
F.<a, b> = NumberField([X^2 - 2, X^2 - 3])
PF.<Y> = F[]
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 \( Y^2 + (-2b - 3)a - 2b - 6 \) over its base field
Defn: \( c \mapsto c \)
\( a \mapsto a \)
\( b \mapsto b \),
```

```
Relative number field endomorphism of Number Field in \( c \) with defining polynomial \( Y^2 + (-2b - 3)a - 2b - 6 \) over its base field
Defn: \( c \mapsto -c \)
\( a \mapsto a \)
\( b \mapsto b \)
```

```
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 a 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:**

```
1.3. Relative Number Fields
119
```
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

cchange_names(names)
Return relative number field isomorphic to self but with the given generator names.

INPUT:

• names – number of names should be at most the number of generators of self, i.e., the number of steps
  in the tower of relative fields.

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

EXAMPLES:

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[X]
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:

(continues on next page)
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, self_into_F, other_into_F, k) such that self_into_F maps self into F, other_into_F maps 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:**

```sage
sage: K.<a, b> = NumberField([x^2 + 5, x^2 - 2])
sage: L.<c, d> = NumberField([x^2 + 5, x^2 - 3])
sage: K.composite_fields(L, 'e')
([Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600],)
sage: K.composite_fields(L, 'e', both_maps=True)
[[Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600,
  Relative number field morphism:
  From: Number Field in a with defining polynomial x^2 + 5 over its base field
  To: Number Field in e with defining polynomial x^8 - 24*x^6 + 464*x^4 + 3840*x^2 + 25600
  Defn: a |---> -9/66560*e^7 + 11/4160*e^5 - 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]
```

**defining_polynomial()**

Return the defining polynomial of this relative number field.

This is exactly the same as **relative_polynomial()**.
EXAMPLES:

```
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:

```
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.disc()
Traceback (most recent call last):
... NotImplementedError: For a relative number field you must use relative␣
˓→discriminant or absolute_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()
```

(continues on next page)
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
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
```

free_module($base=None$, $basis=None$, $map=True$)

Return a vector space over a specified subfield that is isomorphic to this number field, together with the isomorphisms in each direction.

INPUT:

• $base$ – a subfield
  • $basis$ – (optional) a list of elements giving a basis over the subfield
  • $map$ – (default True) whether to return isomorphisms to and from the vector space

EXAMPLES:

```sage
sage: K.<a,b,c> = NumberField([x^2 + 2, x^3 + 2, x^3 + 3]); K
Number Field in a with defining polynomial x^2 + 2 over its base field
sage: V, from_V, to_V = K.free_module()
```

(continues on next page)
sage: to_V(K.0)
(0, 1)
sage: W, from_W, to_W = K.free_module(base=QQ)
sage: w = to_W(K.0); len(w)
18
sage: w[0]
-127917622658689792301282/48787705559800061938765

\textbf{galois\_closure}(\textit{names=\text{None}})

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

\textbf{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 $x^{16} + 16x^{14} + 28x^{12} + 784x^{10} + \ldots$
$+ 19846x^8 - 595280x^6 + 2744476x^4 + 3212848x^2 + 29953729$

\textbf{gen}(\textit{n=0})

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

\textbf{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.gens()
(a, b)
sage: K.gen(0)
a

\textbf{gens}()

Return the generators of this relative number field.

\textbf{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.gens()
(a, b)

\textbf{is\_CM\_extension}()

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

\textbf{EXAMPLES:}

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)
A CM field $K$ such that $K/F$ is not a CM extension

```
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:**

```
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:**

```
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:**

```
sage: k.<a> = NumberField([x^3 - 2, x^2 + x + 1])
sage: k.is_galois()
Traceback (most recent call last):
...
NotImplementedError: For a relative number field L you must use either L.is_galois_relative() or L.is_galois_absolute() as appropriate
```
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); 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, \( other \) can be a relative extension \( M/K' \) with \( base_isom \) 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 \( base_isom \) be an isomorphism from self’s base field to other’s base field:
sage: Kcyc.<zeta9> = CyclotomicField(9)
sage: Rcyc.<zcyc> = PolynomialRing(Kcyc)
sage: phi1 = K.hom([zeta9])
sage: mlcyc = phi1(m1)
sage: L1cyc = Kcyc.extension(zcyc^2 - mlcyc, 'b1cyc')
sage: L1.is_isomorphic_relative(L1cyc, base_isom=phi1)
True
sage: L2.is_isomorphic_relative(L1cyc, base_isom=phi1)
False
sage: phi2 = K.hom([phi1((gamma^(-2))(z9))])
sage: L1.is_isomorphic_relative(L1cyc, base_isom=phi2)
False
sage: L2.is_isomorphic_relative(L1cyc, base_isom=phi2)
True

Omitting 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 using the base_isom parameter.
```

The parameter base_isom can also be used to check if the relative extensions are Galois conjugate:

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

Ring endomorphism of Number Field in z9 with defining polynomial x^6 + x^3 + 1
  Defn: z9 |--> z9^4
```

**lift_to_base(element)**

Lift an element of this extension into the base field if possible, or raise a ValueError if it is not possible.

**EXAMPLES:**

```python
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
```

**ngens()**

Return the number of generators 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.gens()
(a, b)
sage: K.ngens()
2
```

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

EXAMPLES:

```
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:

```
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 \( \mathbb{Z} \):

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

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

```
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())
<class 'cypari2.gen.Gen'>
```

```python
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())
<class 'cypari2.gen.Gen'>
```

### pari_relative_polynomial()

Return the PARI relative polynomial associated to this number field.

This is always a polynomial in x and y, suitable for PARI's rnfinit function. Notice that if this is a relative extension of a relative extension, the base field is the absolute base field.

**EXAMPLES:**

```python
sage: 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)
```

### pari_rnf()

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

**EXAMPLES:**

```python
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], ...]
```

### 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 prec is not None, we simply return places into RealField(prec) and ComplexField(prec) (or RDF, CDF if prec=53).

1.3. Relative Number Fields
There is an optional flag `all_complex`, which defaults to `False`. If `all_complex` is `True`, then the real embeddings are returned as embeddings into CIF instead of RIF.

**EXAMPLES:**

```sage
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 |--> -1.213411662762229634132131377426,
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 |--> -1.213411662762229634130492421800,
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 |--> -2.2360679774997...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 |--> 2.2360679774997... - 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
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
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:**
relative_discriminant()  
Return the relative discriminant of this extension \( L/K \) as an ideal of \( K \). If you want the (rational) discriminant of \( L/Q \), use e.g. \( L.absol...ideal(256)\)

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

EXAMPLES:

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.:  

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

relative_vector_space(base=None, *args, **kwds)  
Return vector space over the base field of self and isomorphisms from the vector space to self and in the other direction.

EXAMPLES:

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)
relativize(alpha, names)
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
• names – name of generator for output field \( K \).

OUTPUT: \( K \) – a relative number field
Also, \( K.\text{structure}() \) returns \( \text{from}_K \) and \( \text{to}_K \), where \( \text{from}_K \) is an isomorphism from \( K \) to self and \( \text{to}_K \) 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.<z,w> = K.relativize(a^2)
sage: z^2
z^2
sage: w^2
-3
sage: L
Number Field in z with defining polynomial x^4 + (-2*w + 4)*x^2 + 4*w + 1 over its base field
sage: L.base_field()
Number Field in w with defining polynomial x^2 + 3
```

Now suppose we have \( K \) below \( L \) below \( M \):

```
sage: M = NumberField(x^8 + 2, 'a'); M
Number Field in a with defining polynomial x^8 + 2
sage: L, L_into_M, _ = M.subfields(4)[0]; L
Number Field in a0 with defining polynomial x^4 + 2
sage: K, K_into_L, _ = L.subfields(2)[0]; K
Number Field in a0_0 with defining polynomial x^2 + 2
sage: K_into_M = L_into_M * K_into_L
sage: L_over_K = L.relativize(K_into_L, 'c'); L_over_K
Number Field in c with defining polynomial x^2 + a0_0 over its base field
sage: L_over_K_to_L, L_to_L_over_K = L_over_K.structure()
sage: M_over_L_over_K = M.relativize(L_into_M * L_over_K_to_L, 'd'); M_over_L_over_K
Number Field in d with defining polynomial x^2 + c over its base field
```
Test relativizing a degree 6 field over its degree 2 and degree 3 subfields, using both an explicit element:

```python
sage: K.<a> = NumberField(x^6 + 2); K
Number Field in a with defining polynomial x^6 + 2
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 K2 (not the generator):

```python
sage: L = K.relativize(K3_into_K, '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()
```

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
```

### 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])
```

```python
sage: len(K.roots_of_unity())
24
```

```python
sage: all(u in K.roots_of_unity() 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[]
```

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

```python
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
```

### 1.3. Relative Number Fields
(Number Field in \(c_0\) with defining polynomial \(x^2 - 24x + 96\), Ring morphism:
  \(\text{From: Number Field in } c_0 \text{ with defining polynomial } x^2 - 24x + 96\)
  \(\text{To: Number Field in } c \text{ with defining polynomial } Y^2 + (-2b - 3)a - 2b - 6\)
  \(\text{ over its base field}\)
  \(\text{Defn: } c_0 \mapsto -4b + 12, \text{ None})

(Number Field in \(c_1\) with defining polynomial \(x^2 - 24x + 120\), Ring morphism:
  \(\text{From: Number Field in } c_1 \text{ with defining polynomial } x^2 - 24x + 120\)
  \(\text{To: Number Field in } c \text{ with defining polynomial } Y^2 + (-2b - 3)a - 2b - 6\)
  \(\text{ over its base field}\)
  \(\text{Defn: } c_1 \mapsto 2b^2a + 12, \text{ None})

(Number Field in \(c_2\) with defining polynomial \(x^2 - 24x + 72\), Ring morphism:
  \(\text{From: Number Field in } c_2 \text{ with defining polynomial } x^2 - 24x + 72\)
  \(\text{To: Number Field in } c \text{ with defining polynomial } Y^2 + (-2b - 3)a - 2b - 6\)
  \(\text{ over its base field}\)
  \(\text{Defn: } c_2 \mapsto -6a + 12, \text{ None})

\[
\text{sage: } K\text{.subfields}(8, 'w')
\]

\[
(\text{Number Field in } w_0 \text{ with defining polynomial } x^8 - 12x^6 + 36x^4 - 36x^2 + 9, \text{ \(\rightarrow\) Ring morphism:}
  \text{\(\text{From: Number Field in } w_0 \text{ with defining polynomial } x^8 - 12x^6 + 36x^4 - \) \(\rightarrow\) 36x^2 + 9}
  \text{\(\text{To: Number Field in } c \text{ with defining polynomial } Y^2 + (-2b - 3)a - 2b - 6\)}
  \(\text{ over its base field}\)
  \(\text{Defn: } w_0 \mapsto (-1/2b^2a + 1/2b + 1/2)c, \text{ Relative number field morphism:}
  \text{\(\text{From: Number Field in } c \text{ with defining polynomial } Y^2 + (-2b - 3)a - 2b - 6\)}
  \(\text{ over its base field}\)
  \(\text{To: Number Field in } w_0 \text{ with defining polynomial } x^8 - 12x^6 + 36x^4 - \) \(\rightarrow\) 36x^2 + 9}
  \(\text{Defn: } c \mapsto -1/3w_0^7 + 4w_0^5 - 12w_0^3 + 11w_0\)
  \(a \mapsto 1/3w_0^6 - 10/3w_0^4 + 5w_0^2\)
  \(b \mapsto -2/3w_0^6 + 7w_0^4 - 14w_0^2 + 6)\)
\]

\text{sage: } K\text{.subfields}(3)

[]

\text{uniformizer}(P, \text{ others='positive'} )

Returns an element of self with valuation 1 at the prime ideal \(P\).

\text{INPUT:}

\begin{itemize}
  \item \text{self} - a number field
  \item \text{P} - a prime ideal of self
  \item \text{others} - either “positive” (default), in which case the element will have non-negative valuation at all other primes of self, or “negative”, in which case the element will have non-positive valuation at all other primes of self.
\end{itemize}

\text{Note:} When \(P\) is principal (e.g. always when self has class number one) the result may or may not be a generator of \(P\)!

\text{EXAMPLES:}
Algebraic Numbers and Number Fields, Release 9.6

```
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(**args, **kwds)**

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
```

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
```

1.3. Relative Number Fields 135
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

```python
class sage.rings.number_field.number_field_element.CoordinateFunction(alpha, W, to_V):
    Bases: object

    This class provides a callable object which expresses elements in terms of powers of a fixed field generator \( \alpha \).

    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
```

```python
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 \( \text{prec} \) is given). Otherwise, if it has a coercion embedding into \( \mathbb{C} \), then the corresponding absolute value is returned. Finally, if there is no coercion embedding, \( i \) defaults to 0.
```
For the computation, the complex field with \texttt{prec} bits of precision is used, defaulting to 53 bits of precision if \texttt{prec} is not provided. The result is in the corresponding real field.

INPUT:

- \texttt{prec} - (default: None) integer bits of precision
- \texttt{i} - (default: None) integer, which embedding to use

EXAMPLES:

\begin{verbatim}
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: a.abs(prec=100)
2.5712815906582353554531872087
sage: a.abs(prec=100, i=1)
2.5712815906582353554531872087
sage: a.abs(100, 2)
2.5712815906582353554531872087

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

sage: K.<b> = NumberField(x^2-2)
sage: a = 1 + b
sage: a.abs(i=0)
0.414213562373095
sage: a.abs(i=1)
2.41421356237309

Check that trac ticket \#16147 is fixed:

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

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

sage: x = polygen(ZZ)
sage: f = x^3 - x - 1
sage: K.<b> = NumberField(f, embedding=1.3)
sage: b.abs()
1.3

However, if a specific embedding is requested, the behavior reverts to that of number fields without a coercion embedding into \( \mathbb{R} \):
Algebraic Numbers and Number Fields, Release 9.6

```
sage: b.abs(i=2)
1.32471795724475
```

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

```
sage: b.abs(prec=53)
1.32471795724475
```

**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.

**INPUT:**

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

**OUTPUT:**

(real) the non-archimedean absolute value of this element with respect to the prime $P$, to the given precision. This is the normalised absolute value, so that the underlying prime number $p$ has absolute value $1/p$.

**EXAMPLES:**

```
sage: K.<a> = NumberField(x^2+5)
sage: [1/K(2).abs_non_arch(P) for P in K.primes_above(2)]
[2.00000000000000]
sage: [1/K(3).abs_non_arch(P) for P in K.primes_above(3)]
[3.00000000000000, 3.00000000000000]
sage: [1/K(5).abs_non_arch(P) for P in K.primes_above(5)]
[5.00000000000000]
```

A relative example:

```
sage: L.<b> = K.extension(x^2-5)
sage: [b.abs_non_arch(P) for P in L.primes_above(b)]
[0.447213595499958, 0.447213595499958]
```

**absolute_different**()

Return the absolute different of this element.

This means the different with respect to the base field $Q$.

**EXAMPLES:**

```
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: a.absolute_different()
0
```

See also:

- **different()**

**absolute_norm**()

Return the absolute norm of this number field element.

**EXAMPLES:**
```
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:**

```
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:**

```
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.ceil()
5
```

This function always succeeds even if a tremendous precision is needed:

```
sage: c = b - 5065701199253/1225243417356 + 2
sage: c.ceil()
3
sage: RIF(c).unique_ceil()
Traceback (most recent call last):
  ...
ValueError: interval does not have a unique ceil
```

If the number field is not embedded, this function is valid only if the element is rational:

```
sage: p = x**5 - 3
sage: K.<a> = NumberField(p)
sage: K(2/3).ceil()
1
sage: a.ceil()
```

(continues on next page)
Traceback (most recent call last):
...
TypeError: ceil not uniquely defined since no real embedding is specified

**charpoly**(var='x')
Return the characteristic polynomial of this number field element.

**EXAMPLES:**

```sage
sage: 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**(prec=53, i=0)
Return the i-th embedding of self in the complex numbers, to the given precision.

**EXAMPLES:**

```sage
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 - 1.091123635971721403560726142*I
sage: a.complex_embedding(20, 1)
-0.62996 + 1.0911*I
sage: a.complex_embedding(20, 2)
1.2599
```

**complex_embeddings**(prec=53)
Return the images of this element in the floating point complex numbers, to the given bits of precision.

**INPUT:**

- **prec** - integer (default: 53) bits of precision

**EXAMPLES:**

```sage
sage: k.<a> = NumberField(x^3 - 2)
sage: a.complex_embeddings()
[-0.629960524947437 - 1.09112363597172*I, -0.629960524947437 + 1.
  →0.91123635971721403560726142*I, 1.25992104989487]
sage: a.complex_embeddings(10)
[-0.63 - 1.1*I, -0.63 + 1.1*I, 1.3]
sage: a.complex_embeddings(100)
[-0.62996052494743658238360530364 - 1.091123635971721403560726142*I, 
  -0.62996052494743658238360530364 + 1.091123635971721403560726142*I, 
  1.2599210498948731647672106073]
```

**conjugate**()
Return the complex conjugate of the number field element.

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

EXAMPLES:

```python
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.

```python
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.

```python
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.

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

An example of a non-cyclotomic CM field.

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

coordinates_in_terms_of_powers()  
Let α 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 \( \alpha: 1, \alpha, \alpha^2, \ldots \)  
(NOT CACHED).

EXAMPLES:

This function allows us to write elements of a number field in terms of a different generator without having to construct a whole separate number field.
This function works even if self only generates a subfield of this number field.

```python
sage: k.<a> = NumberField(x^6 - 5)
sage: alpha = a^3
sage: c = alpha.coordinates_in_terms_of_powers()
sage: c((2/3)*a^3 - 5/3)
[-5/3, 2/3]
sage: c
Coordinate function that writes elements in terms of the powers of a^3
sage: c(a)
Traceback (most recent call last):
... ArithmeticError: vector is not in free module
```

denominator()  
Return the denominator of this element, which is by definition the denominator of the corresponding polynomial representation. I.e., elements of number fields are represented as a polynomial (in reduced form) modulo the modulus of the number field, and the denominator is the denominator of this polynomial.

EXAMPLES:

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

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:

```python
sage: K.<a> = NumberField(x^2+5)
sage: b = (1+a)/2
sage: b.norm()
```

(continues on next page)
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.

INPUT:

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

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 $K^*/(K^*)^d \to L^*/(L^*)^d$ where $L$ is the parent of \texttt{self}. A \texttt{ValueError} is raised if $K$ does not embed into $L$.

ALGORITHM:

All preimages must lie in the Selmer group $K(S, d)$ for a suitable finite set of primes $S$, which reduces the question to a finite set of possibilities. We may take $S$ to be the set of primes which ramify in $L$ together with those for which the valuation of \texttt{self} is not divisible by $d$.

EXAMPLES:

A relative example:

```sage
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]
```

An absolute example:

```sage
sage: K.<zeta> = CyclotomicField(8)
sage: K(1).descend_mod_power(QQ,2)
[1, 2, -1, -2]
sage: a = 17 * K._random_nonzero_element()^2
sage: a.descend_mod_power(QQ,2)
[17, 34, -17, -34]
```

\textbf{different}(K=None)

Return the different of this element with respect to the given base field.

The different of an element $a$ in an extension of number fields $L/K$ is defined as $\text{Diff}_{L/K}(a) = f'(a)$ where $f$ is the characteristic polynomial of $a$ over $K$.

INPUT:
• $K$ – a subfield (embedding of a subfield) of the parent number field of `self`. Default: None, which will amount to `self.parent().base_field()`.

EXAMPLES:

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

The optional argument $K$ can be an embedding of a subfield:

```python
sage: K.<b> = NumberField(x^4 - 2)
sage: (b^2).different()
0
sage: phi = K.base_field().embeddings(K)[0]
sage: b.different(K=phi)
4*b^3
```

Compare the relative different and the absolute different for an element in a relative number field:

```python
sage: K.<a> = NumberFieldTower([x^2 - 17, x^3 - 2])
sage: a.different()
2*a0
sage: a.different(K=QQ)
0
sage: a.absolute_different()
0
```

Observe that for the field extension $\mathbb{Q}(i)/\mathbb{Q}$, the different of the field extension is the ideal generated by the different of $i$:

```python
sage: K.<c> = NumberField(x^2 + 1)
sage: K.different() == K.ideal(c.different())
True
```

See also:

- `absolute_different()`
- `sage.rings.number_field.number_field_rel.NumberField_relative.different()`

`factor()`

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

OUTPUT:

(Factorization) If all the prime ideals in the support are principal, the output is a Factorization as a product of prime elements raised to appropriate powers, with an appropriate unit factor.

Raise ValueError if the factorization of the ideal (self) contains a non-principal prime ideal.

EXAMPLES:

```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:
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:

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:

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:

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:

sage: p = x^5 - 3
sage: K.<a> = NumberField(p)
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

1.4. Number Field Elements
**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 \(3 \sqrt{2}\) in \(Q(3 \sqrt{2})\).

```sage
sage: a.galois_conjugates(K)
[a]
```

Galois conjugates of \(3 \sqrt{2}\) in the field \(Q(\zeta_3, 3 \sqrt{2})\):

```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
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):
  ...*polynomial x^2 + 1 with i = 1*I 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
Sage: R(a).gcd(R(2*a)).parent()
Maximal Order in Number Field in a with defining polynomial x^3 - 7

**global_height** *(prec=None)*

Returns the absolute logarithmic height of this number field element.

**INPUT:**

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

**OUTPUT:**

(real) The absolute logarithmic height of this number field element; that is, the sum of the local heights at all finite and infinite places, scaled by the degree to make the result independent of the parent field.

**EXAMPLES:**

Sage: R.<x> = QQ[]
Sage: K.<a> = NumberField(x^4+3*x^2-17)
Sage: b = a/2
Sage: b.global_height()
0.789780699008...

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

```sage
sage: QQ(6).global_height()
1.79175946922805
sage: K(6).global_height()
1.79175946922805
```

```sage
sage: L.<b> = NumberField((a^2).minpoly())
sage: L.degree()
2
sage: b.global_height()  # element of L (degree 2 field)
1.41660667202811
sage: (a^2).global_height()  # element of K (degree 4 field)
1.41660667202811
```

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

```sage
sage: K.<s> = QuadraticField(2)
sage: s.global_height()
0.346573590279973
sage: (1/s).global_height()  # make sure that 11758 is fixed
0.346573590279973
```

---

**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
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^4+3*x^2-17)
sage: b = a/2
sage: b.global_height_arch()
0.38653407379277...
```

---

**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
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:

```sage
sage: [b.local_height(P) for P in b.support()]
[0.000000000000000, 0.693147180559945, 1.09861228866811, 1.09861228866811]
sage: sum([b.local_height(P, weighted=True) for P in b.support()])
7.16703787691222
```

A relative example:

```sage
sage: PK.<y> = K[]
sage: L.<c> = NumberField(y^2 + a)
sage: (c/10).global_height_non_arch()
18.4206807439524
```

`inverse_mod(I)`

Returns the inverse of self mod the integral ideal $I$.

**INPUT:**

- $I$ - may be an ideal of self.parent(), or an element or list of elements of self.parent() generating a nonzero ideal. A ValueError is raised if $I$ is non-integral or zero. A ZeroDivisionError is raised if $I + (x) \neq (1)$.

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

**EXAMPLES:**

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

sage: k.<a> = NumberField(x^3 + 11)
sage: d = a + 13
sage: d.inverse_mod(a^2)*d - 1 in k.ideal(a^2)
True
sage: d.inverse_mod((5, a + 1))*d - 1 in k.ideal(5, a + 1)
True
sage: K.<b> = k.extension(x^2 + 3)
sage: b.inverse_mod([37, a - b])
7
```

(continues on next page)
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:**

```
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:**

```
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:

```
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 self == $x$.norm($K$). Otherwise, $x$ is None.

ALGORITHM:
Uses PARI’s pari:rnfisnorm. See self._rnfisnorm().

EXAMPLES:

```
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:

```
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:

```
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 from non-Galois number fields
sage: (a/2).is_norm(L, proof=False)
False
```

(continues on next page)
Verify that trac ticket #27469 has been fixed:

```python
sage: L.<z24> = CyclotomicField(24); L
Cyclotomic Field of order 24 and degree 8
sage: K = L.subfield(z24^3, 'z8')[0]; K
Number Field in z8 with defining polynomial x^4 + 1 with z8 = 0.
\rightarrow 7071067811865475? + 0.7071067811865475?*I
sage: flag, c = K(-7).is_norm(K, element=True)
sage: flag
True
sage: c.norm(K)
-7
sage: c in L
True
```

AUTHORS:

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

**is_nth_power(n)**

Return True if self is an n'th power in its parent K.

**EXAMPLES:**

```python
sage: K.<a> = NumberField(x^4-7)
sage: K(7).is_nth_power(2)
True
sage: K(7).is_nth_power(4)
True
sage: K(7).is_nth_power(8)
False
sage: K((a-3)^5).is_nth_power(5)
True
```

**ALGORITHM:** Use PARI to factor $x^n - \text{self}$ in $K$.

**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:

sage: K.<a> = NumberField(x^2 + 2)
sage: p = K.primes_above(2)[0]
sage: K(5).is_padic_square(p)
False

is_rational()
Test whether this number field element is a rational number

See also:
• is_integer() to test if this element is an integer
• is_integral() to test if this element is an algebraic integer

EXAMPLES:

sage: K.<cbrt3> = NumberField(x^3 - 3)
sage: cbrt3.is_rational()
False
sage: (cbrt3**2 - cbrt3 + 1/2).is_rational()
False
sage: K(-12).is_rational()
True
sage: K(0).is_rational()
True
sage: K(1/2).is_rational()
True

is_square(root=False)
Return True if self is a square in its parent number field and otherwise return False.

INPUT:
• root - if True, also return a square root (or None if self is not a perfect square)

EXAMPLES:

sage: m.<b> = NumberField(x^4 - 1789)
sage: b.is_square()
(continues on next page)
False
\texttt{sage: }\texttt{c} = (2/3*b + 5)^2; \texttt{c}
\texttt{4/9*b^2 + 20/3*b + 25}
\texttt{sage: }\texttt{c.is\_square()}
\texttt{True}
\texttt{sage: }\texttt{c.is\_square(\texttt{True})}
\texttt{(True, 2/3*b + 5)}

We also test the functional notation.

\texttt{sage: }\texttt{is\_square(c, \texttt{True})}
\texttt{(True, 2/3*b + 5)}
\texttt{sage: }\texttt{is\_square(c)}
\texttt{True}
\texttt{sage: }\texttt{is\_square(c+1)}
\texttt{False}

\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.

\textbf{EXAMPLES:}

\texttt{sage: }\texttt{F, \textless b\textgreater } = \texttt{NumberField(x^3-3*x-1)}
\texttt{sage: }\texttt{b.is\_totally\_positive()}
\texttt{False}
\texttt{sage: }\texttt{(b^2).is\_totally\_positive()}
\texttt{True}

\textbf{is\_unit()}

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

\textbf{EXAMPLES:}

\texttt{sage: }\texttt{K, \textless a\textgreater } = \texttt{NumberField(x^2 - x - 1)}
\texttt{sage: }\texttt{OK} = \texttt{K.ring\_of\_integers()}
\texttt{sage: }\texttt{OK(a).is\_unit()}
\texttt{True}
\texttt{sage: }\texttt{OK(13).is\_unit()}
\texttt{False}
\texttt{sage: }\texttt{K(13).is\_unit()}
\texttt{True}

It also works for relative fields and orders:

\texttt{sage: }\texttt{K, \textless a, b\textgreater } = \texttt{NumberField([x^4 - 3, x^4 + x^3 + x^2 + x + 1])}
\texttt{sage: }\texttt{OK} = \texttt{K.ring\_of\_integers()}
\texttt{sage: }\texttt{OK(b).is\_unit()}
\texttt{True}
\texttt{sage: }\texttt{OK(a).is\_unit()}
\texttt{False}
\texttt{sage: }\texttt{a.is\_unit()}
\texttt{True}
list()

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

EXAMPLES:

```sage
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
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)
4.110873864173311248751389134256147463156817430812610629374
sage: b.local_height(P, 200)
8.22174772834662
sage: (b^-1).local_height(P)
0.000000000000000
```

A relative example:

```sage
sage: PK.<y> = K[]
sage: L.<c> = NumberField(y^2 + a)
sage: L(1/4).local_height(L.ideal(2, c-a+1))
1.3862943611989
```

local_height_arch(i, prec=None, weighted=False)

Returns the local height of self at the i\text{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:

```
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:

```
sage: L.<b, c> = NumberFieldTower([x^2 - 5, x^3 + x + 3])
sage: [(b + c).local_height_arch(i) for i in range(4)]
[1.238223390757884911842206617439,
 0.02240347229957875780769746914391,
 0.780028961749618,
 1.16048938497298]
```

`matrix(base=None)`

If base is None, return the matrix of right multiplication by the element on the power basis 1, x, x^2, ..., 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:

```
sage: K.<a> = NumberField(QQ['x'].0^3 - 5)
sage: M = a.matrix(); M
[0 1 0]
[0 0 1]
```

(continues on next page)
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:
sage: a.norm(v[1])
a2
sage: a.norm(v[0])
-a2

\textbf{minpoly}(\textit{var}=\text quote\text{ts1}x\text quote\text{ts1})

Return the minimal polynomial of this number field element.

\textbf{EXAMPLES:}

\begin{verbatim}
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
\end{verbatim}

\textbf{multiplicative_order()}

Return the multiplicative order of this number field element.

\textbf{EXAMPLES:}

\begin{verbatim}
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
\end{verbatim}

An example in a relative extension:

\begin{verbatim}
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
\end{verbatim}

\textbf{norm}(\textit{K}=\text{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 $\mathbb{Q}$.

EXAMPLES:

```python
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.

```python
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+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:

```python
sage: z = (a+b+c).norm(L); z.norm(M)
-11
```

If we are in an order, the norm is an integer:

```python
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:

```python
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
```

(continues on next page)
nth_root\((n, \text{all}=\text{False})\)
Return an \(n\)’th root of \(self\) in its parent \(K\).

**EXAMPLES:**

```
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
```

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

**numerator_ideal()**
Return the numerator ideal of this number field element.

The numerator ideal of a number field element \(a\) is the ideal of the ring of integers \(R\) obtained by intersecting \(aR\) with \(R\).

**See also:**
`denominator_ideal()`

**EXAMPLES:**

```
sage: 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
```

**ord\(P\)**
Return the valuation of \(self\) at a given prime ideal \(P\).

**INPUT:**
- \(P\) – a prime ideal of the parent of \(self\)

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

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

Note that in relative number fields, this produces the polynomial of the internal representation of this element:

```sage
sage: R.<y> = K[]
sage: L.<b> = K.extension(y^2 - a)
sage: b.polynomial()
x
```

In some cases this might not be what you are looking for:

```sage
sage: K.<a> = NumberField(x^2 + x + 1)
sage: R.<y> = K[]
sage: L.<b> = K.extension(y^2 + y + 2)
sage: b.polynomial()
1/2*y^3 + 3*y - 1/2
sage: R(list(b))
y
```
relative_norm()
Return the relative norm of this number field element over the next field down in some tower of number
fields.

EXAMPLES:

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

residue_symbol(P, m, check=True)
The m-th power residue symbol for an element self and proper ideal P.

\[
\left( \frac{\alpha}{P} \right) \equiv \alpha^{N(P)/m} \mod P
\]

**Note:** accepts m=1, in which case returns 1

**Note:** can also be called for an ideal from \( \text{sage.rings.number_field_ideal.residue_symbol} \)

**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):

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

The result depends on the number field of the ideal:
sage: K.<a> = NumberField(x - 1)
sage: L.<b> = K.extension(x^2 + 1)
sage: K(7).residue_symbol(K.ideal(11),2)
-1
sage: K(7).residue_symbol(L.ideal(11),2)
1

Cubic Residue:

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

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

sage: K.<w> = NumberField(x^2 - x + 1)
sage: (w^2 + 3).residue_symbol(K.ideal(17),5)
Traceback (most recent call last):
  ... ValueError: The residue symbol to that power is not defined for the number field

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

EXAMPLES:

sage: x = polygen(ZZ)
sage: p = x^7 - 5*x^2 + x + 1
sage: a_AA = AA.polynomial_root(p, RIF(1,2))
sage: K.<a> = NumberField(p, embedding=a_AA)
sage: b = a^5 + a/2 - 1/7
sage: RR(b)
4.13444473767055
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]
sage: a.round()
Traceback (most recent call last):
  ...
TypeError: floor not uniquely defined since no real embedding is specified

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

EXAMPLES:

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, extend=True)
Return the square root of this number in the given number field.

INPUT:

- all – optional boolean (default False); whether to return both square roots
- extend – optional boolean (default True); whether to extend the field by adding the square roots if needed

EXAMPLES:

sage: K.<a> = NumberField(x^2 - 3)
sage: K(3).sqrt()
a
Using the `extend` keyword:

```
sage: K = QuadraticField(-5)
sage: z = K(-7).sqrt(extend=True); z
sqrt(-7)
sage: CyclotomicField(4)(4).sqrt(extend=False)
2
```

If `extend=False` an error is raised, if `self` is not a square:

```
sage: K = QuadraticField(-5)
sage: K(-7).sqrt(extend=False)
Traceback (most recent call last):
...
ValueError: -7 not a square in Number Field in a with defining polynomial x^2 + 5 with a = 2.236067977499799790*I
```

ALGORITHM: Use PARI to factor $x^2 - self$ 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)
```
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)

Return 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))
<class 'sage.rings.integer.Integer'>
```

The function can be applied to elements in relative number fields:
```python
sage: L.<b> = K.extension(x^2 - 3)
sage: [L(6).valuation(P) for P in L.primes_above(2)]
[4]
sage: [L(6).valuation(P) for P in L.primes_above(3)]
[2, 2]
```

```python
vector()

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

EXAMPLES:

```python
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: 0 = K.order(2*a)
sage: (0.1).vector()
(0, 2)
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: (a + b).vector()
(b, 1)
sage: 0 = K.order([a,b])
sage: (0.1).vector()
(-b, 1)
sage: (0.2).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:

```python
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 :func:`charpoly`.

**EXAMPLES:**

```
sage: x = ZZ['x'].0
sage: f = x^10 - 5*x^9 + 15*x^8 - 68*x^7 + 81*x^6 - 221*x^5 + 141*x^4 - 242*x^3 - 13*x^2 - 33*x - 135
sage: K.<a> = NumberField(f, 'a')
sage: a.absolute_charpoly()  
x^10 - 5*x^9 + 15*x^8 - 68*x^7 + 81*x^6 - 221*x^5 + 141*x^4 - 242*x^3 - 13*x^2 - 33*x - 135
sage: a.absolute_charpoly('y')
y^10 - 5*y^9 + 15*y^8 - 68*y^7 + 81*y^6 - 221*y^5 + 141*y^4 - 242*y^3 - 13*y^2 - 33*y - 135
sage: b = -79/9995*a^9 + 52/9995*a^8 + 271/9995*a^7 + 1663/9995*a^6 + 13204/9995*a^5 + 5573/9995*a^4 + 8435/1999*a^3 - 3116/9995*a^2 + 7734/1999*a + 1620/1999
sage: b.absolute_charpoly()  
x^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 :meth:`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 :func:`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.

```
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 :class:`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:**

```python
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: K.<a> = CyclotomicField(1009)
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

Return an element of $\mathbb{Q}[x]$, where this number field element lives in $\mathbb{Q}[x]/(f(x))$.

**EXAMPLES:**

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

### list

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

**EXAMPLES:**

```python
sage: K.<z> = CyclotomicField(3)
sage: (2+3/5*z).list()  # [2, 3/5]

sage: (5*z).list()  # [0, 5]

sage: K(3).list()  # [3, 0]
```

### minpoly

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
sage: a.minpoly('y').parent()
Univariate Polynomial Ring in y over Rational Field
```

**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:**

```
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 **absolute_charpoly**().

**EXAMPLES:**

```
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)
```

(continues on next page)
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
sage: L(a).absolute_minpoly()
x^2 + 2
sage: L(b).absolute_charpoly()
x^8 + 4000*x^7 + 6000004*x^6 + 4000012000*x^5 + 1000012000006*x^4 + ...
   → 4000012000*x^3 + 6000004*x^2 + 4000*x + 1
sage: L(b).absolute_minpoly()
x^2 + 1000*x + 1

\texttt{charpoly(}\texttt{var='}x\texttt{'})

The characteristic polynomial of this element over its base field.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = ZZ['x'].0
tsage: K.<a, b> = QQ.extension([x^2 + 2, x^5 + 400*x^4 + 11*x^2 + 2])
tsage: a.charpoly()
x^2 + 2
tsage: b.charpoly()
x^2 - 2*b*x + b^2
tsage: b.minpoly()
x - b
tsage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
tsage: y = K['y'].0
tsage: L.<c> = K.extension(y^2 + a*y + b)
tsage: c.charpoly()
x^2 + a*x + b
tsage: c.minpoly()
x^2 + a*x + b
tsage: L(a).charpoly()
x^8 - 2*a^2*x - 2
stsage: L(a).minpoly()
x - a
stsage: L(b).charpoly()
x^8 - 2*b*x - 1000*b - 1
stsage: L(b).minpoly()
x - b
\end{verbatim}

\texttt{lift(}\texttt{var='}x\texttt{'})

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

\textbf{EXAMPLES:}

\begin{verbatim}
sage: K.<a> = QuadraticField(-3)
tsage: x = polygen(K)
tsage: L.<b> = K.extension(x^7 + 5)
tsage: u = L(1/2*a + 1/2 + b + (a-9)*b^5)
\end{verbatim}

(continues on next page)
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()
[0, 1, 0]
sage: v = (K.base_field().0 + a)^2 ; v
a^2 + 2*b*a - 1
sage: v.list()
[-1, 2*b, 1]

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

class sage.rings.number_field.number_field_element.OrderElement_absolute
Bases: sage.rings.number_field.number_field_element.NumberFieldElement_absolute

Element of an order in an absolute number field.

EXAMPLES:

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

sage: w.absolute_charpoly()
x^2 + 4
sage: w.absolute_charpoly().parent()
Univariate Polynomial Ring in x over Integer Ring
sage: w.absolute_minpoly()
x^2 + 4
sage: w.absolute_minpoly().parent()
Univariate Polynomial Ring in x over Integer Ring

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^3 - x + 2)
sage: w.inverse_mod(13*OE)
6*w^2 - 6
sage: w * (w.inverse_mod(13)) - 1 in 13*OE
True
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)
```

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:

```python
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)
<class 'sage.rings.number_field.number_field_element.OrderElement_relative'>
```

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

EXAMPLES:

```python
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order()
sage: u, v = OK.basis()
sage: t = 2*u - v; t
-b
sage: t.absolute_charpoly()
x^4 - 6*x^2 + 9
```

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

EXAMPLES:

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

(continues on next page)
sage: OK = K.maximal_order()
sage: _, u, _, v = OK.basis()
sage: t = 2*u - v; t
-b
sage: t.absolute_charpoly()
x^4 - 6*x^2 + 9
sage: t.absolute_minpoly()
x^2 - 3
sage: t.absolute_minpoly().parent()
Univariate Polynomial Ring in x over Integer Ring

\textbf{charpoly}(\textit{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 \(\mathbb{Z}Z\); it should be the ring of integers of the base field.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
sage: charpoly(OK.1)
x^2 + b*x + 1
sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order in Number Field in b with defining polynomial x^2 - 3
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]
\end{verbatim}

\textbf{inverse_mod}(I)

Return an inverse of self modulo the given ideal.

\textbf{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 \(I + (x) \neq (1)\).

\textbf{EXAMPLES:}

\begin{verbatim}
sage: E.<a,b> = NumberField([x^2 - x + 2, x^2 + 1])
sage: OE = E.ring_of_integers()
sage: t = OE(b - a).inverse_mod(17*b)
sage: t*(b - a) - 1 in E.ideal(17*b)
True
sage: t.parent() == OE
True
\end{verbatim}

\textbf{minpoly}(\textit{var='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 \(\mathbb{Z}Z\); it should be the ring of integers of the base field.

\textbf{EXAMPLES:}
```python
sage: x = ZZ['x'].0
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 3])
sage: OK = K.maximal_order(); OK.basis()
[1, 1/2*a - 1/2*b, -1/2*b*a + 1/2, a]
sage: minpoly(OK.1)
x^2 + b*x + 1
sage: charpoly(OK.1).parent()
Univariate Polynomial Ring in x over Maximal Order in Number Field in b with defining polynomial x^2 - 3
sage: _, u, _, v = OK.basis()
sage: t = 2*u - v; t
-b
sage: t.charpoly()
x^2 + 2*b*x + 3
sage: t.minpoly()
x + b
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
class sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian
    Bases: sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic_sqrt

An element of \( \mathbb{Q}[i] \).

Some methods of this class behave slightly differently than the corresponding methods of general elements of quadratic number fields, especially with regard to conversions to parents that can represent complex numbers in rectangular form.

In addition, this class provides some convenience methods similar to methods of symbolic expressions to make the behavior of \( a + I*b \) with rational \( a, b \) closer to that when \( a, b \) are expressions.

EXAMPLES:

```python
sage: type(I)
<class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian'>
sage: mi = QuadraticField(-1, embedding=CC(0,-1)).gen()
sage: type(mi)
<class 'sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_gaussian'>
sage: CC(mi)
-1.00000000000000*I
```

`imag()`

Imaginary part.

EXAMPLES:

```python
sage: (1 + 2*I).imag()
2
sage: (1 + 2*I).imag().parent()
Rational Field
sage: K.<mi> = QuadraticField(-1, embedding=CC(0,-1))
sage: (1 - mi).imag()
1
```

`imag_part()`

Imaginary part.

EXAMPLES:

```python
sage: (1 + 2*I).imag()
2
sage: (1 + 2*I).imag().parent()
Rational Field
sage: K.<mi> = QuadraticField(-1, embedding=CC(0,-1))
sage: (1 - mi).imag()
1
```

`log(*args, **kwds)`

Complex logarithm (standard branch).

EXAMPLES:
real()
Real part.

EXAMPLES:

```
sage: (1 + 2*I).real()
sage: 1
sage: (1 + 2*I).real().parent()
Rational Field
```

real_part()
Real part.

EXAMPLES:

```
sage: (1 + 2*I).real()
sage: 1
sage: (1 + 2*I).real().parent()
Rational Field
```

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:

```
sage: K.<sqrt7> = QuadraticField(7, name='sqrt7')
sage: sqrt7.ceil()
sage: 3
sage: (-sqrt7).ceil()
sage: -2
sage: (1022/313*sqrt7 - 14/23).ceil()
sage: 9
```

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
```
\texttt{sage}: \texttt{f(b)} \\\ \texttt{0}

\textbf{continued\_fraction()}

Return the (finite or ultimately periodic) continued fraction of \texttt{self}.

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage}: \texttt{K.<sqrt2> = QuadraticField(2)}
\texttt{sage}: \texttt{cf = sqrt2.continued_fraction(); cf}
\texttt{[1; (2)*]}
\texttt{sage}: \texttt{cf.n()}
\texttt{1.41421356237310}
\texttt{sage}: \texttt{sqrt2.n()}
\texttt{1.41421356237309}
\texttt{sage}: \texttt{cf.value()}
\texttt{sqrt2}
\texttt{sage}: \texttt{(sqrt2/3 + 1/4).continued_fraction()}
\texttt{[0; 1, (2, 1, 1, 2, 3, 2, 1, 1, 2, 5, 1, 1, 14, 1, 1, 5)*]}
\end{verbatim}

\textbf{continued\_fraction\_list()}

Return the preperiod and the period of the continued fraction expansion of \texttt{self}.

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage}: \texttt{K.<sqrt2> = QuadraticField(2)}
\texttt{sage}: \texttt{sqrt2.continued_fraction_list()}
\texttt{((1,), (2,))}
\texttt{sage}: \texttt{(1/2+sqrt2/3).continued_fraction_list()}
\texttt{((0, 1, 33), (1, 32))}
\end{verbatim}

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

\begin{verbatim}
\texttt{sage}: \texttt{K(123/567).continued_fraction_list()}
\texttt{((0, 4, 1, 1, 1, 1, 3, 2), ())}
\end{verbatim}

\textbf{denominator()}

Return the denominator of \texttt{self}.

This is the LCM of the denominators of the coefficients of \texttt{self}, and thus it may well be \textgreater{} 1 even when the element is an algebraic integer.

\textbf{EXAMPLES:}

\begin{verbatim}
\texttt{sage}: \texttt{K.<a> = NumberField(x^2 - 5)}
\texttt{sage}: \texttt{b = (a + 1)/2}
\texttt{sage}: \texttt{b.denominator()}
\texttt{2}
\texttt{sage}: \texttt{b.is_integral()}
\texttt{True}
\texttt{sage}: \texttt{K.<c> = NumberField(x^2-x+7)}
\texttt{sage}: \texttt{c.denominator()}
\texttt{1}
\end{verbatim}
floor()
Returns the floor of x.

EXAMPLES:

```
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:

```
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()
Return the imaginary part of self.

EXAMPLES:

```
sage: K.<sqrt2> = QuadraticField(2)
sage: sqrt2.imag()
0
sage: parent(sqrt2.imag())
Rational Field
sage: K.<i> = QuadraticField(-1)
sage: i.imag()
1
sage: parent(i.imag())
Rational Field
sage: K.<a> = NumberField(x^2 + x + 1, embedding=CDF.0)
sage: a.imag()
1/2*sqrt3
```

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:

```
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:

```
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()
x^2 + 13
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()
3
sage: a.matrix()
[ 0 1]
[-3 1]
sage: K.<a> = NumberField(x^2+5)
sage: (1+a).norm()
6
```

The norm is multiplicative:

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

(continues on next page)
-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

numerateur()  
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.numerateur()
2*a + 1

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

This is much closer to the internal storage format of the elements than the polynomial representation coeffi-
cients (the output of \( \text{self}.\text{list}() \)), 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)
The real part of a quadratic number field element can be accessed using the `real()` method. Here are some 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
```

The `round()` method returns the round (nearest integer) of a quadratic number field element. Here are some examples:

```python
sage: K.<sqrt7> = QuadraticField(7, name='sqrt7')
sage: sqrt7.round()
3
sage: (-sqrt7).round()
-3
```

The `sign()` method returns the sign of a quadratic number field element (0 if zero, +1 if positive and -1 if negative). Here are some examples:

```python
sage: K.<sqrt2> = QuadraticField(2, name='sqrt2')
sage: K(0).sign()
0
sage: sqrt2.sign()
1
```

(continues on next page)
sage: (sqrt2-2).sign()
-1
sage: (-sqrt2).sign()
-1
sage: (-sqrt2+1).sign()
-1
sage: (-sqrt2+2).sign()
1

sage: K.<a> = QuadraticField(2, embedding=-1.4142)
sage: K(0).sign()
0
sage: a.sign()
-1
sage: (a+1).sign()
-1
sage: (a+2).sign()
1
sage: (a-1).sign()
-1
sage: (-a).sign()
1
sage: (-a-1).sign()
1
sage: (-a-2).sign()
-1

sage: K.<b> = NumberField(x^2 + 2*x + 7, 'b', embedding=CC(-1,-sqrt(6)))
sage: b.sign()
Traceback (most recent call last):
  ... ValueError: a complex number has no sign!

trace() 

EXAMPLES:

sage: K.<a> = NumberField(x^2+7)
sage: a.trace()
-1
sage: a.matrix()
[ 0 1]
[-41 -1]

The trace is additive:
sage: (a+1).trace()
sage: K(3).trace()
sage: (a+4).trace()
sage: (a/3+1).trace()

 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:

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.is_integral()
True

class sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic

Element of an order in a quadratic field.

EXAMPLES:

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

\text{charpoly}(\text{var}'x')

The characteristic polynomial of this element, which is over \mathbb{Z} because this element is an algebraic integer.

EXAMPLES:

\begin{verbatim}
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
\end{verbatim}

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:

\begin{verbatim}
sage: K.<a> = NumberField(x^2-27)
sage: R = K.ring_of_integers()
sage: aa = R.gen(1)
sage: aa.denominator()
3
\end{verbatim}

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:

\begin{verbatim}
sage: OE.<w> = EquationOrder(x^2 - x + 2)
sage: w.inverse_mod(13) == 6*w - 6
True
sage: w*(6*w - 6) - 1
-13
sage: w.inverse_mod(13).parent() == OE
True
sage: w.inverse_mod(2*OE)
Traceback (most recent call last):
...
ZeroDivisionError: w is not invertible modulo Fractional ideal (2)
\end{verbatim}
minpoly\( (\text{var}=x) \)

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
sage: f.parent()
Univariate Polynomial Ring in x over Integer Ring
sage: R(5).minpoly()
x - 5
```

\textbf{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
sage: parent(w.norm())
Integer Ring
```

\textbf{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
sage: parent(b.trace())
Integer Ring
```

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 with a = 1.
-732050807568878*I
sage: f(3/1)
3
```
sage: f(1/2).parent() is K
True

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:

sage: K.<a> = QuadraticField(3)
sage: phi = K.coerce_map_from(ZZ); phi
Natural morphism:
  From: Integer Ring
  To: Number Field in a with defining polynomial x^2 - 3 with a = 1.
→732050807568878?
sage: phi(4)
4
sage: phi(5).parent() is K
True

sage.rings.number_field.number_field_element_quadratic.is_sqrt_disc(ad, bd)

Return True if the pair (ad, bd) is $\sqrt{D}$.

EXAMPLES:

sage: F.<b> = NumberField(x^2 - x + 7)
sage: b.denominator()  # indirect doctest
1

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)
Bases: object

A class to store data for internal use in splitting_field(). It contains two attributes _pol (polynomial), _dm (degree multiple), where _pol is a PARI polynomial and _dm a Sage Integer.

_dm is a multiple of the degree of the splitting field of _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 _dm less than this _dm.

key()

Return a sorting key. Compare first by degree bound, then by polynomial degree, then by discriminant.

EXAMPLES:

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

EXAMPLES:

sage: from sage.rings.number_field.splitting_field import SplittingData
sage: SplittingData(pari("x^123 + x + 1"), 2).poldegree()
123

eception sage.rings.number_field.splitting_field.SplittingFieldAbort(div, mult)
Bases: Exception

Special exception class to indicate an early abort of splitting_field().

EXAMPLES:

sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: raise SplittingFieldAbort(20, 60)
Traceback (most recent call last):
... SplittingFieldAbort: degree of splitting field is a multiple of 20
sage: raise SplittingFieldAbort(12, 12)
Traceback (most recent call last):
... SplittingFieldAbort: degree of splitting field equals 12

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:**

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = (x^3 + 2).splitting_field(); K
Number Field in a with defining polynomial x^6 + 3*x^5 + 6*x^4 + 11*x^3 + 12*x^2 - 3*x + 1
sage: K.<a> = (x^3 - 3*x + 1).splitting_field(); K
Number Field in a with defining polynomial x^3 - 3*x + 1
```

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`.

```python
sage: (x^4 - x + 1).splitting_field('a', simplify=False)
Number Field in a with defining polynomial x^24 - 2780*x^22 + 2*x^21 + 3527512*x^20 -
    2876*x^19 - 270139185*x^18 + 945948*x^17 + 1390511639677*x^16 + 736757420*x^15 -
    506816498313560*x^14 - 822782898220*x^13 + 134120588299548463*x^12 -
    362240696528256*x^11 - 25964582366880639486*x^10 - 9174367224319990*x^9 +
    364942947344730843927*x^8 + 14310332927134072336*x^7 -
    363192569823568746892571*x^6 - 1353403793640477758*x^5 -
    24293393281774560140427565*x^4 + 70673814899934142357628*x^3 -
    98662147508959243128437933*x^2 - 1539841440617805445432660*x +
    18065914012013502602456565991

sage: (x^4 - x + 1).splitting_field('a', simplify=True)
Number Field in a with defining polynomial x^24 + 8*x^22 - 32*x^21 + 308*x^20 -
    5498*x^20 + 4688*x^19 + 6813*x^18 - 32380*x^17 + 49525*x^16 + 102460*x^15 -
    129944*x^14 - 287884*x^13 - 372727*x^12 + 150624*x^11 - 110530*x^10 - 566926*x^9 -
    1062759*x^8 - 779948*x^7 + 863493*x^6 - 1623578*x^5 + 1759513*x^4 - 955624*x^3 -
    459975*x^2 + 141948*x + 53919

sage: (x^4 - x + 1).splitting_field('a', simplify_all=True)
Number Field in a with defining polynomial x^24 - 3*x^23 - 32*x^22 + 308*x^21 -
    163*x^21 + 49*x^20 - 92*x^19 + 2*x^19 + 32*x^18 + 4*x^18 - 3*x^18 - 3*x^18 +
    -2*x^17 - 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^2 - 3*x + 1
```

Reducible polynomials also work:

```python
sage: pol = (x^4 - 1)*(x^2 + 1/2)*(x^2 + 1/3)
sage: pol.splitting_field('a', simplify_all=True)
Number Field in a with defining polynomial x^8 - x^4 + 1
```

**Relative situation:**

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^3 + 2)
sage: S.<t> = PolynomialRing(K)
```

(continues on next page)
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: 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: 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 - 35*x^8 - 32*x^6 + 4*x^5 - x^4 + 2*x^2 -
    -x + 1
    Defn: 1 |--> 1

We can enable verbose messages:

sage: from sage.misc.verbose import set_verbose
sage: set_verbose(2)
sage: K.<a> = (x^3 - x + 1).splitting_field()
verbose 1 (...: splitting_field.py, splitting_field) Starting field: y
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to factor: [(3, 0)]
verbose 2 (...: splitting_field.py, splitting_field) Done factoring (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to handle: [(2, 2), (3, 3)]
verbose 1 (...: splitting_field.py, splitting_field) Bounds for absolute degree: [6, 6]
verbose 2 (...: splitting_field.py, splitting_field) Handling polynomial x^2 + 23
verbose 1 (...: splitting_field.py, splitting_field) New field before simplifying:
    y^2 + 23 (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) New field: y^6 + 3*y^5 + 19*y^4 + 2 + 9*y^2 + 271 (time = ...)
verbose 2 (...: splitting_field.py, splitting_field) Converted polynomials to new field (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to factor: []
verbose 2 (...: splitting_field.py, splitting_field) Done factoring (time = ...)
verbose 1 (...: splitting_field.py, splitting_field) SplittingData to handle: [(3, 3)]
verbose 1 (...: splitting_field.py, splitting_field) Bounds for absolute degree: [6, 6]
verbose 2 (...: splitting_field.py, splitting_field) Handling polynomial x^3 - x + 1
verbose 1 (...: splitting_field.py, splitting_field) New field: y^6 + 3*y^5 + 19*y^4 + 2 + 9*y^2 + 271 (time = ...)

Try all Galois groups in degree 4. We use a quadratic base field such that polgalois() cannot be used:

1.6. Splitting fields of polynomials over number fields
sage: R.<x> = PolynomialRing(QuadraticField(-11))
sage: C2C2pol = x^4 - 10*x^2 + 1
sage: C2C2pol.splitting_field('x')
Number Field in x with defining polynomial x^8 + 24*x^6 + 608*x^4 + 9792*x^2 + 53824
sage: C4pol = x^4 + x^3 + x^2 + x + 1
sage: C4pol.splitting_field('x')
Number Field in x with defining polynomial x^8 - x^7 - 2*x^6 + 5*x^5 + x^4 + 15*x^3 - 18*x^2 - 27*x + 81
sage: D8pol = x^4 - 2
sage: D8pol.splitting_field('x')
Number Field in x with defining polynomial x^16 + 8*x^15 + 68*x^14 + 336*x^13 + 1514*x^12 + 5080*x^11 + 14912*x^10 + 35048*x^9 + 64959*x^8 + 93416*x^7 + 88216*x^6 + 6 + 41608*x^5 - 25586*x^4 - 60048*x^3 - 16628*x^2 + 12008*x + 34961
sage: A4pol = x^4 - 4*x^3 + 14*x^2 - 28*x + 21
sage: A4pol.splitting_field('x')
Number Field in x with defining polynomial x^24 - 20*x^23 + 290*x^22 - 3048*x^21 + 26147*x^20 - 186132*x^19 + 1130626*x^18 - 5913784*x^17 + 26899345*x^16 - 106792132*x^15 + 371066538*x^14 - 1127792656*x^13 + 2991524876*x^12 - 6888328132*x^11 + 13659660064*x^10 - 23000783036*x^9 + 32244796382*x^8 + 371066538*x^7 - 16707053128*x^6 + 1896946429*x^5 + 20383596*x + 593179173
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
```
Try all Galois groups over $\mathbb{Q}$ in degree 5 except for $S_5$ (the latter is infeasible with the current implementation):

\begin{verbatim}
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 + 3585*x^12 + 2965*x^11 + 2250*x^10 + 1625*x^9 + 1150*x^8 + 750*x^7 + 400*x^6 + 275*x^5 + 100*x^4 + 25
sage: A5pol = x^5 - x^4 + 2*x^2 - 2*x + 2
sage: A5pol.splitting_field('x')
Number Field in x with defining polynomial x^60 ...
\end{verbatim}

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):

\begin{verbatim}
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)
... SplittingFieldAbort: degree of splitting field is a multiple of 180
\end{verbatim}

Use the `degree_divisor` attribute to recover the divisor of the degree of the splitting field or `degree_multiple` to recover a multiple:

\begin{verbatim}
sage: from sage.rings.number_field.splitting_field import SplittingFieldAbort
sage: try:  # long time (4s on sage.math, 2014)
...: (x^8+x+1).splitting_field('b', abort_degree=60, simplify=False)
...: except SplittingFieldAbort as e:
...: print(e.degree_divisor)
...: print(e.degree_multiple)
120
1440
\end{verbatim}
1.7 Galois Groups of Number Fields

AUTHORS:

- David Loeffler (2009): rewrite to give explicit homomorphism groups

sage.rings.number_field.galois_group.GaloisGroup
alias of sage.rings.number_field.galois_group.GaloisGroup_v1

class sage.rings.number_field.galois_group.GaloisGroupElement
Bases: sage.groups.perm_gps.permgroup_element.PermutationGroupElement

An element of a Galois group. This is stored as a permutation, but may also be made to act on elements of the field (generally returning elements of its Galois closure).

EXAMPLES:

```sage
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]
(1,5)(2,4)(3,6)
sage: G[4](v)
1/18*y^4
sage: G[4](G[4](v))
-1/36*y^4 - 1/2*y
sage: G[4](G[4](G[4](v)))
1/18*y^4
```

as_hom()

Return the homomorphism L \rightarrow L corresponding to self, where L is the Galois closure of the ambient number field.

EXAMPES:

```sage
sage: G = QuadraticField(-7, 'w').galois_group()
sage: G[1].as_hom()
Ring endomorphism of Number Field in w with defining polynomial x^2 + 7 with w
-> 2.645751311064591?*I
    Defn: w |--> -w
```

ramification_degree(P)

Return the greatest value of v such that s acts trivially modulo P^v. Should only be used if P is prime and s is in the decomposition group of P.

EXAMPLES:

```sage
sage: K.<b> = NumberField(x^3 - 3, 'a').galois_closure()
sage: G = K.galois_group()
sage: P = K.primes_above(3)[0]
sage: s = hom(K, K, 1/18*b^4 - 1/2*b)
```

(continues on next page)
class sage.rings.number_field.galois_group.GaloisGroup_subgroup(ambient, gens=None, gap_group=None, domain=None, category=None, canonicalize=True, check=True)

Bases: sage.groups.galois_group.GaloisSubgroup_perm

A subgroup of a Galois group, as returned by functions such as decomposition_group.

INPUT:

• ambient – the ambient Galois group

• gens – a list of generators for the group

• gap_group – a gap or libgap permutation group, or a string defining one (default: None)

• domain – a set on which this permutation group acts; extracted from ambient if not specified

• category – the category for this object

• canonicalize – if true, sorts and removes duplicates

• check – whether to check that generators actually lie in the ambient group

EXAMPLES:

sage: from sage.rings.number_field.galois_group import GaloisGroup_subgroup
sage: G = NumberField(x^3 - x - 1, 'a').galois_closure('b').galois_group()
sage: GaloisGroup_subgroup( G, [G([(1,2,3),(4,5,6)])])
Subgroup generated by [(1,2,3)(4,5,6)] of (Galois group 6T2 ([3]2) with order 6 of $x^6 - 6*x^4 + 9*x^2 + 23$)

sage: K.<a> = NumberField(x^6-3*x^2-1)
sage: L.<b> = K.galois_closure()
sage: G = L.galois_group()
sage: P = L.primes_above(3)[0]
sage: H = G.decomposition_group(P)
sage: H.order()
3

sage: G = NumberField(x^3 - x - 1, 'a').galois_closure('b').galois_group()
sage: H = G.subgroup([G([(1,2,3),(4,5,6)])])
sage: H
Subgroup generated by [(1,2,3)(4,5,6)] of (Galois group 6T2 ([3]2) with order 6 of $x^6 - 6*x^4 + 9*x^2 + 23$)

Element

alias of GaloisGroupElement

fixed_field(name=None, polred=None, threshold=None)

Return the fixed field of this subgroup (as a subfield of the Galois closure of the number field associated to the ambient Galois group).

INPUT:

• name – a variable name for the new field.

1.7. Galois Groups of Number Fields

195
• **polred** – whether to optimize the generator of the newly created field for a simpler polynomial, using pari’s polredbest. Defaults to True when the degree of the fixed field is at most 8.

• **threshold** – positive number; polred only performed if the cost is at most this threshold

**EXAMPLES:**

```
sage: L.<a> = NumberField(x^4 + 1)
sage: G = L.galois_group()
sage: H = G.decomposition_group(L.primes_above(3)[0])
sage: H.fixed_field()
(Number Field in a0 with defining polynomial x^2 + 2 with a0 = a^3 + a,
 Ring morphism:
   From: Number Field in a0 with defining polynomial x^2 + 2 with a0 = a^3 + a
   To:   Number Field in a with defining polynomial x^4 + 1
   Defn: a0 |--> a^3 + a)
```

You can use the polred option to get a simpler defining polynomial:

```
sage: K.<a> = NumberField(x^5 - 5*x^2 - 3)
sage: G = K.galois_group(); G
Galois group 5T2 (5:2) with order 10 of x^5 - 5*x^2 - 3
sage: sigma, tau = G.gens()
sage: H = G.subgroup([tau])
sage: H.fixed_field(polred=False)
(Number Field in a0 with defining polynomial x^2 + 84375 with a0 = 5*ac^5 + 25*ac^3,
 Ring morphism:
   From: Number Field in a0 with defining polynomial x^2 + 84375 with a0 = 5*ac^5 + 25*ac^3
   To:   Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + 3375
   Defn: a0 |--> 5*ac^5 + 25*ac^3)
sage: H.fixed_field(polred=True)
(Number Field in a0 with defining polynomial x^2 - x + 4 with a0 = -1/30*ac^5 - 1/6*ac^3 + 1/2,
 Ring morphism:
   From: Number Field in a0 with defining polynomial x^2 - x + 4 with a0 = -1/30*ac^5 - 1/6*ac^3 + 1/2
   To:   Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + 3375
   Defn: a0 |--> -1/30*ac^5 - 1/6*ac^3 + 1/2)
sage: G.splitting_field()
Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + 3375
```

An embedding is returned also if the subgroup is trivial (trac ticket #26817):

```
sage: H = G.subgroup([])
sage: H.fixed_field()
(Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + 3375,
 Identity endomorphism of Number Field in ac with defining polynomial x^10 + 10*x^8 + 25*x^6 + 3375)
```

```python
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 G.group(), and to get the corresponding number field do G.number_field(). For a more sophisticated interface use the type=None option.

EXAMPLES:

```python
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: K = QQ[2^(1/3)]
 sage: G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True), K);
˓→G
...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://trac.sagemath.org/28782 for details.
Galois group PARI group [6, -1, 2, "S3"] of degree 3 of the Number Field in a with␣
˓→defining polynomial x^3 - 2 with a = 1.259921049894873?

sage: G.order()
6
sage: G.group()
PARI group [6, -1, 2, "S3"] of degree 3
sage: G.number_field()
Number Field in a with defining polynomial x^3 - 2 with a = 1.259921049894873?
```

group()

Return the underlying abstract group.

EXAMPLES:

```python
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: K = NumberField(x^6 + 2, 'theta')
 sage: G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True),␣
˓→K)
...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://trac.sagemath.org/28782 for details.

sage: H = G.group(); H
PARI group [6, -1, 2, "S3"] of degree 3
sage: P = H.permutation_group(); P
Transitive group number 2 of degree 3
sage: sorted(P)
[((), (2,3), (1,2), (1,2,3), (1,3,2), (1,3)]
```

number_field()

Return the number field of which this is the Galois group.

EXAMPLES:

```python
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: K = NumberField(x^6 + 2, 't')
 sage: G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True),␣
˓→K); G
...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://trac.sagemath.org/28782 for details.
Galois group PARI group [12, -1, 3, "D(6) = S(3)[x]2"] of degree 6 of the␣
˓→Number Field in t with defining polynomial x^6 + 2
sage: G.number_field()
Number Field in t with defining polynomial x^6 + 2
```
order()

Return the order of this Galois group.

EXAMPLES:

```
sage: from sage.rings.number_field.galois_group import GaloisGroup_v1
sage: K = NumberField(x^5 + 2, 'theta_1')
sage: G = GaloisGroup_v1(K.absolute_polynomial().galois_group(pari_group=True), K);
G
...DeprecationWarning: GaloisGroup_v1 is deprecated; please use GaloisGroup_v2
See https://trac.sagemath.org/28782 for details.
Galois group PARI group [20, -1, 3, "F(5) = 5:4"] of degree 5 of the Number Field in theta_1 with defining polynomial x^5 + 2
sage: G.order()
20
```

class sage.rings.number_field.galois_group.GaloisGroup_v2(number_field, algorithm='pari', names=None, gc_numbering=None, _type=None)

Bases: sage.groups.galois_group.GaloisGroup_perm

The Galois group of an (absolute) number field.

Note: We define the Galois group of a non-normal field \( K \) to be the Galois group of its Galois closure \( L \), and elements are stored as permutations of the roots of the defining polynomial of \( L \), not as permutations of the roots (in \( L \)) of the defining polynomial of \( K \). The latter would probably be preferable, but is harder to implement. Thus the permutation group that is returned is always simply-transitive.

The ‘arithmetical’ features (decomposition and ramification groups, Artin symbols etc) are only available for Galois fields.

EXAMPLES:

```
sage: G = NumberField(x^3 - x - 1, 'a').galois_closure('b').galois_group()
sage: G.subgroup([G([1,2,3],[4,5,6])])
Subgroup generated by [(1,2,3)(4,5,6)] of (Galois group 6T2 ([3]2) with order 6 of \( \mathbb{Q}(a) \))
\( x^6 - 6x^4 + 9x^2 + 23 \)
```

Subgroups can be specified using generators (trac ticket #26816):

```
sage: K.<a> = NumberField(x^6 - 6*a^4 + 9*a^2 + 23)
sage: G = K.galois_group()
sage: list(G)
[(),
 (1,2,3)(4,5,6),
 (1,3,2)(4,6,5),
 (1,4)(2,6)(3,5),
 (1,5)(2,4)(3,6),
 (1,6)(2,5)(3,4)]
sage: g = G[1]
sage: h = G[3]
sage: list(G.subgroup([]))
[()]
sage: list(G.subgroup([g]))
(continues on next page)
sage: list(G.subgroup([h]))
[(), (1,4)(2,6)(3,5)]

sage: sorted(G.subgroup([g,h])) == sorted(G)
True

Element
alias of GaloisGroupElement

Subgroup
alias of GaloisGroup_subgroup

artin_symbol(P)
Return the Artin symbol \( \left( \frac{\mathbb{K}}{\mathbb{Q}} P \right) \), where \( \mathbb{K} \) 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:

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), (1,4)(2,3)(5,8)(6,7), (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

sage: QuadraticField(-7,'c').galois_group().artin_symbol(13)
(1,2)
sage: G.artin_symbol(K.primes_above(2)[0])
Traceback (most recent call last):
  ... ValueError: Fractional ideal (...) is ramified

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:

sage: L.<z> = CyclotomicField(7)
sage: G = L.galois_group()
sage: conj = G.complex_conjugation(); conj
(1,3)(2,6)(4,5)
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:

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)]

1.7. Galois Groups of Number Fields 199
decomposition_group($P$)
Decomposition group of a prime ideal $P$, i.e. the subgroup of elements that map $P$ to itself. This is the same as the Galois group of the extension of local fields obtained by completing at $P$.

This function will raise an error if $P$ is not prime or the given number field is not Galois.

$P$ can also be an infinite prime, i.e. an embedding into $\mathbb{R}$ or $\mathbb{C}$.

EXAMPLES:

```python
sage: K.<a> = NumberField(x^4 - 2*x^2 + 2,'b').galois_closure()
sage: P = K.ideal([17, a^2])
sage: G = K.galois_group()
sage: G.decomposition_group(P)
Subgroup generated by [(1,8)(2,7)(3,6)(4,5)] of (Galois group 8T4 ([4]2) with order 8 of x^8 - 20*x^6 + 104*x^4 - 40*x^2 + 1156)
sage: G.decomposition_group(P^2)
Traceback (most recent call last):
  ...
ValueError: Fractional ideal (...) is not a prime ideal
sage: G.decomposition_group(17)
Traceback (most recent call last):
  ...
ValueError: Fractional ideal (17) is not a prime ideal
```

An example with an infinite place:

```python
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
```

easy_order($algorithm$=None)
Return the order of this Galois group if it’s quick to compute.

EXAMPLES:

```python
sage: R.<x> = ZZ[]
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group()
sage: G.easy_order()
6
sage: L.<b> = NumberField(x^72 + 2*x + 2)
sage: H = L.galois_group()
sage: H.easy_order()
```

group()
While GaloisGroup_v1 is being deprecated, this provides public access to the Pari/GAP group in order to keep all aspects of that API.

EXAMPLES:

```python
sage: R.<x> = ZZ[]
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group(type="pari")
...
DeprecationWarning: the different Galois types have been merged into one...
```

(continues on next page)
See https://trac.sagemath.org/28782 for details.

**sage:** G.group()

...DeprecationWarning: the group method is deprecated; you can use `_pol_galgp`

→ if you really need it

See https://trac.sagemath.org/28782 for details.

PARI group [6, -1, 2, "S3"] of degree 3

### inertia_group(P)

Return the inertia group of the prime P, i.e. the group of elements acting trivially modulo P. This is just the 0th ramification group of P.

**EXAMPLES:**

```
sage: K.<b> = NumberField(x^2 - 3, 'a')
sage: G = K.galois_group()
sage: G.inertia_group(K.primes_above(2)[0])
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 - 3)
sage: G.inertia_group(K.primes_above(5)[0])
Subgroup generated by [()] of (Galois group 2T1 (S2) with order 2 of x^2 - 3)
```

### is_galois()

Whether the underlying number field is Galois

**EXAMPLES:**

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

### list()

List of the elements of self.

**EXAMPLES:**

```
sage: NumberField(x^3 - 3*x + 1, 'a').galois_group(names='b').list()
[(), (1,2,3), (1,3,2)]
```

### number_field()

The ambient number field.

**EXAMPLES:**

```
sage: K = NumberField(x^3 - x + 1, 'a')
sage: K.galois_group(names='b').number_field() is K
True
```

### order(algorithm=None, recompute=False)

Return the order of this Galois group

**EXAMPLES:**

```
sage: R.<x> = ZZ[]
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group()
```

(continues on next page)
**sage**: G.order()
6

**pari_label()**
Return the label assigned by Pari for this Galois group, an attempt at giving a human readable description of the group.

**EXAMPLES:**

```
sage: R.<x> = ZZ[]
sage: K.<a> = NumberField(x^8 - x^5 + x^4 - x^3 + 1)
sage: G = K.galois_group()
sage: G.transitive_label()  
'8T44'
sage: G.pari_label()  
'[2^4]S(4)'
```

**ramification_breaks**(P)
Return the set of ramification breaks of the prime ideal P, i.e. the set of indices i such that the ramification group \( G_{i+1} \neq G_i \). This is only defined for Galois fields.

**EXAMPLES:**

```
sage: 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:**

```
sage: K.<b> = NumberField(x^3 - 3,'a').galois_closure()
sage: G = K.galois_group()
sage: P = K.primes_above(3)[0]
sage: G.ramification_group(P, 3)  
Subgroup generated by [(1,2,4)(3,5,6)] of (Galois group 6T2 ([3]2) with order 6 of x^6 + 243)
sage: G.ramification_group(P, 5)  
Subgroup generated by [] of (Galois group 6T2 ([3]2) with order 6 of x^6 + 243)
```

**signature()**
Return 1 if contained in the alternating group, -1 otherwise.

**EXAMPLES:**

```
sage: R.<x> = ZZ[]
sage: K.<a> = NumberField(x^3 - 2)
```

(continues on next page)
```
sage: K.galois_group().signature()
-1
sage: K.<a> = NumberField(x^3 - 3*x - 1)
sage: K.galois_group().signature()
1
```

**transitive_number** *(algorithm=None, recompute=False)*

Regardless of the value of gc_numbering, this gives the transitive number for the action on the roots of the defining polynomial of the original number field, not the Galois closure.

**INPUT:**
- algorithm – string, specify the algorithm to be used
- recompute – boolean, whether to recompute the result even if known by another algorithm

**EXAMPLES:**
```
sage: R.<x> = ZZ[]
sage: K.<a> = NumberField(x^3 + 2*x + 2)
sage: G = K.galois_group()
sage: G.transitive_number()
2
sage: L.<b> = NumberField(x^13 + 2*x + 2)
sage: H = L.galois_group(algorithm="gap")
sage: H.transitive_number() # optional - gap_packages
9
```

**unrank(i)**

Return the i-th element of self.

**INPUT:**
- i – integer between 0 and n-1 where n is the cardinality of this set

**EXAMPLES:**
```
sage: G = NumberField(x^3 - 3*x + 1,'a').galois_group()
sage: [G.unrank(i) for i in range(G.cardinality())]
[(1,2,3), (1,3,2)]
```

### 1.8 Elements of bounded height in number fields

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:**
- [DK2013]
sage.rings.number_field.bdd_height.bdd_height($K$, $height\_bound$, $tolerance=0.01$, $precision=53$)

Compute all elements in the number field $K$ which have relative multiplicative height at most $height\_bound$.

The function can only be called for number fields $K$ with positive unit rank. An error will occur if $K$ is $QQ$ 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 then the bound. This can be controlled by lowering the tolerance.

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

ALGORITHM:

This is an implementation of the revised algorithm (Algorithm 4) in [DK2013].

INPUT:

• $height\_bound$ – real number
• $tolerance$ – (default: 0.01) a rational number in (0,1]
• $precision$ – (default: 53) positive integer

OUTPUT:

• an iterator of number field elements

EXAMPLES:

There are no elements of negative height:

```
sage: from sage.rings.number_field.bdd_height import bdd_height
tsage: K.<g> = NumberField(x^5 - x + 7)
tsage: 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
tsage: K.<g> = QuadraticField(3)
tsage: list(bdd_height(K,1))
[0, -1, 1]
```

```
sage: from sage.rings.number_field.bdd_height import bdd_height
tsage: K.<g> = QuadraticField(36865)
tsage: len(list(bdd_height(K,101))) # long time (4 s)
131
```

```
sage: from sage.rings.number_field.bdd_height import bdd_height
tsage: K.<g> = NumberField(x^6 + 2)
tsage: len(list(bdd_height(K,60))) # long time (5 s)
1899
```

```
sage: from sage.rings.number_field.bdd_height import bdd_height
tsage: K.<g> = NumberField(x^4 - x^3 - 3*x^2 + x + 1)
tsage: len(list(bdd_height(K,10)))
99
```
Compute all elements in the imaginary quadratic field $K$ which have relative multiplicative height at most $\text{height} \_\text{bound}$.

The function will only be called with $K$ an imaginary quadratic field.

If called with $K$ not an imaginary quadratic, the function will likely yield incorrect output.

**ALGORITHM:**

This is an implementation of Algorithm 5 in [DK2013].

**INPUT:**

- $K$ – an imaginary quadratic number field
- $\text{height} \_\text{bound}$ – a real number

**OUTPUT:**

- an iterator of number field elements

**EXAMPLES:**

```sage
from sage.rings.number_field.bdd_height import bdd_height_iq
K.<a> = NumberField(x^2 + 191)
for t in bdd_height_iq(K,8):
    print(exp(2*t.global_height()))
```

```
1.00000000000000
1.00000000000000
1.00000000000000
4.00000000000000
4.00000000000000
4.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
8.00000000000000
```

There are 175 elements of height at most 10 in $\mathbb{Q}(\sqrt{-3})$:

```sage
from sage.rings.number_field.bdd_height import bdd_height_iq
K.<a> = NumberField(x^2 + 3)
len(list(bdd_height_iq(K,10)))
```

```
175
```

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

```sage
from sage.rings.number_field.bdd_height import bdd_height_iq
K.<a> = NumberField(x^2 + x + 1)
list(bdd_height_iq(K,1))
```

```
[0, a + 1, a, -1, -a - 1, -a, 1]
```

A number field has no elements of multiplicative height less than 1:
sage: from sage.rings.number_field.bdd_height import bdd_height_iq
sage: K.<a> = NumberField(x^2 + 5)
sage: list(bdd_height_iq(K,0.9))
[]
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
\textbf{sage.rings.number_field.bdd_height.bdd_norm_pr_ideal_gens}(K, \text{norm_list})

Compute generators for all principal ideals in a number field \(K\) whose norms are in \text{norm_list}.

**INPUT:**

- \(K\) – a number field
- \text{norm_list} – a list of positive integers

**OUTPUT:**

- a dictionary of number field elements, keyed by norm

**EXAMPLES:**

There is only one principal ideal of norm 1, and it is generated by the element 1:

\begin{verbatim}
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]}
\end{verbatim}

\begin{verbatim}
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]}
\end{verbatim}

\begin{verbatim}
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]
\end{verbatim}

\textbf{sage.rings.number_field.bdd_height.integer_points_in_polytope}(\text{matrix}, \text{interval_radius})

Return the set of integer points in the polytope obtained by acting on a cube by a linear transformation.

Given an \(r\)-by-\(r\) matrix \text{matrix} and a real number \text{interval_radius}, this function finds all integer lattice points in the polytope obtained by transforming the cube \([-\text{interval_radius}, \text{interval_radius}]^r\) via the linear map induced by \text{matrix}.

**INPUT:**

- \text{matrix} – a square matrix of real numbers
- \text{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:

\begin{verbatim}
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)]
\end{verbatim}
Integer points inside a parallelogram:

```python
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:

```python
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:

```python
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)]
```
2.1 Morphisms between number fields

This module provides classes to represent ring homomorphisms between number fields (i.e. field embeddings).

class sage.rings.number_field.morphism.CyclotomicFieldHomomorphism_im_gens
Bases: sage.rings.number_field.morphism.NumberFieldHomomorphism_im_gens

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)
sage: L.<b> = NumberField(x^4 - 7)
sage: f = K.embeddings(L)[0]
sage: f.preimage(3*b^2 - 12/7)
3*a - 12/7
sage: f.preimage(b)
Traceback (most recent call last):
  ...
ValueError: Element 'b' is not in the image of this homomorphism.
```

```
sage: F.<b> = QuadraticField(23)
sage: G.<a> = F.extension(x^3+5)
sage: f = F.embeddings(G)[0]
sage: f.preimage(a^3+2*b+3)
2*b - 2
```

class sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs(parent, abs_hom)

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.
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]
```

### 2.2 Sets of homomorphisms between number fields

class sage.rings.number_field.homset.CyclotomicFieldHomset(R, S, category=None)
Bases: sage.rings.number_field.homset.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
```

Element

alias of sage.rings.number_field.morphism.CyclotomicFieldHomomorphism_im_gens

list()
Return a list of all the elements of self (for which the domain is a cyclotomic field).

EXAMPLES:

```
sage: K.<z> = CyclotomicField(12)
sage: G = End(K); G
Automorphism group of Cyclotomic Field of order 12 and degree 4
sage: [g(z) for g in G]
z, z^3 - z, -z, -z^3 + z
sage: 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
```

(continues on next page)
class sage.rings.number_field.homset.NumberFieldHomset(R, S, category=None)

Bases: sage.rings.homset.RingHomset_generic

Set of homomorphisms with domain a given number field.

Element

alias of sage.rings.number_field.morphism.NumberFieldHomomorphism_im_gens
cardinality()

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
```

test()  # indirect doctest

Return a list of all the elements of self.

EXAMPLES:

```
sage: K.<a> = NumberField(x^3 - 3*x + 1)
sage: End(K).list()
[
    Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
        Defn: a |--> a,
    Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
        Defn: a |--> a^2 - 2,
    Ring endomorphism of Number Field in a with defining polynomial x^3 - 3*x + 1
        Defn: a |--> -a^2 - a + 2
]
sage: Hom(K, CyclotomicField(9))[[0]

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:

```
Defn: zeta12 |--> -b^2*a
sage: list(Hom(CyclotomicField(5), K))
[]
sage: Hom(CyclotomicField(11), L).list()
[]
```
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
]

order()
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.homset.RelativeNumberFieldHomset(R, S, category=None)
Bases: sage.rings.number_field.homset.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)
3*cuberoot2 + 3*zeta3

In fact, this phi is a generator for the Kummer Galois group of this cyclic extension:
Element
alias of `sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs`

default_base_hom()
Pick an embedding of the base field of self into the codomain of this homset. This is done in an essentially arbitrary way.

EXAMPLES:

```python
sage: L.<a, b> = NumberField([x^3 - x + 1, x^2 + 23])
sage: M.<c> = NumberField(x^4 + 80*x^2 + 36)
sage: Hom(L, M).default_base_hom()
Ring morphism:
  From: Number Field in b with defining polynomial x^2 + 23
  To:   Number Field in c with defining polynomial x^4 + 80*x^2 + 36
  Defn: b |--> 1/12*c^3 + 43/6*c
```

list()
Return a list of all the elements of self (for which the domain is a relative number field).

EXAMPLES:

```python
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,
...
]
```

An example with an absolute codomain:

```python
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,
...
]
```

(continues on next page)
Defn: \(a \mapsto -z^6 + 2z^2\)
\(b \mapsto z^5 + z^3 - z\)

### 2.3 Embeddings into ambient fields

This module provides classes to handle embeddings of number fields into ambient fields (generally \(\mathbb{R}\) or \(\mathbb{C}\)).

#### Class `sage.rings.number_field.number_field_morphisms.CyclotomicFieldConversion`

Bases: `sage.categories.map.Map`

This allows one to cast one cyclotomic field in another consistently.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.number_field_morphisms import CyclotomicFieldConversion
sage: K1.<z1> = CyclotomicField(12)
sage: K2.<z2> = CyclotomicField(18)
sage: f = CyclotomicFieldConversion(K1, K2)
sage: f(z1^2)
z2^3
sage: f(z1)
Traceback (most recent call last):
  ... ValueError: Element z1 has no image in the codomain
```

Tests from trac ticket #29511:

```python
sage: K.<z> = CyclotomicField(12)
sage: K1.<z1> = CyclotomicField(3)
sage: K(2) in K1 # indirect doctest
True
sage: K1(K(2)) # indirect doctest
2
```

#### Class `sage.rings.number_field.number_field_morphisms.CyclotomicFieldEmbedding`

Bases: `sage.rings.number_field.number_field_morphisms.NumberFieldEmbedding`

Specialized class for converting cyclotomic field elements into a cyclotomic field of higher order. All the real work is done by \_lift\_cyclotomic\_element.

**section()**

Return the section of `self`.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.number_field_morphisms import...
   ...CyclotomicFieldEmbedding
sage: K = CyclotomicField(7)
sage: L = CyclotomicField(21)
sage: f = CyclotomicFieldEmbedding(K, L)
sage: h = f.section()
```
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 \( \mathbb{C} \)).

This is done by factoring the minimal polynomial of the input in the number field of the codomain. This may fail if the element is not actually in the given field.

ambient_field

class sage.rings.number_field.number_field_morphisms.EmbeddedNumberFieldMorphism

Bases: 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;
- \( \mathbb{C} \).

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 with i = I
    To: Number Field in i with defining polynomial x^2 + 1 with i = -I
    Defn: i -> -i
sage: EmbeddedNumberFieldMorphism(K,L,QQbar)
Generic morphism:
    From: Number Field in i with defining polynomial x^2 + 1 with i = I
    To: Number Field in i with defining polynomial x^2 + 1 with i = -I
    Defn: i -> -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()
```

2.3. Embeddings into ambient fields
The image contains a page from a mathematics textbook discussing algebraic numbers and number fields. The page includes code examples and explanations of various methods and classes related to embeddings in the context of number fields. The code examples are executed using the `sage` command-line interface, and the output is shown for illustrative purposes. The page is part of a larger section on morphisms in algebraic number theory.
To: Real Lazy Field  
Defn: a -> 0.94564927392359157  
sage: create_embedding_from_approx(K, 0)  
Generic morphism:  
  From: Number Field in a with defining polynomial x^3 - x + 1/10  
  To: Real Lazy Field  
Defn: a -> 0.101031257881010815  
sage: create_embedding_from_approx(K, -1)  
Generic morphism:  
  From: Number Field in a with defining polynomial x^3 - x + 1/10  
  To: Real Lazy Field  
Defn: a -> -1.0466805318046037  

We can define embeddings from one number field to another:  
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  

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: 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^3-1, CDF.0)  
-0.50000000000000000 + 0.86602540378443817*I  
sage: matching_root(x^3-x, 2, ambient_field=RR)  
1.00000000000000  

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
def root_from_approx(f, a):
    from sage.rings.number_field.number_field_morphisms import root_from_approx
    return root_from_approx(f, a)

sage: R.<x> = QQ[]
sage: root_from_approx(x^2 - 1, -1)
-1
sage: root_from_approx(x^2 - 2, 1)
1.414213562373095?
```

2.4 Structure maps for number fields

Provides isomorphisms between relative and absolute presentations, to and from vector spaces, name changing maps, etc.

EXAMPLES:

```sage
L.<cuberoot2, zeta3> = CyclotomicField(3).extension(x^3 - 2)
sage: K = L.absolute_field('a')
sage: from_K, to_K = K.structure()
sage:
Isomorphism map:
  From: Number Field in a with defining polynomial x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 - 3*x + 1
  To:   Number Field in cuberoot2 with defining polynomial x^3 - 2 over its base field
```
sage: to_K
Isomorphism map:
  From: Number Field in cuberoot2 with defining polynomial x^3 - 2 over its base field
  To: Number Field in a with defining polynomial x^6 - 3*x^5 + 6*x^4 - 11*x^3 + 12*x^2 + 3*x + 1

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 Q-vector space.

EXAMPLES:

sage: L.<a> = NumberField(x^3 - x + 1)
sage: V, fr, to = L.vector_space()
sage: type(to)
<class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>

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 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: to
Isomorphism map:
  From: Number Field in b with defining polynomial x^2 + a0 over its base field
  To: Vector space of dimension 8 over Rational Field
Vector space of dimension 8 over Rational Field

```python
sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapVectorSpaceToRelativeNumberField'>,
 <class 'sage.rings.number_field.maps.MapRelativeNumberFieldToVectorSpace'>)
```

```python
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:

```python
sage: K.<a> = NumberField(x^6 + 4*x^2 + 200)

sage: L = K.relativize(K.subfields(3)[0][1], 'b'); L
Number Field in b with defining polynomial x^2 + a0 over its base field

sage: fr, to = L.structure()

sage: fr
Relative number field morphism:
    From: Number Field in b with defining polynomial x^2 + a0 over its base field
    To:   Number Field in a with defining polynomial x^6 + 4*x^2 + 200
    Defn: b |--> a
           a0 |--> -a^2

sage: to
Ring morphism:
    From: Number Field in a with defining polynomial x^6 + 4*x^2 + 200
    To:   Number Field in b with defining polynomial x^2 + a0 over its base field
    Defn: a |--> b

sage: type(fr), type(to)
(<class 'sage.rings.number_field.homset.RelativeNumberFieldHomset_with_category.element_class'>,
 <class 'sage.rings.number_field.homset.NumberFieldHomset_with_category.element_class'>)
```

```python
sage: M.<c> = L.absolute_field(); M
Number Field in c with defining polynomial x^6 + 4*x^2 + 200

sage: fr, to = M.structure()

sage: fr
Isomorphism map:
    From: Number Field in c with defining polynomial x^6 + 4*x^2 + 200
    To:   Number Field in b with defining polynomial x^2 + a0 over its base field

sage: to
Isomorphism map:
    From: Number Field in b with defining polynomial x^2 + a0 over its base field
    To:   Number Field in c with defining polynomial x^6 + 4*x^2 + 200

sage: type(fr), type(to)
(<class 'sage.rings.number_field.maps.MapAbsoluteToRelativeNumberField'>,
 <class 'sage.rings.number_field.maps.MapRelativeToAbsoluteNumberField'>)

sage: fr(M.gen()), to(fr(M.gen())) == M.gen()
(b, True)
```

(continues on next page)
\texttt{sage}: \texttt{to(L.gen())}, \texttt{fr(to(L.gen()))} == \texttt{L.gen()}
(c, True)
\texttt{sage}: \texttt{(to * fr)(M.gen())} == \texttt{M.gen()}, \texttt{(fr * to)(L.gen())} == \texttt{L.gen()}
(True, True)

\texttt{class sage.rings.number_field.maps.MapRelativeVectorSpaceToRelativeNumberField(V, K)}

Bases: \texttt{sage.rings.number_field.maps.NumberFieldIsomorphism}

\textbf{EXAMPLES:}

\texttt{sage: L.<a> = NumberField(x^4 + 3*x^2 + 1)}
\texttt{sage: K = L.relativize(L.subfields(2)[0][1], 'a'); K}
Number Field in a with defining polynomial \(x^2 - b^0 x + 1\) over its base field
\texttt{sage: V, fr, to = K.relative_vector_space()}
\texttt{sage: V}
Vector space of dimension 2 over Number Field in \(b^0\) with defining polynomial \(x^2 + 1\)
\texttt{sage: fr}
Isomorphism map:
From: Vector space of dimension 2 over Number Field in \(b^0\) with defining polynomial \(x^2 + 1\)
To: Number Field in \(a\) with defining polynomial \(x^2 - b^0 x + 1\) over its base field
\texttt{sage: type(fr)}
<class 'sage.rings.number_field.maps.MapRelativeVectorSpaceToRelativeNumberField'>
\texttt{sage: a0 = K.gen(); b0 = K.base_field().gen()}
\texttt{sage: fr(to(a0 + 2*b0)), fr(V([0, 1])), fr(V([b0, 2*b0]))}
(a + 2*b0, a, 2*b0*a + b0)
\texttt{sage: (fr * to)(K.gen()) == K.gen()}
True
\texttt{sage: (to * fr)(V([1, 2])) == V([1, 2])}
True

\texttt{class sage.rings.number_field.maps.MapVectorSpaceToNumberField(V, K)}

Bases: \texttt{sage.rings.number_field.maps.NumberFieldIsomorphism}

The map to an absolute number field from its underlying \(\mathbb{Q}\)-vector space.

\textbf{EXAMPLES:}

\texttt{sage: K.<a> = NumberField(x^4 + 3*x^2 + 1)}
\texttt{sage: V, fr, to = K.vector_space()}
\texttt{sage: V}
Vector space of dimension 4 over Rational Field
\texttt{sage: fr}
Isomorphism map:
From: Vector space of dimension 4 over Rational Field
To: Number Field in \(a\) with defining polynomial \(x^4 + 3*x + 1\)
\texttt{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
\texttt{sage: type(fr), type(to)}
(<class 'sage.rings.number_field.maps.MapVectorSpaceToNumberField'>, <class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>)
<class 'sage.rings.number_field.maps.MapNumberFieldToVectorSpace'>

sage: fr.is_injective(), fr.is_surjective()
(True, True)

sage: fr.domain(), to.codomain()
(Vector space of dimension 4 over Rational Field, Vector space of dimension 4 over Rational Field)

sage: to.domain(), fr.codomain()
(Number Field in a with defining polynomial x^4 + 3*x + 1, Number Field in a with defining polynomial x^4 + 3*x + 1)

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: 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 + 3*x + 1
          then
            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: to(a), to(a + 1)
((0, 1, 0, 0), (1, 1, 0, 0))

sage: fr(to(a)), fr(V([0, 1, 2, 3]))
(a, 3*a^3 + 2*a^2 + a)

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 Q-vector space. Compare MapRelativeVectorSpaceToRelativeNumberField.

EXAMPLES:

sage: L.<a, b> = NumberField([x^2 + 3, x^2 + 5])
sage: V, fr, to = L.absolute_vector_space()
sage: type(fr)
<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:**

```
sage: K.<a> = NumberField(x^2 - 3)
sage: L.<b> = K.change_names()
sage: from_L, to_L = L.structure()
sage: from_L
Isomorphism given by variable name change map:
    From: Number Field in b with defining polynomial x^2 - 3
    To: Number Field in a with defining polynomial x^2 - 3
sage: to_L
Isomorphism given by variable name change map:
    From: Number Field in a with defining polynomial x^2 - 3
    To: Number Field in b with defining polynomial x^2 - 3
sage: type(from_L), type(to_L)
(<class 'sage.rings.number_field.maps.NameChangeMap'>, <class 'sage.rings.number_field.maps.NameChangeMap'>)
```

**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:**

```
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: isinstance(fr, sage.rings.number_field.maps.NumberFieldIsomorphism)
True
```

**is_injective()**

**EXAMPLES:**

```
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_injective()
True
```

**is_surjective()**

**EXAMPLES:**

```
sage: K.<a> = NumberField(x^4 + 3*x + 1)
sage: V, fr, to = K.vector_space()
sage: fr.is_surjective()
True
```
2.5 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 with a = 1.414213562373095?,
 Identity endomorphism of Number Field in a with defining polynomial x^2 - 2 with a = 1.414213562373095?)
sage: M.structure()
(Isomorphism given by variable name change map:
 From: Number Field in a with defining polynomial x^2 - 2
 To: Number Field in a with defining polynomial x^2 - 2 with a = 1.414213562373095?,
 Isomorphism given by variable name change map:
 From: Number Field in a with defining polynomial x^2 - 2 with a = 1.414213562373095?
 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:

```sage
sage: L is M
False
sage: N.<a> = L.absolute_field()
sage: M is N
True
```

**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.

```python
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.

```python
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.

```python
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

```python
create_structure(field)
```
Return a pair of isomorphisms which go from `field` to `other` and vice versa.

```python
class sage.rings.number_field.structure.RelativeFromRelative(other)
```
Bases: `sage.rings.number_field.structure.NumberFieldStructure`

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_`.

```python
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
ORDERS, IDEALS, IDEAL CLASSES

3.1 Orders in Number Fields

AUTHORS:
• William Stein and Robert Bradshaw (2007-09): initial version

EXAMPLES:
We define an absolute order:

\[
\text{sage: } K.<a> = \text{NumberField}(x^2 + 1); \ O = K.\text{order}(2^a)
\]
\[
\text{sage: } O.\text{basis()}
\]
\[
[1, 2^a]
\]

We compute a basis for an order in a relative extension that is generated by 2 elements:

\[
\text{sage: } K.<a,b> = \text{NumberField}([x^2 + 1, x^2 - 3]); \ O = K.\text{order}([3^a, 2^b])
\]
\[
\text{sage: } O.\text{basis()}
\]
\[
[1, 3^a - 2^b, -6b^a + 6, 3^a]
\]

We compute a maximal order of a degree 10 field:

\[
\text{sage: } K.<a> = \text{NumberField}((x+1)^10 + 17)
\]
\[
\text{sage: } K.\text{maximal_order()}
\]
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:

\[
\text{sage: } O = K.\text{order}(17^a); \ O
\]
Order in Number Field in a with defining polynomial x^10 + 10*x^9 + 45*x^8 + 120*x^7 + ...
\[
-210*x^6 + 252*x^5 + 210*x^4 + 120*x^3 + 45*x^2 + 10*x + 18
\]
\[
\text{sage: } m = O.\text{index_in}(K.\text{maximal_order()}); \ m
\]
2345316516532778911665591944416226304630809183732482257
\[
\text{sage: } \text{factor}(m)
\]
17^45

class sage.rings.number_field.order.AbsoluteOrderFactory
    Bases: sage.rings.number_field.order.OrderFactory

    An order in an (absolute) number field.

    EXAMPLES:
sage: K.<i> = NumberField(x^2 + 1)
sage: K.order(i)
Order in Number Field in i with defining polynomial x^2 + 1

create_key_and_extra_args(K, module_rep, is_maximal=None, check=True, is_maximal_at=())
Return normalized arguments to create an absolute order.

create_object(version, key, is_maximal=None, is_maximal_at=())
Create an absolute order.

reduce_data(order)
Return the data that can be used to pickle an order created by this factory.

This overrides the default implementation to update the latest knowledge about primes at which the order
is maximal.

EXAMPLES:
This also works for relative orders since they are wrapping absolute orders:

sage: L.<a, b> = NumberField([x^2 - 1000003, x^2 - 5*1000099^2])
sage: O = L.maximal_order([5], assume_maximal=None)

sage: s = dumps(O)
sage: loads(s) is O
True

sage: N = L.maximal_order([7], assume_maximal=None)
sage: dumps(N) == s
False

sage: loads(dumps(N)) is O
True

sage.rings.number_field.order.EisensteinIntegers(names='omega')
Return the ring of Eisenstein integers.

This is the ring of all complex numbers of the form \( a + b\omega \) with \( a \) and \( b \) integers and \( \omega = (-1 + \sqrt{-3})/2 \).

EXAMPLES:

sage: R.<omega> = EisensteinIntegers()
sage: R
Eisenstein Integers in Number Field in omega with defining polynomial x^2 + x + 1˓→ with omega = -0.5000000000000000? + 0.8660254037844387?I

sage: factor(3 + omega)
(-1) * (-omega - 3)
sage: CC(omega)
-0.500000000000000 + 0.866025403784439*I

sage: omega.minpoly()
x^2 + x + 1

sage: EisensteinIntegers().basis()
[1, omega]

sage.rings.number_field.order.EquationOrder(f, names, **kwds)
Return the equation order generated by a root of the irreducible polynomial \( f \) or list 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
```

`sage.rings.number_field.order.GaussianIntegers(names='I', latex_name='i')`

Return the ring of Gaussian integers.

This is the ring of all complex numbers of the form \( a + bI \) with \( a \) and \( b \) integers and \( I = \sqrt{-1} \).

**EXAMPLES:**

```python
sage: ZZI.<I> = GaussianIntegers()
sage: ZZI
Gaussian Integers in Number Field in I with defining polynomial x^2 + 1 with I = 1*I
sage: factor(3 + I)
(-I) * (I + 1) * (2*I + 1)
sage: CC(I)
1.00000000000000*I
sage: I.minpoly()
x^2 + 1
sage: GaussianIntegers().basis()
[1, I]
```

**class** `sage.rings.number_field.order.Order(K)`

Bases: `sage.rings.ring.IntegralDomain`, `sage.rings.abc.Order`

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:**

```python
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
```

(continues on next page)
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
Maximal 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()
Return the absolute degree of this order, i.e., 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
sage: o.basis()
[1, 389*z]
sage: o.ambient()
Number Field in z with defining polynomial x^2 - 389

basis()
Return a basis over \( \mathbb{Z} \) of this order.

EXAMPLES:

sage: K.<a> = NumberField(x^3 + x^2 - 16*x + 16)
sage: O = K.maximal_order(); O
Maximal Order in Number Field in a with defining polynomial x^3 + x^2 - 16*x + 16
sage: O.basis()
class_group(proof=None, names='c')

Return the class group of this order.

(Currently only implemented for the maximal order.)

EXAMPLES:

```sage
k.<a> = NumberField(x^2 + 5077)
k = k.maximal_order(); O
Maximal Order in Number Field in a with defining polynomial x^2 + 5077
O.class_group()
Class group of order 22 with structure C22 of Number Field in a with defining
...polynomial x^2 + 5077
```

class_number(proof=None)

Return the class number of this order.

EXAMPLES:

```sage
ZZ[2^(1/3)].class_number()
1
sage: QQ[sqrt(-23)].maximal_order().class_number()
3
sage: ZZ[120*sqrt(-23)].class_number()
288
```

Note that non-maximal orders are only supported in quadratic fields:

```sage
sage: ZZ[120*sqrt(-23)].class_number()
288
sage: ZZ[100*sqrt(3)].class_number()
4
sage: ZZ[11*2^(1/3)].class_number()
Traceback (most recent call last):
... NotImplementedError: computation of class numbers of non-maximal orders not in
... quadratic fields is not implemented
```

coordinates(x)

Return the coordinate vector of x with respect to this order.

INPUT:

• x – an element of the number field of this order.

OUTPUT:

A vector of length n (the degree of the field) giving the coordinates of x with respect to the integral basis of the order. In general this will be a vector of rationals; it will consist of integers if and only if x is in the order.

AUTHOR: John Cremona 2008-11-15

ALGORITHM:
Uses linear algebra. The change-of-basis matrix is cached. Provides simpler implementations for \_contains\_(), \_is\_integral() and \_smallest\_integer().

EXAMPLES:

```python
sage: K.<i> = QuadraticField(-1)
sage: OK = K.ring_of_integers()
sage: OK_basis = OK.basis(); OK_basis
[1, i]
sage: a = 23-14*i
sage: acoords = OK.coordinates(a); acoords
(23, -14)
sage: sum([OK_basis[j]*acoords[j] for j in range(2)]) == a
True
sage: OK.coordinates((120+340*i)/8)
(15, 85/2)
sage: 0 = K.order(3*i)
sage: 0.is_maximal()
False
sage: 0.index_in(OK)
3
sage: acoords = 0.coordinates(a); acoords
(23, -14/3)
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 \textbf{Z}-module.

EXAMPLES:

```python
sage: k.<c> = NumberField(x^3 + x^2 - 2*x+8)
sage: o = k.maximal_order()
sage: o.degree()
3
sage: o.rank()
3
```

\textbf{fraction_field()}

Return the fraction field of this order, which is the ambient number field.

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
```

\textbf{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 \( \mathbb{Z} \)-module contained in the vector space associated to the ambient number field, that corresponds to this order.

**EXAMPLES:**

```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.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 \( \mathbb{Z} \)-module in the absolute_field associated to the relative field:

```python
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:**

```python
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
sage: O.gen(5)
Traceback (most recent call last):
... IndexError: no 5th generator
sage: O.gen(-1)
```

(continues on next page)
Traceback (most recent call last):
...
IndexError: no -1th generator

\textbf{ideal}(\textit{\*args, **kwds})

Return the integral ideal with given generators.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: K.<a> = NumberField(x^2 + 7)
sage: R = K.maximal_order()
sage: R.ideal(2/3 + 7*a, a)
Traceback (most recent call last):
...
ValueError: ideal must be integral; use fractional_ideal to create a non-
→integral ideal.
sage: R.ideal(7*a, 77 + 28*a)
Fractional ideal (7)
sage: R = K.order(4*a)
sage: R.ideal(8)
Traceback (most recent call last):
...
NotImplementedError: ideals of non-maximal orders not yet supported.
\end{verbatim}

This function is called implicitly below:

\begin{verbatim}
sage: R = EquationOrder(x^2 + 2, 'a'); R
Maximal Order in Number Field in a with defining polynomial x^2 + 2
sage: (3,15)*R
Fractional ideal (3)
\end{verbatim}

The zero ideal is handled properly:

\begin{verbatim}
sage: R.ideal(0)
Ideal (0) of Number Field in a with defining polynomial x^2 + 2
\end{verbatim}

\textbf{integral\_closure()}

Return the integral closure of this order.

\textbf{EXAMPLES:}

\begin{verbatim}
sage: K.<a> = QuadraticField(5)
sage: O2 = K.order(2*a); O2
Order in Number Field in a with defining polynomial x^2 - 5 with a = 2.
...
236067977499790?
sage: O2.integral_closure()
Maximal Order in Number Field in a with defining polynomial x^2 - 5 with a = 2.
...
236067977499790?
sage: OK = K.maximal_order()
sage: OK.is_OK.integral\_closure()
True
\end{verbatim}

\textbf{is\_field(\textit{proof}=\texttt{True})}

Return False (because an order is never a field).
EXAMPLES:

```
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:

```
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_noetherian()

Return True (because orders are always Noetherian)

EXAMPLES:

```
sage: L.<alpha> = NumberField(x**4 - x**2 + 7)
sage: O = L.maximal_order() ; O.is_noetherian()
True
sage: E.<w> = NumberField(x^2 - x + 2)
sage: OE = E.ring_of_integers(); OE.is_noetherian()
True
```

is_suborder(other)

Return True if self and other are both orders in the same ambient number field and self is a subset of other.

EXAMPLES:

```
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: W2.<j> = NumberField(x^2 + 1)
sage: P5 = W2.order(5*j)
```

3.1. Orders in Number Fields
This is False because the ambient number fields are not equal:

```
sage: 05.is_suborder(P5)
False
```

We create a field that contains (in no natural way!) \( W \), and of course again is_suborder returns False:

```
sage: K.<z> = NumberField(x^4 + 1)
sage: M = K.order(5*z)
sage: 05.is_suborder(M)
False
```

**krull_dimension()**

Return the Krull dimension of this order, which is 1.

**EXAMPLES:**

```
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: 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:**

```
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
```

```python
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()
True
sage: OK.random_element().parent() is OK
True
```

A relative example:

```python
sage: K.<a, b> = NumberField([x^2 + 2, x^2 + 1000*x + 1])
sage: OK = K.ring_of_integers()
sage: OK.random_element() # random output
(42221/2*b + 61/2)*a + 7037384*b + 7041
sage: OK.random_element().is_integral() # random output
True
sage: OK.random_element().parent() is OK # random output
True
```

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
```

3.1. Orders in Number Fields
**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:**

```sage
k.<c> = NumberField(x^5 + x^2 + 1)
O = k.maximal_order(); O
Maximal Order in Number Field in c with defining polynomial x^5 + x^2 + 1
sage: O.rank()
5
```

**residue_field(prime, names=None, check=False)**

Return the residue field of this order at a given prime, i.e. \(O/\mathfrak{p}O\).

**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:**

```sage
R.<x> = QQ[
K.<a> = NumberField(x^4+3*x^2-17)
P = K.ideal(61).factor()[0][0]
OK = K.maximal_order()
OK.residue_field(P)
Residue field in abar of Fractional ideal (61, a^2 + 30)
Fp.<b> = OK.residue_field(P)
Fp
Residue field in b of Fractional ideal (61, a^2 + 30)
```

**ring_generators()**

Return generators for self as a ring.

**EXAMPLES:**

```sage
K.<i> = NumberField(x^2 + 1)
O = K.maximal_order(); O
Gaussian Integers in Number Field in i with defining polynomial x^2 + 1
sage: O.ring_generators()
[i]
```

This is an example where 2 generators are required (because 2 is an essential discriminant divisor):

```sage
K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
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 with I \rightarrow 1*I
sage: G.some_elements()
[1, I, 2*I, -1, 0, -I, 2, 4*I, -2, -2*I, -4]
```

```python
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 + 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
```

```python
sage: G = GaussianIntegers().valuation(2)
2-adic valuation
```

```python
sage: G = GaussianIntegers().valuation(3)
3-adic valuation
```

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

```python
sage: GaussianIntegers().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
```

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
→ 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[]
sage: v = GaussianIntegers().valuation(GaussValuation(R, QQ.valuation(5)).
→ augmentation(x + 2, infinity))
sage: w = GaussianIntegers().valuation(GaussValuation(R, QQ.valuation(5)).
→ augmentation(x + 1/2, infinity))
sage: v == w
False

See also:

* `NumberField_generic.valuation()`, `pAdicGeneric.valuation()`

`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)
sage: F.ring_of_integers().zeta(6)
1/2*alpha + 1/2
sage: O = F.order([3*alpha])
sage: O.zeta(3)
Traceback (most recent call last):
  ... ArithmeticError: There are no 3rd roots of unity in self.

class `sage.rings.number_field.order.OrderFactory`

Bases: `sage.structure.factory.UniqueFactory`

Abstract base class for factories creating orders, such as `AbsoluteOrderFactory` and `RelativeOrderFactory`.

`get_object(version, key, extra_args)`

Create the order identified by `key`.

This overrides the default implementation to update the maximality of the order if it was explicitly specified.

EXAMPLES:

Even though orders are unique parents, this lets us update their internal state when they are recreated with
more additional information available about them:
sage: L.<a, b> = NumberField([x^2 - 1000003, x^2 - 5*1000099^2])
sage: O = L.maximal_order([2], assume_maximal=None)

sage: O._is_maximal_at(2)
True
sage: O._is_maximal_at(3)  # is None
True

sage: N = L.maximal_order([3], assume_maximal=None)
sage: N
is O
True
sage: N._is_maximal_at(2)
True
sage: N._is_maximal_at(3)
True

class sage.rings.number_field.order.Order_absolute(K, module_rep)
    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
Maximal 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

absolute_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 * 613^2 * 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.

3.1. Orders in Number Fields  241
EXAMPLES:

```
sage: K.<a> = NumberField(x^3 + 2)
sage: O1 = K.order(a); O1
Maximal 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: 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: 0 = K.maximal_order()
```

(continues on next page)
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 is not contained in other.

**INPUT:**
- other – another absolute order with the same ambient number field.

**OUTPUT:**
a rational number

**EXAMPLES:**

```python
sage: k.<i> = NumberField(x^2 + 1)
sage: O1 = k.order(i)
sage: O5 = k.order(5*i)
sage: O5.index_in(O1)
5
sage: k.<a> = NumberField(x^3 + x^2 - 2*x+8)
sage: o = k.maximal_order()
sage: O1 = k.order(a); O1
Order in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1.index_in(o)
2
sage: O2 = k.order(1+2*a); O2
Order in Number Field in a with defining polynomial x^3 + x^2 - 2*x + 8
sage: O1.basis()
[1, a, a^2]
sage: O2.basis()
[1, 2*a, 4*a^2]
sage: o.index_in(O2)
1/16
```

**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)
```

(continues on next page)
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]

is_maximal(p=None)

Return whether this is the maximal order.

INPUT:

- p – an integer prime or None (default: None); if set, return whether this order is maximal at the prime p.

EXAMPLES:

sage: K.<i> = NumberField(x^2 + 1)
sage: K.order(3*i).is_maximal()
False
sage: K.order(5*i).is_maximal()
False
sage: (K.order(3*i) + K.order(5*i)).is_maximal()
True
sage: K.maximal_order().is_maximal()
True

Maximality can be checked at primes when the order is maximal at that prime by construction:

sage: K.maximal_order().is_maximal(p=3)
True

And also at other primes:

sage: K.order(3*i).is_maximal(p=3)
False

An example involving a relative order:

sage: K.<a, b> = NumberField([x^2 + 1, x^2 - 3])
sage: 0 = K.order([3*a, 2*b])
sage: 0.is_maximal()
False

module()

Return the underlying free module corresponding to this order, embedded in the vector space corresponding to the ambient number field.

EXAMPLES:
sage: 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
definitions

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]

class sage.rings.number_field.order.Order_relative(K, absolute_order)
Bases: sage.rings.number_field.order.Order

A relative order in a number field.
A relative order is an order in some relative number field
Invariants of this order may be computed with respect to the contained order.

absolute_discriminant()
Return the absolute discriminant of self, which is the discriminant of the absolute order associated to self.

OUTPUT:
an integer
EXAMPLES:
sage: 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:
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 \( \mathbb{Z} \)-module.

**EXAMPLES:**

```python
sage: K.<a,b> = NumberField([x^2+1, x^2+3])
sage: O = K.order([a,b])
sage: O.basis()
[1, -2*a + b, -b*a - 2, -5*a + 3*b]
sage: z = O.1; z
-2*a + b
sage: z.absolute_minpoly()
x^4 + 14*x^2 + 1
```

**index_in(other)**

Return the index of self in other.

This is a lattice index, so it is a rational number if self is not contained in other.

**INPUT:**

- other – another order with the same ambient absolute number field.

**OUTPUT:**

a rational number

**EXAMPLES:**

```python
sage: 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_maximal(p=None)**

Return whether this is the maximal order.

**INPUT:**

- p – an integer prime or None (default: None); if set, return whether this order is maximal at the prime p.

**EXAMPLES:**
sage: K.<a,b> = NumberField([x^2 + 1, x^2 - 5])
sage: K.order(3*a, b).is_maximal()
False
sage: K.order(5*a, b/2 + 1/2).is_maximal()
False
sage: (K.order(3*a, b) + K.order(5*a, b/2 + 1/2)).is_maximal()
True
sage: K.maximal_order().is_maximal()
True

Maximality can be checked at primes when the order is maximal at that prime by construction:

sage: K.maximal_order().is_maximal(p=3)
True

And at other primes:

sage: K.order(3*a, b).is_maximal(p=3)
False

**is_suborder**(other)
Return 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

**class** sage.rings.number_field.order.RelativeOrderFactory
**Bases:** sage.rings.number_field.order.OrderFactory

An order in a relative number field extension.

**EXAMPLES:**

sage: K.<i> = NumberField(x^2 + 1)
sage: R.<j> = K[]
sage: L.<j> = K.extension(j^2 - 2)
sage: L.order([i, j])
Relative Order in Number Field in j with defining polynomial j^2 - 2 over its base_field

### 3.1. Orders in Number Fields
create_key_and_extra_args\((K, absolute\_order, is\_maximal=None, check=True, is\_maximal\_at=())\)

Return normalized arguments to create a relative order.

create_object\((version, key, is\_maximal=None, is\_maximal\_at=())\)

Create a relative order.

sage.rings.number_field.order.absolute_order_from_module_generators\((gens, check\_integral=True, check\_rank=True, check\_is\_ring=True, is\_maximal=None, allow\_subfield=False, is\_maximal\_at=())\)

INPUT:

- **gens** – list of elements of an absolute number field that generates an order in that number field as a ZZ 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
- **is\_maximal\_at** – a tuple of primes where this order is known to be maximal

OUTPUT:

an absolute order

EXAMPLES:

We have to explicitly import the function, since it is not meant for regular usage:

```
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)
Maximal 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)
Maximal Order in Number Field in a with defining polynomial x^4 - 5
```

(continues on next page)
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 do not check that, we will find that the rank is wrong or that the order is not closed under multiplication:

```
sage: absolute_order_from_module_generators([1/2, i], check_integral=False)
Traceback (most recent call last):
  ... ValueError: the module span of the gens is not closed under multiplication.
sage: R = absolute_order_from_module_generators([1/2, i], check_is_ring=False, ...
Order in Number Field in i with defining polynomial x^2 + 1
sage: R.basis()
[1/2, i]
```

We turn off all check flags and make a really messed up order:

```
sage: R = absolute_order_from_module_generators([1/2, i], check_is_ring=False, ...
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:

```
sage: F.<alpha> = NumberField(x**4+3)
sage: F.order([alpha**2], allow_subfield=True)
Order in Number Field in beta with defining polynomial ... with beta = ...
```

```
sage.rings.number_field.order.absolute_order_from_ring_generators(gens, check_is_integral=True, check_rank=True, is_maximal=None, allow_subfield=False)
```

INPUT:

- `gens` – list of integral elements of an absolute order.
- `check_is_integral` – bool (default: True), whether to check that each generator is integral.
- `check_rank` – bool (default: True), whether to check that the ring generated by gens is of full rank.
• `is_maximal` – bool (or None); set if maximality of the generated order is known

• `allow_subfield` – bool (default: False), if True and the generators do not generate an order, i.e., they generate a subring of smaller rank, instead of raising an error, return an order in a smaller number field.

EXAMPLES:

```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.:

```python
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.:

```python
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.:

```python
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 whether every element of the list `v` of elements of a number field is integral.

**EXAMPLES:**

```python
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=True, check_rank=True, is_maximal=None, allow_subfield=False, is_maximal_at=())

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

EXAMPLES:
We have to explicitly import this function, since it is not 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

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)
Bases: object

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)
number_field_ideal.NumberFieldIdeal, sage.rings.ideal.Ideal_fractional

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
sage: Jinv = I^(-2)
sage: J*Jinv
Fractional ideal (1)

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:

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

divides(other)
Returns True if this ideal divides other and False otherwise.

EXAMPLES:

sage: K.<a> = CyclotomicField(11); K
Cyclotomic Field of order 11 and degree 10
sage: I = K.factor(31)[0][0]; I
Fractional ideal (31, a^5 + 10*a^4 - a^3 + a^2 + 9*a - 1)

(continues on next page)
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:

```python
sage: K.<a> = NumberField(x^3-2)
sage: A = K.ideal(a+1); A; A.norm()
Fractional ideal (a + 1)
3
sage: B = K.ideal(a^2-4*a+2); B; B.norm()
Fractional ideal (a^2 - 4*a + 2)
68
sage: r = A.element_1_mod(B); r
-33
sage: r in A
True
sage: 1-r 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:

```python
sage: K.<i>=NumberField(x^2+1)
sage: I = K.ideal(2+i)
sage: [r for r in I.residues() if I.is_coprime(r)]
[-2*i, -i, i, 2*i]
sage: I.euler_phi()
4
sage: J = I^3
sage: J.euler_phi()
100
sage: len([r for r in J.residues() if J.is_coprime(r)])
```

(continues on next page)
100
sage: J = K.ideal(3-2*i)
sage: I.is_coprime(J)
True
sage: I.euler_phi()*J.euler_phi() == (I*J).euler_phi()
True
sage: L.<b> = K.extension(x^2 - 7)
sage: L.ideal(3).euler_phi()
64

factor()
Factorization of this ideal in terms of prime ideals.

EXAMPLES:

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: F = I.factor(); F
(Fractional ideal (19, 1/2*a^2 + a - 17/2)) * (Fractional ideal (19, 1/2*a^2 - a - 17/2))
sage: type(F)
<class 'sage.structure.factorization.Factorization'>
sage: list(F)
[(Fractional ideal (19, 1/2*a^2 + a - 17/2), 1), (Fractional ideal (19, 1/2*a^2 - a - 17/2), 1)]
sage: F.prod()
Fractional ideal (19)

idealcoprime(J)
Returns l such that l*self is coprime to J.

INPUT:
• J - another integral ideal of the same field as self, which must also be integral.

OUTPUT:
• l - an element such that l*self is coprime to the ideal J

TODO: Extend the implementation to non-integral ideals.

EXAMPLES:

sage: k.<a> = NumberField(x^2 + 23)
sage: A = k.ideal(a+1)
sage: B = k.ideal(3)
sage: A.is_coprime(B)
False
sage: lam = A.idealcoprime(B)
# representation depends, not tested
-1/6*a + 1/6
sage: (lam*A).is_coprime(B)
True

ALGORITHM: Uses Pari function pari:idealcoprime.
\texttt{ideallog}(x, \text{gens=\text{None}, check=True})

Returns the discrete logarithm of $x$ with respect to the generators given in the bid structure of the ideal self, or with respect to the generators \text{gens} if these are given.

INPUT:

- $x$ - a non-zero element of the number field of self, which must have valuation equal to 0 at all prime ideals in the support of the ideal self.
- \text{gens} - a list of elements of the number field which generate $(R/I)^*$, where $R$ is the ring of integers of the field and $I$ is this ideal, or \text{None}. If \text{None}, use the generators calculated by \texttt{idealstar()}.
- \text{check} - if True, do a consistency check on the results. Ignored if \text{gens} is \text{None}.

OUTPUT:

- $l$ - a list of non-negative integers $(x_i)$ such that $x = \prod_i g_i^{x_i}$ in $(R/I)^*$, where $g_i$ are the generators, and the list $(x_i)$ is lexicographically minimal with respect to this requirement. If the $x_i$ generate independent cyclic factors of order $d_i$, as is the case for the default generators calculated by \texttt{idealstar()}, this just means that $0 \leq x_i < d_i$.

A \texttt{ValueError} will be raised if the elements specified in \text{gens} do not in fact generate the unit group (even if the element $x$ is in the subgroup they generate).

**EXAMPLES:**

\begin{verbatim}
sage: k.<a> = NumberField(x^3 - 11)
sage: A = k.ideal(5)
sage: G = A.idealstar(2)
sage: l = A.ideallog(a^2 +3)
sage: r = G(l).value()
sage: (a^2 + 3) - r in A
True
sage: A.small_residue(r)  # random
a^2 - 2
\end{verbatim}

Examples with custom generators:

\begin{verbatim}
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]
\end{verbatim}

A non-example:

\begin{verbatim}
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
\end{verbatim}
**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
Multiplicative Abelian group isomorphic to C24 x C4
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**(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:

```python
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 $R$ representing the elements of $(R/I)^*/U$, where $I$ is this ideal and $U$ is the subgroup of $(R/I)^*$ generated by `subgp_gens`.

EXAMPLES:

```python
sage: k.<a> = NumberField(x^2 + 23)
sage: I = k.ideal(a)
```

(continues on next page)
AUTHOR: Maite Aranes.

**is_S_integral** \(S\)
Return True if this fractional ideal is integral with respect to the list of primes \(S\).

**INPUT:**

- \(S\) - a list of prime ideals (not checked if they are indeed prime).

**Note:** This function assumes that \(S\) is a list of prime ideals, but does not check this. This function will fail if \(S\) is not a list of prime ideals.

**OUTPUT:**

True, if the ideal is \(S\)-integral: that is, if the valuations of the ideal at all primes not in \(S\) are non-negative. False, otherwise.

**EXAMPLES:**

```python
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: 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:**

```
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:

```
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
```

(continues on next page)
\begin{verbatim}
\textbf{is_coprime(j/k)}
False

\textbf{F.<a, b> = NumberField([x^2 - 2, x^2 - 3])}
\textbf{F.ideal(3 - a*b).is_coprime(F.ideal(3))}
False
\end{verbatim}

\textbf{is_maximal()}
Return True if this ideal is maximal. This is equivalent to self being prime, since it is nonzero.

\textbf{EXAMPLES:}
\begin{verbatim}
\textbf{sage: K.<a> = NumberField(x^3 + 3); K}
Number Field in a with defining polynomial x^3 + 3
\textbf{sage: K.ideal(5).is_maximal()}
False
\textbf{sage: K.ideal(7).is_maximal()}
True
\end{verbatim}

\textbf{is_trivial(proof=None)}
Returns True if this is a trivial ideal.

\textbf{EXAMPLES:}
\begin{verbatim}
\textbf{sage: F.<a> = QuadraticField(-5)}
\textbf{sage: I = F.ideal(3)}
\textbf{sage: I.is_trivial()}
False
\textbf{sage: J = F.ideal(5)}
\textbf{sage: J.is_trivial()}
False
\textbf{sage: (I+J).is_trivial()}
True
\end{verbatim}

\textbf{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 $N$.

\textbf{EXAMPLES:}
\begin{verbatim}
\textbf{sage: K.<i>=NumberField(x^2+1)}
\textbf{sage: I = K.ideal((3+4*i)/5); I}
Fractional ideal (4/5*i + 3/5)
\textbf{sage: I.denominator()}
Fractional ideal (2*i + 1)
\textbf{sage: I.numerator()}
Fractional ideal (-i - 2)
\textbf{sage: I.numerator().is_integral() and I.denominator().is_integral()}
True
\textbf{sage: I.numerator() + I.denominator() == K.unit_ideal()}
True
\textbf{sage: I.numerator()/I.denominator() == I}
True
\end{verbatim}
**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:**
```
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:**
```
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)]
sage: I.prime_to_S_part(S)
Fractional ideal (24)
```

**prime_to_idealM_part(M)**
Version for integral ideals of the prime_to_m_part function over $\mathbb{Z}$. 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:

```sage
k.<a> = NumberField(x^2 + 23)
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 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:**

```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 `pari:bnrclassno` function.

**EXAMPLES:**

```sage
K.<z> = QuadraticField(-23)
p = K.primes_above(3)[0]
p.<z> = QuadraticField(-23)
p = K.primes_above(3)[0]
sage: p.<z> = QuadraticField(-23)
p = K.primes_above(3)[0]
sage: p ray_class_number()
3
sage: x = polygen(K)
sage: L.<w> = K.extension(x^3 - z)
sage: I = L.ideal(5)
sage: I ray_class_number()
5184
```

**reduce()**

Return the canonical reduction of the element of \(f\) modulo the ideal \(I (=\text{self})\). 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 \( f + I \) (\( I \) =ideal) 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
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
sage: I.reduce(c) in list(I.residues())
True
```

The reduced element does not necessarily have smaller norm (use `small_residue` for that)

```sage
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):

```sage
sage: k.<a> = NumberField(x^2 + 23)
sage: I = k.ideal(3)
sage: I.reduce(3*a + 1)
-3/2*a - 1/2
sage: k.ring_of_integers().basis()
[1/2*a + 1/2, a]
```

**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()
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
Fractional ideal (389, a^2 - 44*a - 9)
sage: P.residue_class_degree()
2
sage: P.residue_field()
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
Residue field in w of Fractional ideal (389, a^2 - 44*a - 9)
sage: FF((a+1)^390)
36
sage: FF(a)
```

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)
```

(continues on next page)
An example with a relative number field:

```python
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:

```python
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:**

```python
sage: K.<i> = NumberField(x^2+1)
sage: res = K.ideal(2).residues(); res
xmrange_iter([[[0, 1], [0, 1]], <function ...<lambda> at 0x...>])
sage: list(res)
[0, i, 1, i + 1]
sage: list(K.ideal(2+i).residues())
```

(continues on next page)
[-2*i, -i, 0, i, 2*i]
sage: list(K.ideal(i).residues())
[0]
sage: I = K.ideal(3+6*i)
sage: reps = I.residues()
sage: len(list(reps)) == I.norm()
True
sage: all(r == s or not (r-s) in I for r in reps for s in reps) # long time
True
sage: K.<a> = NumberField(x^3-10)
sage: I = K.ideal(a-1)
sage: len(list(I.residues())) == I.norm()
True
sage: K.<z> = CyclotomicField(11)
sage: len(list(K.primes_above(3)[0].residues())) == 3**5 # long time (5s on
sage.math, 2011)
True

small_residue(f)
Given an element $f$ of the ambient number field, 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:

```
sage: k.<a> = NumberField(x^2 + 5)
sage: I = k.ideal(a)
sage: I.small_residue(14)
4
```

```
sage: K.<a> = NumberField(x^5 + 7*x^4 + 18*x^2 + x - 3)
sage: I = K.ideal(5)
sage: a^2 + 5*a - 3
```

support()
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:

```
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)]
```
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:

```
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:

```
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:

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

artin_symbol()

Return the Artin symbol \((K/Q, P)\), where \(K\) is the number field of \(P = \text{self}\). This is the unique element \(s\) of the decomposition group of \(P\) such that \(s(x) = xp \pmod{P}\) 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:

```
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:
\begin{verbatim}
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]
\end{verbatim}

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:**

\begin{verbatim}
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, -14)
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)
\end{verbatim}
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:

```
sage: QuadraticField(-23, 'w').primes_above(7)[0].decomposition_group()
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 + w - 23)
```

free_module()

Return the free \( \mathbb{Z} \)-module contained in the vector space associated to the ambient number field, that corresponds to this ideal.

EXAMPLES:

```
sage: K.<z> = CyclotomicField(7)
sage: I = K.factor(11)[0][0]; I
Fractional ideal (-3*z^4 - 2*z^3 - 2*z^2 - 2)
sage: A = I.free_module()
sage: A
Free module of degree 6 and rank 6 over Integer Ring
User basis matrix:
[[11 0 0 0 0 0]
 [ 0 11 0 0 0 0]
 [ 0 0 11 0 0 0]
 [10 4 5 1 0 0]
 [ 5 1 1 0 1 0]
 [ 5 6 2 1 1 1]]
```

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 0 1 0 0]
```

(continues on next page)
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()
Free module of degree 4 and rank 4 over Integer Ring
User basis matrix:
[ 1/4  0  0  0]
[-3/16 7/16 -1/16 1/16]
[ 3/16 7/16 1/16 1/16]
[ 0 -5/8  0 -1/8]
```

An example of intersecting ideals by intersecting free modules:

```
sage: K.<a> = NumberField(x^3 + x^2 - 2*x + 8)
sage: I = K.factor(2)
sage: p1 = I[0][0]; p2 = I[1][0]
sage: N = p1.free_module().intersection(p2.free_module()); N
Free module of degree 3 and rank 3 over Integer Ring
Echelon basis matrix:
[ 1  1/2  1/2]
[ 0  1  1]
[ 0  0  2]
sage: N.index_in(p1.free_module()).abs()
2
gens_reduced(proof=None)
```

Express this ideal in terms of at most two generators, and one if possible.

This function indirectly uses `bnfisprincipal`, so set `proof=True` if you want to prove correctness (which is the default).

EXAMPLES:

```
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2 + 5)
sage: K.ideal(0).gens_reduced()
(0,)
sage: J = K.ideal([a+2, 9])
sage: J.gens()
(a + 2, 9)
sage: J.gens_reduced() # random sign
(a + 2,
```
sage: K.ideal([a+2, 3]).gens_reduced()
(3, a + 2)

gens_two()
Express this ideal using exactly two generators, the first of which is a generator for the intersection of the
ideal with \( \mathbb{Q} \).

ALGORITHM: uses PARI's pari:idealtwoelt function, which runs in randomized polynomial time and is
very fast in practice.

EXAMPLES:

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)
sage: K.ideal([a+5, a+8]).gens_two()
(3, a + 2)
sage: K.ideal(0).gens_two()
(0, 0)
The second generator is zero if and only if the ideal is generated by a rational, in contrast to the PARI
function pari:idealtwoelt:

sage: I = K.ideal(12)
sage: pari(K).idealtwoelt(I)  # Note that second element is not zero
[12, [0, 12]-]
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:

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:

sage: K.<a> = QuadraticField(-5)
sage: J = K.ideal(23).factor()[0][0]
sage: J.ideal_class_log()
[1]
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()
Subgroup generated by [(1,2)] of (Galois group 2T1 (S2) with order 2 of x^2 + 23)
```

**integral_basis()**

Return a list of generators for this ideal as a \( \mathbb{Z} \)-module.

**EXAMPLES:**

```python
sage: 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
```

272 Chapter 3. Orders, Ideals, Ideal Classes
intersection\((\text{other})\)
Return the intersection of self and other.

EXAMPLES:
```python
sage: K.<a> = QuadraticField(-11)
sage: p = K.ideal((a + 1)/2); q = K.ideal((a + 3)/2)
sage: p.intersection(q) == q.intersection(p) == K.ideal(a-2)
True
```

An example with non-principal ideals:
```python
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:
```python
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()
True
sage: (K.ideal(1) / (3*a+1)).is_integral()
False
```

is_maximal()
Return True if this ideal is maximal. This is equivalent to self being prime and nonzero.

EXAMPLES:
```python
sage: K.<a> = NumberField(x^3 + 3); K
Number Field in a with defining polynomial x^3 + 3
sage: K.ideal(5).is_maximal()
False
sage: K.ideal(7).is_maximal()
True
```

is_prime()
Return True if this ideal is prime.

EXAMPLES:
```python
sage: K.<a> = NumberField(x^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()
False
sage: I = P^5
sage: I.is_principal()
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()
True
```

**is_zero()**

Return True iff self is the zero ideal

Note that $(0)$ is a `NumberFieldIdeal`, not a `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:

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:

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 self.

INPUT:

• self - a prime ideal.

OUTPUT: a PARI “prime ideal”, i.e. a five-component vector \([p, a, e, f, b]\) representing the prime ideal \(pO_K + aO_K, e, f \) as usual, \(a\) as vector of components on the integral basis, \(b\) Lenstra’s constant.

EXAMPLES:

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:
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:

```
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(-30,30).is_integral()  
True
sage: I.random_element().parent() is K
True
```

A relative example:

```
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
17/500002*a^3 + 372753/250001*a^2 - 1494505893/500002*a + 752473260/250001
sage: I.random_element().is_integral()  
True
sage: I.random_element(-100, 200).parent() is K
True
```

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:ideleqred 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)
```
\[ \left( \frac{\alpha}{\mathfrak{P}} \right) \equiv \alpha^{N(\mathfrak{P}) - 1} \mod \mathfrak{P} \]

**Note:** accepts \( m = 1 \), in which case returns 1

**Note:** can also be called for an element from `sage.rings.number_field_element.residue_symbol`

**Note:** \( e \) is coerced into the number field of \( \text{self} \)

**Note:** if \( m = 2, e \) is an integer, and \( \text{self.number_field()} \) 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):
```python
sage: K.<a> = NumberField(x - 1)
sage: K.ideal(11).residue_symbol(7,2)
-1
```

Cubic Residue:

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

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

```python
sage: K.<w> = NumberField(x^2 - x + 1)
sage: K.ideal(17).residue_symbol(w^2 + 3,5)
Traceback (most recent call last):
  ... ValueError: The residue symbol to that power is not defined for the number field
```

```python
smallest_integer()
```

Return the smallest non-negative integer in \( I \cap \mathbb{Z} \), where \( I \) is this ideal. If \( I = 0 \), returns 0.

**EXAMPLES:**

```python
sage: R.<x> = PolynomialRing(QQ)
sage: K.<a> = NumberField(x^2+6)
sage: I = K.ideal([4,a])/7; I
Fractional ideal (2/7, 1/7*a)
sage: I.smallest_integer()
2
```

```python
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: The residue symbol to that power is not defined for the number field
```

(continues on next page)
ValueError: p (= Ideal (0) of Number Field in a with defining polynomial x^5 +\rightarrow^2) must be nonzero

sage: K.ideal(0).valuation(K.factor(2)[0][0])
+Infinity

3.2. Number Field Ideals
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:

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)
- William Stein (2007-09-06)
- Nick Alexander (2009-01)

**EXAMPLES:**

```python
sage: K.<a,b> = NumberField([x^2 + 1, x^2 + 2])
sage: A = K.absolute_field('z')
sage: I = A.factor(7)[0][0]
sage: from_A, to_A = A.structure()
sage: G = [from_A(z) for z in I.gens()]; G
[7, -2*b*a - 1]
sage: K.fractional_ideal(G)
Fractional ideal ((1/2*b + 2)*a - 1/2*b + 2)
sage: K.fractional_ideal(G).absolute_norm().factor()
7^2
```

```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:**

```python
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
```

```python
absolute_ideal(names='a')
```

If this is an ideal in the extension \(L/K\), return the ideal with the same generators in the absolute field \(L/Q\).

**INPUT:**
- names (optional) – string; name of generator of the absolute field

**EXAMPLES:**

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

An example of an inert ideal:

sage: P = F.factor(13)[0][0]; P
Fractional ideal (13)
sage: J = L.ideal(13)
sage: J.absolute_ideal()  
Fractional ideal (13)

Now a non-trivial ideal in $L$ that is principal in the subfield $K$. Since the optional ‘names’ argument is not passed, the generators of the absolute ideal $J$ are returned in terms of the default field generator ‘a’. This does not agree with the generator ‘m’ of the absolute field $F$ defined above:

sage: J = L.ideal(b); J
Fractional ideal (b)
sage: J.absolute_ideal()  
Fractional ideal (a^2)
sage: J.relative_norm()  
Fractional ideal (2)

Now pass ‘m’ as the name for the generator of the absolute field:

sage: J.absolute_ideal('m')  
Fractional ideal (m^2)

Now an ideal not generated by an element of $K$:

sage: J = L.ideal(c); J
Fractional ideal (c)

absolute_norm()

Compute the absolute norm of this fractional ideal in a relative number field, returning a positive integer.

EXAMPLES:

sage: L.<a, b, c> = QQ.extension([x^2 - 23, x^2 - 5, x^2 - 7])

(continues on next page)
absolute_ramification_index()

Return the absolute ramification index of this fractional ideal, assuming it is prime. Otherwise, raise a ValueError.

The absolute ramification index is the power of this prime appearing in the factorization of the rational prime that this prime lies over.

Use relative_ramification_index to obtain the power of this prime occurring in the factorization of the prime ideal of the base field that this prime lies over.

EXAMPLES:

```python
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
```

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 or 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:

```python
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)
```

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)]

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(c)
sage: P = K.ideal((b*a - b - 1)*c/2 + a - 1)
sage: Q = K.ideal((b*a - b - 1)*c/2)
sage: list(I.factor()) == [(P, 2), (Q, 1)]
True
sage: I == P^2*Q
True
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: 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: 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: 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: N == L.ideal_below(); M == K.ideal([-a])
True
sage: Np = L.ideal( [ L(t) for t in M.gens() ])
sage: Np.ideal_below() == M
True
sage: M.parent()
Monoid of ideals of Number Field in a with defining polynomial x^2 + 6
sage: M.ring()
Number Field in a with defining polynomial x^2 + 6
sage: M.ring() is K
True
This example concerns an inert ideal:

sage: K = NumberField(x^4 + 6*x^2 + 24, 'a')
sage: K.factor(7)
Fractional ideal (7)
sage: K0, K0_into_K, _ = K.subfields(2)[0]
sage: K0
Number Field in a0 with defining polynomial x^2 - 6*x + 24
sage: L = K.relativize(K0_into_K, 'c'); L
Number Field in c with defining polynomial x^2 + a0 over its base field
sage: L.base_field() is K0
True
sage: L.ideal(7)
Fractional ideal (7)
sage: L.ideal(7).ideal_below()
Fractional ideal (7)
sage: L.ideal(7).ideal_below().number_field() is K0
True
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)
sage: L_into_K, K_into_L = L.structure()
sage: L.ideal(K_into_L(K0_into_K(w1))).ideal_below() == P1
True
The choice of embedding of quadratic field into quartic field matters:

sage: rho, tau = K0.embeddings(K)
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()
sage: J == K.ideal(b)
True
sage: J.number_field() == F
True
```

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 \( \mathbb{Z} \)-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
# random
[438, -b*a + 309, 219*a - 219*b, 156*a - 154*b]
```

The exact results are somewhat unpredictable, hence the \# random flag, but we can test that they are indeed a basis:

```
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:

```
sage: K.<a, b> = NumberField([x^2 - 17, x^3 - 2])
sage: K.ideal(a + b).is_prime()
True
sage: K.ideal(13).is_prime()
False
```

```
is_principal(proof=None)
Return True if this ideal is principal. If so, set self.__reduced_generators, with length one.

EXAMPLES:

```
sage: K.<a, b> = NumberField([x^2 - 23, x^2 + 1])
sage: I = K.ideal([7, (-1/2*b - 3/2)*a + 3/2*b + 9/2])
sage: I.is_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:

```
sage: K.<a, b> = NumberField([x^2 + 3, x^3 + 4])
sage: K.ideal(17).is_zero()
False
sage: K.ideal(0).is_zero()
True
```
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.ideal(2).norm()
Traceback (most recent call last):
...
NotImplementedError: For a fractional ideal in a relative number field you must use relative_norm or absolute_norm as appropriate
```

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.ideal(2, (a + 2*b + 3)/2)
sage: I.pari_rhnf()
[[1, -2; 0, 1], [[2, 1; 0, 1], 1/2]]
```

For ideals in relative number fields, ramification_index is deliberately not implemented in order to avoid ambiguity. Either relative_ramification_index() or absolute_ramification_index() should be used instead.

EXAMPLES:

```python
sage: 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
```

Compute the relative norm of this fractional ideal in a relative number field, returning an ideal in the base field.

EXAMPLES:

```python
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)
```
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_ramification_index()
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[]
sage: K.<c> = F.extension(Y^2 - (1 + a)*(a + b)*a*b)
sage: I = K.ideal(3, c)
sage: I.residue_class_degree()
[1, 2]
```
residue_class_degree()
Return the residue class degree of this prime.

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.residue_class_degree() for I in K.ideal(c).prime_factors()]
[1, 2]
```

3.3. Relative Number Field Ideals
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:

    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:

    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:
    (integer) 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])
sage: A = K.ideal(a + b)
sage: A.is_prime()
    True
sage: (A*K.ideal(3)).valuation(A)
    1
sage: K.ideal(25).valuation(5)
    Traceback (most recent call last):
    ... ValueError: p (= Fractional ideal (5)) must be a prime

sage.rings.number_field.number_field_ideal_rel.is_NumberFieldFractionalIdeal_rel(x)
    Return True if \( x \) is a fractional ideal of a relative number field.
    EXAMPLES:
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
def is_NumberFieldFractionalIdeal_rel(I)
    return False
sage: is_NumberFieldFractionalIdeal_rel(a + 1)
False
sage: is_NumberFieldFractionalIdeal_rel(ideal(5))
False
sage: k.<a> = NumberField(x^2 + 2)
sage: I = k.ideal([a + 1]); I
Fractional ideal (a + 1)
sage: is_NumberFieldFractionalIdeal_rel(I)
False
sage: R.<x> = QQ[]
sage: K.<a> = NumberField(x^2 + 6)
sage: L.<b> = K.extension(K['x'].gen()^4 + a)
sage: I = L.ideal(b); I
Fractional ideal (6, b)
sage: is_NumberFieldFractionalIdeal_rel(I)
True
sage: N = I.relative_norm(); N
Fractional ideal (-a)
sage: is_NumberFieldFractionalIdeal_rel(N)
False
sage: is_NumberFieldFractionalIdeal(N)
True
```
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

sage: c.exponents()
(1, 0)

Element
alias of FractionalIdealClass
gens_ideals()
Return generating ideals for the (S-)class group.
This is an alias for gens_values().

OUTPUT:
A tuple of ideals, one for each abstract Abelian group generator.
EXAMPLES:

```python
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))
```

definition

**number_field()**

Return the number field that this (S-)class group is attached to.

**EXAMPLES:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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:**

```python
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.
**gens()**

Return generators for a representative ideal in this (S-)ideal class.

**EXAMPLES:**

```
sage: K.<w>=QuadraticField(-23)
sage: OK = K.ring_of_integers()
sage: C = OK.class_group()
sage: P2a,P2b=[P for P,e in (2*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:**

```
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:**

```
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
```
reduce()
Return representative for this ideal class that has been reduced using PARI’s pari:idealred.

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
```

representative_prime(norm_bound=1000)
Return a prime ideal in this ideal class.

INPUT:

norm_bound (positive integer) – upper bound on the norm of primes tested.

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),)
```

```python
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 with a = 31.208973068654475*I
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 with a = 3.7416573867739425*I
sage: T = tuple([])
sage: CT = K.S_class_group(T);CT
S-class group of order 4 with structure C4 of Number Field in a with defining polynomial x^2 + 14 with a = 3.7416573867739425*I
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)
sage: G = K.ideal(3,a+1)
sage: CS(I)
Trivial S-ideal class
sage: CS(J)
Trivial S-ideal class
sage: CS(G)
```

(continues on next page)
EXAMPLES:

```
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)
```

### 3.5 Unit and S-unit groups of Number Fields

**EXAMPLES:**

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4-8*x^2+36)
sage: UK = UnitGroup(K); UK
Unit group with structure C4 x Z of Number Field in a with defining polynomial x^4 - 8*x^2 + 36
```

The first generator is a primitive root of unity in the field:

```
sage: UK.gens()
(u0, u1)
sage: UK.gens_values()  # random
[-1/12*a^3 + 1/6*a, 1/24*a^3 + 1/4*a^2 - 1/12*a - 1]
sage: UK.gen(0).value()
1/12*a^3 - 1/6*a
sage: UK.gen(0)
u0
sage: UK.gen(0) + K.one()  # coerce abstract generator into number field
1/12*a^3 - 1/6*a + 1
```

(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:

```python
sage: UK(1)
1
sage: UK(-1)
\rstrip{0}^2
```

```python
sage: [UK(u) for u in (x^4-1).roots(K, multiplicities=False)]
[1, \rstrip{0}^2, \rstrip{0}, \rstrip{0}^3]
```

```python
sage: UK.fundamental_units()  # random
[1/24*a^3 + 1/4*a^2 - 1/12*a - 1]
```

```python
torsion_gen = UK.torsion_generator(); torsion_gen
\rstrip{0}
torsion_gen.value()
1/12*a^3 - 1/6*a
```

```python
sage: UK.roots_of_unity()
[1/12*a^3 - 1/6*a, -1, -1/12*a^3 + 1/6*a, 1]
```

Exp and log functions provide maps between units as field elements and exponent vectors with respect to the generators:

```python
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)
```

```python
sage: u = UK.fundamental_units()[0]
sage: [UK.log(u^k) == (0,k) for k in range(10)]
[True, True, True, True, True, True, True, True, True, True]
```

S-unit groups may be constructed, where $S$ is a set of primes:

```python
sage: K.<a> = NumberField(x^5-2, 'a')
sage: UK = UnitGroup(K)
sage: UK.rank()
2
```

```python
sage: S = K.ideal(3).prime_factors(); S
[Fractional ideal (3, a + 1), Fractional ideal (3, a - 1)]
sage: SUK = UnitGroup(K,S=tuple(S)); SUK
```

(continues on next page)
S-unit group with structure $C_2 \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ of Number Field in $a$ with defining polynomial $x^6 + 2$ with $S = (\text{Fractional ideal (3, } a + 1), \text{ Fractional ideal (3, } a - 1))$

```python
sage: SUK.primes()
(Fractional ideal (3, a + 1), Fractional ideal (3, a - 1))
```

```python
sage: SUK.rank()
4
```

```python
sage: SUK.gens_values()
[-1, a^2 + 1, -a^5 - a^4 + a^2 + a + 1, a + 1, a - 1]
```

```python
sage: u = 9*prod(SUK.gens_values()); u
-18*a^5 - 18*a^4 - 18*a^3 - 9*a^2 + 9*a + 27
```

```python
sage: SUK.log(u)
(1, 3, 1, 7, 7)
```

```python
sage: u == SUK.exp((1,3,1,7,7))
True
```

A relative number field example:

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

```python
sage: UL = L.unit_group(); UL
Unit group with structure $C_{24} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$ of Number Field in $a$ with defining polynomial $x^2 + x + 1$ over its base field
```

```python
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]
```

```python
sage: UL.zeta_order()
24
```

```python
sage: UL.roots_of_unity()
[-b*a, -b^2*a - b^2, -b^3, -a, -b*a - b, -b^2, b^3*a, -a - 1, -b, b^2*a, b^3*a + b^3, -1, b*a, b^2*a + b^2, b^3, a, b*a + b, b^2, -b^3*a, a + 1, b, -b^2*a, -b^3*a - b^3, 1]
```

A relative extension example, which worked thanks to the code review by F.W.Clarke:
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: u = UK.exp(vec)

sage: UK.log(u)
(13, 6, 7, 8, 9, 10)

sage: u = UK.gens()[-1]

sage: UK.exp(UK.log(u)) == u.value()
True

An S-unit example:

sage: SUK = UnitGroup(K,S=2)
sage: v = (3,1,4,1,5,9,2)
sage: u = SUK.exp(v); u

sage: SUK.log(u)
(3, 1, 4, 1, 5, 9, 2)

(continues on next page)
fundamental_units()

Return generators for the free part of the unit group, as a list.

EXAMPLES:

```python
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 + 23)
sage: U = UnitGroup(K)
sage: U.fundamental_units()  # random
[1/4*a^3 - 7/4*a^2 + 17/4*a - 19/4]
```

log(u)

Return the exponents of the unit u with respect to group generators.

INPUT:

- u – Any object from which an element of the unit group’s number field K may be constructed; an error is raised if an element of K cannot be constructed from u, or if the element constructed is not a unit.

OUTPUT: a list of integers giving the exponents of u with respect to the unit group’s basis.

EXAMPLES:

```python
sage: x = polygen(QQ)
sage: K.<z> = CyclotomicField(13)
sage: UK = UnitGroup(K)
sage: [UK.log(u) for u in UK.gens()]
[(1, 0, 0, 0, 0, 0),
 (0, 1, 0, 0, 0, 0),
 (0, 0, 1, 0, 0, 0),
 (0, 0, 0, 1, 0, 0),
 (0, 0, 0, 0, 1, 0),
 (0, 0, 0, 0, 0, 1)]
sage: vec = [65,6,7,8,9,10]
sage: unit = UK.exp(vec); unit
-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:

```python
sage: SUK = UnitGroup(K,S=2)
sage: v = (3,1,4,1,5,9,2)
sage: u = SUK.exp(v); u
8732*z^11 - 15496*z^10 - 51840*z^9 - 68804*z^8 - 51840*z^7 - 15496*z^6 + 8732*z^5 - 34216*z^3 - 64312*z^2 - 64312*z - 34216
sage: SUK.log(u)
(3, 1, 4, 1, 5, 9, 2)
sage: SUK.log(u) == v
True
```
number_field()

Return the number field associated with this unit group.

EXAMPLES:

```sage
sage: U = UnitGroup(QuadraticField(-23, 'w')); U
Unit group with structure C2 of Number Field in w with defining polynomial x^2+w+23 with w = 4.795831523312720?/I
sage: U.number_field()
Number Field in w with defining polynomial x^2 + 23 with w = 4.795831523312720?/I
```

primes()

Return the (possibly empty) list of primes associated with this S-unit group.

EXAMPLES:

```sage
sage: K.<a> = QuadraticField(-23)
sage: S = tuple(K.ideal(3).prime_factors()); S
(Fractional ideal (3, 1/2*a - 1/2), Fractional ideal (3, 1/2*a + 1/2))
sage: U = UnitGroup(K,S=tuple(S)); U
S-unit group with structure C2 x Z x Z of Number Field in a with defining polynomial x^2 + 23 with a = 4.795831523312720?/I with S = (Fractional ideal (3, 1/2*a - 1/2), Fractional ideal (3, 1/2*a + 1/2))
sage: U.primes() == S
True
```

rank()

Return the rank of the unit group.

EXAMPLES:

```sage
sage: K.<z> = CyclotomicField(13)
sage: UnitGroup(K).rank()
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
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
\[-1/4*a^3 - 1/4*a + 1/2\]

\textbf{zeta}(n=2, all=False)

Return one, or a list of all, primitive n-th root of unity in this unit group.

\textbf{EXAMPLES:}

sage: x = polygen(QQ)
sage: K.<z> = NumberField(x^2 + 3)
sage: U = UnitGroup(K)
sage: U.zeta(1)
1
sage: U.zeta(2)
-1
sage: U.zeta(2, all=True)
[-1]

sage: U.zeta(3)
-1/2*z - 1/2
sage: U.zeta(3, all=True)
[-1/2*z - 1/2, 1/2*z - 1/2]

sage: U.zeta(4)
Traceback (most recent call last):
... ValueError: n (=4) does not divide order of generator

sage: r.<x> = QQ[]
sage: K.<b> = NumberField(x^2+1)
sage: U = UnitGroup(K)
sage: U.zeta(4)
b
sage: U.zeta(4, all=True)
[b, -b]

sage: U.zeta(3)
Traceback (most recent call last):
...
ValueError: n (=3) does not divide order of generator

sage: U.zeta(3, all=True)
[]

\textbf{zeta_order()}

Returns the order of the torsion part of the unit group.

\textbf{EXAMPLES:}

sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^4 - x^2 + 4)
sage: U = UnitGroup(K)
sage: U.zeta_order()
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]
• [Yu2007]
• [AKMRVW]

AUTHORS:

• Alejandra Alvarado, Angelos Koutsianas, Beth Malmskog, Christopher Rasmussen, David Roe, Christelle Vincent, Mckenzie West (2018-04-25 to 2018-11-09): original version

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 = [(0, 1), (4, 0), xi + 2, -xi - 1],
...
   ((1, -1), (0, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
   ((1, 0), (5, 0), xi + 1, -xi),
   ((2, 0), (5, 1), xi, -xi + 1)]

sage: sols = solve_S_unit_equation(K, S, 200)

sage: eq_up_to_order(sols, expected)
True
```

Todo:

• Use Cython to improve timings on the sieve

```python
sage.rings.number_field.S_unit_solver.K0_func(SUK, A, prec=106)
Return the constant $K_0$ from [AKMRVW].

INPUT:

• SUK – a group of S-units
• A – the set of the products of the coefficients of the S-unit equation with each root of unity of $K$
• prec – the precision of the real field (default: 106)

OUTPUT:

The constant $K_0$, a real number.

EXAMPLES:
```
### 3.6. Solve S-unit equation $x + y = 1$

#### REFERENCES:

- [Sma1995] p. 824
- [AKMRVW] arXiv:1903.00977

**sage.rings.number_field.S_unit_solver.K1_func(SUK, v, A, prec=106)**

Return the constant $K_1$ from Smart’s TCDF paper, [Sma1995].

**INPUT:**

- `SUk` – a group of $S$-units
- `v` – an infinite place of $K$ (element of `SUk.number_field().places(prec)`)
- `A` – a list of all products of each potential $a, b$ in the $S$-unit equation $ax + by + 1 = 0$ with each root of unity of $K$
- `prec` – the precision of the real field (default: 106)

**OUTPUT:**

The constant $K_1$, a real number

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import K1_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: K1_func(SUK, phi_real, A)
4.483038368145048508970350163578e16
sage: K1_func(SUK, phi_complex, A)
2.073346189067285101984136298965e17
```

**REFERENCES:**

- [Sma1995] p. 825

**sage.rings.number_field.S_unit_solver.Omega_prime(dK, v, mu_list, prec=106)**

Return the constant Omega’ appearing in [AKMRVW].

**INPUT:**

- `dK` – the degree of a number field $K$
- `v` – a finite place of $K$
- `mu_list` – a list of nonzero elements of $K$. It is assumed that the sublist `mu[1:]` is multiplicatively independent.
- `prec` – the precision of the real field

**EXAMPLES:**
OUTPUT:

The constant $\Omega'$.

EXAMPLES:

```sage
def from sage.rings.number_field.S_unit_solver import mus, Omega_prime
K.<a> = NumberField(x^3 - 3)
SUK = UnitGroup(K, S=tuple(K.primes_above(6)))
v = K.primes_above(3)[0]
mus(SUK, v)
dK = K.degree()
Omega_prime(dK, v, mu_list)
```

0.000487349679922696

REFERENCES:

• [AKMRVW] arXiv:1903:.00977

`sage.rings.number_field.S_unit_solver.Yu_C1_star(n, v, prec=106)`

Return the constant $C_1^*$ appearing in [Yu2007] (1.23).

INPUT:

• $n$ – the number of generators of a multiplicative subgroup of a field $K$
• $v$ – a finite place of $K$ (a fractional ideal)
• $\text{prec}$ – the precision of the real field

OUTPUT:

The constant $C_{1,\text{tar}}$ as a real number.

EXAMPLES:

```sage
def from sage.rings.number_field.S_unit_solver import Yu_C1_star
K.<a> = NumberField(x^2 + 5)
v11 = K.primes_above(11)[0]
Yu_C1_star(1, v11)
```

2.154667761574516556114215527020e6

REFERENCES:

• [Yu2007] p.189,193

`sage.rings.number_field.S_unit_solver.Yu_al_kappa1_cl(p, dK, ep)`

Compute the constants $a(1)$, $\kappa(1)$, and $c(1)$ of [Yu2007].

INPUT:

• $p$ – a rational prime number
• $dK$ – the absolute degree of some number field $K$
• $ep$ – the absolute ramification index of some prime $\mathfrak{p}$ of $K$ lying above $p$

OUTPUT:

The constants $a(1)$, $\kappa(1)$, and $c(1)$.

EXAMPLES:
sage: from sage.rings.number_field.S_unit_solver import Yu_a1_kappa1_c1
sage: Yu_a1_kappa1_c1(5, 10, 3)
(16, 20, 319)

REFERENCES:

• [Yu2007]
sage.rings.number_field.S_unit_solver.Yu_bound(SUK, v, prec=106)

Return \( c_8 \) such that 
\[
  c_8 > e^{\exp(2)/\log(2)} \quad \text{and} \quad \text{ord}_p(\Theta - 1) < c_8 \log B,
\]
where 
\[
  \Theta = \prod_{j=1}^{n} \alpha_j^{b_j}
\]
and \( B \geq \max_j |b_j| \) and \( B \geq 3 \).

INPUT:

• SUK – a group of \( S \)-units
• v – a finite place of \( K \) (a fractional ideal)
• prec – the precision of the real field

OUTPUT:

The constant \( c_8 \) as a real number.

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import Yu_bound
sage: K.<a> = NumberField(x^2 + 11)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(6)))
sage: v = K.primes_above(3)[0]
sage: Yu_bound(SUK, v)
9.03984381033128e9

REFERENCES:

• [Sma1995] p. 825
• [Yu2007] p. 189–193 esp. Theorem 1
• [AKMRVW] arXiv:1903.00977

sage.rings.number_field.S_unit_solver.Yu_condition_115(K, v)

Return True or False, as the number field \( K \) and the finite place \( v \) satisfy condition (1.15) of [Yu2007].

INPUT:

• K – a number field
• v – a finite place of \( K \)

OUTPUT:

True if (1.15) is satisfied, otherwise False.

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import Yu_condition_115
sage: K.<a> = NumberField(x^2 + 5)
sage: v2 = K.primes_above(2)[0]
sage: v11 = K.primes_above(11)[0]
sage: Yu_condition_115(K, v2)
False

(continues on next page)
sage: Yu_condition_115(K, v11)
True

REFERENCES:

• [Yu2007] p. 188

sage.rings.number_field.S_unit_solver.Yu_modifed_height(mu, n, v, prec=106)
Return the value of h(n)(mu) as appearing in [Yu2007] equation (1.21).

INPUT:

• mu – an element of a field K
• n – number of mu_j to be considered in Yu’s Theorem.
• v – a place of K
• prec – the precision of the real field

OUTPUT:

The value h_p(mu).

EXAMPLES:

sage: K.<a> = NumberField(x^2 + 5)
sage: v11 = K.primes_above(11)[0]
sage: from sage.rings.number_field.S_unit_solver import Yu_modified_height
sage: Yu_modified_height(a, 3, v11)
0.8047189562170501873003796666131

If mu is a root of unity, the output is not zero. :: sage: Yu_modified_height(-1, 3, v11)
0.03425564675426243634374205111379

REFERENCES:

• [Yu2007] p. 192

sage.rings.number_field.S_unit_solver.beta_k(betas_and_ns)
Return a pair [β_k, |β_k|_v], where β_k has the smallest nonzero valuation in absolute value of the list betas_and_ns.

INPUT:

• betas_and_ns – a list of pairs [beta, val_v(beta)] outputted from the function where beta is an element of SUK.fundamental_units()

OUTPUT:

The pair [beta_k, v(beta_k)], where beta_k is an element of K and val_v(beta_k) is an integer

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import beta_k
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: betas = [ [beta, beta.valuation(v_fin)] for beta in SUK.fundamental_units() ]
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 $K$
• 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, prec=106)
Return the constant $c_{13}$ from Smart's TCDF paper, [Sma1995].

INPUT:
• SUK – a group of $S$-units
• $v$ – an infinite place of $K$ (element of SUK.number_field().places(prec))
• prec – the precision of the real field (default: 106)

OUTPUT:
The constant $c_{13}$, as a real number

EXAMPLES:
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: c13_func(SUK, phi_real) # abs tol 1e-29
 0.42578591347980347461973286726
sage: c13_func(SUK, phi_complex) # abs tol 1e-29
 0.2128929567399017373098663643363

It is an error to input a finite place.

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 [AKMRVW].

INPUT:
  • SUK – a group of \( S \)-units
  • prec – the precision of the real field (default: 106)

OUTPUT:
The constant \( c_3 \), as a real number

EXAMPLES:

sage: from sage.rings.number_field.S_unit_solver import c3_func
sage: K.<xi> = NumberField(x^3-3)
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
sage: c3_func(SUK) # abs tol 1e-29
 0.42578591347980347461973286726

Note: The numerator should be as close to 1 as possible, especially as the rank of the \( S \)-units grows large

REFERENCES:
  • [AKMRVW] arXiv:1903.00977

sage.rings.number_field.S_unit_solver.c4_func(SUK, v, A, prec=106)
Return the constant \( c_4 \) from Smart’s TCDF paper, [Sma1995].

INPUT:
  • SUK – a group of \( S \)-units
• v – a place of \( K \), finite (a fractional ideal) or infinite (element of \( \text{SUK.number_field().places(prec)} \))

• \( A \) – 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
c4_func(SUK,phi_real,A)
1.000000000000000000000000000000

c4_func(SUK,phi_complex,A)
1.000000000000000000000000000000

c4_func(SUK,v_fin,A)
1.000000000000000000000000000000
```

REFERENCES:

• [Sma1995] p. 824

sage.rings.number_field.S_unit_solver.clean_rfv_dict(rfv_dictionary)
Given a residue field vector dictionary, 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:

```sage
clean_rfv_dict(rfv_dict)
len(rfv_dict)
```

(continues on next page)
sage: clean_rfv_dict(rfv_dict)
sage: len(rfv_dict)
4
sage: rfv_dict
{(1, 3): [3, 2], (2, 1): [4, 6], (3, 0): [6, 6], (5, 4): [3, 6]}

sage.rings.number_field.S_unit_solver.clean_sfs(sfs_list)
Given a list of S-unit equation solutions, remove trivial redundancies.

INPUT:
• sfs_list – a list of solutions to the S-unit equation

OUTPUT:
A list of solutions to the S-unit equation

Note: The function looks for cases where \( x + y = 1 \) and \( y + x = 1 \) appear as separate solutions, and removes one.

EXAMPLES:
The function is not dependent on the number field and removes redundancies in any list.

sage: from sage.rings.number_field.S_unit_solver import clean_sfs
sage: sols = [((1, 0, 0), (0, 0, 1), -1, 2), ((0, 0, 1), (1, 0, 0), 2, -1)]
sage: clean_sfs( sols )
[((1, 0, 0), (0, 0, 1), -1, 2)]

sage.rings.number_field.S_unit_solver.column_Log(SUK, iota, U, prec=106)
Return the log vector of iota; i.e., the logs of all the valuations.

INPUT:
• SUK – a group of S-units
• iota – an element of \( K \)
• U – a list of places (finite or infinite) of \( K \)
• 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

```python
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 `[ [v0, w0], [v1, w1], ..., [vk, wk] ]` where `[vi, wi]` is a pair of complementary exponent vectors modulo `qi - 1`, and all pairs are compatible.
- `split_primes_list` – a list of primes `q0, q1, ..., qk`

**OUTPUT:**

A pair of vectors `[v, w]` satisfying:

1. `v[0] == vi[0]` for all `i`
2. `w[0] == wi[0]` for all `i`
3. `v[j] == vi[j]` modulo `qi - 1` for all `i` and all `j > 0`
4. `w[j] == wi[j]` modulo `qi - 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 `{qi - 1 : qi in split_primes_list}`

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import compatible_system_lift
sage: split_primes_list = [3, 7]
sage: comp_sys = [[[0, 1, 0], [0, 1, 0]], [[0, 3, 4], [0, 1, 2]]]
sage: compatible_system_lift(comp_sys, split_primes_list)
[(0, 3, -2), (0, 1, 2)]
```

```python
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 `[ [v0, w0], [v1, w1], ..., [vk, wk] ]`, 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 \( H = \text{lcm}(qj - 1 : qj \text{ in split\_primes\_list}) \). Then for any compatible system, there is at most one pair of integer exponent vectors \( [v, w] \) such that:
  - every entry of \( v \) and \( w \) is bounded in absolute value by \( H \)
  - for any \( qj \), \( v \) and \( vj \) agree modulo \( (qj - 1) \)
  - for any \( qj \), \( w \) and \( wj \) agree modulo \( (qj - 1) \)

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)]]
```

`sage.rings.number_field.S_unit_solver.compatibile_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),
 (3, 1, 8, 7),
 (3, 7, 2, 1),
 (3, 7, 2, 7),
 (3, 7, 8, 1),
 (3, 7, 8, 7)]
```

The order of the moduli matters.
Given exponent vectors with respect to two moduli, determines if they are compatible.

INPUT:
- \(a0\) – an exponent vector modulo \(m0\)
- \(a1\) – an exponent vector modulo \(m1\) (must have the same length as \(a0\))
- \(g\) – the gcd of \(m0\) and \(m1\)
- \(l\) – the length of \(a0\) and of \(a1\)

OUTPUT:
True if there is an integer exponent vector \(a\) satisfying

\[
\begin{align*}
a[0] &= a0[0] = a1[0] \\
(a[1:]) &= a0[1:] \mod m0 \\
(a[1:]) &= a1[1:] \mod m1
\end{align*}
\]

and False otherwise.

Note:
- Exponent vectors must agree exactly in the first coordinate.
- If exponent vectors are different lengths, an error is raised.

EXAMPLES:

```python
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
```

Constructs a dictionary associating complement vectors to residue field vectors.

INPUT:
- \(rfv\_to\_ev\_dict\) – a dictionary whose keys are residue field vectors and whose values are lists of exponent vectors with the associated residue field vector.
- \(q\) – the characteristic of the residue field

OUTPUT:
A dictionary whose typical key is an exponent vector \(a\), and whose associated value is a list of complementary exponent vectors to \(a\).
EXAMPLES:

In this example, we use the list generated when solving the $S$-unit equation in the case that $K$ is defined by the polynomial $x^2 + x + 1$ and $S$ consists of the primes above 3.

```python
sage: from sage.rings.number_field.S_unit_solver import construct_comp_exp_vec
sage: rfv_to_ev_dict = {(6, 6): [(3, 0)], (5, 6): [(1, 2)], (5, 4): [(5, 3)], (6, 2): [(5, 5)], (2, 5): [(0, 1)], (5, 5): [(3, 4)], (4, 4): [(0, 2)], (6, 3): [(1, 4)], (3, 6): [(5, 4)], (2, 2): [(0, 4)], (3, 5): [(1, 0)], (6, 4): [(1, 1)], (3, 4): [(5, 3)], (2, 6): [(4, 5)], (4, 5): [(4, 3)], (2, 3): [(2, 3)], (4, 2): [(4, 3)], (5, 5): [(5, 3)], (4, 6): [(2, 1)], (3, 4): [(3, 5)], (4, 3): [(0, 5)], (5, 2): [(3, 1)], (2, 4): [(2, 0)]}

sage: construct_comp_exp_vec(rfv_to_ev_dict, 7)
{(0, 1): [(1, 4)],
 (0, 2): [(0, 2)],
 (0, 4): [(3, 0)],
 (0, 5): [(4, 3)],
 (1, 0): [(5, 0)],
 (1, 1): [(2, 0)],
 (1, 2): [(1, 3)],
 (1, 3): [(1, 2)],
 (1, 4): [(0, 1)],
 (2, 0): [(1, 1)],
 (2, 1): [(4, 0)],
 (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's `sage.rings.number_field.S_unit_solver.construct_complement_dictionaries(split_primes_list, SUK, verbose=False)` function is used 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 $q-1$. 

---

Chapter 3. Orders, Ideals, Ideal Classes
If $w$ is an exponent vector in $\text{comp}\_\text{exp}\_\text{vec}[q][v]$, then the residue field vectors modulo $q$ for $v$ and $w$ sum to $[1,1,\ldots,1]$

**Note:**

- The data of $\text{comp}\_\text{exp}\_\text{vec}$ will later be lifted to $\mathbb{Z}$ to look for true $S$-Unit equation solutions.
- During construction, the various dictionaries are compared to each other several times to eliminate as many mod $q$ solutions as possible.
- The authors acknowledge a helpful discussion with Norman Danner which helped formulate this code.

**EXAMPLES:**

```python
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)
sage: SUK = K.S_unit_group(S=K.primes_above(H))
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)],
                 (0, 1, 2): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (0, 3, 2): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (0, 5, 0): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (0, 5, 4): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (1, 0, 0): [(0, 5, 4), (0, 3, 2), (0, 1, 0)],
                 (1, 0, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 2, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 2, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 2, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 4, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 4, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 4, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)]},
               7: {(0, 1, 0): [(1, 0, 0), (1, 4, 4), (1, 2, 2)],
                 (0, 1, 2): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (0, 3, 2): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (0, 3, 4): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (0, 5, 0): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (0, 5, 4): [(0, 1, 2), (0, 3, 4), (0, 5, 0)],
                 (1, 0, 0): [(0, 5, 4), (0, 3, 2), (0, 1, 0)],
                 (1, 0, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 2, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 2, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 2, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 4, 0): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 4, 2): [(1, 0, 4), (1, 4, 2), (1, 2, 0)],
                 (1, 4, 4): [(1, 0, 4), (1, 4, 2), (1, 2, 0)]}}
sage: all(set(actual[p][vec]) == set(expected[p][vec]) for p in [3,7] for vec in actual[p])
True
```

**Return a reverse lookup dictionary, to find the exponent vectors associated to a given residue field vector.**

**INPUT:**

- `rfv_dictionary` – a dictionary whose keys are exponent vectors and whose values are the associated residue field vectors
- `q` – a prime (assumed to split completely in the relevant number field)
- `d` – the number of primes in $K$ above the rational prime $q$
- `verbose` – a boolean flag to indicate more detailed output is desired (default: False)

**OUTPUT:**
A dictionary \( P \) whose keys are residue field vectors and whose values are lists of all exponent vectors which correspond to the given residue field vector.

**Note:**
- For example, if \( rfv\_dictionary[\ e0\ ] = r0 \), then \( P[\ r0\ ] \) is a list which contains \( e0 \).
- During construction, some residue field vectors can be eliminated as coming from solutions to the \( S \)-unit equation. Such vectors are dropped from the keys of the dictionary \( P \).

**EXAMPLES:**
In this example, we use a truncated list generated when solving the \( S \)-unit equation in the case that \( K \) is defined by the polynomial \( x^2 + x + 1 \) and \( S \) consists of the primes above 3:

```sage
from sage.rings.number_field.S_unit_solver import construct_rfv_to_ev
defining_polynomial_for_Kp(prime, prec=106)
```

Return the maximum of all of the \( K_1 \)'s as they are LLL-optimized for each infinite place \( v \).

**INPUT:**
- \( SUK \) – a group of \( S \)-units
- \( A \) – a list of all products of each potential \( a \), \( b \) in the \( S \)-unit equation \( ax + by + 1 = 0 \) with each root of unity of \( K \)
- \( \text{prec} \) – 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 defining_polynomial_for_Kp
```

A polynomial with integer coefficients that is equivalent \( \mod p^\text{prec} \) to a defining polynomial for the completion of \( K \) associated to the specified prime.
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
sage: p5 = K.prime_above(5); p5
Fractional ideal (5, a + 2)
sage: defining_polynomial_for_Kp(p5, 100)
x + 3408332191958133385114942613351834100964285496304040728996961917542037
```

`sage.rings.number_field.S_unit_solver.drop_vector(ev, p, q, 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 `complement_ev_dict`
- `complement_ev_dict` – a dictionary of dictionaries, whose keys are primes `complement_ev_dict[q]` is a dictionary whose keys are exponent vectors modulo `q-1` and whose values are lists of complementary exponent vectors modulo `q-1`

**OUTPUT:**
Returns `True` if `ev` may be dropped from the complement exponent vector dictionary, and `False` if not.

**Note:**
- If `ev` is not compatible with any of the vectors modulo `q-1`, then it can no longer correspond to a solution of the $S$-unit equation. It returns `True` to indicate that it should be removed.

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import drop_vector
sage: drop_vector((1, 2, 5), 7, 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, 0, 2)], (0, 1, 2): [(0, 1, 2), (0, 3, 4), (0, 5, 0)], (0, 5, 4): [(1, 0, 4), (1, 4, 2), (1, 0, 2)], (1, 2, 2): [(0, 5, 4), (0, 3, 2), (0, 1, 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 \( \text{prime} \) and \( e, f \) are the ramification index and residue degree, respectively.

**Note:** \( K \) has to be an absolute number field

EXAMPLES:

```sage
def from_sage_rings_number_field_S_unit_solver_import():
    from sage.rings.number_field.S_unit_solver import embedding_to_Kp
    K.<a> = QuadraticField(17)
    p = K.prime_above(13); p
    Fractional ideal (-a + 2)
    embedding_to_Kp(a-3, p, 15)
    -20542890112375827

def from_sage_rings_number_field_S_unit_solver_eq_up_to_order():
    from sage.rings.number_field.S_unit_solver import eq_up_to_order
    A = [(1,2,3,4),(5,6,7,8)]
    B = [(1,3,2,4),(6,4,5,7)]
    eq_up_to_order(A, B)
```

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
def from_sage_rings_number_field_S_unit_solver_import():
    from sage.rings.number_field.S_unit_solver import eq_up_to_order
    L = [(1,2,3,4),(5,6,7,8)]
```

(continues on next page)
sage: L1 = [L[1],L[0]]
sage: L2 = [(2,1,4,3),(6,5,8,7)]
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 \( p \)-adic logarithm of \( a \) with respect to \( \text{prime} \) modulo \( p^\text{prec} \), where \( p \) is the rational prime below \( \text{prime} \)

Note: Here we take into account the other primes in \( K \) above \( p \) in order to get coefficients with small values

EXAMPLES:

```python
sage: from sage.rings.number_field.S_unit_solver import log_p
sage: K.<a> = NumberField(x^2+14)
sage: p1 = K.primes_above(3)[0]
sage: log_p(a+2, p1, 20)
8255385638/3*a + 15567609440/3
```

```python
sage: K.<a> = NumberField(x^4+14)
sage: p1 = K.primes_above(5)[0]
sage: log_p(1/(a^2-4), p1, 30)
-42392683853751591352946/25*a^3 - 113099841599700611260219/25*a^2 - 8496491270664033599196/5*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 \( p \)-adic logarithm of \( a \) and accuracy \( p^\text{prec} \), where \( p \) is the rational prime below \( \text{prime} \)

ALGORITHM:

3.6. Solve S-unit equation \( x + y = 1 \)
The algorithm is based on the algorithm on page 30 of [Sma1998]

EXAMPLES:

```
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)
12004273678562*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)
56289408832645853692246880484598965434987932048396542150195486806212219509151065765558192523661836055046718599021297293805...
```

```
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 by n) vector with integer coordinates
- $\text{prec}$ : precision of real field (default: 106)

**OUTPUT:**

A lower bound for the square of

$$\ell(\mathcal{L}, \vec{y}) = \begin{cases} 
\min_{\vec{x} \in \mathcal{L}} \| \vec{x} - \vec{y} \| , & \vec{y} \notin \mathcal{L}, \\
\min_{\vec{y} \notin \mathcal{L}} \| \vec{y} \| , & \vec{y} \in \mathcal{L}.
\end{cases}$$

**ALGORITHM:**

The algorithm is based on V.9 and V.10 of [Sma1998]

EXAMPLES:

```
sage: B = random_matrix(ZZ, 2)
sage: while not B.determinant():
    ....:     B = random_matrix(ZZ, 3)
sage: B # random
[-2 -1 -1]
```

(continues on next page)
### sage.rings.number_field.S_unit_solver.mus(SUK, v)

Return a list \([\mu]\), for \(\mu\) defined in [AKMRVW].

**INPUT:**
- \(\text{SUK}\) – 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)
```

```python
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
```

```python
sage: v_fin = tuple(K.primes_above(3))[0]
```

```python
sage: mus(SUK, v_fin)

[xi^2 - 2]
```

**REFERENCES:**
- [AKMRVW]

### sage.rings.number_field.S_unit_solver.p_adic_LLL_bound(SUK, A, prec=106)

Return the maximum of all of the \(K_0\)'s as they are LLL-optimized for each finite place \(v\).

**INPUT:**
- \(\text{SUK}\) – 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 for p-adic LLL calculations (default: 106)

**OUTPUT:**
A bound for the max of exponents in the case that extremal place is finite (see [Sma1995]) as a real number

**EXAMPLES:**

```python
sage: from sage.rings.number_field.S_unit_solver import p_adic_LLL_bound
sage: K.<xi> = NumberField(x^3-3)
```

```python
sage: SUK = UnitGroup(K, S=tuple(K.primes_above(3)))
```

```python
sage: A = SUK.roots_of_unity()
```

```python
sage: prec = 100
```

```python
sage: p_adic_LLL_bound(SUK, A, prec)
```

89

3.6. Solve S-unit equation \(\scriptstyle x + y = 1\)

323
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 $K$
- `B0` – the initial bound
- `M` – a list of elements of $K$, the $\mu_i$’s from Lemma IX.3 of [Sma1998]
- `M_logp` – the $p$-adic logarithm of elements in $M$
- `m0` – an element of $K$, this is $\mu_0$ from Lemma IX.3 of [Sma1998]
- `c3` – a positive real constant
- `prec` – the precision of the calculations (default: 106), i.e., values are known to $O(p^{\text{prec}})$

**OUTPUT:**

A pair consisting of:

1. a new upper bound, an integer
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_adicLLL_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]

sage: Log_p_Mus = [185056824593551109742400*a^2 + 1389583284398773572269676*a + 717897987691852588770249]

sage: mu0 = K(-1)

sage: c3_value = 0.42578591347980

sage: m0_Kv_new, increase_precision = p_adicLLL_bound_one_prime(v, K0_old, Mus, Log_p_Mus, mu0, c3_value, prec)

sage: m0_Kv_new
0
sage: increase_precision
True
```

And now we increase the precision to make it all work:

```python
sage: prec = 106
sage: K0_old = 9.475576673109275443280257946930e17
```

(continues on next page)
sage: Log_p_Mus = [10295636043908673734686387890424583658678662701816*a^2 +...
˓→661450700156368458475507052066889190195530948403866*a]

sage: c3_value = 0.4257859134798034746197327286726

sage: m0_Kv_new, increase_precision = p_adicLLL_bound_one_prime(v, K0_old, Mus,
˓→Log_p_Mus, mu0, c3_value, prec)

sage: m0_Kv_new
476

sage: increase_precision
False

sage.rings.number_field.S_unit_solver.possible_mu0s(SUK, v)
Return a list [\mu_0] of all possible \mu_0 values defined in [AKMRVW].

INPUT:
• SUK – a group of S-units
• v – a finite place of K

OUTPUT:
A list [\mu_0] where each \mu_0 is an element of K

EXAMPLES:

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:
• [AKMRVW]
• [Sma1995] pp. 824-825, but we modify the definition of \(\sigma_tilde\) to make it easier to code

sage.rings.number_field.S_unit_solver.reduction_step_complex_case(place, B0, list_of_gens, torsion_gen, c13)

INPUT:
• place – (ring morphism) an infinite place of a number field \(K\)
• B0 – the initial bound
• list_of_gens – a set of generators of the free part of the group
• torsion_gen – an element of the torsion part of the group
• c13 – a positive real number

OUTPUT:
A tuple consisting of:

1. a new upper bound, an integer
2. a boolean value, True if we have to increase precision, otherwise False

**Note:** The constant $c_{13}$ in Section 5, [AKMRVW] This function does handle both real and non-real infinite places.

**REFERENCES:**
See [Sma1998], [AKMRVW].

**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: p1 = K.places(prec=100)[1]
sage: reduction_step_complex_case(p1, 10^5, G, -1, 2)
(18, False)
```

sage.rings.number_field.S_unit_solver.sieve_below_bound(K, S, bound=10, bump=10, split_primes_list=[], verbose=False)

Return all solutions to the S-unit equation $x + y = 1$ over $K$ with exponents below the given bound.

**INPUT:**

- $K$ – a number field (an absolute extension of the rationals)
- $S$ – a list of finite primes of $K$
- $bound$ – a positive integer upper bound for exponents, solutions with exponents having absolute value below this bound will be found (default: 10)
- $bump$ – a positive integer by which the minimum LCM will be increased if not enough split primes are found in sieving step (default: 10)
- $split_primes_list$ – a list of rational primes that split completely in the extension $K/Q$, used for sieving. For complete list of solutions should have lcm of $\{(p_i-1)\}$ for primes $p_i$ greater than bound (default: [])
- $verbose$ – an optional parameter allowing the user to print information during the sieving process (default: False)

**OUTPUT:**

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

1. The first two entries are tuples $A_i = (a_0, a_1, \ldots, a_t)$ and $B_i = (b_0, b_1, \ldots, b_t)$ of exponents.
2. The last two entries are $S$-units $x_i$ and $y_i$ in $K$ with $x_i + y_i = 1$.
3. If the default generators for the $S$-units of $K$ are $(\rho_0, \rho_1, \ldots, \rho_t)$, then these satisfy $x_i = \prod(\rho_i)^{a_i}$ and $y_i = \prod(\rho_i)^{b_i}$.

**EXAMPLES:**
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 = [....: ((1, -1), (0, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
....: ((0, 1), (4, 0), xi + 2, -xi - 1),
....: ((2, 0), (5, 1), xi, -xi + 1),
....: ((1, 0), (5, 0), xi + 1, -xi)]
sage: eq_up_to_order(sols, expected)
True

sage.rings.number_field.S_unit_solver.sieve_ordering(SUK, q)
Returns ordered data for running sieve on the primes in SUK over the rational prime q.

INPUT:
• SUK – the S-unit group of a number field K
• q – a rational prime number which splits completely in K

OUTPUT:
A list of tuples, [ideals_over_q, residue_fields, rho_images, product_rho_orders], where
1. ideals_over_q is a list of the d = [K : Q] ideals in K over q
2. residue_fields[i] is the residue field of ideals_over_q[i]
3. rho_images[i] is a list of the reductions of the generators in of the S-unit group, modulo ideals_over_q[i]
4. product_rho_orders[i] is the product of the multiplicative orders of the elements in rho_images[i]

Note:
• The list ideals_over_q is sorted so that the product of orders is smallest for ideals_over_q[0], as this will make the later sieving steps more efficient.
• The primes of S must not lie over q.

EXAMPLES:

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 (xi - 3),
 Fractional ideal (-2*xi^2 + 3),
 Fractional ideal (2*xi + 1))
sage: sieve_data[1]
(Residue field of Fractional ideal (xi - 3),
 Residue field of Fractional ideal (-2*xi^2 + 3),
(continues on next page))
Residue field of Fractional ideal \((2*xi + 1))\)

\[
\text{sage: sieve_data[2] = } ([18, 7, 16, 4], [18, 9, 12, 8], [18, 3, 10, 10])
\]

\[
\text{sage: sieve_data[3] = } (486, 648, 11664)
\]

\[
\text{sage.rings.number_field.S_unit_solver.solutions_from_systems(SUK, bound, cs_list, split_primes_list)}
\]

Lifts compatible systems to the integers and returns the S-unit equation solutions the lifts yield.

**INPUT:**

- \(\text{SUK}\) – the group of \(S\)-units where we search for solutions
- \(\text{bound}\) – a bound for the entries of all entries of all lifts
- \(\text{cs_list}\) – a list of compatible systems of exponent vectors modulo \(q - 1\) for various primes \(q\)
- \(\text{split_primes_list}\) – a list of primes giving the moduli of the exponent vectors in \(\text{cs_list}\)

**OUTPUT:**

A list of solutions to the S-unit equation. Each solution is a list:

1. an exponent vector over the integers, \(\text{ev}\)
2. an exponent vector over the integers, \(\text{cv}\)
3. the S-unit corresponding to \(\text{ev}, \text{iota_exp}\)
4. the S-unit corresponding to \(\text{cv}, \text{iota_comp}\)

**Note:**

- Every entry of \(\text{ev}\) is less than or equal to \(\text{bound}\) in absolute value
- Every entry of \(\text{cv}\) is less than or equal to \(\text{bound}\) in absolute value
- \(\text{iota_exp} + \text{iota_comp} == 1\)

**EXAMPLES:**

Given a single compatible system, a solution can be found.

\[
\text{sage: from sage.rings.number_field.S_unit_solver import solutions_from_systems}
\]

\[
\text{sage: K.<xi> = NumberField(x^2-15)}
\]

\[
\text{sage: SUK = K.S_unit_group(S=K.primes_above(2))}
\]

\[
\text{sage: split_primes_list = [7, 17]}
\]

\[
\text{sage: a_compatible_system = [[[(0, 0, 5), (0, 0, 5)], [(0, 0, 15), (0, 0, 15)]]]}
\]

\[
\text{sage: solutions_from_systems(SUK, 20, a_compatible_system, split_primes_list )}
\]

\[
[((0, 0, -1), (0, 0, -1), 1/2, 1/2)]
\]

\[
\text{sage.rings.number_field.S_unit_solver.solve_S_unit_equation(K, S, prec=106, include_exponents=True, include_bound=False, proof=None, verbose=False)}
\]

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 \([(A_1, B_1, x_1, y_1), (A_2, B_2, x_2, y_2), \ldots, (A_n, B_n, x_n, y_n)]\) 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 \( \text{include_exponents} \) 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:

```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: sols = solve_S_unit_equation(K, S, 200)
sage: expected = [(0, 1), (4, 0), xi + 2, -xi - 1),
               (1, -1), (0, -1), 1/3*xi + 2/3, -1/3*xi + 1/3),
               (1, 0), (5, 0), xi + 1, -xi),
               (2, 0), (5, 1), xi, -xi + 1)]
```

In order to see the bound as well use the optional parameter \( \text{include_bound} \):

```python
sage: solutions, bound = solve_S_unit_equation(K, S, 100, include_bound=True)
sage: bound
7
```

You can omit the exponent vectors:

```python
sage: sols = solve_S_unit_equation(K, S, 200, include_exponents=False)
sage: expected = [(xi + 2, -xi - 1), (1/3*xi + 2/3, -1/3*xi + 1/3), (-xi, xi + 1),
               (-xi + 1, xi)]
```

It is an error to use values in \( S \) that are not primes in \( K \):

```
3.6. Solve S-unit equation x + y = 1 329
```
We check the case that the rank is 0:

```python
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 \( 2 \times bound + 1 \).

**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:**

```python
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.

```python
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 [Coh1993]). Therefore, after discarding primes dividing $\Delta(f(x))$ (this includes all ramified primes), any integer $n$ such that $\gcd(f(n), P) > 0$ yields a prime $p|P$ such that $f(x)$ has a root modulo $p$. By the condition on discriminants, this root is a single root. As is well known (see, for example Theorem 4.8.13, p199 of [Coh1993]), the ideal generated by $(p, \alpha - n)$ is prime and of degree one.

**Warning:** It is possible that there are no primes of $K$ of absolute degree one of small prime norm, and it is possible that this algorithm will not find any primes of small norm.

**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
[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:

```
sage: L.<b> = F.extension(x^3 - a)
sage: Ps = L.primes_of_degree_one_list(3)
sage: Ps
[Fractional ideal (17, b - 5), Fractional ideal (23, b - 4), Fractional ideal (31, b - \omega - 2)]
sage: [ P.absolute_norm() for P in Ps ] # random
[17, 23, 31]
sage: all(ZZ(P.absolute_norm()).is_prime() for P in Ps)
True
```

(continues on next page)
sage: all(P.residue_class_degree() == 1 for P in Ps)
True
sage: M.<c> = NumberField(x^2 - x*b^2 + b)

sage: Ps = M.primes_of_degree_one_list(3)
sage: Ps
# random
[Fractional ideal (17, c - 2), Fractional ideal (c - 1), Fractional ideal (41, c + 15)]
sage: [ P.absolute_norm() for P in Ps ]  # random
[17, 31, 41]
sage: all(ZZ(P.absolute_norm()).is_prime() for P in Ps)
True
sage: all(P.residue_class_degree() == 1 for P in Ps)
True

AUTHORS:

• Nick Alexander (2008)

• 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)

Bases: object

Iterator that finds primes of a number field of absolute degree one and bounded small prime norm.

INPUT:

• field – a NumberField.

• num_integer_primes (default: 10000) – an integer. We try to find primes of absolute norm no greater than the num_integer_primes-th prime number. For example, if num_integer_primes is 2, the largest norm found will be 3, since the second prime is 3.

• max_iterations (default: 100) – an integer. We test max_iterations integers to find small primes before raising StopIteration.

AUTHOR:

• Nick Alexander

next()

Return a prime of absolute degree one of small prime norm.

Raises StopIteration if such a prime cannot be easily found.

EXAMPLES:

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)]
3.8 $p$-Selmer groups of number fields

This file contains code to compute $K(S, p)$ where

- $K$ is a number field
- $S$ is a finite set of primes of $K$
- $p$ is a prime number

For $m \geq 2$, $K(S, m)$ is defined to be the finite subgroup of $K^*/(K^*)^m$ consisting of elements represented by $a \in K^*$ whose valuation at all primes not in $S$ is a multiple of $m$. It fits in the short exact sequence

$$1 \to O_{K,S}^*/(O_{K,S}^*)^m \to K(S, m) \to Cl_{K,S}[m] \to 1$$

where $O_{K,S}$ is the group of $S$-units of $K$ and $Cl_{K,S}$ the $S$-class group. When $m = p$ is prime, $K(S, p)$ is a finite-dimensional vector space over $GF(p)$. Its generators come from three sources: units (modulo $p'$th powers); generators of the $p'$th powers of ideals which are not principal but whose $p'$ the powers are principal; and generators coming from the prime ideals in $S$.

The main function here is $pSelmerGroup()$. This will not normally be used by users, who instead will access it through a method of the NumberField class.

AUTHORS:
- John Cremona (2005-2021)

```
sage: from sage.rings.number_field.selmer_group import basis_for_p_cokernel
sage: K.<a> = NumberField(x^2 - x + 58)
sage: S = K.ideal(30).support(); S
[Fractional ideal (2, a), Fractional ideal (2, a + 1), Fractional ideal (3, a + 1), Fractional ideal (5, a + 1), Fractional ideal (5, a + 3)]
sage: C = K.class_group()
sage: C.gens_orders() (continues on next page)
```
(6, 2)
sage: [C(P).exponents() for P in S]
[(5, 0), (1, 0), (3, 1), (1, 1), (5, 1)]
sage: b, f = basis_for_p_cokernel(S, C, 2); b
[Fractional ideal (2), Fractional ideal (15, a + 13), Fractional ideal (5)]
sage: b, f = basis_for_p_cokernel(S, C, 3); b
[Fractional ideal (50, a + 18),
Fractional ideal (10, a + 3),
Fractional ideal (3, a + 1),
Fractional ideal (5)]
sage: b, f = basis_for_p_cokernel(S, C, 5); b
[Fractional ideal (2, a),
Fractional ideal (2, a + 1),
Fractional ideal (3, a + 1),
Fractional ideal (5, a + 1),
Fractional ideal (5, a + 3)]

sage.rings.number_field.selmer_group.coords_in_U_mod_p(u, U, p)
Return coordinates of a unit $u$ with respect to a basis of the $p$-cotorsion $U/U^p$ of the unit group $U$.

**INPUT:**

- $u$ (algebraic unit) – a unit in a number field $K$.
- $U$ (unit group) – the unit group of $K$.
- $p$ (prime) – a prime number.

**OUTPUT:**

The coordinates of the unit $u$ in the $p$-cotorsion group $U/U^p$.

**ALGORITHM:**

Take the coordinate vector of $u$ with respect to the generators of the unit group, drop the coordinate of the roots of unity factor if it is prime to $p$, and reduce the vector mod $p$.

**EXAMPLES:**

sage: from sage.rings.number_field.selmer_group import coords_in_U_mod_p
sage: K.<a> = NumberField(x^4 - 5*x^2 + 1)
sage: U = K.unit_group()
sage: U
Unit group with structure C2 x Z x Z x Z of Number Field in a with defining polynomial x^4 - 5*x^2 + 1
sage: u0, u1, u2, u3 = U.gens_values()
sage: u = u1*u2^2*u3^3
sage: coords_in_U_mod_p(u, U, 2)
[0, 1, 0, 1]
sage: coords_in_U_mod_p(u, U, 3)
[1, 2, 0]
sage: u*=u0
sage: coords_in_U_mod_p(u, U, 2)
[1, 1, 0, 1]
sage: coords_in_U_mod_p(u, U, 3)
[1, 2, 0]
sage.rings.number_field.selmer_group.pSelmerGroup(K, S, p, proof=None, debug=False)

Return the p-Selmer group $K(S, p)$ of the number field $K$ with respect to the prime ideals in $S$

INPUT:
- $K$ (number field) – a number field, or $\mathbb{Q}$.
- $S$ (list) – a list of prime ideals in $K$, or prime numbers when $K$ is $\mathbb{Q}$.
- $p$ (prime) – a prime number.
- $\text{proof}$ - if True then compute the class group provably correctly. Default is True. Call $\text{proof}$.
- $\text{number_field()}$ to change this default globally.
- $\text{debug}$ (boolean, default False) – debug flag.

OUTPUT:
(tuple) $\text{KSp}$, $\text{KSp_gens}$, $\text{from_KSp}$, $\text{to_KSp}$ where
- $\text{KSp}$ is an abstract vector space over $GF(p)$ isomorphic to $K(S, p)$;
- $\text{KSp_gens}$ is a list of elements of $K^*$ generating $K(S, p)$;
- $\text{from_KSp}$ is a function from $KSp$ to $K^*$ implementing the isomorphism from the abstract $K(S, p)$ to $K(S, p)$ as a subgroup of $K^*/(K^*)^p$;
- $\text{to_KSp}$ is a partial function from $K^*$ to $KSp$, defined on elements $a$ whose image in $K^*/(K^*)^p$ lies in $K(S, p)$, mapping them via the inverse isomorphism to the abstract vector space $KSp$.

ALGORITHM:
The list of generators of $K(S, p)$ is the concatenation of three sublists, called $\text{alphalist}$, $\text{betalist}$ and $\text{ulist}$ in the code. Only $\text{alphalist}$ depends on the primes in $S$.
- $\text{ulist}$ is a basis for $U/U^p$ where $U$ is the unit group. This is the list of fundamental units, including the generator of the group of roots of unity if its order is divisible by $p$. These have valuation 0 at all primes.
- $\text{betalist}$ is a list of the generators of the $p$'th powers of ideals which generate the $p$-torsion in the class group (so is empty if the class number is prime to $p$). These have valuation divisible by $p$ at all primes.
- $\text{alphalist}$ is a list of generators for each ideal $A$ in a basis of those ideals supported on $S$ (modulo $p$'th powers of ideals) which are $p$'th powers in the class group. We find $B$ such that $A/B^p$ is principal and take a generator of it, for each $A$ in a generating set. As a special case, if all the ideals in $S$ are principal then $\text{alphalist}$ is a list of their generators.

The map from the abstract space to $K^*$ is easy: we just take the product of the generators to powers given by the coefficient vector. No attempt is made to reduce the resulting product modulo $p$'th powers.

The reverse map is more complicated. Given $a \in K^*$:
- write the principal ideal $(a)$ in the form $AB^p$ with $A$ supported by $S$ and $p$'th power free. If this fails, then $a$ does not represent an element of $K(S, p)$ and an error is raised.
- set $I_S$ to be the group of ideals spanned by $S$ mod $p$'th powers, and $I_{S,p}$ the subgroup of $I_S$ which maps to 0 in $C/C^p$.
- Convert $A$ to an element of $I_{S,p}$, hence find the coordinates of $a$ with respect to the generators in $\text{alphalist}$.
- after dividing out by $A$, now $(a) = B^p$ (with a different $a$ and $B$). Write the ideal class $[B]$, whose $p$'th power is trivial, in terms of the generators of $C[p]$; then $B = (b)B_1$, where the coefficients of $B_1$ with respect to generators of $C[p]$ give the coordinates of the result with respect to the generators in $\text{betalist}$.
- after dividing out by $B$, and by $b^p$, we now have $(a) = (1)$, so $a$ is a unit, which can be expressed in terms of the unit generators.

3.8. $p$-Selmer groups of number fields
EXAMPLES:

Over \( \mathbb{Q} \) the unit contribution is trivial unless \( p = 2 \) and the class group is trivial:

```python
sage: from sage.rings.number_field.selmer_group import pSelmerGroup
sage: QS2, gens, fromQS2, toQS2 = pSelmerGroup(QQ, [2,3], 2)
sage: QS2
Vector space of dimension 3 over Finite Field of size 2
sage: gens
[2, 3, -1]
sage: a = fromQS2([1,1,1]); a.factor()
-1 * 2 * 3
sage: toQS2(-6)
(1, 1, 1)
```

```python
sage: QS3, gens, fromQS3, toQS3 = pSelmerGroup(QQ, [2,13], 3)
sage: QS3
Vector space of dimension 2 over Finite Field of size 3
sage: gens
[2, 13]
sage: a = fromQS3([5,4]); a.factor()
2^5 * 13^4
sage: toQS3(a)
(2, 1)
sage: toQS3(a) == QS3([5,4])
True
```

A real quadratic field with class number 2, where the fundamental unit is a generator, and the class group provides another generator when \( p = 2 \):

```python
sage: K.<a> = QuadraticField(-5)
sage: K.class_number()
2
sage: P2 = K.ideal(2, -a+1)
sage: P3 = K.ideal(3, a+1)
sage: P5 = K.ideal(a)
sage: KS2, gens, fromKS2, toKS2 = pSelmerGroup(K, [P2, P3, P5], 2)
sage: KS2
Vector space of dimension 4 over Finite Field of size 2
sage: gens
[a + 1, a, 2, -1]
```

Each generator must have even valuation at primes not in \( S \):

```python
sage: [K.ideal(g).factor() for g in gens]
[(Fractional ideal (2, a + 1)) * (Fractional ideal (3, a + 1)),
 Fractional ideal (-a),
 (Fractional ideal (2, a + 1))^2, 1]
```

```python
sage: toKS2(10)
(0, 0, 1, 1)
sage: fromKS2([0,0,1,1])
-2
```

(continues on next page)
```
sage: K(10/(-2)).is_square()
True

sage: KS3, gens, fromKS3, toKS3 = pSelmerGroup(K, [P2, P3, P5], 3)
sage: KS3
Vector space of dimension 3 over Finite Field of size 3
sage: gens
[1/2, 1/4*a + 1/4, a]
```

The to and from maps are inverses of each other:

```
sage: K.<a> = QuadraticField(-5)
sage: S = K.ideal(30).support()
sage: KS2, gens, fromKS2, toKS2 = pSelmerGroup(K, S, 2)
sage: KS2
Vector space of dimension 5 over Finite Field of size 2
sage: assert all(toKS2(fromKS2(v))==v for v in KS2)
sage: KS3, gens, fromKS3, toKS3 = pSelmerGroup(K, S, 3)
sage: KS3
Vector space of dimension 4 over Finite Field of size 3
sage: assert all(toKS3(fromKS3(v))==v for v in KS3)
```

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 `sage.rings.number_field.totallyreal_rel`, which handles the non-primitive case using relative extensions.

4.1.1 Algorithm

We use Hunter’s algorithm ([Coh2000], Section 9.3) with modifications due to Takeuchi [Tak1999] and the author [Voi2008].

We enumerate polynomials \( f(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_0 \). 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 [Tak1999] for more details.

4.1.2 Examples

In this first simple example, we compute the totally real quadratic fields of discriminant \( \leq 50 \).

```python
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]]
(continues on next page)```
Next, we compute all totally real quintic fields of discriminant $\leq 10^5$:

```
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 - 5*x^3 + 2*x^2 + 5*x + 1],
 [70601, x^5 - x^4 - 5*x^3 + 2*x^2 + 3*x - 1],
 [81509, x^5 - x^4 - 5*x^3 + 3*x^2 + 5*x - 2],
 [81589, x^5 - 6*x^3 + 8*x - 1],
 [89417, x^5 - 6*x^3 - x^2 + 8*x + 3]]
sage: len(ls)
9
```

We see that there are 9 such fields (up to isomorphism!).

See also [Mar1980].

### 4.1.3 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.

```python
sage.rings.number_field.totallyreal.enumerate_totallyreal_fields_prim(n, B, a=[], verbose=0, return_seqs=False, phc=False, keep_fields=False, t2=False, just_print=False, return_pari_objects=True)
```

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 $\text{length}(a) = d+1$, so in particular always $a[d] = 1$. 

Chapter 4. Totally Real Fields
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 \( B^*\log(B) \); 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

\footnotesize
\texttt{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 \( n \).

\footnotesize
\textbf{Note:} The bounds for \( n > 50 \) are not necessarily optimal.

\footnotesize
\textbf{INPUT:}

- \( n \) (integer) the degree

\textbf{OUTPUT:}

a lower bound on the root discriminant (as a real number)

\textbf{EXAMPLES:}

\footnotesize
\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:
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:
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 \(\leq 2000\).

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 \(\leq 10\).

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 + xF - 1/2*t - 1/2],
[1125, x^4 - 4*x^2 + 4*x + 1, xF^2 + (-1/2*t - 1/2)*xF + 1/2*t - 1/2],
[1600, x^4 - 6*x^2 + 4, xF^2 + xF - 1/2*t - 1/2],
[2000, x^4 - 5*x^2 + 5, xF^2 + xF - 1/2*t - 1/2],
[2225, x^4 - 2*x^3 - 4*x^2 + 5*x + 5, xF^2 + (-1/2*t + 1/2)*xF + 1/2*t - 1/2],
[2525, x^4 - 2*x^3 - 4*x^2 + 5*x + 5, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 1/2],
[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 - 1/2*t - 1/2],
[4400, x^4 - 7*x^2 + 11, xF^2 - 1/2*t - 1/2],
[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 - 1/2*t - 1/2],
[5225, x^4 - x^3 - 8*x^2 + 7*x + 11, xF^2 + (-1/2*t - 1/2)*xF - 1/2*t - 1/2],
[5725, x^4 - x^3 - 8*x^2 + 6*x + 11, xF^2 + (-1/2*t + 1/2)*xF - 1/2*t - 1/2],
[6125, x^4 - x^3 - 9*x^2 + 9*x + 11, xF^2 + (-1/2*t + 1/2)*xF - 1/2*t - 1/2],
[7225, x^4 - 11*x^2 + 9, xF^2 + (-1)*xF - 4],
[7600, x^4 - 9*x^2 + 19, xF^2 - 1/2*t - 1/2],

(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 \(\leq 17 \times 10^9\).

AUTHORS:


\[\texttt{sage: F.<t> = NumberField(ZZx([1,-4,3,1])}
\]
\[\texttt{sage: F.disc()}
\]
\[\texttt{49}
\]
\[\texttt{sage: enumerate_totallyreal_fields_rel(F, 3, 17*10^9) # not tested, too long time (258s...}
\]

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
- \(\text{verbose}\) – 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.

EXAMPLES:
sage: enumerate_totallyreal_fields_all(4, 2000)
[[725, x^4 - x^3 - 3*x^2 + x + 1],
 [1125, x^4 - x^3 - 4*x^2 + 4*x + 1],
 [1600, x^4 - 6*x^2 + 4],
 [1957, x^4 - 4*x^2 - x + 1],
 [2000, x^4 - 5*x^2 + 5]]
sage: enumerate_totallyreal_fields_all(1, 10)
[[1, x - 1]]

This function enumerates (primitive) totally real field extensions of degree \( m > 1 \) of the totally real field \( F \) with discriminant \( d \leq B \); optionally one can specify the first few coefficients, where the sequence \( a \) corresponds to a polynomial by

\[
a[d]*x^n + \ldots + a[0]*x^{(n-d)}
\]

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 [[lower, upper], ...] 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, 6, 8, 8, 10, 10, 10, 10, 12, 12, 14]
sage: len(ls)
19
```

A cubic field:

```
sage: x = polygen(QQ)
sage: K.<a> = NumberField(x^3 - 16*x +16)
sage: eps = 10e-6
sage: C = [[0-eps,5+eps]]
sage: v = sage.rings.number_field.totallyreal_rel.integral_elements_in_box(K, C)
sage: sorted(v)
[-1/4*a^2 + 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)

Bases: object

This class encodes the data used in the enumeration of totally real fields for relative extensions.

We do not give a complete description here. For more information, see the attached functions; all of these are used internally by the functions in totallyreal_rel.py, so see that file for examples and further documentation.

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:

```
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2+1'))
1
sage: sage.rings.number_field.totallyreal_data.easy_is_irreducible_py(pari('x^2-1'))
0
```

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 $\gamma_n$. 

346 Chapter 4. Totally Real Fields
INPUT:
• n – integer

OUTPUT:
• (an upper bound for) the Hermite constant $\gamma_n$

EXAMPLES:

```python
sage: hermite_constant(1) # trivial one-dimensional lattice
1.0
sage: hermite_constant(2) # Eisenstein lattice
1.15470053837925
sage: 2/sqrt(3.)
1.15470053837925
sage: hermite_constant(8) # E_8
2.0
```

Note: The upper bounds used can be found in [CS1999] and [CE2003].

AUTHORS:
• John Voight (2007-09-03)

```
sage.rings.number_field.totallyreal_data.int_has_small_square_divisor(d)
Returns the largest $a$ such that $a^2$ divides $d$ and $a$ has prime divisors < 200.
```

EXAMPLES:

```python
sage: from sage.rings.number_field.totallyreal_data import int_has_small_square_divisor
sage: int_has_small_square_divisor(500)
100
sage: is_prime(691)
True
sage: int_has_small_square_divisor(691)
1
sage: int_has_small_square_divisor(691^2)
1
```

```
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

$$r^x + (n-1-r)^y + z = s_i (i = 1,2,3)$$

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:

```python
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]
```
class sage.rings.number_field.totallyreal_data.tr_data
    Bases: object

This class encodes the data used in the enumeration of totally real fields.

We do not give a complete description here. For more information, see the attached functions; all of these are used internally by the functions in totallyreal.py, so see that file for examples and further documentation.

increment(verbos=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()
    sage: 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 = [...]

348 Chapter 4. Totally Real Fields
4.4 Enumeration of Totally Real Fields: PHC interface

AUTHORS:
  – John Voight (2007-10-10):
    • Zeroth attempt.

\texttt{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.

\textbf{INPUT:}

• \(n\) – integer, the degree
• \(a\) – list of integers, the coefficients

\textbf{OUTPUT:}

list of integers.

\textbf{Note:} This uses Newton’s relations, which are classical.

AUTHORS:
  • John Voight (2007-09-19)

\textbf{EXAMPLES:}

\begin{verbatim}
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]
\end{verbatim}
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 `QQbar`; the field of algebraic reals has abbreviation `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 `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 `NumberFieldElement`.

An algebraic number can be coerced into `ComplexIntervalField` (or `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:
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:

```python
sage: def disc1(b, c, d):
    ....:    return b**2*c**2 - 4*b**3*d - 4*c**3 + 18*b*c*d - 27*d**2
sage: def disc2(s1, s2, s3):
    ....:    return ((s1-s2)*(s1-s3)*(s2-s3))**2
sage: x = polygen(AA)
sage: p = x*(x-2)*(x-4)
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(-1, 1))
sage: s2 = AA.polynomial_root(cp, RIF(1, 3))
sage: s3 = AA.polynomial_root(cp, RIF(3, 5))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
sage: p = p + 1
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(-1, 1))
sage: s2 = AA.polynomial_root(cp, RIF(1, 3))
sage: s3 = AA.polynomial_root(cp, RIF(3, 5))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
sage: p = (x-sqrt(AA(2)))*(x-AA(2).nth_root(3))*(x-sqrt(AA(3)))
sage: cp = AA.common_polynomial(p)
sage: d, c, b, _ = p.list()
sage: s1 = AA.polynomial_root(cp, RIF(1.4, 1.5))
sage: s2 = AA.polynomial_root(cp, RIF(1.7, 1.8))
sage: s3 = AA.polynomial_root(cp, RIF(1.2, 1.3))
sage: disc1(b, c, d) == disc2(s1, s2, s3)
True
```

We can convert from symbolic expressions:

```python
sage: QQbar(sqrt(-5))
2.236067977499790?*I
sage: AA(sqrt(2) + sqrt(3))
3.146264369941973?
sage: QQbar(I)
I
sage: QQbar(I * golden_ratio)
1.618033988749895?*I
sage: AA(golden_ratio)**2 - AA(golden_ratio)
1
```
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
```

However, implicit coercion from Q[\(I\)] is only allowed when it is equipped with a complex embedding:

```
sage: i.parent()
Number Field in I with defining polynomial x^2 + 1 with I = 1*I
sage: QQbar(1) + i
I + 1
sage: K.<im> = QuadraticField(-1, embedding=None)
sage: QQbar(1) + im
Traceback (most recent call last):
  ...TypeError: unsupported operand parent(s) for +: 'Algebraic Field' and 'Number Field in im with defining polynomial x^2 + 1'
```

However, we can explicitly coerce from the abstract number field Q[\(I\)]. (Technically, this is not quite kosher, since we do not know whether the field generator is supposed to map to +\(I\) or −\(I\). We assume that for any quadratic field with polynomial \(x^2 + 1\), the generator maps to +\(I\).):

```
sage: pythag = QQbar(3/5 + 4*im/5); pythag
4/5*I + 3/5
sage: pythag.abs() == 1
True
```

We can implicitly coerce from algebraic reals to algebraic numbers:

```
sage: a = QQbar(1); a, a.parent()
(1, Algebraic Field)
```
Some computation with radicals:

```python
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
```

```python
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):

```python
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):

```python
sage: AA(2)/3
2/3
sage: QQbar(5/7)
5/7
sage: QQbar(1/3 - 1/4*I)
-1/4*I + 1/3
```

We can find the real and imaginary parts of an algebraic number (exactly):

```python
sage: phi
1.618033988749895?
```
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 \( \text{QQbar} \) as \( \text{AA}[I] \):)

```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,
... CIF(RIF(1, 1.2)))
sage: lhs = alpha^630 - 1
sage: rhs_num = (alpha^315 - 1) * (alpha^210 - 1) * (alpha^126 - 1)^2 * (alpha^90 - 1) * (alpha^3 - 1)^3 * (alpha^2 - 1)^5 * (alpha - 1)^3
sage: rhs_den = (alpha^35 - 1) * (alpha^15 - 1)^2 * (alpha^14 - 1)^2 * (alpha^5 - 1)^4 * (alpha - 1)^3
sage: rhs = rhs_num / rhs_den
sage: lhs
2.642040335819351?e44
```

(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:

\[
\text{sage: } \text{rt2} = \text{AA}(\sqrt{2})
\]
\[
\text{sage: } \text{rt3} = \text{AA}(\sqrt{3})
\]
\[
\text{sage: } \text{n} = (\text{rt2} + \text{rt3})^5; \text{n}
\]
\[
308.3018001722975
\]
\[
\text{sage: } \text{sage\_input(n)}
\]
\[
\text{R.<x> = AA[]}
\]
\[
\text{v1 = AA.polynomial\_root(AA.common\_polynomial(x^2 - 2), RIF(RR(1.414213562370949), RR(1.414213562370951))) + AA.polynomial\_root(AA.common\_polynomial(x^2 - 3), RIF(RR(1.7320508075688772), RR(1.7320508075688774)))}
\]
\[
\text{v2 = v1*v1}
\]
\[
\text{v2*v2*v1}
\]
But once exact computation is triggered, the computation tree is discarded, and we get a way to produce the number directly:

\[
\text{sage: } \text{n} == 109*\text{rt2} + 89*\text{rt3}
\]
\[
\text{True}
\]
\[
\text{sage: } \text{sage\_input(n)}
\]
\[
\text{R.<y> = QQ[]}
\]
\[
\text{v = AA.polynomial\_root(AA.common\_polynomial(y^4 - 4*y^2 + 1), RIF(RR(0.51763809020504148), RR(0.51763809020504159)))}
\]
\[
-109*v^3 - 89*v^2 + 327*v + 178
\]
We can also see that some computations (basically, those which are easy to perform exactly) are performed directly, instead of storing the computation tree:

\[
\text{sage: } \text{z3\_3} = \text{QQbar.zeta(3)} * 3
\]
\[
\text{sage: } \text{z4\_4} = \text{QQbar.zeta(4)} * 4
\]
Note that the `verify=True` argument to `sage_input` will always trigger exact computation, so running `sage_input` twice in a row on the same number will actually give different answers. In the following, running `sage_input` on `n` will also trigger exact computation on `rt2`, as you can see by the fact that the third output is different than the first:

```python
sage: 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(1.4142135623730949, 1.4142135623730951))
v*v
sage: sage_input(n, verify=True)  # Verified
AA(2)
```

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))
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?
sage: sage_input(one, verify=True)  # Verified
R1.<x> = QQbar[]
R2.<y> = QQ[]
v = AA.polynomial_root(AA.common_polynomial(y^4 - 4*y^2 + 1), RIF(0.51763809020504148, 1.0000000000000002))
```

We can pickle and unpickle algebraic fields (and they are globally unique):

5.1. Field of Algebraic Numbers 357
We can pickle and unpickle algebraic numbers:

```
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
```

```
sage: t = QQbar(sqrt(2)); type(t._descr)
<class 'sage.rings.qqbar.ANRoot'>
sage: loads(dumps(t)) == QQbar(sqrt(2))
True
```

```
sage: t.exactify(); type(t._descr)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: loads(dumps(t)) == QQbar(sqrt(2))
True
```

```
sage: t = ~QQbar(sqrt(2)); type(t._descr)
<class 'sage.rings.qqbar.ANUUnaryExpr'>
sage: loads(dumps(t)) == 1/QQbar(sqrt(2))
True
```

```
sage: t = QQbar(sqrt(2)) + QQbar(sqrt(3)); type(t._descr)
<class 'sage.rings.qqbar.ANBinaryExpr'>
sage: loads(dumps(t)) == QQbar(sqrt(2)) + QQbar(sqrt(3))
True
```

We can convert elements of `QQbar` and `AA` into the following types: `float`, `complex`, `RDF`, `CDF`, `RR`, `CC`, `RIF`, `CIF`, `ZZ`, and `QQ`, with a few exceptions. (For the arbitrary-precision types, `RR`, `CC`, `RIF`, and `CIF`, it can convert into a field of arbitrary precision.)

Converting from `QQbar` to a real type (`float`, `RDF`, `RR`, `RIF`, `ZZ`, or `QQ`) succeeds only if the `QQbar` is actually real (has an imaginary component of exactly zero). Converting from either `AA` or `QQbar` to `ZZ` or `QQ` succeeds only if the number actually is an integer or rational. If conversion fails, a `ValueError` will be raised.

Here are examples of all of these conversions:

```
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)

(continues on next page)```
[42.0, 3.1428571428571432, 1.6180339887498959, -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+0.9510565162951536j)]

sage: convert_test_all(RDF)
[42.0, 3.1428571428571432, 1.6180339887498959, -13.0, 1.6181818181818182, -2.6457513110645907, None]

sage: convert_test_all(CDF)
[42.0, 3.1428571428571432, 1.6180339887498959, -13.0, 1.6181818181818182, -2.6457513110645907, 0.30901699437494745 + 0.9510565162951536*I]

sage: convert_test_all(RR)
[42.0000000000000, 3.14285714285714, 1.61803398874989, -13.0000000000000, 1.61818181818182, -2.64575131106459, None]

sage: convert_test_all(CC)
[42.0000000000000, 3.14285714285714, 1.61803398874989, -13.0000000000000, 1.61818181818182, -2.64575131106459, 0.309016994374947 + 0.951056516295154*I]

sage: convert_test_all(RIF)
[42.0, 3.142857142857143?, 1.618033988749895?, -13, 1.618181818181819?, -2.645751311064591?, None]

sage: convert_test_all(CIF)
[42.0, 3.142857142857143?, 1.618033988749895?, -13, 1.618181818181819?, -2.645751311064591?, 0.3090169943749474? + 0.9510565162951536?*I]

sage: convert_test_all(ZZ)
[42, None, None, -13, None, None, None]

sage: convert_test_all(QQ)
[42, 22/7, None, -13, 89/55, None, None]

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
1.00000000000000
sage: cy
0.75

sage: ax = polygen(AA)
sage: x2 = AA.polynomial_root(256*ax**8 - 128*ax**7 - 448*ax**6 + 192*ax**5 + 240*ax**4 -
40*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 $x == x2$ and $y == y2$ but this is currently infinitely long.

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'>

exactify()

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)))
R.<x> = AA[]
2*AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949),RR(1.4142135623730951)))
sage: rt2 = sqrt(AA(2))
sage: one = rt2/rt2
sage: n = one+3
sage: sage_input(n)
R.<x> = AA[]
v = AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951)))
v/v + 3
sage: one == 1
True
sage: sage_input(n)
1 + AA(3)
sage: rt3 = QQbar(sqrt(3))
sage: one = rt3/rt3
sage: n = sqrt(AA(2))+one
sage: one == 1
True
sage: sage_input(n)
R.<x> = AA[]
QQbar.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949), RR(1.4142135623730951))) + 1
sage: from sage.rings.qqbar import *
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 subclasses 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:

```
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: rt2.exactify()
sage: rt2._descr.is_simple()
True
sage: rt2b.exactify()
sage: rt2b._descr.is_simple()
False
```
sage: rt2b.simplify()
sage: rt2b._descr.is_simple()
True

neg(n)
Negation of self.

EXAMPLES:

sage: a = QQbar(sqrt(2))
sage: b = a._descr
sage: b.neg(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>

norm(n)
Field norm of self from \( \mathbb{Q} \) to its real subfield \( A \), i.e. the square of the usual complex absolute value.

EXAMPLES:

sage: a = QQbar(sqrt(-7))
sage: b = a._descr
sage: b.norm(a)
<sage.rings.qqbar.ANUnaryExpr object at ...>

real(n)
Real part of self.

EXAMPLES:

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)
Root 3.146264369941972342? of x^2 - 9.89897948556636?

conjugate(n)
Negation of self.
**EXAMPLES:**

\begin{verbatim}
    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
\end{verbatim}

**exactify()**

Return an exact representation of self.

Since self is already exact, just return self.

**EXAMPLES:**

\begin{verbatim}
    sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.
    sage: exactify()
    sage: type(v)
    <class 'sage.rings.qqbar.ANExtensionElement'>
    sage: v.exactify()
    is v
    True
\end{verbatim}

**field_element_value()**

Return the underlying number field element.

**EXAMPLES:**

\begin{verbatim}
    sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.
    sage: exactify()
    sage: v.field_element_value()
    a
\end{verbatim}

**generator()**

Return the `AlgebraicGenerator` object corresponding to self.

**EXAMPLES:**

\begin{verbatim}
    sage: v = (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.
    sage: exactify()
    sage: v.generator()
    Number Field in a with defining polynomial y^2 - y - 1 with a in 1.618033988749895?
\end{verbatim}

**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:**

\begin{verbatim}
    sage: I = QQbar(I)
    sage: sage_input(3+4*I, verify=True)
\end{verbatim}
# Veriﬁed
QQbar(3 + 4*I)
sage: v = QQbar.zeta(3) + QQbar.zeta(5)
sage: v - v == 0
True
sage: sage_input(vector(QQbar, (4-3*I, QQbar.zeta(7))), verify=True)
# Veriﬁed
R.<y> = QQ[]
vector(QQbar, [4 - 3*I, QQbar.polynomial_root(AA.common_polynomial(y^6 + y^5 + y^4 + y^3 + y^2 + y + 1), CIF(RIF(RR(0.62348980185873348), RR(0.62348980185873359)), RIF(RR(0.7818314824680298), RR(0.78183148246802991))))])
sage: sage_input(v, verify=True)
# Veriﬁed
R.<y> = QQ[]
v = QQbar.polynomial_root(AA.common_polynomial(y^8 - y^7 + y^5 - y^4 + y^3 - y + 1), CIF(RIF(RR(0.91354545764260087), RR(0.91354545764260098)), RIF(RR(0.40673664307580015), RR(0.40673664307580021))))
v^5 + v^3
sage: v = QQbar(sqrt(AA(2)))
sage: v.exactify()
sage: sage_input(v, verify=True)
# Veriﬁed
R.<y> = QQ[]
QQbar(AA.polynomial_root(AA.common_polynomial(y^2 - 2), RIF(RR(1.414213562370951)), RIF(RR(1.414213562370951))))
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
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? - 1.57312184970987*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? - 1.57312184970987*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.

5.1. Field of Algebraic Numbers
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.
    _exactify()

sage: type(v)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: v.minpoly()
x^2 - x - 1
```

**neg()**
Negation of self.

EXAMPLES:
Algebraic Numbers and Number Fields, Release 9.6

```python
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:**

```python
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 $2\pi$ times some rational number in $[1/2, -1/2)$, return that rational; otherwise, return None.

**EXAMPLES:**

```python
sage: a = QQbar(sqrt(-2)) + QQbar(sqrt(3))
sage: a.exactify()
sage: b = a._descr
sage: type(b)
<class 'sage.rings.qqbar.ANExtensionElement'>
sage: b.rational_argument(a) is None
True
sage: x = polygen(QQ)
sage: a = (x^4 + 1).roots(QQbar, multiplicities=False)[0]
sage: a.exactify()
sage: b = a._descr
sage: b.rational_argument(a)
-3/8
```

**simplify(n)**
Compute an exact representation for this descriptor, in the smallest possible number field.

**INPUT:**

- n – The element of AA or QQbar corresponding to this descriptor.

**EXAMPLES:**

```python
sage: rt2 = AA(sqrt(2))
sage: rt3 = AA(sqrt(3))
```
sage: rt2b = rt3 + rt2 - rt3
sage: rt2b.exactify()

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 ANDescr that represents an arbitrary rational. This class is private, and should not be used directly.

abs(n)
Absolute value of self.

EXAMPLES:

sage: a = QQbar(3)
sage: b = a._descr
sage: b.abs(a)
3

angle()
Return a rational number $q \in (-1/2, 1/2]$ such that self is a rational multiple of $e^{2\pi i q}$. Always returns 0, since this element is rational.

EXAMPLES:

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:

sage: a = QQbar(1/3)._descr
sage: a.exactify() is a
True

generator()
Return an AlgebraicGenerator object associated to this element. Returns the trivial generator, since self is rational.

EXAMPLES:

sage: QQbar(0)._descr.generator()
Trivial generator

handle_sage_input(sib, coerce, is_gqbar)
Produce an expression which will reproduce this value when evaluated, and an indication of whether this value is worth sharing (always False, for rationals).
EXAMPLES:
```
sage: 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:
```
sage: a = QQbar(3)
sage: b = a._descr
sage: b.invert(a)
1/3
```

is_complex()

Return False, since rational numbers are real

EXAMPLES:
```
sage: QQbar(1/7)._descr.is_complex()
False
```

is_simple()

Checks whether this descriptor represents a value with the same algebraic degree as the number field associated with the descriptor.

This is always true for rational numbers.

EXAMPLES:
```
sage: AA(1/2)._descr.is_simple()
True
```

minpoly()

Return the min poly of self over $\mathbb{Q}$.

EXAMPLES:
```
sage: QQbar(7)._descr.minpoly()
x - 7
```

neg($n$)

Negation of self.
EXAMPLES:

```python
sage: a = QQbar(3)
sage: b = a._descr
type(b)
<class 'sage.rings.qqbar.ANRational'>
sage: b.neg(a)
-3
```

**rational_argument(n)**

Return the argument of self divided by $2\pi$, or None if this element is 0.

EXAMPLES:

```python
sage: QQbar(3)._descr.rational_argument(None)
0
sage: QQbar(-3)._descr.rational_argument(None)
1/2
sage: QQbar(0)._descr.rational_argument(None) is None
True
```

**scale()**

Return a rational number $r$ such that $self$ is equal to $re^{2\pi iq}$ for some $q \in (-1/2, 1/2]$. In other words, just return self as a rational number.

EXAMPLES:

```python
sage: QQbar(-3)._descr.scale()
-3
```

**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:

```python
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 0.5000000000000000? - 2.
\rightarrow 397915761656360?*I
```

**exactify()**

Return either an `ANRational` or an `ANExtensionElement` with the same value as this number.

EXAMPLES:
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...
˓→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 - 3), RIF(RR(1.7320508075688772), RR(1.7320508075688774))...v4 = v3*v3
v5 = v4*v4
AA.polynomial_root(AA.common_polynomial(x^4 - v5*v5*v3), RIF(RR(2.4176921938267881)), RIF(RR(1.4804728524798114), RR(1.4804728524798112)))
```

```sage
sage: sage_input((sqrt(QQbar(-7))**(5/7))**(9/4))
R.<x> = QQbar[]
v1 = QQbar.polynomial_root(AA.common_polynomial(x^2 + 7), CIF(RIF(RR(0)),...
˓→RIF(RR(2.6457513110645903), RR(2.6457513110645907)))))
v2 = v1*v1
v3 = QQbar.polynomial_root(AA.common_polynomial(x^7 - v2*v2*v1), CIF(RIF(RR(0.8693488875796217), RR(0.86934888757962181)), RIF(RR(1.8052215661454434), RR(1.8052215661454436))))
v4 = v3*v3
v5 = v4*v4
QQbar.polynomial_root(AA.common_polynomial(x^4 - v5*v5*v3), CIF(RIF(-RR(3.8954086044650791), -RR(3.8954086044650786)), RIF(RR(2.7639398015408925), RR(2.7639398015408929))))
```

```sage
sage: x = polygen(QQ)
sage: sage_input(AA.polynomial_root(x^2-x-1, RIF(1, 2)), verify=True)
# Verified
R.<y> = QQ[]
AA.polynomial_root(AA.common_polynomial(y^2 - y - 1), RIF(RR(1.6180339887498947), RR(1.6180339887498949)))
```

```sage
(continues on next page)
```
is_complex()  
Whether this is a root in $\overline{\mathbb{Q}}$ (rather than $\mathbb{A}$). Note that this may return True even if the root is actually real, as the second example shows; it does not trigger exact computation to see if the root is real.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: (x^2 - x - 1).roots(ring=AA, multiplicities=False)[1]._descr.is_complex()
False
```

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:

```
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)
```

---

class sage.rings.qqbar.ANUnaryExpr(arg, op)

Bases: sage.rings.qqbar.ANDescr

Initialize this ANUnaryExpr.

EXAMPLES:
```sage
t = ~QQbar(sqrt(2)); type(t._descr)  # indirect doctest
<class 'sage.rings.qqbar.ANUnaryExpr'>
```

**exactify()**

Trigger exact computation of self.

**EXAMPLES:**

```sage
v = (-QQbar(sqrt(2)))._descr
type(v)
<class 'sage.rings.qqbar.ANUnaryExpr'>
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 ANUnaryExpr).

**EXAMPLES:**

```sage
sage_input(-sqrt(AA(2)), verify=True)  # Verified
R.<x> = AA[]
-AA.polynomial_root(AA.common_polynomial(x^2 - 2), RIF(RR(1.4142135623730949),
   ...RR(1.4142135623730951)))(x)
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)))(x)
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))))(x)
```

(continues on next page)

### 5.1. Field of Algebraic Numbers

373
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.<y> = QQ[]
QQbar(QQbar.polynomial_root(AA.common_polynomial(y^2 + y + 1), CIF(RIF(-RR(0.
˓→50000000000000011), -RR(0.49999999999999994)), RIF(RR(0.8660254037844386),␣
˓→RR(0.86602540378443871))).real())
sage: from sage.rings.qqbar import *
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:
˓→{atomic:AA}.common_polynomial}({binop:- {binop:** {gen:x {constr_parent:
˓→{subscr: {atomic:AA}{{atomic:'x'}}} with gens: ('x',)}} {atomic:2} {atomic:2}
˓→}}, {call: {atomic:RIF}({call: {atomic:RR}{{atomic:1.4142135623730949}})},
˓→{call: {atomic:RR}}{{atomic:1.4142135623730951}}}))), True})
sage: unexp.handle_sage_input(sib, False, True)
({call: {atomic:QQbar}{unop:~ {call: {getattr: {atomic:AA}.polynomial_root}(
˓→{call: {getattr: {atomic:AA}.common_polynomial}({binop:- {binop:** {gen:x
˓→{constr_parent: {subscr: {atomic:AA}{{atomic:'x'}}} with gens: ('x',)}}
˓→{atomic:2} {atomic:2}})}, {call: {atomic:RIF}{{call: {atomic:RR}{{atomic:1.
˓→14142135623730949}}}, {call: {atomic:RR}}{{atomic:1.4142135623730951}}}))), 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, sage.rings.abc.AlgebraicField  
The field of all algebraic complex numbers.

algebraic_closure()  
Return the algebraic closure of this field.

As this field is already algebraically closed, just returns self.

EXAMPLES:
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
- \(\text{prec}\) – precision of approximate field to return
- \(\text{extras}\) – (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):
... Not Implemented Error

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 \(\text{gens()}\).

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:

```python
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 QQbar.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: 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:

- poly_degree - default: 2 - degree of the random polynomial over the integers of which the returned algebraic number is a root. This is not necessarily the degree of the minimal polynomial of the number. Increase this parameter to achieve a greater diversity of algebraic numbers, at a cost of greater computation time. You can also vary the distribution of the coefficients but that will not vary the degree of the extension containing the element.

- args, kwds - arguments and keywords passed to the random number generator for elements of ZZ, the integers. See random_element() for details, or see example below.

OUTPUT:

An element of QQbar, the field of algebraic numbers (see sage.rings.qqbar).

ALGORITHM:
A polynomial with degree between 1 and poly_degree, with random integer coefficients is created. A root of this polynomial is chosen at random. The default degree is 2 and the integer coefficients come from a distribution heavily weighted towards 0, ±1, ±2.

EXAMPLES:

```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)
sage: b                  # random
-0.8642649077479498? - 0.5995098147478391?*I
sage: b in QQbar
True
```

Parameters for the distribution of the integer coefficients of the polynomials can be passed on to the random element method for integers. For example, 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 ensure 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()
```

(continues on next page)
I

\begin{verbatim}
  sage: QQbar.zeta(5)
  0.3090169943749474? + 0.9510565162951536?*I
  sage: QQbar.zeta(3000)
  0.999997806755380? + 0.002094393571219374?*I
\end{verbatim}

class sage.rings.qqbar.AlgebraicField_common

Bases: sage.rings.abc.AlgebraicField_common

Common base class for the classes AlgebraicRealField and AlgebraicField.

characteristic()

Return the characteristic of this field.

Since this class is only used for fields of characteristic 0, this always returns 0.

EXAMPLES:

\begin{verbatim}
  sage: AA.characteristic()
  0
\end{verbatim}

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:

\begin{verbatim}
  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.?e-18 + 2.236067977499790?*I
  sage: b = QQbar.polynomial_root(p, RIF(1, 2)); b
  1.732050807568878?
\end{verbatim}

These “common polynomials” can be shared between real and complex roots:

\begin{verbatim}
  sage: p = AA.common_polynomial(x^3 - x - 1)
  sage: r1 = AA.polynomial_root(p, RIF(1.3, 1.4)); r1
  1.32471795724475?
  sage: r2 = QQbar.polynomial_root(p, CIF(RIF(-0.7, -0.6), RIF(0.5, 0.6))); r2
  -0.6623589786223730? + 0.5622795120623013?*I
\end{verbatim}
default_interval_prec()
Return the default interval precision used for root isolation.

EXAMPLES:

```
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:

```
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:

```
sage: from sage.rings.qqbar import AlgebraicGenerator
sage: x = polygen(QQ); f = x^4 + x + 17
sage: nf = NumberField(f, name='a')
sage: b = f.roots(QQbar)[0][0]
sage: root = b._descr
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.conjugate()
Number Field in a with defining polynomial x^4 + x + 17 with a in -1.436449974830917 + 1.374535713065812*I
```

field()
Return the number field attached to self.

EXAMPLES:

```
sage: from sage.rings.qqbar import AlgebraicGenerator
sage: x = polygen(QQ); f = x^4 + x + 17
sage: nf = NumberField(f, name='a')
sage: b = f.roots(QQbar)[0][0]
sage: root = b._descr
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.field()
Number Field in a with defining polynomial x^4 + x + 17 with a in -1.436449974830917 + 1.374535713065812*I
```

```
sage: from sage.rings.qqbar import AlgebraicGenerator
sage: x = polygen(QQ); f = x^4 + x + 17
sage: nf = NumberField(f, name='a')
sage: b = f.roots(QQbar)[0][0]
sage: root = b._descr
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.field()
Number Field in a with defining polynomial x^4 + x + 17 with a in -1.436449974830917 + 1.374535713065812*I
```

5.1. Field of Algebraic Numbers
is_complex()  
Return True if this is a generator for a non-real number field.

EXAMPLES:

```
sage: z7 = QQbar.zeta(7)
sage: g = z7._descr._generator
sage: g.is_complex()
True
```

```
sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator
sage: y = polygen(QQ, 'y')
sage: x = polygen(QQbar)
sage: nf = NumberField(y^2 - y - 1, name='a', check=False)
sage: root = ANRoot(x^2 - x - 1, RIF(1, 2))
sage: gen = AlgebraicGenerator(nf, root)
sage: gen.is_complex()
False
```

is_trivial()  
Return true iff this is the trivial generator (alpha == 1), which does not actually extend the rationals.

EXAMPLES:

```
sage: from sage.rings.qqbar import qq_generator
sage: qq_generator.is_trivial()
True
```

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
```

(continues on next page)
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:

```python
sage: from sage.rings.qqbar import AlgebraicGenerator, ANRoot, qq_generator
sage: _.<y> = QQ['y']
```

```python
def super_poly(super, checked=None):
    # implementation
```

```python
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: gen2_3.super_poly(gen2)
-a^3 + 3*a
sage: gen3.super_poly(gen2_3)
-a^2 + 2
```

union(other)

Given generators alpha and beta, alpha.union(beta) gives a generator for the number field \( Q[\alpha][\beta] \).

EXAMPLES:

```python
sage: from sage.rings.qqbar import ANRoot, AlgebraicGenerator, qq_generator
sage: _.<y> = QQ['y']
```

```python
def union(other):
    # implementation
```

```python
def super_poly(super, checked=None):
    # implementation
```

```python
def union(other):
    # implementation
```

```python
def super_poly(super, checked=None):
    # implementation
```

```python
def union(other):
    # implementation
```

```python
sage: from sage.rings.qqbar import AlgebraicGenerator, ANRoot, qq_generator
sage: _.<y> = QQ['y']
```

```python
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?
```

```python
sage: gen2_3.super_poly(gen2)
-a^3 + 3*a
sage: gen3.super_poly(gen2_3)
-a^2 + 2
```

(continues on next page)
sage: gen2.union(qq_generator) is gen2
   True
sage: qq_generator.union(gen3) is gen3
   True
sage: gen2.union(gen3)
Number Field in a with defining polynomial \(y^4 - 4y^2 + 1\) with \(a\) in \(0\).
   \(\sim\)5176380902050415?

class sage.rings.qqbar.AlgebraicGeneratorRelation(child1, child1_poly, child2, child2_poly, parent)
Bases: sage.structure.sage_object.SageObject
A simple class for maintaining relations in the lattice of algebraic extensions.

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.

_richcmp_(other, op)
Compare two algebraic numbers, lexicographically. (That is, first compare the real components; if the real components are equal, compare the imaginary components.)

EXAMPLES:

sage: x = QQbar.zeta(3); x
   -0.500000000000000? + 0.866025403784439?*I
sage: QQbar(-1) < x
   True
sage: QQbar(-1/2) < x
   True
sage: QQbar(0) > x
   True

One problem with this lexicographic ordering is the fact that if two algebraic numbers have the same real component, that real component has to be compared for exact equality, which can be a costly operation. For the special case where both numbers have the same minimal polynomial, that cost can be avoided, though (see trac ticket #16964):

sage: x = polygen(ZZ)
sage: p = 69721504*x^8 + 251777664*x^6 + 329532012*x^4 + 184429548*x^2 +
   \sim\rightarrow 37344321
sage: sorted(p.roots(QQbar, False))
   [-0.0221204634374361? - 1.090991904211621?*I,
   -0.0221204634374361? + 1.090991904211621?*I,
   -0.8088604911480535?*I,
   0.?e-215 - 0.7598602580415435?*I,
   0.?e-229 + 0.7598602580415435?*I,
   0.8088604911480535?*I,
   0.0221204634374361? - 1.090991904211621?*I,
   0.0221204634374361? + 1.090991904211621?*I]

It also works for comparison of conjugate roots even in a degenerate situation where many roots have the same real part. In the following example, the polynomial \(p_2\) is irreducible and all its roots have real part equal to 1:
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

Though, comparing roots which are not equal or conjugate is much slower because the algorithm needs to check the equality of the real parts:

sage: sorted(p2.roots(QQbar, False))
[1.000000000000000? - 4.016778562562223?*I,
 1.000000000000000? - 3.850538755978243?*I,
 1.000000000000000? - 3.390564396412898?*I,
 ...]

complex_exact(field)
Given a ComplexField, return the best possible approximation of this number in that field. Note that if either component is sufficiently close to the halfway point between two floating-point numbers in the corresponding RealField, then this will trigger exact computation, which may be very slow.

EXAMPLES:

sage: a = QQbar.zeta(9) + QQbar(I) + QQbar.zeta(9).conjugate(); a
1.532088886237957? + 1.000000000000000?*I
sage: a.complex_exact(CIF)
1.532088886237957? + 1*I

complex_number(field)
Given the complex field field compute an accurate approximation of this element in that field.

The approximation will be off by at most two ulp’s in each component, except for components which are very close to zero, which will have an absolute error at most $2^{-\text{prec}+1}$ where prec is the precision of the field.

EXAMPLES:

sage: a = QQbar.zeta(5)
sage: a.complex_number(CC)
0.309016994374947 + 0.951056516295154*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)
```

```python
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 number.

That is, find the smallest positive integer \(n\) such that \(x^n = 1\). If there is no such \(n\), returns \(+\infty\).

We first check that \(\text{abs}(x)\) 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
sage: QQbar(1).multiplicative_order()
1
sage: QQbar(-I).multiplicative_order()
4
sage: QQbar(707/1000 + 707/1000*I).multiplicative_order()
+Infinity
sage: QQbar(3/5 + 4/5*I).multiplicative_order()
+Infinity
```

**norm()**

Return \(self \ast self.conjugate()\).
This is the algebraic definition of norm, if we view QQbar as AA[I].

EXAMPLES:

```python
sage: QQbar(3 + 4*I).norm()
25
sage: type(QQbar(I).norm())
<class 'sage.rings.qqbar.AlgebraicReal'>
sage: QQbar.zeta(1007).norm()
1.000000000000000?
```

**rational_argument()**

Return the argument of self, divided by 2π, as long as this result is rational. Otherwise returns None. Always triggers exact computation.

EXAMPLES:

```python
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()  # long time
2/7
sage: (QQbar.zeta(73)**5).rational_argument()
5/73
sage: (QQbar.zeta(3)**65536).rational_argument()
1/3
```

**real()**

Return the real part of self.

EXAMPLES:

```python
sage: QQbar.zeta(5).real()
0.3090169943749474?
```

### class sage.rings.qqbar.AlgebraicNumberPowQQAction(G, S)

Bases: sage.categories.action.Action

Implement powering of an algebraic number (an element of QQbar or AA) by a rational.

This is always a right action.

**INPUT:**

- G – must be QQ
- S – the parent on which to act, either AA or QQbar.

**Note:** To compute \( x^{a/b} \), we take the \( b \)’th root of \( x \); then we take that to the \( a \)’th power. If \( x \) is a negative algebraic real and \( b \) is odd, take the real \( b \)’th root; otherwise take the principal \( b \)’th root.

EXAMPLES:
In \texttt{QQbar}:

```
\begin{verbatim}
sage: QQbar(2)^(1/2)
1.414213562373095?
sage: QQbar(8)^(2/3)
4
sage: QQbar(8)^(2/3) == 4
True
sage: x = polygen(QQbar)
sage: phi = QQbar.polynomial_root(x^2 - x - 1, RIF(1, 2))
sage: tau = QQbar.polynomial_root(x^2 - x - 1, RIF(-1, 0))
sage: rt5 = QQbar(5)^(1/2)
sage: phi^10 / rt5
55.00363612324742?
sage: tau^10 / rt5
0.003636123247413266?
sage: (phi^10 - tau^10) / rt5
55.00000000000000?
sage: (phi^10 - tau^10) / rt5 == fibonacci(10)
True
sage: QQbar(-8)^(1/3)
1.000000000000000? + 1.732050807568878?*I
sage: (QQbar(-8)^(1/3))^3
-8
sage: a = QQbar.zeta(7)^(1/3); a
0.9555728057861407? + 0.2947551744109043?*I
sage: a == QQbar.zeta(21)
True
sage: QQbar.zeta(7)^6
0.6234898018587335? - 0.7818314824680299?*I
sage: (QQbar.zeta(7)^6)^(1/3) * QQbar.zeta(21)
1.000000000000000? + 0.7e-17*I
\end{verbatim}
```

In \texttt{AA}:

```
\begin{verbatim}
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))
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?
\end{verbatim}
```

(continues on next page)
55.00000000000000?
sage: (phi^10 - tau^10) / rt5 == fibonacci(10)
True
sage: (phi^50 - tau^50) / rt5 == fibonacci(50)
True

\begin{class}\texttt{sage.rings.qqbar.AlgebraicNumber\_base}(parent, x)\end{class}
Bases: \texttt{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).

\texttt{AlgebraicNumber} objects can be created using \texttt{QQbar (== AlgebraicNumberField())}, and \texttt{AlgebraicReal} objects can be created using \texttt{AA (== AlgebraicRealField())}. They can be created either by coercing a rational or a symbolic expression, or by using the \texttt{QQbar.polynomial\_root()} or \texttt{AA.polynomial\_root()} method to construct a particular root of a polynomial with algebraic coefficients. Also, \texttt{AlgebraicNumber} and \texttt{AlgebraicReal} are closed under addition, subtraction, multiplication, division (except by 0), and rational powers (including roots), except that for a negative \texttt{AlgebraicReal}, taking a power with an even denominator returns an \texttt{AlgebraicNumber} instead of an \texttt{AlgebraicReal}.

\texttt{AlgebraicNumber} and \texttt{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.

\textbf{EXAMPLES:}

\begin{sageblock}
sage: sqrt(QQbar(2))
1.414213562373095?
sage: sqrt(QQbar(2))^2 == 2
True
sage: x = polygen(QQbar)
sage: phi = QQbar.polynomial_root(x^2 - x - 1, RIF(1, 2))
sage: phi
1.618033988749895?
sage: phi^2 == phi+1
True
sage: AA(sqrt(65537))
256.00019531175495?
\end{sageblock}

\texttt{as\_number\_field\_element}(\texttt{minimal=False, embedded=False, prec=53})
Return a number field containing this value, a representation of this value as an element of that number

\textbf{5.1. Field of Algebraic Numbers}
field, and a homomorphism from the number field back to $\mathbb{A}$ or $\mathbb{Q}\bar{\mathbb{Q}}$.

**INPUT:**

- **minimal** – Boolean (default: False). Whether to minimize the degree of the extension.
- **embedded** – Boolean (default: False). Whether to make the NumberField embedded.
- **prec** – integer (default: 53). The number of bit of precision to guarantee finding real roots.

This may not return the smallest such number field, unless minimal=True is specified.

To compute a single number field containing multiple algebraic numbers, use the function `number_field_elements_from_algebraics` instead.

**EXAMPLES:**

```python
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?)
```

```python
sage: x = polygen(ZZ)
sage: p = x^3 + x^2 + x + 17
sage: (rt,) = p.roots(ring=AA, multiplicities=False); rt
-2.804642726932742?
```

```python
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?)
```

```python
sage: elt == rt
False
```

```python
sage: AA(elt)
Traceback (most recent call last):
  ...  
ValueError: need a real or complex embedding to convert a non rational element of a number field into an algebraic number
```

```python
sage: hom(elt) == rt
True
```

Creating an element of an embedded number field:

```python
sage: (nf, elt, hom) = rt.as_number_field_element(embedded=True)
```

```python
sage: nf.coerce_embedding()
Generic morphism:  
  From: Number Field in a with defining polynomial y^3 - 2*y^2 - 31*y - 50 with a = 7.237653139801104?  
  To:   Algebraic Real Field  
  Defn: a -> 7.237653139801104?
```

```python
sage: elt
a^2 - 5*a - 19
```
A complex algebraic number as an element of an embedded number field:

```
sage: num = QQbar(sqrt(2) + 3^(1/3)*I)
sage: nf, elt, hom = num.as_number_field_element(embedded=True)
sage: hom(elt).parent()  # QQbar
True
sage: nf.coerce_embedding()  # is not None
True
sage: QQbar(elt) == num == hom(elt)
True
```

We see an example where we do not get the minimal number field unless we specify minimal=True:

```
sage: 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 |---> 1.732050807568878?)
```

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:

```
sage: QQbar(5/3).degree()
1
sage: sqrt(QQbar(2)).degree()
2
sage: QQbar(17).nth_root(5).degree()
5
```
**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 (or ball) field (real or complex, as appropriate) of precision \( p \), compute an interval representation of self with \( \text{diameter()} \) at most \( 2^{-p} \); then round that representation into the given field. Here \( \text{diameter()} \) is relative diameter for intervals not containing 0, and absolute diameter for intervals that do contain 0; thus, if the returned interval does not contain 0, it has at least \( p - 1 \) good bits.

**EXAMPLES:**

```python
sage: RIF64 = RealIntervalField(64)
sage: x = AA(2).sqrt()
sage: y = x*x
sage: y = 1000 * y - 999 * y
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
```

```python
sage: x.interval(CBF)  # abs tol 1e-16
[0.8412535328311812 +/- 3.12e-17] + [0.5406408174555976 +/- 1.79e-17]*I
```

The following implicitly use this method:

```python
sage: RIF(AA(5).sqrt())
2.236067977499790?
```

```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:

```python
sage: x = AA(2).sqrt()
sage: x.interval_fast(RIF)
1.414213562373095?
sage: x.interval_fast(RealIntervalField(200))
1.41421356237309504880168872420969807857?

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:

```python
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:

```python
sage: AA(2).is_square()
True
sage: AA(-2).is_square()
False
sage: QQbar(-2).is_square()
True
```
sage: QQbar(I).is_square()
True

minpoly()

Compute the minimal polynomial of this algebraic number. The minimal polynomial is the monic polynomial of least degree having this number as a root; it is unique.

EXAMPLES:

sage: QQbar(4).sqrt().minpoly()
x - 2
sage: ((QQbar(2).nth_root(4))^2).minpoly()
x^2 - 2
sage: v = sqrt(QQbar(2)) + sqrt(QQbar(3)); v
3.146264369941973?

sage: p = v.minpoly(); p
x^4 - 10*x^2 + 1
sage: p(RR(v.real()))
1.31006316905768e-14

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)
1.600000000000000? + 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)
[1.069553932363986? + 0.2127475047267431?*I, -0.2127475047267431? + 1.069553932363986?*I, -1.069553932363986? - 0.2127475047267431?*I, 0.2127475047267431? - 1.069553932363986?*I]
radical_expression()

Attempt to obtain a symbolic expression using radicals. If no exact symbolic expression can be found, the algebraic number will be returned without modification.

EXAMPLES:

```python
sage: AA(1/sqrt(5)).radical_expression()
sqrt(1/5)
sage: AA(sqrt(5 + sqrt(5))).radical_expression()
sqrt(sqrt(5) + 5)
sage: QQbar.zeta(5).radical_expression()
1/4*sqrt(5) + 1/2*sqrt(-1/2*sqrt(5) - 5/2) - 1/4
sage: a = QQ[x](x^7 - x - 1).roots(AA, False)[0]
sage: a.radical_expression()
1.112775684278706?
sage: a.radical_expression().parent() == SR
False
sage: a = sorted(QQ[x](x^7-x-1).roots(QQbar, False), key=imag)[0]
sage: a.radical_expression()
-0.3636235193291805? - 0.9525611952610331?*I
sage: QQbar.zeta(5).imag().radical_expression()
1/2*sqrt(1/2*sqrt(5) + 5/2)
sage: AA(5/3).radical_expression()
5/3
sage: AA(5/3).radical_expression().parent() == SR
True
sage: QQbar(0).radical_expression()
0
```

simplify()

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()
```

sqrt(all=False, extend=True)

Return the square root(s) of this number.

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:

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: 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: 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: 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: 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: x = polygen(QQ)
sage: 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: 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: x = polygen(QQ); f = x^3 - 7
sage: g = AA.common_polynomial(f)
sage: g.is_complex()
False
sage: QQbar.common_polynomial(x^3 - QQbar(I)).is_complex()
True
```

poly()  
Return the underlying polynomial of self.

EXAMPLES:

```
sage: x = polygen(QQ)
sage: f = x^3 - 7
sage: g = AA.common_polynomial(f)
sage: g.poly()
y^3 - 7
```

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
sage: ((5+AA(5).sqrt())/2).sqrt() == 2*QQbar.zeta(5).imag()
True
sage: AA(3).sqrt() + AA(2).sqrt() < 3
False
```

ceil()  
Return the smallest integer not smaller than self.

EXAMPLES:

```
sage: AA(sqrt(2)).ceil()
2
sage: AA(-sqrt(2)).ceil()
-1
sage: AA(42).ceil()
42
```

conjugate()  
Return the complex conjugate of self, i.e. returns itself.

EXAMPLES:
sage: a = AA(sqrt(2) + sqrt(3))
sage: a.conjugate()
3.146264369941973?
sage: a.conjugate() is a
True

floor()
Return the largest integer not greater than self.

EXAMPLES:

sage: AA(sqrt(2)).floor()
1
sage: AA(-sqrt(2)).floor()
-2
sage: AA(42).floor()
42

imag()
Return the imaginary part of this algebraic real.
It always returns 0.

EXAMPLES:

sage: a = AA(sqrt(2) + sqrt(3))
sage: a.imag()
0
sage: parent(a.imag())
Algebraic Real Field

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)
1.414213562373095?
sage: x.interval_exact(RIF)
1.414213562373095?
sage: y.interval(RIF)
2.000000000000000?
sage: y.interval_exact(RIF)
2
sage: z = 1 + AA(2).sqrt() / 2^200
sage: z.interval(RIF)
1.0000000000000001?
sage: z.interval_exact(RIF)
1.0000000000000001?

multiplicative_order()
Compute the multiplicative order of this real algebraic number.
That is, find the smallest positive integer \( n \) such that \( x^n = 1 \). If there is no such \( n \), returns \(+\infty\).

We first check that \( \text{abs}(x) \) is very close to 1. If so, we compute \( x \) exactly and compare it to 1 and -1.

EXAMPLES:

```
sage: AA(1).multiplicative_order()
sage: AA(-1).multiplicative_order()
sage: AA(5).sqrt().multiplicative_order()
```

\( +\infty \)

\[ \text{real}() \]

Return the real part of this algebraic real.

It always returns \( \text{self} \).

EXAMPLES:

```
sage: a = AA(sqrt(2) + sqrt(3))
sage: a.real()  # 3.146264369941973?
sage: a.real() is a
True
```

\[ \text{real\_exact}(\text{field}) \]

Given a \( \text{RealField} \), 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
```

\[ (-x).real\_exact(RR) \]

```
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
```

(continues on next page)
Given a RealField, compute a good approximation to self in that field. The approximation will be off by at most two ulp’s, except for numbers which are very close to 0, which will have an absolute error at most $2^{-(\text{field.prec()} - 1)}$. Also, the rounding mode of the field is respected.

**EXAMPLES:**

```python
sage: x = AA(2).sqrt()^2
sage: x.real_number(RR)
2.00000000000000
sage: x.real_number(RealField(53, rnd='RNDD'))
1.99999999999999
sage: x.real_number(RealField(53, rnd='RNDU'))
-2.00000000000000
sage: x.real_number(RealField(53, rnd='RNDZ'))
-1.99999999999999
sage: (-x).real_number(RR)
-2.00000000000000
sage: (-x).real_number(RealField(53, rnd='RNDD'))
-2.00000000000000
sage: (-x).real_number(RealField(53, rnd='RNDU'))
-1.99999999999999
sage: (-x).real_number(RealField(53, rnd='RNDZ'))
-1.99999999999999
sage: (x-2).real_number(RR)
5.42101086242752e-20
sage: (x-2).real_number(RealField(53, rnd='RNDD'))
-1.08420217248551e-19
sage: (x-2).real_number(RealField(53, rnd='RNDU'))
2.16840434497101e-19
sage: (x-2).real_number(RealField(53, rnd='RNDZ'))
0.00000000000000
```

The `real_number` method is used to approximate elements of an algebraic number field in a specified real field, with considerations for rounding modes and precision.
### round()

Round self to the nearest integer.

**EXAMPLES:**

```python
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:**

```python
sage: AA(-5).nth_root(7).sign()
-1
sage: (AA(2).sqrt() - AA(2).sqrt()).sign()
0
sage: a = AA(2).sqrt() + AA(3).sqrt() - 58114382797550084497/18470915334626475921
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()
```

(continues on next page)
trunc()
Round self to the nearest integer toward zero.

EXAMPLES:

```
sage: AA(sqrt(2)).trunc()
sage: AA(-sqrt(2)).trunc()
sage: AA(1).trunc()
sage: AA(-1).trunc()
```

class sage.rings.qqbar.AlgebraicRealField

Bases: sage.misc.fast_methods.Singleton, sage.rings.qqbar.AlgebraicField_common, sage.rings.abc.AlgebraicRealField

The field of algebraic reals.

algebraic_closure()
Return the algebraic closure of this field, which is the field \( \mathbb{Q} \) of algebraic numbers.

EXAMPLES:

```
sage: AA.algebraic_closure()
```

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
- \( \text{prec} \) – precision of approximate field to return
- \( \text{extras} \) – (optional) a dict of extra keyword arguments for the RealField constructor

EXAMPLES:

```
sage: AA.completion(infinity, 500)
sage: AA.completion(infinity, prec=53, extras={'type': 'RDF'})
sage: AA.completion(infinity, 53) is RR
```

(continues on next page)
True

```python
sage: AA.completion(7, 10)
Traceback (most recent call last):
...
NotImplementedError
```

**gen**

Return the \( n \)-th element of the tuple returned by `gens()`.

**EXAMPLES:**

```python
sage: AA.gen(0)
1
sage: AA.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, we opt for just returning 1.

**EXAMPLES:**

```python
sage: AA.gens()
(1,)
```

**ngens()**

Return the size of the tuple returned by `gens()`.

**EXAMPLES:**

```python
sage: AA.ngens()
1
```

**polynomial_root**

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:**

```python
sage: x = polygen(AA)
sage: phi = AA.polynomial_root(x^2 - x - 1, RIF(1, 2)); phi
```

(continues on next page)
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
```

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
```

**random_element** (*poly_degree=2, *args, **kws*)

Return a random algebraic real number.

**INPUT:**

- **poly_degree** - default: 2 - degree of the random polynomial over the integers of which the returned algebraic real number is a (real part of a) root. This is not necessarily the degree of the minimal polynomial of the number. Increase this parameter to achieve a greater diversity of algebraic numbers, at a cost of greater computation time. You can also vary the distribution of the coefficients but that will not vary the degree of the extension containing the element.

- **args, kws** - arguments and keywords passed to the random number generator for elements of ZZ, the integers. See `random_element()` for details, or see example below.

**OUTPUT:**

An element of AA, the field of algebraic real numbers (see `sage.rings.qqbar`).

**ALGORITHM:**

We pass all arguments to `AlgebraicField.random_element()`, and then take the real part of the result.

**EXAMPLES:**

```
sage: a = AA.random_element()
sage: a in AA
True
```

```
sage: b = AA.random_element(poly_degree=5)
sage: b in AA
True
```

Parameters for the distribution of the integer coefficients of the polynomials can be passed on to the random element method for integers. For example, we can rule out zero as a coefficient (and therefore as a root) by requesting coefficients between 1 and 10:

```
sage: c = AA.random_element(poly_degree=5, *args, **kws)
sage: c
```

**5.1. Field of Algebraic Numbers**

403
sage: z = [AA.random_element(x=1, y=10) for _ in range(5)]
sage: AA(0) in z
False

zeta($n=2$)
Return an $n$-th root of unity in this field. This will raise a ValueError if $n \neq \{1, 2\}$ since no such root exists.

INPUT:
- $n$ (integer) – default 2

EXAMPLES:

sage: AA.zeta(1)
1
sage: AA.zeta(2)
-1
sage: AA.zeta()
-1
sage: AA.zeta(3)
Traceback (most recent call last):
...
ValueError: no n-th root of unity in algebraic reals

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

sage.rings.qqbar.QQbar = 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:

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 ...>
The code tries to use existing unions of number fields:

```
sage: sqrt17 = QQbar(17).sqrt()
sage: sqrt19 = QQbar(19).sqrt()
sage: a = sqrt17 + sqrt19
sage: b = sqrt17 * sqrt19 - sqrt17 + sqrt19 * (sqrt17 + 2)
sage: b, type(b._descr)
(40.53909377268655?, <class 'sage.rings.qqbar.ANBinaryExpr'>)
sage: a.exactify()
sage: b = sqrt17 * sqrt19 - sqrt17 + sqrt19 * (sqrt17 + 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:**

```
sage: a = QQbar(sqrt(2)) + QQbar(sqrt(3))
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.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.rings.qqbar.an_binop_rational(a, b, op)`

Used to add, subtract, multiply or divide algebraic numbers.

Used when both are actually rational.

**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:

sage: from sage.rings.qqbar import cmp_elements_with_same_minpoly
sage: x = polygen(ZZ)
sage: p = x^2 - 2
sage: a = AA.polynomial_root(p, RIF(1,2))
sage: b = AA.polynomial_root(p, RIF(-2,-1))
sage: cmp_elements_with_same_minpoly(a, b, p)
1
sage: cmp_elements_with_same_minpoly(-a, b, p)
0

sage.rings.qqbar.conjugate_expand(v)
If the interval v (which may be real or complex) includes some purely real numbers, return v' containing v such that v' == v'.conjugate(). Otherwise return v unchanged. (Note that if v' == v'.conjugate(), and v' includes one non-real root of a real polynomial, then v' also includes the conjugate of that root. Also note that the diameter of the return value is at most twice the diameter of the input.)

EXAMPLES:

```
sage: from sage.rings.qqbar import conjugate_expand
sage: conjugate_expand(CIF(RIF(0, 1), RIF(1, 2))).str(style='brackets')
'[0.0000000000000000 .. 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:

```
sage: from sage.rings.qqbar import conjugate_shrink
sage: conjugate_shrink(RIF(3, 4)).str(style='brackets')
'[3.0000000000000000 .. 4.0000000000000000]'
sage: conjugate_shrink(CIF(RIF(1, 2), RIF(1, 2))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000] + [1.0000000000000000 .. 2.0000000000000000]*I'
sage: conjugate_shrink(CIF(RIF(1, 2), RIF(-2, 1))).str(style='brackets')
'[1.0000000000000000 .. 2.0000000000000000]
```

sage.rings.qqbar.do_polred(poly, threshold=32)
Find a polynomial of reasonably small discriminant that generates the same number field as poly, using the PARI polredbest function.

INPUT:

- poly - a monic irreducible polynomial with integer coefficients

5.1. Field of Algebraic Numbers
• **threshold** - an integer used to decide whether to run `polredbest`

**OUTPUT:**

A triple `(elt_fwd, elt_back, new_poly)`, where:

• `new_poly` is the new polynomial defining the same number field,

• `elt_fwd` is a polynomial expression for a root of the new polynomial in terms of a root of the original polynomial,

• `elt_back` is a polynomial expression for a root of the original polynomial in terms of a root of the new polynomial.

**EXAMPLES:**

```python
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)
```

```python
sage: newpol
x^2 - x - 1
```

```python
sage: Kold.<a> = NumberField(oldpol)
```

```python
sage: Knew.<b> = NumberField(newpol)
```  
```python
sage: newpol(fwd(a))
0
```

```python
sage: oldpol(back(b))
0
```

```python
sage: do_polred(x^2 - x - 11)
(1/3*x + 1/3, 3*x - 1, x^2 - x - 11)
```

```python
sage: do_polred(x^3 + 123456)
(-1/4*x, -4*x, x^3 - 1929)
```

This shows that trac ticket #13054 has been fixed:

```python
sage: do_polred(x^4 - 4294967296*x^2 + 54265257667816538374400)
(1/4*x, 4*x, x^4 - 268435456*x^2 + 211973662764908353025)
```

`sage.rings.qqbar.find_zero_result(fn, l)`

1 is a list of some sort. `fn` is a function which maps an element of 1 and a precision into an interval (either real or complex) of that precision, such that for sufficient precision, exactly one element of 1 results in an interval containing 0. Returns that one element of 1.

**EXAMPLES:**

```python
sage: from sage.rings.qqbar import find_zero_result
sage: _.<x> = QQ['x']
```

```python
sage: delta = 10^(-70)
```

```python
sage: p1 = x - 1
```

```python
sage: p2 = x - 1 - delta
```

```python
sage: p3 = x - 1 + delta
```

```python
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()`.

408 Chapter 5. Algebraic Numbers
EXAMPLES:

```python
sage: AA(golden_ratio) # indirect doctest
1.618033988749895?
```

`sage.rings.qqbar.is_AlgebraicField(F)`
Check whether `F` is an `AlgebraicField` instance.

This function is deprecated. Use `isinstance()` with `AlgebraicField` instead.

EXAMPLES:

```python
sage: from sage.rings.qqbar import is_AlgebraicField
sage: [is_AlgebraicField(x) for x in [AA, QQbar, None, 0, "spam"]]
doctest:warning...
DeprecationWarning: is_AlgebraicField is deprecated;
use isinstance(..., sage.rings.abc.AlgebraicField) instead
See https://trac.sagemath.org/32660 for details.
[False, True, False, False, False]
```

`sage.rings.qqbar.is_AlgebraicField_common(F)`
Check whether `F` is an `AlgebraicField_common` instance.

This function is deprecated. Use `isinstance()` with `AlgebraicField_common` instead.

EXAMPLES:

```python
sage: from sage.rings.qqbar import is_AlgebraicField_common
sage: [is_AlgebraicField_common(x) for x in [AA, QQbar, None, 0, "spam"]]
doctest:warning...
DeprecationWarning: is_AlgebraicField_common is deprecated;
use isinstance(..., sage.rings.abc.AlgebraicField_common) instead
See https://trac.sagemath.org/32610 for details.
[True, True, False, False, False]
```

`sage.rings.qqbar.is_AlgebraicNumber(x)`
Test if `x` is an instance of `AlgebraicNumber`. For internal use.

EXAMPLES:

```python
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:

```python
sage: from sage.rings.qqbar import is_AlgebraicReal
sage: is_AlgebraicReal(AA(sqrt(2)))
True
sage: is_AlgebraicReal(QQbar(sqrt(2)))
```

(continues on next page)

5.1. Field of Algebraic Numbers
Algebraic Numbers and Number Fields, Release 9.6

(continued from previous page)

False
sage: is_AlgebraicReal("spam")
False
sage.rings.qqbar.is_AlgebraicRealField(F)
Check whether F is an AlgebraicRealField instance. For internal use.
This function is deprecated. Use isinstance() with AlgebraicRealField instead.
EXAMPLES:
sage: from sage.rings.qqbar import is_AlgebraicRealField
sage: [is_AlgebraicRealField(x) for x in [AA, QQbar, None, 0, "spam"]]
doctest:warning...
DeprecationWarning: is_AlgebraicRealField is deprecated;
use isinstance(..., sage.rings.abc.AlgebraicRealField instead
See https://trac.sagemath.org/32660 for details.
[True, False, False, False, False]
sage.rings.qqbar.isolating_interval(intv_fn, pol)
intv_fn is a function that takes a precision and returns an interval of that precision containing some particular
root of pol. (It must return better approximations as the precision increases.) pol is an irreducible polynomial
with rational coefficients.
Returns an interval containing at most one root of pol.
EXAMPLES:
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.
˓→0000000000000000000000000000000000000000000000000000000000000000000001000000000000000000000000000
˓→

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,
embedded=False, prec=53)
Given a sequence of elements of either AA or QQbar (or a mixture), computes a number field containing all of
these elements, these elements as members of that number field, and a homomorphism from the number field
back to AA or QQbar.
INPUT:
410

Chapter 5. Algebraic Numbers


• **numbers** – a number or list of numbers.

• **minimal** – Boolean (default: False). Whether to minimize the degree of the extension.

• **same_field** – Boolean (default: False). See below.

• **embedded** – Boolean (default: False). Whether to make the NumberField embedded.

• **prec** – integer (default: 53). The number of bit of precision to guarantee finding real roots.

**OUTPUT:**

A tuple with the NumberField, the numbers inside the NumberField, and a homomorphism from the number field back to `AA` or `QQbar`.

This may not return the smallest such number field, unless `minimal=True` is specified.

If `same_field=True` is specified, and all of the elements are from the same field (either `AA` or `QQbar`), the generated homomorphism will map back to that field. Otherwise, if all specified elements are real, the homomorphism might map back to `AA` (and will, if `minimal=True` is specified), even if the elements were in `QQbar`.

Also, a single number can be passed, rather than a sequence; and any values which are not elements of `AA` or `QQbar` will automatically be coerced to `QQbar`.

This function may be useful for efficiency reasons: doing exact computations in the corresponding number field will be faster than doing exact computations directly in `AA` or `QQbar`.

**EXAMPLES:**

We can use this to compute the splitting field of a polynomial. (Unfortunately this takes an unreasonably long time for non-toy examples.):

```sage
sage: x = polygen(QQ)
sage: p = x^3 + x^2 + x + 17
sage: rts = p.roots(ring=QQbar, multiplicities=False)
sage: splitting = number_field_elements_from_algebraics(rts)[0]; splitting
Number Field in a with defining polynomial y^6 - 40*y^4 - 22*y^3 + 873*y^2 + 1386*y + 594
sage: p.roots(ring=splitting)
[(361/29286*a^5 - 19/3254*a^4 - 14359/29286*a^3 + 401/29286*a^2 + 18183/1627*a +
  15939/1627, 1), (49/117144*a^5 - 179/39048*a^4 - 3247/117144*a^3 + 22553/117144*a^2 +
  1744/4881*a - 17195/6508, 1), (-1493/117144*a^5 + 407/39048*a^4 + 60683/117144*a^3 -
  24157/117144*a^2 - 56293/4881*a - 53033/6508, 1)]
sage: rt2 = AA(sqrt(2)); rt2
1.414213562373095?
sage: rt3 = AA(sqrt(3)); rt3
1.732050807568878?
sage: rt3a = QQbar(sqrt(3)); rt3a
1.732050807568878?
sage: qqI = QQbar.zeta(4); qqI
I
sage: z3 = QQbar.zeta(3); z3
-0.500000000000000? + 0.866025403784439?*I
sage: rt2b = rt3 + rt2 - rt3; rt2b
1.414213562373095?
sage: rt2c = z3 + rt2 - rt3; rt2c
1.414213562373095? + 0.?e-19*I
sage: number_field_elements_from_algebraics(rt2)
```

(continues on next page)
(continued from previous page)

```plaintext
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 + 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?)
```

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:

```plaintext
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?)
```

```plaintext
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 Field
   Defn: a |--> 1.732050807568878?)
```

We've created `rt2b` in such a way that sage does not initially know that it's in a degree-2 extension of Q:

```plaintext
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:

```plaintext
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:

```plaintext
sage: number_field_elements_from_algebraics((QQbar(1/2), AA(17)))
(Rational Field, [1/2, 17], Ring morphism:
   From: Rational Field
   To:   Algebraic Real Field
   Defn: 1 |--> 1)
```

Or we can just pass in symbolic expressions, as long as they can be coerced into QQbar:

```plaintext
sage: number_field_elements_from_algebraics((QQbar(r), AA(r)) for r in [rt2, rt3])
```
Here we see an example of doing some computations with number field elements, and then mapping them back into QQbar:

```python
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 - \sqrt{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: nfI^2
-1
sage: sum = 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
```

It is also possible to have an embedded Number Field:

```python
sage: x = polygen(ZZ)
sage: my_num = AA.polynomial_root(x^3-2, RIF(0,3))
sage: res = number_field_elements_from_algebraics(my_num, embedded=True)
sage: res[0].gen_embedding()
1.259921049894873?
Ring morphism:
  From: Number Field in a with defining polynomial y^3 - 2 with a = 1.
  To:   Algebraic Real Field
  Defn: a |---> 1.259921049894873?
```

(continues on next page)
Complex embeddings are possible as well:

```
sage: elems = [sqrt(5), 2^(1/3)+sqrt(3)*I, 3/4]
sage: nf, nums, hom = number_field_elements_from_algebraics(elems, embedded=True)
sage: nf
Number Field in a with defining polynomial y^24 - 6*y^23 + 12*y^22 - 24*y^21 + ....
˓→a = 0.2598678911433438? + 0.0572892247058457?*I
sage: list(map(QQbar, nums)) == elems == list(map(hom, nums))
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)

Check whether the rational \( r \) is an exact \( d \)'th power.

If so, this returns the \( d \)'th root of \( r \); otherwise, this returns \( \text{None} \).

**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 \( \text{None} \).

**EXAMPLES:**

```
sage: from sage.rings.qqbar import short_prec_seq
sage: short_prec_seq()
(64, 128, None)
```
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:
- [Bre1997]

EXAMPLES:

```
sage: from sage.rings.qqbar import tail_prec_seq
tsage: g = tail_prec_seq()
tsage: [next(g), next(g), next(g)]
[256, 512, 1024]
```

To generate cyclotomic elements:

```
sage: UCF = UniversalCyclotomicField(); UCF
Universal Cyclotomic Field

sage: UCF.gen(5)
E(5)
sage: UCF.gen(5,2)
E(5)^2

sage: E = UCF.gen
```

Equality and inequality checks:

```
sage: E(6,2) == E(6)^2 == E(3)
True
```

(continues on next page)
Addition and multiplication:

```sage
sage: E(6)^2 != E(3)
False
```

```sage
sage: f = E(2) + E(3); f
2*E(3) + E(3)^2
```

Inverses:

```sage
sage: f^-1
1/3*E(3) + 2/3*E(3)^2
```

Conjugation and Galois conjugates:

```sage
sage: f.conjugate()
E(3) + 2*E(3)^2
```

```sage
sage: f.galois_conjugates()
[2*E(3) + E(3)^2, E(3) + 2*E(3)^2]
```

One can create matrices and polynomials:

```sage
sage: m = matrix(2,[E(3),1,1,E(4)]); m
[ E(3)  1]
[ 1  E(4)]
```

```sage
sage: m.charpoly()
x^2 + (-E(12)^4 + E(12)^7 + E(12)^11)*x + E(12)^4 + E(12)^7 + E(12)^8
```

```
```
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)
- Sebastian Oehms (2019): add _factor_univariate_polynomial() (see trac ticket #28631)

```
sage: E(3) * x - 1
E(3)*x - 1
```

sage.rings.universal_cyclotomic_field.E(n, k=1)

Return the n-th root of unity as an element of the universal cyclotomic field.

EXAMPLES:

```
sage: E(3)
E(3)
```

```
sage: E(3) + E(5)
-E(15)^2 - 2*E(15)^8 - E(15)^11 - E(15)^13 - E(15)^14
```

sage.rings.universal_cyclotomic_field.UCF_sqrt_int(N, UCF)

Return the square root of the integer N.

EXAMPLES:

```
sage: from sage.rings.universal_cyclotomic_field import UCF_sqrt_int
sage: UCF = UniversalCyclotomicField()
sage: UCF_sqrt_int(0, UCF)
0
sage: UCF_sqrt_int(1, UCF)
1
sage: UCF_sqrt_int(-1, UCF)
E(4)
sage: UCF_sqrt_int(2, UCF)
E(8) - E(8)^3
```

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 + 0.193096429713793*I
```

```
sage: complex(E(7)+E(7,2))
```

(continues on next page)
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
```

**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:
There is an alias \texttt{zeta} also available:

\begin{verbatim}
sage: UCF.zeta(6)
-E(3)^2
\end{verbatim}

\textbf{is\_exact()}  
Return True as this is an exact ring (i.e. not numerical).

\textbf{EXAMPLES:}
\begin{verbatim}
sage: UniversalCyclotomicField().is\_exact()
True
\end{verbatim}

\textbf{one()}  
Return one.

\textbf{EXAMPLES:}
\begin{verbatim}
sage: UCF = UniversalCyclotomicField()
sage: UCF.one()
1
sage: parent(_)
Universal Cyclotomic Field
\end{verbatim}

\textbf{some\_elements()}  
Return a tuple of some elements in the universal cyclotomic field.

\textbf{EXAMPLES:}
\begin{verbatim}
sage: UniversalCyclotomicField().some\_elements()
(0, 1, -1, E(3), E(7) - 2/3*E(7)^2)
sage: all(parent(x) is UniversalCyclotomicField() for x in _)
True
\end{verbatim}

\textbf{zero()}  
Return zero.

\textbf{EXAMPLES:}
\begin{verbatim}
sage: UCF = UniversalCyclotomicField()
sage: UCF.zero()
0
sage: parent(_)
Universal Cyclotomic Field
\end{verbatim}

\textbf{zeta}(n, k=1)  
Return the standard primitive \(n\)-th root of unity.  
If \(k\) is not None, return the \(k\)-th power of it.
EXAMPLES:

```python
sage: UCF = UniversalCyclotomicField()
sage: UCF.gen(15)
E(15)
sage: UCF.gen(7,3)
E(7)^3
sage: UCF.gen(4,2)
-1
```

There is an alias `zeta` also available:

```python
sage: UCF.zeta(6)
-E(3)^2
```

class `sage.rings.universal_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:**

```python
sage: f = 5/2*E(3)+E(5)/7
sage: f.abs()
2.597760303873084?
sage: abs(f)
2.597760303873084?
sage: a = E(8)
sage: abs(a)
1
sage: v, w = vector([a]), vector([a, a])
sage: v.norm(), w.norm()
(1, 1.414213562373095?)
sage: v.norm().parent()
Algebraic Real Field
```

**additive_order()**

Return the additive order.

**EXAMPLES:**

```python
sage: UCF = UniversalCyclotomicField()
sage: UCF.zero().additive_order()
0
sage: UCF.one().additive_order()
+Infinity
sage: UCF.gen(3).additive_order()
+Infinity
```
**conductor()**

Return the conductor of \texttt{self}.

**EXAMPLES:**

```python
sage: E(3).conductor()
3
sage: (E(5) + E(3)).conductor()
15
```

**conjugate()**

Return the complex conjugate.

**EXAMPLES:**

```python
sage: (E(7) + 3*E(7,2) - 5 * E(7,3)).conjugate()
-5*E(7)^4 + 3*E(7)^5 + E(7)^6
```

**denominator()**

Return the denominator of this element.

See also:

- \texttt{is_integral()}

**EXAMPLES:**

```python
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**

Return the Galois conjugates of \texttt{self}.

**INPUT:**

- \texttt{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,]
```

(continues on next page)

---

5.2. Universal cyclotomic field 421
\[ -E(9)^5 + E(9)^7 \]

```python
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
```

```python
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
```

```python
sage: a = E(5) - 2*E(3)
sage: AA(a.imag()) == QQbar(a).imag()
True
```

**inverse()**

**is_integral()**

Return whether `self` is an algebraic integer.

This just wraps `IsIntegralCyclotomic` from GAP.

**See also:**

`denominator()`

**EXAMPLES:**
is_integral()
Test whether this element is an integral element.

is_rational()
Test whether this element is a rational number.

is_real()
Test whether this element is real.

is_square()
EXAMPLES:

minpoly(var='x')
The minimal polynomial of self element over Q.

EXAMPLES:

5.2. Universal cyclotomic field 423
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
sage: parent(_)
Integer Ring
```

**real()**
Return the real part of this element.

**EXAMPLES:**

```python
sage: E(3).real()
-1/2
sage: E(5).real()
```

(continues on next page)
\[ \frac{1}{2}E(5) + \frac{1}{2}E(5)^4 \]

sage: a = E(5) - 2*E(3)
sage: AA(a.real()) == QQbar(a).real()
True

real_part()
Return the real part of this element.

EXAMPLES:

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(extend=True, all=False)
Return a square root of self.

With default options, the output is an element of the universal cyclotomic field when this element is ex-
pressed via a single root of unity (including rational numbers). Otherwise, return an algebraic number.

INPUT:

- **extend** – bool (default: True); if True, might return a square root in the algebraic closure of the
  rationals. If false, return a square root in the universal cyclotomic field or raises an error.
- **all** – bool (default: False); if True, return a list of all square roots.

EXAMPLES:

sage: UCF = UniversalCyclotomicField()
sage: UCF(3).sqrt()
E(12)^7 - E(12)^11
sage: (UCF(3).sqrt())**2
3
sage: r = UCF(-1400 / 143).sqrt()
sage: r**2
-1400/143
sage: E(33).sqrt()
-E(33)^17
sage: E(33).sqrt() ** 2
E(33)
sage: (3 * E(5)).sqrt()
-E(60)^11 + E(60)^31
sage: (3 * E(5)).sqrt() ** 2
3*E(5)

Setting all=True you obtain the two square roots in a list:
In the following situation, Sage is not (yet) able to compute a square root within the universal cyclotomic field:

```
sage: (E(5) + E(5, 2)).sqrt()
0.7476743906106103? + 1.029085513635746?*I
sage: (E(5) + E(5, 2)).sqrt(extend=False)
Traceback (most recent call last):
  ...  
NotImplementedError: sqrt() not fully implemented for elements of UniversalCyclotomicField
```

```
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
[zeta60^10 - 1  zeta60^15]
[  zeta60^12  zeta60^10]
```

Using a non-standard embedding:
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:

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:

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
INDICES AND TABLES

• Index
• Module Index
• Search Page
sage.rings.number_field.bdd_height, 203
sage.rings.number_field.class_group, 291
sage.rings.number_field.galois_group, 194
sage.rings.number_field.homset, 210
sage.rings.number_field.maps, 218
sage.rings.number_field.morphism, 209
sage.rings.number_field.number_field, 1
sage.rings.number_field.number_field_base, 110
sage.rings.number_field.number_field_element, 136
sage.rings.number_field.number_field_element_quadratic, 175
sage.rings.number_field.number_field_ideal, 251
sage.rings.number_field.number_field_ideal_rel, 281
sage.rings.number_field.number_field_morphisms, 214
sage.rings.number_field.number_field_rel, 113
sage.rings.number_field.order, 227
sage.rings.number_field.S_unit_solver, 304
sage.rings.number_field.selmer_group, 333
sage.rings.number_field.small_primes_of_degree_one, 331
sage.rings.number_field.splitting_field, 188
sage.rings.number_field.structure, 224
sage.rings.number_field.totallyreal, 339
sage.rings.number_field.totallyreal_data, 346
sage.rings.number_field.totallyreal_phc, 349
sage.rings.number_field.totallyreal_rel, 342
sage.rings.number_field.unit_group, 297
sage.rings.qqbar, 351
sage.rings.universal_cyclotomic_field, 415
Symbols

_richcmp_() (sage.rings.qqbar.AlgebraicNumber method), 382
_richcmp_() (sage.rings.qqbar.AlgebraicReal method), 396

A

 AA (in module sage.rings.qqbar), 360
 abs() (sage.rings.number_field.number_field_element.NumberFieldElement method), 136
 abs() (sage.rings.qqbar.ANDescr method), 361
 abs() (sage.rings.qqbar.ANExtensionElement method), 363
 abs() (sage.rings.qqbar.ANRational method), 368
 abs() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicFieldElement method),
 abs_hom() (sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism_from_abs method), 209
 abs_non_arch() (sage.rings.number_field.number_field_element.NumberFieldElement method), 138
 abs_val() (sage.rings.number_field.number_field_element.NumberFieldAbsolute method), 11
 absolute_base_field() (sage.rings.number_field.number_field_element.NumberFieldRelative method), 115
 absolute_charpoly() (sage.rings.number_field.number_field_element.NumberFieldElement method), 167
 absolute_charpoly() (sage.rings.number_field.number_field_element.NumberFieldElement method), 170
 absolute_charpoly() (sage.rings.number_field.number_field_element.OrderElement method), 173
 absolute_degree() (sage.rings.number_field.number_field_element.NumberFieldAbsolute method), 12
 absolute_degree() (sage.rings.number_field.number_field_element.NumberFieldGeneric method), 42
 absolute_degree() (sage.rings.number_field.number_field_element.NumberFieldRelative method), 115
 absolute_degree() (sage.rings.number_field.order.Order method), 230

absolute_diff

absolute_differen

absolute_differen

absolute_differen

absolute_differen

absolute_differen

absolute_discrim

absolute_discrim

absolute_discrim

absolute_discrim

absolute_discrim

absolute_field

absolute_generator

absolute_generator

absolute_generator

absolute_generator

absolute_ideal

absolute_minpoly

absolute_minpoly

absolute_minpoly

absolute_minpoly

absolute_norm

absolute_norm
Algebraic Numbers and Number Fields, Release 9.6

B

bach_bound() (sage.rings.number_field.number_field_base.NumberField_base_method), 110

base_field() (sage.rings.number_field.number_field.NumberField_method), 14

basis() (sage.rings.number_field.number_field.NumberField_method), 119

basis() (sage.rings.number_field.number_field_rel.NumberField_relative_method), 119

basis() (sage.rings.number_field.number_field_base.NumberField_base_method), 267

basis() (sage.rings.number_field.order.Order method), 230

basis() (sage.rings.number_field.order.Order_absolute_method), 242

basis() (sage.rings.number_field.order.Order_relative_method), 246

basis_for_p_cokernel() (in module sage.rings.number_field.selmer_group), 333

basis_to_module() (in module sage.rings.number_field.number_field_ideal), 279

bdd_height() (in module sage.rings.number_field.bdd_height), 203

bdd_height() (in module sage.rings.number_field.bdd_height), 204

bdd_height_iq() (in module sage.rings.number_field.bdd_height), 206

bdd_norm_pr_gens_iq() (in module sage.rings.number_field.bdd_height), 206

bdd_norm_pr_ideal_gens() (in module sage.rings.number_field.bdd_height), 206

beta_k() (in module sage.rings.number_field.S_unit_solver), 308

change_names() (sage.rings.number_field.number_field_field_NumberField_generic_method), 120

change_names() (sage.rings.number_field.order.Order_absolute_method), 140

change_names() (sage.rings.number_field.order.Order_method), 168

change_names() (sage.rings.number_field.number_field_element_NumberField_element_method), 171

change_names() (sage.rings.number_field.number_field_element_quadratic_NumberField_element_quadratic_method), 174

change_names() (sage.rings.number_field.number_field_element_quadratic_NumberField_element_quadratic_method), 177

change_names() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic_method), 180

ClassGroup (class in sage.rings.number_field.class_group), 292

clean_rfv_dict() (in module sage.rings.number_field.S_unit_solver), 311

clean_sfs() (in module sage.rings.number_field.S_unit_solver), 312

clear_denominators() (in module sage.rings.qqbar), 406

closest() (in module sage.rings.number_field.number_field_morphisms), 216

cmp_elements_with_same_minpoly() (in module sage.rings.qqbar), 406

coefficients_to_power_sums() (in module sage.rings.number_field.totallyreal_phc), 349

column_log() (in module sage.rings.number_field.S_unit_solver), 312

column_Log() (in module sage.rings.number_field.S_unit_solver), 319

coefficients_to_power_sums() (in module sage.rings.number_field.order.Order_element_quadratic), 378

compatible_system_lift() (in module
(sage.rings.number_field.order.AbsoluteOrderFactory, 228)
defining_polynomial()
  (sage.rings.number_field.number_field_number_field_NumberField_generic method), 52
create_key_and_extra_args()
  (sage.rings.number_field.order.RelativeOrderFactory method), 247
create_object() (sage.rings.number_field_number_field_NumberField method), 4
create_object() (sage.rings.number_field_number_field_NumberField method), 9
create_object() (sage.rings.number_field.order.AbsoluteOrderFactory method), 110
create_object() (sage.rings.number_field.order.RelativeOrderFactory method), 122
create_structure() (sage.rings.number_field.structure.AbsoluteFromRelative method), 9
cx_LLL_bound() (in module sage.rings.number_field.S_unit_solver), 318
create_object() (sage.rings.number_field.number_field_NumberField method), 318
create_object() (sage.rings.number_field.number_field_NumberField method), 320
create_object() (sage.rings.number_field.number_field_NumberField method), 224
create_object() (sage.rings.number_field.number_field_NumberField method), 225
cyclotomic_field_convertion (class in module sage.rings.number_field.number_field_morphisms), 214
CyclotomicFieldEmbedding (class in module sage.rings.number_field.number_field_morphisms), 214
CyclotomicFieldFactory (class in module sage.rings.number_field.number_field), 2
CyclotomicFieldHomomorphism_im_gens (class in module sage.rings.number_field.number_field), 209
CyclotomicFieldHomset (class in module sage.rings.number_field_homset), 210
decomposition_group() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 199
decomposition_group() (sage.rings.number_field.number_field_ideal.NumberFieldElement_ideal method), 269
decomposition_type() (sage.rings.number_field_number_field_NumberField method), 50
default_base_hom() (sage.rings.number_field.homset.RelativeNumberFieldHomset method), 213
default_interval_prec() (sage.rings.qqbar.AlgebraicField_common method), 378
defining_polynomial()
field_element_value()
  (sage.rings.qqbar.ANExtensionElement method), 364

find_zero_result() (in module sage.rings.qqbar), 408

fixed_field() (sage.rings.number_field.galois_group.GaloisGroup_v1 method), 195

floor() (sage.rings.number_field.number_field_element.NumberFieldElement method), 145

gcd() (sage.rings.number_field.number_field_element.NumberFieldElement method), 146

gen() (sage.rings.number_field_number_field_NumberField_generic method), 269

fractional_ideal() (sage.rings.number_field.number_field_relative.NumberFieldRelative method), 56

fractional_ideal() (sage.rings.number_field_number_field.Order method), 229

FractionalIdealClass (class in sage.rings.number_field_class_group), 293

gens() (sage.rings.number_field_class_group.FractionalIdealClass method), 293

free_module() (sage.rings.number_field_number_field.Absolute method), 18

free_module() (sage.rings.number_field_number_field_ideal.NumberFieldIdeal method), 269

free_module() (sage.rings.number_field_number_field_ideal_re.NumberFieldIdeal_re method), 284

free_module() (sage.rings.number_field_number_field_re.NumberFieldRelative method), 123

free_module() (sage.rings.number_field_number_field.Order method), 233

fundamental_units()
  (sage.rings.number_field_unit_group.UnitGroup method), 301

gcd()

GaloisGroup_v1 (class in sage.rings.number_field_galois_group), 196

GaloisGroup_v2 (class in sage.rings.number_field_galois_group), 198

GaloisGroupElement (class in sage.rings.number_field_galois_group), 194

GaussianField() (in module sage.rings.number_field.number_field), 4

GaussianIntegers() (in module sage.rings.number_field_element_quadratic), 229

gen() (sage.rings.number_field_number_field_element.NumberFieldElement method), 146

gen() (sage.rings.number_field_number_field_NumberField_generic method), 58

gen() (sage.rings.qqbar.AlgebraicField method), 375

gen() (sage.rings.qqbar.AlgebraicRealField method), 375

gen() (sage.rings.qqbar.AlgebraicRealField method), 269

gen() (sage.rings.qqbar.AlgebraicRealField method), 402

gens() (sage.rings.number_field_class_group.FractionalIdealClass method), 293

gens() (sage.rings.number_field_number_field_NumberField_relative method), 124

gens() (sage.rings.number_field_number_field_AlgebraicField method), 375

gens() (sage.rings.qqbar.AlgebraicRealField method), 402

gens() (sage.rings.qqbar.AlgebraicRealField method), 269

gens() (sage.rings.qqbar.AlgebraicRealField method), 375

gens() (sage.rings.qqbar.AlgebraicRealField method), 402

gens() (sage.rings.number_field_number_field_ideal.NumberFieldIdeal method), 145

gens() (sage.rings.number_field_number_field_ideal.NumberFieldIdeal method), 57

GaloisGroup (in module sage.rings.number_field_galois_group), 194

GaloisGroup_subgroup (class in sage.rings.number_field_galois_group), 195

global height() (sage.rings.number_field_number_field_element_NumberFieldElement method), 147
global_height_arch()
(sage.rings.number_field.number_field_element.NumberFieldElement.
method), 148

global_height_non_arch()
(sage.rings.number_field.number_field_element.NumberFieldElement.
method), 148

handle_sage_input()
(sage.rings.qqbar.ANBinaryExpr method), 360

handle_sage_input()
(sage.rings.qqbar.ANExtensionElement method), 364

handle_sage_input()
(sage.rings.qqbar.ANRational method), 368

handle_sage_input()
(sage.rings.qqbar.ANUnaryExpr method), 373

hermite_constant()
(in module
sage.rings.number_field.totallyreal_data), 346

hilbert_class_field()
(sage.rings.number_field.number_field_element.NumberFieldElement.
method), 101

hilbert_class_field_defining_polynomial()
(sage.rings.number_field.number_field_element.NumberField_element.
method), 101

hilbert_class_polynomial()
(sage.rings.number_field.number_field_element.NumberField_element.
method), 102

hilbert_conductor()
(sage.rings.number_field.number_field_element.NumberField_element.
method), 19

hilbert_symbol()
(sage.rings.number_field.number_field_element.NumberField_element.
method), 19

hilbert_symbol_negative_at_S()
(sage.rings.number_field.number_field_element.NumberField_element.
method), 23

ideal()
(sage.rings.number_field.order.Order methods), 234

ideal_below()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
method), 284

ideal_class_log()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
method), 271

idealchinese()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
method), 255

ideals_of_bdd_norm()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
method), 60

idealstar()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
method), 256

im_gens()
(sage.rings.number_field.morphism.RelativeNumberFieldHomomorphism.
method), 201

imag()
(sage.rings.number_field.number_field_element_number_field_element_quadratic.
method), 179

imag()
(sage.rings.number_field.number_field_element_number_field_element_quadratic.
method), 179

imag()
(sage.rings.qqbar.AlgebraicNumber method), 384

imag()
(sage.rings.qqbar.AlgebraicReal method), 397

imag()
(sage.rings.qqbar.ANUnaryExpr method), 397

imag()
(sage.rings.qqbar.ANUnaryExpr method), 397

imag()
(sage.rings.universal_cyclotomic_field.UniversalCyclotomicField.
method), 422

imag_part()
(sage.rings.number_field.number_field_element_number_field_element_quadratic.
method), 176

imag_part()
(sage.rings.universal_cyclotomic_field.UniversalCyclotomicField.
method), 422

increment()
(sage.rings.number_field.number_field_element_number_field_element_tr
_data
rel
method), 349

index_in()
(sage.rings.number_field.order.Order absolute
method), 243

index_in()
(sage.rings.number_field.order.Order relative
method), 246

inertia_group()
(sage.rings.number_field.galois_group.GaloisGroup_v2.
method), 201

inertia_group()
(sage.rings.number_field.galois_group.GaloisGroup_v2.
method), 201

integer_points_in_polytope()
(in module
sage.rings.number_field.totallyreal_data), 347

integral_closure()
(sage.rings.number_field.order.Order methods), 234

integral_basis()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
method), 286

integral_basis()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
method), 286

integral_basis()
(sage.rings.number_field.number_field_element_number_field_element
rel_ideal
rel_ideal
method), 272

index_in()
(sage.rings.number_field.order.Order absolute
method), 243

index_in()
(sage.rings.number_field.order.Order relative
method), 246

inertia_group()
(sage.rings.number_field.galois_group.GaloisGroup_v2.
method), 201

inertia_group()
(sage.rings.number_field.galois_group.GaloisGroup_v2.
method), 201

int_has_small_square_divisor()
(in module
sage.rings.number_field.totallyreal_data), 347

integer_points_in_polytope()
(in module
sage.rings.number_field.totallyreal_data), 347

integral_closure()
(sage.rings.number_field.order.Order methods), 234
integral_elements_in_box() (in module \texttt{sage.rings.number_field.totallyreal}), 345
\texttt{sage.rings.number_field.totallyreal_rel})

integral_split() (\texttt{sage.rings.number_field.number_field_ideal.NumberFieldIdeal}), 272
\texttt{sage.rings.number_field.number_field_ideal.NumberFieldIdeal_rel})

intersection() (\texttt{sage.rings.number_field.number_field_ideal.NumberFieldIdeal}), 272

is_abelian() (\texttt{sage.rings.number_field.number_field_ideal.NumberField_abelian}), 35
\texttt{sage.rings.number_field.number_field_ideal.NumberField_abelian_rel})

is_absolute() (\texttt{sage.rings.number_field.number_field_field.Field}), 124
\texttt{sage.rings.number_field.number_field_field.Field_relative})

is_Abelian() (\texttt{sage.rings.number_field.number_field_ideal.NumberFieldAbelian}), 104
\texttt{sage.rings.number_field.number_field_ideal.NumberFieldAbelian_rel})

is_AlgebraicField() (\texttt{sage.rings.number_field.number_field.Field}), 61
\texttt{sage.rings.number_field.number_field.Field_relative})

is_CM() (\texttt{sage.rings.number_field.number_field.NumberField_relative}), 61
\texttt{sage.rings.number_field.number_field.NumberField_relative})

is_CM_extension() (\texttt{sage.rings.number_field.number_field.NumberField_relative}), 124
\texttt{sage.rings.number_field.number_field.NumberField_relative})

is_complex() (\texttt{sage.rings.number_field.number_field.AlgebraicGenerator}), 390
\texttt{sage.rings.number_field.number_field.AlgebraicGenerator_relative})

is_CM() (\texttt{sage.rings.number_field.number_field.CyclotomicField}), 201
\texttt{sage.rings.number_field.number_field.CyclotomicField_relative})

is_CyclotomicField() (\texttt{sage.rings.number_field.number_field.NumberField_cyclotomic}), 409
\texttt{sage.rings.number_field.number_field.NumberField_quadratic})

is_exact() (\texttt{sage.rings.number_field.number_field_ideal.AlgebraicIdeal}), 362
\texttt{sage.rings.number_field.number_field_ideal.AlgebraicIdeal_relative})

is_field() (\texttt{sage.rings.number_field.number_field.Field}), 63
\texttt{sage.rings.number_field.number_field.Field_relative})

is_free() (\texttt{sage.rings.number_field.number_field.Field_relative}), 201
\texttt{sage.rings.number_field.number_field.Field_relative})

is_fundamental_discriminant() (\texttt{sage.rings.number_field.number_field_Discriminant}), 201
\texttt{sage.rings.number_field.number_field_Discriminant_relative})

is_global_field() (\texttt{sage.rings.number_field.number_field.GlobalField}), 201
\texttt{sage.rings.number_field.number_field.GlobalField_relative})

is_relative_field() (\texttt{sage.rings.number_field.number_field.RealField}), 201
\texttt{sage.rings.number_field.number_field.RealField_relative})

is_root() (\texttt{sage.rings.number_field.number_field.algebraic.Algebraic}), 201
\texttt{sage.rings.number_field.number_field.algebraic.Algebraic_relative})
is_galois() (sage.rings.number_field.number_field_rel.NumberField_rel.NumberField_element), 125

is_galois_absolute() is_NumberFieldFractionalIdeal() (in module sage.rings.number_field.number_field_rel.NumberField_rel.NumberField_element_number_field_ideal_rel), 125

is_galois_relative() is_NumberFieldFractionalIdeal_rel() (in module sage.rings.number_field.number_field_rel.NumberField_rel.NumberField_element_number_field_ideal_rel), 126

is_injective() (sage.rings.number_field.maps.NumberField_homset()) (in module sage.rings.number_field.number_field), 223

is_integer() (sage.rings.number_field.number_field_element.NumberFieldElement_quadratic is_isomorphic_relative().NumberFieldElement_quadratic), 150

is_integer() (sage.rings.number_field.number_field_element.NumberFieldElement_quadratic is:NumberFieldOrder() (in module sage.rings.number_field.order), 250

is_integral() (sage.rings.number_field.number_field_element.NumberFieldElement is_one().NumberFieldElement), 150

is_integral() (sage.rings.number_field.number_field_element.NumberFieldElement is_principal().NumberFieldElement_quadratic), 180

is_integral() (sage.rings.number_field.number_field_element.NumberFieldElement_quadric is_principal().NumberField_element), 273

is_integral() (sage.rings.number_field.number_field_element_number_field_ideal_rel.NumberField_element_number_field_ideal_rel, 287

is_integral() (sage.rings.number_field.number_field_element_number_field_ideal_rel.NumberField_element_number_field_ideal_rel, 422

is_integralel_closed() (sage.rings.number_field.order.Order is_principal().NumberField_element, 235

is_isomorphic() (sage.rings.number_field.number_field_element.NumberFieldElement is_prime().NumberField_element), 63

is_isomorphic() (sage.rings.number_field.number_field_element.NumberFieldElement is_prime().NumberField_element), 105

is_isomorphic_relative() (sage.rings.number_field.number_field_element.NumberField_element, 153

is_maximal() (sage.rings.number_field.number_field_element.NumberField_element, 260

is_maximal() (sage.rings.number_field.number_field_element.NumberField_element, 273

is_maximal() (sage.rings.number_field.order.Order is_real().UniversalCyclofieldElement, 423

is_maximal() (sage.rings.number_field.order.Order is_real().UniversalCyclofieldElement), 244

is_noetherian() (sage.rings.number_field.order.Order is_real_positive().NumberField_element, 246

is_norm() (sage.rings.number_field.number_field_element.NumberField_element is_relative().NumberField_element), 235

is_nth_power() (sage.rings.number_field.number_field_element.NumberField_element is_relative().NumberField_element), 150

is_NumberField() (sage.rings.number_field.number_field_base.is_S_integral().NumberField_element, 112

is_NumberFieldElement() (in module sage.rings.number_field.number_field_element_number_field_base), 152

is_S_integral() (sage.rings.number_field.number_field_element_number_field_rel, 258

is_S_unit() (sage.rings.number_field.number_field_element_number_field_rel, 258

is_RelativeNumberField() (in module sage.rings.number_field.number_field_element_number_field_rel), 135

is_NumberField() (in module sage.rings.number_field.number_field_base, 126

is_Integer() (sage.rings.integer.is_Integer(), 175

is_Integer() (sage.rings.integer.is_Integer(), 279

is_Galois() (in module Algebraic Numbers and Number Fields, Release 9.6, 125

is_Galois_relative() (in module Algebraic Numbers and Number Fields, Release 9.6, 126

is_Galois_absolute() (in module Algebraic Numbers and Number Fields, Release 9.6, 125

is_injective() (in module Algebraic Numbers and Number Fields, Release 9.6, 223

is_integer() (in module Algebraic Numbers and Number Fields, Release 9.6, 150

is_integral() (in module Algebraic Numbers and Number Fields, Release 9.6, 180

is_integral() (in module Algebraic Numbers and Number Fields, Release 9.6, 273

is_integral() (in module Algebraic Numbers and Number Fields, Release 9.6, 422

is_integralel_closed() (in module Algebraic Numbers and Number Fields, Release 9.6, 235

is_isomorphic() (in module Algebraic Numbers and Number Fields, Release 9.6, 63

is_isomorphic() (in module Algebraic Numbers and Number Fields, Release 9.6, 105

is_isomorphic_relative() (in module Algebraic Numbers and Number Fields, Release 9.6, 153

is_maximal() (in module Algebraic Numbers and Number Fields, Release 9.6, 260

is_maximal() (in module Algebraic Numbers and Number Fields, Release 9.6, 273

is_maximal() (in module Algebraic Numbers and Number Fields, Release 9.6, 423

is_noetherian() (in module Algebraic Numbers and Number Fields, Release 9.6, 235

is_norm() (in module Algebraic Numbers and Number Fields, Release 9.6, 150

is_nth_power() (in module Algebraic Numbers and Number Fields, Release 9.6, 152

is_NumberField() (in module Algebraic Numbers and Number Fields, Release 9.6, 112

is_NumberFieldElement() (in module Algebraic Numbers and Number Fields, Release 9.6, 258

is_S_integral() (in module Algebraic Numbers and Number Fields, Release 9.6, 258

is_S_unit() (in module Algebraic Numbers and Number Fields, Release 9.6, 258
is_simple() (sage.rings.qqbar.ANDescr method), 362
is_simple() (sage.rings.qqbar.ANExtensionElement method), 366
is_simple() (sage.rings.qqbar.ANRational method), 369
is_sqrt_disc() (in module sage.rings.number_field.number_field_element_gaussian), 188
is_square() (sage.rings.number_field_number_field_element_gaussian method), 153
is_square() (sage.rings.qqbar.AlgebraicNumber_base method), 252
is_suborder() (sage.rings.number_field.order.Order method), 235
is_suborder() (sage.rings.number_field.order.Order_rel method), 247
is_surjective() (sage.rings.number_field.maps.NumberFieldHomsetrequiring_number_field_homset method), 223
is_totally_imaginary() (sage.rings.number_field_number_field_element_gaussian method), 64
is_totally_positive() (sage.rings.number_field_number_field_element_gaussian method), 154
is_totally_real() (sage.rings.number_field_number_field_element_gaussian method), 64
is_trivial() (sage.rings.qqbar.AlgebraicGenerator method), 380
is_unit() (sage.rings.number_field_number_field_element_gaussian method), 154
is_zero() (sage.rings.number_field_number_field_ideal method), 274
is_zero() (sage.rings.number_field_number_field_ideal_rel method), 287
isolating_interval() (in module sage.rings.qqbar), 410

K
K0_func() (in module sage.rings.number_field_S_unit_s solving), 304
K1_func() (in module sage.rings.number_field_S_unit_s solving), 305
key() (sage.rings.number_field_splitting_field.SplittingData method), 188
krull_dimension() (sage.rings.number_field_order.Order method), 236

L
lagrange_degree_3() (in module sage.rings.number_field_totallyreal_data), 347
left_import() (in module sage.rings.universal_cyclotomic_field), 427
latex_variable_name() (sage.rings.number_field_number_field_element_gaussian method), 64

M
MapAbsoluteToRelativeNumberField (class in sage.rings.number_field.maps), 219
MapNumberFieldToVectorSpace (class in sage.rings.number_field.maps), 219
MapRelativeNumberFieldToRelativeVectorSpace (class in sage.rings.number_field.maps), 219
Algebraic Numbers and Number Fields, Release 9.6

sage.rings.number_field.number_field_element_quadratic

NumberFieldElement_quadratic_sqrt (class in sage.rings.number_field.number_field_element_quadratic), 177

NumberFieldElement_relative (class in sage.rings.number_field.number_field_element), 170

NumberFieldEmbedding (class in sage.rings.number_field.number_field_morphisms), 216

NumberFieldFactory (class in sage.rings.number_field.number_field), 8

NumberFieldFractionalIdeal (class in sage.rings.number_field.number_field_ideal), 252

NumberFieldFractionalIdeal_rel (class in sage.rings.number_field.number_field_ideal_rel), 281

NumberFieldHomomorphism_im_gens (class in sage.rings.number_field.morphism), 209

NumberFieldHomset (class in sage.rings.number_field.homset), 211

NumberFieldIdeal (class in sage.rings.number_field.number_field_ideal), 266

NumberFieldIsomorphism (class in sage.rings.number_field.maps), 223

NumberFieldStructure (class in sage.rings.number_field.structure), 225

NumberFieldTower() (in module sage.rings.number_field.number_field), 22

numerator() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 182

numerator() (sage.rings.number_field.number_field_ideal.NumberFieldFractionalIdeal method), 260

numerator_ideal() (sage.rings.number_field.number_field_element.NumberFieldElement method), 160

Order (class in sage.rings.number_field.order), 229

Order_absolute (class in sage.rings.number_field.order), 241

Order_relative (class in sage.rings.number_field.order), 245

OrderElement_absolute (class in sage.rings.number_field.number_field_element), 172

OrderElement_quadratic (class in sage.rings.number_field.number_field_element_quadratic), 185

OrderElement_relative (class in sage.rings.number_field.number_field_element), 173

OrderFactory (class in sage.rings.number_field.order), 240

pari_absolute_base_polynomial() (sage.rings.number_field.number_field_rel.NumberField_relative method), 128

pari_absolute_base_polynomial() (sage.rings.number_field.number_field.order method), 128

pari_bnf() (sage.rings.number_field.number_field.NumberField_generic method), 128

pari_field() (sage.rings.qqbar.AlgebraicGenerator method), 380

pari_polynomial() (sage.rings.number_field.number_field.order method), 128

PariAbsoluteBase_polynomial() (sage.rings.number_field.number_field.rel.NumberField_relative method), 233

odlyzko_bound_totallyreal() (in module sage.rings.number_field.totallyreal), 341

OK() (sage.rings.number_field.number_field_base.NumberFieldGeneric method), 110

Omega_prime() (in module sage.rings.number_field.S_unit_solver), 305

one() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 419

optimized_representation() (sage.rings.number_field.number_field.NumberField_element_quadratic method), 25

optimized_subfields() (sage.rings.number_field.number_field.NumberField_element method), 25

pari_absolute_base_polynomial() (sage.rings.number_field.number_field.rel.NumberField_relative method), 233

pari_bnf() (sage.rings.number_field.number_field.NumberField_generic method), 128

pari_field() (sage.rings.qqbar.AlgebraicGenerator method), 380

pari_polynomial() (sage.rings.number_field.number_field.order method), 128

pari_bnf() (sage.rings.number_field.number_field.NumberField_generic method), 128

pari_field() (sage.rings.qqbar.AlgebraicGenerator method), 380

pari_polynomial() (sage.rings.number_field.number_field.order method), 128

446 Index
Algebraic Numbers and Number Fields, Release 9.6

pari_prime() (sage.rings.number_field.number_field_ideal.NumberFieldIdeal_part() method), 275

pari_relative_polynomial() (sage.rings.number_field.number_field_rel.NumberField_relative(method), 129

pari_rnfnorm_data() (sage.rings.number_field.number_field.NumberField_generic method), 72

pari_zk() (sage.rings.number_field.number_field_number_field Полynomials and Number Fields), 72

parts() (sage.rings.number_field.number_field_element_quadratic.NumberFieldElement_quadratic method), 182

places() (sage.rings.number_field.number_field.NumberField_absolut(method), 78

places() (sage.rings.number_field.number_field_rel.NumberField_relative(method), 129

poldegree() (sage.rings.number_field.spli9ting_field.SplittingData method), 78

poly() (sage.rings.qbar.AlgebraicPolynomialTracker(method), 396

polynomial() (sage.rings.number_field.number_field.NumberField_generic method), 79

polynomial() (sage.rings.number_field.number_field_element.NumberFieldElement method), 161

polynomial() (sage.rings.number_field.number_field_element.NumberFieldElement method), 130

polynomial_ntl() (sage.rings.number_field.number_field.NumberField_element_number_field(method), 107

polynomial_quotient_ring() (sage.rings.number_field.number_field.NumberField_quotient_ring() method), 73

polynomial_root() (sage.rings.qbar.AlgebraicField method), 376

polynomial_root() (sage.rings.qbar.AlgebraicRealField method), 402

possible_mut() (in module sage.rings.number_field.S_unit_solver, 325

power_basis() (sage.rings.number_field_number_field() method), 73

prec_seq() (in module sage.rings.qbar, 414

preimage() (sage.rings.number_field.morphism.NumberFieldHomomorphism_in method), 209

prime_above() (sage.rings.number_field.number_field.NumberField_num() method), 73

prime_factors() (sage.rings.number_field_number_field() method), 75

prime_factors() (sage.rings.number_field_number_field() method), 260
split_primes_large_lcm() (in module sage.rings.number_field.number_field.S_unit_solver), 330

splitting_field() (in module sage.rings.number_field.splitting_field), 189

SplittingData (class in sage.rings.number_field.splitting_field), 188

SplittingFieldAbort, 189

sqrt() (sage.rings.number_field.number_field_element.NumberFieldElement method), 164

sqrt() (sage.rings.qqbar.AlgebraicNumber_base method), 419

specified_complex_embedding() (sage.rings.number_field.number_field.NumberField method), 89

transitive_number() (sage.rings.number_field.galois_group.GaloisGroup_v2 method), 203

UCF_sqrt_int() (in module sage.rings.universal_cyclotomic_field), 417

UCFtoQQbar (class in sage.rings.universal_cyclotomic_field.generic)

uniformizer() (sage.rings.number_field.number_field_number_field_element_quadratic.Order method), 134

unit() (sage.rings.qqbar.AlgebraicGenerator method), 381

unit_group() (sage.rings.number_field.number_field_number_field_element_quadratic.Order method), 95

unit_group_trivial() (sage.rings.number_field.unit_group)

units() (sage.rings.number_field.number_field_element_quadratic.OrderElement_quadratic method), 96

UniversalCyclotomicField (class in sage.rings.universal_cyclotomic_field), 418

UniversalCyclotomicFieldElement (class in sage.rings.universal_cyclotomic_field), 420

valuation() (sage.rings.number_field.number_field_element_quadratic.Order method), 97

valuation() (sage.rings.number_field.number_field_element.Order method), 166

valuation() (sage.rings.number_field.number_field_element_order.Order method), 172

valuation() (sage.rings.number_field.number_field_element_number_field_element QuotientOrder method), 278

valuation() (sage.rings.number_field.number_field_element_number_field_ideal.QuotientOrder method), 290

valuation() (sage.rings.number_field.number_field_element.Order method), 239

vector() (sage.rings.number_field.number_field_element_number_field_element method), 167

vector_space() (sage.rings.number_field.number_field_element_number_field_ideal method), 135
W

weed_fields() (in module sage.rings.number_field.totallyreal), 342

Y

Yu_a1_kappa1_c1() (in module sage.rings.number_field.S_unit_solver), 306
Yu_bound() (in module sage.rings.number_field.S_unit_solver), 307
Yu_C1_star() (in module sage.rings.number_field.S_unit_solver), 306
Yu_condition_115() (in module sage.rings.number_field.S_unit_solver), 307
Yu_modified_height() (in module sage.rings.number_field.S_unit_solver), 308

Z

Z_to_quadratic_field_element (class in sage.rings.number_field.number_field_element_quadratic), 188
zero() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 419
zeta() (sage.rings.number_field.number_field.NumberField_cyclotomic method), 37
zeta() (sage.rings.number_field.number_field.NumberField_generic method), 98
zeta() (sage.rings.number_field.order.Order method), 240
zeta() (sage.rings.number_field.unit_group.UnitGroup method), 303
zeta() (sage.rings.qqbar.AlgebraicField method), 377
zeta() (sage.rings.qqbar.AlgebraicRealField method), 404
zeta() (sage.rings.universal_cyclotomic_field.UniversalCyclotomicField method), 419
zeta_coefficients() (sage.rings.number_field.number_field.NumberField_generic method), 99
zeta_order() (sage.rings.number_field.number_field.NumberField_cyclotomic method), 38
zeta_order() (sage.rings.number_field.number_field.NumberField_generic method), 99
zeta_order() (sage.rings.number_field.unit_group.UnitGroup method), 303