Quaternion Algebras

Release 10.6

The Sage Development Team

Jun 27, 2025

CONTENTS

CHAPTER

ONE

QUATERNION ALGEBRAS

AUTHORS:

- Jon Bobber (2009): rewrite
- William Stein (2009): rewrite
- Julian Rueth (2014-03-02): use UniqueFactory for caching
- Peter Bruin (2021): do not require the base ring to be a field
- Lorenz Panny (2022): *QuaternionOrder.isomorphism_to()*, *QuaternionFractionalIdeal_rational.minimal_element()*
- Sebastian A. Spindler (2024): extend ramification functionality to number fields, adapt *QuaternionAlgebra_ab.maximal_order()* to allow for extension of an order
- Eloi Torrents (2024): construct quaternion algebras over number fields from ramification

This code is partly based on Sage code by David Kohel from 2005.

class sage.algebras.quatalg.quaternion_algebra.**QuaternionAlgebraFactory**

Bases: UniqueFactory

Construct a quaternion algebra.

INPUT:

There are four input formats:

- QuaternionAlgebra(a, b), where a and b can be coerced to units in a common field K of characteristic different from 2.
- QuaternionAlgebra(K, a, b), where K is a ring in which 2 is a unit and a and b are units of K.
- QuaternionAlgebra(D), where $D \geq 1$ is a squarefree integer. This constructs a quaternion algebra of discriminant D over $K = \mathbf{Q}$. Suitable nonzero rational numbers a, b as above are deduced from D.
- QuaternionAlgebra(K, primes, inv_archimedean), where K is a number field or $\mathbf{Q}$, primes is a list of prime ideals of K and inv_archimedean is a list of local invariants (0 or $\frac{1}{2}$) specifying the ramification at the (infinite) real places of K. This constructs a quaternion algebra ramified exacly at the places given by primes and those (algebraic) real embeddings of K indexed in K.embeddings(AA) by l with inv_archimedean[l] = 1/2.

OUTPUT:

The quaternion algebra $(a, b)_K$ over K generated by i, j subject to $i^2 = a$, $j^2 = b$, and $ji = -ij$.

EXAMPLES:

QuaternionAlgebra(a, b) – return the quaternion algebra $(a, b)_K$, where the base ring K is a suitably chosen field containing a and b:

```
sage: QuaternionAlgebra(-2,-3)
Quaternion Algebra (-2, -3) with base ring Rational Field
sage: QuaternionAlgebra(GF(5)(2), GF(5)(3))
Quaternion Algebra (2, 3) with base ring Finite Field of size 5
sage: QuaternionAlgebra(2, GF(5)(3))
Quaternion Algebra (2, 3) with base ring Finite Field of size 5
sage: QuaternionAlgebra(QQ[sqrt(2)](-1), -5)                                 #␣
↪needs sage.symbolic
Quaternion Algebra (-1, -5) with base ring Number Field in sqrt2
 with defining polynomial x^2 - 2 with sqrt2 = 1.414213562373095?
sage: QuaternionAlgebra(sqrt(-1), sqrt(-3))                                  #␣
↪needs sage.symbolic
Quaternion Algebra (I, sqrt(-3)) with base ring Symbolic Ring
sage: QuaternionAlgebra(1r,1)
Quaternion Algebra (1, 1) with base ring Rational Field
sage: A.<t> = ZZ[]
sage: QuaternionAlgebra(-1, t)
Quaternion Algebra (-1, t) with base ring
 Fraction Field of Univariate Polynomial Ring in t over Integer Ring
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(2),-Integer(3))
Quaternion Algebra (-2, -3) with base ring Rational Field
>>> QuaternionAlgebra(GF(Integer(5))(Integer(2)), GF(Integer(5))(Integer(3)))
Quaternion Algebra (2, 3) with base ring Finite Field of size 5
>>> QuaternionAlgebra(Integer(2), GF(Integer(5))(Integer(3)))
Quaternion Algebra (2, 3) with base ring Finite Field of size 5
>>> QuaternionAlgebra(QQ[sqrt(Integer(2))](-Integer(1)), -Integer(5))       ␣
↪                       # needs sage.symbolic
Quaternion Algebra (-1, -5) with base ring Number Field in sqrt2
 with defining polynomial x^2 - 2 with sqrt2 = 1.414213562373095?
>>> QuaternionAlgebra(sqrt(-Integer(1)), sqrt(-Integer(3)))                  ␣
↪              # needs sage.symbolic
Quaternion Algebra (I, sqrt(-3)) with base ring Symbolic Ring
>>> QuaternionAlgebra(1,Integer(1))
Quaternion Algebra (1, 1) with base ring Rational Field
>>> A = ZZ['t']; (t,) = A._first_ngens(1)
>>> QuaternionAlgebra(-Integer(1), t)
Quaternion Algebra (-1, t) with base ring
 Fraction Field of Univariate Polynomial Ring in t over Integer Ring
```

Python ints and floats may be passed to the `QuaternionAlgebra(a, b)` constructor, as may all pairs of nonzero elements of a domain not of characteristic 2.

The following tests address the issues raised in Issue #10601:

```
sage: QuaternionAlgebra(1r,1)
Quaternion Algebra (1, 1) with base ring Rational Field
sage: QuaternionAlgebra(1,1.0r)
Quaternion Algebra (1.00000000000000, 1.00000000000000) with base ring
 Real Field with 53 bits of precision
sage: QuaternionAlgebra(0,0)
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
ValueError: defining elements of quaternion algebra (0, 0)
are not invertible in Rational Field
sage: QuaternionAlgebra(GF(2)(1),1)
Traceback (most recent call last):
...
ValueError: 2 is not invertible in Finite Field of size 2
sage: a = PermutationGroupElement([1,2,3])
sage: QuaternionAlgebra(a, a)
Traceback (most recent call last):
...
ValueError: a and b must be elements of a ring with characteristic not 2
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(1,Integer(1))
Quaternion Algebra (1, 1) with base ring Rational Field
>>> QuaternionAlgebra(Integer(1),1.0)
Quaternion Algebra (1.00000000000000, 1.00000000000000) with base ring
 Real Field with 53 bits of precision
>>> QuaternionAlgebra(Integer(0),Integer(0))
Traceback (most recent call last):
...
ValueError: defining elements of quaternion algebra (0, 0)
are not invertible in Rational Field
>>> QuaternionAlgebra(GF(Integer(2))(Integer(1)),Integer(1))
Traceback (most recent call last):
...
ValueError: 2 is not invertible in Finite Field of size 2
>>> a = PermutationGroupElement([Integer(1),Integer(2),Integer(3)])
>>> QuaternionAlgebra(a, a)
Traceback (most recent call last):
...
ValueError: a and b must be elements of a ring with characteristic not 2
```

`QuaternionAlgebra(K, a, b)` – return the quaternion algebra defined by (a, b) over the ring K:

```
sage: QuaternionAlgebra(QQ, -7, -21)
Quaternion Algebra (-7, -21) with base ring Rational Field
sage: QuaternionAlgebra(QQ[sqrt(2)], -2,-3)                                  #␣
↪needs sage.symbolic
Quaternion Algebra (-2, -3) with base ring Number Field in sqrt2
 with defining polynomial x^2 - 2 with sqrt2 = 1.414213562373095?
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QQ, -Integer(7), -Integer(21))
Quaternion Algebra (-7, -21) with base ring Rational Field
>>> QuaternionAlgebra(QQ[sqrt(Integer(2))], -Integer(2),-Integer(3))          ␣
↪                        # needs sage.symbolic
Quaternion Algebra (-2, -3) with base ring Number Field in sqrt2
 with defining polynomial x^2 - 2 with sqrt2 = 1.414213562373095?
```

`QuaternionAlgebra(D)` – D is a squarefree integer; return a rational quaternion algebra of discriminant D:

```
sage: QuaternionAlgebra(1)
Quaternion Algebra (-1, 1) with base ring Rational Field
sage: QuaternionAlgebra(2)
Quaternion Algebra (-1, -1) with base ring Rational Field
sage: QuaternionAlgebra(7)
Quaternion Algebra (-1, -7) with base ring Rational Field
sage: QuaternionAlgebra(2*3*5*7)
Quaternion Algebra (-22, 210) with base ring Rational Field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(Integer(1))
Quaternion Algebra (-1, 1) with base ring Rational Field
>>> QuaternionAlgebra(Integer(2))
Quaternion Algebra (-1, -1) with base ring Rational Field
>>> QuaternionAlgebra(Integer(7))
Quaternion Algebra (-1, -7) with base ring Rational Field
>>> QuaternionAlgebra(Integer(2)*Integer(3)*Integer(5)*Integer(7))
Quaternion Algebra (-22, 210) with base ring Rational Field
```

`QuaternionAlgebra(K, primes, inv_archimedean)` – return the quaternion algebra over K with the ramification specified by `primes` and `inv_archimedean`:

```
sage: QuaternionAlgebra(QQ, [(2), (3)], [0])
Quaternion Algebra (-1, 3) with base ring Rational Field
sage: QuaternionAlgebra(QQ, [(2), (3)], [1/2])
Traceback (most recent call last):
...
ValueError: quaternion algebra over the rationals must have an even number of␣
↪ramified places

sage: x = polygen(ZZ, 'x')
sage: K.<w> = NumberField(x^2-x-1)
sage: P = K.prime_above(2)
sage: Q = K.prime_above(3)
sage: A = QuaternionAlgebra(K, [P,Q], [0,0])
sage: A.discriminant()
Fractional ideal (6)
sage: A = QuaternionAlgebra(K, [P,Q], [1/2,0])
Traceback (most recent call last):
...
ValueError: quaternion algebra over a number field must have an even number of␣
↪ramified places
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QQ, [(Integer(2)), (Integer(3))], [Integer(0)])
Quaternion Algebra (-1, 3) with base ring Rational Field
>>> QuaternionAlgebra(QQ, [(Integer(2)), (Integer(3))], [Integer(1)/Integer(2)])
Traceback (most recent call last):
...
ValueError: quaternion algebra over the rationals must have an even number of␣
↪ramified places
```

(continues on next page)

(continued from previous page)

```
>>> x = polygen(ZZ, 'x')
>>> K = NumberField(x**Integer(2)-x-Integer(1), names=('w',)); (w,) = K._first_
↪ngens(1)
>>> P = K.prime_above(Integer(2))
>>> Q = K.prime_above(Integer(3))
>>> A = QuaternionAlgebra(K, [P,Q], [Integer(0),Integer(0)])
>>> A.discriminant()
Fractional ideal (6)
>>> A = QuaternionAlgebra(K, [P,Q], [Integer(1)/Integer(2),Integer(0)])
Traceback (most recent call last):
...
ValueError: quaternion algebra over a number field must have an even number of␣
↪ramified places
```

The construction via ramification fails if the base field is not a number field:

```
sage: QuaternionAlgebra(RR, [], [])
Traceback (most recent call last):
...
ValueError: quaternion algebra construction via ramification only works over a␣
↪number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR, [], [])
Traceback (most recent call last):
...
ValueError: quaternion algebra construction via ramification only works over a␣
↪number field
```

The list of local invariants must specify the ramification data at all real places of the number field K:

```
sage: QuaternionAlgebra(QuadraticField(5), [], [0])
Traceback (most recent call last):
...
ValueError: must specify ramification at all real places of the number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QuadraticField(Integer(5)), [], [Integer(0)])
Traceback (most recent call last):
...
ValueError: must specify ramification at all real places of the number field
```

The list of local invariants specifying the ramification at the real places may only contain 0 and $\frac{1}{2}$:

```
sage: QuaternionAlgebra(QuadraticField(5), [], [0,1])
Traceback (most recent call last):
...
ValueError: list of local invariants specifying ramification should contain only␣
↪0 and 1/2
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QuadraticField(Integer(5)), [], [Integer(0),Integer(1)])
```

(continues on next page)

(continued from previous page)

```
Traceback (most recent call last):
...
ValueError: list of local invariants specifying ramification should contain only␣
↪0 and 1/2
```

Similarly, the list of finite ramified places must consist of primes or prime ideals of the number field K:

```
sage: QuaternionAlgebra(QuadraticField(5), ['water',2], [])
Traceback (most recent call last):
...
ValueError: quaternion algebra constructor requires a list of primes specifying␣
↪the ramification
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QuadraticField(Integer(5)), ['water',Integer(2)], [])
Traceback (most recent call last):
...
ValueError: quaternion algebra constructor requires a list of primes specifying␣
↪the ramification
```

If the coefficients a and b in the definition of the quaternion algebra are not integral, then a slower generic type is used for arithmetic:

```
sage: type(QuaternionAlgebra(-1,-3).0)
<... 'sage.algebras.quatalg.quaternion_algebra_element.QuaternionAlgebraElement_
↪rational_field'>
sage: type(QuaternionAlgebra(-1,-3/2).0)
<... 'sage.algebras.quatalg.quaternion_algebra_element.QuaternionAlgebraElement_
↪generic'>
```

```
>>> from sage.all import *
>>> type(QuaternionAlgebra(-Integer(1),-Integer(3)).gen(0))
<... 'sage.algebras.quatalg.quaternion_algebra_element.QuaternionAlgebraElement_
↪rational_field'>
>>> type(QuaternionAlgebra(-Integer(1),-Integer(3)/Integer(2)).gen(0))
<... 'sage.algebras.quatalg.quaternion_algebra_element.QuaternionAlgebraElement_
↪generic'>
```

Make sure caching is sane:

```
sage: A = QuaternionAlgebra(2,3); A
Quaternion Algebra (2, 3) with base ring Rational Field
sage: B = QuaternionAlgebra(GF(5)(2),GF(5)(3)); B
Quaternion Algebra (2, 3) with base ring Finite Field of size 5
sage: A is QuaternionAlgebra(2,3)
True
sage: B is QuaternionAlgebra(GF(5)(2),GF(5)(3))
True
sage: Q = QuaternionAlgebra(2); Q
Quaternion Algebra (-1, -1) with base ring Rational Field
sage: Q is QuaternionAlgebra(QQ,-1,-1)
True
```

(continues on next page)

(continued from previous page)

```
sage: Q is QuaternionAlgebra(-1,-1)
True
sage: Q.<ii,jj,kk> = QuaternionAlgebra(15); Q.variable_names()
('ii', 'jj', 'kk')
sage: QuaternionAlgebra(15).variable_names()
('i', 'j', 'k')
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(Integer(2),Integer(3)); A
Quaternion Algebra (2, 3) with base ring Rational Field
>>> B = QuaternionAlgebra(GF(Integer(5))(Integer(2)),GF(Integer(5))(Integer(3)));
↪B
Quaternion Algebra (2, 3) with base ring Finite Field of size 5
>>> A is QuaternionAlgebra(Integer(2),Integer(3))
True
>>> B is QuaternionAlgebra(GF(Integer(5))(Integer(2)),GF(Integer(5))(Integer(3)))
True
>>> Q = QuaternionAlgebra(Integer(2)); Q
Quaternion Algebra (-1, -1) with base ring Rational Field
>>> Q is QuaternionAlgebra(QQ,-Integer(1),-Integer(1))
True
>>> Q is QuaternionAlgebra(-Integer(1),-Integer(1))
True
>>> Q = QuaternionAlgebra(Integer(15), names=('ii', 'jj', 'kk',)); (ii, jj, kk,)
↪= Q._first_ngens(3); Q.variable_names()
('ii', 'jj', 'kk')
>>> QuaternionAlgebra(Integer(15)).variable_names()
('i', 'j', 'k')
```

create_key(*arg0*, *arg1=None*, *arg2=None*, *names='i,j,k'*)

Create a key that uniquely determines a quaternion algebra.

create_object(*version*, *key*, ***extra_args*)

Create the object from the key (extra arguments are ignored). This is only called if the object was not found in the cache.

class sage.algebras.quatalg.quaternion_algebra.**QuaternionAlgebra_ab**(*base_ring*, *a*, *b*, *names='i,j,k'*)

Bases: `QuaternionAlgebra_abstract`

A quaternion algebra of the form $(a, b)_K$.

See `QuaternionAlgebra` for many more examples.

INPUT:

- `base_ring` – a commutative ring K in which 2 is invertible
- `a`, `b` – units of K
- `names` – string (default: `'i,j,k'`); names of the generators

OUTPUT:

The quaternion algebra (a, b) over K generated by i and j subject to $i^2 = a$, $j^2 = b$, and $ji = -ij$.

EXAMPLES:

```
sage: QuaternionAlgebra(QQ, -7, -21)  # indirect doctest
Quaternion Algebra (-7, -21) with base ring Rational Field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QQ, -Integer(7), -Integer(21))  # indirect doctest
Quaternion Algebra (-7, -21) with base ring Rational Field
```

discriminant()

Return the discriminant of this quaternion algebra.

The discriminant of a quaternion algebra over a number field is the product of the finite places at which the algebra ramifies.

OUTPUT:

The discriminant of this quaternion algebra, given as

- an element of $\mathbf{Z}$ if the algebra is defined over $\mathbf{Q}$,
- an integral fractional ideal of the base number field, otherwise.

EXAMPLES:

```
sage: QuaternionAlgebra(210, -22).discriminant()
210
sage: QuaternionAlgebra(19).discriminant()
19
sage: QuaternionAlgebra(-1, -1).discriminant()
2
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(Integer(210), -Integer(22)).discriminant()
210
>>> QuaternionAlgebra(Integer(19)).discriminant()
19
>>> QuaternionAlgebra(-Integer(1), -Integer(1)).discriminant()
2
```

Some examples over number fields:

```
sage: K = QuadraticField(3)
sage: L = QuadraticField(-15)
sage: QuaternionAlgebra(K, -1, -1).discriminant()
Fractional ideal (1)
sage: QuaternionAlgebra(L, -1, -1).discriminant()
Fractional ideal (2)
```

```
>>> from sage.all import *
>>> K = QuadraticField(Integer(3))
>>> L = QuadraticField(-Integer(15))
>>> QuaternionAlgebra(K, -Integer(1), -Integer(1)).discriminant()
Fractional ideal (1)
>>> QuaternionAlgebra(L, -Integer(1), -Integer(1)).discriminant()
Fractional ideal (2)
```

We can also use number field elements as invariants:

```
sage: x = polygen(ZZ, 'x')
sage: F.<a> = NumberField(x^2 - x - 1)
sage: QuaternionAlgebra(F, 2*a, F(-1)).discriminant()
Fractional ideal (2)
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> F = NumberField(x**Integer(2) - x - Integer(1), names=('a',)); (a,) = F._
↪first_ngens(1)
>>> QuaternionAlgebra(F, Integer(2)*a, F(-Integer(1))).discriminant()
Fractional ideal (2)
```

The method does not make sense over an arbitrary base ring:

```
sage: QuaternionAlgebra(RR(2.),1).discriminant()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR(RealNumber('2.')),Integer(1)).discriminant()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

gen (*i=0*)

Return the i-th generator of `self`.

INPUT:

- `i` – integer (default: 0)

EXAMPLES:

```
sage: Q.<ii,jj,kk> = QuaternionAlgebra(QQ,-1,-2); Q
Quaternion Algebra (-1, -2) with base ring Rational Field
sage: Q.gen(0)
ii
sage: Q.gen(1)
jj
sage: Q.gen(2)
kk
sage: Q.gens()
(ii, jj, kk)
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ,-Integer(1),-Integer(2), names=('ii', 'jj', 'kk',
↪)); (ii, jj, kk,) = Q._first_ngens(3); Q
Quaternion Algebra (-1, -2) with base ring Rational Field
>>> Q.gen(Integer(0))
ii
>>> Q.gen(Integer(1))
jj
```

(continues on next page)

(continued from previous page)

```
>>> Q.gen(Integer(2))
kk
>>> Q.gens()
(ii, jj, kk)
```

gens()

Return the generators of `self`.

EXAMPLES:

```
sage: Q.<ii,jj,kk> = QuaternionAlgebra(QQ,-1,-2); Q
Quaternion Algebra (-1, -2) with base ring Rational Field
sage: Q.gens()
(ii, jj, kk)
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ,-Integer(1),-Integer(2), names=('ii', 'jj', 'kk',
↪)); (ii, jj, kk,) = Q._first_ngens(3); Q
Quaternion Algebra (-1, -2) with base ring Rational Field
>>> Q.gens()
(ii, jj, kk)
```

ideal(*gens*, *left_order=None*, *right_order=None*, *check=True*, ***kwds*)

Return the quaternion ideal with given gens over $\mathbf{Z}$.

Neither a left or right order structure need be specified.

INPUT:

- `gens` – list of elements of this quaternion order
- `check` – boolean (default: `True`)
- `left_order` – a quaternion order or `None`
- `right_order` – a quaternion order or `None`

EXAMPLES:

```
sage: R = QuaternionAlgebra(-11,-1)
sage: R.ideal([2*a for a in R.basis()])
Fractional ideal (2, 2*i, 2*j, 2*k)
```

```
>>> from sage.all import *
>>> R = QuaternionAlgebra(-Integer(11),-Integer(1))
>>> R.ideal([Integer(2)*a for a in R.basis()])
Fractional ideal (2, 2*i, 2*j, 2*k)
```

inner_product_matrix()

Return the inner product matrix associated to `self`, i.e. the Gram matrix of the reduced norm as a quadratic form on `self`. The standard basis 1, i, j, k is orthogonal, so this matrix is just the diagonal matrix with diagonal entries 1, a, b, ab.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-5,-19)
sage: Q.inner_product_matrix()
[  2   0   0   0]
[  0  10   0   0]
[  0   0  38   0]
[  0   0   0 190]

sage: R.<a,b> = QQ[]; Q.<i,j,k> = QuaternionAlgebra(Frac(R),a,b)
sage: Q.inner_product_matrix()
[    2      0      0      0]
[    0   -2*a      0      0]
[    0      0   -2*b      0]
[    0      0      0  2*a*b]
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(5),-Integer(19), names=('i', 'j', 'k',));␣
↪(i, j, k,) = Q._first_ngens(3)
>>> Q.inner_product_matrix()
[  2   0   0   0]
[  0  10   0   0]
[  0   0  38   0]
[  0   0   0 190]

>>> R = QQ['a, b']; (a, b,) = R._first_ngens(2); Q =␣
↪QuaternionAlgebra(Frac(R),a,b, names=('i', 'j', 'k',)); (i, j, k,) = Q._
↪first_ngens(3)
>>> Q.inner_product_matrix()
[    2      0      0      0]
[    0   -2*a      0      0]
[    0      0   -2*b      0]
[    0      0      0  2*a*b]
```

invariants()

Return the structural invariants a, b of this quaternion algebra: `self` is generated by i, j subject to $i^2 = a$, $j^2 = b$ and $ji = -ij$.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(15)
sage: Q.invariants()
(-3, 5)
sage: i^2
-3
sage: j^2
5
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(Integer(15), names=('i', 'j', 'k',)); (i, j, k,) =␣
↪Q._first_ngens(3)
>>> Q.invariants()
(-3, 5)
>>> i**Integer(2)
-3
```

(continues on next page)

(continued from previous page)

```
>>> j**Integer(2)
5
```

is_definite()

Check whether this quaternion algebra is definite.

A quaternion algebra over **Q** is definite if it ramifies at the unique real place of **Q**, which happens if and only if both of its invariants are negative (see Exercise 2.4(c) in [Voi2021]).

EXAMPLES:

```
sage: QuaternionAlgebra(QQ,-5,-2).is_definite()
True
sage: QuaternionAlgebra(1).is_definite()
False
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QQ,-Integer(5),-Integer(2)).is_definite()
True
>>> QuaternionAlgebra(Integer(1)).is_definite()
False
```

The method does not make sense over an arbitrary base ring:

```
sage: QuaternionAlgebra(RR(2.), 1).is_definite()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR(RealNumber('2.')), Integer(1)).is_definite()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers
```

is_isomorphic(*A*)

Check whether this quaternion algebra is isomorphic to `A`.

Currently only implemented for quaternion algebras defined over a number field; based on Main Theorem 14.6.1 in [Voi2021], noting that **Q** has a unique infinite place.

INPUT:

- `A` – a quaternion algebra defined over a number field

EXAMPLES:

```
sage: B = QuaternionAlgebra(-46, -87)
sage: A = QuaternionAlgebra(-58, -69)
sage: A == B
False
sage: B.is_isomorphic(A)
True
```

```
>>> from sage.all import *
>>> B = QuaternionAlgebra(-Integer(46), -Integer(87))
>>> A = QuaternionAlgebra(-Integer(58), -Integer(69))
>>> A == B
False
>>> B.is_isomorphic(A)
True
```

Checking ramification at both finite and infinite places, the method correctly distinguishes isomorphism classes of quaternion algebras that the discriminant can not distinguish:

```
sage: K = QuadraticField(3)
sage: A = QuaternionAlgebra(K, -1, -1)
sage: B = QuaternionAlgebra(K, 1, -1)
sage: A.discriminant() == B.discriminant()
True
sage: B.is_isomorphic(A)
False
```

```
>>> from sage.all import *
>>> K = QuadraticField(Integer(3))
>>> A = QuaternionAlgebra(K, -Integer(1), -Integer(1))
>>> B = QuaternionAlgebra(K, Integer(1), -Integer(1))
>>> A.discriminant() == B.discriminant()
True
>>> B.is_isomorphic(A)
False
```

is_totally_definite()

Check whether this quaternion algebra is totally definite.

A quaternion algebra defined over a number field is totally definite if it ramifies at all Archimedean places of its base field. In particular, the base number field has to be totally real (see 14.5.8 in [Voi2021]).

EXAMPLES:

```
sage: QuaternionAlgebra(QQ, -5, -2).is_totally_definite()
True

sage: K = QuadraticField(3)
sage: QuaternionAlgebra(K, -1, -1).is_totally_definite()
True
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QQ, -Integer(5), -Integer(2)).is_totally_definite()
True

>>> K = QuadraticField(Integer(3))
>>> QuaternionAlgebra(K, -Integer(1), -Integer(1)).is_totally_definite()
True
```

We can also use number field elements as invariants:

```
sage: x = polygen(ZZ, 'x')
sage: F.<a> = NumberField(x^2 - x - 1)
sage: QuaternionAlgebra(F, 2*a, F(-1)).is_totally_definite()
False
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> F = NumberField(x**Integer(2) - x - Integer(1), names=('a',)); (a,) = F._
↪first_ngens(1)
>>> QuaternionAlgebra(F, Integer(2)*a, F(-Integer(1))).is_totally_definite()
False
```

The method does not make sense over an arbitrary base ring:

```
sage: QuaternionAlgebra(RR(2.), 1).is_totally_definite()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR(RealNumber('2.')), Integer(1)).is_totally_definite()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

maximal_order(*take_shortcuts=True*, *order_basis=None*)

Return a maximal order in this quaternion algebra.

If `order_basis` is specified, the resulting maximal order will contain the order of the quaternion algebra given by this basis. The algorithm used is from [Voi2012].

INPUT:

- `take_shortcuts` – boolean (default: `True`); if the discriminant is prime and the invariants of the algebra are of a nice form, use Proposition 5.2 of [Piz1980].
- `order_basis` – (default: `None`) a basis of an order of this quaternion algebra

OUTPUT: a maximal order in this quaternion algebra

EXAMPLES:

```
sage: QuaternionAlgebra(-1,-7).maximal_order()
Order of Quaternion Algebra (-1, -7) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)

sage: QuaternionAlgebra(-1,-1).maximal_order().basis()
(1/2 + 1/2*i + 1/2*j + 1/2*k, i, j, k)

sage: QuaternionAlgebra(-1,-11).maximal_order().basis()
(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)

sage: QuaternionAlgebra(-1,-3).maximal_order().basis()
(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)

```

(continues on next page)

(continued from previous page)

```
sage: QuaternionAlgebra(-3,-1).maximal_order().basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)

sage: QuaternionAlgebra(-2,-5).maximal_order().basis()
(1/2 + 1/2*j + 1/2*k, 1/4*i + 1/2*j + 1/4*k, j, k)

sage: QuaternionAlgebra(-5,-2).maximal_order().basis()
(1/2 + 1/2*i - 1/2*k, 1/2*i + 1/4*j - 1/4*k, i, -k)

sage: QuaternionAlgebra(-17,-3).maximal_order().basis()
(1/2 + 1/2*j, 1/2*i + 1/2*k, -1/3*j - 1/3*k, k)

sage: QuaternionAlgebra(-3,-17).maximal_order().basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, -1/3*i + 1/3*k, -k)

sage: QuaternionAlgebra(-17*9,-3).maximal_order().basis()
(1, 1/3*i, 1/6*i + 1/2*j, 1/2 + 1/3*j + 1/18*k)

sage: QuaternionAlgebra(-2, -389).maximal_order().basis()
(1/2 + 1/2*j + 1/2*k, 1/4*i + 1/2*j + 1/4*k, j, k)
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(1),-Integer(7)).maximal_order()
Order of Quaternion Algebra (-1, -7) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)

>>> QuaternionAlgebra(-Integer(1),-Integer(1)).maximal_order().basis()
(1/2 + 1/2*i + 1/2*j + 1/2*k, i, j, k)

>>> QuaternionAlgebra(-Integer(1),-Integer(11)).maximal_order().basis()
(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)

>>> QuaternionAlgebra(-Integer(1),-Integer(3)).maximal_order().basis()
(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)

>>> QuaternionAlgebra(-Integer(3),-Integer(1)).maximal_order().basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)

>>> QuaternionAlgebra(-Integer(2),-Integer(5)).maximal_order().basis()
(1/2 + 1/2*j + 1/2*k, 1/4*i + 1/2*j + 1/4*k, j, k)

>>> QuaternionAlgebra(-Integer(5),-Integer(2)).maximal_order().basis()
(1/2 + 1/2*i - 1/2*k, 1/2*i + 1/4*j - 1/4*k, i, -k)

>>> QuaternionAlgebra(-Integer(17),-Integer(3)).maximal_order().basis()
(1/2 + 1/2*j, 1/2*i + 1/2*k, -1/3*j - 1/3*k, k)

>>> QuaternionAlgebra(-Integer(3),-Integer(17)).maximal_order().basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, -1/3*i + 1/3*k, -k)

>>> QuaternionAlgebra(-Integer(17)*Integer(9),-Integer(3)).maximal_order().
↪basis()
```

(continues on next page)

(continued from previous page)

```
(1, 1/3*i, 1/6*i + 1/2*j, 1/2 + 1/3*j + 1/18*k)

>>> QuaternionAlgebra(-Integer(2), -Integer(389)).maximal_order().basis()
(1/2 + 1/2*j + 1/2*k, 1/4*i + 1/2*j + 1/4*k, j, k)
```

If you want bases containing 1, switch off `take_shortcuts`:

```
sage: QuaternionAlgebra(-3,-89).maximal_order(take_shortcuts=False)
Order of Quaternion Algebra (-3, -89) with base ring Rational Field
 with basis (1, 1/2 + 1/2*i, j, 1/2 + 1/6*i + 1/2*j + 1/6*k)

sage: QuaternionAlgebra(1,1).maximal_order(take_shortcuts=False)     # Matrix␣
↪ring
Order of Quaternion Algebra (1, 1) with base ring Rational Field
 with basis (1, 1/2 + 1/2*i, j, 1/2*j + 1/2*k)

sage: QuaternionAlgebra(-22,210).maximal_order(take_shortcuts=False)
Order of Quaternion Algebra (-22, 210) with base ring Rational Field
 with basis (1, i, 1/2*i + 1/2*j, 1/2 + 17/22*i + 1/44*k)

sage: for d in ( m for m in range(1, 750) if is_squarefree(m) ):        #␣
↪long time (3s)
....:     A = QuaternionAlgebra(d)
....:     assert A.maximal_order(take_shortcuts=False).is_maximal()
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(3),-Integer(89)).maximal_order(take_
↪shortcuts=False)
Order of Quaternion Algebra (-3, -89) with base ring Rational Field
 with basis (1, 1/2 + 1/2*i, j, 1/2 + 1/6*i + 1/2*j + 1/6*k)

>>> QuaternionAlgebra(Integer(1),Integer(1)).maximal_order(take_
↪shortcuts=False)     # Matrix ring
Order of Quaternion Algebra (1, 1) with base ring Rational Field
 with basis (1, 1/2 + 1/2*i, j, 1/2*j + 1/2*k)

>>> QuaternionAlgebra(-Integer(22),Integer(210)).maximal_order(take_
↪shortcuts=False)
Order of Quaternion Algebra (-22, 210) with base ring Rational Field
 with basis (1, i, 1/2*i + 1/2*j, 1/2 + 17/22*i + 1/44*k)

>>> for d in ( m for m in range(Integer(1), Integer(750)) if is_squarefree(m)␣
↪):        # long time (3s)
...     A = QuaternionAlgebra(d)
...     assert A.maximal_order(take_shortcuts=False).is_maximal()
```

Specifying an order basis gives an extension of orders:

```
sage: A.<i,j,k> = QuaternionAlgebra(-292, -732)
sage: alpha = A.random_element()
sage: while alpha.is_zero():
....:     alpha = A.random_element()
```

(continues on next page)

(continued from previous page)

```
sage: conj = [alpha * b * alpha.inverse() for b in [k,i,j]]
sage: order_basis = tuple(conj) + (A.one(),)
sage: O = A.quaternion_order(basis=order_basis)
sage: R = A.maximal_order(order_basis=order_basis)
sage: O <= R and R.is_maximal()
True
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(292), -Integer(732), names=('i', 'j', 'k',
↪)); (i, j, k,) = A._first_ngens(3)
>>> alpha = A.random_element()
>>> while alpha.is_zero():
...     alpha = A.random_element()
>>> conj = [alpha * b * alpha.inverse() for b in [k,i,j]]
>>> order_basis = tuple(conj) + (A.one(),)
>>> O = A.quaternion_order(basis=order_basis)
>>> R = A.maximal_order(order_basis=order_basis)
>>> O <= R and R.is_maximal()
True
```

We do not support number fields other than the rationals yet:

```
sage: K = QuadraticField(5)
sage: QuaternionAlgebra(K,-1,-1).maximal_order()
Traceback (most recent call last):
...
NotImplementedError: maximal order only implemented
for rational quaternion algebras
```

```
>>> from sage.all import *
>>> K = QuadraticField(Integer(5))
>>> QuaternionAlgebra(K,-Integer(1),-Integer(1)).maximal_order()
Traceback (most recent call last):
...
NotImplementedError: maximal order only implemented
for rational quaternion algebras
```

modp_splitting_data(*p*)

Return mod p splitting data for this quaternion algebra at the unramified prime p.

This is 2×2 matrices I, J, K over the finite field $\mathbf{F}_p$ such that if the quaternion algebra has generators i, j, k, then $I^2 = i^2$, $J^2 = j^2$, $IJ = K$ and $IJ = -JI$.

Note

Currently only implemented when p is odd and the base ring is $\mathbf{Q}$.

INPUT:

- `p` – unramified odd prime

OUTPUT: 2-tuple of matrices over finite field

EXAMPLES:

```
sage: Q = QuaternionAlgebra(-15, -19)
sage: Q.modp_splitting_data(7)
(
[0 6]  [6 1]  [6 6]
[1 0], [1 1], [6 1]
)
sage: Q.modp_splitting_data(next_prime(10^5))
(
[    0 99988]  [97311     4]  [99999 59623]
[    1     0], [13334  2692], [97311     4]
)
sage: I,J,K = Q.modp_splitting_data(23)
sage: I
[0 8]
[1 0]
sage: I^2
[8 0]
[0 8]
sage: J
[19  2]
[17  4]
sage: J^2
[4 0]
[0 4]
sage: I*J == -J*I
True
sage: I*J == K
True
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(15), -Integer(19))
>>> Q.modp_splitting_data(Integer(7))
(
[0 6]  [6 1]  [6 6]
[1 0], [1 1], [6 1]
)
>>> Q.modp_splitting_data(next_prime(Integer(10)**Integer(5)))
(
[    0 99988]  [97311     4]  [99999 59623]
[    1     0], [13334  2692], [97311     4]
)
>>> I,J,K = Q.modp_splitting_data(Integer(23))
>>> I
[0 8]
[1 0]
>>> I**Integer(2)
[8 0]
[0 8]
>>> J
[19  2]
[17  4]
```

(continues on next page)

(continued from previous page)

```
>>> J**Integer(2)
[4 0]
[0 4]
>>> I*J == -J*I
True
>>> I*J == K
True
```

The following is a good test because of the asserts in the code:

```
sage: v = [Q.modp_splitting_data(p) for p in primes(20,1000)]
```

```
>>> from sage.all import *
>>> v = [Q.modp_splitting_data(p) for p in primes(Integer(20),Integer(1000))]
```

Proper error handling:

```
sage: Q.modp_splitting_data(5)
Traceback (most recent call last):
...
NotImplementedError: algorithm for computing local splittings
not implemented in general (currently require the first invariant
to be coprime to p)

sage: Q.modp_splitting_data(2)
Traceback (most recent call last):
...
NotImplementedError: p must be odd
```

```
>>> from sage.all import *
>>> Q.modp_splitting_data(Integer(5))
Traceback (most recent call last):
...
NotImplementedError: algorithm for computing local splittings
not implemented in general (currently require the first invariant
to be coprime to p)

>>> Q.modp_splitting_data(Integer(2))
Traceback (most recent call last):
...
NotImplementedError: p must be odd
```

modp_splitting_map(p)

Return Python map from the (p-integral) quaternion algebra to the set of 2×2 matrices over $\mathbf{F}_p$.

INPUT:

- `p` – prime number

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-1, -7)
sage: f = Q.modp_splitting_map(13)
```

(continues on next page)

(continued from previous page)

```
sage: a = 2+i-j+3*k; b = 7+2*i-4*j+k
sage: f(a*b)
[12  3]
[10  5]
sage: f(a)*f(b)
[12  3]
[10  5]
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(1), -Integer(7), names=('i', 'j', 'k',));␣
↪(i, j, k,) = Q._first_ngens(3)
>>> f = Q.modp_splitting_map(Integer(13))
>>> a = Integer(2)+i-j+Integer(3)*k; b = Integer(7)+Integer(2)*i-
↪Integer(4)*j+k
>>> f(a*b)
[12  3]
[10  5]
>>> f(a)*f(b)
[12  3]
[10  5]
```

order_with_level(*level*)

Return an order in this quaternion algebra with given level.

INPUT:

- `level` – positive integer

Currently this is only implemented when the base field is the rational numbers and the level is divisible by at most one power of a prime that ramifies in this quaternion algebra.

EXAMPLES:

```
sage: A.<i,j,k> = QuaternionAlgebra(5)
sage: level = 2 * 5 * 17
sage: O = A.order_with_level(level); O
Order of Quaternion Algebra (-2, -5) with base ring Rational Field with basis␣
↪(1/2 + 1/2*j + 7/2*k, 1/2*i + 19/2*k, j + 7*k, 17*k)
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(Integer(5), names=('i', 'j', 'k',)); (i, j, k,) = A.
↪_first_ngens(3)
>>> level = Integer(2) * Integer(5) * Integer(17)
>>> O = A.order_with_level(level); O
Order of Quaternion Algebra (-2, -5) with base ring Rational Field with basis␣
↪(1/2 + 1/2*j + 7/2*k, 1/2*i + 19/2*k, j + 7*k, 17*k)
```

Check that the order has the right index in the maximal order:

```
sage: L = O.free_module()
sage: N = A.maximal_order().free_module()
sage: L.index_in(N) == level / 5
True
```

```
>>> from sage.all import *
>>> L = O.free_module()
>>> N = A.maximal_order().free_module()
>>> L.index_in(N) == level / Integer(5)
True
```

quaternion_order(*basis*, *check=True*)

Return the order of this quaternion order with given basis.

INPUT:

- `basis` – list of 4 elements of `self`
- `check` – boolean (default: `True`)

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-11,-1)
sage: Q.quaternion_order([1,i,j,k])
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1, i, j, k)
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(11),-Integer(1), names=('i', 'j', 'k',));␣
↪(i, j, k,) = Q._first_ngens(3)
>>> Q.quaternion_order([Integer(1),i,j,k])
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1, i, j, k)
```

We test out `check=False`:

```
sage: Q.quaternion_order([1,i,j,k], check=False)
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1, i, j, k)
sage: Q.quaternion_order([i,j,k], check=False)
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (i, j, k)
```

```
>>> from sage.all import *
>>> Q.quaternion_order([Integer(1),i,j,k], check=False)
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1, i, j, k)
>>> Q.quaternion_order([i,j,k], check=False)
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (i, j, k)
```

ramified_places(*inf=True*)

Return the places of the base number field at which this quaternion algebra ramifies.

INPUT:

- `inf` – bool (default: `True`)

OUTPUT:

The non-Archimedean (AKA finite) places at which this quaternion algebra ramifies, given as

- elements of **Z** (sorted small to large) if the base field is **Q**,
- integral fractional ideals of the base number field, otherwise.

Additionally, if `inf` is set to `True`, then the Archimedean (AKA infinite) places at which the quaternion algebra ramifies are also returned, given as

- the embeddings of **Q** into **R** if the base field is **Q**, or
- the embeddings of the base number field into the Algebraic Real Field.

Note

Any Archimedean place at which a quaternion algebra ramifies has to be real (see 14.5.8 in [Voi2021]).

EXAMPLES:

```
sage: QuaternionAlgebra(210,-22).ramified_places()
([2, 3, 5, 7], [])
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(Integer(210),-Integer(22)).ramified_places()
([2, 3, 5, 7], [])
```

For a definite quaternion algebra we get ramification at the unique infinite place of **Q**:

```
sage: QuaternionAlgebra(-1, -1).ramified_places()
([2],
 [Ring morphism:
    From: Rational Field
    To:   Real Field with 53 bits of precision
    Defn: 1 |--> 1.00000000000000])
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(1), -Integer(1)).ramified_places()
([2],
 [Ring morphism:
    From: Rational Field
    To:   Real Field with 53 bits of precision
    Defn: 1 |--> 1.00000000000000])
```

Extending the base field can resolve all ramification:

```
sage: F = QuadraticField(-1)
sage: QuaternionAlgebra(F, -1, -1).ramified_places()
([], [])
```

```
>>> from sage.all import *
>>> F = QuadraticField(-Integer(1))
>>> QuaternionAlgebra(F, -Integer(1), -Integer(1)).ramified_places()
([], [])
```

Extending the base field can also resolve all ramification at finite places while still leaving some ramification at infinite places:

```
sage: K = QuadraticField(3)
sage: QuaternionAlgebra(K, -1, -1).ramified_places()
([],
 [Ring morphism:
    From: Number Field in a with defining polynomial x^2 - 3 with a = 1.
↪732050807568878?
    To:   Algebraic Real Field
    Defn: a |--> -1.732050807568878?,
  Ring morphism:
    From: Number Field in a with defining polynomial x^2 - 3 with a = 1.
↪732050807568878?
    To:   Algebraic Real Field
    Defn: a |--> 1.732050807568878?])
```

```
>>> from sage.all import *
>>> K = QuadraticField(Integer(3))
>>> QuaternionAlgebra(K, -Integer(1), -Integer(1)).ramified_places()
([],
 [Ring morphism:
    From: Number Field in a with defining polynomial x^2 - 3 with a = 1.
↪732050807568878?
    To:   Algebraic Real Field
    Defn: a |--> -1.732050807568878?,
  Ring morphism:
    From: Number Field in a with defining polynomial x^2 - 3 with a = 1.
↪732050807568878?
    To:   Algebraic Real Field
    Defn: a |--> 1.732050807568878?])
```

Extending the base field can also get rid of ramification at infinite places while still leaving some ramification at finite places:

```
sage: L = QuadraticField(-15)
sage: QuaternionAlgebra(L, -1, -1).ramified_places()
([Fractional ideal (2, 1/2*a + 1/2), Fractional ideal (2, 1/2*a - 1/2)], [])
```

```
>>> from sage.all import *
>>> L = QuadraticField(-Integer(15))
>>> QuaternionAlgebra(L, -Integer(1), -Integer(1)).ramified_places()
([Fractional ideal (2, 1/2*a + 1/2), Fractional ideal (2, 1/2*a - 1/2)], [])
```

We can use number field elements as invariants as well:

```
sage: x = polygen(ZZ, 'x')
sage: F.<a> = NumberField(x^2 - x - 1)
sage: QuaternionAlgebra(F, 2*a, F(-1)).ramified_places()
([Fractional ideal (2)],
 [Ring morphism:
    From: Number Field in a with defining polynomial x^2 - x - 1
    To: Algebraic Real Field
    Defn: a |--> -0.618033988749895?])
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> F = NumberField(x**Integer(2) - x - Integer(1), names=('a',)); (a,) = F._
↪first_ngens(1)
>>> QuaternionAlgebra(F, Integer(2)*a, F(-Integer(1))).ramified_places()
([Fractional ideal (2)],
 [Ring morphism:
    From: Number Field in a with defining polynomial x^2 - x - 1
    To: Algebraic Real Field
    Defn: a |--> -0.618033988749895?])
```

The method does not make sense over an arbitrary base ring:

```
sage: QuaternionAlgebra(RR(2.),1).ramified_places()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR(RealNumber('2.')),Integer(1)).ramified_places()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

ramified_primes()

Return the (finite) primes of the base number field at which this quaternion algebra ramifies.

OUTPUT:

The list of finite primes at which this quaternion algebra ramifies, given as

- elements of $\mathbf{Z}$ (sorted small to large) if the base field is $\mathbf{Q}$,
- integral fractional ideals of the base number field, otherwise.

EXAMPLES:

```
sage: QuaternionAlgebra(-58, -69).ramified_primes()
[3, 23, 29]
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(58), -Integer(69)).ramified_primes()
[3, 23, 29]
```

Under field extensions, the number of ramified primes can increase or decrease:

```
sage: K = QuadraticField(3)
sage: L = QuadraticField(-15)
sage: QuaternionAlgebra(-1, -1).ramified_primes()
[2]
sage: QuaternionAlgebra(K, -1, -1).ramified_primes()
[]
sage: QuaternionAlgebra(L, -1, -1).ramified_primes()
[Fractional ideal (2, 1/2*a + 1/2), Fractional ideal (2, 1/2*a - 1/2)]
```

```
>>> from sage.all import *
>>> K = QuadraticField(Integer(3))
>>> L = QuadraticField(-Integer(15))
>>> QuaternionAlgebra(-Integer(1), -Integer(1)).ramified_primes()
[2]
>>> QuaternionAlgebra(K, -Integer(1), -Integer(1)).ramified_primes()
[]
>>> QuaternionAlgebra(L, -Integer(1), -Integer(1)).ramified_primes()
[Fractional ideal (2, 1/2*a + 1/2), Fractional ideal (2, 1/2*a - 1/2)]
```

We can also use number field elements as invariants:

```
sage: x = polygen(ZZ, 'x')
sage: F.<a> = NumberField(x^2 - x - 1)
sage: QuaternionAlgebra(F, 2*a, F(-1)).ramified_primes()
[Fractional ideal (2)]
```

```
>>> from sage.all import *
>>> x = polygen(ZZ, 'x')
>>> F = NumberField(x**Integer(2) - x - Integer(1), names=('a',)); (a,) = F._
↪first_ngens(1)
>>> QuaternionAlgebra(F, Integer(2)*a, F(-Integer(1))).ramified_primes()
[Fractional ideal (2)]
```

The method does not make sense over an arbitrary base ring:

```
sage: QuaternionAlgebra(RR(2.),1).ramified_primes()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR(RealNumber('2.')),Integer(1)).ramified_primes()
Traceback (most recent call last):
...
ValueError: base field must be rational numbers or a number field
```

class sage.algebras.quatalg.quaternion_algebra.**QuaternionAlgebra_abstract**

Bases: `Parent`

basis()

Return the fixed basis of `self`, which is 1, i, j, k, where i, j, k are the generators of `self`.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ,-5,-2)
sage: Q.basis()
(1, i, j, k)

sage: Q.<xyz,abc,theta> = QuaternionAlgebra(GF(9,'a'),-5,-2)
sage: Q.basis()
(1, xyz, abc, theta)
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ,-Integer(5),-Integer(2), names=('i', 'j', 'k',));
↪ (i, j, k,) = Q._first_ngens(3)
>>> Q.basis()
(1, i, j, k)

>>> Q = QuaternionAlgebra(GF(Integer(9),'a'),-Integer(5),-Integer(2), names=(
↪'xyz', 'abc', 'theta',)); (xyz, abc, theta,) = Q._first_ngens(3)
>>> Q.basis()
(1, xyz, abc, theta)
```

The basis is cached:

```
sage: Q.basis() is Q.basis()
True
```

```
>>> from sage.all import *
>>> Q.basis() is Q.basis()
True
```

free_module()

Return the free module associated to `self` with inner product given by the reduced norm.

EXAMPLES:

```
sage: A.<t> = LaurentPolynomialRing(GF(3))
sage: B = QuaternionAlgebra(A, -1, t)
sage: B.free_module()
Ambient free quadratic module of rank 4 over the principal ideal domain
 Univariate Laurent Polynomial Ring in t over Finite Field of size 3
 Inner product matrix:
  [2 0 0 0]
  [0 2 0 0]
  [0 0 t 0]
  [0 0 0 t]
```

```
>>> from sage.all import *
>>> A = LaurentPolynomialRing(GF(Integer(3)), names=('t',)); (t,) = A._first_
↪ngens(1)
>>> B = QuaternionAlgebra(A, -Integer(1), t)
>>> B.free_module()
Ambient free quadratic module of rank 4 over the principal ideal domain
 Univariate Laurent Polynomial Ring in t over Finite Field of size 3
 Inner product matrix:
  [2 0 0 0]
  [0 2 0 0]
  [0 0 t 0]
  [0 0 0 t]
```

inner_product_matrix()

Return the inner product matrix associated to `self`.

This is the Gram matrix of the reduced norm as a quadratic form on `self`. The standard basis 1, i, j, k is orthogonal, so this matrix is just the diagonal matrix with diagonal entries 2, $2a$, $2b$, $2ab$.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-5,-19)
sage: Q.inner_product_matrix()
[  2   0   0   0]
[  0  10   0   0]
[  0   0  38   0]
[  0   0   0 190]
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(5),-Integer(19), names=('i', 'j', 'k',));␣
↪(i, j, k,) = Q._first_ngens(3)
>>> Q.inner_product_matrix()
[  2   0   0   0]
[  0  10   0   0]
[  0   0  38   0]
[  0   0   0 190]
```

is_commutative()

Return `False` always, since all quaternion algebras are noncommutative.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -3,-7)
sage: Q.is_commutative()
False
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ, -Integer(3),-Integer(7), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_commutative()
False
```

is_division_algebra()

Check whether this quaternion algebra is a division algebra, i.e. whether every nonzero element in it is invertible.

Currently only implemented for quaternion algebras defined over a number field.

EXAMPLES:

```
sage: QuaternionAlgebra(QQ,-5,-2).is_division_algebra()
True
sage: QuaternionAlgebra(2,9).is_division_algebra()
False
sage: K.<z> = QuadraticField(3)
sage: QuaternionAlgebra(K, 1+z, 3-z).is_division_algebra()
False
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QQ,-Integer(5),-Integer(2)).is_division_algebra()
True
>>> QuaternionAlgebra(Integer(2),Integer(9)).is_division_algebra()
False
```

(continues on next page)

(continued from previous page)

```
>>> K = QuadraticField(Integer(3), names=('z',)); (z,) = K._first_ngens(1)
>>> QuaternionAlgebra(K, Integer(1)+z, Integer(3)-z).is_division_algebra()
False
```

By checking ramification, the method correctly recognizes division quaternion algebras over a number field even if they have trivial discriminant:

```
sage: L = QuadraticField(5)
sage: A = QuaternionAlgebra(L, -1, -1)
sage: A.discriminant()
Fractional ideal (1)
sage: A.is_division_algebra()
True
```

```
>>> from sage.all import *
>>> L = QuadraticField(Integer(5))
>>> A = QuaternionAlgebra(L, -Integer(1), -Integer(1))
>>> A.discriminant()
Fractional ideal (1)
>>> A.is_division_algebra()
True
```

The method is not implemented over arbitrary base rings yet:

```
sage: QuaternionAlgebra(RR(2.),1).is_division_algebra()
Traceback (most recent call last):
...
NotImplementedError: base ring must be rational numbers or a number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR(RealNumber('2.')),Integer(1)).is_division_algebra()
Traceback (most recent call last):
...
NotImplementedError: base ring must be rational numbers or a number field
```

is_exact()

Return `True` if elements of this quaternion algebra are represented exactly, i.e. there is no precision loss when doing arithmetic. A quaternion algebra is exact if and only if its base field is exact.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -3, -7)
sage: Q.is_exact()
True
sage: Q.<i,j,k> = QuaternionAlgebra(Qp(7), -3, -7)
sage: Q.is_exact()
False
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ, -Integer(3), -Integer(7), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_exact()
```

(continues on next page)

(continued from previous page)

```
True
>>> Q = QuaternionAlgebra(Qp(Integer(7)), -Integer(3), -Integer(7), names=('i
↪', 'j', 'k',)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_exact()
False
```

is_field(*proof=True*)

Return `False` always, since all quaternion algebras are noncommutative and all fields are commutative.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -3, -7)
sage: Q.is_field()
False
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ, -Integer(3), -Integer(7), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_field()
False
```

is_finite()

Return `True` if the quaternion algebra is finite as a set.

Algorithm: A quaternion algebra is finite if and only if the base field is finite.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -3, -7)
sage: Q.is_finite()
False
sage: Q.<i,j,k> = QuaternionAlgebra(GF(5), -3, -7)
sage: Q.is_finite()
True
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ, -Integer(3), -Integer(7), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_finite()
False
>>> Q = QuaternionAlgebra(GF(Integer(5)), -Integer(3), -Integer(7), names=('i
↪', 'j', 'k',)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_finite()
True
```

is_integral_domain(*proof=True*)

Return `False` always, since all quaternion algebras are noncommutative and integral domains are commutative (in Sage).

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -3, -7)
sage: Q.is_integral_domain()
False
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ, -Integer(3), -Integer(7), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_integral_domain()
False
```

is_matrix_ring()

Check whether this quaternion algebra is isomorphic to the 2x2 matrix ring over the base ring.

Currently only implemented for quaternion algebras defined over a number field.

EXAMPLES:

```
sage: QuaternionAlgebra(QQ,-5,-2).is_matrix_ring()
False
sage: QuaternionAlgebra(2,9).is_matrix_ring()
True
sage: K.<z> = QuadraticField(3)
sage: QuaternionAlgebra(K, 1+z, 3-z).is_matrix_ring()
True
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(QQ,-Integer(5),-Integer(2)).is_matrix_ring()
False
>>> QuaternionAlgebra(Integer(2),Integer(9)).is_matrix_ring()
True
>>> K = QuadraticField(Integer(3), names=('z',)); (z,) = K._first_ngens(1)
>>> QuaternionAlgebra(K, Integer(1)+z, Integer(3)-z).is_matrix_ring()
True
```

By checking ramification, the method is able to recognize that quaternion algebras (defined over a number field) with trivial discriminant need not be matrix rings:

```
sage: L = QuadraticField(5)
sage: A = QuaternionAlgebra(L, -1, -1)
sage: A.discriminant()
Fractional ideal (1)
sage: A.is_matrix_ring()
False
```

```
>>> from sage.all import *
>>> L = QuadraticField(Integer(5))
>>> A = QuaternionAlgebra(L, -Integer(1), -Integer(1))
>>> A.discriminant()
Fractional ideal (1)
>>> A.is_matrix_ring()
False
```

The method is not implemented over arbitrary base rings yet:

```
sage: QuaternionAlgebra(RR(2.),1).is_matrix_ring()
Traceback (most recent call last):
...
NotImplementedError: base ring must be rational numbers or a number field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(RR(RealNumber('2.')),Integer(1)).is_matrix_ring()
Traceback (most recent call last):
...
NotImplementedError: base ring must be rational numbers or a number field
```

is_noetherian()

Return `True` always, since any quaternion algebra is a Noetherian ring (because it is a finitely generated module over a field).

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -3, -7)
sage: Q.is_noetherian()
True
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ, -Integer(3), -Integer(7), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3)
>>> Q.is_noetherian()
True
```

ngens()

Return the number of generators of the quaternion algebra as a K-vector space, not including 1.

This value is always 3: the algebra is spanned by the standard basis 1, i, j, k.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ,-5,-2)
sage: Q.ngens()
3
sage: Q.gens()
(i, j, k)
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ,-Integer(5),-Integer(2), names=('i', 'j', 'k',));
↪ (i, j, k,) = Q._first_ngens(3)
>>> Q.ngens()
3
>>> Q.gens()
(i, j, k)
```

order()

Return the number of elements of the quaternion algebra, or `+Infinity` if the algebra is not finite.

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(QQ, -3, -7)
sage: Q.order()
+Infinity
sage: Q.<i,j,k> = QuaternionAlgebra(GF(5), -3, -7)
sage: Q.order()
625
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(QQ, -Integer(3), -Integer(7), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3)
>>> Q.order()
+Infinity
>>> Q = QuaternionAlgebra(GF(Integer(5)), -Integer(3), -Integer(7), names=('i
↪', 'j', 'k',)); (i, j, k,) = Q._first_ngens(3)
>>> Q.order()
625
```

random_element(**args*, ***kwds*)

Return a random element of this quaternion algebra.

The `args` and `kwds` are passed to the `random_element` method of the base ring.

EXAMPLES:

```
sage: g = QuaternionAlgebra(QQ[sqrt(2)], -3, 7).random_element()             #␣
↪needs sage.symbolic
sage: g.parent() is QuaternionAlgebra(QQ[sqrt(2)], -3, 7)                    #␣
↪needs sage.symbolic
True
sage: g = QuaternionAlgebra(-3, 19).random_element()
sage: g.parent() is QuaternionAlgebra(-3, 19)
True
sage: g = QuaternionAlgebra(GF(17)(2), 3).random_element()
sage: g.parent() is QuaternionAlgebra(GF(17)(2), 3)
True
```

```
>>> from sage.all import *
>>> g = QuaternionAlgebra(QQ[sqrt(Integer(2))], -Integer(3), Integer(7)).
↪random_element()            # needs sage.symbolic
>>> g.parent() is QuaternionAlgebra(QQ[sqrt(Integer(2))], -Integer(3),␣
↪Integer(7))                   # needs sage.symbolic
True
>>> g = QuaternionAlgebra(-Integer(3), Integer(19)).random_element()
>>> g.parent() is QuaternionAlgebra(-Integer(3), Integer(19))
True
>>> g = QuaternionAlgebra(GF(Integer(17))(Integer(2)), Integer(3)).random_
↪element()
>>> g.parent() is QuaternionAlgebra(GF(Integer(17))(Integer(2)), Integer(3))
True
```

Specify the numerator and denominator bounds:

```
sage: g = QuaternionAlgebra(-3,19).random_element(10^6, 10^6)
sage: for h in g:
....:     assert h.numerator() in range(-10^6, 10^6 + 1)
....:     assert h.denominator() in range(10^6 + 1)

sage: g = QuaternionAlgebra(-3,19).random_element(5, 4)
sage: for h in g:
....:     assert h.numerator() in range(-5, 5 + 1)
....:     assert h.denominator() in range(4 + 1)
```

```
>>> from sage.all import *
>>> g = QuaternionAlgebra(-Integer(3),Integer(19)).random_
↪element(Integer(10)**Integer(6), Integer(10)**Integer(6))
>>> for h in g:
...     assert h.numerator() in range(-Integer(10)**Integer(6),␣
↪Integer(10)**Integer(6) + Integer(1))
...     assert h.denominator() in range(Integer(10)**Integer(6) + Integer(1))

>>> g = QuaternionAlgebra(-Integer(3),Integer(19)).random_element(Integer(5),␣
↪Integer(4))
>>> for h in g:
...     assert h.numerator() in range(-Integer(5), Integer(5) + Integer(1))
...     assert h.denominator() in range(Integer(4) + Integer(1))
```

vector_space()

Alias for *free_module()*.

EXAMPLES:

```
sage: QuaternionAlgebra(-3,19).vector_space()
Ambient quadratic space of dimension 4 over Rational Field
Inner product matrix:
  [   2    0    0    0]
  [   0    6    0    0]
  [   0    0  -38    0]
  [   0    0    0 -114]
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(3),Integer(19)).vector_space()
Ambient quadratic space of dimension 4 over Rational Field
Inner product matrix:
  [   2    0    0    0]
  [   0    6    0    0]
  [   0    0  -38    0]
  [   0    0    0 -114]
```

class sage.algebras.quatalg.quaternion_algebra.**QuaternionFractionalIdeal**(*ring*, *gens*, *coerce=True*, ***kwds*)

Bases: Ideal_fractional

class sage.algebras.quatalg.quaternion_algebra.**QuaternionFractionalIdeal_rational**(*Q*, *basis*, *left_order=None*, *right_order=None*, *check=True*)

Bases: *QuaternionFractionalIdeal*

A fractional ideal in a rational quaternion algebra.

INPUT:

- left_order – a quaternion order or None

- `right_order` – a quaternion order or `None`
- `basis` – tuple of length 4 of elements in of ambient quaternion algebra whose **Z**-span is an ideal
- `check` – boolean (default: `True`); if `False`, do no type checking.

basis()

Return a basis for this fractional ideal.

OUTPUT: tuple

EXAMPLES:

```
sage: QuaternionAlgebra(-11,-1).maximal_order().unit_ideal().basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().unit_ideal().
↪basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
```

basis_matrix()

Return basis matrix M in Hermite normal form for `self` as a matrix with rational entries.

If Q is the ambient quaternion algebra, then the **Z**-span of the rows of M viewed as linear combinations of Q.basis() = $[1, i, j, k]$ is the fractional ideal `self`. Also, `M * M.denominator()` is an integer matrix in Hermite normal form.

OUTPUT: matrix over **Q**

EXAMPLES:

```
sage: QuaternionAlgebra(-11,-1).maximal_order().unit_ideal().basis_matrix()
[1/2 1/2   0   0]
[  0   1   0   0]
[  0   0 1/2 1/2]
[  0   0   0   1]
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().unit_ideal().
↪basis_matrix()
[1/2 1/2   0   0]
[  0   1   0   0]
[  0   0 1/2 1/2]
[  0   0   0   1]
```

conjugate()

Return the ideal with generators the conjugates of the generators for `self`.

OUTPUT: a quaternionic fractional ideal

EXAMPLES:

```
sage: I = BrandtModule(3,5).right_ideals()[1]; I
Fractional ideal (8, 40*i, 6 + 28*i + 2*j, 4 + 18*i + 2*k)
sage: I.conjugate()
Fractional ideal (2 + 2*j + 28*k, 2*i + 4*j + 34*k, 8*j + 32*k, 40*k)
```

```
>>> from sage.all import *
>>> I = BrandtModule(Integer(3),Integer(5)).right_ideals()[Integer(1)]; I
Fractional ideal (8, 40*i, 6 + 28*i + 2*j, 4 + 18*i + 2*k)
>>> I.conjugate()
Fractional ideal (2 + 2*j + 28*k, 2*i + 4*j + 34*k, 8*j + 32*k, 40*k)
```

cyclic_right_subideals(*p*, *alpha=None*)

Let I = `self`. This function returns the right subideals J of I such that I/J is an $\mathbf{F}_p$-vector space of dimension 2.

INPUT:

- `p` – prime number (see below)
- `alpha` – (default: `None`) element of quaternion algebra, which can be used to parameterize the order of the ideals J. More precisely the J's are the right annihilators of $(1, 0)\alpha^i$ for $i = 0, 1, 2, ..., p$

OUTPUT: list of right ideals

Note

Currently, p must satisfy a bunch of conditions, or a `NotImplementedError` is raised. In particular, p must be odd and unramified in the quaternion algebra, must be coprime to the index of the right order in the maximal order, and also coprime to the normal of `self`. (The Brandt modules code has a more general algorithm in some cases.)

EXAMPLES:

```
sage: B = BrandtModule(2,37); I = B.right_ideals()[0]
sage: I.cyclic_right_subideals(3)
[Fractional ideal (12, 444*i, 8 + 404*i + 4*j, 2 + 150*i + 2*j + 2*k),
 Fractional ideal (12, 444*i, 4 + 256*i + 4*j, 10 + 150*i + 2*j + 2*k),
 Fractional ideal (12, 444*i, 8 + 256*i + 4*j, 6 + 298*i + 2*j + 2*k),
 Fractional ideal (12, 444*i, 4 + 404*i + 4*j, 6 + 2*i + 2*j + 2*k)]

sage: B = BrandtModule(5,389); I = B.right_ideals()[0]
sage: C = I.cyclic_right_subideals(3); C
[Fractional ideal (12, 4668*i, 10 + 3426*i + 2*j, 6 + 379*i + k),
 Fractional ideal (12, 4668*i, 2 + 3426*i + 2*j, 6 + 3491*i + k),
 Fractional ideal (12, 4 + 1556*i, 6 + 942*i + 6*j, 693*i + 2*j + k),
 Fractional ideal (12, 8 + 1556*i, 6 + 942*i + 6*j, 2 + 1007*i + 4*j + k)]
sage: [(I.free_module()/J.free_module()).invariants() for J in C]
[(3, 3), (3, 3), (3, 3), (3, 3)]
sage: I.scale(3).cyclic_right_subideals(3)
[Fractional ideal (36, 14004*i, 30 + 10278*i + 6*j, 18 + 1137*i + 3*k),
 Fractional ideal (36, 14004*i, 6 + 10278*i + 6*j, 18 + 10473*i + 3*k),
 Fractional ideal (36, 12 + 4668*i, 18 + 2826*i + 18*j, 2079*i + 6*j + 3*k),
 Fractional ideal (36, 24 + 4668*i, 18 + 2826*i + 18*j, 6 + 3021*i + 12*j +␣
↪3*k)]
sage: C = I.scale(1/9).cyclic_right_subideals(3); C
[Fractional ideal (4/3, 1556/3*i, 10/9 + 1142/3*i + 2/9*j, 2/3 + 379/9*i + 1/
↪9*k),
 Fractional ideal (4/3, 1556/3*i, 2/9 + 1142/3*i + 2/9*j, 2/3 + 3491/9*i + 1/
↪9*k),
```

(continues on next page)

(continued from previous page)

```
 Fractional ideal (4/3, 4/9 + 1556/9*i, 2/3 + 314/3*i + 2/3*j, 77*i + 2/9*j +␣
↪1/9*k),
 Fractional ideal (4/3, 8/9 + 1556/9*i, 2/3 + 314/3*i + 2/3*j, 2/9 + 1007/9*i␣
↪+ 4/9*j + 1/9*k)]
sage: [(I.scale(1/9).free_module()/J.free_module()).invariants() for J in C]
[(3, 3), (3, 3), (3, 3), (3, 3)]

sage: Q.<i,j,k> = QuaternionAlgebra(-2,-5)
sage: I = Q.ideal([Q(1),i,j,k])
sage: I.cyclic_right_subideals(3)
[Fractional ideal (3, 3*i, 2 + j, i + k),
 Fractional ideal (3, 3*i, 1 + j, 2*i + k),
 Fractional ideal (3, 2 + i, 3*j, 2*j + k),
 Fractional ideal (3, 1 + i, 3*j, j + k)]
```

```
>>> from sage.all import *
>>> B = BrandtModule(Integer(2),Integer(37)); I = B.right_ideals()[Integer(0)]
>>> I.cyclic_right_subideals(Integer(3))
[Fractional ideal (12, 444*i, 8 + 404*i + 4*j, 2 + 150*i + 2*j + 2*k),
 Fractional ideal (12, 444*i, 4 + 256*i + 4*j, 10 + 150*i + 2*j + 2*k),
 Fractional ideal (12, 444*i, 8 + 256*i + 4*j, 6 + 298*i + 2*j + 2*k),
 Fractional ideal (12, 444*i, 4 + 404*i + 4*j, 6 + 2*i + 2*j + 2*k)]

>>> B = BrandtModule(Integer(5),Integer(389)); I = B.right_
↪ideals()[Integer(0)]
>>> C = I.cyclic_right_subideals(Integer(3)); C
[Fractional ideal (12, 4668*i, 10 + 3426*i + 2*j, 6 + 379*i + k),
 Fractional ideal (12, 4668*i, 2 + 3426*i + 2*j, 6 + 3491*i + k),
 Fractional ideal (12, 4 + 1556*i, 6 + 942*i + 6*j, 693*i + 2*j + k),
 Fractional ideal (12, 8 + 1556*i, 6 + 942*i + 6*j, 2 + 1007*i + 4*j + k)]
>>> [(I.free_module()/J.free_module()).invariants() for J in C]
[(3, 3), (3, 3), (3, 3), (3, 3)]
>>> I.scale(Integer(3)).cyclic_right_subideals(Integer(3))
[Fractional ideal (36, 14004*i, 30 + 10278*i + 6*j, 18 + 1137*i + 3*k),
 Fractional ideal (36, 14004*i, 6 + 10278*i + 6*j, 18 + 10473*i + 3*k),
 Fractional ideal (36, 12 + 4668*i, 18 + 2826*i + 18*j, 2079*i + 6*j + 3*k),
 Fractional ideal (36, 24 + 4668*i, 18 + 2826*i + 18*j, 6 + 3021*i + 12*j +␣
↪3*k)]
>>> C = I.scale(Integer(1)/Integer(9)).cyclic_right_subideals(Integer(3)); C
[Fractional ideal (4/3, 1556/3*i, 10/9 + 1142/3*i + 2/9*j, 2/3 + 379/9*i + 1/
↪9*k),
 Fractional ideal (4/3, 1556/3*i, 2/9 + 1142/3*i + 2/9*j, 2/3 + 3491/9*i + 1/
↪9*k),
 Fractional ideal (4/3, 4/9 + 1556/9*i, 2/3 + 314/3*i + 2/3*j, 77*i + 2/9*j +␣
↪1/9*k),
 Fractional ideal (4/3, 8/9 + 1556/9*i, 2/3 + 314/3*i + 2/3*j, 2/9 + 1007/9*i␣
↪+ 4/9*j + 1/9*k)]
>>> [(I.scale(Integer(1)/Integer(9)).free_module()/J.free_module()).
↪invariants() for J in C]
[(3, 3), (3, 3), (3, 3), (3, 3)]

>>> Q = QuaternionAlgebra(-Integer(2),-Integer(5), names=('i', 'j', 'k',));␣
```

(continues on next page)

(continued from previous page)

```
↪(i, j, k,) = Q._first_ngens(3)
>>> I = Q.ideal([Q(Integer(1)),i,j,k])
>>> I.cyclic_right_subideals(Integer(3))
[Fractional ideal (3, 3*i, 2 + j, i + k),
 Fractional ideal (3, 3*i, 1 + j, 2*i + k),
 Fractional ideal (3, 2 + i, 3*j, 2*j + k),
 Fractional ideal (3, 1 + i, 3*j, j + k)]
```

The general algorithm is not yet implemented here:

```
sage: I.cyclic_right_subideals(3)[0].cyclic_right_subideals(3)
Traceback (most recent call last):
...
NotImplementedError: general algorithm not implemented
(The given basis vectors must be linearly independent.)
```

```
>>> from sage.all import *
>>> I.cyclic_right_subideals(Integer(3))[Integer(0)].cyclic_right_
↪subideals(Integer(3))
Traceback (most recent call last):
...
NotImplementedError: general algorithm not implemented
(The given basis vectors must be linearly independent.)
```

free_module()

Return the underlying free $\mathbf{Z}$-module corresponding to this ideal.

OUTPUT: free $\mathbf{Z}$-module of rank 4 embedded in an ambient $\mathbf{Q}^4$

EXAMPLES:

```
sage: X = BrandtModule(3,5).right_ideals()
sage: X[0]
Fractional ideal (4, 20*i, 2 + 8*i + 2*j, 18*i + 2*k)
sage: X[0].free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[ 2  0  2  8]
[ 0  2  0 18]
[ 0  0  4 16]
[ 0  0  0 20]
sage: X[0].scale(1/7).free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[ 2/7    0  2/7  8/7]
[   0  2/7    0 18/7]
[   0    0  4/7 16/7]
[   0    0    0 20/7]

sage: QuaternionAlgebra(-11,-1).maximal_order().unit_ideal().basis_matrix()
[1/2 1/2   0   0]
[  0   1   0   0]
[  0   0 1/2 1/2]
```

(continues on next page)

(continued from previous page)

```
[  0   0   0   1]
```

```
>>> from sage.all import *
>>> X = BrandtModule(Integer(3),Integer(5)).right_ideals()
>>> X[Integer(0)]
Fractional ideal (4, 20*i, 2 + 8*i + 2*j, 18*i + 2*k)
>>> X[Integer(0)].free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[ 2  0  2  8]
[ 0  2  0 18]
[ 0  0  4 16]
[ 0  0  0 20]
>>> X[Integer(0)].scale(Integer(1)/Integer(7)).free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[ 2/7    0  2/7  8/7]
[   0  2/7    0 18/7]
[   0    0  4/7 16/7]
[   0    0    0 20/7]

>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().unit_ideal().
↪basis_matrix()
[1/2 1/2   0   0]
[  0   1   0   0]
[  0   0 1/2 1/2]
[  0   0   0   1]
```

The free module method is also useful since it allows for checking if one ideal is contained in another, computing quotients I/J, etc.:

```
sage: X = BrandtModule(3,17).right_ideals()
sage: I = X[0].intersection(X[2]); I
Fractional ideal (2 + 2*j + 164*k, 2*i + 4*j + 46*k, 16*j + 224*k, 272*k)
sage: I.free_module().is_submodule(X[3].free_module())
False
sage: I.free_module().is_submodule(X[1].free_module())
True
sage: X[0].free_module() / I.free_module()
Finitely generated module V/W over Integer Ring with invariants (4, 4)
```

```
>>> from sage.all import *
>>> X = BrandtModule(Integer(3),Integer(17)).right_ideals()
>>> I = X[Integer(0)].intersection(X[Integer(2)]); I
Fractional ideal (2 + 2*j + 164*k, 2*i + 4*j + 46*k, 16*j + 224*k, 272*k)
>>> I.free_module().is_submodule(X[Integer(3)].free_module())
False
>>> I.free_module().is_submodule(X[Integer(1)].free_module())
True
>>> X[Integer(0)].free_module() / I.free_module()
Finitely generated module V/W over Integer Ring with invariants (4, 4)
```

This shows that the issue at Issue #6760 is fixed:

```
sage: R.<i,j,k> = QuaternionAlgebra(-1, -13)
sage: I = R.ideal([2+i, 3*i, 5*j, j+k]); I
Fractional ideal (2 + i, 3*i, j + k, 5*k)
sage: I.free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[2 1 0 0]
[0 3 0 0]
[0 0 1 1]
[0 0 0 5]
```

```
>>> from sage.all import *
>>> R = QuaternionAlgebra(-Integer(1), -Integer(13), names=('i', 'j', 'k',));
↪(i, j, k,) = R._first_ngens(3)
>>> I = R.ideal([Integer(2)+i, Integer(3)*i, Integer(5)*j, j+k]); I
Fractional ideal (2 + i, 3*i, j + k, 5*k)
>>> I.free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[2 1 0 0]
[0 3 0 0]
[0 0 1 1]
[0 0 0 5]
```

gram_matrix()

Return the Gram matrix of this fractional ideal.

OUTPUT: 4×4 matrix over $\mathbf{Q}$

EXAMPLES:

```
sage: I = BrandtModule(3,5).right_ideals()[1]; I
Fractional ideal (8, 40*i, 6 + 28*i + 2*j, 4 + 18*i + 2*k)
sage: I.gram_matrix()
[ 256    0  192  128]
[   0 6400 4480 2880]
[ 192 4480 3328 2112]
[ 128 2880 2112 1408]
```

```
>>> from sage.all import *
>>> I = BrandtModule(Integer(3),Integer(5)).right_ideals()[Integer(1)]; I
Fractional ideal (8, 40*i, 6 + 28*i + 2*j, 4 + 18*i + 2*k)
>>> I.gram_matrix()
[ 256    0  192  128]
[   0 6400 4480 2880]
[ 192 4480 3328 2112]
[ 128 2880 2112 1408]
```

intersection(J)

Return the intersection of the ideals `self` and J.

EXAMPLES:

```
sage: X = BrandtModule(3,5).right_ideals()
sage: I = X[0].intersection(X[1]); I
Fractional ideal (2 + 6*j + 4*k, 2*i + 4*j + 34*k, 8*j + 32*k, 40*k)
```

```
>>> from sage.all import *
>>> X = BrandtModule(Integer(3),Integer(5)).right_ideals()
>>> I = X[Integer(0)].intersection(X[Integer(1)]); I
Fractional ideal (2 + 6*j + 4*k, 2*i + 4*j + 34*k, 8*j + 32*k, 40*k)
```

is_equivalent(*J*, *B=10*, *certificate=False*, *side=None*)

Check whether `self` and `J` are equivalent as ideals. Tests equivalence as right ideals by default. Requires the underlying rational quaternion algebra to be definite.

INPUT:

- `J` – a fractional quaternion ideal with same order as `self`
- `B` – a bound to compute and compare theta series before doing the full equivalence test
- `certificate` – if `True` returns an element alpha such that alpha*J = I or J*alpha = I for right and left ideals respectively
- `side` – if `'left'` performs left equivalence test. If `'right' ``or ``None` performs right ideal equivalence test

OUTPUT: boolean, or (boolean, alpha) if `certificate` is `True`

EXAMPLES:

```
sage: R = BrandtModule(3,5).right_ideals(); len(R)
2
sage: OO = R[0].left_order()
sage: S = OO.right_ideal([3*a for a in R[0].basis()])
sage: R[0].is_equivalent(S)
doctest:...: DeprecationWarning: is_equivalent is deprecated,
please use is_left_equivalent or is_right_equivalent
accordingly instead
See https://github.com/sagemath/sage/issues/37100 for details.
True
```

```
>>> from sage.all import *
>>> R = BrandtModule(Integer(3),Integer(5)).right_ideals(); len(R)
2
>>> OO = R[Integer(0)].left_order()
>>> S = OO.right_ideal([Integer(3)*a for a in R[Integer(0)].basis()])
>>> R[Integer(0)].is_equivalent(S)
doctest:...: DeprecationWarning: is_equivalent is deprecated,
please use is_left_equivalent or is_right_equivalent
accordingly instead
See https://github.com/sagemath/sage/issues/37100 for details.
True
```

is_integral()

Check whether the quaternion fractional ideal `self` is integral.

An ideal in a quaternion algebra is integral if and only if it is contained in its left order. If the left order is already defined this method just checks this definition, otherwise it uses one of the alternative definitions from Lemma 16.2.8 of [Voi2021].

EXAMPLES:

```
sage: R.<i,j,k> = QuaternionAlgebra(QQ, -1,-11)
sage: I = R.ideal([2 + 2*j + 140*k, 2*i + 4*j + 150*k, 8*j + 104*k, 152*k])
sage: I.is_integral()
True
sage: O = I.left_order()
sage: I.is_integral()
True
sage: I = R.ideal([1/2 + 2*j + 140*k, 2*i + 4*j + 150*k, 8*j + 104*k, 152*k])
sage: I.is_integral()
False
```

```
>>> from sage.all import *
>>> R = QuaternionAlgebra(QQ, -Integer(1),-Integer(11), names=('i', 'j', 'k',
↪)); (i, j, k,) = R._first_ngens(3)
>>> I = R.ideal([Integer(2) + Integer(2)*j + Integer(140)*k, Integer(2)*i +␣
↪Integer(4)*j + Integer(150)*k, Integer(8)*j + Integer(104)*k,␣
↪Integer(152)*k])
>>> I.is_integral()
True
>>> O = I.left_order()
>>> I.is_integral()
True
>>> I = R.ideal([Integer(1)/Integer(2) + Integer(2)*j + Integer(140)*k,␣
↪Integer(2)*i + Integer(4)*j + Integer(150)*k, Integer(8)*j + Integer(104)*k,
↪ Integer(152)*k])
>>> I.is_integral()
False
```

is_left_equivalent(*J*, *B=10*, *certificate=False*)

Check whether `self` and `J` are equivalent as left ideals. Requires the underlying rational quaternion algebra to be definite.

INPUT:

- `J` – a fractional quaternion left ideal with same order as `self`
- `B` – a bound to compute and compare theta series before doing the full equivalence test
- `certificate` – if `True` returns an element alpha such that J*alpha=I

OUTPUT: boolean, or (boolean, alpha) if `certificate` is `True`

is_primitive()

Check whether the quaternion fractional ideal `self` is primitive.

An integral left $\mathcal{O}$-ideal for some order $\mathcal{O}$ is called primitive if for all integers $n > 1$ it is not contained in $n\mathcal{O}$.

EXAMPLES:

```
sage: A.<i,j,k> = QuaternionAlgebra(QQ, -1,-11)
sage: I = A.ideal([1/2 + 1/2*i + 1/2*j + 3/2*k, i + k, j + k, 2*k])
sage: I.is_primitive()
```

(continues on next page)

(continued from previous page)

```
True
sage: (2*I).is_primitive()
False
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(QQ, -Integer(1),-Integer(11), names=('i', 'j', 'k',
↪)); (i, j, k,) = A._first_ngens(3)
>>> I = A.ideal([Integer(1)/Integer(2) + Integer(1)/Integer(2)*i + Integer(1)/
↪Integer(2)*j + Integer(3)/Integer(2)*k, i + k, j + k, Integer(2)*k])
>>> I.is_primitive()
True
>>> (Integer(2)*I).is_primitive()
False
```

is_principal(*certificate=False*)

Check whether `self` is principal as a full rank quaternion ideal. Requires the underlying quaternion algebra to be definite. Independent of whether `self` is a left or a right ideal.

INPUT:

- `certificate` – if `True` returns a generator alpha s.t. $I = \alpha O$ where O is the right order of I

OUTPUT: boolean, or (boolean, alpha) if `certificate` is `True`

EXAMPLES:

```
sage: B.<i,j,k> = QuaternionAlgebra(419)
sage: O = B.quaternion_order([1/2 + 3/2*j, 1/6*i + 2/3*j + 1/2*k, 3*j, k])
sage: beta = O.random_element()
sage: while beta.is_zero():
....:     beta = O.random_element()
sage: I = O*beta
sage: bool, alpha = I.is_principal(True)
sage: bool
True
sage: I == O*alpha
True
```

```
>>> from sage.all import *
>>> B = QuaternionAlgebra(Integer(419), names=('i', 'j', 'k',)); (i, j, k,) =␣
↪B._first_ngens(3)
>>> O = B.quaternion_order([Integer(1)/Integer(2) + Integer(3)/Integer(2)*j,␣
↪Integer(1)/Integer(6)*i + Integer(2)/Integer(3)*j + Integer(1)/Integer(2)*k,
↪ Integer(3)*j, k])
>>> beta = O.random_element()
>>> while beta.is_zero():
...     beta = O.random_element()
>>> I = O*beta
>>> bool, alpha = I.is_principal(True)
>>> bool
True
>>> I == O*alpha
True
```

is_right_equivalent(*J*, *B=10*, *certificate=False*)

Check whether `self` and `J` are equivalent as right ideals. Requires the underlying rational quaternion algebra to be definite.

INPUT:

- `J` – a fractional quaternion right ideal with same order as `self`
- `B` – a bound to compute and compare theta series before doing the full equivalence test
- `certificate` – if `True` returns an element alpha such that alpha*J=I

OUTPUT: boolean, or (boolean, alpha) if `certificate` is `True`

EXAMPLES:

```
sage: R = BrandtModule(3,5).right_ideals(); len(R)
2
sage: R[0].is_right_equivalent(R[1])
False

sage: R[0].is_right_equivalent(R[0])
True
sage: OO = R[0].left_order()
sage: S = OO.right_ideal([3*a for a in R[0].basis()])
sage: R[0].is_right_equivalent(S, certificate=True)
(True, 1/3)
sage: -1/3*S == R[0]
True

sage: B = QuaternionAlgebra(101)
sage: i,j,k = B.gens()
sage: I = B.maximal_order().unit_ideal()
sage: beta = B.random_element()
sage: while beta.is_zero():
....:     beta = B.random_element()
sage: J = beta*I
sage: bool, alpha = I.is_right_equivalent(J, certificate=True)
sage: bool
True
sage: alpha*J == I
True
```

```
>>> from sage.all import *
>>> R = BrandtModule(Integer(3),Integer(5)).right_ideals(); len(R)
2
>>> R[Integer(0)].is_right_equivalent(R[Integer(1)])
False

>>> R[Integer(0)].is_right_equivalent(R[Integer(0)])
True
>>> OO = R[Integer(0)].left_order()
>>> S = OO.right_ideal([Integer(3)*a for a in R[Integer(0)].basis()])
>>> R[Integer(0)].is_right_equivalent(S, certificate=True)
(True, 1/3)
>>> -Integer(1)/Integer(3)*S == R[Integer(0)]
```

(continues on next page)

(continued from previous page)

```
True

>>> B = QuaternionAlgebra(Integer(101))
>>> i,j,k = B.gens()
>>> I = B.maximal_order().unit_ideal()
>>> beta = B.random_element()
>>> while beta.is_zero():
...       beta = B.random_element()
>>> J = beta*I
>>> bool, alpha = I.is_right_equivalent(J, certificate=True)
>>> bool
True
>>> alpha*J == I
True
```

left_order()

Return the left order associated to this fractional ideal.

OUTPUT: an order in a quaternion algebra

EXAMPLES:

```
sage: B = BrandtModule(11)
sage: R = B.maximal_order()
sage: I = R.unit_ideal()
sage: I.left_order()
Order of Quaternion Algebra (-1, -11) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)
```

```
>>> from sage.all import *
>>> B = BrandtModule(Integer(11))
>>> R = B.maximal_order()
>>> I = R.unit_ideal()
>>> I.left_order()
Order of Quaternion Algebra (-1, -11) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)
```

We do a consistency check:

```
sage: B = BrandtModule(11,19); R = B.right_ideals()
sage: [r.left_order().discriminant() for r in R]
[209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209,␣
↪209, 209, 209]
```

```
>>> from sage.all import *
>>> B = BrandtModule(Integer(11),Integer(19)); R = B.right_ideals()
>>> [r.left_order().discriminant() for r in R]
[209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209, 209,␣
↪209, 209, 209]
```

minimal_element()

Return an element in this quaternion ideal of minimal norm.

If the ideal is a principal lattice, this method can be used to find a generator; see [Piz1980], Corollary 1.20.

EXAMPLES:

```
sage: Quat.<i,j,k> = QuaternionAlgebra(-3,-101)
sage: O = Quat.maximal_order(); O
Order of Quaternion Algebra (-3, -101) with base ring Rational Field with␣
↪basis (1/2 + 1/2*i, 1/2*j - 1/2*k, -1/3*i + 1/3*k, -k)
sage: (O * 5).minimal_element()
5
sage: alpha = 1/2 + 1/6*i + j + 55/3*k
sage: I = O*141 + O*alpha; I.norm()
141
sage: el = I.minimal_element(); el
13/2 - 7/6*i + j + 2/3*k
sage: el.reduced_norm()
282
```

```
>>> from sage.all import *
>>> Quat = QuaternionAlgebra(-Integer(3),-Integer(101), names=('i', 'j', 'k',
↪)); (i, j, k,) = Quat._first_ngens(3)
>>> O = Quat.maximal_order(); O
Order of Quaternion Algebra (-3, -101) with base ring Rational Field with␣
↪basis (1/2 + 1/2*i, 1/2*j - 1/2*k, -1/3*i + 1/3*k, -k)
>>> (O * Integer(5)).minimal_element()
5
>>> alpha = Integer(1)/Integer(2) + Integer(1)/Integer(6)*i + j + Integer(55)/
↪Integer(3)*k
>>> I = O*Integer(141) + O*alpha; I.norm()
141
>>> el = I.minimal_element(); el
13/2 - 7/6*i + j + 2/3*k
>>> el.reduced_norm()
282
```

multiply_by_conjugate(J)

Return product of `self` and the conjugate Jbar of J.

INPUT:

- `J` – a quaternion ideal

OUTPUT: a quaternionic fractional ideal

EXAMPLES:

```
sage: R = BrandtModule(3,5).right_ideals()
sage: R[0].multiply_by_conjugate(R[1])
Fractional ideal (32, 160*i, 8 + 112*i + 8*j, 16 + 72*i + 8*k)
sage: R[0]*R[1].conjugate()
Fractional ideal (32, 160*i, 8 + 112*i + 8*j, 16 + 72*i + 8*k)
```

```
>>> from sage.all import *
>>> R = BrandtModule(Integer(3),Integer(5)).right_ideals()
>>> R[Integer(0)].multiply_by_conjugate(R[Integer(1)])
Fractional ideal (32, 160*i, 8 + 112*i + 8*j, 16 + 72*i + 8*k)
```

(continues on next page)

(continued from previous page)

```
>>> R[Integer(0)]*R[Integer(1)].conjugate()
Fractional ideal (32, 160*i, 8 + 112*i + 8*j, 16 + 72*i + 8*k)
```

norm()

Return the reduced norm of this fractional ideal.

OUTPUT: rational number

EXAMPLES:

```
sage: M = BrandtModule(37)
sage: C = M.right_ideals()
sage: [I.norm() for I in C]
[16, 32, 32]

sage: # optional - magma
sage: (a,b) = M.quaternion_algebra().invariants()
sage: magma.eval('A<i,j,k> := QuaternionAlgebra<Rationals() | %s, %s>' % (a,
↪b))
''
sage: magma.eval('O := QuaternionOrder(%s)' % str(list(C[0].right_order().
↪basis())))
''
sage: [ magma('rideal<O | %s>' % str(list(I.basis()))).Norm() for I in C]
[16, 32, 32]

sage: A.<i,j,k> = QuaternionAlgebra(-1,-1)
sage: R = A.ideal([i,j,k,1/2 + 1/2*i + 1/2*j + 1/2*k])      # this is␣
↪actually an order, so has reduced norm 1
sage: R.norm()
1
sage: [ J.norm() for J in R.cyclic_right_subideals(3) ]      # enumerate␣
↪maximal right R-ideals of reduced norm 3, verify their norms
[3, 3, 3, 3]
```

```
>>> from sage.all import *
>>> M = BrandtModule(Integer(37))
>>> C = M.right_ideals()
>>> [I.norm() for I in C]
[16, 32, 32]

>>> # optional - magma
>>> (a,b) = M.quaternion_algebra().invariants()
>>> magma.eval('A<i,j,k> := QuaternionAlgebra<Rationals() | %s, %s>' % (a,b))
''
>>> magma.eval('O := QuaternionOrder(%s)' % str(list(C[Integer(0)].right_
↪order().basis())))
''
>>> [ magma('rideal<O | %s>' % str(list(I.basis()))).Norm() for I in C]
[16, 32, 32]

>>> A = QuaternionAlgebra(-Integer(1),-Integer(1), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
```

(continues on next page)

(continued from previous page)

```
>>> R = A.ideal([i,j,k,Integer(1)/Integer(2) + Integer(1)/Integer(2)*i +␣
↪Integer(1)/Integer(2)*j + Integer(1)/Integer(2)*k])      # this is actually␣
↪an order, so has reduced norm 1
>>> R.norm()
1
>>> [ J.norm() for J in R.cyclic_right_subideals(Integer(3)) ]      #␣
↪enumerate maximal right R-ideals of reduced norm 3, verify their norms
[3, 3, 3, 3]
```

primitive_decomposition()

Let I = `self`. If I is an integral left $\mathcal{O}$-ideal return its decomposition as an equivalent primitive ideal and an integer such that their product is the initial ideal.

OUTPUTS: A primitive ideal equivalent to I, i.e. an equivalent ideal not contained in $n\mathcal{O}$ for any $n > 0$, and the smallest integer g such that $I \subset g\mathcal{O}$.

EXAMPLES:

```
sage: A.<i,j,k> = QuaternionAlgebra(QQ, -1,-11)
sage: I = A.ideal([1/2 + 1/2*i + 1/2*j + 3/2*k, i + k, j + k, 2*k])
sage: I.primitive_decomposition()
(Fractional ideal (1/2 + 1/2*i + 1/2*j + 3/2*k, i + k, j + k, 2*k), 1)
sage: J = A.ideal([7/2 + 7/2*i + 49/2*j + 91/2*k, 7*i + 21*k, 35*j + 35*k,␣
↪70*k])
sage: Jequiv, g = J.primitive_decomposition()
sage: Jequiv*g == J
True
sage: Jequiv, g
(Fractional ideal (10, 5 + 5*i, 3 + j, 13/2 + 7/2*i + 1/2*j + 1/2*k), 7)
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(QQ, -Integer(1),-Integer(11), names=('i', 'j', 'k',
↪)); (i, j, k,) = A._first_ngens(3)
>>> I = A.ideal([Integer(1)/Integer(2) + Integer(1)/Integer(2)*i + Integer(1)/
↪Integer(2)*j + Integer(3)/Integer(2)*k, i + k, j + k, Integer(2)*k])
>>> I.primitive_decomposition()
(Fractional ideal (1/2 + 1/2*i + 1/2*j + 3/2*k, i + k, j + k, 2*k), 1)
>>> J = A.ideal([Integer(7)/Integer(2) + Integer(7)/Integer(2)*i +␣
↪Integer(49)/Integer(2)*j + Integer(91)/Integer(2)*k, Integer(7)*i +␣
↪Integer(21)*k, Integer(35)*j + Integer(35)*k, Integer(70)*k])
>>> Jequiv, g = J.primitive_decomposition()
>>> Jequiv*g == J
True
>>> Jequiv, g
(Fractional ideal (10, 5 + 5*i, 3 + j, 13/2 + 7/2*i + 1/2*j + 1/2*k), 7)
```

pullback(J, *side=None*)

Compute the ideal which is the pullback of `self` through an ideal `J`.

Uses Lemma 2.1.7 of [Ler2022]. Only works for integral ideals.

INPUT:

- `J` – a fractional quaternion ideal with norm coprime to `self` and either left order equal to the right order of `self`, or vice versa

- `side` – string (default: `None`); set to `'left'` or `'right'` to perform pullback of left or right ideals respectively. If `None` the side is determined by the matching left and right orders

OUTPUT: a fractional quaternion ideal

EXAMPLES:

```
sage: B = QuaternionAlgebra(419)
sage: i,j,k = B.gens()
sage: I1 = B.ideal([1/2 + 3/2*j + 2*k, 1/2*i + j + 3/2*k, 3*j, 3*k])
sage: I2 = B.ideal([1/2 + 9/2*j, 1/2*i + 9/2*k, 5*j, 5*k])
sage: I3 = I1.pushforward(I2, side='left')
sage: I3.left_order() == I2.right_order()
True
sage: I3.pullback(I2, side='left') == I1
True
```

```
>>> from sage.all import *
>>> B = QuaternionAlgebra(Integer(419))
>>> i,j,k = B.gens()
>>> I1 = B.ideal([Integer(1)/Integer(2) + Integer(3)/Integer(2)*j +␣
↪Integer(2)*k, Integer(1)/Integer(2)*i + j + Integer(3)/Integer(2)*k,␣
↪Integer(3)*j, Integer(3)*k])
>>> I2 = B.ideal([Integer(1)/Integer(2) + Integer(9)/Integer(2)*j, Integer(1)/
↪Integer(2)*i + Integer(9)/Integer(2)*k, Integer(5)*j, Integer(5)*k])
>>> I3 = I1.pushforward(I2, side='left')
>>> I3.left_order() == I2.right_order()
True
>>> I3.pullback(I2, side='left') == I1
True
```

pushforward(*J*, *side=None*)

Compute the ideal which is the pushforward of `self` through an ideal `J`.

Uses Lemma 2.1.7 of [Ler2022]. Only works for integral ideals.

INPUT:

- `J` – a fractional quaternion ideal with norm coprime to `self` and either the same left order or right order as `self`
- `side` – string (default: `None`); set to `'left'` or `'right'` to perform pushforward of left or right ideals respectively. If `None` the side is determined by the matching left or right orders

OUTPUT: a fractional quaternion ideal

EXAMPLES:

```
sage: B = QuaternionAlgebra(419)
sage: i,j,k = B.gens()
sage: I1 = B.ideal([1/2 + 3/2*j + 2*k, 1/2*i + j + 3/2*k, 3*j, 3*k])
sage: I2 = B.ideal([1/2 + 9/2*j, 1/2*i + 9/2*k, 5*j, 5*k])
sage: I1.left_order() == I2.left_order()
True
sage: I1.pushforward(I2, side='left')
Fractional ideal (3, 15*i, 3/2 + 10*i + 1/2*j, 1 + 9/10*i + 1/10*k)
```

```
>>> from sage.all import *
>>> B = QuaternionAlgebra(Integer(419))
>>> i,j,k = B.gens()
>>> I1 = B.ideal([Integer(1)/Integer(2) + Integer(3)/Integer(2)*j +␣
↪Integer(2)*k, Integer(1)/Integer(2)*i + j + Integer(3)/Integer(2)*k,␣
↪Integer(3)*j, Integer(3)*k])
>>> I2 = B.ideal([Integer(1)/Integer(2) + Integer(9)/Integer(2)*j, Integer(1)/
↪Integer(2)*i + Integer(9)/Integer(2)*k, Integer(5)*j, Integer(5)*k])
>>> I1.left_order() == I2.left_order()
True
>>> I1.pushforward(I2, side='left')
Fractional ideal (3, 15*i, 3/2 + 10*i + 1/2*j, 1 + 9/10*i + 1/10*k)
```

quadratic_form()

Return the normalized quadratic form associated to this quaternion ideal.

OUTPUT: quadratic form

EXAMPLES:

```
sage: I = BrandtModule(11).right_ideals()[1]
sage: Q = I.quadratic_form(); Q
Quadratic form in 4 variables over Rational Field with coefficients:
[ 2 0 3 2 ]
[ * 2 2 1 ]
[ * * 3 2 ]
[ * * * 2 ]
sage: Q.theta_series(10)
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + 36*q^8 + 36*q^9 +␣
↪O(q^10)
sage: I.theta_series(10)
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + 36*q^8 + 36*q^9 +␣
↪O(q^10)
```

```
>>> from sage.all import *
>>> I = BrandtModule(Integer(11)).right_ideals()[Integer(1)]
>>> Q = I.quadratic_form(); Q
Quadratic form in 4 variables over Rational Field with coefficients:
[ 2 0 3 2 ]
[ * 2 2 1 ]
[ * * 3 2 ]
[ * * * 2 ]
>>> Q.theta_series(Integer(10))
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + 36*q^8 + 36*q^9 +␣
↪O(q^10)
>>> I.theta_series(Integer(10))
1 + 12*q^2 + 12*q^3 + 12*q^4 + 12*q^5 + 24*q^6 + 24*q^7 + 36*q^8 + 36*q^9 +␣
↪O(q^10)
```

quaternion_algebra()

Return the ambient quaternion algebra that contains this fractional ideal.

This is an alias for $self.ring()$.

EXAMPLES:

```
sage: I = BrandtModule(3, 5).right_ideals()[1]; I
Fractional ideal (8, 40*i, 6 + 28*i + 2*j, 4 + 18*i + 2*k)
sage: I.quaternion_algebra()
Quaternion Algebra (-1, -3) with base ring Rational Field
```

```
>>> from sage.all import *
>>> I = BrandtModule(Integer(3), Integer(5)).right_ideals()[Integer(1)]; I
Fractional ideal (8, 40*i, 6 + 28*i + 2*j, 4 + 18*i + 2*k)
>>> I.quaternion_algebra()
Quaternion Algebra (-1, -3) with base ring Rational Field
```

random_element(**args*, ***kwds*)

Return a random element in the rational fractional ideal `self`.

EXAMPLES:

```
sage: B.<i,j,k> = QuaternionAlgebra(211)
sage: I = B.ideal([1, 1/4*j, 20*(i+k), 2/3*i])
sage: I.random_element() in I
True
```

```
>>> from sage.all import *
>>> B = QuaternionAlgebra(Integer(211), names=('i', 'j', 'k',)); (i, j, k,) =␣
↪B._first_ngens(3)
>>> I = B.ideal([Integer(1), Integer(1)/Integer(4)*j, Integer(20)*(i+k),␣
↪Integer(2)/Integer(3)*i])
>>> I.random_element() in I
True
```

reduced_basis()

Let $I =$ `self` be a fractional ideal in a (rational) definite quaternion algebra. This function returns an LLL reduced basis of I.

OUTPUT: a tuple of four elements in I forming an LLL reduced basis of I as a lattice

EXAMPLES:

```
sage: B = BrandtModule(2,37); I = B.right_ideals()[0]
sage: I
Fractional ideal (4, 148*i, 108*i + 4*j, 2 + 2*i + 2*j + 2*k)
sage: I.reduced_basis()
(4, 2 + 2*i + 2*j + 2*k, 2 - 14*i + 14*j - 2*k, 2 - 2*i - 14*j + 14*k)
sage: l = I.reduced_basis()
sage: assert all(l[i].reduced_norm() <= l[i+1].reduced_norm() for i in␣
↪range(len(l) - 1))

sage: B = QuaternionAlgebra(next_prime(2**50))
sage: O = B.maximal_order()
sage: i,j,k = B.gens()
sage: alpha = 1/2 - 1/2*i + 3/2*j - 7/2*k
sage: I = O*alpha + O*3089622859
sage: I.reduced_basis()[0]
1/2*i + j + 5/2*k
```

```
>>> from sage.all import *
>>> B = BrandtModule(Integer(2),Integer(37)); I = B.right_ideals()[Integer(0)]
>>> I
Fractional ideal (4, 148*i, 108*i + 4*j, 2 + 2*i + 2*j + 2*k)
>>> I.reduced_basis()
(4, 2 + 2*i + 2*j + 2*k, 2 - 14*i + 14*j - 2*k, 2 - 2*i - 14*j + 14*k)
>>> l = I.reduced_basis()
>>> assert all(l[i].reduced_norm() <= l[i+Integer(1)].reduced_norm() for i in␣
↪range(len(l) - Integer(1)))

>>> B = QuaternionAlgebra(next_prime(Integer(2)**Integer(50)))
>>> O = B.maximal_order()
>>> i,j,k = B.gens()
>>> alpha = Integer(1)/Integer(2) - Integer(1)/Integer(2)*i + Integer(3)/
↪Integer(2)*j - Integer(7)/Integer(2)*k
>>> I = O*alpha + O*Integer(3089622859)
>>> I.reduced_basis()[Integer(0)]
1/2*i + j + 5/2*k
```

right_order()

Return the right order associated to this fractional ideal.

OUTPUT: an order in a quaternion algebra

EXAMPLES:

```
sage: I = BrandtModule(389).right_ideals()[1]; I
Fractional ideal (8, 8*i, 2 + 6*i + 2*j, 6 + 3*i + k)
sage: I.right_order()
Order of Quaternion Algebra (-2, -389) with base ring Rational Field
 with basis (1/2 + 1/2*j + 1/2*k, 1/4*i + 1/2*j + 1/4*k, j, k)
sage: I.left_order()
Order of Quaternion Algebra (-2, -389) with base ring Rational Field
 with basis (1/2 + 1/2*j + 3/2*k, 1/8*i + 1/4*j + 9/8*k, j + k, 2*k)
```

```
>>> from sage.all import *
>>> I = BrandtModule(Integer(389)).right_ideals()[Integer(1)]; I
Fractional ideal (8, 8*i, 2 + 6*i + 2*j, 6 + 3*i + k)
>>> I.right_order()
Order of Quaternion Algebra (-2, -389) with base ring Rational Field
 with basis (1/2 + 1/2*j + 1/2*k, 1/4*i + 1/2*j + 1/4*k, j, k)
>>> I.left_order()
Order of Quaternion Algebra (-2, -389) with base ring Rational Field
 with basis (1/2 + 1/2*j + 3/2*k, 1/8*i + 1/4*j + 9/8*k, j + k, 2*k)
```

The following is a big consistency check. We take reps for all the right ideal classes of a certain order, take the corresponding left orders, then take ideals in the left orders and from those compute the right order again:

```
sage: B = BrandtModule(11,19); R = B.right_ideals()
sage: O = [r.left_order() for r in R]
sage: J = [O[i].left_ideal(R[i].basis()) for i in range(len(R))]
sage: len(set(J))
18
sage: len(set([I.right_order() for I in J]))
```

(continues on next page)

(continued from previous page)

```
1
sage: J[0].right_order() == B.order_of_level_N()
True
```

```
>>> from sage.all import *
>>> B = BrandtModule(Integer(11),Integer(19)); R = B.right_ideals()
>>> O = [r.left_order() for r in R]
>>> J = [O[i].left_ideal(R[i].basis()) for i in range(len(R))]
>>> len(set(J))
18
>>> len(set([I.right_order() for I in J]))
1
>>> J[Integer(0)].right_order() == B.order_of_level_N()
True
```

scale(*alpha*, *left=False*)

Scale the fractional ideal `self` by multiplying the basis by `alpha`.

INPUT:

- α – nonzero element of quaternion algebra
- `left` – boolean (default: `False`); if `True` multiply α on the left, otherwise multiply α on the right

OUTPUT: a new fractional ideal

EXAMPLES:

```
sage: B = BrandtModule(5,37); I = B.right_ideals()[0]
sage: i,j,k = B.quaternion_algebra().gens(); I
Fractional ideal (4, 148*i, 2 + 106*i + 2*j, 2 + 147*i + k)
sage: I.scale(i)
Fractional ideal (296, 4*i, 2 + 2*i + 2*j, 212 + 2*i + 2*k)
sage: I.scale(i, left=True)
Fractional ideal (296, 4*i, 294 + 2*i + 2*j, 84 + 2*i + 2*k)
sage: I.scale(i, left=False)
Fractional ideal (296, 4*i, 2 + 2*i + 2*j, 212 + 2*i + 2*k)
sage: i * I.gens()[0]
4*i
sage: I.gens()[0] * i
4*i
```

```
>>> from sage.all import *
>>> B = BrandtModule(Integer(5),Integer(37)); I = B.right_ideals()[Integer(0)]
>>> i,j,k = B.quaternion_algebra().gens(); I
Fractional ideal (4, 148*i, 2 + 106*i + 2*j, 2 + 147*i + k)
>>> I.scale(i)
Fractional ideal (296, 4*i, 2 + 2*i + 2*j, 212 + 2*i + 2*k)
>>> I.scale(i, left=True)
Fractional ideal (296, 4*i, 294 + 2*i + 2*j, 84 + 2*i + 2*k)
>>> I.scale(i, left=False)
Fractional ideal (296, 4*i, 2 + 2*i + 2*j, 212 + 2*i + 2*k)
>>> i * I.gens()[Integer(0)]
4*i
```

(continues on next page)

(continued from previous page)

```
>>> I.gens()[Integer(0)] * i
4*i
```

The scaling element must be nonzero:

```
sage: B.<i,j,k> = QuaternionAlgebra(419)
sage: O = B.quaternion_order([1/2 + 3/2*j, 1/6*i + 2/3*j + 1/2*k, 3*j, k])
sage: O * O.zero()
Traceback (most recent call last):
...
ValueError: the scaling factor must be nonzero
```

```
>>> from sage.all import *
>>> B = QuaternionAlgebra(Integer(419), names=('i', 'j', 'k',)); (i, j, k,) =␣
↪B._first_ngens(3)
>>> O = B.quaternion_order([Integer(1)/Integer(2) + Integer(3)/Integer(2)*j,␣
↪Integer(1)/Integer(6)*i + Integer(2)/Integer(3)*j + Integer(1)/Integer(2)*k,
↪ Integer(3)*j, k])
>>> O * O.zero()
Traceback (most recent call last):
...
ValueError: the scaling factor must be nonzero
```

theta_series(*B*, *var='q'*)

Return normalized theta series of `self`, as a power series over **Z** in the variable `var`, which is 'q' by default.

The normalized theta series is by definition

$$\theta_I(q) = \sum_{x\in I} q^{\frac{N(x)}{N(I)}}.$$

INPUT:

- `B` – positive integer
- `var` – string (default: `'q'`)

OUTPUT: power series

EXAMPLES:

```
sage: I = BrandtModule(11).right_ideals()[1]; I
Fractional ideal (8, 8*i, 6 + 4*i + 2*j, 4 + 2*i + 2*k)
sage: I.norm()
32
sage: I.theta_series(5)
1 + 12*q^2 + 12*q^3 + 12*q^4 + O(q^5)
sage: I.theta_series(5,'T')
1 + 12*T^2 + 12*T^3 + 12*T^4 + O(T^5)
sage: I.theta_series(3)
1 + 12*q^2 + O(q^3)
```

```
>>> from sage.all import *
>>> I = BrandtModule(Integer(11)).right_ideals()[Integer(1)]; I
Fractional ideal (8, 8*i, 6 + 4*i + 2*j, 4 + 2*i + 2*k)
```

(continues on next page)

(continued from previous page)

```
>>> I.norm()
32
>>> I.theta_series(Integer(5))
1 + 12*q^2 + 12*q^3 + 12*q^4 + O(q^5)
>>> I.theta_series(Integer(5),'T')
1 + 12*T^2 + 12*T^3 + 12*T^4 + O(T^5)
>>> I.theta_series(Integer(3))
1 + 12*q^2 + O(q^3)
```

theta_series_vector(B)

Return theta series coefficients of `self`, as a vector of B integers.

INPUT:

- `B` – positive integer

OUTPUT: vector over $\mathbf{Z}$ with B entries

EXAMPLES:

```
sage: I = BrandtModule(37).right_ideals()[1]; I
Fractional ideal (8, 8*i, 2 + 6*i + 2*j, 6 + 3*i + k)
sage: I.theta_series_vector(5)
(1, 0, 2, 2, 6)
sage: I.theta_series_vector(10)
(1, 0, 2, 2, 6, 4, 8, 6, 10, 10)
sage: I.theta_series_vector(5)
(1, 0, 2, 2, 6)
```

```
>>> from sage.all import *
>>> I = BrandtModule(Integer(37)).right_ideals()[Integer(1)]; I
Fractional ideal (8, 8*i, 2 + 6*i + 2*j, 6 + 3*i + k)
>>> I.theta_series_vector(Integer(5))
(1, 0, 2, 2, 6)
>>> I.theta_series_vector(Integer(10))
(1, 0, 2, 2, 6, 4, 8, 6, 10, 10)
>>> I.theta_series_vector(Integer(5))
(1, 0, 2, 2, 6)
```

class sage.algebras.quatalg.quaternion_algebra.**QuaternionOrder**(A, *basis*, *check=True*)

Bases: `Parent`

An order in a quaternion algebra.

EXAMPLES:

```
sage: QuaternionAlgebra(-1,-7).maximal_order()
Order of Quaternion Algebra (-1, -7) with base ring Rational Field with basis (1/
↪2 + 1/2*j, 1/2*i + 1/2*k, j, k)
sage: type(QuaternionAlgebra(-1,-7).maximal_order())
<class 'sage.algebras.quatalg.quaternion_algebra.QuaternionOrder_with_category'>
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(1),-Integer(7)).maximal_order()
```

(continues on next page)

(continued from previous page)

```
Order of Quaternion Algebra (-1, -7) with base ring Rational Field with basis (1/
↪2 + 1/2*j, 1/2*i + 1/2*k, j, k)
>>> type(QuaternionAlgebra(-Integer(1),-Integer(7)).maximal_order())
<class 'sage.algebras.quatalg.quaternion_algebra.QuaternionOrder_with_category'>
```

basis()

Return fix choice of basis for this quaternion order.

EXAMPLES:

```
sage: QuaternionAlgebra(-11,-1).maximal_order().basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
```

basis_matrix()

Return the basis matrix of this quaternion order, for the specific basis returned by `basis()`.

OUTPUT: matrix over **Q**

EXAMPLES:

```
sage: O = QuaternionAlgebra(-11,-1).maximal_order()
sage: O.basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
sage: O.basis_matrix()
[ 1/2  1/2    0    0]
[   0    0  1/2 -1/2]
[   0    1    0    0]
[   0    0    0   -1]
```

```
>>> from sage.all import *
>>> O = QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order()
>>> O.basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
>>> O.basis_matrix()
[ 1/2  1/2    0    0]
[   0    0  1/2 -1/2]
[   0    1    0    0]
[   0    0    0   -1]
```

Note that the returned matrix is *not* necessarily the same as the basis matrix of the `unit_ideal()`:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-1,-11)
sage: O = Q.quaternion_order([j,i,-1,k])
sage: O.basis_matrix()
[ 0  0  1  0]
[ 0  1  0  0]
[-1  0  0  0]
[ 0  0  0  1]
sage: O.unit_ideal().basis_matrix()
```

(continues on next page)

(continued from previous page)

```
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(1),-Integer(11), names=('i', 'j', 'k',));␣
↪(i, j, k,) = Q._first_ngens(3)
>>> O = Q.quaternion_order([j,i,-Integer(1),k])
>>> O.basis_matrix()
[ 0  0  1  0]
[ 0  1  0  0]
[-1  0  0  0]
[ 0  0  0  1]
>>> O.unit_ideal().basis_matrix()
[1 0 0 0]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
```

discriminant()

Return the discriminant of this order.

This is defined as $\sqrt{det(Tr(e_i\bar{e}_j))}$, where $\{e_i\}$ is the basis of the order.

OUTPUT: rational number

EXAMPLES:

```
sage: QuaternionAlgebra(-11,-1).maximal_order().discriminant()
11
sage: S = BrandtModule(11,5).order_of_level_N()
sage: S.discriminant()
55
sage: type(S.discriminant())
<... 'sage.rings.rational.Rational'>
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().discriminant()
11
>>> S = BrandtModule(Integer(11),Integer(5)).order_of_level_N()
>>> S.discriminant()
55
>>> type(S.discriminant())
<... 'sage.rings.rational.Rational'>
```

free_module()

Return the free **Z**-module that corresponds to this order inside the vector space corresponding to the ambient quaternion algebra.

OUTPUT: a free **Z**-module of rank 4

EXAMPLES:

```
sage: R = QuaternionAlgebra(-11,-1).maximal_order()
sage: R.basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
sage: R.free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[1/2 1/2   0   0]
[  0   1   0   0]
[  0   0 1/2 1/2]
[  0   0   0   1]
```

```
>>> from sage.all import *
>>> R = QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order()
>>> R.basis()
(1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
>>> R.free_module()
Free module of degree 4 and rank 4 over Integer Ring
Echelon basis matrix:
[1/2 1/2   0   0]
[  0   1   0   0]
[  0   0 1/2 1/2]
[  0   0   0   1]
```

gen(*n*)

Return the n-th generator.

INPUT:

- `n` – integer between 0 and 3, inclusive

EXAMPLES:

```
sage: R = QuaternionAlgebra(-11,-1).maximal_order(); R
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
sage: R.gen(0)
1/2 + 1/2*i
sage: R.gen(1)
1/2*j - 1/2*k
sage: R.gen(2)
i
sage: R.gen(3)
-k
```

```
>>> from sage.all import *
>>> R = QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order(); R
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
>>> R.gen(Integer(0))
1/2 + 1/2*i
>>> R.gen(Integer(1))
1/2*j - 1/2*k
>>> R.gen(Integer(2))
i
```

(continues on next page)

(continued from previous page)

```
>>> R.gen(Integer(3))
-k
```

gens()

Return generators for `self`.

EXAMPLES:

```
sage: QuaternionAlgebra(-1,-7).maximal_order().gens()
(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(1),-Integer(7)).maximal_order().gens()
(1/2 + 1/2*j, 1/2*i + 1/2*k, j, k)
```

intersection(*other*)

Return the intersection of this order with other.

INPUT:

- `other` – a quaternion order in the same ambient quaternion algebra

OUTPUT: a quaternion order

EXAMPLES:

```
sage: R = QuaternionAlgebra(-11,-1).maximal_order()
sage: R.intersection(R)
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1/2 + 1/2*i, i, 1/2*j + 1/2*k, k)
```

```
>>> from sage.all import *
>>> R = QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order()
>>> R.intersection(R)
Order of Quaternion Algebra (-11, -1) with base ring Rational Field
 with basis (1/2 + 1/2*i, i, 1/2*j + 1/2*k, k)
```

We intersect various orders in the quaternion algebra ramified at 11:

```
sage: B = BrandtModule(11,3)
sage: R = B.maximal_order(); S = B.order_of_level_N()
sage: R.intersection(S)
Order of Quaternion Algebra (-1, -11) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 5/2*k, j, 3*k)
sage: R.intersection(S) == S
True
sage: B = BrandtModule(11,5)
sage: T = B.order_of_level_N()
sage: S.intersection(T)
Order of Quaternion Algebra (-1, -11) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 23/2*k, j, 15*k)
```

```
>>> from sage.all import *
>>> B = BrandtModule(Integer(11),Integer(3))
```

(continues on next page)

(continued from previous page)

```
>>> R = B.maximal_order(); S = B.order_of_level_N()
>>> R.intersection(S)
Order of Quaternion Algebra (-1, -11) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 5/2*k, j, 3*k)
>>> R.intersection(S) == S
True
>>> B = BrandtModule(Integer(11),Integer(5))
>>> T = B.order_of_level_N()
>>> S.intersection(T)
Order of Quaternion Algebra (-1, -11) with base ring Rational Field
 with basis (1/2 + 1/2*j, 1/2*i + 23/2*k, j, 15*k)
```

is_maximal()

Check whether the order of `self` is maximal in the ambient quaternion algebra.

Only implemented for quaternion algebras over number fields; for reference, see Theorem 15.5.5 in [Voi2021].

EXAMPLES:

```
sage: p = 11
sage: B = QuaternionAlgebra(QQ, -1, -p)
sage: i, j, k = B.gens()
sage: O0_basis = (1, i, (i+j)/2, (1+i*j)/2)
sage: O0 = B.quaternion_order(O0_basis)
sage: O0.is_maximal()
True
sage: O1 = B.quaternion_order([1, i, j, i*j])
sage: O1.is_maximal()
False
```

```
>>> from sage.all import *
>>> p = Integer(11)
>>> B = QuaternionAlgebra(QQ, -Integer(1), -p)
>>> i, j, k = B.gens()
>>> O0_basis = (Integer(1), i, (i+j)/Integer(2), (Integer(1)+i*j)/Integer(2))
>>> O0 = B.quaternion_order(O0_basis)
>>> O0.is_maximal()
True
>>> O1 = B.quaternion_order([Integer(1), i, j, i*j])
>>> O1.is_maximal()
False
```

isomorphism_to(*other*, *conjugator*, *B*)

Compute an isomorphism from this quaternion order O to another order O' in the same quaternion algebra.

INPUT:

- `conjugator` – boolean (default: `False`); if `True` this method returns a single quaternion $\gamma \in O \cap O'$ of minimal norm such that $O' = \gamma^{-1} O \gamma$, rather than the ring isomorphism it defines
- `B` – positive integer; bound on theta series coefficients to rule out non isomorphic orders

Note

This method is currently only implemented for maximal orders in definite quaternion orders over **Q**. For a general algorithm, see [KV2010] (Problem `IsConjugate`).

EXAMPLES:

```
sage: Quat.<i,j,k> = QuaternionAlgebra(-1, -19)
sage: O0 = Quat.quaternion_order([1, i, (i+j)/2, (1+k)/2])
sage: O1 = Quat.quaternion_order([1, 667*i, 1/2+j/2+9*i, (222075/2*i+333*j+k/
↪2)/667])
sage: iso = O0.isomorphism_to(O1)
sage: iso
Ring morphism:
  From: Order of Quaternion Algebra (-1, -19) with base ring Rational Field
         with basis (1, i, 1/2*i + 1/2*j, 1/2 + 1/2*k)
  To:   Order of Quaternion Algebra (-1, -19) with base ring Rational Field
         with basis (1, 667*i, 1/2 + 9*i + 1/2*j, 222075/1334*i + 333/667*j +␣
↪1/1334*k)
  Defn: i |--> 629/667*i + 36/667*j - 36/667*k
        j |--> -684/667*i + 648/667*j + 19/667*k
        k |--> 684/667*i + 19/667*j + 648/667*k
sage: iso(1)
1
sage: iso(i)
629/667*i + 36/667*j - 36/667*k
sage: iso(i/3)
Traceback (most recent call last):
...
TypeError: 1/3*i fails to convert into the map's domain ...
```

```
>>> from sage.all import *
>>> Quat = QuaternionAlgebra(-Integer(1), -Integer(19), names=('i', 'j', 'k',
↪)); (i, j, k,) = Quat._first_ngens(3)
>>> O0 = Quat.quaternion_order([Integer(1), i, (i+j)/Integer(2),␣
↪(Integer(1)+k)/Integer(2)])
>>> O1 = Quat.quaternion_order([Integer(1), Integer(667)*i, Integer(1)/
↪Integer(2)+j/Integer(2)+Integer(9)*i, (Integer(222075)/
↪Integer(2)*i+Integer(333)*j+k/Integer(2))/Integer(667)])
>>> iso = O0.isomorphism_to(O1)
>>> iso
Ring morphism:
  From: Order of Quaternion Algebra (-1, -19) with base ring Rational Field
         with basis (1, i, 1/2*i + 1/2*j, 1/2 + 1/2*k)
  To:   Order of Quaternion Algebra (-1, -19) with base ring Rational Field
         with basis (1, 667*i, 1/2 + 9*i + 1/2*j, 222075/1334*i + 333/667*j +␣
↪1/1334*k)
  Defn: i |--> 629/667*i + 36/667*j - 36/667*k
        j |--> -684/667*i + 648/667*j + 19/667*k
        k |--> 684/667*i + 19/667*j + 648/667*k
>>> iso(Integer(1))
1
```

(continues on next page)

(continued from previous page)

```
>>> iso(i)
629/667*i + 36/667*j - 36/667*k
>>> iso(i/Integer(3))
Traceback (most recent call last):
...
TypeError: 1/3*i fails to convert into the map's domain ...
```

```
sage: gamma = O0.isomorphism_to(O1, conjugator=True); gamma
-36 + j + k
sage: gamma in O0
True
sage: gamma in O1
True
sage: O1.unit_ideal() == ~gamma * O0 * gamma
True
```

```
>>> from sage.all import *
>>> gamma = O0.isomorphism_to(O1, conjugator=True); gamma
-36 + j + k
>>> gamma in O0
True
>>> gamma in O1
True
>>> O1.unit_ideal() == ~gamma * O0 * gamma
True
```

ALGORITHM:

Find a generator of the principal lattice $N \cdot O \cdot O'$ where $N = [O : OcapO']$ using *QuaternionFractionalIdeal_rational.minimal_element()*. An isomorphism is given by conjugation by such an element. Works providing reduced norm of conjugation element is not a ramified prime times a square. To cover cases where it is we repeat the check for orders conjugated by i, j, and k.

left_ideal(*gens*, *check*, *is_basis=True*)

Return the left ideal of this order generated by the given generators.

INPUT:

- `gens` – list of elements of this quaternion order
- `check` – boolean (default: `True`)
- `is_basis` – boolean (default: `False`); if `True` then `gens` must be a **Z**-basis of the ideal

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-11,-1)
sage: R = Q.maximal_order()
sage: R.left_ideal([a*2 for a in R.basis()], is_basis=True)
Fractional ideal (1 + i, 2*i, j + k, 2*k)
sage: R.left_ideal([a*(i+j) for a in R.basis()], is_basis=True)
Fractional ideal (1/2 + 1/2*i + 1/2*j + 13/2*k, i + j, 6*j + 6*k, 12*k)
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(11),-Integer(1), names=('i', 'j', 'k',));␣
```

(continues on next page)

(continued from previous page)

```
↪(i, j, k,) = Q._first_ngens(3)
>>> R = Q.maximal_order()
>>> R.left_ideal([a*Integer(2) for a in R.basis()], is_basis=True)
Fractional ideal (1 + i, 2*i, j + k, 2*k)
>>> R.left_ideal([a*(i+j) for a in R.basis()], is_basis=True)
Fractional ideal (1/2 + 1/2*i + 1/2*j + 13/2*k, i + j, 6*j + 6*k, 12*k)
```

It is also possible to pass a generating set (rather than a basis), or a single generator:

```
sage: R.left_ideal([i+j])
Fractional ideal (12, 6 + 6*i, i + j, 13/2 + 1/2*i + 1/2*j + 1/2*k)
sage: R.left_ideal(i+j)
Fractional ideal (12, 6 + 6*i, i + j, 13/2 + 1/2*i + 1/2*j + 1/2*k)
sage: R.left_ideal([2, 1+j]) == R*2 + R*(1+j)
True
```

```
>>> from sage.all import *
>>> R.left_ideal([i+j])
Fractional ideal (12, 6 + 6*i, i + j, 13/2 + 1/2*i + 1/2*j + 1/2*k)
>>> R.left_ideal(i+j)
Fractional ideal (12, 6 + 6*i, i + j, 13/2 + 1/2*i + 1/2*j + 1/2*k)
>>> R.left_ideal([Integer(2), Integer(1)+j]) == R*Integer(2) +␣
↪R*(Integer(1)+j)
True
```

ngens()

Return the number of generators (which is 4).

EXAMPLES:

```
sage: QuaternionAlgebra(-1,-7).maximal_order().ngens()
4
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(1),-Integer(7)).maximal_order().ngens()
4
```

one()

Return the multiplicative unit of this quaternion order.

EXAMPLES:

```
sage: QuaternionAlgebra(-1,-7).maximal_order().one()
1
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(1),-Integer(7)).maximal_order().one()
1
```

quadratic_form()

Return the normalized quadratic form associated to this quaternion order.

OUTPUT: quadratic form

EXAMPLES:

```
sage: R = BrandtModule(11,13).order_of_level_N()
sage: Q = R.quadratic_form(); Q
Quadratic form in 4 variables over Rational Field with coefficients:
[ 14 253 55 286 ]
[ * 1455 506 3289 ]
[ * * 55 572 ]
[ * * * 1859 ]
sage: Q.theta_series(10)
1 + 2*q + 2*q^4 + 4*q^6 + 4*q^8 + 2*q^9 + O(q^10)
```

```
>>> from sage.all import *
>>> R = BrandtModule(Integer(11),Integer(13)).order_of_level_N()
>>> Q = R.quadratic_form(); Q
Quadratic form in 4 variables over Rational Field with coefficients:
[ 14 253 55 286 ]
[ * 1455 506 3289 ]
[ * * 55 572 ]
[ * * * 1859 ]
>>> Q.theta_series(Integer(10))
1 + 2*q + 2*q^4 + 4*q^6 + 4*q^8 + 2*q^9 + O(q^10)
```

quaternion_algebra()

Return ambient quaternion algebra that contains this quaternion order.

EXAMPLES:

```
sage: QuaternionAlgebra(-11,-1).maximal_order().quaternion_algebra()
Quaternion Algebra (-11, -1) with base ring Rational Field
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().quaternion_
↪algebra()
Quaternion Algebra (-11, -1) with base ring Rational Field
```

random_element(**args*, ***kwds*)

Return a random element of this order.

The args and kwds are passed to the random_element method of the integer ring, and we return an element of the form

$$ae_1 + be_2 + ce_3 + de_4$$

where e_1, …, e_4 are the basis of this order and a, b, c, d are random integers.

EXAMPLES:

```
sage: QuaternionAlgebra(-11,-1).maximal_order().random_element()   # random
-4 - 4*i + j - k
sage: QuaternionAlgebra(-11,-1).maximal_order().random_element(-10,10)   #␣
↪random
-9/2 - 7/2*i - 7/2*j - 3/2*k
```

```
>>> from sage.all import *
>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().random_
↪element()   # random
-4 - 4*i + j - k
>>> QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order().random_
↪element(-Integer(10),Integer(10))   # random
-9/2 - 7/2*i - 7/2*j - 3/2*k
```

right_ideal(*gens*, *check*, *is_basis=True*)

Return the right ideal of this order generated by the given generators.

INPUT:

- `gens` – list of elements of this quaternion order
- `check` – boolean (default: `True`)
- `is_basis` – boolean (default: `False`); if `True` then `gens` must be a **Z**-basis of the ideal

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-11,-1)
sage: R = Q.maximal_order()
sage: R.right_ideal([2*a for a in R.basis()], is_basis=True)
Fractional ideal (1 + i, 2*i, j + k, 2*k)
sage: R.right_ideal([(i+j)*a for a in R.basis()], is_basis=True)
Fractional ideal (1/2 + 1/2*i + 1/2*j + 11/2*k, i + j, 6*j + 6*k, 12*k)
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(11),-Integer(1), names=('i', 'j', 'k',));␣
↪(i, j, k,) = Q._first_ngens(3)
>>> R = Q.maximal_order()
>>> R.right_ideal([Integer(2)*a for a in R.basis()], is_basis=True)
Fractional ideal (1 + i, 2*i, j + k, 2*k)
>>> R.right_ideal([(i+j)*a for a in R.basis()], is_basis=True)
Fractional ideal (1/2 + 1/2*i + 1/2*j + 11/2*k, i + j, 6*j + 6*k, 12*k)
```

It is also possible to pass a generating set (rather than a basis), or a single generator:

```
sage: R.right_ideal([i+j])
Fractional ideal (12, 6 + 6*i, i + j, 11/2 + 1/2*i + 1/2*j + 1/2*k)
sage: R.right_ideal(i+j)
Fractional ideal (12, 6 + 6*i, i + j, 11/2 + 1/2*i + 1/2*j + 1/2*k)
sage: R.right_ideal([2, 1+j]) == 2*R + (1+j)*R
True
```

```
>>> from sage.all import *
>>> R.right_ideal([i+j])
Fractional ideal (12, 6 + 6*i, i + j, 11/2 + 1/2*i + 1/2*j + 1/2*k)
>>> R.right_ideal(i+j)
Fractional ideal (12, 6 + 6*i, i + j, 11/2 + 1/2*i + 1/2*j + 1/2*k)
>>> R.right_ideal([Integer(2), Integer(1)+j]) == Integer(2)*R +␣
↪(Integer(1)+j)*R
True
```

ternary_quadratic_form(*include_basis=False*)

Return the ternary quadratic form associated to this order.

INPUT:

- `include_basis` – boolean (default: `False`); if `True` also return a basis for the dimension 3 subspace G

OUTPUT: QuadraticForm

- optional basis for dimension 3 subspace

This function computes the positive definition quadratic form obtained by letting G be the trace zero subspace of $\mathbf{Z} + 2*$ `self`, which has rank 3, and restricting the pairing `QuaternionAlgebraElement_abstract.pair()`:

```
(x,y) = (x.conjugate()*y).reduced_trace()
```

to G.

APPLICATIONS: Ternary quadratic forms associated to an order in a rational quaternion algebra are useful in computing with Gross points, in decided whether quaternion orders have embeddings from orders in quadratic imaginary fields, and in computing elements of the Kohnen plus subspace of modular forms of weight 3/2.

EXAMPLES:

```
sage: R = BrandtModule(11,13).order_of_level_N()
sage: Q = R.ternary_quadratic_form(); Q
Quadratic form in 3 variables over Rational Field with coefficients:
[ 5820 1012 13156 ]
[ * 55 1144 ]
[ * * 7436 ]
sage: factor(Q.disc())
2^4 * 11^2 * 13^2
```

```
>>> from sage.all import *
>>> R = BrandtModule(Integer(11),Integer(13)).order_of_level_N()
>>> Q = R.ternary_quadratic_form(); Q
Quadratic form in 3 variables over Rational Field with coefficients:
[ 5820 1012 13156 ]
[ * 55 1144 ]
[ * * 7436 ]
>>> factor(Q.disc())
2^4 * 11^2 * 13^2
```

The following theta series is a modular form of weight 3/2 and level 4*11*13:

```
sage: Q.theta_series(100)
1 + 2*q^23 + 2*q^55 + 2*q^56 + 2*q^75 + 4*q^92 + O(q^100)
```

```
>>> from sage.all import *
>>> Q.theta_series(Integer(100))
1 + 2*q^23 + 2*q^55 + 2*q^56 + 2*q^75 + 4*q^92 + O(q^100)
```

unit_ideal()

Return the unit ideal in this quaternion order.

EXAMPLES:

```
sage: R = QuaternionAlgebra(-11,-1).maximal_order()
sage: I = R.unit_ideal(); I
Fractional ideal (1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
```

```
>>> from sage.all import *
>>> R = QuaternionAlgebra(-Integer(11),-Integer(1)).maximal_order()
>>> I = R.unit_ideal(); I
Fractional ideal (1/2 + 1/2*i, 1/2*j - 1/2*k, i, -k)
```

`sage.algebras.quatalg.quaternion_algebra.`**`basis_for_quaternion_lattice`**`(`*gens*, *reverse=True*`)`

Return a basis for the **Z**-lattice in a quaternion algebra spanned by the given gens.

INPUT:

- `gens` – list of elements of a single quaternion algebra
- `reverse` – when computing the HNF do it on the basis $(k, j, i, 1)$ instead of $(1, i, j, k)$; this ensures that if `gens` are the generators for a fractional ideal (in particular, an order), the first returned basis vector equals the norm of the ideal (in case of an order, 1)

EXAMPLES:

```
sage: from sage.algebras.quatalg.quaternion_algebra import basis_for_quaternion_
↪lattice
sage: A.<i,j,k> = QuaternionAlgebra(-1,-7)
sage: basis_for_quaternion_lattice([i+j, i-j, 2*k, A(1/3)])
[1/3, 2*i, i + j, 2*k]
sage: basis_for_quaternion_lattice([A(1),i,j,k])
[1, i, j, k]
```

```
>>> from sage.all import *
>>> from sage.algebras.quatalg.quaternion_algebra import basis_for_quaternion_
↪lattice
>>> A = QuaternionAlgebra(-Integer(1),-Integer(7), names=('i', 'j', 'k',)); (i, j,
↪ k,) = A._first_ngens(3)
>>> basis_for_quaternion_lattice([i+j, i-j, Integer(2)*k, A(Integer(1)/
↪Integer(3))])
[1/3, 2*i, i + j, 2*k]
>>> basis_for_quaternion_lattice([A(Integer(1)),i,j,k])
[1, i, j, k]
```

`sage.algebras.quatalg.quaternion_algebra.`**`intersection_of_row_modules_over_ZZ`**`(`*v*`)`

Intersect the **Z**-modules with basis matrices the full rank 4×4 **Q**-matrices in the list v.

The returned intersection is represented by a 4×4 matrix over **Q**. This can also be done using modules and intersection, but that would take over twice as long because of overhead, hence this function.

EXAMPLES:

```
sage: a = matrix(QQ,4,[-2, 0, 0, 0, 0, -1, -1, 1, 2, -1/2, 0, 0, 1, 1, -1, 0])
sage: b = matrix(QQ,4,[0, -1/2, 0, -1/2, 2, 1/2, -1, -1/2, 1, 2, 1, -2, 0, -1/2, -
↪2, 0])
sage: c = matrix(QQ,4,[0, 1, 0, -1/2, 0, 0, 2, 2, 0, -1/2, 1/2, -1, 1, -1, -1/2,␣
↪0])
sage: v = [a,b,c]
```

(continues on next page)

(continued from previous page)

```
sage: from sage.algebras.quatalg.quaternion_algebra import intersection_of_row_
↪modules_over_ZZ
sage: M = intersection_of_row_modules_over_ZZ(v); M
[   2    0   -1   -1]
[  -4    1    1   -3]
[  -3 19/2   -1   -4]
[   2   -3   -8    4]
sage: M2 = a.row_module(ZZ).intersection(b.row_module(ZZ)).intersection(c.row_
↪module(ZZ))
sage: M.row_module(ZZ) == M2
True
```

```
>>> from sage.all import *
>>> a = matrix(QQ,Integer(4),[-Integer(2), Integer(0), Integer(0), Integer(0),␣
↪Integer(0), -Integer(1), -Integer(1), Integer(1), Integer(2), -Integer(1)/
↪Integer(2), Integer(0), Integer(0), Integer(1), Integer(1), -Integer(1),␣
↪Integer(0)])
>>> b = matrix(QQ,Integer(4),[Integer(0), -Integer(1)/Integer(2), Integer(0), -
↪Integer(1)/Integer(2), Integer(2), Integer(1)/Integer(2), -Integer(1), -
↪Integer(1)/Integer(2), Integer(1), Integer(2), Integer(1), -Integer(2),␣
↪Integer(0), -Integer(1)/Integer(2), -Integer(2), Integer(0)])
>>> c = matrix(QQ,Integer(4),[Integer(0), Integer(1), Integer(0), -Integer(1)/
↪Integer(2), Integer(0), Integer(0), Integer(2), Integer(2), Integer(0), -
↪Integer(1)/Integer(2), Integer(1)/Integer(2), -Integer(1), Integer(1), -
↪Integer(1), -Integer(1)/Integer(2), Integer(0)])
>>> v = [a,b,c]
>>> from sage.algebras.quatalg.quaternion_algebra import intersection_of_row_
↪modules_over_ZZ
>>> M = intersection_of_row_modules_over_ZZ(v); M
[   2    0   -1   -1]
[  -4    1    1   -3]
[  -3 19/2   -1   -4]
[   2   -3   -8    4]
>>> M2 = a.row_module(ZZ).intersection(b.row_module(ZZ)).intersection(c.row_
↪module(ZZ))
>>> M.row_module(ZZ) == M2
True
```

sage.algebras.quatalg.quaternion_algebra.**is_QuaternionAlgebra**(A)

Return `True` if `A` is of the QuaternionAlgebra data type.

EXAMPLES:

```
sage: sage.algebras.quatalg.quaternion_algebra.is_
↪QuaternionAlgebra(QuaternionAlgebra(QQ,-1,-1))
doctest:warning...
DeprecationWarning: the function is_QuaternionAlgebra is deprecated;
use 'isinstance(..., QuaternionAlgebra_abstract)' instead
See https://github.com/sagemath/sage/issues/37896 for details.
True
sage: sage.algebras.quatalg.quaternion_algebra.is_QuaternionAlgebra(ZZ)
False
```

```
>>> from sage.all import *
>>> sage.algebras.quatalg.quaternion_algebra.is_
↪QuaternionAlgebra(QuaternionAlgebra(QQ,-Integer(1),-Integer(1)))
doctest:warning...
DeprecationWarning: the function is_QuaternionAlgebra is deprecated;
use 'isinstance(..., QuaternionAlgebra_abstract)' instead
See https://github.com/sagemath/sage/issues/37896 for details.
True
>>> sage.algebras.quatalg.quaternion_algebra.is_QuaternionAlgebra(ZZ)
False
```

`sage.algebras.quatalg.quaternion_algebra.`**`maxord_solve_aux_eq`**`(`*a*, *b*, *p*`)`

Given `a` and `b` and an even prime ideal `p` find (y,z,w) with y a unit mod p^{2e} such that

$$1 - ay^2 - bz^2 + abw^2 \equiv 0 mod p^{2e},$$

where e is the ramification index of p.

Currently only $p = 2$ is implemented by hardcoding solutions.

INPUT:

- `a` – integer with $v_p(a) = 0$
- `b` – integer with $v_p(b) \in \{0, 1\}$
- `p` – even prime ideal (actually only `p=ZZ(2)` is implemented)

OUTPUT: a tuple (y, z, w)

EXAMPLES:

```
sage: from sage.algebras.quatalg.quaternion_algebra import maxord_solve_aux_eq
sage: for a in [1,3]:
....:     for b in [1,2,3]:
....:         (y,z,w) = maxord_solve_aux_eq(a, b, 2)
....:         assert mod(y, 4) == 1 or mod(y, 4) == 3
....:         assert mod(1 - a*y^2 - b*z^2 + a*b*w^2, 4) == 0
```

```
>>> from sage.all import *
>>> from sage.algebras.quatalg.quaternion_algebra import maxord_solve_aux_eq
>>> for a in [Integer(1),Integer(3)]:
...     for b in [Integer(1),Integer(2),Integer(3)]:
...         (y,z,w) = maxord_solve_aux_eq(a, b, Integer(2))
...         assert mod(y, Integer(4)) == Integer(1) or mod(y, Integer(4)) ==␣
↪Integer(3)
...         assert mod(Integer(1) - a*y**Integer(2) - b*z**Integer(2) +␣
↪a*b*w**Integer(2), Integer(4)) == Integer(0)
```

`sage.algebras.quatalg.quaternion_algebra.`**`normalize_basis_at_p`**`(`*e*, *p*, *B=<method 'pair' of 'sage.algebras.quatalg.quaternion_algebra_element.QuaternionAlgebraElement_abstract' objects>*`)`

Compute a (at `p`) normalized basis from the given basis `e` of a $\mathbf{Z}$-module.

The returned basis is (at `p`) a $\mathbf{Z}_p$ basis for the same module, and has the property that with respect to it the quadratic form induced by the bilinear form B is represented as a orthogonal sum of atomic forms multiplied by p-powers.

If $p \neq 2$ this means that the form is diagonal with respect to this basis.

If $p = 2$ there may be additional 2-dimensional subspaces on which the form is represented as $2^e(ax^2 + bxy + cx^2)$ with $0 = v_2(b) = v_2(a) \leq v_2(c)$.

INPUT:

- `e` – list; basis of a $\mathbf{Z}$ module (WARNING: will be modified!)
- `p` – prime for at which the basis should be normalized
- `B` – (default: *QuaternionAlgebraElement_abstract.pair()*) a bilinear form with respect to which to normalize

OUTPUT:

- A list containing two-element tuples: The first element of each tuple is a basis element, the second the valuation of the orthogonal summand to which it belongs. The list is sorted by ascending valuation.

EXAMPLES:

```
sage: from sage.algebras.quatalg.quaternion_algebra import normalize_basis_at_p
sage: A.<i,j,k> = QuaternionAlgebra(-1, -1)
sage: e = [A(1), i, j, k]
sage: normalize_basis_at_p(e, 2)
[(1, 0), (i, 0), (j, 0), (k, 0)]

sage: A.<i,j,k> = QuaternionAlgebra(210)
sage: e = [A(1), i, j, k]
sage: normalize_basis_at_p(e, 2)
[(1, 0), (i, 1), (j, 1), (k, 2)]

sage: A.<i,j,k> = QuaternionAlgebra(286)
sage: e = [A(1), k, 1/2*j + 1/2*k, 1/2 + 1/2*i + 1/2*k]
sage: normalize_basis_at_p(e, 5)
[(1, 0), (1/2*j + 1/2*k, 0), (-5/6*j + 1/6*k, 1), (1/2*i, 1)]

sage: A.<i,j,k> = QuaternionAlgebra(-1,-7)
sage: e = [A(1), k, j, 1/2 + 1/2*i + 1/2*j + 1/2*k]
sage: normalize_basis_at_p(e, 2)
[(1, 0), (1/2 + 1/2*i + 1/2*j + 1/2*k, 0), (-34/105*i - 463/735*j + 71/105*k, 1),
 (1/7*i - 8/49*j + 1/7*k, 1)]
```

```
>>> from sage.all import *
>>> from sage.algebras.quatalg.quaternion_algebra import normalize_basis_at_p
>>> A = QuaternionAlgebra(-Integer(1), -Integer(1), names=('i', 'j', 'k',)); (i,␣
↪j, k,) = A._first_ngens(3)
>>> e = [A(Integer(1)), i, j, k]
>>> normalize_basis_at_p(e, Integer(2))
[(1, 0), (i, 0), (j, 0), (k, 0)]

>>> A = QuaternionAlgebra(Integer(210), names=('i', 'j', 'k',)); (i, j, k,) = A._
↪first_ngens(3)
>>> e = [A(Integer(1)), i, j, k]
>>> normalize_basis_at_p(e, Integer(2))
[(1, 0), (i, 1), (j, 1), (k, 2)]
```

(continues on next page)

(continued from previous page)

```
>>> A = QuaternionAlgebra(Integer(286), names=('i', 'j', 'k',)); (i, j, k,) = A._
↪first_ngens(3)
>>> e = [A(Integer(1)), k, Integer(1)/Integer(2)*j + Integer(1)/Integer(2)*k,␣
↪Integer(1)/Integer(2) + Integer(1)/Integer(2)*i + Integer(1)/Integer(2)*k]
>>> normalize_basis_at_p(e, Integer(5))
[(1, 0), (1/2*j + 1/2*k, 0), (-5/6*j + 1/6*k, 1), (1/2*i, 1)]

>>> A = QuaternionAlgebra(-Integer(1),-Integer(7), names=('i', 'j', 'k',)); (i, j,
↪ k,) = A._first_ngens(3)
>>> e = [A(Integer(1)), k, j, Integer(1)/Integer(2) + Integer(1)/Integer(2)*i +␣
↪Integer(1)/Integer(2)*j + Integer(1)/Integer(2)*k]
>>> normalize_basis_at_p(e, Integer(2))
[(1, 0), (1/2 + 1/2*i + 1/2*j + 1/2*k, 0), (-34/105*i - 463/735*j + 71/105*k, 1),
 (1/7*i - 8/49*j + 1/7*k, 1)]
```

sage.algebras.quatalg.quaternion_algebra.**unpickle_QuaternionAlgebra_v0**(**key*)

The 0-th version of pickling for quaternion algebras.

EXAMPLES:

```
sage: Q = QuaternionAlgebra(-5,-19)
sage: t = (QQ, -5, -19, ('i', 'j', 'k'))
sage: sage.algebras.quatalg.quaternion_algebra.unpickle_QuaternionAlgebra_v0(*t)
Quaternion Algebra (-5, -19) with base ring Rational Field
sage: loads(dumps(Q)) == Q
True
sage: loads(dumps(Q)) is Q
True
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(5),-Integer(19))
>>> t = (QQ, -Integer(5), -Integer(19), ('i', 'j', 'k'))
>>> sage.algebras.quatalg.quaternion_algebra.unpickle_QuaternionAlgebra_v0(*t)
Quaternion Algebra (-5, -19) with base ring Rational Field
>>> loads(dumps(Q)) == Q
True
>>> loads(dumps(Q)) is Q
True
```

CHAPTER

TWO

ELEMENTS OF QUATERNION ALGEBRAS

Sage allows for computation with elements of quaternion algebras over a nearly arbitrary base field of characteristic not 2. Sage also has very highly optimized implementation of arithmetic in rational quaternion algebras and quaternion algebras over number fields.

class `sage.algebras.quatalg.quaternion_algebra_element.`**`QuaternionAlgebraElement_abstract`**

Bases: `AlgebraElement`

`coefficient_tuple``()`

Return 4-tuple of coefficients of this quaternion.

EXAMPLES:

```
sage: K.<x> = QQ['x']
sage: Q.<i,j,k> = QuaternionAlgebra(Frac(K),-5,-2)
sage: a = 1/2*x^2 + 2/3*x*i - 3/4*j + 5/7*k
sage: type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_generic'>
sage: a.coefficient_tuple()
(1/2*x^2, 2/3*x, -3/4, 5/7)
```

```
>>> from sage.all import *
>>> K = QQ['x']; (x,) = K._first_ngens(1)
>>> Q = QuaternionAlgebra(Frac(K),-Integer(5),-Integer(2), names=('i', 'j', 'k
↪',)); (i, j, k,) = Q._first_ngens(3)
>>> a = Integer(1)/Integer(2)*x**Integer(2) + Integer(2)/Integer(3)*x*i -␣
↪Integer(3)/Integer(4)*j + Integer(5)/Integer(7)*k
>>> type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_generic'>
>>> a.coefficient_tuple()
(1/2*x^2, 2/3*x, -3/4, 5/7)
```

`conjugate``()`

Return the conjugate of the quaternion: if $\theta = x + yi + zj + wk$, return $x - yi - zj - wk$; that is, return theta.reduced_trace() - theta.

EXAMPLES:

```
sage: A.<i,j,k> = QuaternionAlgebra(QQ,-5,-2)
sage: a = 3*i - j + 2
```

(continues on next page)

(continued from previous page)

```
sage: type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_rational_field'>
sage: a.conjugate()
2 - 3*i + j
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(QQ,-Integer(5),-Integer(2), names=('i', 'j', 'k',));
↪ (i, j, k,) = A._first_ngens(3)
>>> a = Integer(3)*i - j + Integer(2)
>>> type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_rational_field'>
>>> a.conjugate()
2 - 3*i + j
```

The “universal” test:

```
sage: K.<x,y,z,w,a,b> = QQ[]
sage: Q.<i,j,k> = QuaternionAlgebra(a,b)
sage: theta = x+y*i+z*j+w*k
sage: theta.conjugate()
x + (-y)*i + (-z)*j + (-w)*k
```

```
>>> from sage.all import *
>>> K = QQ['x, y, z, w, a, b']; (x, y, z, w, a, b,) = K._first_ngens(6)
>>> Q = QuaternionAlgebra(a,b, names=('i', 'j', 'k',)); (i, j, k,) = Q._first_
↪ngens(3)
>>> theta = x+y*i+z*j+w*k
>>> theta.conjugate()
x + (-y)*i + (-z)*j + (-w)*k
```

is_constant()

Return `True` if this quaternion is constant, i.e., has no i, j, or k term.

OUTPUT: boolean

EXAMPLES:

```
sage: A.<i,j,k> = QuaternionAlgebra(-1,-2)
sage: A(1).is_constant()
True
sage: A(1+i).is_constant()
False
sage: A(i).is_constant()
False
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(1),-Integer(2), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
>>> A(Integer(1)).is_constant()
True
```

(continues on next page)

(continued from previous page)

```
>>> A(Integer(1)+i).is_constant()
False
>>> A(i).is_constant()
False
```

matrix(*action='right'*)

Return the matrix of right or left multiplication of `self` on the basis for the ambient quaternion algebra.

In particular, if action is `'right'` (the default), returns the matrix of the mapping sending `x` to `x*self`.

INPUT:

- `action` – (default: `'right'`) `'right'` or `'left'`

OUTPUT: a matrix

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-3,-19)
sage: a = 2/3 -1/2*i + 3/5*j - 4/3*k
sage: a.matrix()
[  2/3  -1/2   3/5  -4/3]
[  3/2   2/3     4   3/5]
[-57/5 -76/3   2/3   1/2]
[   76 -57/5  -3/2   2/3]
sage: a.matrix() == a.matrix(action='right')
True
sage: a.matrix(action='left')
[  2/3  -1/2   3/5  -4/3]
[  3/2   2/3    -4  -3/5]
[-57/5  76/3   2/3  -1/2]
[   76  57/5   3/2   2/3]
sage: (i*a,j*a,k*a)
(3/2 + 2/3*i + 4*j + 3/5*k, -57/5 - 76/3*i + 2/3*j + 1/2*k, 76 - 57/5*i - 3/
↪2*j + 2/3*k)
sage: a.matrix(action='foo')
Traceback (most recent call last):
...
ValueError: action must be either 'left' or 'right'
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(3),-Integer(19), names=('i', 'j', 'k',));␣
↪(i, j, k,) = Q._first_ngens(3)
>>> a = Integer(2)/Integer(3) -Integer(1)/Integer(2)*i + Integer(3)/
↪Integer(5)*j - Integer(4)/Integer(3)*k
>>> a.matrix()
[  2/3  -1/2   3/5  -4/3]
[  3/2   2/3     4   3/5]
[-57/5 -76/3   2/3   1/2]
[   76 -57/5  -3/2   2/3]
>>> a.matrix() == a.matrix(action='right')
True
>>> a.matrix(action='left')
[  2/3  -1/2   3/5  -4/3]
```

(continues on next page)

(continued from previous page)

```
[  3/2   2/3    -4  -3/5]
[-57/5  76/3   2/3  -1/2]
[   76  57/5   3/2   2/3]
>>> (i*a,j*a,k*a)
(3/2 + 2/3*i + 4*j + 3/5*k, -57/5 - 76/3*i + 2/3*j + 1/2*k, 76 - 57/5*i - 3/
↪2*j + 2/3*k)
>>> a.matrix(action='foo')
Traceback (most recent call last):
...
ValueError: action must be either 'left' or 'right'
```

We test over a more generic base field:

```
sage: K.<x> = QQ['x']
sage: Q.<i,j,k> = QuaternionAlgebra(Frac(K),-5,-2)
sage: a = 1/2*x^2 + 2/3*x*i - 3/4*j + 5/7*k
sage: type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_generic'>
sage: a.matrix()
[1/2*x^2   2/3*x    -3/4     5/7]
[-10/3*x 1/2*x^2   -25/7    -3/4]
[    3/2    10/7 1/2*x^2  -2/3*x]
[  -50/7     3/2  10/3*x 1/2*x^2]
```

```
>>> from sage.all import *
>>> K = QQ['x']; (x,) = K._first_ngens(1)
>>> Q = QuaternionAlgebra(Frac(K),-Integer(5),-Integer(2), names=('i', 'j', 'k
↪',)); (i, j, k,) = Q._first_ngens(3)
>>> a = Integer(1)/Integer(2)*x**Integer(2) + Integer(2)/Integer(3)*x*i -␣
↪Integer(3)/Integer(4)*j + Integer(5)/Integer(7)*k
>>> type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_generic'>
>>> a.matrix()
[1/2*x^2   2/3*x    -3/4     5/7]
[-10/3*x 1/2*x^2   -25/7    -3/4]
[    3/2    10/7 1/2*x^2  -2/3*x]
[  -50/7     3/2  10/3*x 1/2*x^2]
```

pair(*right*)

Return the result of pairing `self` and `right`, which should both be elements of a quaternion algebra. The pairing is `(x,y) = (x.conjugate()*y).reduced_trace()`.

INPUT:

- `right` – quaternion

EXAMPLES:

```
sage: A.<i,j,k>=QuaternionAlgebra(-1,-2)
sage: (1+i+j-2*k).pair(2/3+5*i-3*j+k)
-26/3
```

(continues on next page)

(continued from previous page)

```
sage: x = 1+i+j-2*k; y = 2/3+5*i-3*j+k
sage: x.pair(y)
-26/3
sage: y.pair(x)
-26/3
sage: (x.conjugate()*y).reduced_trace()
-26/3
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(1),-Integer(2), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
>>> (Integer(1)+i+j-Integer(2)*k).pair(Integer(2)/Integer(3)+Integer(5)*i-
↪Integer(3)*j+k)
-26/3
>>> x = Integer(1)+i+j-Integer(2)*k; y = Integer(2)/Integer(3)+Integer(5)*i-
↪Integer(3)*j+k
>>> x.pair(y)
-26/3
>>> y.pair(x)
-26/3
>>> (x.conjugate()*y).reduced_trace()
-26/3
```

reduced_characteristic_polynomial(*var='x'*)

Return the reduced characteristic polynomial of this quaternion algebra element, which is $X^2 - tX + n$, where t is the reduced trace and n is the reduced norm.

INPUT:

- `var` – string (default: `'x'`); indeterminate of characteristic polynomial

EXAMPLES:

```
sage: A.<i,j,k>=QuaternionAlgebra(-1,-2)
sage: i.reduced_characteristic_polynomial()
x^2 + 1
sage: j.reduced_characteristic_polynomial()
x^2 + 2
sage: (i+j).reduced_characteristic_polynomial()
x^2 + 3
sage: (2+j+k).reduced_trace()
4
sage: (2+j+k).reduced_characteristic_polynomial('T')
T^2 - 4*T + 8
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(1),-Integer(2), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
>>> i.reduced_characteristic_polynomial()
x^2 + 1
>>> j.reduced_characteristic_polynomial()
x^2 + 2
>>> (i+j).reduced_characteristic_polynomial()
```

(continues on next page)

(continued from previous page)

```
x^2 + 3
>>> (Integer(2)+j+k).reduced_trace()
4
>>> (Integer(2)+j+k).reduced_characteristic_polynomial('T')
T^2 - 4*T + 8
```

reduced_norm()

Return the reduced norm of self: if $\theta = x + yi + zj + wk$, then θ has reduced norm $x^2 - ay^2 - bz^2 + abw^2$.

EXAMPLES:

```
sage: K.<x,y,z,w,a,b> = QQ[]
sage: Q.<i,j,k> = QuaternionAlgebra(a,b)
sage: theta = x+y*i+z*j+w*k
sage: theta.reduced_norm()
w^2*a*b - y^2*a - z^2*b + x^2
```

```
>>> from sage.all import *
>>> K = QQ['x, y, z, w, a, b']; (x, y, z, w, a, b,) = K._first_ngens(6)
>>> Q = QuaternionAlgebra(a,b, names=('i', 'j', 'k',)); (i, j, k,) = Q._first_
↪ngens(3)
>>> theta = x+y*i+z*j+w*k
>>> theta.reduced_norm()
w^2*a*b - y^2*a - z^2*b + x^2
```

reduced_trace()

Return the reduced trace of self: if $\theta = x + yi + zj + wk$, then θ has reduced trace $2x$.

EXAMPLES:

```
sage: K.<x,y,z,w,a,b> = QQ[]
sage: Q.<i,j,k> = QuaternionAlgebra(a,b)
sage: theta = x+y*i+z*j+w*k
sage: theta.reduced_trace()
2*x
```

```
>>> from sage.all import *
>>> K = QQ['x, y, z, w, a, b']; (x, y, z, w, a, b,) = K._first_ngens(6)
>>> Q = QuaternionAlgebra(a,b, names=('i', 'j', 'k',)); (i, j, k,) = Q._first_
↪ngens(3)
>>> theta = x+y*i+z*j+w*k
>>> theta.reduced_trace()
2*x
```

class sage.algebras.quatalg.quaternion_algebra_element.**QuaternionAlgebraElement_generic**

Bases: *QuaternionAlgebraElement_abstract*

class sage.algebras.quatalg.quaternion_algebra_element.**QuaternionAlgebraElement_number_field**

Bases: *QuaternionAlgebraElement_abstract*

EXAMPLES:

```
sage: K.<a> = QQ[2^(1/3)]; Q.<i,j,k> = QuaternionAlgebra(K,-a,a+1)          #␣
↪needs sage.symbolic
sage: Q([a,-2/3,a^2-1/2,a*2])           # implicit doctest                  #␣
↪needs sage.symbolic
a + (-2/3)*i + (a^2 - 1/2)*j + 2*a*k
```

```
>>> from sage.all import *
>>> K = QQ[Integer(2)**(Integer(1)/Integer(3))]; (a,) = K._first_ngens(1); Q =␣
↪QuaternionAlgebra(K,-a,a+Integer(1), names=('i', 'j', 'k',)); (i, j, k,) = Q._
↪first_ngens(3)# needs sage.symbolic
>>> Q([a,-Integer(2)/Integer(3),a**Integer(2)-Integer(1)/Integer(2),
↪a*Integer(2)])           # implicit doctest                  # needs sage.
↪symbolic
a + (-2/3)*i + (a^2 - 1/2)*j + 2*a*k
```

class sage.algebras.quatalg.quaternion_algebra_element.
QuaternionAlgebraElement_rational_field

Bases: *QuaternionAlgebraElement_abstract*

coefficient_tuple()

Return 4-tuple of rational numbers which are the coefficients of this quaternion.

EXAMPLES:

```
sage: A.<i,j,k> = QuaternionAlgebra(-1,-2)
sage: (2/3 + 3/5*i + 4/3*j - 5/7*k).coefficient_tuple()
(2/3, 3/5, 4/3, -5/7)
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(1),-Integer(2), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
>>> (Integer(2)/Integer(3) + Integer(3)/Integer(5)*i + Integer(4)/
↪Integer(3)*j - Integer(5)/Integer(7)*k).coefficient_tuple()
(2/3, 3/5, 4/3, -5/7)
```

conjugate()

Return the conjugate of this quaternion.

EXAMPLES:

```
sage: A.<i,j,k> = QuaternionAlgebra(QQ,-5,-2)
sage: a = 3*i - j + 2
sage: type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_rational_field'>
sage: a.conjugate()
2 - 3*i + j
sage: b = 1 + 1/3*i + 1/5*j - 1/7*k
sage: b.conjugate()
1 - 1/3*i - 1/5*j + 1/7*k
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(QQ,-Integer(5),-Integer(2), names=('i', 'j', 'k',));
```

(continues on next page)

(continued from previous page)

```
↪ (i, j, k,) = A._first_ngens(3)
>>> a = Integer(3)*i - j + Integer(2)
>>> type(a)
<class 'sage.algebras.quatalg.quaternion_algebra_element.
↪QuaternionAlgebraElement_rational_field'>
>>> a.conjugate()
2 - 3*i + j
>>> b = Integer(1) + Integer(1)/Integer(3)*i + Integer(1)/Integer(5)*j -␣
↪Integer(1)/Integer(7)*k
>>> b.conjugate()
1 - 1/3*i - 1/5*j + 1/7*k
```

denominator()

Return the lowest common multiple of the denominators of the coefficients of i, j and k for this quaternion.

EXAMPLES:

```
sage: A = QuaternionAlgebra(QQ, -1, -1)
sage: A.<i,j,k> = QuaternionAlgebra(QQ, -1, -1)
sage: a = (1/2) + (1/5)*i + (5/12)*j + (1/13)*k
sage: a
1/2 + 1/5*i + 5/12*j + 1/13*k
sage: a.denominator()
780
sage: lcm([2, 5, 12, 13])
780
sage: (a * a).denominator()
608400
sage: (a + a).denominator()
390
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(QQ, -Integer(1), -Integer(1))
>>> A = QuaternionAlgebra(QQ, -Integer(1), -Integer(1), names=('i', 'j', 'k',
↪)); (i, j, k,) = A._first_ngens(3)
>>> a = (Integer(1)/Integer(2)) + (Integer(1)/Integer(5))*i + (Integer(5)/
↪Integer(12))*j + (Integer(1)/Integer(13))*k
>>> a
1/2 + 1/5*i + 5/12*j + 1/13*k
>>> a.denominator()
780
>>> lcm([Integer(2), Integer(5), Integer(12), Integer(13)])
780
>>> (a * a).denominator()
608400
>>> (a + a).denominator()
390
```

denominator_and_integer_coefficient_tuple()

Return 5-tuple d, x, y, z, w, where this rational quaternion is equal to $(x + yi + zj + wk)/d$ and x, y, z, w do not share a common factor with d.

OUTPUT: 5-tuple of Integers

EXAMPLES:

```
sage: A.<i,j,k>=QuaternionAlgebra(-1,-2)
sage: (2 + 3*i + 4/3*j - 5*k).denominator_and_integer_coefficient_tuple()
(3, 6, 9, 4, -15)
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(1),-Integer(2), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
>>> (Integer(2) + Integer(3)*i + Integer(4)/Integer(3)*j - Integer(5)*k).
↪denominator_and_integer_coefficient_tuple()
(3, 6, 9, 4, -15)
```

integer_coefficient_tuple()

Return the integer part of this quaternion, ignoring the common denominator.

OUTPUT: 4-tuple of Integers

EXAMPLES:

```
sage: A.<i,j,k>=QuaternionAlgebra(-1,-2)
sage: (2 + 3*i + 4/3*j - 5*k).integer_coefficient_tuple()
(6, 9, 4, -15)
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(1),-Integer(2), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
>>> (Integer(2) + Integer(3)*i + Integer(4)/Integer(3)*j - Integer(5)*k).
↪integer_coefficient_tuple()
(6, 9, 4, -15)
```

is_constant()

Return `True` if this quaternion is constant, i.e., has no i, j, or k term.

OUTPUT: boolean

EXAMPLES:

```
sage: A.<i,j,k>=QuaternionAlgebra(-1,-2)
sage: A(1/3).is_constant()
True
sage: A(-1).is_constant()
True
sage: (1+i).is_constant()
False
sage: j.is_constant()
False
```

```
>>> from sage.all import *
>>> A = QuaternionAlgebra(-Integer(1),-Integer(2), names=('i', 'j', 'k',));␣
↪(i, j, k,) = A._first_ngens(3)
>>> A(Integer(1)/Integer(3)).is_constant()
True
>>> A(-Integer(1)).is_constant()
```

(continues on next page)

(continued from previous page)

```
True
>>> (Integer(1)+i).is_constant()
False
>>> j.is_constant()
False
```

reduced_norm()

Return the reduced norm of `self`.

Given a quaternion $x + yi + zj + wk$, this is $x^2 - ay^2 - bz^2 + abw^2$.

EXAMPLES:

```
sage: K.<i,j,k> = QuaternionAlgebra(QQ, -5, -2)
sage: i.reduced_norm()
5
sage: j.reduced_norm()
2
sage: a = 1/3 + 1/5*i + 1/7*j + k
sage: a.reduced_norm()
22826/2205
```

```
>>> from sage.all import *
>>> K = QuaternionAlgebra(QQ, -Integer(5), -Integer(2), names=('i', 'j', 'k',
↪)); (i, j, k,) = K._first_ngens(3)
>>> i.reduced_norm()
5
>>> j.reduced_norm()
2
>>> a = Integer(1)/Integer(3) + Integer(1)/Integer(5)*i + Integer(1)/
↪Integer(7)*j + k
>>> a.reduced_norm()
22826/2205
```

reduced_trace()

Return the reduced trace of `self`.

This is $2x$ if `self` is $x + iy + zj + wk$.

EXAMPLES:

```
sage: K.<i,j,k> = QuaternionAlgebra(QQ, -5, -2)
sage: i.reduced_trace()
0
sage: j.reduced_trace()
0
sage: a = 1/3 + 1/5*i + 1/7*j + k
sage: a.reduced_trace()
2/3
```

```
>>> from sage.all import *
>>> K = QuaternionAlgebra(QQ, -Integer(5), -Integer(2), names=('i', 'j', 'k',
↪)); (i, j, k,) = K._first_ngens(3)
```

(continues on next page)

(continued from previous page)

```
>>> i.reduced_trace()
0
>>> j.reduced_trace()
0
>>> a = Integer(1)/Integer(3) + Integer(1)/Integer(5)*i + Integer(1)/
↪Integer(7)*j + k
>>> a.reduced_trace()
2/3
```

sage.algebras.quatalg.quaternion_algebra_element.**unpickle_QuaternionAlgebraElement_generic_v0**(**args*)

EXAMPLES:

```
sage: K.<X> = QQ[]
sage: Q.<i,j,k> = QuaternionAlgebra(Frac(K), -5,-19); z = 2/3 + i*X - X^2*j + X^
↪3*k
sage: f, t = z.__reduce__()
sage: sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_generic_v0(*t)
2/3 + X*i + (-X^2)*j + X^3*k
sage: sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_generic_v0(*t) == z
True
```

```
>>> from sage.all import *
>>> K = QQ['X']; (X,) = K._first_ngens(1)
>>> Q = QuaternionAlgebra(Frac(K), -Integer(5),-Integer(19), names=('i', 'j', 'k',
↪)); (i, j, k,) = Q._first_ngens(3); z = Integer(2)/Integer(3) + i*X -␣
↪X**Integer(2)*j + X**Integer(3)*k
>>> f, t = z.__reduce__()
>>> sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_generic_v0(*t)
2/3 + X*i + (-X^2)*j + X^3*k
>>> sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_generic_v0(*t) == z
True
```

sage.algebras.quatalg.quaternion_algebra_element.**unpickle_QuaternionAlgebraElement_number_field_v0**(**a*

EXAMPLES:

```
sage: # needs sage.symbolic
sage: K.<a> = QQ[2^(1/3)]; Q.<i,j,k> = QuaternionAlgebra(K, -3, a); z = i + j
sage: f, t = z.__reduce__()
sage: sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_number_field_v0(*t)
i + j
sage: sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_number_field_v0(*t) == z
True
```

```
>>> from sage.all import *
>>> # needs sage.symbolic
```

(continues on next page)

(continued from previous page)

```
>>> K = QQ[Integer(2)**(Integer(1)/Integer(3))]; (a,) = K._first_ngens(1); Q =␣
↪QuaternionAlgebra(K, -Integer(3), a, names=('i', 'j', 'k',)); (i, j, k,) = Q._
↪first_ngens(3); z = i + j
>>> f, t = z.__reduce__()
>>> sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_number_field_v0(*t)
i + j
>>> sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_number_field_v0(*t) == z
True
```

sage.algebras.quatalg.quaternion_algebra_element.**unpickle_QuaternionAlgebraElement_rational_field_v0**

EXAMPLES:

```
sage: Q.<i,j,k> = QuaternionAlgebra(-5,-19); a = 2/3 + i*5/7 - j*2/5 +19/2
sage: f, t = a.__reduce__()
sage: sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_rational_field_v0(*t)
61/6 + 5/7*i - 2/5*j
```

```
>>> from sage.all import *
>>> Q = QuaternionAlgebra(-Integer(5),-Integer(19), names=('i', 'j', 'k',)); (i,␣
↪j, k,) = Q._first_ngens(3); a = Integer(2)/Integer(3) + i*Integer(5)/Integer(7)␣
↪- j*Integer(2)/Integer(5) +Integer(19)/Integer(2)
>>> f, t = a.__reduce__()
>>> sage.algebras.quatalg.quaternion_algebra_element.unpickle_
↪QuaternionAlgebraElement_rational_field_v0(*t)
61/6 + 5/7*i - 2/5*j
```

CHAPTER

THREE

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

INDEX