Quadratic Forms

Release 10.1

The Sage Development Team

Aug 21, 2023

CONTENTS

CHAPTER

ONE

QUADRATIC FORMS OVERVIEW

AUTHORS:

- Jon Hanke (2007-06-19)
- Anna Haensch (2010-07-01): Formatting and ReSTification
- Simon Brandhorst (2019-10-15): *quadratic_form_from_invariants()*

sage.quadratic_forms.quadratic_form.**DiagonalQuadraticForm**(*R*, *diag*)

Return a quadratic form over R which is a sum of squares.

INPUT:

- R – ring
- diag – list/tuple of elements coercible to R

OUTPUT: quadratic form

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7]); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 0 0 0 ]
[ * 3 0 0 ]
[ * * 5 0 ]
[ * * * 7 ]
```

class sage.quadratic_forms.quadratic_form.**QuadraticForm**(*R*, *n=None*, *entries=None*, *unsafe_initialization=False*, *number_of_automorphisms=None*, *determinant=None*)

Bases: SageObject

The QuadraticForm class represents a quadratic form in n variables with coefficients in the ring R.

INPUT:

The constructor may be called in any of the following ways.

1. QuadraticForm(R, n, entries), where
   - R – ring for which the quadratic form is defined
   - n – an integer ≥ 0
   - entries – a list of $n(n+1)/2$ coefficients of the quadratic form in R (given lexicographically, or equivalently, by rows of the matrix)

2. `QuadraticForm(p)`, where

   - `p` – a homogeneous polynomial of degree 2

3. `QuadraticForm(R, n)`, where

   - `R` – a ring
   - `n` – a symmetric $n \times n$ matrix with even diagonal (relative to R)

4. `QuadraticForm(R)`, where

   - `R` – a symmetric $n \times n$ matrix with even diagonal (relative to its base ring)

If the keyword argument `unsafe_initialize` is True, then the subsequent fields may by used to force the external initialization of various fields of the quadratic form. Currently the only fields which can be set are:

- `number_of_automorphisms`
- `determinant`

OUTPUT: quadratic form

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1,2,3,4,5,6]); Q
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 2 3 ]
[ * 4 5 ]
[ * * 6 ]
```

```
sage: Q = QuadraticForm(QQ, 3, [1,2,3,4/3,5,6]); Q
Quadratic form in 3 variables over Rational Field with coefficients:
[ 1 2 3 ]
[ * 4/3 5 ]
[ * * 6 ]
sage: Q[0,0]
1
sage: Q[0,0].parent()
Rational Field
```

```
sage: Q = QuadraticForm(QQ, 7, range(28)); Q
Quadratic form in 7 variables over Rational Field with coefficients:
[ 0 1 2 3 4 5 6 ]
[ * 7 8 9 10 11 12 ]
[ * * 13 14 15 16 17 ]
[ * * * 18 19 20 21 ]
[ * * * * 22 23 24 ]
[ * * * * * 25 26 ]
[ * * * * * * 27 ]
```

```
sage: Q = QuadraticForm(QQ, 2, range(1,4))
sage: A = Matrix(ZZ, 2, 2, [-1,0,0,1])
sage: Q(A)
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 -2 ]
[ * 3 ]
```

```
sage: m = matrix(2, 2, [1,2,3,4])
sage: m + m.transpose()
[2 5]
[5 8]
sage: QuadraticForm(m + m.transpose())
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 1 5 ]
[ * 4 ]
```

```
sage: P.<x,y,z> = QQ[]
sage: p = x^2 + 2*x*y + x*z/2 + y^2 + y*z/3
sage: QuadraticForm(p)
Quadratic form in 3 variables over Rational Field with coefficients:
[ 1 2 1/2 ]
[ * 1 1/3 ]
[ * * 0 ]
```

```
sage: QuadraticForm(ZZ, m + m.transpose())
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 1 5 ]
[ * 4 ]
```

```
sage: QuadraticForm(QQ, m + m.transpose())
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 5 ]
[ * 4 ]
```

CS_genus_symbol_list(*force_recomputation=False*)

Return the list of Conway-Sloane genus symbols in increasing order of primes dividing 2*det.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3,4])
sage: Q.CS_genus_symbol_list()
[Genus symbol at 2:    [2^-2 4^1 8^1]_6, Genus symbol at 3:     1^3 3^-1]
```

GHY_mass__maximal()

Use the GHY formula to compute the mass of a (maximal?) quadratic lattice. This works for any number field.

REFERENCES:

See [GHY, Prop 7.4 and 7.5, p121] and [GY, Thrm 10.20, p25].

OUTPUT:

a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.GHY_mass__maximal()
```

Gram_det()

Return the determinant of the Gram matrix of Q.

Note: This is defined over the fraction field of the ring of the quadratic form, but is often not defined over the same ring as the quadratic form.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.Gram_det()
2
```

Gram_matrix()

Return a (symmetric) Gram matrix A for the quadratic form Q, meaning that

$$Q(x) = x^t \cdot A \cdot x,$$

defined over the base ring of Q. If this is not possible, then a `TypeError` is raised.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: A = Q.Gram_matrix(); A
[1 0 0 0]
[0 3 0 0]
[0 0 5 0]
[0 0 0 7]
sage: A.base_ring()
Integer Ring
```

Gram_matrix_rational()

Return a (symmetric) Gram matrix A for the quadratic form Q, meaning that

$$Q(x) = x^t \cdot A \cdot x,$$

defined over the fraction field of the base ring.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: A = Q.Gram_matrix_rational(); A
[1 0 0 0]
[0 3 0 0]
[0 0 5 0]
[0 0 0 7]
sage: A.base_ring()
Rational Field
```

Hessian_matrix()

Return the Hessian matrix A for which $Q(X) = (1/2)X^t \cdot A \cdot X$.

EXAMPLES:

```
sage: Q = QuadraticForm(QQ, 2, range(1,4)); Q
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 2 ]
[ * 3 ]
```

(continues on next page)

(continued from previous page)

```
sage: Q.Hessian_matrix()
[2 2]
[2 6]
sage: Q.matrix().base_ring()
Rational Field
```

Kitaoka_mass_at_2()

Return the local mass of the quadratic form when $p = 2$, according to Theorem 5.6.3 on pp108–9 of Kitaoka's Book "The Arithmetic of Quadratic Forms".

OUTPUT: a rational number > 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.Kitaoka_mass_at_2()   # WARNING:  WE NEED TO CHECK THIS CAREFULLY!
1/2
```

Pall_mass_density_at_odd_prime(p)

Return the local representation density of a form (for representing itself) defined over $\mathbf{Z}$, at some prime $p > 2$.

REFERENCES:

Pall's article "The Weight of a Genus of Positive n-ary Quadratic Forms" appearing in Proc. Symp. Pure Math. VIII (1965), pp95–105.

INPUT:

- p – a prime number > 2

OUTPUT: a rational number.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1,0,0,1,0,1])
sage: Q.Pall_mass_density_at_odd_prime(3)
[(0, Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 0 0 ]
[ * 1 0 ]
[ * * 1 ])] [(0, 3, 8)] [8/9] 8/9
8/9
```

Watson_mass_at_2()

Return the local mass of the quadratic form when $p = 2$, according to Watson's Theorem 1 of "The 2-adic density of a quadratic form" in Mathematika 23 (1976), pp 94–106.

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.Watson_mass_at_2()  # WARNING:  WE NEED TO CHECK THIS CAREFULLY!    #␣
↪optional - sage.symbolic
384
```

add_symmetric(*c*, *i*, *j*, *in_place=False*)

Perform the substitution $x_j \longmapsto x_j + c \cdot x_i$, which has the effect (on associated matrices) of symmetrically adding c times the j-th row/column to the i-th row/column.

NOTE: This is meant for compatibility with previous code, which implemented a matrix model for this class. It is used in the method *local_normal_form()*.

INPUT:

- c – an element of self.base_ring()
- i, j – integers ≥ 0

OUTPUT:

a *QuadraticForm* (by default, otherwise none)

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, range(1,7)); Q
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 2 3 ]
[ * 4 5 ]
[ * * 6 ]
sage: Q.add_symmetric(-1, 1, 0)
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 0 3 ]
[ * 3 2 ]
[ * * 6 ]
sage: Q.add_symmetric(-3/2, 2, 0)     # ERROR: -3/2 isn't in the base ring ZZ
Traceback (most recent call last):
...
RuntimeError: this coefficient cannot be coerced
to an element of the base ring for the quadratic form
```

```
sage: Q = QuadraticForm(QQ, 3, range(1,7)); Q
Quadratic form in 3 variables over Rational Field with coefficients:
[ 1 2 3 ]
[ * 4 5 ]
[ * * 6 ]
sage: Q.add_symmetric(-3/2, 2, 0)
Quadratic form in 3 variables over Rational Field with coefficients:
[ 1 2 0 ]
[ * 4 2 ]
[ * * 15/4 ]
```

adjoint()

This gives the adjoint (integral) quadratic form associated to the given form, essentially defined by taking the adjoint of the matrix.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,5])
sage: Q.adjoint()
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 5 -2 ]
[ * 1 ]
```

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q.adjoint()
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 39 2 8 ]
[ * 19 4 ]
[ * * 8 ]
```

adjoint_primitive()

Return the primitive adjoint of the quadratic form, which is the smallest discriminant integer-valued quadratic form whose matrix is a scalar multiple of the inverse of the matrix of the given quadratic form.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.adjoint_primitive()
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 3 -2 ]
[ *  1 ]
```

anisotropic_primes()

Return a list with all of the anisotropic primes of the quadratic form.

The infinite place is denoted by -1.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.anisotropic_primes()                                                  #␣
↪optional - sage.libs.pari
[2, -1]

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.anisotropic_primes()                                                  #␣
↪optional - sage.libs.pari
[2, -1]

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1,1])
sage: Q.anisotropic_primes()                                                  #␣
↪optional - sage.libs.pari
[-1]
```

antiadjoint()

This gives an (integral) form such that its adjoint is the given form.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q.adjoint().antiadjoint()
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 0 -1 ]
[ * 2 -1 ]
[ * * 5 ]
sage: Q.antiadjoint()                                                         #␣
↪optional - sage.symbolic
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
ValueError: not an adjoint
```

automorphism_group()

Return the group of automorphisms of the quadratic form.

OUTPUT: a `MatrixGroup`

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.automorphism_group()
Matrix group over Rational Field with 3 generators (
[ 0  0  1]  [1 0 0]  [ 1  0  0]
[-1  0  0]  [0 0 1]  [ 0 -1  0]
[ 0  1  0], [0 1 0], [ 0  0  1]
)
```

```
sage: DiagonalQuadraticForm(ZZ, [1,3,5,7]).automorphism_group()
Matrix group over Rational Field with 4 generators (
[-1  0  0  0]  [ 1  0  0  0]  [ 1  0  0  0]  [ 1  0  0  0]
[ 0 -1  0  0]  [ 0 -1  0  0]  [ 0  1  0  0]  [ 0  1  0  0]
[ 0  0 -1  0]  [ 0  0  1  0]  [ 0  0 -1  0]  [ 0  0  1  0]
[ 0  0  0 -1], [ 0  0  0  1], [ 0  0  0  1], [ 0  0  0 -1]
)
```

The smallest possible automorphism group has order two, since we can always change all signs:

```
sage: Q = QuadraticForm(ZZ, 3, [2, 1, 2, 2, 1, 3])
sage: Q.automorphism_group()
Matrix group over Rational Field with 1 generators (
[-1  0  0]
[ 0 -1  0]
[ 0  0 -1]
)
```

automorphisms()

Return the list of the automorphisms of the quadratic form.

OUTPUT: a list of matrices

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.number_of_automorphisms()
48
sage: 2^3 * factorial(3)
48
sage: len(Q.automorphisms())
48
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.number_of_automorphisms()
```

(continues on next page)

(continued from previous page)

```
16
sage: aut = Q.automorphisms()
sage: len(aut)
16
sage: all(Q(M) == Q for M in aut)
True

sage: Q = QuadraticForm(ZZ, 3, [2, 1, 2, 2, 1, 3])
sage: sorted(Q.automorphisms())
[
[-1  0  0]  [1 0 0]
[ 0 -1  0]  [0 1 0]
[ 0  0 -1], [0 0 1]
]
```

base_change_to(**args*, ***kwds*)

Deprecated: Use *change_ring()* instead. See github issue #35248 for details.

base_ring()

Return the ring over which the quadratic form is defined.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.base_ring()
Integer Ring
```

basiclemma(M)

Find a number represented by self and coprime to M.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [2, 1, 3])
sage: Q.basiclemma(6)
71
```

basiclemmavec(M)

Find a vector where the value of the quadratic form is coprime to M.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [2, 1, 5])
sage: Q.basiclemmavec(10)
(6, 5)
sage: Q(_)
227
```

basis_of_short_vectors(*show_lengths=False*)

Return a basis for $\mathbf{Z}^n$ made of vectors with minimal lengths $Q(v)$.

OUTPUT:

a tuple of vectors, and optionally a tuple of values for each vector.

This uses pari:qfminim.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.basis_of_short_vectors()
((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1))
sage: Q.basis_of_short_vectors(True)
(((1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0), (0, 0, 0, 1)), (1, 3, 5, 7))
```

The returned vectors are immutable:

```
sage: v = Q.basis_of_short_vectors()[0]
sage: v
(1, 0, 0, 0)
sage: v[0] = 0
Traceback (most recent call last):
...
ValueError: vector is immutable; please change a copy instead (use copy())
```

bilinear_map(v, w)

Return the value of the associated bilinear map on two vectors

Given a quadratic form Q over some base ring R with characteristic not equal to 2, this gives the image of two vectors with coefficients in R under the associated bilinear map B, given by the relation $2B(v,w) = Q(v) + Q(w) - Q(v+w)$.

INPUT:

- `v`, `w` – two vectors

OUTPUT: an element of the base ring R.

EXAMPLES:

First, an example over $\mathbf{Z}$:

```
sage: Q = QuadraticForm(ZZ, 3, [1,4,0,1,4,1])
sage: v = vector(ZZ, (1,2,0))
sage: w = vector(ZZ, (0,1,1))
sage: Q.bilinear_map(v, w)
8
```

This also works over $\mathbf{Q}$:

```
sage: Q = QuadraticForm(QQ, 2, [1/2,2,1])
sage: v = vector(QQ, (1,1))
sage: w = vector(QQ, (1/2,2))
sage: Q.bilinear_map(v, w)
19/4
```

The vectors must have the correct length:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,7,7])
sage: v = vector((1,2))
sage: w = vector((1,1,1))
sage: Q.bilinear_map(v, w)
Traceback (most recent call last):
```

(continues on next page)

(continued from previous page)

```
...
TypeError: vectors must have length 3
```

This does not work if the characteristic is 2:

```
sage: Q = DiagonalQuadraticForm(GF(2), [1,1,1])                          #␣
↪optional - sage.rings.finite_rings
sage: v = vector((1,1,1))                                                #␣
↪optional - sage.rings.finite_rings
sage: w = vector((1,1,1))                                                #␣
↪optional - sage.rings.finite_rings
sage: Q.bilinear_map(v, w)                                               #␣
↪optional - sage.rings.finite_rings
Traceback (most recent call last):
...
TypeError: not defined for rings of characteristic 2
```

change_ring(*R*)

Alters the quadratic form to have all coefficients defined over the new base ring R. Here R must be coercible to from the current base ring.

Note: This is preferable to performing an explicit coercion through the *base_ring()* method, which does not affect the individual coefficients. This is particularly useful for performing fast modular arithmetic evaluations.

INPUT:

- R – a ring

OUTPUT: quadratic form

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1]); Q
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 1 0 ]
[ * 1 ]

sage: Q1 = Q.change_ring(IntegerModRing(5)); Q1
Quadratic form in 2 variables over Ring of integers modulo 5 with coefficients:
[ 1 0 ]
[ * 1 ]

sage: Q1([35,11])
1
```

cholesky_decomposition(*bit_prec=53*)

Give the Cholesky decomposition of this quadratic form Q as a real matrix of precision `bit_prec`.

RESTRICTIONS:

Q must be given as a *QuadraticForm* defined over $\mathbf{Z}$, $\mathbf{Q}$, or some real field. If it is over some real field, then an error is raised if the precision given is not less than the defined precision of the real field defining the quadratic form!

REFERENCE:

- Cohen's "A Course in Computational Algebraic Number Theory" book, p 103.

INPUT:

- `bit_prec` – a natural number (default 53)

OUTPUT: an upper triangular real matrix of precision `bit_prec`.

Todo: If we only care about working over the real double field (RDF), then we can use the method `cholesky()` present for square matrices over that.

Note: There is a note in the original code reading

```
Finds the Cholesky decomposition of a quadratic form -- as an upper-triangular␣
↪matrix!
(It's assumed to be global, hence twice the form it refers to.)  <-- Python␣
↪revision asks:  Is this true?!? =|
```

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.cholesky_decomposition()
[ 1.00000000000000 0.000000000000000 0.000000000000000]
[0.000000000000000  1.00000000000000 0.000000000000000]
[0.000000000000000 0.000000000000000  1.00000000000000]
```

```
sage: Q = QuadraticForm(QQ, 3, range(1,7)); Q
Quadratic form in 3 variables over Rational Field with coefficients:
[ 1 2 3 ]
[ * 4 5 ]
[ * * 6 ]
sage: Q.cholesky_decomposition()
[ 1.00000000000000  1.00000000000000  1.50000000000000]
[0.000000000000000  3.00000000000000 0.333333333333333]
[0.000000000000000 0.000000000000000  3.41666666666667]
```

clifford_conductor()

This is the product of all primes where the Clifford invariant is -1

Note: For ternary forms, this is the discriminant of the quaternion algebra associated to the quadratic space (i.e. the even Clifford algebra).

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q.clifford_invariant(2)                                          #␣
↪optional - sage.libs.pari
1
sage: Q.clifford_invariant(37)                                         #␣
```

(continues on next page)

(continued from previous page)

```
↪optional - sage.libs.pari
-1
sage: Q.clifford_conductor()                                                   #␣
↪optional - sage.libs.pari
37

sage: DiagonalQuadraticForm(ZZ, [1, 1, 1]).clifford_conductor()              #␣
↪optional - sage.libs.pari
2
sage: QuadraticForm(ZZ, 3, [2, -2, 0, 2, 0, 5]).clifford_conductor()          #␣
↪optional - sage.libs.pari
30
```

For hyperbolic spaces, the Clifford conductor is 1:

```
sage: H = QuadraticForm(ZZ, 2, [0, 1, 0])
sage: H.clifford_conductor()                                                   #␣
↪optional - sage.libs.pari
1
sage: (H + H).clifford_conductor()                                             #␣
↪optional - sage.libs.pari
1
sage: (H + H + H).clifford_conductor()                                         #␣
↪optional - sage.libs.pari
1
sage: (H + H + H + H).clifford_conductor()                                     #␣
↪optional - sage.libs.pari
1
```

clifford_invariant(*p*)

This is the Clifford invariant, i.e. the class in the Brauer group of the Clifford algebra for even dimension, of the even Clifford Algebra for odd dimension.

See Lam (AMS GSM 67) p. 117 for the definition, and p. 119 for the formula relating it to the Hasse invariant.

EXAMPLES:

For hyperbolic spaces, the Clifford invariant is +1:

```
sage: H = QuadraticForm(ZZ, 2, [0, 1, 0])
sage: H.clifford_invariant(2)                                                  #␣
↪optional - sage.libs.pari
1
sage: (H + H).clifford_invariant(2)                                            #␣
↪optional - sage.libs.pari
1
sage: (H + H + H).clifford_invariant(2)                                        #␣
↪optional - sage.libs.pari
1
sage: (H + H + H + H).clifford_invariant(2)                                    #␣
↪optional - sage.libs.pari
1
```

coefficients()

Return the matrix of upper triangular coefficients, by reading across the rows from the main diagonal.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.coefficients()
[1, 2, 3]
```

complementary_subform_to_vector(v)

Find the $(n-1)$-dimensional quadratic form orthogonal to the vector v.

Note: This is usually not a direct summand!

Note: There is a minor difference in the cancellation code here (form the C++ version) since the notation `Q[i,j]` indexes coefficients of the quadratic polynomial here, not the symmetric matrix. Also, it produces a better splitting now, for the full lattice (as opposed to a sublattice in the C++ code) since we now extend v to a unimodular matrix.

INPUT:

- `v` – a list of `self.dim()` integers

OUTPUT: a *QuadraticForm* over $\mathbf{Z}$

EXAMPLES:

```
sage: Q1 = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q1.complementary_subform_to_vector([1,0,0,0])
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 7 0 0 ]
[ * 5 0 ]
[ * * 3 ]
```

```
sage: Q1.complementary_subform_to_vector([1,1,0,0])
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 7 0 0 ]
[ * 5 0 ]
[ * * 12 ]
```

```
sage: Q1.complementary_subform_to_vector([1,1,1,1])
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 880 -480 -160 ]
[ * 624 -96 ]
[ * * 240 ]
```

compute_definiteness()

Compute whether the given quadratic form is positive-definite, negative-definite, indefinite, degenerate, or the zero form.

This caches one of the following strings in `self.__definiteness_string`: "pos_def", "neg_def", "indef", "zero", "degenerate". It is called from all routines like: *is_positive_definite()*, *is_negative_definite()*, *is_indefinite()*, etc.

Note: A degenerate form is considered neither definite nor indefinite.

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1,1])
sage: Q.compute_definiteness()
sage: Q.is_positive_definite()
True
sage: Q.is_negative_definite()
False
sage: Q.is_indefinite()
False
sage: Q.is_definite()
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [])
sage: Q.compute_definiteness()
sage: Q.is_positive_definite()
True
sage: Q.is_negative_definite()
True
sage: Q.is_indefinite()
False
sage: Q.is_definite()
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,0,-1])
sage: Q.compute_definiteness()
sage: Q.is_positive_definite()
False
sage: Q.is_negative_definite()
False
sage: Q.is_indefinite()
False
sage: Q.is_definite()
False
```

compute_definiteness_string_by_determinants()

Compute the (positive) definiteness of a quadratic form by looking at the signs of all of its upper-left subdeterminants. See also `compute_definiteness()` for more documentation.

OUTPUT: string describing the definiteness

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1,1])
sage: Q.compute_definiteness_string_by_determinants()
'pos_def'
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [])
sage: Q.compute_definiteness_string_by_determinants()
'zero'
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,0,-1])
sage: Q.compute_definiteness_string_by_determinants()
'degenerate'
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-1])
sage: Q.compute_definiteness_string_by_determinants()
'indefinite'
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [-1,-1])
sage: Q.compute_definiteness_string_by_determinants()
'neg_def'
```

content()

Return the GCD of the coefficients of the quadratic form.

> **Warning:** Only works over Euclidean domains (probably just **Z**).

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1, 1])
sage: Q.matrix().gcd()
2
sage: Q.content()
1
sage: DiagonalQuadraticForm(ZZ, [1, 1]).is_primitive()
True
sage: DiagonalQuadraticForm(ZZ, [2, 4]).is_primitive()
False
sage: DiagonalQuadraticForm(ZZ, [2, 4]).primitive()
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 1 0 ]
[ * 2 ]
```

conway_cross_product_doubled_power(p)

Compute twice the power of p which evaluates the 'cross product' term in Conway's mass formula.

INPUT:

- p – a prime number > 0

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1,8))
sage: Q.conway_cross_product_doubled_power(2)
18
sage: Q.conway_cross_product_doubled_power(3)
```

(continues on next page)

(continued from previous page)

```
10
sage: Q.conway_cross_product_doubled_power(5)
6
sage: Q.conway_cross_product_doubled_power(7)
6
sage: Q.conway_cross_product_doubled_power(11)
0
sage: Q.conway_cross_product_doubled_power(13)
0
```

conway_diagonal_factor(p)

Compute the diagonal factor of Conway's p-mass.

INPUT:

- p – a prime number > 0

OUTPUT: a rational number > 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1,6))
sage: Q.conway_diagonal_factor(3)
81/256
```

conway_mass()

Compute the mass by using the Conway-Sloane mass formula.

OUTPUT: a rational number > 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.conway_mass()                                                     #␣
↪optional - sage.symbolic
1/48

sage: Q = DiagonalQuadraticForm(ZZ, [7,1,1])
sage: Q.conway_mass()                                                     #␣
↪optional - sage.symbolic
3/16

sage: Q = QuadraticForm(ZZ, 3, [7, 2, 2, 2, 0, 2]) + DiagonalQuadraticForm(ZZ,␣
↪[1])
sage: Q.conway_mass()                                                     #␣
↪optional - sage.symbolic
3/32

sage: Q = QuadraticForm(Matrix(ZZ, 2, [2,1,1,2]))
sage: Q.conway_mass()                                                     #␣
↪optional - sage.symbolic
1/12
```

conway_octane_of_this_unimodular_Jordan_block_at_2()

Determines the 'octane' of this full unimodular Jordan block at the prime $p = 2$. This is an invariant defined (mod 8), ad.

This assumes that the form is given as a block diagonal form with unimodular blocks of size ≤ 2 and the 1×1 blocks are all in the upper leftmost position.

OUTPUT: an integer $0 \leq x \leq 7$

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.conway_octane_of_this_unimodular_Jordan_block_at_2()
0
sage: Q = DiagonalQuadraticForm(ZZ, [1,5,13])
sage: Q.conway_octane_of_this_unimodular_Jordan_block_at_2()
3
sage: Q = DiagonalQuadraticForm(ZZ, [3,7,13])
sage: Q.conway_octane_of_this_unimodular_Jordan_block_at_2()
7
```

conway_p_mass(p)

Compute Conway's p-mass.

INPUT:

- p – a prime number > 0

OUTPUT: a rational number > 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1, 6))
sage: Q.conway_p_mass(2)
16/3
sage: Q.conway_p_mass(3)
729/256
```

conway_species_list_at_2()

Return an integer called the 'species' which determines the type of the orthogonal group over the finite field $\mathbf{F}_p$.

This assumes that the given quadratic form is a unimodular Jordan block at an odd prime p. When the dimension is odd then this number is always positive, otherwise it may be positive or negative.

Note: The species of a zero dimensional form is always 0+, so we interpret the return value of zero as positive here! =)

OUTPUT: a list of integers

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1,10))
sage: Q.conway_species_list_at_2()
[1, 5, 1, 1, 1, 1]
```

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1,8))
sage: Q.conway_species_list_at_2()
[1, 3, 1, 1, 1]
```

conway_species_list_at_odd_prime(p)

Return an integer called the 'species' which determines the type of the orthogonal group over the finite field $\mathbf{F}_p$.

This assumes that the given quadratic form is a unimodular Jordan block at an odd prime p. When the dimension is odd then this number is always positive, otherwise it may be positive or negative (or zero, but that is considered positive by convention).

INPUT:

- p – a positive prime number

OUTPUT: a list of integers

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1,10))
sage: Q.conway_species_list_at_odd_prime(3)
[6, 2, 1]
```

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1,8))
sage: Q.conway_species_list_at_odd_prime(3)
[5, 2]
sage: Q.conway_species_list_at_odd_prime(5)
[-6, 1]
```

conway_standard_mass()

Return the infinite product of the standard mass factors.

OUTPUT: a rational number > 0

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [2, -2, 0, 3, -5, 4])
sage: Q.conway_standard_mass()                                              #␣
↪optional - sage.symbolic
1/6
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.conway_standard_mass()                                              #␣
↪optional - sage.symbolic
1/6
```

conway_standard_p_mass(p)

Compute the standard (generic) Conway-Sloane p-mass.

INPUT:

- p – a prime number > 0

OUTPUT: a rational number > 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.conway_standard_p_mass(2)
2/3
```

conway_type_factor()

This is a special factor only present in the mass formula when $p = 2$.

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, range(1,8))
sage: Q.conway_type_factor()
4
```

count_congruence_solutions(*p*, *k*, *m*, *zvec*, *nzvec*)

Count all solutions of $Q(x) = m \pmod{p^k}$ satisfying the additional congruence conditions described in `QuadraticForm.count_congruence_solutions_as_vector()`.

INPUT:

- `p` – prime number > 0
- `k` – an integer > 0
- `m` – an integer (depending only on mod p^k)
- `zvec`, `nzvec` – lists of integers in `range(self.dim())`, or `None`

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.count_congruence_solutions(3, 1, 0, None, None)
15
```

count_congruence_solutions__bad_type(*p*, *k*, *m*, *zvec*, *nzvec*)

Count the bad-type solutions of $Q(x) = m \pmod{p^k}$ satisfying the additional congruence conditions described in `QuadraticForm.count_congruence_solutions_as_vector()`.

INPUT:

- `p` – prime number > 0
- `k` – an integer > 0
- `m` – an integer (depending only on mod p^k)
- `zvec`, `nzvec` – lists of integers up to dim(Q)

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.count_congruence_solutions__bad_type(3, 1, 0, None, None)
2
```

count_congruence_solutions__bad_type_I(*p*, *k*, *m*, *zvec*, *nzvec*)

Count the bad-typeI solutions of $Q(x) = m \pmod{p^k}$ satisfying the additional congruence conditions described in `QuadraticForm.count_congruence_solutions_as_vector()`.

INPUT:

- `p` – prime number > 0
- `k` – an integer > 0
- `m` – an integer (depending only on mod p^k)

- zvec, nzvec – lists of integers up to $\dim(Q)$

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.count_congruence_solutions__bad_type_I(3, 1, 0, None, None)
0
```

count_congruence_solutions__bad_type_II(*p*, *k*, *m*, *zvec*, *nzvec*)

Count the bad-typeII solutions of $Q(x) = m \pmod{p^k}$ satisfying the additional congruence conditions described in *QuadraticForm.count_congruence_solutions_as_vector()*.

INPUT:

- p – prime number > 0
- k – an integer > 0
- m – an integer (depending only on mod p^k)
- zvec, nzvec – lists of integers up to $\dim(Q)$

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.count_congruence_solutions__bad_type_II(3, 1, 0, None, None)
2
```

count_congruence_solutions__good_type(*p*, *k*, *m*, *zvec*, *nzvec*)

Count the good-type solutions of $Q(x) = m \pmod{p^k}$ satisfying the additional congruence conditions described in *QuadraticForm.count_congruence_solutions_as_vector()*.

INPUT:

- p – prime number > 0
- k – an integer > 0
- m – an integer (depending only on mod p^k)
- zvec, nzvec – lists of integers up to $\dim(Q)$

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.count_congruence_solutions__good_type(3, 1, 0, None, None)
12
```

count_congruence_solutions__zero_type(*p*, *k*, *m*, *zvec*, *nzvec*)

Count the zero-type solutions of $Q(x) = m \pmod{p^k}$ satisfying the additional congruence conditions described in *QuadraticForm.count_congruence_solutions_as_vector()*.

INPUT:

- p – prime number > 0
- k – an integer > 0
- m – an integer (depending only on mod p^k)
- zvec, nzvec – lists of integers up to $\dim(Q)$

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.count_congruence_solutions__zero_type(3, 1, 0, None, None)
1
```

count_congruence_solutions_as_vector(*p*, *k*, *m*, *zvec*, *nzvec*)

Return the number of integer solution vectors x satisfying the congruence $Q(x) = m \pmod{p^k}$ of each solution type (i.e. All, Good, Zero, Bad, BadI, BadII) which satisfy the additional congruence conditions of having certain coefficients = 0 (mod p) and certain collections of coefficients not congruent to the zero vector (mod p).

A solution vector x satisfies the additional congruence conditions specified by `zvec` and `nzvec` (and therefore is counted) iff both of the following conditions hold:

1. $x_i = 0$ (mod p) for all i in `zvec`
2. $x_i \neq 0$ (mod p) for all i in `nzvec`

REFERENCES:

See Hanke's (????) paper "Local Densities and explicit bounds…", p??? for the definitions of the solution types and congruence conditions.

INPUT:

- `p` – prime number > 0
- `k` – an integer > 0
- `m` – an integer (depending only on mod p^k)
- `zvec`, `nzvec` – lists of integers in `range(self.dim())`, or `None`

OUTPUT:

a list of six integers ≥ 0 representing the solution types: [All, Good, Zero, Bad, BadI, BadII]

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.count_congruence_solutions_as_vector(3, 1, 1, [], [])
[0, 0, 0, 0, 0, 0]
sage: Q.count_congruence_solutions_as_vector(3, 1, 1, None, [])
[0, 0, 0, 0, 0, 0]
sage: Q.count_congruence_solutions_as_vector(3, 1, 1, [], None)
[6, 6, 0, 0, 0, 0]
sage: Q.count_congruence_solutions_as_vector(3, 1, 1, None, None)
[6, 6, 0, 0, 0, 0]
sage: Q.count_congruence_solutions_as_vector(3, 1, 2, None, None)
[6, 6, 0, 0, 0, 0]
sage: Q.count_congruence_solutions_as_vector(3, 1, 0, None, None)
[15, 12, 1, 2, 0, 2]
```

count_modp_solutions__by_Gauss_sum(*p*, *m*)

Return the number of solutions of $Q(x) = m \pmod{p}$ of a non-degenerate quadratic form over the finite field $\mathbf{Z}/p\mathbf{Z}$, where p is a prime number > 2.

Note: We adopt the useful convention that a zero-dimensional quadratic form has exactly one solution always (i.e. the empty vector).

These are defined in Table 1 on p363 of Hanke's "Local Densities…" paper.

INPUT:

- p – a prime number > 2
- m – an integer

OUTPUT: an integer ≥ 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: [Q.count_modp_solutions__by_Gauss_sum(3, m)  for m in range(3)]
[9, 6, 12]

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,2])
sage: [Q.count_modp_solutions__by_Gauss_sum(3, m)  for m in range(3)]
[9, 12, 6]
```

delta()

This is the omega of the adjoint form, which is the same as the omega of the reciprocal form.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,37])
sage: Q.delta()
148
```

det()

Return the determinant of the Gram matrix of $2 \cdot Q$, or equivalently the determinant of the Hessian matrix of Q.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.det()
8
```

dim()

Return the number of variables of the quadratic form.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.dim()
2
sage: parent(Q.dim())
Integer Ring
sage: Q = QuadraticForm(Q.matrix())
sage: Q.dim()
2
sage: parent(Q.dim())
Integer Ring
```

disc()

Return the discriminant of the quadratic form, defined as

- $(-1)^n \det(B)$ for even dimension $2n$
- $\det(B)/2$ for odd dimension

where $2Q(x) = x^t Bx$.

This agrees with the usual discriminant for binary and ternary quadratic forms.

EXAMPLES:

```
sage: DiagonalQuadraticForm(ZZ, [1]).disc()
1
sage: DiagonalQuadraticForm(ZZ, [1,1]).disc()
-4
sage: DiagonalQuadraticForm(ZZ, [1,1,1]).disc()
4
sage: DiagonalQuadraticForm(ZZ, [1,1,1,1]).disc()
16
```

discrec()

Return the discriminant of the reciprocal form.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,37])
sage: Q.disc()
148
sage: Q.discrec()
5476
sage: [4 * 37, 4 * 37^2]
[148, 5476]
```

divide_variable(*c*, *i*, *in_place=False*)

Replace the variables x_i by $(x_i)/c$ in the quadratic form (replacing the original form if the `in_place` flag is True).

Here c must be an element of the base ring defining the quadratic form, and the division must be defined in the base ring.

INPUT:

- `c` – an element of `self.base_ring()`
- `i` – an integer ≥ 0

OUTPUT:

a *QuadraticForm* (by default, otherwise none)

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,9,5,7])
sage: Q.divide_variable(3, 1)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 0 0 0 ]
[ * 1 0 0 ]
[ * * 5 0 ]
[ * * * 7 ]
```

elementary_substitution(*c*, *i*, *j*, *in_place=False*)

Perform the substitution $x_i \longmapsto x_i + c \cdot x_j$ (replacing the original form if the `in_place` flag is True).

INPUT:

- `c` – an element of `self.base_ring()`
- `i`, `j` – integers ≥ 0

OUTPUT:

a `QuadraticForm` (by default, otherwise none)

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, range(1,11)); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 2 3 4 ]
[ * 5 6 7 ]
[ * * 8 9 ]
[ * * * 10 ]

sage: Q.elementary_substitution(c=1, i=0, j=3)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 2 3 6 ]
[ * 5 6 9 ]
[ * * 8 12 ]
[ * * * 15 ]
```

```
sage: R = QuadraticForm(ZZ, 4, range(1,11)); R
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 2 3 4 ]
[ * 5 6 7 ]
[ * * 8 9 ]
[ * * * 10 ]
```

```
sage: M = Matrix(ZZ, 4, 4, [1,0,0,1, 0,1,0,0, 0,0,1,0, 0,0,0,1]); M
[1 0 0 1]
[0 1 0 0]
[0 0 1 0]
[0 0 0 1]
sage: R(M)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 2 3 6 ]
[ * 5 6 9 ]
[ * * 8 12 ]
[ * * * 15 ]
```

extract_variables(*QF*, *var_indices*)

Extract the variables (in order) whose indices are listed in `var_indices`, to give a new quadratic form.

INPUT:

- `var_indices` – a list of integers ≥ 0

OUTPUT: a `QuadraticForm`

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, range(10)); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 0 1 2 3 ]
[ * 4 5 6 ]
[ * * 7 8 ]
[ * * * 9 ]
sage: Q.extract_variables([1,3])
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 4 6 ]
[ * 9 ]
```

find_entry_with_minimal_scale_at_prime(p)

Finds the entry of the quadratic form with minimal scale at the prime p, preferring diagonal entries in case of a tie. (I.e. If we write the quadratic form as a symmetric matrix M, then this entry `M[i,j]` has the minimal valuation at the prime p.)

Note: This answer is independent of the kind of matrix (Gram or Hessian) associated to the form.

INPUT:

- p – a prime number > 0

OUTPUT: a pair of integers ≥ 0

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [6, 2, 20]); Q
Quadratic form in 2 variables over Integer Ring with coefficients:
  [ 6 2 ]
  [ * 20 ]
sage: Q.find_entry_with_minimal_scale_at_prime(2)
(0, 1)
sage: Q.find_entry_with_minimal_scale_at_prime(3)
(1, 1)
sage: Q.find_entry_with_minimal_scale_at_prime(5)
(0, 0)
```

find_p_neighbor_from_vec(p, y)

Return the p-neighbor of `self` defined by `y`.

Let (L, q) be a lattice with $b(L, L) \subseteq \mathbf{Z}$ which is maximal at p. Let $y \in L$ with $b(y, y) \in p^2\mathbf{Z}$ then the p-neighbor of L at y is given by $\mathbf{Z}y/p + L_y$ where $L_y = \{x \in L | b(x, y) \in p\mathbf{Z}\}$ and $b(x, y) = q(x + y) - q(x) - q(y)$ is the bilinear form associated to q.

INPUT:

- p – a prime number
- y – a vector with $q(y) \in p\mathbf{Z}$
- odd – (default: `False`) if $p = 2$, return also odd neighbors

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: v = vector([0,2,1,1])
```

(continues on next page)

(continued from previous page)

```
sage: X = Q.find_p_neighbor_from_vec(3, v); X                               #␣
↪optional - sage.libs.pari
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 0 0 0 ]
[ * 1 4 4 ]
[ * * 5 12 ]
[ * * * 9 ]
```

Since the base ring and the domain are not yet separate, for rational, half integral forms we just pretend the base ring is **Z**:

```
sage: Q = QuadraticForm(QQ, matrix.diagonal([1,1,1,1]))
sage: v = vector([1,1,1,1])
sage: Q.find_p_neighbor_from_vec(2, v)                                      #␣
↪optional - sage.libs.pari
Quadratic form in 4 variables over Rational Field with coefficients:
[ 1/2 1 1 1 ]
[ * 1 1 2 ]
[ * * 1 2 ]
[ * * * 2 ]
```

find_primitive_p_divisible_vector__next(*p*, *v=None*)

Find the next p-primitive vector (up to scaling) in L/pL whose value is p-divisible, where the last vector returned was v. For an initial call, no v needs to be passed.

Return vectors whose last non-zero entry is normalized to 0 or 1 (so no lines are counted repeatedly). The ordering is by increasing the first non-normalized entry. If we have tested all (lines of) vectors, then return None.

OUTPUT: vector or None

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [10,1,4])
sage: v = Q.find_primitive_p_divisible_vector__next(5); v
(1, 1)
sage: v = Q.find_primitive_p_divisible_vector__next(5, v); v
(1, 0)
sage: v = Q.find_primitive_p_divisible_vector__next(5, v); v
sage: Q = QuadraticForm(QQ, matrix.diagonal([1,1,1,1]))
sage: v = Q.find_primitive_p_divisible_vector__next(2)
sage: Q(v)
2
```

find_primitive_p_divisible_vector__random(*p*)

Find a random p-primitive vector in L/pL whose value is p-divisible.

Note: Since there are about $p^{(n-2)}$ of these lines, we have a $1/p$ chance of randomly finding an appropriate vector.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [10,1,4])
sage: v = Q.find_primitive_p_divisible_vector__random(5)
sage: tuple(v) in ((1, 0), (1, 1), (2, 0), (2, 2), (3, 0), (3, 3), (4, 0), (4,␣
→4))
True
sage: 5.divides(Q(v))
True
sage: Q = QuadraticForm(QQ, matrix.diagonal([1,1,1,1]))
sage: v = Q.find_primitive_p_divisible_vector__random(2)
sage: Q(v)
2
```

static from_polynomial(*poly*)

Construct a `QuadraticForm` from a multivariate polynomial. Inverse of `polynomial()`.

EXAMPLES:

```
sage: R.<x,y,z> = ZZ[]
sage: f = 5*x^2 - x*z - 3*y*z - 2*y^2 + 9*z^2
sage: Q = QuadraticForm.from_polynomial(f); Q
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 5 0 -1 ]
[ * -2 -3 ]
[ * * 9 ]
sage: Q.polynomial()
5*x0^2 - 2*x1^2 - x0*x2 - 3*x1*x2 + 9*x2^2
sage: Q.polynomial()(R.gens()) == f
True
```

The method fails if the given polynomial is not a quadratic form:

```
sage: QuadraticForm.from_polynomial(x^3 + x*z + 5*y^2)
Traceback (most recent call last):
...
ValueError: polynomial has monomials of degree != 2
```

gcd()

Return the greatest common divisor of the coefficients of the quadratic form (as a polynomial).

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, range(1, 21, 2))
sage: Q.gcd()
1

sage: Q = QuadraticForm(ZZ, 4, range(0, 20, 2))
sage: Q.gcd()
2
```

static genera(*sig_pair*, *determinant*, *max_scale=None*, *even=False*)

Return a list of all global genera with the given conditions.

Here a genus is called global if it is non-empty.

INPUT:

- `sig_pair` – a pair of non-negative integers giving the signature
- `determinant` – an integer; the sign is ignored
- `max_scale` – (default: `None`) an integer; the maximum scale of a jordan block
- `even` – boolean (default: `False`)

OUTPUT:

A list of all (non-empty) global genera with the given conditions.

EXAMPLES:

```
sage: QuadraticForm.genera((4,0), 125, even=True)
[Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^1 5^3, Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^-2 5^1 25^-1, Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^2 5^1 25^1, Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^3 125^1]
```

global_genus_symbol()

Return the genus of two times a quadratic form over **Z**.

These are defined by a collection of local genus symbols (a la Chapter 15 of Conway-Sloane [CS1999]), and a signature.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3,4])
sage: Q.global_genus_symbol()
Genus of
[2 0 0 0]
[0 4 0 0]
[0 0 6 0]
[0 0 0 8]
Signature:  (4, 0)
Genus symbol at 2:    [2^-2 4^1 8^1]_6
Genus symbol at 3:     1^3 3^-1
```

```
sage: Q = QuadraticForm(ZZ, 4, range(10))
sage: Q.global_genus_symbol()
Genus of
[ 0  1  2  3]
[ 1  8  5  6]
```

(continues on next page)

(continued from previous page)

```
[ 2  5 14  8]
[ 3  6  8 18]
Signature:  (3, 1)
Genus symbol at 2:    1^-4
Genus symbol at 563:     1^3 563^-1
```

has_equivalent_Jordan_decomposition_at_prime(*other*, *p*)

Determine if the given quadratic form has a Jordan decomposition equivalent to that of `self`.

INPUT:

- `other` – a `QuadraticForm`

OUTPUT: boolean

EXAMPLES:

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 0, -1, 1, 0, 3])
sage: Q2 = QuadraticForm(ZZ, 3, [1, 0, 0, 2, -2, 6])
sage: Q3 = QuadraticForm(ZZ, 3, [1, 0, 0, 1, 0, 11])
sage: [Q1.level(), Q2.level(), Q3.level()]
[44, 44, 44]
sage: Q1.has_equivalent_Jordan_decomposition_at_prime(Q2, 2)                    #␣
↪optional - sage.libs.pari
False
sage: Q1.has_equivalent_Jordan_decomposition_at_prime(Q2, 11)                   #␣
↪optional - sage.libs.pari
False
sage: Q1.has_equivalent_Jordan_decomposition_at_prime(Q3, 2)                    #␣
↪optional - sage.libs.pari
False
sage: Q1.has_equivalent_Jordan_decomposition_at_prime(Q3, 11)                   #␣
↪optional - sage.libs.pari
True
sage: Q2.has_equivalent_Jordan_decomposition_at_prime(Q3, 2)                    #␣
↪optional - sage.libs.pari
True
sage: Q2.has_equivalent_Jordan_decomposition_at_prime(Q3, 11)                   #␣
↪optional - sage.libs.pari
False
```

has_integral_Gram_matrix()

Return whether the quadratic form has an integral Gram matrix (with respect to its base ring).

A warning is issued if the form is defined over a field, since in that case the return is trivially true.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [7,8,9])
sage: Q.has_integral_Gram_matrix()
True
```

```
sage: Q = QuadraticForm(ZZ, 2, [4,5,6])
sage: Q.has_integral_Gram_matrix()
False
```

hasse_conductor()

This is the product of all primes where the Hasse invariant equals -1

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q.hasse_invariant(2)                                                  #␣
↪optional - sage.libs.pari
-1
sage: Q.hasse_invariant(37)                                                 #␣
↪optional - sage.libs.pari
-1
sage: Q.hasse_conductor()                                                   #␣
↪optional - sage.libs.pari
74

sage: DiagonalQuadraticForm(ZZ, [1, 1, 1]).hasse_conductor()               #␣
↪optional - sage.libs.pari
1
sage: QuadraticForm(ZZ, 3, [2, -2, 0, 2, 0, 5]).hasse_conductor()           #␣
↪optional - sage.libs.pari
10
```

hasse_invariant(*p*)

Compute the Hasse invariant at a prime p or at infinity, as given on p55 of Cassels's book. If Q is diagonal with coefficients a_i, then the (Cassels) Hasse invariant is given by

$$c_p = \prod_{i<j} (a_i, a_j)_p$$

where $(a, b)_p$ is the Hilbert symbol at p. The underlying quadratic form must be non-degenerate over $\mathbf{Q}_p$ for this to make sense.

> **Warning:** This is different from the O'Meara Hasse invariant, which allows $i \leq j$ in the product. That is given by the method *hasse_invariant__OMeara()*.

Note: We should really rename this `hasse_invariant__Cassels`, and set *hasse_invariant()* as a front-end to it.

INPUT:

- p – a prime number > 0 or -1 for the infinite place

OUTPUT: 1 or -1

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.rational_diagonal_form()
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 0 ]
[ * 2 ]
sage: [Q.hasse_invariant(p) for p in prime_range(20)]                       #␣
```

(continues on next page)

(continued from previous page)

```
↪optional - sage.libs.pari
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]          #␣
↪optional - sage.libs.pari
[1, 1, 1, 1, 1, 1, 1, 1]
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-1])
sage: [Q.hasse_invariant(p) for p in prime_range(20)]                  #␣
↪optional - sage.libs.pari
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]          #␣
↪optional - sage.libs.pari
[-1, 1, 1, 1, 1, 1, 1, 1]
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-1,5])
sage: [Q.hasse_invariant(p) for p in prime_range(20)]                  #␣
↪optional - sage.libs.pari
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]          #␣
↪optional - sage.libs.pari
[-1, 1, 1, 1, 1, 1, 1, 1]
```

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 23)                                    #␣
↪optional - sage.rings.number_field
sage: Q = DiagonalQuadraticForm(K, [-a, a + 2])                        #␣
↪optional - sage.rings.number_field
sage: [Q.hasse_invariant(p) for p in K.primes_above(19)]               #␣
↪optional - sage.rings.number_field
[-1, 1]
```

hasse_invariant__OMeara(p)

Compute the O'Meara Hasse invariant at a prime p.

This is defined on p167 of O'Meara's book. If Q is diagonal with coefficients a_i, then the (Cassels) Hasse invariant is given by

$$c_p = \prod_{i \leq j} (a_i, a_j)_p$$

where $(a, b)_p$ is the Hilbert symbol at p.

> **Warning:** This is different from the (Cassels) Hasse invariant, which only allows $i < j$ in the product. That is given by the method hasse_invariant(p).

INPUT:

- p – a prime number > 0 or -1 for the infinite place

OUTPUT: 1 or -1

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: Q.rational_diagonal_form()
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 0 ]
[ * 2 ]
sage: [Q.hasse_invariant(p) for p in prime_range(20)]                    #␣
↪optional - sage.libs.pari
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]            #␣
↪optional - sage.libs.pari
[1, 1, 1, 1, 1, 1, 1, 1]
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-1])
sage: [Q.hasse_invariant(p) for p in prime_range(20)]                    #␣
↪optional - sage.libs.pari
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]            #␣
↪optional - sage.libs.pari
[-1, 1, 1, 1, 1, 1, 1, 1]
```

```
sage: Q = DiagonalQuadraticForm(ZZ,[1,-1,-1])
sage: [Q.hasse_invariant(p) for p in prime_range(20)]                    #␣
↪optional - sage.libs.pari
[-1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]            #␣
↪optional - sage.libs.pari
[-1, 1, 1, 1, 1, 1, 1, 1]
```

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 23)                                      #␣
↪optional - sage.rings.number_field
sage: Q = DiagonalQuadraticForm(K, [-a, a + 2])                          #␣
↪optional - sage.rings.number_field
sage: [Q.hasse_invariant__OMeara(p) for p in K.primes_above(19)]         #␣
↪optional - sage.rings.number_field
[1, 1]
```

is_adjoint()

Determine if the given form is the adjoint of another form.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q.is_adjoint()                                                     #␣
↪optional - sage.symbolic
False
sage: Q.adjoint().is_adjoint()
True
```

is_anisotropic(*p*)

Check if the quadratic form is anisotropic over the p-adic numbers $\mathbf{Q}_p$ or $\mathbf{R}$.

INPUT:

- p – a prime number > 0 or -1 for the infinite place

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1])
sage: Q.is_anisotropic(2)                                               #␣
↪optional - sage.libs.pari
True
sage: Q.is_anisotropic(3)                                               #␣
↪optional - sage.libs.pari
True
sage: Q.is_anisotropic(5)                                               #␣
↪optional - sage.libs.pari
False
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-1])
sage: Q.is_anisotropic(2)                                               #␣
↪optional - sage.libs.pari
False
sage: Q.is_anisotropic(3)                                               #␣
↪optional - sage.libs.pari
False
sage: Q.is_anisotropic(5)                                               #␣
↪optional - sage.libs.pari
False
```

```
sage: [DiagonalQuadraticForm(ZZ,                                        #␣
↪optional - sage.libs.pari
....:                         [1, -least_quadratic_nonresidue(p)]).is_
↪anisotropic(p)
....:  for p in prime_range(3, 30)]
[True, True, True, True, True, True, True, True, True]
```

```
sage: [DiagonalQuadraticForm(ZZ, [1, -least_quadratic_nonresidue(p),    #␣
↪optional - sage.libs.pari
....:                              p, -p*least_quadratic_nonresidue(p)]).is_
↪anisotropic(p)
....:  for p in prime_range(3, 30)]
[True, True, True, True, True, True, True, True, True]
```

is_definite()

Determines if the given quadratic form is (positive or negative) definite.

Note: A degenerate form is considered neither definite nor indefinite.

Note: The zero-dimensional form is considered indefinite.

OUTPUT: boolean – True or False

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [-1,-3,-5])
sage: Q.is_definite()
True

sage: Q = DiagonalQuadraticForm(ZZ, [1,-3,5])
sage: Q.is_definite()
False
```

is_even(*allow_rescaling_flag=True*)

Return true iff after rescaling by some appropriate factor, the form represents no odd integers. For more details, see `parity()`.

Requires that Q is defined over $\mathbf{Z}$.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1, 0, 1])
sage: Q.is_even()
False
sage: Q = QuadraticForm(ZZ, 2, [1, 1, 1])
sage: Q.is_even()
True
```

is_globally_equivalent_to(*other*, *return_matrix=False*)

Determine if the current quadratic form is equivalent to the given form over $\mathbf{Z}$.

If `return_matrix` is True, then we return the transformation matrix M so that `self(M) == other`.

INPUT:

- `self`, `other` – positive definite integral quadratic forms
- `return_matrix` – (boolean, default `False`) return the transformation matrix instead of a boolean

OUTPUT:

- if `return_matrix` is `False`: a boolean
- if `return_matrix` is `True`: either `False` or the transformation matrix

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: M = Matrix(ZZ, 4, 4, [1,2,0,0, 0,1,0,0, 0,0,1,0, 0,0,0,1])
sage: Q1 = Q(M)
sage: Q.is_globally_equivalent_to(Q1)
True
sage: MM = Q.is_globally_equivalent_to(Q1, return_matrix=True)
sage: Q(MM) == Q1
True
```

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q2 = QuadraticForm(ZZ, 3, [2, 1, 2, 2, 1, 3])
sage: Q3 = QuadraticForm(ZZ, 3, [8, 6, 5, 3, 4, 2])
sage: Q1.is_globally_equivalent_to(Q2)
False
sage: Q1.is_globally_equivalent_to(Q2, return_matrix=True)
```

(continues on next page)

(continued from previous page)

```
False
sage: Q1.is_globally_equivalent_to(Q3)
True
sage: M = Q1.is_globally_equivalent_to(Q3, True); M
[-1 -1  0]
[ 1  1  1]
[-1  0  0]
sage: Q1(M) == Q3
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1, -1])
sage: Q.is_globally_equivalent_to(Q)
Traceback (most recent call last):
...
ValueError: not a definite form in QuadraticForm.is_globally_equivalent_to()
```

ALGORITHM: this uses the PARI function pari:qfisom, implementing an algorithm by Plesken and Souvignier.

is_hyperbolic(p)

Check if the quadratic form is a sum of hyperbolic planes over the p-adic numbers $\mathbf{Q}_p$ or over the real numbers $\mathbf{R}$.

REFERENCES:

This criterion follows from Cassels's "Rational Quadratic Forms":

- local invariants for hyperbolic plane (Lemma 2.4, p58)
- direct sum formulas (Lemma 2.3, p58)

INPUT:

- p – a prime number > 0 or -1 for the infinite place

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1])
sage: Q.is_hyperbolic(-1)                                                    #␣
↪optional - sage.libs.pari
False
sage: Q.is_hyperbolic(2)                                                     #␣
↪optional - sage.libs.pari
False
sage: Q.is_hyperbolic(3)                                                     #␣
↪optional - sage.libs.pari
False
sage: Q.is_hyperbolic(5)     # Here -1 is a square, so it's true.           #␣
↪optional - sage.libs.pari
True
sage: Q.is_hyperbolic(7)                                                     #␣
↪optional - sage.libs.pari
False
sage: Q.is_hyperbolic(13)    # Here -1 is a square, so it's true.           #␣
```

(continues on next page)

(continued from previous page)

```
↪optional - sage.libs.pari
True
```

is_indefinite()

Determines if the given quadratic form is indefinite.

Note: A degenerate form is considered neither definite nor indefinite.

Note: The zero-dimensional form is not considered indefinite.

OUTPUT: boolean – True or False

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [-1,-3,-5])
sage: Q.is_indefinite()
False
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-3,5])
sage: Q.is_indefinite()
True
```

is_isotropic(p)

Checks if Q is isotropic over the p-adic numbers $\mathbf{Q}_p$ or $\mathbf{R}$.

INPUT:

- p – a prime number > 0 or -1 for the infinite place

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1])
sage: Q.is_isotropic(2)                                                    #␣
↪optional - sage.libs.pari
False
sage: Q.is_isotropic(3)                                                    #␣
↪optional - sage.libs.pari
False
sage: Q.is_isotropic(5)                                                    #␣
↪optional - sage.libs.pari
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-1])
sage: Q.is_isotropic(2)                                                    #␣
↪optional - sage.libs.pari
True
sage: Q.is_isotropic(3)                                                    #␣
↪optional - sage.libs.pari
True
```

(continues on next page)

(continued from previous page)

```
sage: Q.is_isotropic(5)                                                     #␣
↪optional - sage.libs.pari
True
```

```
sage: [DiagonalQuadraticForm(ZZ,                                            #␣
↪optional - sage.libs.pari
....:                         [1, -least_quadratic_nonresidue(p)]).is_
↪isotropic(p)
....:  for p in prime_range(3, 30)]
[False, False, False, False, False, False, False, False, False]
```

```
sage: [DiagonalQuadraticForm(ZZ, [1, -least_quadratic_nonresidue(p),        #␣
↪optional - sage.libs.pari
....:                              p, -p*least_quadratic_nonresidue(p)]).is_
↪isotropic(p)
....:  for p in prime_range(3, 30)]
[False, False, False, False, False, False, False, False, False]
```

is_locally_equivalent_to(*other*, *check_primes_only=False*, *force_jordan_equivalence_test=False*)

Determine if the current quadratic form (defined over $\mathbf{Z}$) is locally equivalent to the given form over the real numbers and the p-adic integers for every prime p.

This works by comparing the local Jordan decompositions at every prime, and the dimension and signature at the real place.

INPUT:

- `other` – a `QuadraticForm`

OUTPUT: boolean

EXAMPLES:

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q2 = QuadraticForm(ZZ, 3, [2, 1, 2, 2, 1, 3])
sage: Q1.is_globally_equivalent_to(Q2)
False
sage: Q1.is_locally_equivalent_to(Q2)
True
```

is_locally_represented_number(*m*)

Determine if the rational number m is locally represented by the quadratic form.

INPUT:

- `m` – an integer

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.is_locally_represented_number(2)
True
sage: Q.is_locally_represented_number(7)
False
```

(continues on next page)

(continued from previous page)

```
sage: Q.is_locally_represented_number(-1)
False
sage: Q.is_locally_represented_number(28)
False
sage: Q.is_locally_represented_number(0)
True
```

is_locally_represented_number_at_place(m, p)

Determine if the rational number m is locally represented by the quadratic form at the (possibly infinite) prime p.

INPUT:

- `m` – an integer
- `p` – a prime number > 0 or 'infinity'

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.is_locally_represented_number_at_place(7, infinity)
True
sage: Q.is_locally_represented_number_at_place(7, 2)
False
sage: Q.is_locally_represented_number_at_place(7, 3)
True
sage: Q.is_locally_represented_number_at_place(7, 5)
True
sage: Q.is_locally_represented_number_at_place(-1, infinity)
False
sage: Q.is_locally_represented_number_at_place(-1, 2)
False
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1,-1])
sage: Q.is_locally_represented_number_at_place(7, infinity)      # long time (8.
↪5 s)
True
sage: Q.is_locally_represented_number_at_place(7, 2)             # long time
True
sage: Q.is_locally_represented_number_at_place(7, 3)             # long time
True
sage: Q.is_locally_represented_number_at_place(7, 5)             # long time
True
```

is_locally_universal_at_all_places()

Determine if the quadratic form represents $\mathbf{Z}_p$ for all finite/non-archimedean primes, and represents all real numbers.

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.is_locally_universal_at_all_places()
False
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.is_locally_universal_at_all_places()
False
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1,-1])
sage: Q.is_locally_universal_at_all_places()        # long time (8.5 s)
True
```

is_locally_universal_at_all_primes()

Determine if the quadratic form represents $\mathbf{Z}_p$ for all finite/non-archimedean primes.

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.is_locally_universal_at_all_primes()
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.is_locally_universal_at_all_primes()
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.is_locally_universal_at_all_primes()
False
```

is_locally_universal_at_prime(p)

Determine if the (integer-valued/rational) quadratic form represents all of $\mathbf{Z}_p$.

INPUT:

- p – a positive prime number or "infinity"

OUTPUT: boolean

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.is_locally_universal_at_prime(2)
True
sage: Q.is_locally_universal_at_prime(3)
True
sage: Q.is_locally_universal_at_prime(5)
True
sage: Q.is_locally_universal_at_prime(infinity)
False
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.is_locally_universal_at_prime(2)
```

(continues on next page)

(continued from previous page)

```
False
sage: Q.is_locally_universal_at_prime(3)
True
sage: Q.is_locally_universal_at_prime(5)
True
sage: Q.is_locally_universal_at_prime(infinity)
False
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,-1])
sage: Q.is_locally_universal_at_prime(infinity)
True
```

is_negative_definite()

Determines if the given quadratic form is negative-definite.

Note: A degenerate form is considered neither definite nor indefinite.

Note: The zero-dimensional form is considered both positive definite and negative definite.

OUTPUT: boolean – True or False

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [-1,-3,-5])
sage: Q.is_negative_definite()
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-3,5])
sage: Q.is_negative_definite()
False
```

is_odd(*allow_rescaling_flag=True*)

Return true iff after rescaling by some appropriate factor, the form represents some odd integers. For more details, see `parity()`.

Requires that Q is defined over $\mathbf{Z}$.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1, 0, 1])
sage: Q.is_odd()
True
sage: Q = QuadraticForm(ZZ, 2, [1, 1, 1])
sage: Q.is_odd()
False
```

is_positive_definite()

Determines if the given quadratic form is positive-definite.

Note: A degenerate form is considered neither definite nor indefinite.

Note: The zero-dimensional form is considered both positive definite and negative definite.

OUTPUT: boolean – True or False

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5])
sage: Q.is_positive_definite()
True
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,-3,5])
sage: Q.is_positive_definite()
False
```

is_primitive()

Determines if the given integer-valued form is primitive (i.e. not an integer (>1) multiple of another integer-valued quadratic form).

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [2,3,4])
sage: Q.is_primitive()
True
sage: Q = QuadraticForm(ZZ, 2, [2,4,8])
sage: Q.is_primitive()
False
```

is_rationally_isometric(*other*, *return_matrix=False*)

Determine if two regular quadratic forms over a number field are isometric.

INPUT:

- `other` – a quadratic form over a number field
- `return_matrix` – (boolean, default `False`) return the transformation matrix instead of a boolean; this is currently only implemented for forms over `QQ`

OUTPUT:

- if `return_matrix` is `False`: a boolean
- if `return_matrix` is `True`: either `False` or the transformation matrix

EXAMPLES:

```
sage: V = DiagonalQuadraticForm(QQ, [1, 1, 2])
sage: W = DiagonalQuadraticForm(QQ, [2, 2, 2])
sage: V.is_rationally_isometric(W)                                        #␣
↪optional - sage.libs.pari
True
```

```
sage: x = polygen(ZZ, 'x')
sage: K.<a> = NumberField(x^2 - 3)                                        #␣
↪optional - sage.rings.number_field
sage: V = QuadraticForm(K, 4, [1, 0, 0, 0, 2*a, 0, 0, a, 0, 2]); V        #␣
↪optional - sage.rings.number_field
```

(continues on next page)

(continued from previous page)

```
Quadratic form in 4 variables over Number Field in a
 with defining polynomial x^2 - 3 with coefficients:
[ 1 0 0 0 ]
[ * 2*a 0 0 ]
[ * * a 0 ]
[ * * * 2 ]
sage: W = QuadraticForm(K, 4, [1, 2*a, 4, 6, 3, 10, 2, 1, 2, 5]); W          #␣
↪optional - sage.rings.number_field
Quadratic form in 4 variables over Number Field in a
 with defining polynomial x^2 - 3 with coefficients:
[ 1 2*a 4 6 ]
[ * 3 10 2 ]
[ * * 1 2 ]
[ * * * 5 ]
sage: V.is_rationally_isometric(W)                                           #␣
↪optional - sage.rings.number_field
False
```

```
sage: K.<a> = NumberField(x^4 + 2*x + 6)                                     #␣
↪optional - sage.rings.number_field
sage: V = DiagonalQuadraticForm(K, [a, 2, 3, 2, 1]); V                       #␣
↪optional - sage.rings.number_field
Quadratic form in 5 variables over Number Field in a
 with defining polynomial x^4 + 2*x + 6 with coefficients:
[ a 0 0 0 0 ]
[ * 2 0 0 0 ]
[ * * 3 0 0 ]
[ * * * 2 0 ]
[ * * * * 1 ]
sage: W = DiagonalQuadraticForm(K, [a, a, a, 2, 1]); W                       #␣
↪optional - sage.rings.number_field
Quadratic form in 5 variables over Number Field in a
 with defining polynomial x^4 + 2*x + 6 with coefficients:
[ a 0 0 0 0 ]
[ * a 0 0 0 ]
[ * * a 0 0 ]
[ * * * 2 0 ]
[ * * * * 1 ]
sage: V.is_rationally_isometric(W)                                           #␣
↪optional - sage.rings.number_field
False
```

```
sage: K.<a> = NumberField(x^2 - 3)                                           #␣
↪optional - sage.rings.number_field
sage: V = DiagonalQuadraticForm(K, [-1, a, -2*a])                            #␣
↪optional - sage.rings.number_field
sage: W = DiagonalQuadraticForm(K, [-1, -a, 2*a])                            #␣
↪optional - sage.rings.number_field
sage: V.is_rationally_isometric(W)                                           #␣
↪optional - sage.rings.number_field
True
```

(continues on next page)

(continued from previous page)

```
sage: V = DiagonalQuadraticForm(QQ, [1, 1, 2])                              #␣
↪optional - sage.rings.number_field
sage: W = DiagonalQuadraticForm(QQ, [2, 2, 2])                              #␣
↪optional - sage.rings.number_field
sage: T = V.is_rationally_isometric(W, True); T                             #␣
↪optional - sage.rings.number_field
[   0    0    1]
[-1/2 -1/2    0]
[ 1/2 -1/2    0]
sage: V.Gram_matrix() == T.transpose() * W.Gram_matrix() * T                #␣
↪optional - sage.rings.number_field
True

sage: T = W.is_rationally_isometric(V, True); T                             #␣
↪optional - sage.rings.number_field
[ 0 -1  1]
[ 0 -1 -1]
[ 1  0  0]
sage: W.Gram_matrix() == T.T * V.Gram_matrix() * T                          #␣
↪optional - sage.rings.number_field
True
```

```
sage: L = QuadraticForm(QQ, 3, [2, 2, 0, 2, 2, 5])
sage: M = QuadraticForm(QQ, 3, [2, 2, 0, 3, 2, 3])
sage: L.is_rationally_isometric(M, True)                                    #␣
↪optional - sage.libs.pari
False
```

```
sage: A = DiagonalQuadraticForm(QQ, [1, 5])
sage: B = QuadraticForm(QQ, 2, [1, 12, 81])
sage: T = A.is_rationally_isometric(B, True); T                             #␣
↪optional - sage.libs.pari
[  1  -2]
[  0 1/3]
sage: A.Gram_matrix() == T.T * B.Gram_matrix() * T                          #␣
↪optional - sage.libs.pari
True
```

```
sage: C = DiagonalQuadraticForm(QQ, [1, 5, 9])
sage: D = DiagonalQuadraticForm(QQ, [6, 30, 1])
sage: T = C.is_rationally_isometric(D, True); T                             #␣
↪optional - sage.libs.pari
[   0 -5/6  1/2]
[   0  1/6  1/2]
[  -1    0    0]
sage: C.Gram_matrix() == T.T * D.Gram_matrix() * T                          #␣
↪optional - sage.libs.pari
True
```

```
sage: E = DiagonalQuadraticForm(QQ, [1, 1])
sage: F = QuadraticForm(QQ, 2, [17, 94, 130])
sage: T = F.is_rationally_isometric(E, True); T                                    #␣
↪optional - sage.libs.pari
[      -4 -189/17]
[      -1  -43/17]
sage: F.Gram_matrix() == T.T * E.Gram_matrix() * T                                 #␣
↪optional - sage.libs.pari
True
```

is_zero(*v*, *p=0*)

Determine if the vector v is on the conic $Q(x) = 0 \pmod p$.

EXAMPLES:

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q1.is_zero([0,1,0], 2)
True
sage: Q1.is_zero([1,1,1], 2)
True
sage: Q1.is_zero([1,1,0], 2)
False
```

is_zero_nonsingular(*v*, *p=0*)

Determine if the vector v is on the conic $Q(x) = 0 \pmod p$, and that this point is non-singular point of the conic.

EXAMPLES:

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q1.is_zero_nonsingular([1,1,1], 2)
True
sage: Q1.is_zero([1, 19, 2], 37)
True
sage: Q1.is_zero_nonsingular([1, 19, 2], 37)
False
```

is_zero_singular(*v*, *p=0*)

Determine if the vector v is on the conic $Q(x) = 0 \pmod p$, and that this point is singular point of the conic.

EXAMPLES:

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q1.is_zero([1,1,1], 2)
True
sage: Q1.is_zero_singular([1,1,1], 2)
False
sage: Q1.is_zero_singular([1, 19, 2], 37)
True
```

jordan_blocks_by_scale_and_unimodular(*p*, *safe_flag=True*)

Return a list of pairs (s_i, L_i) where L_i is a maximal p^{s_i}-unimodular Jordan component which is further decomposed into block diagonals of block size ≤ 2.

For each L_i the 2×2 blocks are listed after the 1×1 blocks (which follows from the convention of the `local_normal_form()` method).

Note: The decomposition of each L_i into smaller blocks is not unique!

The `safe_flag` argument allows us to select whether we want a copy of the output, or the original output. By default `safe_flag = True`, so we return a copy of the cached information. If this is set to `False`, then the routine is much faster but the return values are vulnerable to being corrupted by the user.

INPUT:

- `p` – a prime number > 0

OUTPUT:

A list of pairs (s_i, L_i) where:

- s_i is an integer,
- L_i is a block-diagonal unimodular quadratic form over $\mathbf{Z}_p$.

Note: These forms L_i are defined over the p-adic integers, but by a matrix over $\mathbf{Z}$ (or $\mathbf{Q}$?).

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,9,5,7])
sage: Q.jordan_blocks_by_scale_and_unimodular(3)
[(0, Quadratic form in 3 variables over Integer Ring with coefficients:
       [ 1 0 0 ]
       [ * 5 0 ]
       [ * * 7 ]),
 (2, Quadratic form in 1 variables over Integer Ring with coefficients:
       [ 1 ])]
```

```
sage: Q2 = QuadraticForm(ZZ, 2, [1,1,1])
sage: Q2.jordan_blocks_by_scale_and_unimodular(2)
[(-1, Quadratic form in 2 variables over Integer Ring with coefficients:
        [ 2 2 ]
        [ * 2 ])]
sage: Q = Q2 + Q2.scale_by_factor(2)
sage: Q.jordan_blocks_by_scale_and_unimodular(2)
[(-1, Quadratic form in 2 variables over Integer Ring with coefficients:
        [ 2 2 ]
        [ * 2 ]),
 (0, Quadratic form in 2 variables over Integer Ring with coefficients:
        [ 2 2 ]
        [ * 2 ])]
```

jordan_blocks_in_unimodular_list_by_scale_power(p)

Return a list of Jordan components, whose component at index i should be scaled by the factor p^i.

This is only defined for integer-valued quadratic forms (i.e., forms with base ring $\mathbf{Z}$), and the indexing only works correctly for $p = 2$ when the form has an integer Gram matrix.

INPUT:

- `self` – a quadratic form over $\mathbf{Z}$, which has integer Gram matrix if $p = 2$
- `p` – a prime number > 0

OUTPUT: a list of p-unimodular quadratic forms

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [2, -2, 0, 3, -5, 4])
sage: Q.jordan_blocks_in_unimodular_list_by_scale_power(2)
Traceback (most recent call last):
...
TypeError: the given quadratic form has a Jordan component with a negative␣
↪scale exponent

sage: Q.scale_by_factor(2).jordan_blocks_in_unimodular_list_by_scale_power(2)
[Quadratic form in 2 variables over Integer Ring with coefficients:
   [ 0 2 ]
   [ * 0 ],
 Quadratic form in 0 variables over Integer Ring with coefficients:
   ,
 Quadratic form in 1 variables over Integer Ring with coefficients:
   [ 345 ]]

sage: Q.jordan_blocks_in_unimodular_list_by_scale_power(3)
[Quadratic form in 2 variables over Integer Ring with coefficients:
   [ 2 0 ]
   [ * 10 ],
 Quadratic form in 1 variables over Integer Ring with coefficients:
   [ 2 ]]
```

level()

Determines the level of the quadratic form over a PID, which is a generator for the smallest ideal N of R such that $N \cdot$ (the matrix of $2 * Q$)$^{(-1)}$ is in R with diagonal in $2R$.

Over $\mathbf{Z}$ this returns a non-negative number.

(Caveat: This always returns the unit ideal when working over a field!)

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, range(1,4))
sage: Q.level()
8

sage: Q1 = QuadraticForm(QQ, 2, range(1,4))
sage: Q1.level()      # random
UserWarning: Warning -- The level of a quadratic form over a field is always 1.
Do you really want to do this?!?
1

sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.level()
420
```

level__Tornaria()

Return the level of the quadratic form, defined as

- level(B) for even dimension,
- level(B)/4 for odd dimension,

where $2Q(x) = x^t \cdot B \cdot x$.

This agrees with the usual level for even dimension…

EXAMPLES:

```
sage: DiagonalQuadraticForm(ZZ, [1]).level__Tornaria()
1
sage: DiagonalQuadraticForm(ZZ, [1,1]).level__Tornaria()
4
sage: DiagonalQuadraticForm(ZZ, [1,1,1]).level__Tornaria()
1
sage: DiagonalQuadraticForm(ZZ, [1,1,1,1]).level__Tornaria()
4
```

level_ideal()

Determine the level of the quadratic form (over R), which is the smallest ideal N of R such that $N \cdot$ (the matrix of $2Q$)$^{(-1)}$ is in R with diagonal in $2R$. (Caveat: This always returns the principal ideal when working over a field!)

> **Warning:** This only works over a PID ring of integers for now! (Waiting for Sage fractional ideal support.)

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, range(1,4))
sage: Q.level_ideal()
Principal ideal (8) of Integer Ring

sage: Q1 = QuadraticForm(QQ, 2, range(1,4))
sage: Q1.level_ideal()
Principal ideal (1) of Rational Field

sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.level_ideal()
Principal ideal (420) of Integer Ring
```

list_external_initializations()

Return a list of the fields which were set externally at creation, and not created through the usual *QuadraticForm* methods. These fields are as good as the external process that made them, and are thus not guaranteed to be correct.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,0,5])
sage: Q.list_external_initializations()
[]
sage: T = Q.theta_series()                                                  #␣
↪optional - sage.libs.pari
sage: Q.list_external_initializations()                                     #␣
```

(continues on next page)

(continued from previous page)

```
↪optional - sage.libs.pari
[]
sage: Q = QuadraticForm(ZZ, 2, [1,0,5], unsafe_initialization=False,        #␣
↪optional - sage.libs.pari
....:                        number_of_automorphisms=3, determinant=0)
sage: Q.list_external_initializations()                                    #␣
↪optional - sage.libs.pari
[]
```

```
sage: Q = QuadraticForm(ZZ, 2, [1,0,5], unsafe_initialization=False,        #␣
↪optional - sage.libs.pari
....:                        number_of_automorphisms=3, determinant=0)
sage: Q.list_external_initializations()                                    #␣
↪optional - sage.libs.pari
[]
sage: Q = QuadraticForm(ZZ, 2, [1,0,5], unsafe_initialization=True,         #␣
↪optional - sage.libs.pari
....:                        number_of_automorphisms=3, determinant=0)
sage: Q.list_external_initializations()                                    #␣
↪optional - sage.libs.pari
['number_of_automorphisms', 'determinant']
```

lll()

Return an LLL-reduced form of Q (using PARI).

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, range(1,11))
sage: Q.is_definite()
True
sage: Q.lll()
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 0 1 0 ]
[ * 4 3 3 ]
[ * * 6 3 ]
[ * * * 6 ]
```

local_badII_density_congruence(*p*, *m*, *Zvec=None*, *NZvec=None*)

Find the Bad-type II local density of Q representing m at p. (Assuming that $p > 2$ and Q is given in local diagonal form.)

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_badII_density_congruence(2, 1, None, None)
0
sage: Q.local_badII_density_congruence(2, 2, None, None)
0
sage: Q.local_badII_density_congruence(2, 4, None, None)
0
sage: Q.local_badII_density_congruence(3, 1, None, None)
0
sage: Q.local_badII_density_congruence(3, 6, None, None)
0
sage: Q.local_badII_density_congruence(3, 9, None, None)
0
sage: Q.local_badII_density_congruence(3, 27, None, None)
0
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,3,9,9])
sage: Q.local_badII_density_congruence(3, 1, None, None)
0
sage: Q.local_badII_density_congruence(3, 3, None, None)
0
sage: Q.local_badII_density_congruence(3, 6, None, None)
0
sage: Q.local_badII_density_congruence(3, 9, None, None)
4/27
sage: Q.local_badII_density_congruence(3, 18, None, None)
4/9
```

local_badI_density_congruence(*p*, *m*, *Zvec=None*, *NZvec=None*)

Find the Bad-type I local density of Q representing m at p. (Assuming that $p > 2$ and Q is given in local diagonal form.)

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_badI_density_congruence(2, 1, None, None)
0
sage: Q.local_badI_density_congruence(2, 2, None, None)
1
sage: Q.local_badI_density_congruence(2, 4, None, None)
0
sage: Q.local_badI_density_congruence(3, 1, None, None)
0
sage: Q.local_badI_density_congruence(3, 6, None, None)
```

(continues on next page)

(continued from previous page)

```
0
sage: Q.local_badI_density_congruence(3, 9, None, None)
0
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.local_badI_density_congruence(2, 1, None, None)
0
sage: Q.local_badI_density_congruence(2, 2, None, None)
0
sage: Q.local_badI_density_congruence(2, 4, None, None)
0
sage: Q.local_badI_density_congruence(3, 2, None, None)
0
sage: Q.local_badI_density_congruence(3, 6, None, None)
0
sage: Q.local_badI_density_congruence(3, 9, None, None)
0
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,3,9])
sage: Q.local_badI_density_congruence(3, 1, None, None)
0
sage: Q.local_badI_density_congruence(3, 3, None, None)
4/3
sage: Q.local_badI_density_congruence(3, 6, None, None)
4/3
sage: Q.local_badI_density_congruence(3, 9, None, None)
0
sage: Q.local_badI_density_congruence(3, 18, None, None)
0
```

`local_bad_density_congruence`(*p*, *m*, *Zvec=None*, *NZvec=None*)

Find the Bad-type local density of Q representing m at p, allowing certain congruence conditions mod p.

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_bad_density_congruence(2, 1, None, None)
0
sage: Q.local_bad_density_congruence(2, 2, None, None)
1
sage: Q.local_bad_density_congruence(2, 4, None, None)
0
sage: Q.local_bad_density_congruence(3, 1, None, None)
```

(continues on next page)

(continued from previous page)

```
0
sage: Q.local_bad_density_congruence(3, 6, None, None)
0
sage: Q.local_bad_density_congruence(3, 9, None, None)
0
sage: Q.local_bad_density_congruence(3, 27, None, None)
0
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,3,9,9])
sage: Q.local_bad_density_congruence(3, 1, None, None)
0
sage: Q.local_bad_density_congruence(3, 3, None, None)
4/3
sage: Q.local_bad_density_congruence(3, 6, None, None)
4/3
sage: Q.local_bad_density_congruence(3, 9, None, None)
4/27
sage: Q.local_bad_density_congruence(3, 18, None, None)
4/9
sage: Q.local_bad_density_congruence(3, 27, None, None)
8/27
```

local_density(*p*, *m*)

Return the local density.

Note: This screens for imprimitive forms, and puts the quadratic form in local normal form, which is a *requirement* of the routines performing the computations!

INPUT:

- p – a prime number > 0
- m – an integer

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])   # NOTE: This is already in␣
↪local normal form for *all* primes p!
sage: Q.local_density(p=2, m=1)
1
sage: Q.local_density(p=3, m=1)
8/9
sage: Q.local_density(p=5, m=1)
24/25
sage: Q.local_density(p=7, m=1)
48/49
sage: Q.local_density(p=11, m=1)
120/121
```

local_density_congruence(*p*, *m*, *Zvec=None*, *NZvec=None*)

Find the local density of Q representing m at p, allowing certain congruence conditions mod p.

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.local_density_congruence(p=2, m=1, Zvec=None, NZvec=None)
1
sage: Q.local_density_congruence(p=3, m=1, Zvec=None, NZvec=None)
8/9
sage: Q.local_density_congruence(p=5, m=1, Zvec=None, NZvec=None)
24/25
sage: Q.local_density_congruence(p=7, m=1, Zvec=None, NZvec=None)
48/49
sage: Q.local_density_congruence(p=11, m=1, Zvec=None, NZvec=None)
120/121
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_density_congruence(2, 1, None, None)
1
sage: Q.local_density_congruence(2, 2, None, None)
1
sage: Q.local_density_congruence(2, 4, None, None)
3/2
sage: Q.local_density_congruence(3, 1, None, None)
2/3
sage: Q.local_density_congruence(3, 6, None, None)
4/3
sage: Q.local_density_congruence(3, 9, None, None)
14/9
sage: Q.local_density_congruence(3, 27, None, None)
2
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,3,9,9])
sage: Q.local_density_congruence(3, 1, None, None)
2
sage: Q.local_density_congruence(3, 3, None, None)
4/3
sage: Q.local_density_congruence(3, 6, None, None)
4/3
sage: Q.local_density_congruence(3, 9, None, None)
2/9
sage: Q.local_density_congruence(3, 18, None, None)
4/9
```

local_genus_symbol(p)

Return the Conway-Sloane genus symbol of 2 times a quadratic form defined over $\mathbf{Z}$ at a prime number p.

This is defined (in the class *Genus_Symbol_p_adic_ring*) to be a list of tuples (one for each Jordan component $p^m \cdot A$ at p, where A is a unimodular symmetric matrix with coefficients the p-adic integers) of the following form:

- If $p > 2$, then return triples of the form $[m, n, d]$ where
  - m = valuation of the component
  - n = rank of A
  - d = det(A) in $\{1, u\}$ for normalized quadratic non-residue u.
- If $p = 2$, then return quintuples of the form $[m, n, s, d, o]$ where
  - m = valuation of the component
  - n = rank of A
  - d = det(A) in $\{1, 3, 5, 7\}$
  - s = 0 (or 1) if A is even (or odd)
  - o = oddity of A (= 0 if s = 0) in $\mathbf{Z}/8\mathbf{Z}$ = the trace of the diagonalization of A

Note: The Conway-Sloane convention for describing the prime $p = -1$ is not supported here, and neither is the convention for including the 'prime' Infinity. See note on p370 of Conway-Sloane (3rd ed) [CS1999] for a discussion of this convention.

INPUT:

- `p` – a prime number > 0

OUTPUT:

a Conway-Sloane genus symbol at p, which is an instance of the class *Genus_Symbol_p_adic_ring*.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3,4])
sage: Q.local_genus_symbol(2)
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: Q.local_genus_symbol(3)
Genus symbol at 3:     1^3 3^-1
sage: Q.local_genus_symbol(5)
Genus symbol at 5:     1^4
```

local_good_density_congruence(*p*, *m*, *Zvec=None*, *NZvec=None*)

Find the Good-type local density of Q representing m at p. (Front end routine for parity specific routines for p.)

Todo: Add documentation about the additional congruence conditions `Zvec` and `NZvec`.

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_good_density_congruence(2, 1, None, None)
1
sage: Q.local_good_density_congruence(3, 1, None, None)
2/3
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.local_good_density_congruence(2, 1, None, None)
1
sage: Q.local_good_density_congruence(3, 1, None, None)
8/9
```

local_good_density_congruence_even(*m*, *Zvec*, *NZvec*)

Find the Good-type local density of Q representing m at $p = 2$. (Assuming Q is given in local diagonal form.)

The additional congruence condition arguments `Zvec` and `NZvec` can be either a list of indices or None. `Zvec=[]` is equivalent to `Zvec=None` which both impose no additional conditions, but `NZvec=[]` returns no solutions always while `NZvec=None` imposes no additional condition.

> **Warning:** Here the indices passed in `Zvec` and `NZvec` represent indices of the solution vector x of $Q(x) = m \pmod{p^k}$, and *not* the Jordan components of Q. They therefore are required (and assumed) to include either all or none of the indices of a given Jordan component of Q. This is only important when $p = 2$ since otherwise all Jordan blocks are 1×1, and so there the indices and Jordan blocks coincide.

Todo: Add type checking for `Zvec` and `NZvec`, and that Q is in local normal form.

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and 2-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_good_density_congruence_even(1, None, None)
1
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.local_good_density_congruence_even(1, None, None)
1
sage: Q.local_good_density_congruence_even(2, None, None)
```

(continues on next page)

(continued from previous page)

```
3/2
sage: Q.local_good_density_congruence_even(3, None, None)
1
sage: Q.local_good_density_congruence_even(4, None, None)
1/2
```

```
sage: Q = QuadraticForm(ZZ, 4, range(10))
sage: Q[0,0] = 5
sage: Q[1,1] = 10
sage: Q[2,2] = 15
sage: Q[3,3] = 20
sage: Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 5 1 2 3 ]
[ * 10 5 6 ]
[ * * 15 8 ]
[ * * * 20 ]
sage: Q.theta_series(20)                                                    #␣
↪optional - sage.libs.pari
1 + 2*q^5 + 2*q^10 + 2*q^14 + 2*q^15 + 2*q^16 + 2*q^18 + O(q^20)
sage: Q.local_normal_form(2)                                                #␣
↪optional - sage.libs.pari sage.rings.padics
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 0 1 0 0 ]
[ * 0 0 0 ]
[ * * 0 1 ]
[ * * * 0 ]
sage: Q.local_good_density_congruence_even(1, None, None)
3/4
sage: Q.local_good_density_congruence_even(2, None, None)
1
sage: Q.local_good_density_congruence_even(5, None, None)
3/4
```

local_good_density_congruence_odd(*p*, *m*, *Zvec*, *NZvec*)

Find the Good-type local density of Q representing m at p. (Assuming that $p > 2$ and Q is given in local diagonal form.)

The additional congruence condition arguments `Zvec` and `NZvec` can be either a list of indices or None. `Zvec=[]` is equivalent to `Zvec=None`, which both impose no additional conditions, but `NZvec=[]` returns no solutions always while `NZvec=None` imposes no additional condition.

Todo: Add type checking for `Zvec`, `NZvec`, and that Q is in local normal form.

INPUT:

- `self` – quadratic form Q, assumed to be diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_good_density_congruence_odd(3, 1, None, None)
2/3
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.local_good_density_congruence_odd(3, 1, None, None)
8/9
```

local_normal_form(p)

Return a locally integrally equivalent quadratic form over the p-adic integers $\mathbf{Z}_p$ which gives the Jordan decomposition.

The Jordan components are written as sums of blocks of size ≤ 2 and are arranged by increasing scale, and then by increasing norm. This is equivalent to saying that we put the 1×1 blocks before the 2×2 blocks in each Jordan component.

INPUT:

- p – a positive prime number

OUTPUT: a quadratic form over $\mathbf{Z}$

> **Warning:** Currently this only works for quadratic forms defined over $\mathbf{Z}$.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [10,4,1])
sage: Q.local_normal_form(5)
Quadratic form in 2 variables over Integer Ring with coefficients:
  [ 1 0 ]
  [ * 6 ]
```

```
sage: Q.local_normal_form(3)
Quadratic form in 2 variables over Integer Ring with coefficients:
  [ 10 0 ]
  [ * 15 ]

sage: Q.local_normal_form(2)
Quadratic form in 2 variables over Integer Ring with coefficients:
  [ 1 0 ]
  [ * 6 ]
```

local_primitive_density(p, m)

Return the local primitive density – should be called by the user. =)

NOTE: This screens for imprimitive forms, and puts the quadratic form in local normal form, which is a *requirement* of the routines performing the computations!

INPUT:

- p – a prime number > 0

- `m` – an integer

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, range(10))
sage: Q[0,0] = 5
sage: Q[1,1] = 10
sage: Q[2,2] = 15
sage: Q[3,3] = 20
sage: Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 5 1 2 3 ]
[ * 10 5 6 ]
[ * * 15 8 ]
[ * * * 20 ]
sage: Q.theta_series(20)                                                    #␣
↪optional - sage.libs.pari
1 + 2*q^5 + 2*q^10 + 2*q^14 + 2*q^15 + 2*q^16 + 2*q^18 + O(q^20)
sage: Q.local_normal_form(2)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 0 1 0 0 ]
[ * 0 0 0 ]
[ * * 0 1 ]
[ * * * 0 ]

sage: Q.local_primitive_density(2, 1)
3/4
sage: Q.local_primitive_density(5, 1)
24/25

sage: Q.local_primitive_density(2, 5)
3/4
sage: Q.local_density(2, 5)
3/4
```

`local_primitive_density_congruence`(*p*, *m*, *Zvec=None*, *NZvec=None*)

Find the primitive local density of Q representing m at p, allowing certain congruence conditions mod p.

Note: The following routine is not used internally, but is included for consistency.

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.local_primitive_density_congruence(p=2, m=1, Zvec=None, NZvec=None)
1
sage: Q.local_primitive_density_congruence(p=3, m=1, Zvec=None, NZvec=None)
8/9
sage: Q.local_primitive_density_congruence(p=5, m=1, Zvec=None, NZvec=None)
24/25
sage: Q.local_primitive_density_congruence(p=7, m=1, Zvec=None, NZvec=None)
48/49
sage: Q.local_primitive_density_congruence(p=11, m=1, Zvec=None, NZvec=None)
120/121
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_primitive_density_congruence(2, 1, None, None)
1
sage: Q.local_primitive_density_congruence(2, 2, None, None)
1
sage: Q.local_primitive_density_congruence(2, 4, None, None)
1
sage: Q.local_primitive_density_congruence(3, 1, None, None)
2/3
sage: Q.local_primitive_density_congruence(3, 6, None, None)
4/3
sage: Q.local_primitive_density_congruence(3, 9, None, None)
4/3
sage: Q.local_primitive_density_congruence(3, 27, None, None)
4/3
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,3,9,9])
sage: Q.local_primitive_density_congruence(3, 1, None, None)
2
sage: Q.local_primitive_density_congruence(3, 3, None, None)
4/3
sage: Q.local_primitive_density_congruence(3, 6, None, None)
4/3
sage: Q.local_primitive_density_congruence(3, 9, None, None)
4/27
sage: Q.local_primitive_density_congruence(3, 18, None, None)
4/9
sage: Q.local_primitive_density_congruence(3, 27, None, None)
8/27
sage: Q.local_primitive_density_congruence(3, 81, None, None)
8/27
sage: Q.local_primitive_density_congruence(3, 243, None, None)
8/27
```

local_representation_conditions(*recompute_flag=False*, *silent_flag=False*)

> **Warning:** This only works correctly for forms in >=3 variables, which are locally universal at almost all primes!

This class finds the local conditions for a number to be integrally represented by an integer-valued quadratic form. These conditions are stored in `self.__local_representability_conditions` and consist of a list of 9 element vectors, with one for each prime with a local obstruction (though only the first 5 are meaningful unless $p = 2$). The first element is always the prime p where the local obstruction occurs, and the next 8 (or 4) entries represent square-classes in the p-adic integers $\mathbf{Z}_p$, and are labeled by the $\mathbf{Q}_p$ square-classes $t \cdot (\mathbf{Q}_p)^2$ with t given as follows:

- for $p > 2$, `[ * 1 u p u p * * * * ]`,
- for $p = 2$, `[ * 1 3 5 7 2 6 10 14 ]`.

The integer appearing in each place tells us how p-divisible a number needs to be in that square-class in order to be locally represented by Q. A negative number indicates that the entire $\mathbf{Q}_p$ square-class is not represented, while a positive number x indicates that $t \cdot p^{(2 \cdot x)} (\mathbf{Z}_p)^2$ is locally represented but $t \cdot p^{(2 \cdot (x-1))}$ $(\mathbf{Z}_p)^2$ is not.

As an example, the vector `[2 3 0 0 0 0 2 0 infinity]` tells us that all positive integers are locally represented at $p = 2$ except those of the forms:

- $2^6 \cdot u \cdot r^2$ with $u = 1$ (mod 8)
- $2^5 \cdot u \cdot r^2$ with $u = 3$ (mod 8)
- $2 \cdot u \cdot r^2$ with $u = 7$ (mod 8)

At the real numbers, the vector which looks like `[infinity, 0, infinity, None, None, None, None, None, None]` means that Q is negative definite (i.e., the 0 tells us all positive reals are represented). The real vector always appears, and is listed before the other ones.

OUTPUT:

A list of 9-element vectors describing the representation obstructions at primes dividing the level.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [])
sage: Q.local_representation_conditions()
This 0-dimensional form only represents zero.

sage: Q = DiagonalQuadraticForm(ZZ, [5])
sage: Q.local_representation_conditions()
This 1-dimensional form only represents square multiples of 5.

sage: Q1 = DiagonalQuadraticForm(ZZ, [1,1])
sage: Q1.local_representation_conditions()
This 2-dimensional form represents the p-adic integers of even
valuation for all primes p except [2].
For these and the reals, we have:
 Reals:   [0, +Infinity]
 p = 2:   [0, +Infinity, 0, +Infinity, 0, +Infinity, 0, +Infinity]

sage: Q1 = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q1.local_representation_conditions()
This form represents the p-adic integers Z_p for all primes p except
[2].  For these and the reals, we have:
 Reals:   [0, +Infinity]
 p = 2:   [0, 0, 0, +Infinity, 0, 0, 0, 0]

sage: Q1 = DiagonalQuadraticForm(ZZ, [1,1,1,1])
```

(continues on next page)

(continued from previous page)

```
sage: Q1.local_representation_conditions()
This form represents the p-adic integers Z_p for all primes p except
[].  For these and the reals, we have:
 Reals:   [0, +Infinity]

sage: Q1 = DiagonalQuadraticForm(ZZ, [1,3,3,3])
sage: Q1.local_representation_conditions()
This form represents the p-adic integers Z_p for all primes p except
[3].  For these and the reals, we have:
 Reals:   [0, +Infinity]
 p = 3:   [0, 1, 0, 0]

sage: Q2 = DiagonalQuadraticForm(ZZ, [2,3,3,3])
sage: Q2.local_representation_conditions()
This form represents the p-adic integers Z_p for all primes p except
[3].  For these and the reals, we have:
 Reals:   [0, +Infinity]
 p = 3:   [1, 0, 0, 0]

sage: Q3 = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q3.local_representation_conditions()
This form represents the p-adic integers Z_p for all primes p except
[].  For these and the reals, we have:
 Reals:   [0, +Infinity]
```

local_zero_density_congruence(*p*, *m*, *Zvec=None*, *NZvec=None*)

Find the Zero-type local density of Q representing m at p, allowing certain congruence conditions mod p.

INPUT:

- `self` – quadratic form Q, assumed to be block diagonal and p-integral
- `p` – a prime number
- `m` – an integer
- `Zvec`, `NZvec` – non-repeating lists of integers in `range(self.dim())` or `None`

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: Q.local_zero_density_congruence(2, 2, None, None)
0
sage: Q.local_zero_density_congruence(2, 4, None, None)
1/2
sage: Q.local_zero_density_congruence(3, 6, None, None)
0
sage: Q.local_zero_density_congruence(3, 9, None, None)
2/9
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.local_zero_density_congruence(2, 2, None, None)
0
```

(continues on next page)

(continued from previous page)

```
sage: Q.local_zero_density_congruence(2, 4, None, None)
1/4
sage: Q.local_zero_density_congruence(3, 6, None, None)
0
sage: Q.local_zero_density_congruence(3, 9, None, None)
8/81
```

mass__by_Siegel_densities(*odd_algorithm='Pall'*, *even_algorithm='Watson'*)

Return the mass of transformations (det 1 and -1).

Warning: This is broken right now…

INPUT:

- `odd_algorithm` – algorithm to be used when $p > 2$; `"Pall"` (only one choice for now)
- `even_algorithm` – algorithm to be used when $p = 2$; either `"Kitaoka"` or `"Watson"` (the default)

REFERENCES:

- Nipp's Book "Tables of Quaternary Quadratic Forms".
- Papers of Pall (only for $p > 2$) and Watson (for $p = 2$ – tricky!).
- Siegel, Milnor-Hussemoller, Conway-Sloane Paper IV, Kitoaka (all of which have problems…)

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: m = Q.mass__by_Siegel_densities(); m                                  #␣
↪optional - sage.symbolic
1/384
sage: m - (2^Q.dim() * factorial(Q.dim()))^(-1)                              #␣
↪optional - sage.symbolic
0
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: m = Q.mass__by_Siegel_densities(); m                                  #␣
↪optional - sage.symbolic
1/48
sage: m - (2^Q.dim() * factorial(Q.dim()))^(-1)                              #␣
↪optional - sage.symbolic
0
```

mass_at_two_by_counting_mod_power(*k*)

Compute the local mass at $p = 2$ assuming that it's stable (mod 2^k).

Note: This is **way** too slow to be useful, even when $k = 1$.

Todo: Remove this routine, or try to compile it!

INPUT:

- k – an integer ≥ 1

OUTPUT: a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.mass_at_two_by_counting_mod_power(1)
4
```

matrix()

Return the Hessian matrix A for which $Q(X) = (1/2)X^t \cdot A \cdot X$.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, range(6))
sage: Q.matrix()
[ 0  1  2]
[ 1  6  4]
[ 2  4 10]
```

minkowski_reduction()

Find a Minkowski-reduced form equivalent to the given one.

This means that

$$Q(v_k) \leq Q(s_1 \cdot v_1 + ... + s_n \cdot v_n)$$

for all s_i where $\gcd(s_k, ... s_n) = 1$.

Note: When Q has dim ≤ 4 we can take all s_i in $\{1, 0, -1\}$.

REFERENCES:

- Schulze-Pillot's paper on "An algorithm for computing genera of ternary and quaternary quadratic forms", p138.
- Donaldson's 1979 paper "Minkowski Reduction of Integral Matrices", p203.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, [30, 17, 11, 12, 29, 25, 62, 64, 25, 110])
sage: Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 30 17 11 12 ]
[ * 29 25 62 ]
[ * * 64 25 ]
[ * * * 110 ]
sage: Q.minkowski_reduction()
(
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 30 17 11 -5 ]
[ * 29 25 4 ]
[ * * 64 0 ]
[ * * * 77 ]                                                   ,
```

(continues on next page)

(continued from previous page)

```
[ 1  0  0  0]
[ 0  1  0 -1]
[ 0  0  1  0]
[ 0  0  0  1]
)
```

```
sage: Q = QuadraticForm(ZZ,4,[1, -2, 0, 0, 2, 0, 0, 2, 0, 2])
sage: Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 -2 0 0 ]
[ * 2 0 0 ]
[ * * 2 0 ]
[ * * * 2 ]
sage: Q.minkowski_reduction()
(
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 0 0 0 ]
[ * 1 0 0 ]
[ * * 2 0 ]
[ * * * 2 ]                                                     ,

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

minkowski_reduction_for_4vars__SP()

Find a Minkowski-reduced form equivalent to the given one. This means that

$$Q(v_k) \leq Q(s_1 \cdot v_1 + ... + s_n \cdot v_n)$$

for all s_i where GCD$(s_k, ...s_n)$ = 1.

Note: When Q has dim ≤ 4, we can take all s_i in $\{1, 0, -1\}$.

REFERENCES:

- Schulze-Pillot's paper on "An algorithm for computing genera of ternary and quaternary quadratic forms", p138.
- Donaldson's 1979 paper "Minkowski Reduction of Integral Matrices", p203.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, [30,17,11,12,29,25,62,64,25,110]); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 30 17 11 12 ]
[ * 29 25 62 ]
[ * * 64 25 ]
[ * * * 110 ]
sage: Q.minkowski_reduction_for_4vars__SP()
```

(continues on next page)

(continued from previous page)

```
(
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 29 -17 25 4 ]
[ * 30 -11 5 ]
[ * * 64 0 ]
[ * * * 77 ]                                                            ,

[ 0  1  0  0]
[ 1  0  0 -1]
[ 0  0  1  0]
[ 0  0  0  1]
)
```

multiply_variable(*c*, *i*, *in_place=False*)

Replace the variables x_i by $c \cdot x_i$ in the quadratic form (replacing the original form if the `in_place` flag is True).

Here c must be an element of the base ring defining the quadratic form.

INPUT:

- `c` – an element of `self.base_ring()`
- `i` – an integer ≥ 0

OUTPUT: a `QuadraticForm` (by default, otherwise none)

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,9,5,7])
sage: Q.multiply_variable(5, 0)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 25 0 0 0 ]
[ * 9 0 0 ]
[ * * 5 0 ]
[ * * * 7 ]
```

neighbor_iteration(*seeds*, *p*, *mass=None*, *max_classes=1000*, *algorithm=None*, *max_neighbors=1000*, *verbose=False*)

Return all classes in the p-neighbor graph of `self`.

Starting from the given seeds, this function successively finds p-neighbors until no new quadratic form (class) is obtained.

INPUT:

- `seeds` – a list of quadratic forms in the same genus
- `p` – a prime number
- `mass` – (optional) a rational number; the mass of this genus
- `max_classes` – (default: `1000`) break the computation when `max_classes` are found
- `algorithm` – (optional) one of `'orbits'`, `'random'`, `'exhaustion'`
- `max_random_trys` – (default: `1000`) the maximum number of neighbors computed for a single lattice

OUTPUT: a list of quadratic forms

EXAMPLES:

```
sage: from sage.quadratic_forms.quadratic_form__neighbors import neighbor_
↪iteration
sage: Q = QuadraticForm(ZZ, 3, [1, 0, 0, 2, 1, 3])
sage: Q.det()
46
sage: mass = Q.conway_mass()                                          #␣
↪optional - sage.symbolic
sage: g1 = neighbor_iteration([Q], 3,                         # long time   #␣
↪optional - sage.symbolic
....:                         mass=mass, algorithm='random')
sage: g2 = neighbor_iteration([Q], 3, algorithm='exhaustion')  # long time
sage: g3 = neighbor_iteration([Q], 3, algorithm='orbits')              #␣
↪optional - sage.libs.gap
sage: mass == sum(1/q.number_of_automorphisms() for q in g1)   # long time   #␣
↪optional - sage.symbolic
True
sage: mass == sum(1/q.number_of_automorphisms() for q in g2)   # long time   #␣
↪optional - sage.symbolic
True
sage: mass == sum(1/q.number_of_automorphisms() for q in g3)              #␣
↪optional - sage.libs.gap sage.symbolic
True
```

number_of_automorphisms()

Return the number of automorphisms (of det 1 and -1) of the quadratic form.

OUTPUT: an integer ≥ 2.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, 0, 1, 0, 1], unsafe_initialization=True)
sage: Q.number_of_automorphisms()
48
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.number_of_automorphisms()
384
sage: 2^4 * factorial(4)
384
```

omega()

This is the content of the adjoint of the primitive associated quadratic form.

Ref: See Dickson's "Studies in Number Theory".

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,37])
sage: Q.omega()
4
```

orbits_lines_mod_p(p)

Let (L, q) be a lattice. This returns representatives of the orbits of lines in L/pL under the orthogonal group of q.

INPUT:

- p – a prime number

OUTPUT: a list of vectors over GF(p)

EXAMPLES:

```
sage: from sage.quadratic_forms.quadratic_form__neighbors import orbits_lines_
↪mod_p
sage: Q = QuadraticForm(ZZ, 3, [1, 0, 0, 2, 1, 3])
sage: Q.orbits_lines_mod_p(2)                                               #␣
↪optional - sage.libs.gap sage.libs.pari
[(0, 0, 1),
(0, 1, 0),
(0, 1, 1),
(1, 0, 0),
(1, 0, 1),
(1, 1, 0),
(1, 1, 1)]
```

parity(*allow_rescaling_flag=True*)

Return the parity ("even" or "odd") of an integer-valued quadratic form over $\mathbf{Z}$, defined up to similitude/rescaling of the form so that its Jordan component of smallest scale is unimodular. After this rescaling, we say a form is even if it only represents even numbers, and odd if it represents some odd number.

If the allow_rescaling_flag is set to False, then we require that the quadratic form have a Gram matrix with coefficients in $\mathbf{Z}$, and look at the unimodular Jordan block to determine its parity. This returns an error if the form is not integer-matrix, meaning that it has Jordan components at $p = 2$ which do not have an integer scale.

We determine the parity by looking for a 1×1 block in the 0-th Jordan component, after a possible rescaling.

INPUT:

- self – a quadratic form with base ring $\mathbf{Z}$, which we may require to have integer Gram matrix.

OUTPUT:

One of the strings: "even" or "odd"

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [4, -2, 0, 2, 3, 2]); Q
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 4 -2 0 ]
[ * 2 3 ]
[ * * 2 ]
sage: Q.parity()
'even'
```

```
sage: Q = QuadraticForm(ZZ, 3, [4, -2, 0, 2, 3, 1]); Q
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 4 -2 0 ]
[ * 2 3 ]
[ * * 1 ]
sage: Q.parity()
'even'
```

```
sage: Q = QuadraticForm(ZZ, 3, [4, -2, 0, 2, 2, 2]); Q
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 4 -2 0 ]
[ * 2 2 ]
[ * * 2 ]
sage: Q.parity()
'even'
```

```
sage: Q = QuadraticForm(ZZ, 3, [4, -2, 0, 2, 2, 1]); Q
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 4 -2 0 ]
[ * 2 2 ]
[ * * 1 ]
sage: Q.parity()
'odd'
```

polynomial(*names='x'*)

Return the quadratic form as a polynomial in n variables.

INPUT:

- `self` - a quadratic form over a commutative ring
- `names` - specification of the names of the variables; see `PolynomialRing()`

OUTPUT: The polynomial form of the quadratic form.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(QQ,[1, 3, 5, 7])
sage: P = Q.polynomial(); P
x0^2 + 3*x1^2 + 5*x2^2 + 7*x3^2
```

```
sage: x = polygen(ZZ, 'x')
sage: F.<a> = NumberField(x^2 - 5)                                      #␣
↪optional - sage.rings.number_field
sage: Z = F.ring_of_integers()                                          #␣
↪optional - sage.rings.number_field
sage: Q = QuadraticForm(Z, 3, [2*a, 3*a, 0, 1 - a, 0, 2*a + 4])        #␣
↪optional - sage.rings.number_field
sage: P = Q.polynomial(names='y'); P                                    #␣
↪optional - sage.rings.number_field
2*a*y0^2 + 3*a*y0*y1 + (-a + 1)*y1^2 + (2*a + 4)*y2^2
sage: Q = QuadraticForm(F, 4,                                           #␣
↪optional - sage.rings.number_field
....:                   [a, 3*a, 0, 1 - a, a - 3, 0, 2*a + 4, 4 + a, 0, 1])
sage: Q.polynomial(names='z')                                           #␣
↪optional - sage.rings.number_field
a*z0^2 + (3*a)*z0*z1 + (a - 3)*z1^2 + (a + 4)*z2^2
+ (-a + 1)*z0*z3 + (2*a + 4)*z1*z3 + z3^2
sage: B.<i,j,k> = QuaternionAlgebra(F,-1,-1)                            #␣
↪optional - sage.rings.number_field
sage: Q = QuadraticForm(B, 3, [2*a, 3*a, i, 1 - a, 0, 2*a + 4])        #␣
↪optional - sage.rings.number_field
```

(continues on next page)

(continued from previous page)

```
sage: Q.polynomial()                                                    #␣
↪optional - sage.rings.number_field
Traceback (most recent call last):
...
ValueError: Can only create polynomial rings over commutative rings.
```

primitive()

Return a primitive version of an integer-valued quadratic form, defined over $\mathbf{Z}$.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [2,3,4])
sage: Q.primitive()
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 2 3 ]
[ * 4 ]
sage: Q = QuadraticForm(ZZ, 2, [2,4,8])
sage: Q.primitive()
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 1 2 ]
[ * 4 ]
```

rational_diagonal_form(*return_matrix=False*)

Return a diagonal form equivalent to the given quadratic from over the fraction field of its defining ring.

INPUT:

- `return_matrix` – (boolean, default: False) also return the transformation matrix

OUTPUT: either the diagonal quadratic form D (if `return_matrix` is false) or the pair (D, T) (if `return_matrix` is true) where

- D – the diagonalized form of this quadratic form
- T – transformation matrix. This is such that `T.transpose() * self.matrix() * T` gives `D.matrix()`.

Both D and T are defined over the fraction field of the base ring of the given form.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [0,1,-1])
sage: Q
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 0 1 ]
[ * -1 ]
sage: Q.rational_diagonal_form()
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1/4 0 ]
[ * -1 ]
```

If we start with a diagonal form, we get back the same form defined over the fraction field:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.rational_diagonal_form()
Quadratic form in 4 variables over Rational Field with coefficients:
```

(continues on next page)

(continued from previous page)

```
[ 1 0 0 0 ]
[ * 3 0 0 ]
[ * * 5 0 ]
[ * * * 7 ]
```

In the following example, we check the consistency of the transformation matrix:

```
sage: Q = QuadraticForm(ZZ, 4, range(10))
sage: D, T = Q.rational_diagonal_form(return_matrix=True)
sage: D
Quadratic form in 4 variables over Rational Field with coefficients:
[ -1/16 0 0 0 ]
[ * 4 0 0 ]
[ * * 13 0 ]
[ * * * 563/52 ]
sage: T
[      1       0      11 149/26]
[   -1/8       1      -2 -10/13]
[      0       0       1 -29/26]
[      0       0       0      1]
sage: T.transpose() * Q.matrix() * T
[   -1/8       0       0      0]
[      0       8       0      0]
[      0       0      26      0]
[      0       0       0 563/26]
sage: D.matrix()
[   -1/8       0       0      0]
[      0       8       0      0]
[      0       0      26      0]
[      0       0       0 563/26]
```

```
sage: Q1 = QuadraticForm(ZZ, 4, [1, 1, 0, 0, 1, 0, 0, 1, 0, 18])
sage: Q1
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 1 0 0 ]
[ * 1 0 0 ]
[ * * 1 0 ]
[ * * * 18 ]
sage: Q1.rational_diagonal_form(return_matrix=True)
(
Quadratic form in 4 variables over Rational Field with coefficients:
[ 1 0 0 0 ]
[ * 3/4 0 0 ]
[ * * 1 0 ]
[ * * * 18 ]                                                          ,

[   1 -1/2    0    0]
[   0    1    0    0]
[   0    0    1    0]
[   0    0    0    1]
)
```

PARI returns a singular transformation matrix for this case:

```
sage: Q = QuadraticForm(QQ, 2, [1/2, 1, 1/2])
sage: Q.rational_diagonal_form()
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1/2 0 ]
[ * 0 ]
```

This example cannot be computed by PARI:

```
sage: Q = QuadraticForm(RIF, 4, range(10))
sage: Q.__pari__()
Traceback (most recent call last):
...
TypeError
sage: Q.rational_diagonal_form()
Quadratic form in 4 variables over Real Interval Field with 53 bits of precision
with coefficients:
[ 5 0.?e-14 0.?e-13 0.?e-13 ]
[ * -0.05000000000000? 0.?e-12 0.?e-12 ]
[ * * 13.00000000000? 0.?e-10 ]
[ * * * 10.8269230769? ]
```

reciprocal()

This gives the reciprocal quadratic form associated to the given form. This is defined as the multiple of the primitive adjoint with the same content as the given form.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,37])
sage: Q.reciprocal()
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 37 0 0 ]
[ * 37 0 ]
[ * * 1 ]
sage: Q.reciprocal().reciprocal()
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 0 0 ]
[ * 1 0 ]
[ * * 37 ]
sage: Q.reciprocal().reciprocal() == Q
True
```

reduced_binary_form()

Find a form which is reduced in the sense that no further binary form reductions can be done to reduce the original form.

EXAMPLES:

```
sage: QuadraticForm(ZZ, 2, [5,5,2]).reduced_binary_form()                #␣
↪optional - sage.symbolic
(
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 2 -1 ]
[ * 2 ]                                                              ,
```

(continues on next page)

(continued from previous page)

```
[ 0 -1]
[ 1  1]
)
```

reduced_binary_form1()

Reduce the form $ax^2 + bxy + cy^2$ to satisfy the reduced condition $|b| \le a \le c$, with $b \ge 0$ if $a = c$. This reduction occurs within the proper class, so all transformations are taken to have determinant 1.

EXAMPLES:

```
sage: QuadraticForm(ZZ, 2, [5,5,2]).reduced_binary_form1()              #␣
↪optional - sage.symbolic
(
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 2 -1 ]
[ * 2 ]                                                          ,

[ 0 -1]
[ 1  1]
)
```

reduced_ternary_form__Dickson()

Find the unique reduced ternary form according to the conditions of Dickson's "Studies in the Theory of Numbers", pp164-171.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1, 1, 1])
sage: Q.reduced_ternary_form__Dickson()
Traceback (most recent call last):
...
NotImplementedError
```

representation_number_list(*B*)

Return the vector of representation numbers < B.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1,1,1,1,1])
sage: Q.representation_number_list(10)                                  #␣
↪optional - sage.libs.pari
[1, 16, 112, 448, 1136, 2016, 3136, 5504, 9328, 12112]
```

representation_vector_list(*B*, *maxvectors=100000000*)

Find all vectors v where $Q(v) < B$.

This only works for positive definite quadratic forms.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1, 1])
sage: Q.representation_vector_list(10)                                  #␣
↪optional - sage.libs.pari
[[(0, 0)],
```

(continues on next page)

(continued from previous page)

```
 [(0, 1), (0, -1), (1, 0), (-1, 0)],
 [(1, 1), (-1, -1), (1, -1), (-1, 1)],
 [],
 [(0, 2), (0, -2), (2, 0), (-2, 0)],
 [(1, 2), (-1, -2), (1, -2), (-1, 2), (2, 1), (-2, -1), (2, -1), (-2, 1)],
 [],
 [],
 [(2, 2), (-2, -2), (2, -2), (-2, 2)],
 [(0, 3), (0, -3), (3, 0), (-3, 0)]]
sage: list(map(len, _))                                                  #␣
↪optional - sage.libs.pari
[1, 4, 4, 0, 4, 8, 0, 0, 4, 4]
sage: Q.representation_number_list(10)                                   #␣
↪optional - sage.libs.pari
[1, 4, 4, 0, 4, 8, 0, 0, 4, 4]
```

scale_by_factor(*c*, *change_value_ring_flag=False*)

Scale the values of the quadratic form by the number c, if this is possible while still being defined over its base ring.

If the flag is set to true, then this will alter the value ring to be the field of fractions of the original ring (if necessary).

INPUT:

- `c` – a scalar in the fraction field of the value ring of the form

OUTPUT: a quadratic form of the same dimension

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [3,9,18,27])
sage: Q.scale_by_factor(3)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 9 0 0 0 ]
[ * 27 0 0 ]
[ * * 54 0 ]
[ * * * 81 ]

sage: Q.scale_by_factor(1/3)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 0 0 0 ]
[ * 3 0 0 ]
[ * * 6 0 ]
[ * * * 9 ]
```

set_number_of_automorphisms(*num_autos*)

Set the number of automorphisms to be the value given. No error checking is performed, to this may lead to erroneous results.

The fact that this result was set externally is recorded in the internal list of external initializations, accessible by the method *list_external_initializations()*.

OUTPUT: None

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1, 1, 1])
sage: Q.list_external_initializations()
[]
sage: Q.set_number_of_automorphisms(-3)
sage: Q.number_of_automorphisms()
-3
sage: Q.list_external_initializations()
['number_of_automorphisms']
```

shimura_mass__maximal()

Use Shimura's exact mass formula to compute the mass of a maximal quadratic lattice. This works for any totally real number field, but has a small technical restriction when n is odd.

OUTPUT:

a rational number

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.shimura_mass__maximal()
```

short_primitive_vector_list_up_to_length(*len_bound*, *up_to_sign_flag=False*)

Return a list of lists of short primitive vectors v, sorted by length, with $Q(v) <$ `len_bound`. The list in output $[i]$ indexes all vectors of length i. If the `up_to_sign_flag` is set to `True`, then only one of the vectors of the pair $[v, -v]$ is listed.

Note: This processes the PARI/GP output to always give elements of type $\mathbf{Z}$.

OUTPUT: a list of lists of vectors.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.short_vector_list_up_to_length(5, True)
[[(0, 0, 0, 0)],
 [(1, 0, 0, 0)],
 [],
 [(0, 1, 0, 0)],
 [(1, 1, 0, 0), (1, -1, 0, 0), (2, 0, 0, 0)]]
sage: Q.short_primitive_vector_list_up_to_length(5, True)
[[], [(1, 0, 0, 0)], [], [(0, 1, 0, 0)], [(1, 1, 0, 0), (1, -1, 0, 0)]]
```

short_vector_list_up_to_length(*len_bound*, *up_to_sign_flag=False*)

Return a list of lists of short vectors v, sorted by length, with $Q(v) <$ `len_bound`.

INPUT:

- `len_bound` – bound for the length of the vectors
- `up_to_sign_flag` – (default: `False`) if set to True, then only one of the vectors of the pair $[v, -v]$ is listed.

OUTPUT:

A list of lists of vectors such that entry $[i]$ contains all vectors of length i.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.short_vector_list_up_to_length(3)
[[(0, 0, 0, 0)], [(1, 0, 0, 0), (-1, 0, 0, 0)], []]
sage: Q.short_vector_list_up_to_length(4)
[[(0, 0, 0, 0)],
 [(1, 0, 0, 0), (-1, 0, 0, 0)],
 [],
 [(0, 1, 0, 0), (0, -1, 0, 0)]]
sage: Q.short_vector_list_up_to_length(5)
[[(0, 0, 0, 0)],
 [(1, 0, 0, 0), (-1, 0, 0, 0)],
 [],
 [(0, 1, 0, 0), (0, -1, 0, 0)],
 [(1, 1, 0, 0),
  (-1, -1, 0, 0),
  (1, -1, 0, 0),
  (-1, 1, 0, 0),
  (2, 0, 0, 0),
  (-2, 0, 0, 0)]]
sage: Q.short_vector_list_up_to_length(5, True)
[[(0, 0, 0, 0)],
 [(1, 0, 0, 0)],
 [],
 [(0, 1, 0, 0)],
 [(1, 1, 0, 0), (1, -1, 0, 0), (2, 0, 0, 0)]]
sage: Q = QuadraticForm(matrix(6, [2, 1, 1, 1, -1, -1, 1, 2, 1, 1, -1, -1, 1, 1,
↪ 2, 0, -1, -1, 1, 1, 0, 2, 0, -1, -1, -1, -1, 0, 2, 1, -1, -1, -1, -1, 1, 2]))
sage: vs = Q.short_vector_list_up_to_length(8)
sage: [len(vs[i]) for i in range(len(vs))]
[1, 72, 270, 720, 936, 2160, 2214, 3600]
sage: vs = Q.short_vector_list_up_to_length(30)  # long time (28s on sage.math,␣
↪2014)
sage: [len(vs[i]) for i in range(len(vs))]       # long time
[1, 72, 270, 720, 936, 2160, 2214, 3600, 4590, 6552, 5184, 10800, 9360, 12240,␣
↪13500, 17712, 14760, 25920, 19710, 26064, 28080, 36000, 25920, 47520, 37638,␣
↪43272, 45900, 59040, 46800, 75600]
```

The cases of `len_bound < 2` led to exception or infinite runtime before.

```
sage: Q.short_vector_list_up_to_length(-1)
[]
sage: Q.short_vector_list_up_to_length(0)
[]
sage: Q.short_vector_list_up_to_length(1)
[[(0, 0, 0, 0, 0, 0)]]
```

In the case of quadratic forms that are not positive definite an error is raised.

```
sage: QuadraticForm(matrix(2, [2, 0, 0, -2])).short_vector_list_up_to_length(3)
Traceback (most recent call last):
...
ValueError: Quadratic form must be positive definite in order to enumerate␣
```

(continues on next page)

(continued from previous page)

```
↪short vectors
```

Check that PARI does not return vectors which are too long:

```
sage: Q = QuadraticForm(matrix(2, [72, 12, 12, 120]))
sage: len_bound_pari = 2*22953421 - 2; len_bound_pari
45906840
sage: vs = list(Q.__pari__().qfminim(len_bound_pari)[2])  # long time (18s on␣
↪sage.math, 2014)
sage: v = vs[0]; v  # long time
[66, -623]~
sage: v.Vec() * Q.__pari__() * v  # long time
45902280
```

siegel_product(u)

Compute the infinite product of local densities of the quadratic form for the number u.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.theta_series(11)
1 + 8*q + 24*q^2 + 32*q^3 + 24*q^4 + 48*q^5 + 96*q^6 + 64*q^7 + 24*q^8 + 104*q^
↪9 + 144*q^10 + O(q^11)

sage: Q.siegel_product(1)
8
sage: Q.siegel_product(2)      # This one is wrong -- expect 24, and the higher␣
↪powers of 2 don't work... =(
24
sage: Q.siegel_product(3)
32
sage: Q.siegel_product(5)
48
sage: Q.siegel_product(6)
96
sage: Q.siegel_product(7)
64
sage: Q.siegel_product(9)
104

sage: Q.local_density(2,1)
1
sage: M = 4; len([v  for v in mrange([M,M,M,M])  if Q(v) % M == 1]) / M^3
1
sage: M = 16; len([v  for v in mrange([M,M,M,M])  if Q(v) % M == 1]) / M^3  #␣
↪long time (2s on sage.math, 2014)
1

sage: Q.local_density(2,2)
3/2
sage: M = 4; len([v  for v in mrange([M,M,M,M])  if Q(v) % M == 2]) / M^3
3/2
sage: M = 16; len([v  for v in mrange([M,M,M,M])  if Q(v) % M == 2]) / M^3  #␣
```

(continues on next page)

(continued from previous page)

```
↪long time (2s on sage.math, 2014)
3/2
```

signature()

Return the signature of the quadratic form, defined as:

number of positive eigenvalues − number of negative eigenvalues

of the matrix of the quadratic form.

OUTPUT: an integer

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,0,0,-4,3,11,3])
sage: Q.signature()
3
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,-3,-4])
sage: Q.signature()
0
```

```
sage: Q = QuadraticForm(ZZ, 4, range(10)); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 0 1 2 3 ]
[ * 4 5 6 ]
[ * * 7 8 ]
[ * * * 9 ]
sage: Q.signature()
2
```

signature_vector()

Return the triple (p, n, z) of integers where

- p = number of positive eigenvalues
- n = number of negative eigenvalues
- z = number of zero eigenvalues

for the symmetric matrix associated to Q.

OUTPUT: a triple of integers ≥ 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,0,0,-4])
sage: Q.signature_vector()
(1, 1, 2)
```

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,-3,-4])
sage: Q.signature_vector()
(2, 2, 0)
```

```
sage: Q = QuadraticForm(ZZ, 4, range(10)); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 0 1 2 3 ]
[ * 4 5 6 ]
[ * * 7 8 ]
[ * * * 9 ]
sage: Q.signature_vector()
(3, 1, 0)
```

solve(*c=0*)

Return a vector x such that `self(x) == c`.

INPUT:

- `c` – (default: 0) a rational number

OUTPUT: A non-zero vector x satisfying `self(x) == c`.

ALGORITHM:

Uses PARI's pari:qfsolve. Algorithm described by Jeroen Demeyer; see comments on github issue #19112

EXAMPLES:

```
sage: F = DiagonalQuadraticForm(QQ, [1, -1]); F
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 0 ]
[ * -1 ]
sage: F.solve()
(1, 1)
sage: F.solve(1)
(1, 0)
sage: F.solve(2)
(3/2, -1/2)
sage: F.solve(3)
(2, -1)
```

```
sage: F = DiagonalQuadraticForm(QQ, [1, 1, 1, 1])
sage: F.solve(7)
(1, 2, -1, -1)
sage: F.solve()
Traceback (most recent call last):
...
ArithmeticError: no solution found (local obstruction at -1)
```

```
sage: Q = QuadraticForm(QQ, 2, [17, 94, 130])
sage: x = Q.solve(5); x
(17, -6)
sage: Q(x)
5

sage: Q.solve(6)
Traceback (most recent call last):
...
ArithmeticError: no solution found (local obstruction at 3)
```

(continues on next page)

(continued from previous page)

```
sage: G = DiagonalQuadraticForm(QQ, [5, -3, -2])
sage: x = G.solve(10); x
(3/2, -1/2, 1/2)
sage: G(x)
10

sage: F = DiagonalQuadraticForm(QQ, [1, -4])
sage: x = F.solve(); x
(2, 1)
sage: F(x)
0
```

```
sage: F = QuadraticForm(QQ, 4, [0, 0, 1, 0, 0, 0, 1, 0, 0, 0]); F
Quadratic form in 4 variables over Rational Field with coefficients:
[ 0 0 1 0 ]
[ * 0 0 1 ]
[ * * 0 0 ]
[ * * * 0 ]
sage: F.solve(23)
(23, 0, 1, 0)
```

Other fields besides the rationals are currently not supported:

```
sage: F = DiagonalQuadraticForm(GF(11), [1, 1])
sage: F.solve()
Traceback (most recent call last):
...
TypeError: solving quadratic forms is only implemented over QQ
```

split_local_cover()

Tries to find subform of the given (positive definite quaternary) quadratic form Q of the form

$$d \cdot x^2 + T(y, z, w)$$

where $d > 0$ is as small as possible.

This is done by exhaustive search on small vectors, and then comparing the local conditions of its sum with its complementary lattice and the original quadratic form Q.

OUTPUT: a *QuadraticForm* over $\mathbf{Z}$

EXAMPLES:

```
sage: Q1 = DiagonalQuadraticForm(ZZ, [7,5,3])
sage: Q1.split_local_cover()                                          #␣
↪optional - sage.symbolic
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 3 0 0 ]
[ * 5 0 ]
[ * * 7 ]
```

sum_by_coefficients_with(*right*)

Return the sum (on coefficients) of two quadratic forms of the same size.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,4,10]); Q
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 1 4 ]
[ * 10 ]
sage: Q + Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 4 0 0 ]
[ * 10 0 0 ]
[ * * 1 4 ]
[ * * * 10 ]

sage: Q2 = QuadraticForm(ZZ, 2, [1,4,-10])
sage: Q.sum_by_coefficients_with(Q2)
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 2 8 ]
[ * 0 ]
```

swap_variables(*r*, *s*, *in_place=False*)

Switch the variables x_r and x_s in the quadratic form (replacing the original form if the in_place flag is True).

INPUT:

- r, s – integers ≥ 0

OUTPUT:

a *QuadraticForm* (by default, otherwise none)

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 4, range(1,11))
sage: Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 2 3 4 ]
[ * 5 6 7 ]
[ * * 8 9 ]
[ * * * 10 ]

sage: Q.swap_variables(0,2)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 8 6 3 9 ]
[ * 5 2 7 ]
[ * * 1 4 ]
[ * * * 10 ]

sage: Q.swap_variables(0,2).swap_variables(0,2)
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 1 2 3 4 ]
[ * 5 6 7 ]
[ * * 8 9 ]
[ * * * 10 ]
```

theta_by_cholesky(*q_prec*)

Uses the real Cholesky decomposition to compute (the q-expansion of) the theta function of the quadratic form as a power series in q with terms correct up to the power $q^{\text{q\_prec}}$. (So its error is $O(q^{\text{q\_prec}+1})$.)

REFERENCE:

Cohen's "A Course in Computational Algebraic Number Theory" book, p 102.

EXAMPLES:

```
# Check the sum of 4 squares form against Jacobi's formula
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Theta = Q.theta_by_cholesky(10)
sage: Theta
1 + 8*q + 24*q^2 + 32*q^3 + 24*q^4 + 48*q^5 + 96*q^6
 + 64*q^7 + 24*q^8 + 104*q^9 + 144*q^10
sage: Expected = [1] + [8*sum([d for d in divisors(n) if d%4 != 0])
....:                    for n in range(1, 11)]
sage: Expected
[1, 8, 24, 32, 24, 48, 96, 64, 24, 104, 144]
sage: Theta.list() == Expected
True
```

```
# Check the form x^2 + 3y^2 + 5z^2 + 7w^2 represents everything except 2 and 22.
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Theta = Q.theta_by_cholesky(50)
sage: Theta_list = Theta.list()
sage: [m  for m in range(len(Theta_list))  if Theta_list[m] == 0]
[2, 22]
```

theta_by_pari(*Max*, *var_str='q'*, *safe_flag=True*)

Use PARI/GP to compute the theta function as a power series (or vector) up to the precision $O(q^{Max})$. This also caches the result for future computations.

If `var_str = ''`, then we return a vector v where `v[i]` counts the number of vectors of length i.

The `safe_flag` allows us to select whether we want a copy of the output, or the original output. It is only meaningful when a vector is returned, otherwise a copy is automatically made in creating the power series. By default `safe_flag=True`, so we return a copy of the cached information. If this is set to False, then the routine is much faster but the return values are vulnerable to being corrupted by the user.

INPUT:

- `Max` – an integer ≥ 0
- `var_str` – a string

OUTPUT: a power series or a vector

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Prec = 100
sage: compute = Q.theta_by_pari(Prec, '')                                    #␣
↪optional - sage.libs.pari
sage: exact = [1] + [8 * sum([d  for d in divisors(i)  if d % 4 != 0])      #␣
↪optional - sage.libs.pari
....:                 for i in range(1, Prec)]
```

(continues on next page)

(continued from previous page)

```
sage: compute == exact                                                    # optional - sage.libs.pari
True
```

theta_series(*Max=10*, *var_str='q'*, *safe_flag=True*)

Compute the theta series as a power series in the variable given in `var_str` (which defaults to `'q'`), up to the specified precision $O(q^{Max})$.

This uses the PARI/GP function pari:qfrep, wrapped by the theta_by_pari() method. This caches the result for future computations.

The `safe_flag` allows us to select whether we want a copy of the output, or the original output. It is only meaningful when a vector is returned, otherwise a copy is automatically made in creating the power series. By default `safe_flag` = True, so we return a copy of the cached information. If this is set to False, then the routine is much faster but the return values are vulnerable to being corrupted by the user.

Todo: Allow the option `Max='mod_form'` to give enough coefficients to ensure we determine the theta series as a modular form. This is related to the Sturm bound, but we will need to be careful about this (particularly for half-integral weights!).

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.theta_series()                                                    # optional - sage.libs.pari
1 + 2*q + 2*q^3 + 6*q^4 + 2*q^5 + 4*q^6 + 6*q^7 + 8*q^8 + 14*q^9 + O(q^10)

sage: Q.theta_series(25)                                                  # optional - sage.libs.pari
1 + 2*q + 2*q^3 + 6*q^4 + 2*q^5 + 4*q^6 + 6*q^7 + 8*q^8 + 14*q^9 + 4*q^10
+ 12*q^11 + 18*q^12 + 12*q^13 + 12*q^14 + 8*q^15 + 34*q^16 + 12*q^17 + 8*q^18
+ 32*q^19 + 10*q^20 + 28*q^21 + 16*q^23 + 44*q^24 + O(q^25)
```

theta_series_degree_2(*Q*, *prec*)

Compute the theta series of degree 2 for the quadratic form Q.

INPUT:

- `prec` – an integer

OUTPUT:

dictionary, where:

- keys are $\mathrm{GL}_2(\mathbf{Z})$-reduced binary quadratic forms (given as triples of coefficients)
- values are coefficients

EXAMPLES:

```
sage: Q2 = QuadraticForm(ZZ, 4, [1,1,1,1, 1,0,0, 1,0, 1])
sage: S = Q2.theta_series_degree_2(10)
sage: S[(0,0,2)]
24
sage: S[(1,0,1)]
```

(continues on next page)

(continued from previous page)

```
144
sage: S[(1,1,1)]
192
```

AUTHORS:

- Gonzalo Tornaria (2010-03-23)

REFERENCE:

- Raum, Ryan, Skoruppa, Tornaria, 'On Formal Siegel Modular Forms' (preprint)

vectors_by_length(*bound*)

Return a list of short vectors together with their values.

This is a naive algorithm which uses the Cholesky decomposition, but does not use the LLL-reduction algorithm.

INPUT:

- bound – an integer ≥ 0

OUTPUT:

- a list L of length (bound + 1) whose entry L[i] is a list of all vectors of length i.

REFERENCES:

This is a slightly modified version of Cohn's Algorithm 2.7.5 in "A Course in Computational Number Theory", with the increment step moved around and slightly re-indexed to allow clean looping.

Note: We could speed this up for very skew matrices by using LLL first, and then changing coordinates back, but for our purposes the simpler method is efficient enough.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1])
sage: Q.vectors_by_length(5)                                              #␣
↪optional - sage.symbolic
[[[0, 0]],
 [[0, -1], [-1, 0]],
 [[-1, -1], [1, -1]],
 [],
 [[0, -2], [-2, 0]],
 [[-1, -2], [1, -2], [-2, -1], [2, -1]]]
```

```
sage: Q1 = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q1.vectors_by_length(5)                                             #␣
↪optional - sage.symbolic
[[[0, 0, 0, 0]],
 [[-1, 0, 0, 0]],
 [],
 [[0, -1, 0, 0]],
 [[-1, -1, 0, 0], [1, -1, 0, 0], [-2, 0, 0, 0]],
 [[0, 0, -1, 0]]]
```

```
sage: Q = QuadraticForm(ZZ, 4, [1,1,1,1, 1,0,0, 1,0, 1])
sage: list(map(len, Q.vectors_by_length(2)))                              #␣
↪optional - sage.symbolic
[1, 12, 12]
```

```
sage: Q = QuadraticForm(ZZ, 4, [1,-1,-1,-1, 1,0,0, 4,-3, 4])
sage: list(map(len, Q.vectors_by_length(3)))                              #␣
↪optional - sage.symbolic
[1, 3, 0, 3]
```

xi(*p*)

Return the value of the genus characters Xi_p… which may be missing one character. We allow -1 as a prime.

REFERENCES:

Dickson's "Studies in the Theory of Numbers"

EXAMPLES:

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 1, 1, 14, 3, 14])
sage: Q2 = QuadraticForm(ZZ, 3, [2, -1, 0, 2, 0, 50])
sage: [Q1.omega(), Q2.omega()]
[5, 5]
sage: [Q1.hasse_invariant(5),                     # equivalent over Q_5       #␣
↪optional - sage.libs.pari
....:  Q2.hasse_invariant(5)]
[1, 1]
sage: [Q1.xi(5), Q2.xi(5)]                        # not equivalent over Z_5   #␣
↪optional - sage.libs.pari
[1, -1]
```

xi_rec(*p*)

Return Xi(*p*) for the reciprocal form.

EXAMPLES:

```
sage: Q1 = QuadraticForm(ZZ, 3, [1, 1, 1, 14, 3, 14])
sage: Q2 = QuadraticForm(ZZ, 3, [2, -1, 0, 2, 0, 50])
sage: [Q1.clifford_conductor(),                   # equivalent over Q         #␣
↪optional - sage.libs.pari
....:  Q2.clifford_conductor()]
[3, 3]
sage: Q1.is_locally_equivalent_to(Q2)             # not in the same genus     #␣
↪optional - sage.libs.pari
False
sage: [Q1.delta(), Q2.delta()]                                                #␣
↪optional - sage.libs.pari
[480, 480]
sage: factor(480)                                                             #␣
↪optional - sage.libs.pari
2^5 * 3 * 5
sage: list(map(Q1.xi_rec, [-1,2,3,5]))                                        #␣
↪optional - sage.libs.pari
```

(continues on next page)

(continued from previous page)

```
[-1, -1, -1, 1]
sage: list(map(Q2.xi_rec, [-1,2,3,5]))                          #␣
↪optional - sage.libs.pari
[-1, -1, -1, -1]
```

sage.quadratic_forms.quadratic_form.**is_QuadraticForm**(*Q*)

Determine if the object Q is an element of the *QuadraticForm* class.

This function is deprecated.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: from sage.quadratic_forms.quadratic_form import is_QuadraticForm
sage: is_QuadraticForm(Q)
doctest:...: DeprecationWarning: the function is_QuadraticForm is deprecated;
use isinstance(x, sage.quadratic_forms.quadratic_form.QuadraticForm) instead...
True
sage: is_QuadraticForm(2)
False
```

sage.quadratic_forms.quadratic_form.**quadratic_form_from_invariants**(*F*, *rk*, *det*, *P*, *sminus*)

Return a rational quadratic form with given invariants.

INPUT:

- F – the base field; currently only QQ is allowed
- rk – integer; the rank
- det – rational; the determinant
- P – a list of primes where Cassel's Hasse invariant is negative
- sminus – integer; the number of negative eigenvalues of any Gram matrix

OUTPUT:

- a quadratic form with the specified invariants

Let $(a_1, \ldots, a_n)$ be the gram marix of a regular quadratic space. Then Cassel's Hasse invariant is defined as

$$\prod_{i<j}(a_i, a_j),$$

where (a_i, a_j) denotes the Hilbert symbol.

ALGORITHM:

We follow [Kir2016].

EXAMPLES:

```
sage: P = [3,5]
sage: q = quadratic_form_from_invariants(QQ,2,-15,P,1)
sage: q
Quadratic form in 2 variables over Rational Field with coefficients:
[ 5 0 ]
[ * -3 ]
```

(continues on next page)

(continued from previous page)

```
sage: all(q.hasse_invariant(p) == -1 for p in P)
True
```

CHAPTER

TWO

BINARY QUADRATIC FORMS WITH INTEGER COEFFICIENTS

This module provides a specialized class for working with a binary quadratic form $ax^2 + bxy + cy^2$, stored as a triple of integers (a, b, c).

EXAMPLES:

```
sage: Q = BinaryQF([1, 2, 3])
sage: Q
x^2 + 2*x*y  + 3*y^2
sage: Q.discriminant()
-8
sage: Q.reduced_form()                                                      #␣
↪optional - sage.libs.pari
x^2 + 2*y^2
sage: Q(1, 1)
6
```

AUTHORS:

- Jon Hanke (2006-08-08):
  - Appended to add the methods *BinaryQF_reduced_representatives()*, *is_reduced()*, and `__add__` on 8-3-2006 for Coding Sprint #2.
  - Added Documentation and *complex_point()* method on 8-8-2006.
- Nick Alexander: add doctests and clean code for Doc Days 2
- William Stein (2009-08-05): composition; some ReSTification.
- William Stein (2009-09-18): make immutable.
- Justin C. Walker (2011-02-06): Add support for indefinite forms.

class sage.quadratic_forms.binary_qf.**BinaryQF**(*a*, *b=None*, *c=None*)

Bases: `SageObject`

A binary quadratic form over $\mathbf{Z}$.

INPUT:

One of the following:

- `a` – either a 3-tuple of integers, or a quadratic homogeneous polynomial in two variables with integer coefficients
- `a`, `b`, `c` – three integers

OUTPUT:

The binary quadratic form $ax^2 + bxy + cy^2$.

EXAMPLES:

```
sage: b = BinaryQF([1, 2, 3])
sage: b.discriminant()
-8
sage: b1 = BinaryQF(1, 2, 3)
sage: b1 == b
True
sage: R.<x, y> = ZZ[]
sage: BinaryQF(x^2 + 2*x*y + 3*y^2) == b
True
sage: BinaryQF(1, 0, 1)
x^2 + y^2
```

complex_point()

Return the point in the complex upper half-plane associated to `self`.

This form, $ax^2 + bxy + cy^2$, must be definite with negative discriminant $b^2 - 4ac < 0$.

OUTPUT:

- the unique complex root of $ax^2 + bx + c$ with positive imaginary part

EXAMPLES:

```
sage: Q = BinaryQF([1, 0, 1])
sage: Q.complex_point()
1.00000000000000*I
```

content()

Return the content of the form, i.e., the $\gcd$ of the coefficients.

EXAMPLES:

```
sage: Q = BinaryQF(22, 14, 10)
sage: Q.content()
2
sage: Q = BinaryQF(4, 4, -15)
sage: Q.content()
1
```

cycle(*proper=False*)

Return the cycle of reduced forms to which `self` belongs.

This is based on Algorithm 6.1 of [BUVO2007].

INPUT:

- `self` – reduced, indefinite form of non-square discriminant
- `proper` – boolean (default: `False`); if `True`, return the proper cycle

The proper cycle of a form f consists of all reduced forms that are properly equivalent to f. This is useful when testing for proper equivalence (or equivalence) between indefinite forms.

The cycle of f is a technical tool that is used when computing the proper cycle. Our definition of the cycle is slightly different from the one in [BUVO2007]. In our definition, the cycle consists of all reduced forms g, such that the a-coefficient of g has the same sign as the a-coefficient of f, and g can be obtained from f by performing a change of variables, and then multiplying by the determinant of the change-of-variables matrix. It is important to note that g might not be equivalent to f (because of multiplying by the determinant). However, either g or $-g$ must be equivalent to f. Also note that the cycle does contain f. (Under the definition in [BUVO2007], the cycle might not contain f, because all forms in the cycle are required to have positive a-coefficient, even if the a-coefficient of f is negative.)

EXAMPLES:

```
sage: Q = BinaryQF(14, 17, -2)
sage: Q.cycle()
[14*x^2 + 17*x*y - 2*y^2,
 2*x^2 + 19*x*y - 5*y^2,
 5*x^2 + 11*x*y - 14*y^2]
sage: Q.cycle(proper=True)
[14*x^2 + 17*x*y - 2*y^2,
 -2*x^2 + 19*x*y + 5*y^2,
 5*x^2 + 11*x*y - 14*y^2,
 -14*x^2 + 17*x*y + 2*y^2,
 2*x^2 + 19*x*y - 5*y^2,
 -5*x^2 + 11*x*y + 14*y^2]

sage: Q = BinaryQF(1, 8, -3)
sage: Q.cycle()
[x^2 + 8*x*y - 3*y^2,
 3*x^2 + 4*x*y - 5*y^2,
 5*x^2 + 6*x*y - 2*y^2,
 2*x^2 + 6*x*y - 5*y^2,
 5*x^2 + 4*x*y - 3*y^2,
 3*x^2 + 8*x*y - y^2]
sage: Q.cycle(proper=True)
[x^2 + 8*x*y - 3*y^2,
 -3*x^2 + 4*x*y + 5*y^2,
  5*x^2 + 6*x*y - 2*y^2,
  -2*x^2 + 6*x*y + 5*y^2,
  5*x^2 + 4*x*y - 3*y^2,
  -3*x^2 + 8*x*y + y^2]

sage: Q = BinaryQF(1, 7, -6)
sage: Q.cycle()
[x^2 + 7*x*y - 6*y^2,
 6*x^2 + 5*x*y - 2*y^2,
 2*x^2 + 7*x*y - 3*y^2,
 3*x^2 + 5*x*y - 4*y^2,
 4*x^2 + 3*x*y - 4*y^2,
 4*x^2 + 5*x*y - 3*y^2,
 3*x^2 + 7*x*y - 2*y^2,
 2*x^2 + 5*x*y - 6*y^2,
 6*x^2 + 7*x*y - y^2]
```

det()

Return the determinant of the matrix associated to `self`.

The determinant is used by Gauss and by Conway-Sloane, for whom an integral quadratic form has coefficients $(a, 2b, c)$ with a, b, c integers.

OUTPUT:

The determinant of the matrix:

```
[  a  b/2]
[b/2    c]
```

as a rational.

REMARK:

This is just $-D/4$ where D is the discriminant. The return type is rational even if b (and hence D) is even.

EXAMPLES:

```
sage: q = BinaryQF(1, -1, 67)
sage: q.determinant()
267/4
```

determinant()

Return the determinant of the matrix associated to `self`.

The determinant is used by Gauss and by Conway-Sloane, for whom an integral quadratic form has coefficients $(a, 2b, c)$ with a, b, c integers.

OUTPUT:

The determinant of the matrix:

```
[  a  b/2]
[b/2    c]
```

as a rational.

REMARK:

This is just $-D/4$ where D is the discriminant. The return type is rational even if b (and hence D) is even.

EXAMPLES:

```
sage: q = BinaryQF(1, -1, 67)
sage: q.determinant()
267/4
```


discriminant()

Return the discriminant of `self`.

Given a form $ax^2 + bxy + cy^2$, this returns $b^2 - 4ac$.

EXAMPLES:

```
sage: Q = BinaryQF([1, 2, 3])
sage: Q.discriminant()
-8
```

static from_polynomial(*poly*)

Construct a *BinaryQF* from a bivariate polynomial with integer coefficients. Inverse of *polynomial()*.

EXAMPLES:

```
sage: R.<u,v> = ZZ[]
sage: f = u^2 + 419*v^2
sage: Q = BinaryQF.from_polynomial(f); Q
x^2 + 419*y^2
sage: Q.polynomial()
x^2 + 419*y^2
sage: Q.polynomial()(R.gens()) == f
True
```

The method fails if the given polynomial is not a quadratic form:

```
sage: BinaryQF.from_polynomial(u^3 - 5*v)
Traceback (most recent call last):
...
ValueError: polynomial has monomials of degree != 2
```

…or if the coefficients aren't integers:

```
sage: BinaryQF.from_polynomial(u^2/7 + v^2)
Traceback (most recent call last):
...
TypeError: no conversion of this rational to integer
```

has_fundamental_discriminant()

Return whether the discriminant D of this form is a fundamental discriminant (i.e. D is the smallest element of its squareclass with $D = 0$ or 1 modulo 4).

EXAMPLES:

```
sage: Q = BinaryQF([1, 0, 1])
sage: Q.discriminant()
-4
sage: Q.has_fundamental_discriminant()                                    #␣
↪optional - sage.libs.pari
True

sage: Q = BinaryQF([2, 0, 2])
sage: Q.discriminant()
-16
sage: Q.has_fundamental_discriminant()                                    #␣
↪optional - sage.libs.pari
False
```

is_equivalent(*other*, *proper=True*)

Return whether `self` is equivalent to `other`.

INPUT:

- `proper` – bool (default: `True`); if `True` use proper equivalence
- `other` – a binary quadratic form

EXAMPLES:

```
sage: Q3 = BinaryQF(4, 4, 15)
sage: Q2 = BinaryQF(4, -4, 15)
sage: Q2.is_equivalent(Q3)                                                      #␣
↪optional - sage.libs.pari
True
sage: a = BinaryQF([33, 11, 5])
sage: b = a.reduced_form(); b                                                   #␣
↪optional - sage.libs.pari
5*x^2 - x*y + 27*y^2
sage: a.is_equivalent(b)                                                        #␣
↪optional - sage.libs.pari
True
sage: a.is_equivalent(BinaryQF((3, 4, 5)))                                      #␣
↪optional - sage.libs.pari
False
```

Some indefinite examples:

```
sage: Q1 = BinaryQF(9, 8, -7)
sage: Q2 = BinaryQF(9, -8, -7)
sage: Q1.is_equivalent(Q2, proper=True)                                         #␣
↪optional - sage.libs.pari
False
sage: Q1.is_equivalent(Q2, proper=False)                                        #␣
↪optional - sage.libs.pari
True
```

is_indef()

Return whether `self` is indefinite, i.e., has positive discriminant.

EXAMPLES:

```
sage: Q = BinaryQF(1, 3, -5)
sage: Q.is_indef()
True
```

is_indefinite()

Return whether `self` is indefinite, i.e., has positive discriminant.

EXAMPLES:

```
sage: Q = BinaryQF(1, 3, -5)
sage: Q.is_indef()
True
```

is_negative_definite()

Return `True` if `self` is negative definite, i.e., has negative discriminant with $a < 0$.

EXAMPLES:

```
sage: Q = BinaryQF(-1, 3, -5)
sage: Q.is_positive_definite()
False
sage: Q.is_negative_definite()
True
```

is_negdef()

Return True if self is negative definite, i.e., has negative discriminant with $a < 0$.

EXAMPLES:

```
sage: Q = BinaryQF(-1, 3, -5)
sage: Q.is_positive_definite()
False
sage: Q.is_negative_definite()
True
```

is_nonsingular()

Return whether this form is nonsingular, i.e., has non-zero discriminant.

EXAMPLES:

```
sage: Q = BinaryQF(1, 3, -5)
sage: Q.is_nonsingular()
True
sage: Q = BinaryQF(1, 2, 1)
sage: Q.is_nonsingular()
False
```

is_posdef()

Return True if self is positive definite, i.e., has negative discriminant with $a > 0$.

EXAMPLES:

```
sage: Q = BinaryQF(195751, 37615, 1807)
sage: Q.is_positive_definite()
True
sage: Q = BinaryQF(195751, 1212121, -1876411)
sage: Q.is_positive_definite()
False
```

is_positive_definite()

Return True if self is positive definite, i.e., has negative discriminant with $a > 0$.

EXAMPLES:

```
sage: Q = BinaryQF(195751, 37615, 1807)
sage: Q.is_positive_definite()
True
sage: Q = BinaryQF(195751, 1212121, -1876411)
sage: Q.is_positive_definite()
False
```

is_primitive()

Return whether the form $ax^2 + bxy + cy^2$ satisfies $\gcd(a, b, c) = 1$, i.e., is primitive.

EXAMPLES:

```
sage: Q = BinaryQF([6, 3, 9])
sage: Q.is_primitive()
False
```

(continues on next page)

(continued from previous page)

```
sage: Q = BinaryQF([1, 1, 1])
sage: Q.is_primitive()
True

sage: Q = BinaryQF([2, 2, 2])
sage: Q.is_primitive()
False

sage: rqf = BinaryQF_reduced_representatives(-23*9, primitive_only=False)
sage: [qf.is_primitive() for qf in rqf]
[True, True, True, False, True, True, False, False, True]
sage: rqf
[x^2 + x*y + 52*y^2,
2*x^2 - x*y + 26*y^2,
2*x^2 + x*y + 26*y^2,
3*x^2 + 3*x*y + 18*y^2,
4*x^2 - x*y + 13*y^2,
4*x^2 + x*y + 13*y^2,
6*x^2 - 3*x*y + 9*y^2,
6*x^2 + 3*x*y + 9*y^2,
8*x^2 + 7*x*y + 8*y^2]
sage: [qf for qf in rqf if qf.is_primitive()]
[x^2 + x*y + 52*y^2,
2*x^2 - x*y + 26*y^2,
2*x^2 + x*y + 26*y^2,
4*x^2 - x*y + 13*y^2,
4*x^2 + x*y + 13*y^2,
8*x^2 + 7*x*y + 8*y^2]
```

See also:

content()

is_reduced()

Return whether `self` is reduced.

Let $f = ax^2 + bxy + cy^2$ be a binary quadratic form of discriminant D.

- If f is positive definite ($D < 0$ and $a > 0$), then f is reduced if and only if $|b| \leq a \leq c$, and $b \geq 0$ if either $a = b$ or $a = c$.
- If f is negative definite ($D < 0$ and $a < 0$), then f is reduced if and only if the positive definite form with coefficients $(-a, b, -c)$ is reduced.
- If f is indefinite ($D > 0$), then f is reduced if and only if $|\sqrt{D} - 2|a|| < b < \sqrt{D}$ or $[a = 0$ and $-b < 2c \leq b]$ or $[c = 0$ and $-b < 2a \leq b]$.

EXAMPLES:

```
sage: Q = BinaryQF([1, 2, 3])
sage: Q.is_reduced()
False

sage: Q = BinaryQF([2, 1, 3])
```

(continues on next page)

(continued from previous page)

```
sage: Q.is_reduced()
True

sage: Q = BinaryQF([1, -1, 1])
sage: Q.is_reduced()
False

sage: Q = BinaryQF([1, 1, 1])
sage: Q.is_reduced()
True
```

Examples using indefinite forms:

```
sage: f = BinaryQF(-1, 2, 2)
sage: f.is_reduced()
True
sage: BinaryQF(1, 9, 4).is_reduced()
False
sage: BinaryQF(1, 5, -1).is_reduced()
True
```

is_reducible()

Return whether this form is reducible and cache the result.

A binary form q is called reducible if it is the product of two linear forms $q = (ax + by)(cx + dy)$, or equivalently if its discriminant is a square.

EXAMPLES:

```
sage: q = BinaryQF([1, 0, -1])
sage: q.is_reducible()
True
```

> **Warning:** Despite the similar name, this method is unrelated to reduction of binary quadratic forms as implemented by *reduced_form()* and *is_reduced()*.

is_singular()

Return whether `self` is singular, i.e., has zero discriminant.

EXAMPLES:

```
sage: Q = BinaryQF(1, 3, -5)
sage: Q.is_singular()
False
sage: Q = BinaryQF(1, 2, 1)
sage: Q.is_singular()
True
```

is_weakly_reduced()

Check if the form $ax^2 + bxy + cy^2$ satisfies $|b| \leq a \leq c$, i.e., is weakly reduced.

EXAMPLES:

```
sage: Q = BinaryQF([1, 2, 3])
sage: Q.is_weakly_reduced()
False

sage: Q = BinaryQF([2, 1, 3])
sage: Q.is_weakly_reduced()
True

sage: Q = BinaryQF([1, -1, 1])
sage: Q.is_weakly_reduced()
True
```

is_zero()

Return whether `self` is identically zero.

EXAMPLES:

```
sage: Q = BinaryQF(195751, 37615, 1807)
sage: Q.is_zero()
False
sage: Q = BinaryQF(0, 0, 0)
sage: Q.is_zero()
True
```

matrix_action_left(M)

Return the binary quadratic form resulting from the left action of the 2-by-2 matrix M on `self`.

Here the action of the matrix $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ on the form $Q(x, y)$ produces the form $Q(ax + cy, bx + dy)$.

EXAMPLES:

```
sage: Q = BinaryQF([2, 1, 3]); Q
2*x^2 + x*y + 3*y^2
sage: M = matrix(ZZ, [[1, 2], [3, 5]])
sage: Q.matrix_action_left(M)
16*x^2 + 83*x*y + 108*y^2
```

matrix_action_right(M)

Return the binary quadratic form resulting from the right action of the 2-by-2 matrix M on `self`.

Here the action of the matrix $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ on the form $Q(x, y)$ produces the form $Q(ax + by, cx + dy)$.

EXAMPLES:

```
sage: Q = BinaryQF([2, 1, 3]); Q
2*x^2 + x*y + 3*y^2
sage: M = matrix(ZZ, [[1, 2], [3, 5]])
sage: Q.matrix_action_right(M)
32*x^2 + 109*x*y + 93*y^2
```

polynomial()

Return `self` as a homogeneous 2-variable polynomial.

EXAMPLES:

```
sage: Q = BinaryQF([1, 2, 3])
sage: Q.polynomial()
x^2 + 2*x*y + 3*y^2

sage: Q = BinaryQF([-1, -2, 3])
sage: Q.polynomial()
-x^2 - 2*x*y + 3*y^2

sage: Q = BinaryQF([0, 0, 0])
sage: Q.polynomial()
0
```

reduced_form(*transformation=False*, *algorithm='default'*)

Return a reduced form equivalent to `self`.

INPUT:

- `self` – binary quadratic form of non-square discriminant
- `transformation` – boolean (default: False): if `True`, return both the reduced form and a matrix whose *matrix_action_right()* transforms `self` into the reduced form.
- `algorithm` – string; the algorithm to use. Valid options are:
  - `'default'` – let Sage pick an algorithm (default)
  - `'pari'` – use PARI (pari:qfbred or pari:qfbredsl2)
  - `'sage'` – use Sage

See also:

- *is_reduced()*
- *is_equivalent()*

EXAMPLES:

```
sage: a = BinaryQF([33, 11, 5])
sage: a.is_reduced()
False
sage: b = a.reduced_form(); b                                          #␣
↪optional - sage.libs.pari
5*x^2 - x*y + 27*y^2
sage: b.is_reduced()                                                   #␣
↪optional - sage.libs.pari
True

sage: a = BinaryQF([15, 0, 15])
sage: a.is_reduced()
True
sage: b = a.reduced_form(); b                                          #␣
↪optional - sage.libs.pari
15*x^2 + 15*y^2
sage: b.is_reduced()                                                   #␣
↪optional - sage.libs.pari
True
```

Examples of reducing indefinite forms:

```
sage: f = BinaryQF(1, 0, -3)
sage: f.is_reduced()
False
sage: g = f.reduced_form(); g                                          #␣
↪optional - sage.libs.pari
x^2 + 2*x*y - 2*y^2
sage: g.is_reduced()                                                   #␣
↪optional - sage.libs.pari
True

sage: q = BinaryQF(1, 0, -1)
sage: q.reduced_form()                                                 #␣
↪optional - sage.libs.pari
x^2 + 2*x*y

sage: BinaryQF(1, 9, 4).reduced_form(transformation=True)              #␣
↪optional - sage.libs.pari
(
                     [ 0 -1]
4*x^2 + 7*x*y - y^2, [ 1  2]
)
sage: BinaryQF(3, 7, -2).reduced_form(transformation=True)             #␣
↪optional - sage.libs.pari
(
                       [1 0]
3*x^2 + 7*x*y - 2*y^2, [0 1]
)
sage: BinaryQF(-6, 6, -1).reduced_form(transformation=True)            #␣
↪optional - sage.libs.pari
(
                      [ 0 -1]
-x^2 + 2*x*y + 2*y^2, [ 1 -4]
)
```

`small_prime_value`(*Bmax=1000*)

Return a prime represented by this (primitive positive definite) binary form.

INPUT:

- `Bmax` – a positive bound on the representing integers

OUTPUT: a prime number represented by the form

Note: This is a very elementary implementation which just substitutes values until a prime is found.

EXAMPLES:

```
sage: [Q.small_prime_value()                                           #␣
↪optional - sage.libs.pari
....:  for Q in BinaryQF_reduced_representatives(-23, primitive_only=True)]
[23, 2, 2]
sage: [Q.small_prime_value()                                           #␣
```

(continues on next page)

(continued from previous page)

```
↪optional - sage.libs.pari
....:  for Q in BinaryQF_reduced_representatives(-47, primitive_only=True)]
[47, 2, 2, 3, 3]
```

solve_integer(*n*, *algorithm*)

Solve $Q(x, y) = n$ in integers x and y where Q is this quadratic form.

INPUT:

- `n` – a positive integer
- `algorithm` – `"general"` (default) or `"cornacchia"`

To use the Cornacchia algorithm, the quadratic form must have $a = 1$ and $b = 0$ and $c > 0$, and `n` must be a prime or four times a prime (but this is not checked).

OUTPUT:

A tuple (x, y) of integers satisfying $Q(x, y) = n$, or `None` if no solution exists.

ALGORITHM: pari:qfbsolve or pari:qfbcornacchia

EXAMPLES:

```
sage: Q = BinaryQF([1, 0, 419])
sage: Q.solve_integer(773187972)                                         #␣
↪optional - sage.libs.pari
(4919, 1337)
```

If Q is of the form $[1, 0, c]$ as above and n is a prime or four times a prime, Cornacchia's algorithm can be used, which is typically much faster than the general method:

```
sage: Q = BinaryQF([1, 0, 12345])
sage: n = 2^99 + 5273
sage: Q.solve_integer(n)
(-67446480057659, 7139620553488)
sage: Q.solve_integer(n, algorithm='cornacchia')
(67446480057659, 7139620553488)
sage: timeit('Q.solve_integer(n)')                          # not tested
125 loops, best of 3: 3.13 ms per loop
sage: timeit('Q.solve_integer(n, algorithm="cornacchia")')  # not tested
625 loops, best of 3: 18.6 μs per loop
```

```
sage: Qs = BinaryQF_reduced_representatives(-23, primitive_only=True)    #␣
↪optional - sage.libs.pari
sage: Qs                                                                  #␣
↪optional - sage.libs.pari
[x^2 + x*y + 6*y^2, 2*x^2 - x*y + 3*y^2, 2*x^2 + x*y + 3*y^2]
sage: [Q.solve_integer(3) for Q in Qs]                                    #␣
↪optional - sage.libs.pari
[None, (0, -1), (0, -1)]
sage: [Q.solve_integer(5) for Q in Qs]                                    #␣
↪optional - sage.libs.pari
[None, None, None]
sage: [Q.solve_integer(6) for Q in Qs]                                    #␣
```

(continues on next page)

(continued from previous page)

```
↪optional - sage.libs.pari
[(1, -1), (1, -1), (-1, -1)]
```

sage.quadratic_forms.binary_qf.**BinaryQF_reduced_representatives**(*D*, *primitive_only=False*, *proper=True*)

Return representatives for the classes of binary quadratic forms of discriminant D.

INPUT:

- D – (integer) a discriminant
- primitive_only – (boolean; default: True): if True, only return primitive forms.
- proper – (boolean; default: True)

OUTPUT:

(list) A lexicographically-ordered list of inequivalent reduced representatives for the (im)proper equivalence classes of binary quadratic forms of discriminant D. If primitive_only is True then imprimitive forms (which only exist when D is not fundamental) are omitted; otherwise they are included.

EXAMPLES:

```
sage: BinaryQF_reduced_representatives(-4)
[x^2 + y^2]

sage: BinaryQF_reduced_representatives(-163)
[x^2 + x*y + 41*y^2]

sage: BinaryQF_reduced_representatives(-12)
[x^2 + 3*y^2, 2*x^2 + 2*x*y + 2*y^2]

sage: BinaryQF_reduced_representatives(-16)
[x^2 + 4*y^2, 2*x^2 + 2*y^2]

sage: BinaryQF_reduced_representatives(-63)
[x^2 + x*y + 16*y^2, 2*x^2 - x*y + 8*y^2, 2*x^2 + x*y + 8*y^2,
 3*x^2 + 3*x*y + 6*y^2, 4*x^2 + x*y + 4*y^2]
```

The number of inequivalent reduced binary forms with a fixed negative fundamental discriminant D is the class number of the quadratic field $\mathbf{Q}(\sqrt{D})$:

```
sage: len(BinaryQF_reduced_representatives(-13*4))
2
sage: QuadraticField(-13*4, 'a').class_number()
2
sage: p = next_prime(2^20); p
1048583
sage: len(BinaryQF_reduced_representatives(-p))                        #␣
↪optional - sage.libs.pari
689
sage: QuadraticField(-p, 'a').class_number()                           #␣
↪optional - sage.libs.pari sage.rings.number_field
689
```

(continues on next page)

(continued from previous page)

```
sage: BinaryQF_reduced_representatives(-23*9)
[x^2 + x*y + 52*y^2,
2*x^2 - x*y + 26*y^2,
2*x^2 + x*y + 26*y^2,
3*x^2 + 3*x*y + 18*y^2,
4*x^2 - x*y + 13*y^2,
4*x^2 + x*y + 13*y^2,
6*x^2 - 3*x*y + 9*y^2,
6*x^2 + 3*x*y + 9*y^2,
8*x^2 + 7*x*y + 8*y^2]
sage: BinaryQF_reduced_representatives(-23*9, primitive_only=True)          #␣
↪optional - sage.libs.pari
[x^2 + x*y + 52*y^2,
2*x^2 - x*y + 26*y^2,
2*x^2 + x*y + 26*y^2,
4*x^2 - x*y + 13*y^2,
4*x^2 + x*y + 13*y^2,
8*x^2 + 7*x*y + 8*y^2]
```

CHAPTER

THREE

CONSTRUCTIONS OF QUADRATIC FORMS

sage.quadratic_forms.constructions.**BezoutianQuadraticForm**(*f*, *g*)

Compute the Bezoutian of two polynomials defined over a common base ring. This is defined by

$$\mathrm{Bez}(f,g) := \frac{f(x)g(y) - f(y)g(x)}{y - x}$$

and has size defined by the maximum of the degrees of f and g.

INPUT:

- `f`, `g` – polynomials in $R[x]$, for some ring R

OUTPUT: a quadratic form over R

EXAMPLES:

```
sage: R = PolynomialRing(ZZ, 'x')
sage: f = R([1,2,3])
sage: g = R([2,5])
sage: Q = BezoutianQuadraticForm(f, g); Q                                    #␣
↪optional - sage.libs.singular
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 1 -12 ]
[ * -15 ]
```

AUTHORS:

- Fernando Rodriguez-Villegas, Jonathan Hanke – added on 11/9/2008

sage.quadratic_forms.constructions.**HyperbolicPlane_quadratic_form**(*R*, *r=1*)

Constructs the direct sum of r copies of the quadratic form xy representing a hyperbolic plane defined over the base ring R.

INPUT:

- R: a ring
- `n` (integer, default 1) number of copies

EXAMPLES:

```
sage: HyperbolicPlane_quadratic_form(ZZ)
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 0 1 ]
[ * 0 ]
```

CHAPTER

FOUR

RANDOM QUADRATIC FORMS

This file contains a set of routines to create a random quadratic form.

sage.quadratic_forms.random_quadraticform.**random_quadraticform**(*R*, *n*, *rand_arg_list=[]*)

Create a random quadratic form in n variables defined over the ring R.

The last (and optional) argument `rand_arg_list` is a list of at most 3 elements which is passed (as at most 3 separate variables) into the method `R.random_element()`.

INPUT:

- `R` – a ring
- `n` – an integer ≥ 0
- `rand_arg_list` – a list of at most 3 arguments which can be taken by `R.random_element()`.

OUTPUT: A quadratic form over the ring R.

EXAMPLES:

```
sage: random_quadraticform(ZZ, 3, [1,5])    # random
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 3 2 3 ]
[ * 1 4 ]
[ * * 3 ]

sage: random_quadraticform(ZZ, 3, [-5,5])    # random
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 3 2 -5 ]
[ * 2 -2 ]
[ * * -5 ]

sage: random_quadraticform(ZZ, 3, [-50,50])    # random
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 8 -23 ]
[ * 0 0 ]
[ * * 6 ]
```

sage.quadratic_forms.random_quadraticform.**random_quadraticform_with_conditions**(*R*, *n*, *condition_list=[]*, *rand_arg_list=[]*)

Create a random quadratic form in n variables defined over the ring R satisfying a list of boolean (i.e. True/False) conditions.

The conditions c appearing in the list must be boolean functions which can be called either as `Q.c()` or `c(Q)`, where `Q` is the random quadratic form.

The last (and optional) argument `rand_arg_list` is a list of at most 3 elements which is passed (as at most 3 separate variables) into the method `R.random_element()`.

EXAMPLES:

```
sage: check = QuadraticForm.is_positive_definite
sage: Q = random_quadraticform_with_conditions(ZZ, 3, [check], [-5, 5])
sage: Q    # random
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 3 -2 -5 ]
[ * 2 2 ]
[ * * 3 ]
```

`sage.quadratic_forms.random_quadraticform.`**`random_ternaryqf`**`(`*rand_arg_list=[]*`)`

Create a random ternary quadratic form.

The last (and optional) argument `rand_arg_list` is a list of at most 3 elements which is passed (as at most 3 separate variables) into the method `R.random_element()`.

INPUT:

- `rand_arg_list` – a list of at most 3 arguments which can be taken by `R.random_element()`.

OUTPUT: A ternary quadratic form.

EXAMPLES:

```
sage: random_ternaryqf()  # random
Ternary quadratic form with integer coefficients:
[1 1 4]
[-1 1 -1]
sage: random_ternaryqf([-1, 2])  # random
Ternary quadratic form with integer coefficients:
[1 0 1]
[-1 -1 -1]
sage: random_ternaryqf([-10, 10, "uniform"])  # random
Ternary quadratic form with integer coefficients:
[7 -8 2]
[0 3 -6]
```

`sage.quadratic_forms.random_quadraticform.`**`random_ternaryqf_with_conditions`**`(`*condition_list=[]*, *rand_arg_list=[]*`)`

Create a random ternary quadratic form satisfying a list of boolean (i.e. True/False) conditions.

The conditions c appearing in the list must be boolean functions which can be called either as `Q.c()` or `c(Q)`, where `Q` is the random ternary quadratic form.

The last (and optional) argument `rand_arg_list` is a list of at most 3 elements which is passed (as at most 3 separate variables) into the method `R.random_element()`.

EXAMPLES:

```
sage: check = TernaryQF.is_positive_definite
sage: Q = random_ternaryqf_with_conditions([check], [-5, 5])
sage: Q    # random
```

(continues on next page)

(continued from previous page)

```
Ternary quadratic form with integer coefficients:
[3 4 2]
[2 -2 -1]
```

CHAPTER

FIVE

ROUTINES FOR COMPUTING SPECIAL VALUES OF L-FUNCTIONS

- *gamma__exact()* – Exact values of the Γ function at integers and half-integers
- *zeta__exact()* – Exact values of the Riemann ζ function at critical values
- *quadratic_L_function__exact()* – Exact values of the Dirichlet L-functions of quadratic characters at critical values
- *quadratic_L_function__numerical()* – Numerical values of the Dirichlet L-functions of quadratic characters in the domain of convergence

sage.quadratic_forms.special_values.**QuadraticBernoulliNumber**(k, d)

Compute k-th Bernoulli number for the primitive quadratic character associated to $\chi(x) = \left(\frac{d}{x}\right)$.

EXAMPLES:

Let us create a list of some odd negative fundamental discriminants:

```
sage: test_set = [d for d in srange(-163, -3, 4)                    # optional - sage.libs.pari
....:             if d.is_fundamental_discriminant()]
```

In general, we have $B_{1,\chi_d} = -2h/w$ for odd negative fundamental discriminants:

```
sage: all(QuadraticBernoulliNumber(1, d)                            # optional - sage.libs.pari
....:       == -len(BinaryQF_reduced_representatives(d))
....:     for d in test_set)
True
```

REFERENCES:

- [Iwa1972], pp 7-16.

sage.quadratic_forms.special_values.**gamma__exact**(n)

Evaluates the exact value of the Γ function at an integer or half-integer argument.

EXAMPLES:

```
sage: gamma__exact(4)
6
sage: gamma__exact(3)
2
sage: gamma__exact(2)
1
sage: gamma__exact(1)
```

(continues on next page)

(continued from previous page)

```
1

sage: gamma__exact(1/2)                                                    #␣
↪optional - sage.symbolic
sqrt(pi)
sage: gamma__exact(3/2)                                                    #␣
↪optional - sage.symbolic
1/2*sqrt(pi)
sage: gamma__exact(5/2)                                                    #␣
↪optional - sage.symbolic
3/4*sqrt(pi)
sage: gamma__exact(7/2)                                                    #␣
↪optional - sage.symbolic
15/8*sqrt(pi)

sage: gamma__exact(-1/2)                                                   #␣
↪optional - sage.symbolic
-2*sqrt(pi)
sage: gamma__exact(-3/2)                                                   #␣
↪optional - sage.symbolic
4/3*sqrt(pi)
sage: gamma__exact(-5/2)                                                   #␣
↪optional - sage.symbolic
-8/15*sqrt(pi)
sage: gamma__exact(-7/2)                                                   #␣
↪optional - sage.symbolic
16/105*sqrt(pi)
```

sage.quadratic_forms.special_values.**quadratic_L_function__exact**(*n*, *d*)

Return the exact value of a quadratic twist of the Riemann Zeta function by $\chi_d(x) = \left(\frac{d}{x}\right)$.

The input n must be a critical value.

EXAMPLES:

```
sage: quadratic_L_function__exact(1, -4)                                   #␣
↪optional - sage.libs.pari sage.symbolic
1/4*pi
sage: quadratic_L_function__exact(-4, -4)                                  #␣
↪optional - sage.libs.pari
5/2
sage: quadratic_L_function__exact(2, 1)                                    #␣
↪optional - sage.libs.pari sage.symbolic
1/6*pi^2
```

REFERENCES:

- [Iwa1972], pp 16-17, Special values of $L(1-n,\chi)$ and $L(n,\chi)$
- [IR1990]
- [Was1997]

sage.quadratic_forms.special_values.**quadratic_L_function__numerical**(*n*, *d*, *num_terms=1000*)

Evaluate the Dirichlet L-function (for quadratic character) numerically (in a very naive way).

EXAMPLES:

First, let us test several values for a given character:

```
sage: RR = RealField(100)
sage: for i in range(5):
....:     print("L({}, (-4/.)): {}".format(1+2*i,
....:             RR(quadratic_L_function__exact(1+2*i, -4))
....:                - quadratic_L_function__numerical(RR(1+2*i), -4, 10000)))
L(1, (-4/.)): 0.000049999999500000024999996962707
L(3, (-4/.)): 4.99999970000003...e-13
L(5, (-4/.)): 4.99999922759382...e-21
L(7, (-4/.)): ...e-29
L(9, (-4/.)): ...e-29
```

This procedure fails for negative special values, as the Dirichlet series does not converge here:

```
sage: quadratic_L_function__numerical(-3, -4, 10000)
Traceback (most recent call last):
...
ValueError: the Dirichlet series does not converge here
```

Test for several characters that the result agrees with the exact value, to a given accuracy

```
sage: for d in range(-20,0):  # long time (2s on sage.math 2014)
....:     if abs(RR(quadratic_L_function__numerical(1, d, 10000)
....:                - quadratic_L_function__exact(1, d))) > 0.001:
....:         print("We have a problem at d = {}: exact = {}, numerical = {}".
↪format(d,
....:                 RR(quadratic_L_function__exact(1, d)),
....:                 RR(quadratic_L_function__numerical(1, d))))
```

sage.quadratic_forms.special_values.**zeta__exact**(*n*)

Return the exact value of the Riemann Zeta function

The argument must be a critical value, namely either positive even or negative odd.

See for example [Iwa1972], p13, Special value of $\zeta(2k)$

EXAMPLES:

Let us test the accuracy for negative special values:

```
sage: RR = RealField(100)
sage: for i in range(1,10):
....:     print("zeta({}): {}".format(1-2*i, RR(zeta__exact(1-2*i)) - zeta(RR(1-
↪2*i))))
zeta(-1): 0.00000000000000000000000000000
zeta(-3): 0.00000000000000000000000000000
zeta(-5): 0.00000000000000000000000000000
zeta(-7): 0.00000000000000000000000000000
zeta(-9): 0.00000000000000000000000000000
zeta(-11): 0.00000000000000000000000000000
zeta(-13): 0.00000000000000000000000000000
zeta(-15): 0.00000000000000000000000000000
zeta(-17): 0.00000000000000000000000000000
```

Let us test the accuracy for positive special values:

```
sage: all(abs(RR(zeta__exact(2*i)) - zeta(RR(2*i))) < 10**(-28) for i in range(1,
↪10))
True
```

REFERENCES:

- [Iwa1972]
- [IR1990]
- [Was1997]

CHAPTER

SIX

OPTIMIZED COUNTING OF CONGRUENCE SOLUTIONS

sage.quadratic_forms.count_local_2.**CountAllLocalTypesNaive**(*Q*, *p*, *k*, *m*, *zvec*, *nzvec*)

This is an internal routine, which is called by `sage.quadratic_forms.quadratic_form.QuadraticForm.count_congruence_solutions_by_type` `QuadraticForm.count_congruence_solutions_by_type()`. See the documentation of that method for more details.

INPUT:

- `Q` – quadratic form over $\mathbf{Z}$
- `p` – prime number > 0
- `k` – an integer > 0
- `m` – an integer (depending only on mod p^k)
- `zvec`, `nzvec` – a list of integers in `range(Q.dim())`, or `None`

OUTPUT:

a list of six integers ≥ 0 representing the solution types: `[All, Good, Zero, Bad, BadI, BadII]`

EXAMPLES:

```
sage: from sage.quadratic_forms.count_local_2 import CountAllLocalTypesNaive
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3])
sage: CountAllLocalTypesNaive(Q, 3, 1, 1, None, None)
[6, 6, 0, 0, 0, 0]
sage: CountAllLocalTypesNaive(Q, 3, 1, 2, None, None)
[6, 6, 0, 0, 0, 0]
sage: CountAllLocalTypesNaive(Q, 3, 1, 0, None, None)
[15, 12, 1, 2, 0, 2]
```

sage.quadratic_forms.count_local_2.**count_modp__by_gauss_sum**(*n*, *p*, *m*, *Qdet*)

Return the number of solutions of $Q(x) = m$ over the finite field $\mathbf{Z}/p\mathbf{Z}$, where p is a prime number > 2 and Q is a non-degenerate quadratic form of dimension $n \geq 1$ and has Gram determinant `Qdet`.

REFERENCE:

These are defined in Table 1 on p363 of Hanke's "Local Densities…" paper.

INPUT:

- `n` – an integer ≥ 1
- `p` – a prime number > 2
- `m` – an integer
- `Qdet` – a integer which is non-zero mod p

OUTPUT: an integer ≥ 0

EXAMPLES:

```
sage: from sage.quadratic_forms.count_local_2 import count_modp__by_gauss_sum

sage: count_modp__by_gauss_sum(3, 3, 0, 1)    # for Q = x^2 + y^2 + z^2  => Gram␣
↪Det = 1 (mod 3)
9
sage: count_modp__by_gauss_sum(3, 3, 1, 1)    # for Q = x^2 + y^2 + z^2  => Gram␣
↪Det = 1 (mod 3)
6
sage: count_modp__by_gauss_sum(3, 3, 2, 1)    # for Q = x^2 + y^2 + z^2  => Gram␣
↪Det = 1 (mod 3)
12

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: [Q.count_congruence_solutions(3, 1, m, None, None)
....:     == count_modp__by_gauss_sum(3, 3, m, 1)
....:  for m in range(3)]
[True, True, True]


sage: count_modp__by_gauss_sum(3, 3, 0, 2)    # for Q = x^2 + y^2 + 2*z^2  => Gram␣
↪Det = 2 (mod 3)
9
sage: count_modp__by_gauss_sum(3, 3, 1, 2)    # for Q = x^2 + y^2 + 2*z^2  => Gram␣
↪Det = 2 (mod 3)
12
sage: count_modp__by_gauss_sum(3, 3, 2, 2)    # for Q = x^2 + y^2 + 2*z^2  => Gram␣
↪Det = 2 (mod 3)
6

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,2])
sage: [Q.count_congruence_solutions(3, 1, m, None, None)
....:     == count_modp__by_gauss_sum(3, 3, m, 2)
....:  for m in range(3)]
[True, True, True]
```

CHAPTER

SEVEN

EXTRA FUNCTIONS FOR QUADRATIC FORMS

sage.quadratic_forms.extras.**extend_to_primitive**(*A_input*)

Given a matrix (resp. list of vectors), extend it to a square matrix (resp. list of vectors), such that its determinant is the gcd of its minors (i.e. extend the basis of a lattice to a "maximal" one in $\mathbf{Z}^n$).

Author(s): Gonzalo Tornaria and Jonathan Hanke.

INPUT:

a matrix, or a list of length n vectors (in the same space)

OUTPUT:

a square matrix, or a list of n vectors (resp.)

EXAMPLES:

```
sage: A = Matrix(ZZ, 3, 2, range(6))
sage: extend_to_primitive(A)
[ 0  1 -1]
[ 2  3  0]
[ 4  5  0]

sage: extend_to_primitive([vector([1,2,3])])
[(1, 2, 3), (0, 1, 1), (-1, 0, 0)]
```

sage.quadratic_forms.extras.**is_triangular_number**(*n*, *return_value=False*)

Return whether `n` is a triangular number.

A *triangular number* is a number of the form $k(k+1)/2$ for some non-negative integer n. See Wikipedia article Triangular_number. The sequence of triangular number is references as A000217 in the Online encyclopedia of integer sequences (OEIS).

If you want to get the value of k for which $n = k(k+1)/2$ set the argument `return_value` to `True` (see the examples below).

INPUT:

- `n` - an integer
- `return_value` - a boolean set to `False` by default. If set to `True` the function returns a pair made of a boolean and the value v such that $v(v+1)/2 = n$.

EXAMPLES:

```
sage: is_triangular_number(3)
True
```

(continues on next page)

(continued from previous page)

```
sage: is_triangular_number(3, return_value=True)
(True, 2)

sage: is_triangular_number(2)
False
sage: is_triangular_number(2, return_value=True)
(False, None)

sage: is_triangular_number(25*(25+1)/2)
True

sage: is_triangular_number(10^6 * (10^6 +1)/2, return_value=True)
(True, 1000000)
```

sage.quadratic_forms.extras.**least_quadratic_nonresidue**(p)

Return the smallest positive integer quadratic non-residue in $\mathbf{Z}/p\mathbf{Z}$ for primes $p > 2$.

EXAMPLES:

```
sage: least_quadratic_nonresidue(5)
2
sage: [least_quadratic_nonresidue(p) for p in prime_range(3, 100)]             #␣
↪optional - sage.libs.pari
[2, 2, 3, 2, 2, 3, 2, 5, 2, 3, 2, 3, 2, 5, 2, 2, 2, 2, 7, 5, 3, 2, 3, 5]
```

CHAPTER

EIGHT

GENUS

AUTHORS:

- David Kohel & Gabriele Nebe (2007): First created
- Simon Brandhorst (2018): various bugfixes and printing
- Simon Brandhorst (2018): enumeration of genera
- Simon Brandhorst (2020): genus representative

`sage.quadratic_forms.genera.genus.`**`Genus`**(*A*, *factored_determinant=None*)

Given a nonsingular symmetric matrix A, return the genus of A.

INPUT:

- `A` – a symmetric matrix with integer coefficients
- `factored_determinant` – (default: `None`) a `Factorization` object, the factored determinant of `A`

OUTPUT:

A *`GenusSymbol_global_ring`* object, encoding the Conway-Sloane genus symbol of the quadratic form whose Gram matrix is A.

EXAMPLES:

```
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: Genus(A)
Genus of
[1 1]
[1 2]
Signature:  (2, 0)
Genus symbol at 2:    [1^2]_2

sage: A = Matrix(ZZ, 2, 2, [2, 1, 1, 2])
sage: Genus(A, A.det().factor())
Genus of
[2 1]
[1 2]
Signature:  (2, 0)
Genus symbol at 2:    1^-2
Genus symbol at 3:     1^-1 3^-1
```

class `sage.quadratic_forms.genera.genus.`**`GenusSymbol_global_ring`**(*signature_pair*, *local_symbols*, *representative=None*, *check=True*)

Bases: `object`

This represents a collection of local genus symbols (at primes) and signature information which represent the genus of a non-degenerate integral lattice.

INPUT:

- `signature_pair` – a tuple of two non-negative integers
- `local_symbols` – a list of `Genus_Symbol_p_adic_ring` instances sorted by their primes
- `representative` – (default: `None`) integer symmetric matrix; the Gram matrix of a representative of this genus
- `check` – (default: `True`) a boolean; checks the input

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import GenusSymbol_global_ring,␣
↪LocalGenusSymbol
sage: A = matrix.diagonal(ZZ, [2, 4, 6, 8])
sage: local_symbols = [LocalGenusSymbol(A, p) for p in (2*A.det()).prime_divisors()]
sage: G = GenusSymbol_global_ring((4, 0),local_symbols, representative=A);G
Genus of
[2 0 0 0]
[0 4 0 0]
[0 0 6 0]
[0 0 0 8]
Signature:  (4, 0)
Genus symbol at 2:    [2^-2 4^1 8^1]_6
Genus symbol at 3:     1^3 3^-1
```

See also:

`Genus()` to create a `GenusSymbol_global_ring` from the Gram matrix directly.

det()

Return the determinant of this genus, where the determinant is the Hessian determinant of the quadratic form whose Gram matrix is the Gram matrix giving rise to this global genus symbol.

OUTPUT: an integer

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [1, -2, 3, 4])
sage: GS = Genus(A)
sage: GS.determinant()
-24
```

determinant()

Return the determinant of this genus, where the determinant is the Hessian determinant of the quadratic form whose Gram matrix is the Gram matrix giving rise to this global genus symbol.

OUTPUT: an integer

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [1, -2, 3, 4])
sage: GS = Genus(A)
sage: GS.determinant()
-24
```

dim()

Return the dimension of this genus.

EXAMPLES:

```
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: G = Genus(A)
sage: G.dimension()
2
```

dimension()

Return the dimension of this genus.

EXAMPLES:

```
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: G = Genus(A)
sage: G.dimension()
2
```

direct_sum(*other*)

Return the genus of the direct sum of `self` and `other`.

The direct sum is defined as the direct sum of representatives.

EXAMPLES:

```
sage: G = IntegralLattice("A4").twist(3).genus()
sage: G.direct_sum(G)
Genus of
None
Signature:  (8, 0)
Genus symbol at 2:    1^8
Genus symbol at 3:     3^8
Genus symbol at 5:     1^6 5^2
```

discriminant_form()

Return the discriminant form associated to this genus.

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [2, -4, 6, 8])
sage: GS = Genus(A)
sage: GS.discriminant_form()
Finite quadratic module over Integer Ring with invariants (2, 2, 4, 24)
Gram matrix of the quadratic form with values in Q/2Z:
[  1/2     0   1/2     0]
[    0   3/2     0     0]
[  1/2     0   3/4     0]
[    0     0     0 25/24]
```

(continues on next page)

(continued from previous page)

```
sage: A = matrix.diagonal(ZZ, [1, -4, 6, 8])
sage: GS = Genus(A)
sage: GS.discriminant_form()
Finite quadratic module over Integer Ring with invariants (2, 4, 24)
Gram matrix of the quadratic form with values in Q/Z:
[ 1/2  1/2    0]
[ 1/2  3/4    0]
[   0    0 1/24]
```

is_even()

Return if this genus is even.

EXAMPLES:

```
sage: G = Genus(Matrix(ZZ,2,[2,1,1,2]))
sage: G.is_even()
True
```

level()

Return the level of this genus.

This is the denominator of the inverse Gram matrix of a representative.

EXAMPLES:

```
sage: G = Genus(matrix.diagonal([2, 4, 18]))
sage: G.level()
36
```

local_symbol(*p*)

Return a copy of the local symbol at the prime p.

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [2, -4, 6, 8])
sage: GS = Genus(A)
sage: GS.local_symbol(3)
Genus symbol at 3:     1^-3 3^-1
```

local_symbols()

Return a copy of the list of local symbols of this symbol.

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [2, -4, 6, 8])
sage: GS = Genus(A)
sage: GS.local_symbols()
[Genus symbol at 2:    [2^-2 4^1 8^1]_4,
 Genus symbol at 3:     1^-3 3^-1]
```

mass(*backend='sage'*)

Return the mass of this genus.

The genus must be definite. Let $L_1, ...L_n$ be a complete list of representatives of the isometry classes in this genus. Its mass is defined as

$$\sum_{i=1}^{n} \frac{1}{|O(L_i)|}.$$

INPUT:

- `backend` – default: `'sage'`, or `'magma'`

OUTPUT: a rational number

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import genera
sage: G = genera((8,0), 1, even=True)[0]
sage: G.mass()                                                              #␣
↪optional - sage.symbolic
1/696729600
sage: G.mass(backend='magma')  # optional - magma
1/696729600
```

The E_8 lattice is unique in its genus:

```
sage: E8 = QuadraticForm(G.representative())
sage: E8.number_of_automorphisms()
696729600
```

norm()

Return the norm of this genus.

Let L be a lattice with bilinear form b. The scale of (L, b) is defined as the ideal generated by $\{b(x, x)|x \in L\}$.

EXAMPLES:

```
sage: G = Genus(matrix.diagonal([6, 4, 18]))
sage: G.norm()
2
sage: G = Genus(matrix(ZZ, 2, [0, 1, 1, 0]))
sage: G.norm()
2
```

rank()

Return the dimension of this genus.

EXAMPLES:

```
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: G = Genus(A)
sage: G.dimension()
2
```

rational_representative()

Return a representative of the rational bilinear form defined by this genus.

OUTPUT: a diagonal_matrix

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import genera
sage: G = genera((8,0), 1)[0]
sage: G
Genus of
None
Signature:  (8, 0)
Genus symbol at 2:    1^8
sage: G.rational_representative()
[1 0 0 0 0 0 0 0]
[0 1 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0]
[0 0 0 1 0 0 0 0]
[0 0 0 0 1 0 0 0]
[0 0 0 0 0 2 0 0]
[0 0 0 0 0 0 1 0]
[0 0 0 0 0 0 0 2]
```

representative()

Return a representative in this genus.

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import genera
sage: g = genera([1,3], 24)[0]
sage: g
Genus of
None
Signature:  (1, 3)
Genus symbol at 2:    [1^-1 2^3]_0
Genus symbol at 3:     1^3 3^1
```

A representative of g is not known yet. Let us trigger its computation:

```
sage: g.representative()
[ 0  0  0  2]
[ 0 -1  0  0]
[ 0  0 -6  0]
[ 2  0  0  0]
sage: g == Genus(g.representative())
True
```

representatives(*backend=None*, *algorithm=None*)

Return a list of representatives for the classes in this genus

INPUT:

- backend – (default: None)
- algorithm – (default: None)

OUTPUT: a list of Gram matrices

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import genera
sage: G = Genus(matrix.diagonal([1, 1, 7]))
```

(continues on next page)

(continued from previous page)

```
sage: G.representatives()
(
[1 0 0]  [1 0 0]
[0 2 1]  [0 1 0]
[0 1 4], [0 0 7]
)
```

Indefinite genera work as well:

```
sage: G = Genus(matrix(ZZ, 3, [6,3,0, 3,6,0, 0,0,2]))
sage: G.representatives()
(
[2 0 0]  [ 2  1  0]
[0 6 3]  [ 1  2  0]
[0 3 6], [ 0  0 18]
)
```

For positive definite forms the magma backend is available:

```
sage: G = Genus(matrix.diagonal([1, 1, 7]))
sage: G.representatives(backend="magma")  # optional - magma
(
[1 0 0]  [ 1  0  0]
[0 1 0]  [ 0  2 -1]
[0 0 7], [ 0 -1  4]
)
```

scale()

Return the scale of this genus.

Let L be a lattice with bilinear form b. The scale of (L, b) is defined as the ideal $b(L, L)$.

OUTPUT: an integer

EXAMPLES:

```
sage: G = Genus(matrix.diagonal([2, 4, 18]))
sage: G.scale()
2
```

signature()

Return the signature of this genus.

The signature is $p - n$ where p is the number of positive eigenvalues and n the number of negative eigenvalues.

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [1, -2, 3, 4, 8, -11])
sage: GS = Genus(A)
sage: GS.signature()
2
```

signature_pair()

Return the signature pair (p, n) of the (non-degenerate) global genus symbol, where p is the number of positive eigenvalues and n is the number of negative eigenvalues.

OUTPUT: a pair of integers (p, n), each ≥ 0

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [1, -2, 3, 4, 8, -11])
sage: GS = Genus(A)
sage: GS.signature_pair()
(4, 2)
```

signature_pair_of_matrix()

Return the signature pair (p, n) of the (non-degenerate) global genus symbol, where p is the number of positive eigenvalues and n is the number of negative eigenvalues.

OUTPUT: a pair of integers (p, n), each ≥ 0

EXAMPLES:

```
sage: A = matrix.diagonal(ZZ, [1, -2, 3, 4, 8, -11])
sage: GS = Genus(A)
sage: GS.signature_pair()
(4, 2)
```

spinor_generators(*proper*)

Return the spinor generators.

INPUT:

- `proper` – boolean

OUTPUT: a list of primes not dividing the determinant

EXAMPLES:

```
sage: g = matrix(ZZ, 3, [2,1,0, 1,2,0, 0,0,18])
sage: gen = Genus(g)
sage: gen.spinor_generators(False)
[5]
```

class sage.quadratic_forms.genera.genus.**Genus_Symbol_p_adic_ring**(*prime*, *symbol*, *check=True*)

Bases: `object`

Local genus symbol over a p-adic ring.

The genus symbol of a component $p^m A$ for odd prime $= p$ is of the form (m, n, d), where

- m = valuation of the component
- n = rank of A
- $d = det(A) \in \{1, u\}$ for a normalized quadratic non-residue u.

The genus symbol of a component $2^m A$ is of the form (m, n, s, d, o), where

- m = valuation of the component
- n = rank of A
- d = det(A) in $\{1, 3, 5, 7\}$
- s = 0 (or 1) if even (or odd)
- o = oddity of A (= 0 if s = 0) in $Z/8Z$ = the trace of the diagonalization of A

The genus symbol is a list of such symbols (ordered by m) for each of the Jordan blocks $A_1, ..., A_t$.

REFERENCE:

[CS1999] Conway and Sloane 3rd edition, Chapter 15, Section 7.

> **Warning:** This normalization seems non-standard, and we should review this entire class to make sure that we have our doubling conventions straight throughout! This is especially noticeable in the determinant and excess methods!!

INPUT:

- `prime` – a prime number
- `symbol` – the list of invariants for Jordan blocks $A_t, ..., A_t$ given as a list of lists of integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = diagonal_matrix(ZZ, [1, 2, 3, 4])
sage: p = 2
sage: s2 = p_adic_symbol(A, p, 2); s2
[[0, 2, 3, 1, 4], [1, 1, 1, 1, 1], [2, 1, 1, 1, 1]]
sage: G2 = Genus_Symbol_p_adic_ring(p,s2); G2
Genus symbol at 2:    [1^-2 2^1 4^1]_6

sage: A = diagonal_matrix(ZZ, [1, 2, 3, 4])
sage: p = 3
sage: s3 = p_adic_symbol(A, p, 1); s3
[[0, 3, -1], [1, 1, 1]]
sage: G3 = Genus_Symbol_p_adic_ring(p,s3); G3
Genus symbol at 3:     1^-3 3^1
```

automorphous_numbers()

Return generators of the automorphous square classes at this prime.

A p-adic square class r is called automorphous if it is the spinor norm of a proper p-adic integral automorphism of this form. These classes form a group. See [CS1999] Chapter 15, 9.6 for details.

OUTPUT:

a list of integers representing the square classes of generators of the automorphous numbers

EXAMPLES:

The following examples are given in [CS1999] 3rd edition, Chapter 15, 9.6 pp. 392:

```
sage: A = matrix.diagonal([3, 16])
sage: G = Genus(A)
sage: sym2 = G.local_symbols()[0]; sym2
Genus symbol at 2:    [1^-1]_3:[16^1]_1
sage: sym2.automorphous_numbers()
[3, 5]

sage: A = matrix(ZZ, 3, [2,1,0, 1,2,0, 0,0,18])
```

(continues on next page)

(continued from previous page)

```
sage: G = Genus(A)
sage: sym = G.local_symbols()
sage: sym[0]
Genus symbol at 2:    1^-2 [2^1]_1
sage: sym[0].automorphous_numbers()
[1, 3, 5, 7]
sage: sym[1]
Genus symbol at 3:     1^-1 3^-1 9^-1
sage: sym[1].automorphous_numbers()
[1, 3]
```

Note that the generating set given is not minimal. The first supplementation rule is used here:

```
sage: A = matrix.diagonal([2, 2, 4])
sage: G = Genus(A)
sage: sym = G.local_symbols()
sage: sym[0]
Genus symbol at 2:    [2^2 4^1]_3
sage: sym[0].automorphous_numbers()
[1, 2, 3, 5, 7]
```

but not there:

```
sage: A = matrix.diagonal([2, 2, 32])
sage: G = Genus(A)
sage: sym = G.local_symbols()
sage: sym[0]
Genus symbol at 2:    [2^2]_2:[32^1]_1
sage: sym[0].automorphous_numbers()
[1, 2, 5]
```

Here the second supplementation rule is used:

```
sage: A = matrix.diagonal([2, 2, 64])
sage: G = Genus(A)
sage: sym = G.local_symbols()
sage: sym[0]
Genus symbol at 2:    [2^2]_2:[64^1]_1
sage: sym[0].automorphous_numbers()
[1, 2, 5]
```

canonical_symbol()

Return (and cache) the canonical p-adic genus symbol. This is only really affects the 2-adic symbol, since when $p > 2$ the symbol is already canonical.

OUTPUT:

a list of lists of integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring
```

(continues on next page)

(continued from previous page)

```
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2.symbol_tuple_
↪list()
[[0, 2, 1, 1, 2]]
sage: G2.canonical_symbol()
[[0, 2, 1, 1, 2]]

sage: A = Matrix(ZZ, 2, 2, [1, 0, 0, 2])
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2.symbol_tuple_
↪list()
[[0, 1, 1, 1, 1], [1, 1, 1, 1, 1]]
sage: G2.canonical_symbol()   # Oddity fusion occurred here!
[[0, 1, 1, 1, 2], [1, 1, 1, 1, 0]]

sage: A = DiagonalQuadraticForm(ZZ, [1,2,3,4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2.symbol_tuple_
↪list()
[[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]
sage: G2.canonical_symbol()   # Oddity fusion occurred here!
[[1, 2, -1, 1, 6], [2, 1, 1, 1, 0], [3, 1, 1, 1, 0]]

sage: A = Matrix(ZZ, 2, 2, [2, 1, 1, 2])
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2.symbol_tuple_
↪list()
[[0, 2, 3, 0, 0]]
sage: G2.canonical_symbol()
[[0, 2, -1, 0, 0]]


sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G3.symbol_tuple_
↪list()
[[0, 3, 1], [1, 1, -1]]
sage: G3.canonical_symbol()
[[0, 3, 1], [1, 1, -1]]
```

Note: See [CS1999] Conway-Sloane 3rd edition, pp. 381-382 for definitions and examples.

Todo: Add an example where sign walking occurs!

compartments()

Compute the indices for each of the compartments in this local genus symbol if it is associated to the prime $p = 2$ (and raise an error for all other primes).

OUTPUT: a list of non-negative integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.compartments()
[[0, 1, 2]]
```

det()

Returns the (p-part of the) determinant (square-class) of the Hessian matrix of the quadratic form (given by regarding the integral symmetric matrix which generated this genus symbol as the Gram matrix of Q) associated to this local genus symbol.

OUTPUT: an integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.determinant()
128

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G3
Genus symbol at 3:     1^3 3^-1
sage: G3.determinant()
3
```

determinant()

Returns the (p-part of the) determinant (square-class) of the Hessian matrix of the quadratic form (given by regarding the integral symmetric matrix which generated this genus symbol as the Gram matrix of Q) associated to this local genus symbol.

OUTPUT: an integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.determinant()
```

(continues on next page)

(continued from previous page)

```
128

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G3
Genus symbol at 3:     1^3 3^-1
sage: G3.determinant()
3
```

dim()

Return the dimension of a quadratic form associated to this genus symbol.

OUTPUT: a non-negative integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.dimension()
4

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G3
Genus symbol at 3:     1^3 3^-1
sage: G3.dimension()
4
```

dimension()

Return the dimension of a quadratic form associated to this genus symbol.

OUTPUT: a non-negative integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.dimension()
4

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G3
Genus symbol at 3:     1^3 3^-1
```

(continues on next page)

(continued from previous page)

```
sage: G3.dimension()
4
```

direct_sum(*other*)

Return the local genus of the direct sum of two representatives.

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring
sage: A = matrix.diagonal([1, 2, 3, 4])
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [1^-2 2^1 4^1]_6
sage: G2.direct_sum(G2)
Genus symbol at 2:    [1^4 2^2 4^2]_4
```

excess()

Returns the p-excess of the quadratic form whose Hessian matrix is the symmetric matrix A. When $p = 2$, the p-excess is called the oddity.

> **Warning:** This normalization seems non-standard, and we should review this entire class to make sure that we have our doubling conventions straight throughout!

REFERENCE:

[CS1999] Conway and Sloane Book, 3rd edition, pp 370-371.

OUTPUT: an integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: AC = diagonal_matrix(ZZ, [1, 3, -3])
sage: p=2; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
1
sage: p=3; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0
sage: p=5; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0
sage: p=7; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0
sage: p=11; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0

sage: AC = 2 * diagonal_matrix(ZZ, [1, 3, -3])
sage: p=2; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
1
sage: p=3; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0
```

(continues on next page)

(continued from previous page)

```
sage: p=5; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0
sage: p=7; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0
sage: p=11; Genus_Symbol_p_adic_ring(p, p_adic_symbol(AC, p, 2)).excess()
0

sage: A = 2*diagonal_matrix(ZZ, [1, 2, 3, 4])
sage: p = 2; Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)).excess()
2
sage: p = 3; Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)).excess()
6
sage: p = 5; Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)).excess()
0
sage: p = 7; Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)).excess()
0
sage: p = 11; Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)).excess()
0
```

gram_matrix(*check=True*)

Return a Gram matrix of a representative of this local genus.

INPUT:

- `check` (default: `True`) – double check the result

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring
sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2))
sage: G2.gram_matrix()
[2 0|0|0]
[0 6|0|0]
[---+-+-]
[0 0|4|0]
[---+-+-]
[0 0|0|8]
```

is_even()

Return if the underlying p-adic lattice is even.

If p is odd, every lattice is even.

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import LocalGenusSymbol
sage: M0 = matrix(ZZ, [1])
sage: G0 = LocalGenusSymbol(M0, 2)
sage: G0.is_even()
False
sage: G1 = LocalGenusSymbol(M0, 3)
```

(continues on next page)

(continued from previous page)

```
sage: G1.is_even()
True
sage: M2 = matrix(ZZ, [2])
sage: G2 = LocalGenusSymbol(M2, 2)
sage: G2.is_even()
True
```

level()

Return the maximal scale of a Jordan component.

EXAMPLES:

```
sage: G = Genus(matrix.diagonal([2, 4, 18]))
sage: G.local_symbol(2).level()
4
```

mass()

Return the local mass m_p of this genus as defined by Conway.

See Equation (3) in [CS1988].

EXAMPLES:

```
sage: G = Genus(matrix.diagonal([1, 3, 9]))
sage: G.local_symbol(3).mass()
9/8
```

norm()

Return the norm of this local genus.

Let L be a lattice with bilinear form b. The norm of (L, b) is defined as the ideal generated by $\{b(x, x)|x \in L\}$.

EXAMPLES:

```
sage: G = Genus(matrix.diagonal([2, 4, 18]))
sage: G.local_symbol(2).norm()
2
sage: G = Genus(matrix(ZZ,2,[0, 1, 1, 0]))
sage: G.local_symbol(2).norm()
2
```

number_of_blocks()

Return the number of positive dimensional symbols/Jordan blocks.

OUTPUT: a non-negative integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2))
```

(continues on next page)

(continued from previous page)

```
sage: G2.symbol_tuple_list()
[[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]
sage: G2.number_of_blocks()
3

sage: A = DiagonalQuadraticForm(ZZ, [1,2,3,4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2))
sage: G3.symbol_tuple_list()
[[0, 3, 1], [1, 1, -1]]
sage: G3.number_of_blocks()
2
```

prime()

Return the prime number p of this p-adic local symbol.

OUTPUT: an integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import LocalGenusSymbol
sage: M1 = matrix(ZZ, [2])
sage: p = 2
sage: G0 = LocalGenusSymbol(M1, 2)
sage: G0.prime()
2
```

rank()

Return the dimension of a quadratic form associated to this genus symbol.

OUTPUT: a non-negative integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.dimension()
4

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G3
Genus symbol at 3:     1^3 3^-1
sage: G3.dimension()
4
```

scale()

Return the scale of this local genus.

Let L be a lattice with bilinear form b. The scale of (L, b) is defined as the ideal $b(L, L)$.

OUTPUT: an integer

EXAMPLES:

```
sage: G = Genus(matrix.diagonal([2, 4, 18]))
sage: G.local_symbol(2).scale()
2
sage: G.local_symbol(3).scale()
1
```

symbol_tuple_list()

Return a copy of the underlying list of lists of integers defining the genus symbol.

OUTPUT: a list of lists of integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 3
sage: G3 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G3
Genus symbol at 3:     1^3 3^-1
sage: G3.symbol_tuple_list()
[[0, 3, 1], [1, 1, -1]]
sage: type(G3.symbol_tuple_list())
<... 'list'>

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.symbol_tuple_list()
[[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]
sage: type(G2.symbol_tuple_list())
<... 'list'>
```

trains()

Compute the indices for each of the trains in this local genus symbol if it is associated to the prime $p = 2$ (and raise an error for all other primes).

OUTPUT: a list of non-negative integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol
sage: from sage.quadratic_forms.genera.genus import Genus_Symbol_p_adic_ring

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p = 2
sage: G2 = Genus_Symbol_p_adic_ring(p, p_adic_symbol(A, p, 2)); G2
Genus symbol at 2:    [2^-2 4^1 8^1]_6
sage: G2.trains()
[[0, 1, 2]]
```

sage.quadratic_forms.genera.genus.**LocalGenusSymbol**(*A*, *p*)

Return the local symbol of A at the prime p.

INPUT:

- A – a symmetric, non-singular matrix with coefficients in $\mathbf{Z}$
- p – a prime number

OUTPUT:

A *Genus_Symbol_p_adic_ring* object, encoding the Conway-Sloane genus symbol at p of the quadratic form whose Gram matrix is A.

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import LocalGenusSymbol
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: LocalGenusSymbol(A, 2)
Genus symbol at 2:    [1^2]_2
sage: LocalGenusSymbol(A, 3)
Genus symbol at 3:     1^2

sage: A = Matrix(ZZ, 2, 2, [1, 0, 0, 2])
sage: LocalGenusSymbol(A, 2)
Genus symbol at 2:    [1^1 2^1]_2
sage: LocalGenusSymbol(A, 3)
Genus symbol at 3:     1^-2
```

sage.quadratic_forms.genera.genus.**M_p**(*species*, *p*)

Return the diagonal factor M_p as a function of the species.

EXAMPLES:

These examples are taken from Table 2 of [CS1988]:

```
sage: from sage.quadratic_forms.genera.genus import M_p
sage: M_p(0, 2)
1
sage: M_p(1, 2)
1/2
sage: M_p(-2, 2)
1/3
sage: M_p(2, 2)
1
sage: M_p(3, 2)
2/3
sage: M_p(-4, 2)
8/15
sage: M_p(4, 2)
8/9
sage: M_p(5, 2)
32/45
```

sage.quadratic_forms.genera.genus.**basis_complement**(*B*)

Given an echelonized basis matrix B (over a field), calculate a matrix whose rows form a basis complement (to the rows of B).

INPUT:

- B – matrix over a field in row echelon form

OUTPUT: a rectangular matrix over a field

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import basis_complement

sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 1])
sage: B = A.kernel().echelonized_basis_matrix(); B
[ 1 -1]
sage: basis_complement(B)
[0 1]
```

sage.quadratic_forms.genera.genus.**canonical_2_adic_compartments**(*genus_symbol_quintuple_list*)

Given a 2-adic local symbol (as the underlying list of quintuples) this returns a list of lists of indices of the `genus_symbol_quintuple_list` which are in the same compartment. A compartment is defined to be a maximal interval of Jordan components all (scaled) of type I (i.e. odd).

INPUT:

- `genus_symbol_quintuple_list` – a quintuple of integers (with certain restrictions)

OUTPUT: a list of lists of integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import LocalGenusSymbol
sage: from sage.quadratic_forms.genera.genus import canonical_2_adic_compartments

sage: A = Matrix(ZZ, 2, 2, [1,1,1,2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 2, 1, 1, 2]]
sage: canonical_2_adic_compartments(G2.symbol_tuple_list())
[[0]]

sage: A = Matrix(ZZ, 2, 2, [1,0,0,2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 1, 1, 1, 1], [1, 1, 1, 1, 1]]
sage: canonical_2_adic_compartments(G2.symbol_tuple_list())
[[0, 1]]

sage: A = DiagonalQuadraticForm(ZZ, [1,2,3,4]).Hessian_matrix()
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]
sage: canonical_2_adic_compartments(G2.symbol_tuple_list())
[[0, 1, 2]]

sage: A = Matrix(ZZ, 2, 2, [2,1,1,2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 2, 3, 0, 0]]
sage: canonical_2_adic_compartments(G2.symbol_tuple_list())   # No compartments␣
↪here!
[]
```

Note: See [CS1999] Conway-Sloane 3rd edition, pp. 381-382 for definitions and examples.

`sage.quadratic_forms.genera.genus.canonical_2_adic_reduction`(*genus_symbol_quintuple_list*)

Given a 2-adic local symbol (as the underlying list of quintuples) this returns a canonical 2-adic symbol (again as a raw list of quintuples of integers) which has at most one minus sign per train and this sign appears on the smallest dimensional Jordan component in each train. This results from applying the "sign-walking" and "oddity fusion" equivalences.

INPUT:

- `genus_symbol_quintuple_list` – a quintuple of integers (with certain restrictions)
- `compartments` – a list of lists of distinct integers (optional)

OUTPUT: a list of lists of distinct integers.

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import LocalGenusSymbol
sage: from sage.quadratic_forms.genera.genus import canonical_2_adic_reduction

sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 2, 1, 1, 2]]
sage: canonical_2_adic_reduction(G2.symbol_tuple_list())
[[0, 2, 1, 1, 2]]

sage: A = Matrix(ZZ, 2, 2, [1, 0, 0, 2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 1, 1, 1, 1], [1, 1, 1, 1, 1]]
sage: canonical_2_adic_reduction(G2.symbol_tuple_list())   # Oddity fusion occurred␣
↪here!
[[0, 1, 1, 1, 2], [1, 1, 1, 1, 0]]

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]
sage: canonical_2_adic_reduction(G2.symbol_tuple_list())   # Oddity fusion occurred␣
↪here!
[[1, 2, -1, 1, 6], [2, 1, 1, 1, 0], [3, 1, 1, 1, 0]]

sage: A = Matrix(ZZ, 2, 2, [2, 1, 1, 2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 2, 3, 0, 0]]
sage: canonical_2_adic_reduction(G2.symbol_tuple_list())
[[0, 2, -1, 0, 0]]
```

Note: See [CS1999] Conway-Sloane 3rd edition, pp. 381-382 for definitions and examples.

Todo: Add an example where sign walking occurs!

sage.quadratic_forms.genera.genus.**canonical_2_adic_trains**(*genus_symbol_quintuple_list*, *compartments=None*)

Given a 2-adic local symbol (as the underlying list of quintuples) this returns a list of lists of indices of the `genus_symbol_quintuple_list` which are in the same train. A train is defined to be a maximal interval of Jordan components so that at least one of each adjacent pair (allowing zero-dimensional Jordan components) is (scaled) of type I (i.e. odd). Note that an interval of length one respects this condition as there is no pair in this interval. In particular, every Jordan component is part of a train.

INPUT:

- `genus_symbol_quintuple_list` – a quintuple of integers (with certain restrictions).
- `compartments` – this argument is deprecated

OUTPUT: a list of lists of distinct integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import LocalGenusSymbol
sage: from sage.quadratic_forms.genera.genus import canonical_2_adic_compartments
sage: from sage.quadratic_forms.genera.genus import canonical_2_adic_trains

sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 2, 1, 1, 2]]
sage: canonical_2_adic_trains(G2.symbol_tuple_list())
[[0]]

sage: A = Matrix(ZZ, 2, 2, [1,0,0,2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 1, 1, 1, 1], [1, 1, 1, 1, 1]]
sage: canonical_2_adic_compartments(G2.symbol_tuple_list())
[[0, 1]]

sage: A = DiagonalQuadraticForm(ZZ, [1,2,3,4]).Hessian_matrix()
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]
sage: canonical_2_adic_trains(G2.symbol_tuple_list())
[[0, 1, 2]]

sage: A = Matrix(ZZ, 2, 2, [2, 1, 1, 2])
sage: G2 = LocalGenusSymbol(A, 2); G2.symbol_tuple_list()
[[0, 2, 3, 0, 0]]
sage: canonical_2_adic_trains(G2.symbol_tuple_list())
[[0]]
sage: symbol = [[0, 1,  1, 1, 1], [1, 2, -1, 0, 0], [2, 1,  1, 1, 1],
....:           [3, 1,  1, 1, 1], [4, 1,  1, 1, 1], [5, 2, -1, 0, 0],
....:           [7, 1,  1, 1, 1], [10, 1, 1, 1, 1], [11, 1, 1, 1, 1], [12, 1, 1, 1,␣
↪1]]
sage: canonical_2_adic_trains(symbol)
[[0, 1, 2, 3, 4, 5], [6], [7, 8, 9]]
```

Check that github issue #24818 is fixed:

```
sage: symbol = [[0, 1,  1, 1, 1], [1, 3, 1, 1, 1]]
sage: canonical_2_adic_trains(symbol)
```

(continues on next page)

(continued from previous page)

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

Note: See [CS1999], pp. 381-382 for definitions and examples.

sage.quadratic_forms.genera.genus.**genera**(*sig_pair*, *determinant*, *max_scale=None*, *even=False*)

Return a list of all global genera with the given conditions.

Here a genus is called global if it is non-empty.

INPUT:

- `sig_pair` – a pair of non-negative integers giving the signature
- `determinant` – an integer; the sign is ignored
- `max_scale` – (default: `None`) an integer; the maximum scale of a jordan block
- `even` – boolean (default: `False`)

OUTPUT:

A list of all (non-empty) global genera with the given conditions.

EXAMPLES:

```
sage: QuadraticForm.genera((4,0), 125, even=True)
[Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^1 5^3, Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^-2 5^1 25^-1, Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^2 5^1 25^1, Genus of
None
Signature:  (4, 0)
Genus symbol at 2:    1^-4
Genus symbol at 5:     1^3 125^1]
```

sage.quadratic_forms.genera.genus.**is_2_adic_genus**(*genus_symbol_quintuple_list*)

Given a 2-adic local symbol (as the underlying list of quintuples) check whether it is the 2-adic symbol of a 2-adic form.

INPUT:

- `genus_symbol_quintuple_list` – a quintuple of integers (with certain restrictions).

OUTPUT: boolean

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import LocalGenusSymbol, is_2_adic_
↪genus

sage: A = Matrix(ZZ, 2, 2, [1,1,1,2])
sage: G2 = LocalGenusSymbol(A, 2)
sage: is_2_adic_genus(G2.symbol_tuple_list())
True

sage: A = Matrix(ZZ, 2, 2, [1,1,1,2])
sage: G3 = LocalGenusSymbol(A, 3)
sage: is_2_adic_genus(G3.symbol_tuple_list())  # This raises an error
Traceback (most recent call last):
...
TypeError: The genus symbols are not quintuples,
so it's not a genus symbol for the prime p=2.

sage: A = Matrix(ZZ, 2, 2, [1,0,0,2])
sage: G2 = LocalGenusSymbol(A, 2)
sage: is_2_adic_genus(G2.symbol_tuple_list())
True
```

sage.quadratic_forms.genera.genus.**is_GlobalGenus**(G)

Return if G represents the genus of a global quadratic form or lattice.

INPUT:

- G – *GenusSymbol_global_ring* object

OUTPUT: boolean

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import is_GlobalGenus
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 2])
sage: G = Genus(A)
sage: is_GlobalGenus(G)
True
sage: G = Genus(matrix.diagonal([2, 2, 2, 2]))
sage: G._local_symbols[0]._symbol = [[0,2,3,0,0], [1,2,5,1,0]]
sage: G._representative=None
sage: is_GlobalGenus(G)
False
```

sage.quadratic_forms.genera.genus.**is_even_matrix**(A)

Determines if the integral symmetric matrix A is even (i.e. represents only even numbers). If not, then it returns the index of an odd diagonal entry. If it is even, then we return the index -1.

INPUT:

- A – symmetric integer matrix

OUTPUT: a pair of the form (boolean, integer)

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import is_even_matrix
```

(continues on next page)

(continued from previous page)

```
sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 1])
sage: is_even_matrix(A)
(False, 0)

sage: A = Matrix(ZZ, 2, 2, [2, 1, 1, 2])
sage: is_even_matrix(A)
(True, -1)
```

sage.quadratic_forms.genera.genus.**p_adic_symbol**(*A*, *p*, *val*)

Given a symmetric matrix A and prime p, return the genus symbol at p.

Todo: Some description of the definition of the genus symbol.

INPUT:

- A – symmetric matrix with integer coefficients
- p – prime number
- val – non-negative integer; valuation of the maximal elementary divisor of A needed to obtain enough precision. Calculation is modulo p to the val+3.

OUTPUT: a list of lists of integers

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import p_adic_symbol

sage: A = DiagonalQuadraticForm(ZZ, [1, 2, 3, 4]).Hessian_matrix()
sage: p_adic_symbol(A, 2, 2)
[[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]

sage: p_adic_symbol(A, 3, 1)
[[0, 3, 1], [1, 1, -1]]
```

sage.quadratic_forms.genera.genus.**signature_pair_of_matrix**(*A*)

Computes the signature pair (p, n) of a non-degenerate symmetric matrix, where

- p is the number of positive eigenvalues of A
- n is the number of negative eigenvalues of A

INPUT:

- A – symmetric matrix (assumed to be non-degenerate)

OUTPUT: (p, n) – a pair (tuple) of integers.

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import signature_pair_of_matrix

sage: A = Matrix(ZZ, 2, 2, [-1, 0, 0, 3])
sage: signature_pair_of_matrix(A)
(1, 1)
```

(continues on next page)

(continued from previous page)

```
sage: A = Matrix(ZZ, 2, 2, [-1, 1, 1, 7])
sage: signature_pair_of_matrix(A)
(1, 1)

sage: A = Matrix(ZZ, 2, 2, [3, 1, 1, 7])
sage: signature_pair_of_matrix(A)
(2, 0)

sage: A = Matrix(ZZ, 2, 2, [-3, 1, 1, -11])
sage: signature_pair_of_matrix(A)
(0, 2)


sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 1])
sage: signature_pair_of_matrix(A)
Traceback (most recent call last):
...
ArithmeticError: given matrix is not invertible
```

sage.quadratic_forms.genera.genus.**split_odd**(A)

Given a non-degenerate Gram matrix $A(\mod 8)$, return a splitting `[u] + B` such that u is odd and B is not even.

INPUT:

- `A` – an odd symmetric matrix with integer coefficients (which admits a splitting as above).

OUTPUT:

a pair `(u, B)` consisting of an odd integer u and an odd integral symmetric matrix B.

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import is_even_matrix
sage: from sage.quadratic_forms.genera.genus import split_odd

sage: A = Matrix(ZZ, 2, 2, [1, 2, 2, 3])
sage: is_even_matrix(A)
(False, 0)
sage: split_odd(A)
(1, [-1])

sage: A = Matrix(ZZ, 2, 2, [1, 2, 2, 5])
sage: split_odd(A)
(1, [1])

sage: A = Matrix(ZZ, 2, 2, [1, 1, 1, 1])
sage: is_even_matrix(A)
(False, 0)
sage: split_odd(A)      # This fails because no such splitting exists. =(
Traceback (most recent call last):
...
RuntimeError: The matrix A does not admit a non-even splitting.
```

(continues on next page)

(continued from previous page)

```
sage: A = Matrix(ZZ, 2, 2, [1, 2, 2, 6])
sage: split_odd(A)      # This fails because no such splitting exists. =(
Traceback (most recent call last):
...
RuntimeError: The matrix A does not admit a non-even splitting.
```

sage.quadratic_forms.genera.genus.**trace_diag_mod_8**(A)

Return the trace of the diagonalised form of A of an integral symmetric matrix which is diagonalizable mod 8. (Note that since the Jordan decomposition into blocks of size $\leq$ 2 is not unique here, this is not the same as saying that A is always diagonal in any 2-adic Jordan decomposition!)

INPUT:

- `A` – symmetric matrix with coefficients in $\mathbf{Z}$ which is odd in $\mathbf{Z}/2\mathbf{Z}$ and has determinant not divisible by 8.

OUTPUT: an integer

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import is_even_matrix
sage: from sage.quadratic_forms.genera.genus import split_odd
sage: from sage.quadratic_forms.genera.genus import trace_diag_mod_8

sage: A = Matrix(ZZ, 2, 2, [1, 2, 2, 3])
sage: is_even_matrix(A)
(False, 0)
sage: split_odd(A)
(1, [-1])
sage: trace_diag_mod_8(A)
0

sage: A = Matrix(ZZ, 2, 2, [1, 2, 2, 5])
sage: split_odd(A)
(1, [1])
sage: trace_diag_mod_8(A)
2
```

sage.quadratic_forms.genera.genus.**two_adic_symbol**(A, *val*)

Given a symmetric matrix A and prime p, return the genus symbol at p.

The genus symbol of a component $2^m f$ is of the form `(m,n,s,d[,o])`, where

- `m` = valuation of the component
- `n` = dimension of f
- `d` = det(f) in {1,3,5,7}
- `s` = 0 (or 1) if even (or odd)
- `o` = oddity of f (= 0 if s = 0) in $\mathbf{Z}/8\mathbf{Z}$

INPUT:

- `A` – symmetric matrix with integer coefficients, non-degenerate
- `val` – non-negative integer; valuation of maximal 2-elementary divisor

OUTPUT:

a list of lists of integers (representing a Conway-Sloane 2-adic symbol)

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.genus import two_adic_symbol

sage: A = diagonal_matrix(ZZ, [1, 2, 3, 4])
sage: two_adic_symbol(A, 2)
[[0, 2, 3, 1, 4], [1, 1, 1, 1, 1], [2, 1, 1, 1, 1]]
```

CHAPTER

NINE

NORMAL FORMS FOR P-ADIC QUADRATIC AND BILINEAR FORMS

We represent a quadratic or bilinear form by its $n \times n$ Gram matrix G. Then two p-adic forms G and G' are integrally equivalent if and only if there is a matrix B in $GL(n, \mathbf{Z}_p)$ such that $G' = BGB^T$.

This module allows the computation of a normal form. This means that two p-adic forms are integrally equivalent if and only if they have the same normal form. Further, we can compute a transformation into normal form (up to finite precision).

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.normal_form import p_adic_normal_form
sage: G1 = Matrix(ZZ,4, [2, 0, 0, 1, 0, 2, 0, 1, 0, 0, 4, 2, 1, 1, 2, 6])
sage: G1
[2 0 0 1]
[0 2 0 1]
[0 0 4 2]
[1 1 2 6]
sage: G2 = Matrix(ZZ,4, [2, 1, 1, 0, 1, 2, 0, 0, 1, 0, 2, 0, 0, 0, 0, 16])
sage: G2
[ 2  1  1  0]
[ 1  2  0  0]
[ 1  0  2  0]
[ 0  0  0 16]
```

A computation reveals that both forms are equivalent over $\mathbf{Z}_2$:

```
sage: D1, U1 = p_adic_normal_form(G1,2, precision=30)
sage: D2, U2 = p_adic_normal_form(G1,2, precision=30)
sage: D1
[        2         1         0         0]
[        1         2         0         0]
[        0         0 2^2 + 2^3         0]
[        0         0         0       2^4]
sage: D2
[        2         1         0         0]
[        1         2         0         0]
[        0         0 2^2 + 2^3         0]
[        0         0         0       2^4]
```

Moreover, we have computed the 2-adic isomorphism:

```
sage: U = U2.inverse()*U1
sage: U*G1*U.T
```

(continues on next page)

(continued from previous page)

```
[          2 2^31 + 2^32 2^32 + 2^33           1]
[2^31 + 2^32           2        2^32           1]
[2^32 + 2^33        2^32         2^2           2]
[          1           1           2     2 + 2^2]
```

As you can see this isomorphism is only up to the precision we set before:

```
sage: (U*G1*U.T).change_ring(IntegerModRing(2^30))
[2 0 0 1]
[0 2 0 1]
[0 0 4 2]
[1 1 2 6]
```

If you are only interested if the forms are isomorphic, there are much faster ways:

```
sage: q1 = QuadraticForm(G1)
sage: q2 = QuadraticForm(G2)
sage: q1.is_locally_equivalent_to(q2,2)
True
```

SEEALSO:

```
:mod:`~sage.quadratic_forms.genera.genus`
:meth:`~sage.quadratic_forms.quadratic_form.QuadraticForm.is_locally_equivalent_to`
:meth:`~sage.modules.torsion_quadratic_module.TorsionQuadraticModule.normal_form`
```

AUTHORS:

- Simon Brandhorst (2018-01): initial version

`sage.quadratic_forms.genera.normal_form.`**`collect_small_blocks`**(*G*)

Return the blocks as list.

INPUT:

- `G` – a block_diagonal matrix consisting of 1 by 1 and 2 by 2 blocks

OUTPUT:

- a list of 1 by 1 and 2 by 2 matrices – the blocks

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.normal_form import collect_small_blocks
sage: W1 = Matrix([1])
sage: V = Matrix(ZZ, 2, [2, 1, 1, 2])
sage: L = [W1, V, V, W1, W1, V, W1, V]
sage: G = Matrix.block_diagonal(L)
sage: L == collect_small_blocks(G)
True
```

`sage.quadratic_forms.genera.normal_form.`**`p_adic_normal_form`**(*G*, *p*, *precision=None*, *partial=False*, *debug=False*)

Return the transformation to the p-adic normal form of a symmetric matrix.

Two `p-adic` quadratic forms are integrally equivalent if and only if their Gram matrices have the same normal form.

Let p be odd and u be the smallest non-square modulo p. The normal form is a block diagonal matrix with blocks $p^k G_k$ such that G_k is either the identity matrix or the identity matrix with the last diagonal entry replaced by u.

If $p = 2$ is even, define the 1 by 1 matrices:

```
sage: W1 = Matrix([1]); W1
[1]
sage: W3 = Matrix([3]); W3
[3]
sage: W5 = Matrix([5]); W5
[5]
sage: W7 = Matrix([7]); W7
[7]
```

and the 2 by 2 matrices:

```
sage: U = Matrix(2,[0,1,1,0]); U
[0 1]
[1 0]
sage: V = Matrix(2,[2,1,1,2]); V
[2 1]
[1 2]
```

For $p = 2$ the partial normal form is a block diagonal matrix with blocks $2^k G_k$ such that G_k is a block diagonal matrix of the form $[U, \ldots, U, V, Wa, Wb]$ where we allow V, Wa, Wb to be 0×0 matrices.

Further restrictions to the full normal form apply. We refer to [MirMor2009] IV Definition 4.6. for the details.

INPUT:

- `G` – a symmetric n by n matrix in $\mathbf{Q}$
- `p` – a prime number – it is not checked whether it is prime
- `precision` – if not set, the minimal possible is taken
- `partial` – boolean (default: `False`) if set, only the partial normal form is returned.

OUTPUT:

- `D` – the jordan matrix over $\mathbf{Q}_p$
- `B` – invertible transformation matrix over $\mathbf{Z}_p$, i.e., $D = B * G * B^T$

EXAMPLES:

```
sage: from sage.quadratic_forms.genera.normal_form import p_adic_normal_form
sage: D4 = Matrix(ZZ, 4, [2,-1,-1,-1,-1,2,0,0,-1,0,2,0,-1,0,0,2])
sage: D4
[ 2 -1 -1 -1]
[-1  2  0  0]
[-1  0  2  0]
[-1  0  0  2]
sage: D, B = p_adic_normal_form(D4, 2)
sage: D
[  2   1   0   0]
[  1   2   0   0]
[  0   0 2^2   2]
[  0   0   2 2^2]
```

(continues on next page)

(continued from previous page)

```
sage: D == B * D4 * B.T
True
sage: A4 = Matrix(ZZ, 4, [2, -1, 0, 0, -1, 2, -1, 0, 0, -1, 2, -1, 0, 0, -1, 2])
sage: A4
[ 2 -1  0  0]
[-1  2 -1  0]
[ 0 -1  2 -1]
[ 0  0 -1  2]
sage: D, B = p_adic_normal_form(A4, 2)
sage: D
[0 1 0 0]
[1 0 0 0]
[0 0 2 1]
[0 0 1 2]
```

We can handle degenerate forms:

```
sage: A4_extended = Matrix(ZZ, 5, [2, -1, 0, 0, -1, -1, 2, -1, 0, 0, 0, -1, 2, -1,
→0, 0, 0, -1, 2, -1, -1, 0, 0, -1, 2])
sage: D, B = p_adic_normal_form(A4_extended, 5)
sage: D
[1 0 0 0 0]
[0 1 0 0 0]
[0 0 1 0 0]
[0 0 0 5 0]
[0 0 0 0 0]
```

and denominators:

```
sage: A4dual = A4.inverse()
sage: D, B = p_adic_normal_form(A4dual, 5)
sage: D
[5^-1    0    0    0]
[   0    1    0    0]
[   0    0    1    0]
[   0    0    0    1]
```

CHAPTER

TEN

SOLVING QUADRATIC EQUATIONS

Interface to the PARI/GP quadratic forms code of Denis Simon.

AUTHORS:

- Denis Simon (GP code)
- Nick Alexander (Sage interface)
- Jeroen Demeyer (2014-09-23): use PARI instead of GP scripts, return vectors instead of tuples (github issue #16997).
- Tyler Gaona (2015-11-14): added the $solve$ method

`sage.quadratic_forms.qfsolve.`**`qfparam`**(G, sol)

Parametrize the conic defined by the matrix G.

INPUT:

- `G` – a 3×3-matrix over $\mathbf{Q}$
- `sol` – a triple of rational numbers providing a solution to $x \cdot G \cdot x^t = 0$

OUTPUT:

A triple of polynomials that parametrizes all solutions of $x \cdot G \cdot x^t = 0$ up to scaling.

ALGORITHM:

Uses PARI/GP function pari:qfparam.

EXAMPLES:

```
sage: from sage.quadratic_forms.qfsolve import qfsolve, qfparam
sage: M = Matrix(QQ, [[0, 0, -12], [0, -12, 0], [-12, 0, -1]]); M
[  0   0 -12]
[  0 -12   0]
[-12   0  -1]
sage: sol = qfsolve(M)
sage: ret = qfparam(M, sol); ret
(-12*t^2 - 1, 24*t, 24)
sage: ret.parent()
Ambient free module of rank 3 over the principal ideal domain
 Univariate Polynomial Ring in t over Rational Field
```

`sage.quadratic_forms.qfsolve.`**`qfsolve`**(G)

Find a solution $x = (x_0, ..., x_n)$ to $xGx^t = 0$ for an $n \times n$-matrix `G` over $\mathbf{Q}$.

OUTPUT:

If a solution exists, return a vector of rational numbers x. Otherwise, returns -1 if no solution exists over the reals or a prime p if no solution exists over the p-adic field $\mathbf{Q}_p$.

ALGORITHM:

Uses PARI/GP function pari:qfsolve.

EXAMPLES:

```
sage: from sage.quadratic_forms.qfsolve import qfsolve
sage: M = Matrix(QQ, [[0, 0, -12], [0, -12, 0], [-12, 0, -1]]); M
[  0   0 -12]
[  0 -12   0]
[-12   0  -1]
sage: sol = qfsolve(M); sol
(1, 0, 0)
sage: sol.parent()
Vector space of dimension 3 over Rational Field

sage: M = Matrix(QQ, [[1, 0, 0], [0, 1, 0], [0, 0, 1]])
sage: ret = qfsolve(M); ret
-1
sage: ret.parent()
Integer Ring

sage: M = Matrix(QQ, [[1, 0, 0], [0, 1, 0], [0, 0, -7]])
sage: qfsolve(M)
7

sage: M = Matrix(QQ, [[3, 0, 0, 0], [0, 5, 0, 0], [0, 0, -7, 0], [0, 0, 0, -11]])
sage: qfsolve(M)
(3, 4, -3, -2)
```

`sage.quadratic_forms.qfsolve.`**`solve`**(*self*, *c=0*)

Return a vector x such that `self(x) == c`.

INPUT:

- `c` – (default: 0) a rational number

OUTPUT: A non-zero vector x satisfying `self(x) == c`.

ALGORITHM:

Uses PARI's pari:qfsolve. Algorithm described by Jeroen Demeyer; see comments on github issue #19112

EXAMPLES:

```
sage: F = DiagonalQuadraticForm(QQ, [1, -1]); F
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 0 ]
[ * -1 ]
sage: F.solve()
(1, 1)
sage: F.solve(1)
(1, 0)
sage: F.solve(2)
(3/2, -1/2)
```

(continues on next page)

(continued from previous page)

```
sage: F.solve(3)
(2, -1)
```

```
sage: F = DiagonalQuadraticForm(QQ, [1, 1, 1, 1])
sage: F.solve(7)
(1, 2, -1, -1)
sage: F.solve()
Traceback (most recent call last):
...
ArithmeticError: no solution found (local obstruction at -1)
```

```
sage: Q = QuadraticForm(QQ, 2, [17, 94, 130])
sage: x = Q.solve(5); x
(17, -6)
sage: Q(x)
5

sage: Q.solve(6)
Traceback (most recent call last):
...
ArithmeticError: no solution found (local obstruction at 3)

sage: G = DiagonalQuadraticForm(QQ, [5, -3, -2])
sage: x = G.solve(10); x
(3/2, -1/2, 1/2)
sage: G(x)
10

sage: F = DiagonalQuadraticForm(QQ, [1, -4])
sage: x = F.solve(); x
(2, 1)
sage: F(x)
0
```

```
sage: F = QuadraticForm(QQ, 4, [0, 0, 1, 0, 0, 0, 1, 0, 0, 0]); F
Quadratic form in 4 variables over Rational Field with coefficients:
[ 0 0 1 0 ]
[ * 0 0 1 ]
[ * * 0 0 ]
[ * * * 0 ]
sage: F.solve(23)
(23, 0, 1, 0)
```

Other fields besides the rationals are currently not supported:

```
sage: F = DiagonalQuadraticForm(GF(11), [1, 1])
sage: F.solve()
Traceback (most recent call last):
...
TypeError: solving quadratic forms is only implemented over QQ
```

CHAPTER

ELEVEN

HELPER CODE FOR TERNARY QUADRATIC FORMS

sage.quadratic_forms.ternary.**evaluate**(*a*, *b*, *c*, *r*, *s*, *t*, *v*)

Function to evaluate the ternary quadratic form (a, b, c, r, s, t) in a 3-tuple v.

EXAMPLES:

```
sage: from sage.quadratic_forms.ternary import evaluate
sage: Q = TernaryQF([1, 2, 3, -1, 0, 0])
sage: v = (1, -1, 19)
sage: Q(v)
1105
sage: evaluate(1, 2, 3, -1, 0, 0, v)
1105
```

sage.quadratic_forms.ternary.**extend**(*v*)

Return the coefficients of a matrix M such that M has determinant gcd(v) and the first column is v.

EXAMPLES:

```
sage: from sage.quadratic_forms.ternary import extend
sage: v = (6, 4, 12)
sage: m = extend(v)
sage: M = matrix(3, m); M
[ 6  1  0]
[ 4  1  0]
[12  0  1]
sage: M.det()
2
sage: v = (-12, 20, 30)
sage: m = extend(v)
sage: M = matrix(3, m)
sage: M
[-12   1   0]
[ 20  -2   1]
[ 30   0  -7]
sage: M.det()
2
```

sage.quadratic_forms.ternary.**primitivize**(*v0*, *v1*, *v2*, *p*)

Given a 3-tuple v not singular mod p, return a primitive 3-tuple version of v mod p.

EXAMPLES:

```
sage: from sage.quadratic_forms.ternary import primitivize
sage: primitivize(12, 13, 14, 5)
(3, 2, 1)
sage: primitivize(12, 13, 15, 5)
(4, 1, 0)
```

sage.quadratic_forms.ternary.**pseudorandom_primitive_zero_mod_p**(a, b, c, r, s, t, p)

Find a zero of the form $(a, b, 1)$ of the ternary quadratic form given by the coefficients (a, b, c, r, s, t) mod p, where p is a odd prime that doesn't divide the discriminant.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 2, -1, 0, 0])
sage: p = 1009
sage: from sage.quadratic_forms.ternary import pseudorandom_primitive_zero_mod_p
sage: v = pseudorandom_primitive_zero_mod_p(1, 2, 2, -1, 0, 0, p)              #␣
↪optional - sage.libs.pari
sage: v[2]                                                                      #␣
↪optional - sage.libs.pari
1
sage: Q(v)%p                                                                    #␣
↪optional - sage.libs.pari
0
```

sage.quadratic_forms.ternary.**red_mfact**(a, b)

Auxiliary function for reduction that finds the reduction factor of integers a, b.

INPUT:

- a, b – integers

OUTPUT: Integer

EXAMPLES:

```
sage: from sage.quadratic_forms.ternary import red_mfact
sage: red_mfact(0, 3)
0
sage: red_mfact(-5, 100)
9
```

CHAPTER

TWELVE

TERNARY QUADRATIC FORM WITH INTEGER COEFFICIENTS

AUTHOR:

- Gustavo Rama

Based in code of Gonzalo Tornaria

The form $a \cdot x^2 + b \cdot y^2 + c \cdot z^2 + r \cdot yz + s \cdot xz + t \cdot xy$ is stored as a tuple `(a, b, c, r, s, t)` of integers.

class `sage.quadratic_forms.ternary_qf.`**`TernaryQF`**`(v)`

Bases: `SageObject`

The `TernaryQF` class represents a quadratic form in 3 variables with coefficients in $\mathbf{Z}$.

INPUT:

- `v` – a list or tuple of 6 entries: `[a,b,c,r,s,t]`

OUTPUT:

- the ternary quadratic form $a \cdot x^2 + b \cdot y^2 + c \cdot z^2 + r \cdot y \cdot z + s \cdot x \cdot z + t \cdot x \cdot y$.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 3, 4, 5, 6]); Q
Ternary quadratic form with integer coefficients:
[1 2 3]
[4 5 6]
sage: A = matrix(ZZ, 3, [1, -7, 1, 0, -2, 1, 0, -1, 0])
sage: Q(A)
Ternary quadratic form with integer coefficients:
[1 187 9]
[-85 8 -31]
sage: TestSuite(TernaryQF).run()
```

`adjoint``()`

Return the adjoint form associated to the given ternary quadratic form.

That is, the Hessian matrix of the adjoint form is twice the classical adjoint matrix of the Hessian matrix of the given form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 17, 0, 0, 1])
sage: Q.adjoint()
Ternary quadratic form with integer coefficients:
[68 68 3]
```

(continues on next page)

(continued from previous page)

```
[0 0 -68]
sage: Q.adjoint().matrix() == 2*Q.matrix().adjoint_classical()
True
```

automorphism_spin_norm(*A*)

Return the spin norm of the automorphism A.

EXAMPLES:

```
sage: Q = TernaryQF([9, 12, 30, -26, -28, 20])
sage: A = matrix(ZZ, 3, [9, 10, -10, -6, -7, 6, 2, 2, -3])
sage: A.det()
1
sage: Q(A) == Q
True
sage: Q.automorphism_spin_norm(A)
7
```

automorphism_symmetries(*A*)

Given the automorphism A, if A is the identity, return the empty list. Otherwise, return a list of two vectors v_1, v_2 such that the product of the symmetries of the ternary quadratic form given by the two vectors is A.

EXAMPLES:

```
sage: Q = TernaryQF([9, 12, 30, -26, -28, 20])
sage: A = matrix(ZZ, 3, [9, 10, -10, -6, -7, 6, 2, 2, -3])
sage: Q(A) == Q
True
sage: v1, v2 = Q.automorphism_symmetries(A)
sage: v1, v2
((8, -6, 2), (1, -5/4, -1/4))
sage: A1 = Q.symmetry(v1)
sage: A1
[    9     9   -13]
[   -6 -23/4  39/4]
[    2   9/4  -9/4]
sage: A2 = Q.symmetry(v2)
sage: A2
[    1     1     3]
[    0  -1/4 -15/4]
[    0  -1/4   1/4]
sage: A1*A2 == A
True
sage: Q.automorphism_symmetries(identity_matrix(ZZ,3))
[]
```

automorphisms(*slow=True*)

Return a list with the automorphisms of the definite ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 7, 0, 0, 0])
sage: auts = Q.automorphisms(); auts
```

(continues on next page)

(continued from previous page)

```
[
[-1  0  0]  [-1  0  0]  [ 0 -1  0]  [ 0 -1  0]  [ 0  1  0]  [ 0  1  0]
[ 0 -1  0]  [ 0  1  0]  [-1  0  0]  [ 1  0  0]  [-1  0  0]  [ 1  0  0]
[ 0  0  1], [ 0  0 -1], [ 0  0 -1], [ 0  0  1], [ 0  0  1], [ 0  0 -1],
[ 1  0  0]  [1 0 0]
[ 0 -1  0]  [0 1 0]
[ 0  0 -1], [0 0 1]
]
sage: all(Q == Q(A) for A in auts)
True
sage: Q = TernaryQF([3, 4, 5, 3, 3, 2])
sage: Q.automorphisms(slow=False)
[
[1 0 0]
[0 1 0]
[0 0 1]
]
sage: Q = TernaryQF([4, 2, 4, 3, -4, -5])
sage: auts = Q.automorphisms(slow=False)
sage: auts
[
[1 0 0]  [ 2 -1 -1]
[0 1 0]  [ 3 -2 -1]
[0 0 1], [ 0  0 -1]
]
sage: A = auts[1]
sage: Q(A) == Q
True
sage: Qr, M_red = Q.reduced_form_eisenstein()
sage: Qr
Ternary quadratic form with integer coefficients:
[1 2 3]
[-1 0 -1]
sage: Q(A*M_red) == Qr
True
```

basic_lemma(p)

Find a number represented by `self` and coprime to the prime p.

EXAMPLES:

```
sage: Q = TernaryQF([3, 3, 3, -2, 0, -1])
sage: Q.basic_lemma(3)
4
```

coefficient(n)

Return the n-th coefficient of the ternary quadratic form.

INPUT:

- n – integer with $0 \leq n \leq 5$.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 3, 4, 5, 6]); Q
Ternary quadratic form with integer coefficients:
[1 2 3]
[4 5 6]
sage: Q.coefficient(2)
3
sage: Q.coefficient(5)
6
```

coefficients()

Return the list of coefficients of the ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 3, 4, 5, 6]); Q
Ternary quadratic form with integer coefficients:
[1 2 3]
[4 5 6]
sage: Q.coefficients()
(1, 2, 3, 4, 5, 6)
```

content()

Return the greatest common divisor of the coefficients of the given ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 2, 0, 0, 0])
sage: Q.content()
1
sage: Q = TernaryQF([2, 4, 6, 0, 0, 0])
sage: Q.content()
2
sage: Q.scale_by_factor(100).content()
200
```

delta()

Return the omega of the adjoint of the given ternary quadratic form, which is the same as the omega of the reciprocal form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 2, -1, 0, -1])
sage: Q.delta()
208
sage: Q.adjoint().omega()
208
sage: Q = TernaryQF([1, -1, 1, 0, 0, 0])
sage: Q.delta()
4
sage: Q.omega()
4
```

disc()

Return the discriminant of the ternary quadratic form, this is the determinant of the matrix divided by 2.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 2, 0, -1, 4])
sage: Q.disc()
-25
sage: Q.matrix().det()
-50
```

divisor()

Return the content of the adjoint form associated to the given form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 17, 0, 0, 0])
sage: Q.divisor()
4
```

find_p_neighbor_from_vec(*p*, *v*, *mat=False*)

Finds the reduced equivalent of the p-neighbor of this ternary quadratic form associated to a given vector v satisfying:

1. $Q(v) = 0$ (mod p)
2. v is a non-singular point of the conic $Q(v) = 0$ (mod p).

REFERENCES:

Gonzalo Tornaria's Thesis, Thrm 3.5, p34.

EXAMPLES:

```
sage: Q = TernaryQF([1, 3, 3, -2, 0, -1]); Q
Ternary quadratic form with integer coefficients:
[1 3 3]
[-2 0 -1]
sage: Q.disc()
29
sage: v = (9, 7, 1)
sage: v in Q.find_zeros_mod_p(11)                                          #␣
↪optional - sage.libs.pari
True
sage: Q11, M = Q.find_p_neighbor_from_vec(11, v, mat=True)                 #␣
↪optional - sage.libs.pari
sage: Q11                                                                  #␣
↪optional - sage.libs.pari
Ternary quadratic form with integer coefficients:
[1 2 4]
[-1 -1 0]
sage: M                                                                    #␣
↪optional - sage.libs.pari
[    -1  -5/11   7/11]
[     0 -10/11   3/11]
[     0  -3/11  13/11]
sage: Q(M) == Q11                                                          #␣
↪optional - sage.libs.pari
True
```

find_p_neighbors(*p*, *mat=False*)

Find a list with all the reduced equivalent of the p-neighbors of this ternary quadratic form, given by the zeros mod p of the form. See *find_p_neighbor_from_vec()* for more information.

EXAMPLES:

```
sage: Q0 = TernaryQF([1, 3, 3, -2, 0, -1])
sage: Q0
Ternary quadratic form with integer coefficients:
[1 3 3]
[-2 0 -1]
sage: neig = Q0.find_p_neighbors(5)                                    #␣
↪optional - sage.libs.pari
sage: len(neig)                                                        #␣
↪optional - sage.libs.pari
6
sage: Q1 = TernaryQF([1, 1, 10, 1, 1, 1])
sage: Q2 = TernaryQF([1, 2, 4, -1, -1, 0])
sage: neig.count(Q0)                                                   #␣
↪optional - sage.libs.pari
2
sage: neig.count(Q1)                                                   #␣
↪optional - sage.libs.pari
1
sage: neig.count(Q2)                                                   #␣
↪optional - sage.libs.pari
3
```

find_zeros_mod_p(*p*)

Find the zeros of the given ternary quadratic positive definite form modulo a prime p, where p doesn't divide the discriminant of the form.

EXAMPLES:

```
sage: Q = TernaryQF([4, 7, 8, -4, -1, -3])
sage: Q.is_positive_definite()
True
sage: Q.disc().factor()
3 * 13 * 19
sage: Q.find_zeros_mod_p(2)
[(1, 0, 0), (1, 1, 0), (0, 0, 1)]
sage: zeros_17 = Q.find_zeros_mod_p(17)                                #␣
↪optional - sage.libs.pari
sage: len(zeros_17)                                                    #␣
↪optional - sage.libs.pari
18
sage: [Q(v)%17 for v in zeros_17]                                      #␣
↪optional - sage.libs.pari
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

is_definite()

Determine if the ternary quadratic form is definite.

EXAMPLES:

```
sage: Q = TernaryQF([10, 10, 1, -1, 2, 3])
sage: Q.is_definite()
True
sage: (-Q).is_definite()
True
sage: Q = TernaryQF([1, 1, 2, -3, 0, -1])
sage: Q.is_definite()
False
```

is_eisenstein_reduced()

Determine if the ternary quadratic form is Eisenstein reduced.

That is, if we have a ternary quadratic form:

```
[a b c]
[r s t]
```

then

1. $a \leq b \leq c$;
2. r, s, and t are all positive or all nonpositive;
3. $a \geq |t|$; $a \geq |s|$; $b \geq |r|$;
4. $a + b + r + s + t \geq 0$;
5. $a = t$ implies $s \leq 2 \cdot r$; $a = s$ implies $t \leq 2 \cdot r$; $b = r$ implies $t \leq 2 \cdot s$;
6. $a = -t$ implies $s = 0$; $a = -s$ implies $t = 0$; $b = -r$ implies $t = 0$;
7. $a + b + r + s + t = 0$ implies $2 \cdot a + 2 \cdot s + t \leq 0$;
8. $a = b$ implies $|r| \leq |s|$; $b = c$ implies $|s| \leq |t|$.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 1, 0, 0, 0])
sage: Q.is_eisenstein_reduced()
True
sage: Q = TernaryQF([34, 14, 44, 12, 25, -22])
sage: Q.is_eisenstein_reduced()
False
```

is_negative_definite()

Determine if the ternary quadratic form is negative definite.

EXAMPLES:

```
sage: Q = TernaryQF([-8, -9, -10, 1, 9, -3])
sage: Q.is_negative_definite()
True
sage: Q = TernaryQF([-4, -1, 6, -5, 1, -5])
sage: Q((0, 0, 1))
6
sage: Q.is_negative_definite()
False
```

is_positive_definite()

Determine if the ternary quadratic form is positive definite.

EXAMPLES:

```
sage: Q = TernaryQF([10, 10, 1, -1, 2, 3])
sage: Q.is_positive_definite()
True
sage: (-Q).is_positive_definite()
False
sage: Q = TernaryQF([1, 1, 0, 0, 0, 0])
sage: Q.is_positive_definite()
False
sage: Q = TernaryQF([1, 1, 1, -1, -2, -3])
sage: Q((1,1,1))
-3
sage: Q.is_positive_definite()
False
```

is_primitive()

Determine if the ternary quadratic form is primitive.

This means that the greatest common divisor of the coefficients of the form is 1.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 3, 4, 5, 6])
sage: Q.is_primitive()
True
sage: Q.content()
1
sage: Q = TernaryQF([10, 10, 10, 5, 5, 5])
sage: Q.content()
5
sage: Q.is_primitive()
False
```

level()

Return the level of the ternary quadratic form, which is 4 times the discriminant divided by the divisor.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 2, -1, 0, -1])
sage: Q.level()
52
sage: 4*Q.disc()/Q.divisor()
52
```

matrix()

Return the Hessian matrix associated to the ternary quadratic form. That is, if Q is a ternary quadratic form, $Q(x,y,z) = a \cdot x^2 + b \cdot y^2 + c \cdot z^2 + r \cdot y \cdot z + s \cdot x \cdot z + t \cdot x \cdot y$, then the Hessian matrix associated to Q is

```
[2\cdot a t s]
[t 2\cdot b r]
[s r 2\cdot c]
```

EXAMPLES:

```
sage: Q = TernaryQF([1,1,2,0,-1,4]); Q
Ternary quadratic form with integer coefficients:
[1 1 2]
[0 -1 4]
sage: M = Q.matrix(); M
[ 2  4 -1]
[ 4  2  0]
[-1  0  4]
sage: v = vector((1, 2, 3))
sage: Q(v)
28
sage: (v*M*v.column())[0]//2
28
```

number_of_automorphisms(*slow=True*)

Return the number of automorphisms of the definite ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 7, 0, 0, 0])
sage: A = matrix(ZZ, 3, [0, 1, 0, -1, 5, 0, -8, -1, 1])
sage: A.det()
1
sage: Q1 = Q(A); Q1
Ternary quadratic form with integer coefficients:
[449 33 7]
[-14 -112 102]
sage: Q1.number_of_automorphisms()
8
sage: Q = TernaryQF([-19, -7, -6, -12, 20, 23])
sage: Q.is_negative_definite()
True
sage: Q.number_of_automorphisms(slow=False)
24
```

omega()

Return the content of the adjoint of the primitive associated ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([4, 11, 12, 0, -4, 0])
sage: Q.omega()
176
sage: Q.primitive().adjoint().content()
176
```

polynomial(*names='x,y,z'*)

Return the polynomial associated to the ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 0, 2, -3, -1]); Q
Ternary quadratic form with integer coefficients:
```

(continues on next page)

(continued from previous page)

```
[1 1 0]
[2 -3 -1]
sage: p = Q.polynomial(); p
x^2 - x*y + y^2 - 3*x*z + 2*y*z
sage: p.parent()
Multivariate Polynomial Ring in x, y, z over Integer Ring
```

possible_automorphisms = None

primitive()

Return the primitive version of the ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([2, 2, 2, 1, 1, 1])
sage: Q.is_primitive()
True
sage: Q.primitive()
Ternary quadratic form with integer coefficients:
[2 2 2]
[1 1 1]
sage: Q.primitive() == Q
True
sage: Q = TernaryQF([10, 10, 10, 5, 5, 5])
sage: Q.primitive()
Ternary quadratic form with integer coefficients:
[2 2 2]
[1 1 1]
```

pseudorandom_primitive_zero_mod_p(p)

Return a tuple of the form $v = (a, b, 1)$ such that is a zero of the given ternary quadratic positive definite form modulo an odd prime p, where p doesn't divides the discriminant of the form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 11, 0, -1, 0])
sage: Q.disc()
43
sage: Q.pseudorandom_primitive_zero_mod_p(3)  # random
(1, 2, 1)
sage: Q((1, 2, 1))
15
sage: v = Q.pseudorandom_primitive_zero_mod_p(1009)                    #␣
↪optional - sage.libs.pari
sage: Q(v) % 1009                                                       #␣
↪optional - sage.libs.pari
0
sage: v[2]                                                              #␣
↪optional - sage.libs.pari
1
```

quadratic_form()

Return a *QuadraticForm* with the same coefficients as `self` over $\mathbf{Z}$.

EXAMPLES:

```
sage: Q = TernaryQF([1, 2, 3, 1, 1, 1])
sage: QF1 = Q.quadratic_form(); QF1
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 1 1 1 ]
[ * 2 1 ]
[ * * 3 ]
sage: QF2 = QuadraticForm(ZZ, 3, [1, 1, 1, 2, 1, 3])
sage: bool(QF1 == QF2)
True
```

reciprocal()

Return the reciprocal quadratic form associated to the given form.

This is defined as the multiple of the primitive adjoint with the same content as the given form.

EXAMPLES:

```
sage: Q = TernaryQF([2, 2, 14, 0, 0, 0])
sage: Q.reciprocal()
Ternary quadratic form with integer coefficients:
[14 14 2]
[0 0 0]
sage: Q.content()
2
sage: Q.reciprocal().content()
2
sage: Q.adjoint().content()
16
```

reciprocal_reduced()

Return the reduced form of the reciprocal form of the given ternary quadratic form.

EXAMPLES:

```
sage: Q = TernaryQF([1, 1, 3, 0, -1, 0])
sage: Qrr = Q.reciprocal_reduced(); Qrr
Ternary quadratic form with integer coefficients:
[4 11 12]
[0 -4 0]
sage: Q.is_eisenstein_reduced()
True
sage: Qr = Q.reciprocal()
sage: Qr.reduced_form_eisenstein(matrix=False) == Qrr
True
```

reduced_form_eisenstein(*matrix=True*)

Return the Eisenstein reduced form equivalent to the given positive ternary quadratic form, which is unique.

EXAMPLES:

```
sage: Q = TernaryQF([293, 315, 756, 908, 929, 522])
sage: Qr, m = Q.reduced_form_eisenstein()
sage: Qr
```

(continues on next page)

(continued from previous page)

```
Ternary quadratic form with integer coefficients:
[1 2 2]
[-1 0 -1]
sage: Qr.is_eisenstein_reduced()
True
sage: m
[ -54  137  -38]
[ -23   58  -16]
[  47 -119   33]
sage: m.det()
1
sage: Q(m) == Qr
True
sage: Q = TernaryQF([12,36,3,14,-7,-19])
sage: Q.reduced_form_eisenstein(matrix = False)
Ternary quadratic form with integer coefficients:
[3 8 20]
[3 2 1]
```

scale_by_factor(k)

Scale the values of the ternary quadratic form by the number k.

OUTPUT:

If k times the content of the ternary quadratic form is an integer, return a ternary quadratic form; otherwise, return a quadratic form of dimension 3.

EXAMPLES:

```
sage: Q = TernaryQF([2, 2, 4, 0, -2, 8])
sage: Q
Ternary quadratic form with integer coefficients:
[2 2 4]
[0 -2 8]
sage: Q.scale_by_factor(5)
Ternary quadratic form with integer coefficients:
[10 10 20]
[0 -10 40]
sage: Q.scale_by_factor(1/2)
Ternary quadratic form with integer coefficients:
[1 1 2]
[0 -1 4]
sage: Q.scale_by_factor(1/3)
Quadratic form in 3 variables over Rational Field with coefficients:
[ 2/3 8/3 -2/3 ]
[ * 2/3 0 ]
[ * * 4/3 ]
```

symmetry(v)

Return A, the automorphism of the ternary quadratic form such that:

- $Av = -v$,
- $Au = 0$, if u is orthogonal to v,

where v is a given vector.

EXAMPLES:

```
sage: Q = TernaryQF([4, 5, 8, 5, 2, 2])
sage: v = vector((1,1,1))
sage: M = Q.symmetry(v)
sage: M
[  7/13 -17/26 -23/26]
[ -6/13   9/26 -23/26]
[ -6/13 -17/26   3/26]
sage: M.det()
-1
sage: M*v
(-1, -1, -1)
sage: v1 = vector((23, 0, -12))
sage: v2 = vector((0, 23, -17))
sage: v1*Q.matrix()*v
0
sage: v2*Q.matrix()*v
0
sage: M*v1 == v1
True
sage: M*v2 == v2
True
```

xi(p)

Return the value of the genus characters Xi_p… which may be missing one character. We allow -1 as a prime.

REFERENCES:

Dickson's "Studies in the Theory of Numbers"

EXAMPLES:

```
sage: Q1 = TernaryQF([26, 42, 53, -36, -17, -3])
sage: Q2 = Q1.find_p_neighbors(2)[1]
sage: Q1.omega()
3
sage: Q1.xi(3), Q2.xi(3)
(-1, -1)
```

xi_rec(p)

Return Xi(p) for the reciprocal form.

EXAMPLES:

```
sage: Q1 = TernaryQF([1, 1, 7, 0, 0, 0])
sage: Q2 = Q1.find_p_neighbors(3)[0]
sage: Q1.delta()
28
sage: Q1.xi_rec(7), Q2.xi_rec(7)
(1, 1)
```

sage.quadratic_forms.ternary_qf.**find_a_ternary_qf_by_level_disc**(N, d)

Find a reduced ternary quadratic form given its discriminant d and level N. If $N|4d$ and $d|N^2$, then it may be a form with that discriminant and level.

EXAMPLES:

```
sage: Q1 = find_a_ternary_qf_by_level_disc(44, 11); Q1
Ternary quadratic form with integer coefficients:
[1 1 3]
[0 -1 0]
sage: Q2 = find_a_ternary_qf_by_level_disc(44, 11^2 * 16)
sage: Q2
Ternary quadratic form with integer coefficients:
[3 15 15]
[-14 -2 -2]
sage: Q1.is_eisenstein_reduced()
True
sage: Q1.level()
44
sage: Q1.disc()
11
sage: find_a_ternary_qf_by_level_disc(44, 22)
sage: find_a_ternary_qf_by_level_disc(44, 33)
Traceback (most recent call last):
...
ValueError: There are no ternary forms of this level and discriminant
```

sage.quadratic_forms.ternary_qf.**find_all_ternary_qf_by_level_disc**(N, d)

Find the coefficients of all the reduced ternary quadratic forms given its discriminant d and level N.

If $N|4d$ and $d|N^2$, then it may be some forms with that discriminant and level.

EXAMPLES:

```
sage: find_all_ternary_qf_by_level_disc(44, 11)
[Ternary quadratic form with integer coefficients:
[1 1 3]
[0 -1 0], Ternary quadratic form with integer coefficients:
[1 1 4]
[1 1 1]]
sage: find_all_ternary_qf_by_level_disc(44, 11^2 * 16)
[Ternary quadratic form with integer coefficients:
[3 15 15]
[-14 -2 -2], Ternary quadratic form with integer coefficients:
[4 11 12]
[0 -4 0]]
sage: Q = TernaryQF([1, 1, 3, 0, -1, 0])
sage: Q.is_eisenstein_reduced()
True
sage: Q.reciprocal_reduced()
Ternary quadratic form with integer coefficients:
[4 11 12]
[0 -4 0]
sage: find_all_ternary_qf_by_level_disc(44, 22)
[]
```

(continues on next page)

(continued from previous page)

```
sage: find_all_ternary_qf_by_level_disc(44, 33)
Traceback (most recent call last):
...
ValueError: There are no ternary forms of this level and discriminant
```

CHAPTER

THIRTEEN

EVALUATION

`sage.quadratic_forms.quadratic_form__evaluate.`**`QFEvaluateMatrix`**$(Q, M, Q2)$

Evaluate this quadratic form Q on a matrix M of elements coercible to the base ring of the quadratic form, which in matrix notation is given by:

$$Q_2 = M^t \cdot Q \cdot M.$$

Note: This is a Python wrapper for the fast evaluation routine `QFEvaluateMatrix_cdef()`. This routine is for internal use and is called more conveniently as `Q(M)`. The inclusion of `Q2` as an argument is to avoid having to create a *`QuadraticForm()`* here, which for now creates circular imports.

INPUT:

- `Q` – *`QuadraticForm`* over a base ring R
- `M` – a `Q.dim()` $\times$ `Q2.dim()` matrix of elements of R

OUTPUT: a *`QuadraticForm`* over R

EXAMPLES:

```
sage: from sage.quadratic_forms.quadratic_form__evaluate import QFEvaluateMatrix
sage: Q = QuadraticForm(ZZ, 4, range(10)); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 0 1 2 3 ]
[ * 4 5 6 ]
[ * * 7 8 ]
[ * * * 9 ]
sage: Q2 = QuadraticForm(ZZ, 2)
sage: M = Matrix(ZZ, 4, 2, [1,0,0,0, 0,1,0,0]); M
[1 0]
[0 0]
[0 1]
[0 0]
sage: QFEvaluateMatrix(Q, M, Q2)
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 0 2 ]
[ * 7 ]
```

`sage.quadratic_forms.quadratic_form__evaluate.`**`QFEvaluateVector`**(Q, v)

Evaluate this quadratic form Q on a vector or matrix of elements coercible to the base ring of the quadratic form.

If a vector is given, then the output will be the ring element $Q(v)$, but if a matrix is given, then the output will be the quadratic form Q' which in matrix notation is given by:

$$Q' = v^t \cdot Q \cdot v.$$

Note: This is a Python wrapper for the fast evaluation routine `QFEvaluateVector_cdef()`. This routine is for internal use and is called more conveniently as `Q(M)`.

INPUT:

- `Q` – *QuadraticForm* over a base ring R
- `v` – a tuple or list (or column matrix) of `Q.dim()` elements of R

OUTPUT: an element of R

EXAMPLES:

```
sage: from sage.quadratic_forms.quadratic_form__evaluate import QFEvaluateVector
sage: Q = QuadraticForm(ZZ, 4, range(10)); Q
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 0 1 2 3 ]
[ * 4 5 6 ]
[ * * 7 8 ]
[ * * * 9 ]
sage: QFEvaluateVector(Q, (1,0,0,0))
0
sage: QFEvaluateVector(Q, (1,0,1,0))
9
```

CHAPTER

FOURTEEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

q

INDEX

D

E

F

G

H

J

K

L

M

N

O

P

Q

R

S