Sage Reference Manual: Quadratic Forms

Release 6.3

The Sage Development Team

August 11, 2014

CONTENTS

CHAPTER

ONE

QUADRATIC FORMS OVERVIEW

AUTHORS:

- Jon Hanke (2007-06-19)
- Anna Haensch (2010-07-01): Formatting and ReSTification

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

Returns 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])
sage: 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: sage.structure.sage_object.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 lexographically, or equivalently, by rows of the matrix)

2. `QuadraticForm(R, n)`, where
   - R – a ring
   - n – a symmetric $n \times n$ matrix with even diagonal (relative to R)
3. `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])
sage: 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])
sage: 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))
sage: 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: 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*)

Returns 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 : [[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]],
Genus symbol at 3 : [[0, 3, 1], [1, 1, -1]]]
```

GHY_mass__maximal()

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

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

INPUT: none

OUTPUT: a rational number

EXAMPLE:

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

Gram_det()

Gives 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()

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

$$Q(x) = x^t * A * x,$$

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()

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

$$Q(x) = x^t * A * x,$$

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()

Returns the Hessian matrix A for which Q(X) = $(1/2) * X^t * A * X$.

EXAMPLES:

```
sage: Q = QuadraticForm(QQ, 2, range(1,4))
sage: Q
Quadratic form in 2 variables over Rational Field with coefficients:
[ 1 2 ]
[ * 3 ]
sage: Q.Hessian_matrix()
[2 2]
[2 6]
sage: Q.matrix().base_ring()
Rational Field
```

Kitaoka_mass_at_2()

Returns 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".

INPUT: none

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)

Returns the local representation density of a form (for representing itself) defined over ZZ, 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()

Returns 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.

INPUT: none

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!
384
```

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

Performs the substitution $x_j - - > x_j + c * x_i$, which has the effect (on associated matrices) of symmetrically adding $c * j$-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 local_normal_form() method.

INPUT: c – an element of Q.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: Oops!  This coefficient can't be coerced to an element of the base ring for th
```

```
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()

Returns 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()

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

INPUT: None

OUTPUT: Returns a list of prime numbers >0.

EXAMPLES:

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

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

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

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()
Traceback (most recent call last):
...
ValueError: not an adjoint
```

automorphisms()

Return a list of the automorphisms of the quadratic form.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.number_of_automorphisms()                    # optional -- souvigner
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()                    # optional -- souvigner
16
sage: aut = Q.automorphisms()
sage: len(aut)
16
sage: print([Q(M) == Q for M in aut])
[True, True, True, True, True, True, True, True, True, True, True, True, True, True, T

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

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

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

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

base_ring()

Gives 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*)

Finds 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*)

Finds 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*, *safe_flag=True*)

Return a basis for ZZ^n made of vectors with minimal lengths Q(v).

The safe_flag 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.

OUTPUT: a list of vectors, and optionally a list of values for each vector.

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])
```

bilinear_map(*v*, *w*)

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

This does not work if the characteristic is 2:

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

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 **Z**, **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: From 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`.

TO DO: If we only care about working over the real double field (RDF), then we can use the `cholesky()` method 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 a
##////////////////////////////////////////////////////////////////////////////////////////
```

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)
1
sage: Q.clifford_invariant(37)
-1
sage: Q.clifford_conductor()
37

sage: DiagonalQuadraticForm(ZZ, [1, 1, 1]).clifford_conductor()
2
sage: QuadraticForm(ZZ, 3, [2, -2, 0, 2, 0, 5]).clifford_conductor()
30
```

For hyperbolic spaces, the clifford conductor is 1:

```
sage: H = QuadraticForm(ZZ, 2, [0, 1, 0])
sage: H.clifford_conductor()
1
sage: (H + H).clifford_conductor()
1
sage: (H + H + H).clifford_conductor()
1
sage: (H + H + H + H).clifford_conductor()
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)
1
sage: (H + H).clifford_invariant(2)
1
sage: (H + H + H).clifford_invariant(2)
1
sage: (H + H + H + H).clifford_invariant(2)
1
```

coefficients()

Gives 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*)

Finds the $(n-1)$-dim'l quadratic form orthogonal to the vector v.

Note: This is usually not a direct summand!

Technical Notes: 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 ZZ

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

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

sage: Q1.complementary_subform_to_vector([1,1,1,1])
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 624 -480 -672 ]
[ * 880 -1120 ]
[ * * 1008 ]
```

compute_definiteness()

Computes 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. Note: The zero-dim'l form is considered both positive definite and negative definite.

INPUT: QuadraticForm

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 self.compute_definiteness() for more documentation.

INPUT: None

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()

Returns 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)

Computes 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)
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)

Computes 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.

INPUT: none

OUTPUT: a rational number > 0

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.conway_mass()
1/48

sage: Q = DiagonalQuadraticForm(ZZ, [7,1,1])
sage: Q.conway_mass()
3/16

sage: Q = QuadraticForm(ZZ, 3, [7, 2, 2, 2, 0, 2]) + DiagonalQuadraticForm(ZZ, [1])
sage: Q.conway_mass()
3/32
```

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 1x1 blocks are all in the upper leftmost position.

INPUT: none

OUTPUT: an integer 0 <= x <= 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)

Computes 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()

Returns an integer called the 'species' which determines the type of the orthogonal group over the finite field 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 dim'l 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)

Returns an integer called the 'species' which determines the type of the orthogonal group over the finite field 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).

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

INPUT: 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()

Returns the infinite product of the standard mass factors.

INPUT: none

OUTPUT: a rational number > 0

EXAMPLES:

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

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.conway_standard_mass()
1/6
```

conway_standard_p_mass(p)

Computes 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 $p = 2$.

INPUT: none

OUTPUT: a rational number

EXAMPLES:

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

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

Counts all solutions of Q(x) $= m(mod p^k)$ 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 – a list 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*)

Counts the bad-type solutions of Q(x) = $m (mod p^k)$ 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 – a list 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*)

Counts the bad-typeI solutions of Q(x) = $m (mod p^k)$ 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 – a list 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*)

Counts the bad-typeII solutions of Q(x) = $m (mod p^k)$ 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 – a list 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*)

Counts the good-type solutions of Q(x) = m (mod 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 – a list 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*)

Counts the zero-type solutions of Q(x) = $m(modp^k)$ 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 – a list 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*)

Gives the number of integer solution vectors x satisfying the congruence Q(x) = $m(modp^k)$ 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.$x[i] == 0(modp)$ for all i in zvec

2.$x[i]! = 0(modp)foralli$ 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 – a list 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)

Returns the number of solutions of $Q(x) = m(modp)$ of a non-degenerate quadratic form over the finite field Z/pZ, 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()

Gives the determinant of the Gram matrix of 2*Q, or equivalently the determinant of the Hessian matrix of Q.

(Note: This is always defined over the same ring as the quadratic form.)

EXAMPLES:

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

dim()

Gives the number of variables of the quadratic form.

EXAMPLES:

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

disc()

Returns the discriminant of the quadratic form, defined as

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

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

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()

Returns 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 Q.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 $x_i --> x_i + c * x_j$ (replacing the original form if the in_place flag is True).

INPUT: c – an element of Q.base_ring()

i, j – 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.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))
sage: 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])
sage: 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(*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, v)

Finds the p-neighbor of this quadratic form associated to a given vector v satisfying:

1. $Q(v) = 0 \pmod p$

2. v is a non-singular point of the conic $Q(v) = 0 \pmod p$.

Reference: Gonzalo Tornaria's Thesis, Thrm 3.5, p34.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ,[1,1,1,1])
sage: v = vector([0,2,1,1])
sage: X = Q.find_p_neighbor_from_vec(3,v); X
Quadratic form in 4 variables over Integer Ring with coefficients:
[ 3 10 0 -4 ]
[ * 9 0 -6 ]
[ * * 1 0 ]
[ * * * 2 ]
```

find_primitive_p_divisible_vector__next(p, v=*None*)

Finds 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 intial call, no v needs to be passed.

Returns 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
```

find_primitive_p_divisible_vector__random(p)

Finds 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.

Warning: If there are local obstructions for this to happen, then this algorithm will never terminate... =(We should check for this too!

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [10,1,4])
sage: Q.find_primitive_p_divisible_vector__random(5)     # random
(1, 1)
sage: Q.find_primitive_p_divisible_vector__random(5)     # random
(1, 0)
sage: Q.find_primitive_p_divisible_vector__random(5)     # random
(2, 0)
sage: Q.find_primitive_p_divisible_vector__random(5)     # random
(2, 2)
sage: Q.find_primitive_p_divisible_vector__random(5)     # random
(3, 3)
sage: Q.find_primitive_p_divisible_vector__random(5)     # random
(3, 3)
sage: Q.find_primitive_p_divisible_vector__random(5)     # random
(2, 0)
```

gcd()

Returns 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
```

global_genus_symbol()

Returns the genus of a two times a quadratic form over ZZ. These are defined by a collection of local genus symbols (a la Chapter 15 of Conway-Sloane), 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]

sage: Q = QuadraticForm(ZZ, 4, range(10))
sage: Q.global_genus_symbol()
Genus of [ 0  1  2  3]
[ 1  8  5  6]
[ 2  5 14  8]
[ 3  6  8 18]
```

has_equivalent_Jordan_decomposition_at_prime(*other*, *p*)

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

INPUT: 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)
False
sage: Q1.has_equivalent_Jordan_decomposition_at_prime(Q2,11)
False
sage: Q1.has_equivalent_Jordan_decomposition_at_prime(Q3,2)
False
sage: Q1.has_equivalent_Jordan_decomposition_at_prime(Q3,11)
True
sage: Q2.has_equivalent_Jordan_decomposition_at_prime(Q3,2)
True
sage: Q2.has_equivalent_Jordan_decomposition_at_prime(Q3,11)
False
```

has_integral_Gram_matrix()

Returns 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)
-1
sage: Q.hasse_invariant(37)
-1
sage: Q.hasse_conductor()
74

sage: DiagonalQuadraticForm(ZZ, [1, 1, 1]).hasse_conductor()
1
sage: QuadraticForm(ZZ, 3, [2, -2, 0, 2, 0, 5]).hasse_conductor()
10
```

hasse_invariant(p)

Computes the Hasse invariant at a prime p, 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 Q_p for this to make sense.

WARNING: This is different from the O'Meara Hasse invariant, which allows $i <= j$ in the product. That is given by the method hasse_invariant__OMeara(p).

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

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)]
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]
[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)]
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]
[-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)]
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]
[-1, 1, 1, 1, 1, 1, 1, 1]

sage: K.<a>=NumberField(x^2-23)
sage: Q=DiagonalQuadraticForm(K,[-a,a+2])
sage: [Q.hasse_invariant(p) for p in K.primes_above(19)]
[1, -1]
```

hasse_invariant__OMeara(p)

Computes the O'Meara Hasse invariant at a prime p, as given on p167 of O'Meara's book. If Q is diagonal with coefficients a_i, then the (Cassels) Hasse invariant is given by

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

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

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)]
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]
[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)]
[1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]
[-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)]
[-1, 1, 1, 1, 1, 1, 1, 1]
sage: [Q.hasse_invariant__OMeara(p) for p in prime_range(20)]
[-1, 1, 1, 1, 1, 1, 1, 1]

sage: K.<a>=NumberField(x^2-23)
sage: Q=DiagonalQuadraticForm(K,[-a,a+2])
sage: [Q.hasse_invariant__OMeara(p) for p in K.primes_above(19)]
[1, 1]
```

is_adjoint()

Determines 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()
False
sage: Q.adjoint().is_adjoint()
True
```

is_anisotropic(p)

Checks if the quadratic form is anisotropic over the p-adic numbers Q_p.

INPUT: p – a prime number > 0

OUTPUT: boolean

EXAMPLES:

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

sage: Q = DiagonalQuadraticForm(ZZ, [1,-1])
sage: Q.is_anisotropic(2)
False
sage: Q.is_anisotropic(3)
False
sage: Q.is_anisotropic(5)
False

sage: [DiagonalQuadraticForm(ZZ, [1, -least_quadratic_nonresidue(p)]).is_anisotropic(p)  for
[True, True, True, True, True, True, True, True, True]

sage: [DiagonalQuadraticForm(ZZ, [1, -least_quadratic_nonresidue(p), p, -p*least_quadratic_r
[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-dim'l form is considered indefinite.

INPUT: None

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*)

Returns 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 ZZ.

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__souvigner(*other*, *return_transformation=False*)

Uses the Souvigner code to compute the number of automorphisms.

INPUT: a QuadraticForm

OUTPUT: boolean, and optionally a matrix

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, -1, 2, -1, 5])
sage: Q1 = QuadraticForm(ZZ, 3, [8, 6, 5, 3, 4, 2])
sage: M = Q.is_globally_equivalent__souvigner(Q1, True) ; M   # optional -- souvigner
[ 0  0 -1]
[ 1  0  0]
[-1  1  1]
sage: Q1(M) == Q                                              # optional -- souvigner
True
```

is_globally_equivalent_to(*other*, *return_matrix=False*, *check_theta_to_precision='sturm'*, *check_local_equivalence=True*)

Determines if the current quadratic form is equivalent to the given form over ZZ. If return_matrix is True, then we also return the transformation matrix M so that self(M) == other.

INPUT: a QuadraticForm

OUTPUT: boolean, and optionally a 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.(Q1)                                                  # optional -- souvigner
True
sage: MM = Q.is_globally_equivalent_to(Q1, return_matrix=True)  # optional -- souvigner
sage: Q(MM) == Q1                                             # optional -- souvigner
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)                        # optional -- souvigner
False
sage: Q1.is_globally_equivalent_to(Q3)                        # optional -- souvigner
True
sage: M = Q1.is_globally_equivalent_to(Q3, True) ; M          # optional -- souvigner
[-1 -1  0]
[ 1  1  1]
[-1  0  0]
sage: Q1(M) == Q3                                             # optional -- souvigner
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()
```

is_hyperbolic(*p*)

Checks if the quadratic form is a sum of hyperbolic planes over the p-adic numbers Q_p.

REFERENCES:

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

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

INPUT: p – a prime number > 0

OUTPUT: boolean

EXAMPLES:

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

is_indefinite()

Determines if the given quadratic form is indefinite.

Note: A degenerate form is considered neither definite nor indefinite. Note: The zero-dim'l form is not considered indefinite.

INPUT: None

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 Q_p.

INPUT: p – a prime number > 0

OUTPUT: boolean

EXAMPLES:

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

sage: Q = DiagonalQuadraticForm(ZZ, [1,-1])
sage: Q.is_isotropic(2)
True
sage: Q.is_isotropic(3)
True
sage: Q.is_isotropic(5)
True
```

```
sage: [DiagonalQuadraticForm(ZZ, [1, -least_quadratic_nonresidue(p)]).is_isotropic(p)  for p
[False, False, False, False, False, False, False, False, False]

sage: [DiagonalQuadraticForm(ZZ, [1, -least_quadratic_nonresidue(p), p, -p*least_quadratic_n
[False, False, False, False, False, False, False, False, False]
```

is_locally_equivalent_to(*other*, *check_primes_only=False*, *force_jordan_equivalence_test=False*)
Determines if the current quadratic form (defined over ZZ) 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: 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)                          # optional -- souvigner
False
sage: Q1.is_locally_equivalent_to(Q2)
True
```

is_locally_represented_number(m)
Determines 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
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)
Determines 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()

Determines if the quadratic form represents Z_p for all finite/non-archimedean primes, and represents all real numbers.

INPUT: none

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()

Determines if the quadratic form represents Z_p for all finite/non-archimedean primes.

INPUT: none

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*)

Determines if the (integer-valued/rational) quadratic form represents all of 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)
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-dim'l form is considered both positive definite and negative definite.

INPUT: None

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*)

Returns 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 ZZ.

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-dim'l form is considered both positive definite and negative definite.

INPUT: None

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

Determines if the vector v is on the conic Q(x) = 0 (mod 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*)

Determines if the vector v is on the conic Q(x) = 0 (mod 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*)

Determines if the vector v is on the conic Q(x) = 0 (mod 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*)

Returns 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 2x2 blocks are listed after the 1x1 blocks (which follows from the convention of the `local_normal_form()` method).

..note

```
The decomposition of each `L_i` into smaller block 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*)

Returns 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 ZZ), and the indexing only works correctly for p=2 when the form has an integer Gram matrix.

INPUT: self – a quadratic form over ZZ, 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: Oops!  The given quadratic form has a Jordan component with a negative scale expo
This routine requires an integer-matrix quadratic form for the output indexing to work prope

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 * (the matrix of 2*Q)^(-1) is in R with diagonal in 2*R.

Over **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 real
1

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

level__Tornaria()

Returns the level of the quadratic form, defined as

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

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

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()

Determines the level of the quadratic form (over R), which is the smallest ideal N of R such that N * (the matrix of 2*Q)^(-1) is in R with diagonal in 2*R. (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()

Returns 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()
sage: Q.list_external_initializations()
```

```
[]
sage: Q = QuadraticForm(ZZ, 2, [1,0,5], unsafe_initialization=False, number_of_automorphisms
sage: Q.list_external_initializations()
[]

sage: Q = QuadraticForm(ZZ, 2, [1,0,5], unsafe_initialization=False, number_of_automorphisms
sage: Q.list_external_initializations()
[]
sage: Q = QuadraticForm(ZZ, 2, [1,0,5], unsafe_initialization=True, number_of_automorphisms=
sage: Q.list_external_initializations()
['number_of_automorphisms', 'determinant']
```

lll()

Returns 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*)

Finds 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: Q – quadratic form 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*)

Finds 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: Q – quadratic form 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)
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*)

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

INPUT: Q – quadratic form 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)
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*)

Gives the local 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 = DiagonalQuadraticForm(ZZ, [1,1,1,1])   ## NOTE: This is already in local normal fo
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*)

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

INPUT: Q – quadratic form 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*)

Returns the Conway-Sloane genus symbol of 2 times a quadratic form defined over ZZ at a prime number p. This is defined (in the Genus_Symbol_p_adic_ring() class in the quadratic_forms/genera subfolder) to be a list of tuples (one for each Jordan component p^m*A at p, where A is a unimodular symmetric matrix with coefficients the p-adic integers) of the following form:

1. **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.

2. **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 Z/8Z** = 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) for a discussion of this convention.

INPUT:

-p – a prime number > 0

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

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,2,3,4])
sage: Q.local_genus_symbol(2)
Genus symbol at 2 : [[1, 2, 3, 1, 4], [2, 1, 1, 1, 1], [3, 1, 1, 1, 1]]
sage: Q.local_genus_symbol(3)
Genus symbol at 3 : [[0, 3, 1], [1, 1, -1]]
sage: Q.local_genus_symbol(5)
Genus symbol at 5 : [[0, 4, 1]]
```

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

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

TO DO: Add Documentation about the additional congruence conditions Zvec and NZvec.

INPUT: Q – quadratic form 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*)

Finds 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) = $m (mod p^k)$, 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 1x1, and so there the indices and Jordan blocks coincide.

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

INPUT: Q – quadratic form 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)
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)
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_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*)

Finds 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.

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

INPUT: Q – quadratic form 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*)

Returns the a locally integrally equivalent quadratic form over the p-adic integers 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 1x1 blocks before the 2x2 blocks in each Jordan component.)

INPUT: p – a positive prime number.

OUTPUT: a quadratic form over ZZ

WARNING: Currently this only works for quadratic forms defined over ZZ.

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*)

Gives 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)
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*)

Finds 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: Q – quadratic form 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 Z_p, and are labeled by the Q_p square-classes $t * (Q_p)^2$ with t given as follows:

$p > 2$ ==> [* 1 u p u p * * * *]

$p = 2$ ==> [* 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 Q_p square-class is not represented, while a positive number x indicates that $t*p^{(2*x)}(Z_p)^2$ is locally represented but $t*p^{(2*(x-1))}(Z_p)^2$ 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:

$2^6 * u * r^2$ with $u = 1(mod 8)$

$2^5 * u * r^2$ with $u = 3(mod 8)$

$2 * u * r^2$ 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.

INPUT: none

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])
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*)

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

INPUT: Q – quadratic form 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
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'*)

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

WARNING: THIS IS BROKEN RIGHT NOW... =(

Optional Arguments:

- When p > 2 – odd_algorithm = "Pall" (only one choice for now)
- When p = 2 – even_algorithm = "Kitaoka" or "Watson"

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: Q.mass__by_Siegel_densities()
1/384
sage: Q.mass__by_Siegel_densities() - (2^Q.dim() * factorial(Q.dim()))^(-1)
0

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1])
sage: Q.mass__by_Siegel_densities()
1/48
sage: Q.mass__by_Siegel_densities() - (2^Q.dim() * factorial(Q.dim()))^(-1)
0
```

mass_at_two_by_counting_mod_power(k)

Computes 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!!!

TO DO: Remove this routine, or try to compile it!

INPUT: k – an integer >= 1

OUTPUT: a rational number

EXAMPLE:

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

matrix()

Returns the Hessian matrix A for which Q(X) = $(1/2) * X^t * A * X$.

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

$$Q(v_k) <= Q(s_1 * v_1 + ... + s_n * v_n)$$

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 ]                                                    ,

[ 1  0  0  0]
[ 0  1  0 -1]
[ 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) <= Q($s_1 * v_1 + ... + s_n * v_n$)

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_for_4vars__SP()
(
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 $c * x_i$ 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 Q.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 ]
```

number_of_automorphisms(*recompute=False*)

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

If recompute is True, then we will recompute the cached result.

OUTPUT: an integer >= 2.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 3, [1, 0, 0, 1, 0, 1], unsafe_initialization=True, number_of_aut
sage: Q.list_external_initializations()
['number_of_automorphisms']
sage: Q.number_of_automorphisms()
-1
sage: Q.number_of_automorphisms(recompute=True)           # optional -- souvigner
48
sage: Q.list_external_initializations()                   # optional -- souvigner
[]

sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1])
sage: Q.number_of_automorphisms()                         # optional -- souvigner
```

```
384
sage: 2^4 * factorial(4)
384

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

number_of_automorphisms__souvigner()

Uses the Souvigner code to compute the number of automorphisms.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1,1,1,1,1])
sage: Q.number_of_automorphisms__souvigner()                    # optional -- souvigr
3840
sage: 2^5 * factorial(5)
3840
```

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
```

parity(*allow_rescaling_flag=True*)

Returns the parity ("even" or "odd") of an integer-valued quadratic form over ZZ, 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 ZZ, 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 1x1 block in the 0-th Jordan component, after a possible rescaling.

INPUT:

self – a quadratic form with base_ring ZZ, 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'*)

Returns the polynomial in 'n' variables of the quadratic form in the ring 'R[names].'

INPUT:

-'self' - a quadratic form over a commatitive ring. -'names' - the name of the variables. Digits will be appended to the name for each different canonical variable e.g x1, x2, x3 etc.

OUTPUT:

The polynomial form of the quadratic form.

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(QQ,[1, 3, 5, 7])
sage: P = Q.polynomial(); P
2*x0^2 + 6*x1^2 + 10*x2^2 + 14*x3^2

sage: F.<a> = NumberField(x^2 - 5)
sage: Z = F.ring_of_integers()
sage: Q = QuadraticForm(Z,3,[2*a, 3*a, 0 , 1 - a, 0, 2*a + 4])
sage: P = Q.polynomial(names='y'); P
4*a*y0^2 + 6*a*y0*y1 + (-2*a + 2)*y1^2 + (4*a + 8)*y2^2
sage: Q = QuadraticForm(F,4,[a, 3*a, 0, 1 - a, a - 3, 0, 2*a + 4, 4 + a, 0, 1])
sage: Q.polynomial(names='z')
(2*a)*z0^2 + (6*a)*z0*z1 + (2*a - 6)*z1^2 + (2*a + 8)*z2^2 + (-2*a + 2)*z0*z3 + (4*a + 8)*z1
sage: B.<i,j,k> = QuaternionAlgebra(F,-1,-1)
sage: Q = QuadraticForm(B, 3, [2*a, 3*a, i, 1 - a, 0, 2*a + 4])
sage: Q.polynomial()
Traceback (most recent call last):
...
ValueError: Can only create polynomial rings over commutative rings.
```

primitive()

Returns a primitive version of an integer-valued quadratic form, defined over ZZ.

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*)

Returns a diagonal form equivalent to Q over the fraction field of its defining ring. If the return_matrix is True, then we return the transformation matrix performing the diagonalization as the second argument.

INPUT: none

OUTPUT: Q – the diagonalized form of this quadratic form (optional) T – matrix which diagonalizes Q (over it's fraction field)

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:
[ -2 0 ]
[ * 1/8 ]

sage: Q = DiagonalQuadraticForm(ZZ, [1,3,5,7])
sage: Q.rational_diagonal_form(return_matrix=True)
(
Quadratic form in 4 variables over Rational Field with coefficients:
[ 1 0 0 0 ]
[ * 3 0 0 ]
[ * * 5 0 ]
[ * * * 7 ]                                                              ,

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

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]
)
```

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()
(
Quadratic form in 2 variables over Integer Ring with coefficients:
[ 2 -1 ]
[ * 2 ]                                                         ,

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

reduced_binary_form1()

Reduce the form $ax^2 + bxy + cy^2$ to satisfy the reduced condition $|b| \leq a \leq c$, with $b \geq 0$ 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()
(
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: TO DO
```

representation_number_list(*B*)

Returns 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)
[1, 16, 112, 448, 1136, 2016, 3136, 5504, 9328, 12112]
```

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

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

EXAMPLES:

```
sage: Q = DiagonalQuadraticForm(ZZ, [1, 1])
sage: Q.representation_vector_list(10)
[[(0, 0)],
 [(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: map(len, _)
[1, 4, 4, 0, 4, 8, 0, 0, 4, 4]
sage: Q.representation_number_list(10)
[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().

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

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.

INPUT: none

OUTPUT: a rational number

EXAMPLE:

```
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 ZZ.

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, -
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,
```

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 short vectors
```

Sometimes, PARI does not compute short vectors correctly. It returns too long vectors.

```
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
[-65, 623]~
sage: v.Vec() * Q._pari_() * v  # long time
45907800
```

siegel_product(*u*)

Computes 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

sage: Q.siegel_product(1)
8
sage: Q.siegel_product(2)      ## This one is wrong -- expect 24, and the higher powers of 2
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
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  # long time (2s
3/2
```

TESTS:

```
sage: [1] + [Q.siegel_product(ZZ(a))  for a in range(1,11)] == Q.theta_series(11).list()  #
True
```

signature()

Returns the signature of the quadratic form, defined as:

number of positive eigenvalues - number of negative eigenvalues

of the matrix of the quadratic form.

INPUT: None

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()

Returns 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.

INPUT:

(none)

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)
```

split_local_cover()

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

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

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 it's complementary lattice and the original quadratic form Q.

INPUT: none

OUTPUT: a QuadraticForm over ZZ

EXAMPLES:

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

sum_by_coefficients_with(*right*)

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

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,4,10])
sage: 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: From 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 $O(q^M ax)$. 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, '')
sage: exact = [1] + [8 * sum([d  for d in divisors(i)  if d % 4 != 0])  for i in range(1, Pr
sage: compute == exact
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 $O(q^m ax)$.

This uses the PARI/GP function 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.

TO DO: 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'll need to be careful about this (particularly for half-integral weights!).

EXAMPLES:

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

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)]
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*)

Returns 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.

Reference: 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)
[[[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)
[[[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: map(len, Q.vectors_by_length(2))
[1, 12, 12]

sage: Q = QuadraticForm(ZZ, 4, [1,-1,-1,-1, 1,0,0, 4,-3, 4])
sage: map(len, Q.vectors_by_length(3))
[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.

Reference: 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), Q2.hasse_invariant(5)]     # equivalent over Q_5
[1, 1]
sage: [Q1.xi(5), Q2.xi(5)]                               # not equivalent over Z_5
[1, -1]
```

xi_rec(*p*)

Returns 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(), Q2.clifford_conductor()]   # equivalent over Q
[3, 3]
sage: Q1.is_locally_equivalent_to(Q2)                # not in the same genus
False
sage: [Q1.delta(), Q2.delta()]
[480, 480]
sage: factor(480)
2^5 * 3 * 5
sage: map(Q1.xi_rec, [-1,2,3,5])
[-1, -1, -1, 1]
sage: map(Q2.xi_rec, [-1,2,3,5])
[-1, -1, -1, -1]
```

sage.quadratic_forms.quadratic_form.**QuadraticForm__constructor**(*R*, *n=None*, *entries=None*)

Wrapper for the QuadraticForm class constructor. This is meant for internal use within the QuadraticForm class code only. You should instead directly call QuadraticForm().

EXAMPLES:

```
sage: from sage.quadratic_forms.quadratic_form import QuadraticForm__constructor
sage: QuadraticForm__constructor(ZZ, 3)   ## Makes a generic quadratic form over the integers
Quadratic form in 3 variables over Integer Ring with coefficients:
[ 0 0 0 ]
[ * 0 0 ]
[ * * 0 ]
```

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

Determines if the object Q is an element of the QuadraticForm class.

EXAMPLES:

```
sage: Q = QuadraticForm(ZZ, 2, [1,2,3])
sage: from sage.quadratic_forms.quadratic_form import is_QuadraticForm
sage: is_QuadraticForm(Q)  ##random
True
sage: is_QuadraticForm(2)  ##random
False
```

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()
x^2 + 2*y^2
sage: Q(1, 1)
6
```

TESTS:

```
sage: Q == loads(dumps(Q))
True
```

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.

class `sage.quadratic_forms.binary_qf.`**`BinaryQF`**(*abc*)
Bases: `sage.structure.sage_object.SageObject`

A binary quadratic form over **Z**.

INPUT:

- *v* – a list or tuple of 3 entries: [a,b,c], or a quadratic homogeneous polynomial in two variables with integer coefficients

OUTPUT:

the binary quadratic form a*x^2 + b*x*y + c*y^2.

EXAMPLES:

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

complex_point()

Returns the point in the complex upper half-plane associated to this (positive definite) quadratic form.

For positive definite forms with negative discriminants, this is a root τ of $ax^2 + bx + c$ with the imaginary part of τ greater than 0.

EXAMPLES:

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

discriminant()

Returns the discriminant $b^2 - 4ac$ of the binary form $ax^2 + bxy + cy^2$.

EXAMPLES:

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

has_fundamental_discriminant()

Checks if 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 mod 4).

EXAMPLES:

```
sage: Q = BinaryQF([1,0,1])
sage: Q.discriminant()
-4
sage: Q.has_fundamental_discriminant()
True

sage: Q = BinaryQF([2,0,2])
sage: Q.discriminant()
-16
sage: Q.has_fundamental_discriminant()
False
```

is_equivalent(*right*)

Return true if self and right are equivalent, i.e., have the same reduced form.

INPUT:

- `right` – a binary quadratic form

EXAMPLES:

```
sage: a = BinaryQF([33,11,5])
sage: b = a.reduced_form(); b
5*x^2 - x*y + 27*y^2
```

```
sage: a.is_equivalent(b)
True
sage: a.is_equivalent(BinaryQF((3,4,5)))
False
```

is_primitive()

Checks if 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

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)
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]
```

is_reduced()

Checks if the quadratic form is reduced, i.e., if the form $ax^2 + bxy + cy^2$ satisfies $|b| \leq a \leq c$, and that $b \geq 0$ if either $a = b$ or $a = c$.

EXAMPLES:

```
sage: Q = BinaryQF([1,2,3])
sage: Q.is_reduced()
False

sage: Q = BinaryQF([2,1,3])
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
```

is_weakly_reduced()

Checks 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
```

matrix_action_left(M)

Return the binary quadratic form resulting from the left action of the 2-by-2 matrix `M` on the quadratic form `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_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 the quadratic form `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_right(M)
32*x^2 + 109*x*y + 93*y^2
```

polynomial()

Returns the binary quadratic form 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()

Return the unique reduced form equivalent to `self`. See also `is_reduced()`.

EXAMPLES:

```
sage: a = BinaryQF([33,11,5])
sage: a.is_reduced()
False
sage: b = a.reduced_form(); b
5*x^2 - x*y + 27*y^2
sage: b.is_reduced()
True

sage: a = BinaryQF([15,0,15])
sage: a.is_reduced()
True
sage: b = a.reduced_form(); b
15*x^2 + 15*y^2
sage: b.is_reduced()
True
```

sage.quadratic_forms.binary_qf.**BinaryQF_reduced_representatives**(*D*, *primitive_only=False*)

Returns a list of inequivalent reduced representatives for the equivalence classes of positive definite binary forms of discriminant D.

INPUT:

- *D* – (integer) A negative discriminant.
- `primitive_only` – (bool, default False) flag controlling whether only primitive forms are included.

OUTPUT:

(list) A lexicographically-ordered list of inequivalent reduced representatives for the equivalence classes of positive definite binary 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*
```

The number of inequivalent reduced binary forms with a fixed negative fundamental discriminant D is the class number of the quadratic field $Q(\sqrt{D})$:

```
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))
689
sage: QuadraticField(-p, 'a').class_number()
689

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)
[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]
```

TESTS:

```
sage: BinaryQF_reduced_representatives(5)
Traceback (most recent call last):
...
ValueError: discriminant must be negative and congruent to 0 or 1 modulo 4
```

CHAPTER

THREE

SOME EXTRAS

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
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

CREATING 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: Q = random_quadraticform_with_conditions(ZZ, 3, [QuadraticForm.is_positive_definite], [-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: Q = random_ternaryqf_with_conditions([TernaryQF.is_positive_definite], [-5, 5])
sage: Q    ## RANDOM
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 range(-163, -3, 4) if is_fundamental_discriminant(d)]
```

In general, we have $B_{1,\chi_d} = -2h/w$ for odd negative fundamental discriminants:

```
sage: all(QuadraticBernoulliNumber(1, d) == -len(BinaryQF_reduced_representatives(d)) for d in t
True
```

REFERENCES:

•[Iwasawa], 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)
1

sage: gamma__exact(1/2)
sqrt(pi)
sage: gamma__exact(3/2)
```

```
1/2*sqrt(pi)
sage: gamma__exact(5/2)
3/4*sqrt(pi)
sage: gamma__exact(7/2)
15/8*sqrt(pi)

sage: gamma__exact(-1/2)
-2*sqrt(pi)
sage: gamma__exact(-3/2)
4/3*sqrt(pi)
sage: gamma__exact(-5/2)
-8/15*sqrt(pi)
sage: gamma__exact(-7/2)
16/105*sqrt(pi)
```

TESTS:

```
sage: gamma__exact(1/3)
Traceback (most recent call last):
...
TypeError: you must give an integer or half-integer argument
```

sage.quadratic_forms.special_values.**quadratic_L_function__exact**(n, d)

Returns 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)
1/4*pi
sage: quadratic_L_function__exact(-4, -4)
5/2
```

TESTS:

```
sage: quadratic_L_function__exact(2, -4)
Traceback (most recent call last):
...
TypeError: n must be a critical value (i.e. odd > 0 or even <= 0)
```

REFERENCES:

- [Iwasawa], pp 16-17, Special values of $L(1-n, \chi)$ and $L(n, \chi)$
- [IreRos]
- [WashCyc]

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(" + str(1+2*i) + ", (-4/.)): ", RR(quadratic_L_function__exact(1+2*i, -4)) -
L(1, (-4/.)):  0.00004999999950000024999996962707
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
....:         print "Oops!  We have a problem at d = ", d, "    exact = ", RR(quadratic_L_functi
```

sage.quadratic_forms.special_values.**zeta__exact**(*n*)

Returns 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 [Iwasawa], 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(" + str(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
```

TESTS:

```
sage: zeta__exact(5)
Traceback (most recent call last):
...
TypeError: n must be a critical value (i.e. even > 0 or odd < 0)
```

REFERENCES:

CHAPTER

SIX

OPTIMISED CYTHON CODE FOR COUNTING CONGRUENCE SOLUTIONS

Optimised Cython code for counting 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_c` `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*)

Returns the number of solutions of Q(x) = m over the finite field Z/pZ, where p is a prime number > 2 and Q is a non-degenerate quadratic form of dimension n >= 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)
[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 (moc
9
sage: count_modp__by_gauss_sum(3, 3, 1, 2)     ## for Q = x^2 + y^2 + 2*z^2  => Gram Det = 2 (moc
12
sage: count_modp__by_gauss_sum(3, 3, 2, 2)     ## for Q = x^2 + y^2 + 2*z^2  => Gram Det = 2 (moc
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)
[True, True, True]
```

sage.quadratic_forms.count_local_2.**extract_sublist_indices**(*Biglist*, *Smalllist*)

Returns the indices of Biglist which index the entries of Smalllist appearing in Biglist. (Note that Smalllist may not be a sublist of Biglist.)

NOTE 1: This is an internal routine which deals with re-indexing lists, and is not exported to the QuadraticForm namespace!

NOTE 2: This should really by applied only when BigList has no repeated entries.

TO DO: *** Please revisit this routine, and eliminate it! ***

INPUT: Biglist, Smalllist – two lists of a common type, where Biglist has no repeated entries.

OUTPUT: a list of integers >= 0

EXAMPLES:

```
sage: from sage.quadratic_forms.count_local_2 import extract_sublist_indices

sage: biglist = [1,3,5,7,8,2,4]
sage: sublist = [5,3,2]
sage: sublist == [biglist[i]  for i in extract_sublist_indices(biglist, sublist)]  ## Ok wheneve
True

sage: extract_sublist_indices([1,2,3,6,9,11], [1,3,2,9])
[0, 2, 1, 4]
```

```
sage: extract_sublist_indices([1,2,3,6,9,11], [1,3,10,2,9,0])
[0, 2, 1, 4]

sage: extract_sublist_indices([1,3,5,3,8], [1,5])
Traceback (most recent call last):
...
TypeError: Biglist must not have repeated entries!
```

CHAPTER

SEVEN

INDICES AND TABLES

- Index
- Module Index
- Search Page

BIBLIOGRAPHY

[Iwasawa] Iwasawa, *Lectures on p-adic L-functions*

[IreRos] Ireland and Rosen, *A Classical Introduction to Modern Number Theory*

[WashCyc] Washington, *Cyclotomic Fields*

PYTHON MODULE INDEX

q

INDEX

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

V

W

X

Z