CQF VERSION 0.5
MANUAL

PRZEMYSŁAW KOPROWSKI

CQF is a Magma [2] package for doing computations in algebraic theory of quadratic forms. The package can be downloaded from the author's webpage at `http://www.pkoprowski.eu/cqf`. The package is released under standard MIT license. The full statement of the license is given at the end of this manual.

The package was developed on version 2.26 of Magma. The author has not tested it on other versions (yet), but will appropriate any reports on how it works in different setups. Questions and comments should be sent via email to `przemyslaw.koprowski@us.edu.pl`.

1. Quick start guide

We present here a quick glimpse of the use of the package. Full description of the functions used in this tutorial can be found in the following sections. We assume that the reader is at least mildly familiar with Magma. Otherwise we suggest to consult [4] prior to reading this manual.

First we need to load the package (see Section 2):

```
> AttachSpec("path_to_cqf/cqf.spec");
```

Next, we fix a field we will be working with. Let's take $K = \mathbb{Q}(\sqrt{37})$.

```
> K<sqrt37> := QuadraticField(37);
```

Define a quadratic form $q = \langle 3+2\sqrt{37}, -7+\sqrt{37}, 31-5\sqrt{37}, 514-54\sqrt{37}\rangle$ over K (see Section 4.1):

```
> q := DiagonalQuadraticForm(K,
>      [3+2*sqrt37, -7+sqrt37, 31-5*sqrt37, 514-54*sqrt37]
> ); q;
(2*sqrt37 + 3, sqrt37 - 7, -5*sqrt37 + 31, -54*sqrt37 + 514)
```

and check whether the form is isotropic (see Section 4.3):

```
> IsIsotropic(q);
true
```

Indeed it is. We may find its isotropic vector

```
> v := IsotropicVector(q); v;
(1/7*(-3*sqrt37 - 16) 1/14*(-23*sqrt37 - 139) 1/11398*(1121*sqrt37 + 5365)
    1/22796*(-81*sqrt37 - 469))
```

Check that v is really an isotropic vector. This means that q vanishes at v:

```
> Evaluate(q, v);
0
```

Let's see if q happens to be hyperbolic:

Date: December 31, 2021.

```
> IsHyperbolic(q);
false
```

The isotropic form of dimension 4 that is not hyperbolic must have Witt index equal 1:

```
> WittIndex(q);
1
```

To round this tutorial off we will determine the Witt class of K. It is known (see e.g. [12, Section 20.2]) that there are precisely seven Witt classes of quadratic fields represented by $\mathbb{Q}(\sqrt{d})$ for $d \in \{-1, \pm 2, \pm 7, \pm 17\}$. The following code finds the one which is Witt equivalent to K:

```
> reps := { QuadraticField(j) : j in [-1,2,-2,7,-7,17,-17] };
> for L in reps do
>   if AreWittEquivalent(K, L) then
>     print "K is Witt equivalent to", L;
>     break;
>   end if;
> end for;
K is Witt equivalent to Quadratic Field with defining polynomial
$.1^2 - 2 over the Rational Field
```

It turn out that K is in the same Witt class as $\mathbb{Q}(\sqrt{2})$. See Section 5 for a description of `AreWittEquivalent`.

2. Package installation and loading

The CQF package can be downloaded for free from the author's web page

`http://www.pkoprowski.eu/cqf`

The package comes in form of a zip archive that must be extracted an put on disc in a place where Magma is able to find it. To test the installation go to the directory where CQF resides and execute

```
magma tests.magma
```

If it prints "All tests passed" this means that the installation is correct.

The CQF package is loaded into Magma by attaching its spec file:

```
> AttachSpec("path_to_cqf/cqf.spec");
```

One can also add this command to Magma startup file (specified by the environment variable `MAGMA_USER_SPEC`) to auto-load the package when Magma starts. For details about Magma startup environment consult [3, Sections 2.3.9 and 4.3].

3. Sums of squares

A sum of squares of elements of a given field is a frequent object of research in quadratic form theory. CQF provides the following functions for working with sums of squares.

`IsSumOfSquares(a)`

`IsSOS(a)`

Checks if a is a sum of squares in its parent field.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`LengthOfSumOfSquares(a)`

`LengthOfSOS(a)`

Given a nonzero element a of a field K, this function determines the minimal number of summands needed to express a as a sum of squares. It returns 0 when a is not a sum of squares in K.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Check that 7 is a sum of four squares in $\mathbb{Q}$:

```
> IsSOS(7);
true
> LengthOfSOS(7);
4
```

Example. In the next example we take a number field $K = \mathbb{Q}(\theta)$, where θ is a root of the polynomial $x^4 - 5x^2 + 25$. We then show that $\theta + 1$ is a sum of three squares in K.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(x^4 - 5*x^2 + 25);
> IsSumOfSquares(theta + 1);
true
> LengthOfSumOfSquares(theta + 1);
3
```

`SolveSumOfSquares(a)`

`SolveSOS(a)`

This function decomposes totally positive elements into sums of squares of minimal lengths. That is, given an element a of a field K, which is a sum of squares of length n, this function constructs $c_1, \ldots, c_n \in K$ such that $c_1^2 + \cdots + c_n^2 = K$.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Write 7 as a sum of four squares in $\mathbb{Q}$:

```
> C := SolveSOS(7); C;
[ 1, -2, -1, 1 ]
> &+ [ c^2 : c in C ];
7
```

Example. A more interesting example concerns a sum of squares in a number field. We will take again the same element θ that we used with the command `LengthOfSumOfSquares`.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(x^4 - 5*x^2 + 25);
> C := SolveSumOfSquares(theta + 1); C;
[
    1/2*(theta + 2),
    -1/4*theta,
    1/4*(-theta^2 + 5)
```

```
]
> &+ [ c^2 : c in C ];
theta + 1
```

`Level(K)`

The *level* of a field is by definition (see e.g. [9, Definition XI.2.1]) the length of -1. Hence `Level(K)` is equivalent to `LengthOfSOS(K!-1)`. In particular, if K is formally real, then `Level(K)` returns zero.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`IsFormallyReal(K)`

`IsReal(K)`

A field K is called *formally real* (or shortly *real*), when -1 is not a sum of squares in K. Hence this function is equivalent to `IsSOS(K!-1)`.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`PythagorasNumber(K)`

The *Pythagoras Number* $P(K)$ of a field K is by definition (see [9, Definition XI.5.5]) the supremum of the lengths of all nonzero sums of squares in K. Hence, Pythagoras number equal $n \in \mathbb{N}$ means that every sum of squares in K is in fact a sum of n squares.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`PythagorasElement(K)`

This function returns a sum of squares of maximal length in K. Hence it returns an element $a \in K$ such that the length of a equals the Pythagoras number of K.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Take a number field $K = \mathbb{Q}(\theta)$, where θ is a root of the polynomial $x^4 - 5x^2 + 25$. We show that the $P(K) = 3$ and that $\theta + 1$ is an element of the maximal length.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(x^4 - 5*x^2 + 25);
> PythagorasNumber(K);
3
> a := PythagorasElement(K); a;
theta + 1
> LengthOfSOS(a);
3
```

4. Quadratic forms

Magma has a built-in type `ModTupFld` for representing quadratic spaces. CQF introduced an additional category `DiagQuadFrm` that represents *diagonal* quadratic forms. Objects of this type can be converted to quadratic spaces or multivariate polynomials. Conversely, any quadratic space over a field of odd characteristic can be diagonalized and converted into a diagonal quadratic form.

4.1. **Construction.**

`DiagonalQuadraticForm(K, [a1,...,an])`

`DiagonalQuadraticForm([a1,...,an])`

Constructs a quadratic form $\langle a_1, \ldots, a_n \rangle = a_1 x_1^2 + \cdots + a_n x_n^2$ over a field K. If the field is not specified, it is assumed to be the universe of the list of coefficients.

Example. Construct a form $\langle 1, 2 - 5 \rangle$ over the rationals:

```
> DiagonalQuadraticForm( Rationals(), [1,2,-5] );
(1, 2, -5)
```

`HyperbolicForm(K, n)`

Construct a hyperbolic form over K of dimension n, i.e. an orthogonal sum of $n/2$ copies of $\langle 1, -1 \rangle$. The dimension must be even, otherwise an error is thrown.

`PfisterForm(K, [a1,...,an])`

`PfisterForm([a1,...,an])`

Construct an n-fold Pfister form $\langle\langle a_1, \ldots, a_n \rangle\rangle = \langle 1, a_1 \rangle \otimes \cdots \otimes \langle 1, a_n \rangle$. If the base field K is not specified, it is assumed to be the universe of the list of coefficients. For more information on Pfister forms see [9, Chapter X] or [12, Chapter 17].

Example. Construct a two-fold Pfister form $\langle\langle 2, 3 \rangle\rangle = \langle 1, 2, 3, 6 \rangle$ over $\mathbb{Q}(i)$:

```
> PfisterForm( QuadraticField(-1), [2,3] );
(1, 2, 3, 6)
```

`Subform(q, I)`

Given a quadratic form $q = \langle a_1, \ldots, a_n \rangle$ and a sequence of indices $I = [i_1, \ldots, i_k] \subseteq [1, \ldots, n]$, this function constructs a new form q', whose coefficients are $a_{i_1}, \ldots, a_{i_k}$.

Example. Extract a subform $q' = \langle 2, 8, 256 \rangle$ out of $q = \langle 2, 4, 8, 16, 32, 64, 128, 256 \rangle$:

```
> q := DiagonalQuadraticForm(Integers(), [ 2^i : i in [1..8] ]); q;
(2, 4, 8, 16, 32, 64, 128, 256)
> q_ := Subform(q, [1,3,8]); q_;
(2, 8, 256)
```

The indices in I may be repeated, hence despite the function's name, q' is not necessarily sensu stricte a *sub*-form of q.

```
> q_ := Subform(q, [1,3,1]); q_;
(2, 8, 2)
```

`RandomDiagonalQuadraticForm(K, n)`

`RandomDQF(K, n)`

Construct a non-degenerate form of dimension n over K with random coefficients. The construction of random elements in infinite fields is very simplistic, with no guarantee about their distribution. This function is mostly intended for construction of examples rather than any kind of Monte Carlo or Las Vegas algorithms.

4.2. **Conversion.**

`Coefficients(q)`

`ElementToSequence(q)`

`Eltseq(q)`

The sequence of the coefficients of the diagonal quadratic form q.

`QuadraticFormPolynomial(q)`

Converts a diagonal quadratic form $\langle a_1, \ldots, a_n \rangle$ into a multivariate quadratic polynomial $a_1 x_1^2 + \cdots + a_n x_n^2$.

Example. Convert a form $q = \langle 1, 2, 2 \rangle$ over $\mathbb{F}_{11}$ into a polynomial $x_1^2 + 2x_2^2 + 2x_3^2$:

```
> q := DiagonalQuadraticForm( GF(11), [1,2,2] ); q;
(1,2,2)
> QuadraticFormPolynomial(q);
$.1^2 + 2*$.2^2 + 2*$.3^2
> P<x1,x2,x3> := PolynomialRing(GF(11), 3);
> P ! QuadraticFormPolynomial(q);
x1^2 + 2*x2^2 + 2*x3^2
```

`QuadraticFormMatrix(q)`

Convert a diagonal quadratic form into a diagonal matrix representing this form.

Example. We use the same form as in the previous example.

```
> q := DiagonalQuadraticForm( GF(11), [1,2,2] ); q;
(1,2,2)
> QuadraticFormMatrix(q);
[ 1  0  0]
[ 0  2  0]
[ 0  0  2]
```

`QuadraticSpace(q)`

Converts a diagonal quadratic form into a quadratic space (Magma's object of type `ModTupFld`).

`Diagonalization(V)`

Given a quadratic space (V, q) over a field of odd characteristic, this function constructs a diagonal quadratic form isometric with (V, q).

Example. Construct a quadratic space $V = (\mathbb{F}_{53}^3, q)$, where q is a *non-diagonal* quadratic form $x^2 + xy + 3xz - 2yz + y^2 + z^2$. Diagonalize it to get a *diagonal* quadratic form $q' = \langle 1, 14, 30 \rangle$. Then check that $V' = (\mathbb{F}_{53}^3, q')$ is indeed isometric to V.

```
> P<x,y,z> := PolynomialRing(GF(53),3);
> q := x^2 + x*y +3*x*z - 2*y*z + y^2 +z^2;
> V := QuadraticSpace(q);
> q_ := Diagonalization(V); q_;
(1, 14, 30)
> V_ := QuadraticSpace(q_);
> IsIsometric(V, V_);
true Mapping from: ModTupFld: V to ModTupFld: V_ given by a rule
```

4.3. **Invariants.**

`BaseRing(q)`

`BaseField(q)`

Returns the base field of the diagonal quadratic form q.

`Dimension(q)`

The dimension of the diagonal quadratic form q.

`Determinant(q)`

The *determinant* of the diagonal quadratic form, i.e. the product of its coefficients.

`Discriminant(q)`

The *discriminant* (a.k.a. *signed determinant*) of the quadratic form. By definition (see e.g. [9, p. 30])

$$\mathrm{disc}\langle a_1, \ldots, a_n \rangle := (-1)^{\frac{n\cdot(n-1)}{2}} \cdot a_1 \cdots a_n.$$

`IsNondegenerate(q)`

`IsNondegenerate(V)`

A quadratic form is *degenerate* if there is a nonzero vector orthogonal to the whole space iff the discriminant of the form is zero. This function checks whether a given quadratic forms is not degenerate. The argument can be either a diagonal quadratic form or a quadratic space (not necessarily diagonal).

`IsDegenerate(q)`

`IsDegenerate(V)`

A quadratic form is *degenerate* if there is a nonzero vector orthogonal to the whole space iff the discriminant of the form is zero. This function checks whether a given quadratic forms is degenerate. The argument can be either a diagonal quadratic form or a quadratic space (not necessarily diagonal). It is just a shortcut for `not IsNondegenerate(q)`.

`IsIsotropic(q)`

`IsIsotropic(V)`

A non-degenerate quadratic space (V, q) is called *isotropic* if there is a self-orthogonal vector $v \in V$ i.e. a vector such that $q(v) = 0$. This function checks if the given quadratic form is isotropic. An argument can be either a diagonal quadratic form (`DiagQuadFrm`) or a quadratic space (`ModTupFld`). It generalizes a built-in Magma's function of the same name, that works over the rationals and finite fields only.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Take a form $\langle 2, 3\theta, 3\theta, 1, 1\rangle$ over a number field $K := \mathbb{Q}(\theta)$, where θ is a root of $2x^4 + 3x^3 + x^2 + 3x + 1$. We show that it is isotropic.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(2*x^4 + 3*x^3 + x^2 + 3*x + 1);
> q := DiagonalQuadraticForm(K, [2, 3*theta, 3*theta, 1, 1]); q;
(2, 3*theta, 3*theta, 1, 1)
> IsIsotropic( q );
true
```

`IsAnisotropic(q)`

`IsAnisotropic(V)`

This function is just a shortcut `not IsIsotropic(q)`. Provided only for convenience, to let the user write a cleaner code.

`IsIsotropic(q, P)`

`IsIsotropic(V, P)`

Let (V, q) be a quadratic space over a global field K and P be a place of K. This function checks whether the localization of (V, q) at P (that is the quadratic space $(V \otimes K_P, q)$) is isotropic. The argument P can be either a place of K or the associated prime ideal.

Supports: rationals, number fields, global rational function fields, global function fields

Example. Take a form $\langle 2, 3\theta-1, 3\theta+1, 1, -9\theta^2+3\theta\rangle$ over a number field $K := \mathbb{Q}(\theta)$, where θ is a root of $x^8 - x^6 - x^5 - 3x^4 + 4x^2 - 1$ (denoted 8.4.15908237.1 in [10]). We show that it is isotropic over the unique prime of K laying over 3.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(x^8 - x^6 - x^5 - 3*x^4 + 4*x^2 - 1);
> q := DiagonalQuadraticForm(K, [2, 3*theta - 1, 3*theta + 1, 1,
>     -9*theta^2 + 3*theta]); q;
(2, 3*theta - 1, 3*theta + 1, 1, -9*theta^2 + 3*theta)
> P := Decomposition(K, 3)[1][1]; P;
Place at Principal Prime Ideal
Generator:
    [3, 0, 0, 0, 0, 0, 0, 0]
> IsIsotropic( q, P );
true
```

`IsHyperbolic(q)`

`IsHyperbolic(V)`

A quadratic space over a field of characteristic $\neq 2$ is called *hyperbolic* if it decomposes into an orthogonal sum of hyperbolic planes (a *hyperbolic plane* is a 2-dimension isotropic space, it is isometric to $\langle 1, -1\rangle$). Equivalently a quadratic space is hyperbolic if it contains a subspace equal to its orthogonal complement. This function checks if the given quadratic form is hyperbolic. The argument can be either a diagonal quadratic form (`DiagQuadFrm`) or a quadratic space (`ModTupFld`). It generalizes a built-in Magma's function of the same name, that works over the rationals and finite fields only.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Take the same form $\langle 2, 3\theta, 3\theta, 1, 1\rangle$ as in the previous example. We show that it is not hyperbolic.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(2*x^4 + 3*x^3 + x^2 + 3*x + 1);
> q := DiagonalQuadraticForm(K, [2, 3*theta, 3*theta, 1, 1]);
> IsHyperbolic( q );
false
```

`IsHyperbolic(q, P)`

`IsHyperbolic(V, P)`

Let (V, q) be a quadratic space over a global field K and P be a place of K. This function checks whether the localization of (V, q) at P (that is the quadratic space $(V \otimes K_P, q)$) is hyperbolic. The argument P can be either a place of K or the associated prime ideal.

Supports: rationals, number fields, global rational function fields, global function fields

Example. Take a quadratic form q over $K = \mathbb{Q}(\sqrt{7})$:

```
> K<sqrt7> := QuadraticField(7);
> q := DiagonalQuadraticForm(K, [2*sqrt7, 2*sqrt7, 1-sqrt7, 1-sqrt7]);
```

Take the unique (ramified) place P of K that divides 2 and check that the localization $q \otimes K_P$ is hyperbolic:

```
> P := Decomposition(K, 2)[1][1];
> IsHyperbolic(q, P);
true
```

Supports: rationals, number fields, global rational function fields, global function fields

`WittIndex(q)`

`WittIndex(V)`

`AnisotropicDimension(q)`

`AnisotropicDimension(V)`

Witt decomposition theorem states that every non-degenerate quadratic space (V, q) decomposes into an orthogonal sum of a hyperbolic and anisotropic subspaces.

The dimensions of both these summands is an invariant of the quadratic space and is independent of the actual decomposition. The function `AnisotropicDimension` returns the dimension of the anisotropic part. On the other hand, the number of hyperbolic planes contained in the given quadratic space (i.e. half of the dimension of the hyperbolic part) is called the *Witt index* (see e.g. [9, Definition I.4.3]) of the quadratic space and denoted $\operatorname{ind} V$. The anisotropic dimension and the Witt index are connected by the formula

$$\operatorname{ind} V = \tfrac{1}{2} \cdot (\dim V - \dim V_a),$$

where V_a is the anisotropic part of V. Both `WittIndex` and `AnisotropicDimension` can be used with either diagonal quadratic forms (`DiagQuadFrm`) or quadratic spaces (`ModTupFld`).

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Once again we use consider the same quadratic form $\langle 2, 3\theta, 3\theta, 1, 1\rangle$ as in the previous two examples. We know it is not hyperbolic but it is isotropic. It contains a single hyperbolic plane, so its Witt index is 1 and consequently the dimension of its anisotropic part is 3.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(2*x^4 + 3*x^3 + x^2 + 3*x + 1);
> q := DiagonalQuadraticForm(K, [2, 3*theta, 3*theta, 1, 1]);
> WittIndex( q );
1
> AnisotropicDimension( q );
3
```

`AnisotropicDimension(q, P)`

This function computes the dimension of the anisotropic part (see the previous command for explanation) of the localization $q \otimes K_P$ of the diagonal quadratic form q at the prime P. Here, P can be either a place or a prime ideal.

Supports: rationals, number fields, global rational function fields, global function fields

Example. Take a quadratic form q over $K = \mathbb{Q}(\sqrt{7})$ and check that the form is anisotropic (see function `IsIsotropic` above):

```
> K<sqrt7> := QuadraticField(7);
> q := DiagonalQuadraticForm(K, [2*sqrt7, 2*sqrt7, 1-sqrt7, 1-sqrt7]);
> IsIsotropic(q);
false
```

Take the unique (ramified) place P of K that divides 2 and compute the dimension of the anisotropic part of the localization $q \otimes K_P$:

```
> P := Decomposition(K, 2)[1][1];
> AnisotropicDimension(q, P);
0
```

The result is zero, hence the localization must be hyperbolic. Indeed it is (see function `IsHyperbolic` above):

```
> IsHyperbolic(q, P);
true
```

`AnisotropicPart( q )`

Let q be a diagonal quadratic form. Witt decomposition theorem states that it decomposes into an orthogonal sum of a hyperbolic and anisotropic subspaces:

$$q \cong q_a \perp w \times \langle 1, -1 \rangle.$$

The form q_a is unique up to isometry and is called an *anisotropic part* of q. This function construct the anisotropic part of a given form.

Example. Again (see examples for `IsIsotropic`, `IsHyperbolic` and `AnisotropicDimension`) take the form $\langle 2, 3\theta, 3\theta, 1, 1 \rangle$ over the number field $K := \mathbb{Q}(\theta)$, where θ is a root of $2x^4 + 3x^3 + x^2 + 3x + 1$. From the previous examples we know that its is isotropic of anisotropic dimension 3.

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(2*x^4 + 3*x^3 + x^2 + 3*x + 1);
> q := DiagonalQuadraticForm(K, [2, 3*theta, 3*theta, 1, 1]); q;
(2, 3*theta, 3*theta, 1, 1)
> AnisotropicDimension(q);
3
```

We will compute its anisotropic part:

```
> qa := AnisotropicPart(q); qa;
(1, 2*theta^3 + 3*theta^2 - theta + 1, 36*theta^3 + 36*theta^2 + 18*theta)
```

Let us see that this form is indeed anisotropic and the two forms are similar (the description of `AreSimilar` below):

```
> IsIsotropic(qa);
false
> AreSimilar(q, qa);
true
```

Warning: Over global fields, this function can take a considerable time to evaluate. Especially when the places that divide the coefficients of q have large degrees.

`AreIsometric(q1, q2)`

`AreIsometric(q1, V2)`

`AreIsometric(V1, q2)`

`AreIsometric(V1, V2)`

This function checks if two quadratic spaces (V_1, q_1), (V_2, q_2) are isometric, i.e. if there is a linear isomorphism $\varphi : V_1 \to V_2$ such that $q_2\big(\varphi(v)\big) = q_1(v)$ for every $v \in V_1$. The arguments can be any combination of diagonal quadratic forms (`DiagQuadFrm`) and quadratic spaces (`ModTupFld`).

The function is called `AreIsometric` rather than `IsIsometric` to avoid collision with the built-in function of the latter name, which works solely over finite fields but returns also the isometry φ.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`AreSimilar(q1, q2)`

`AreSimilar(q1, V2)`

`AreSimilar(V1, q2)`

`AreSimilar(V1, V2)`

Two quadratic forms q_1, q_2 are *similar* if there are non-negative integers n_1, n_2 such that the forms $q_1 \perp n_1\langle 1, -1\rangle$ and $q_2 \perp n_2\langle 1, -1\rangle$ are isometric. Equivalently, two forms are similar if and only if their anisotropic parts are isometric. This function checks whether two quadratic spaces (V_1, q_1), (V_2, q_2) are similar. The arguments can be any combination of diagonal quadratic forms (`DiagQuadFrm`) and quadratic spaces (`ModTupFld`).

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`HasseMinkowskiInvariant(q, P)`

`WittInvariant(q,P)`

Let q be a diagonal quadratic form over a global field K of characteristic $\neq 2$. This function computes the Hasse invariant (see e.g. [9, Definition V.3.17]) of the localization $q \otimes K_P$ in the completion of K at P. Here P can be given either as a place or a prime ideal of K. This functions generalize built-in functions of the same names that work only over $\mathbb{Q}$.

Be aware that the name `WittInvariant` can be a bit misleading since it is **not** the Witt invariant in sens of [9, p. 117]. The relation between these two notions is explained in [9, Proposition V.3.20]. We use the above name for consistency with the built-in function.

Supports: rationals, number fields, global rational function fields, global function fields

4.4. **Other functions.**

`Evaluate(q, v)`

Evaluates the diagonal quadratic form q at v. So, if $q = \langle a_1, \ldots, a_n\rangle$ and $v = (v_1, \ldots, v_n)$, this function returns $a_1v_1^2 + \cdots + a_nv_n^2$. The argument v can be either a vector or a sequence.

`IsotropicVector(q)`

`algorithm` MonStgElt *Default:* `"v2"`

Given an isotropic quadratic form this function finds its isotropic vector, i.e. a vector v such that $q(v)$ is zero. For forms over the rationals, it uses the built-in function `IsotropicSubspace` (based on Simon's algorithm, see [11]). The optional parameter `"algorithm"` is meaningful only for forms of dimension ≥ 4 over global fields. It may take only one of the two values: either `"v1"` (then the function uses the algorithms described in [7]) or `"v2"`. The two algorithm can radically differ in computing time and size of the output (recall that an isotropic vector is not unique). In both directions (see the example below). Hence if one of the algorithm is too time-consuming, one may try the other one.

Example. Denote $t := \sqrt{-65}$ and consider the quadratic number field $K = \mathbb{Q}(t)$:

```
> K<t> := QuadraticField(-65);
```

Take the quadratic form $q := \langle -6t + 1, 10t - 8, 3t - 5, -10t + 4, 8, 6t - 10\rangle$:

```
> q1 := DiagonalQuadraticForm(K, [
>      -6*t + 1, 10*t - 8, 3*t - 5, -10*t + 4, 8, 6*t - 10
> ]); q1;
(-6*t + 1, 10*t - 8, 3*t - 5, -10*t + 4, 8, 6*t - 10)
```

We shall construct a copy of it, because the constructed isotropic vectors are cached between successive execution of the function:

```
> q2 := Subform(q1, [1..6]); q2;
(-6*t + 1, 10*t - 8, 3*t - 5, -10*t + 4, 8, 6*t - 10)
```

We will Construct an isotropic vector for q_1. To this end we will use *both* available algorithms and compare their running times and output sizes. First, let us try algorithm v1:

```
> time v := IsotropicVector(q1 : algorithm := "v1");
Time: 0.920
> v;
(0 -95964*t - 538932 0 1/3*(200863*t - 470183) 1/3914*(298982167*t +
    13766405641) 1/1957*(171354439*t + 588406159))
```

Thus, v above is an isotropic vector. We may verify it using the command `Evaluate`, described earlier:

```
> Evaluate(q1, v);
0
```

To streamline the comparison of output size, compute the number of characters of the output form of v:

```
> s := Sprintf("%o", v); #s;
115
```

Now, we try the other algorithm. This time we will not print the obtained isotropic vector. Instead we will only verify the correctness and check the size of the output.

```
> time w := IsotropicVector(q2 : algorithm := "v2");
Time: 0.260
> Evaluate(q2, w);
0
> s := Sprintf("%o", w); #s;
1890
```

As we can see, for this particular quadratic form, the second algorithm turned out to be nearly four times faster, but its output is over sixteen times bigger.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`OrthogonalSum(q1, q2)`

`q1 + q2`

The orthogonal sum $q_1 \perp q_2$ of two diagonal quadratic forms q_1, q_2 over the same field.

`q1 - q2`

The "orthogonal difference" i.e. a sum $q_1 \perp (-q_2)$ of two diagonal quadratic forms q_1, q_2 over the same field.

`TensorProduct(q1, q2)`

The tensor product $q_1 \otimes q_2$ of two diagonal quadratic forms q_1, q_2 over the same field.

Example. Let us compute the tensor product

$$\langle 1,2,3\rangle \otimes \langle 1,-1\rangle = \langle 1,2,3,-1,-2,-3\rangle.$$

```
> q1 := DiagonalQuadraticForm( Rationals(), [1, 2, 3]); q1;
(1, 2, 3)
> q2 := HyperbolicForm( Rationals(), 2 ); q2;
(1, -1)
> TensorProduct(q1, q2);
(1, 2, 3, -1, -2, -3)
```

`RandomShuffle( q )`

Two forms $q = \langle a_1, \ldots, a_n\rangle$ and $q' = \langle a'_1, \ldots, a'_n\rangle$ are called 2-*connected* if for some $1 \leq i \neq j \leq n$ there is an isometry $\langle a_i, a_j\rangle \cong \langle a'_i, a'_j\rangle$. Two forms q, q' are 2-connected *in* n *steps* if there are form $q = q_1, q_2, \ldots, q_n = q'$ such that q_i and q_{i+1} are 2-connected for every $i < n$.

Given a diagonal quadratic form q, thsi function constructs a new form q', which is 2-connected to q in n steps, where n is an optional parameter. By default $n = 3$. In every step this function changes of two random coefficients a and b using the isometry

$$\langle a+b, (a+b)\cdot a\cdot b\rangle.$$

Example. We will construct a form that is 2-connected in 2 steps to $\langle 1,2,3,4,5,6\rangle$:

```
> q := DiagonalQuadraticForm( Rationals(), [1..6] ); q;
(1, 2, 3, 4, 5, 6)
> RandomShuffle(q : n := 2);
(1, 2, 120, 4, 14, 672)
```

5. Witt ring and Witt equivalence

Recall that two non-degenerate quadratic forms q_1, q_2 are said to be *similar* if there are non-negative integers m, n such that $q_1 \perp nH$ is isometric to $q_2 \perp nH$, where H stands for a hyperbolic plane. The set WK of similarity classes of non-degenerate forms posses a ring structure, were addition and multiplication are induced by orthogonal sum and tensor product, respectively. The ring WK is called the *Witt ring* of K. CQF provides two new data-types `RngWittQF` and `RngWittQFElt` that represent a Witt ring of a field and a similarity class (i.e. element of the Witt ring), respectively.

5.1. **Ring construction.**

`WittRing(K)`

This function constructs the Witt ring WK of the field K. Although K can be *any* field that Magma is able to work with, for most fields very little can be done with the resulting object. Full functionality is limited to the fields supported by CQF, these are: finite field, rationals, number fields, global function fields, reals and the field of complex numbers.

Warning: There is a built-in function of the same name `WittRing(K, n)` that constructs the ring of *Witt vectors* of length n over K. This is a completely different object! Be careful not to mistake these two commands. `WittRing` with just *one* argument creates the Witt ring of similarity classes of quadratic forms and with *two* arguments constructs the ring of Witt vectors.

Example. Let's see that $W\mathbb{F}_7$ is isomorphic to $\mathbb{Z}_4$.

```
> W := WittRing(GF(7));
> print W : Maximal;
Witt ring of Finite field of size 7, isomorphic to Residue class ring
of integers modulo 4
```

5.2. **Working with elements.**

`W ! q`
`W ! a`

Let W be the Witt ring of a field K. If q is a diagonal quadratic form over K, this operator construct the class $[q]$ of q in the Witt ring (i.e. the similarity class of q). In a an element of K, this operator constructs the Witt class of the 1-dimensional form $\langle a \rangle$.

`Zero(W)`

This function constructs the zero element of the Witt ring W, i.e. the class of hyperbolic forms.

`One(W)`

This function constructs the unit element of the Witt ring W, i.e. the class of the form $\langle 1 \rangle$. This function is equivalent to $W!1$.

`IsZero(q)`

This function checks if q is the null element of the Witt ring.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`IsOne(q)`

This function checks if q is the unit element of the Witt ring.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`q1 + q2`
`q1 - q2`
`q1 * q2`
`-q1`
`n * q1`

Standard ring arithmetic operators.

Warning: The result is not simplified. Hence the same Witt class constructed by two different sequences of operators can have different representations! Use the operators `eq` and `ne` to check equality.

`q1 eq q2`
`q1 ne q2`

Standard comparison operators.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Construct the Witt ring $W\mathbb{Q}$ of the field of rational numbers:

```
> QQ := Rationals();
> WQ := WittRing(QQ); WQ;
Witt ring of Rational Field
```

Take the Witt class of the form $\langle 1, 2, -7\rangle$ and add the class of $\langle 1\rangle$. Check that the result is not the null element of $W\mathbb{Q}$:

```
> q := DiagonalQuadraticForm(QQ, [1,2,-7]);
> _q := WQ ! q; _q;
[(1, 2, -7)]
> _q := _q - One(WQ); _q;
[(1, 2, -7, -1)]
> IsZero(_q);
false
```

Alternatively the last line can be

```
> _q eq Zero(WQ);
false
```

`IsTorsion(q)`

Returns true if and only if q is a torsion element of the Witt ring. For an example use, see the example following the command `Height` below.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

`Parent(q)`

Given an element $q \in W$ of a Witt ring, this function returns the Witt ring W.

`BaseField(q)`

`BaseRing(q)`

Given an element q of a Witt ring WK of a field K, this function returns K.

5.3. **Invariants.**

`#W`

Returns the cardinality of the Witt ring W.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. The Witt ring of $\mathbb{Q}$ is infinite, while the Witt ring of $\mathbb{C}$ has just two elements:

```
> WQ := WittRing(Rationals());
> #WQ;
Infinity
> WC := WittRing(ComplexField());
> #WC;
2
```

`BaseField(W)`
`BaseRing(W)`

Given a Witt ring WK of a field K, this function returns K.

`Height(K)`

This function computes the *height* of K, which by definition (see [9, Definition XI.5.4]) is the exponent of the torsion part of WK.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. The Witt ring $W\mathbb{R}$ of the reals is isomorphic to $\mathbb{Z}$, hence torsion-free. Consequently the height of $\mathbb{R}$ equals 1;

```
> Height( RealField() );
4
```

Example. The height of $\mathbb{Q}$ is 4 since $4 \cdot q = 0$ for every torsion element $[q] \in W\mathbb{Q}$ while for $n < 4$ the form $n \cdot \langle 1, -7\rangle$ is not hyperbolic. To see this construct the Witt ring $W\mathbb{Q}$ of the rationals:

```
> QQ := Rationals();
> WQ := WittRing(QQ); WQ;
Witt ring of Rational Field
```

Now, construct the Witt class of $\langle 1, -7\rangle$:

```
> _q := WQ ! DiagonalQuadraticForm(QQ, [1,-7]);
> _q;
[(1, -7)]
```

See that it is a torsion element:

```
> IsTorsion(_q);
true
```

Compute the height h of $\mathbb{Q}$ and see that $h \cdot [\langle 1, -7\rangle]$ is the null element of the Witt ring:

```
> h := Height(QQ); h;
4
> IsZero(h*_q);
true
```

Finally, let's see that $k \cdot [\langle 1, -7\rangle]$ is nonzero for $0 < k < h$ (for explanation of the command `exists` see [3, Section 9.7]):

```
> exists{ k : k in [1..h-1] | IsZero(k*_q) };
false
```

5.4. **Witt equivalence.**

`AreWittEquivalent(K, L)`

Two fields K, L are *Witt equivalent* if their Witt rings are isomorphic. This function checks whether two fields are Witt equivalent.

Supports: finite fields, rationals, number fields, global rational function fields, global function fields, reals, complex numbers

Example. Let's see that $\mathbb{C}$ and $\mathbb{F}_2$ are Witt equivalent. In the Witt rings of both fields are isomorphic to $\mathbb{Z}_2$:

```
> AreWittEquivalent( ComplexField(), GF(2) );
true
> WittRing( ComplexField() ) : Maximal;
Witt ring of Complex field of precision 30, isomorphic to Residue
class ring of integers modulo 2
> WittRing( GF(2) ) : Maximal;
Witt ring of Finite field of size 2, isomorphic to Residue class
ring of integers modulo 2
```

Example. We will now consider a more complicated example. Take two quartic fields $K = \mathbb{Q}(\theta)$ where $\theta^4 - 2\theta^3 - 5\theta^2 + 6\theta - 8 = 0$ and $L = \mathbb{Q}(\eta)$ where $\eta^4 - 2\eta^3 - \eta^2 + 2\eta + 8 = 0$:

```
> _<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(x^4-2*x^3-5*x^2+6*x-8);
> L<eta> := NumberField(x^4-2*x^3-x^2+2*x+8);
```

Compute the heights of both fields:

```
> Height(K);
4
> Height(L);
8
```

It follows that the torsion parts of their Witt rings differ in size. In particular the fields cannot be Witt equivalent. Indeed, they are not:

```
> AreWittEquivalent(K, L);
false
```

6. Extensions to Magma's built-in functions

`HilbertSymbol(a, b, P)`

This function computes the Hilbert symbol $(a, b)_P$, where P is either a place or a prime ideal and a, b are two elements of the same global field K. It generalizes to global function fields a built-in function of the same name.

Supports: global rational function fields, global function fields

Example. Take a function field of an elliptic curve $y^2 - x^3 + 3x + 1 = 0$ over $\mathbb{F}_{257}$:

```
> F := GF(257);
> FX<X> := FunctionField(F);
> _<Y> := PolynomialRing(Fx);
> K<y> := FunctionField(y^2 - x^3 + 3*x + 1);
> x := K!X;
```

Take a random place P of degree 2 and compute the Hilbert symbol $(x, y)_P$:

```
> _, P := HasRandomPlace(K, 2);
> HilbertSymbol(x, y, P);
1
```

`IsLocalSquare(a, P)`

Given an element a of a global field K, this function checks whether a is a square in the completion K_P. This is the same as the built in command of the same name, but in addition it allows P to be:

- an ideal in $\mathbb{Z}$,
- a place of a global field (either number field or function field).

Supports: rationals, number fields, global rational function fields, global function fields

`LegendreSymbol(a, P)`

Let K be a global field, $a \in K$ its element and P a prime of K. This function computes the Legendre symbol $\left(\frac{a}{P}\right)$, i.e. it checks whether a is a quadratic residue modulo P. The function returns 0 if $a \in P$, -1 if a is not a quadratic residue, and 1 if A is a quadratic residue modulo P. The argument P can be either a place or a prime ideal of K.

Magma has a function of the same name that works over the rationals. CQF extends it to other global fields.

Supports: number fields, global rational function fields, global function fields

Example. Take a quadratic field $K = \mathbb{Q}\left(\sqrt{-37}\right)$. We will compute the Legendre symbol $\left(\frac{a}{P}\right)$, where $a = 2 + \sqrt{-37}$ and P is the unique dyadic place of K.

```
> K<theta> := QuadraticField(-37);
> P := Decomposition(K, 2)[1][1];
> LegendreSymbol(2+theta, P);
1
```

`LocalMultiplicativeGroupModSquares(P)`

Let K be a global function field and P its place. The local square class group $K_P^\times / K_P^{\times 2}$ may be treated as a vectors space V of dimension 2 over $\mathbb{F}_2$. This function returns V with a map $K^\times \to V$. This is an analog for function field of the built-in function of the same name that works with number fields.

Supports: global function fields

7. Other functions

In addition to the ones mentioned so far, CQF provides also some other functions that are not directly connected to quadratic forms theory. The functions listed below are used as subsidiary procedures for the core of CQF, yet they still may be of some independent interest to the user.

`Find( L, f )`

Let L be a sequence and f be a predicate (i.e. function returning true/false) defined on the universe of L. This function returns the index of the first element in L for which f returns `True`.

Example. Consider a sequence $[1, 3, 7, -5, 4, 2, -1, 6]$. We will find the index of the first even number in this sequence.

```
> L := [ 1, 3, 7, -5, 4, 2, -1, 6 ];
> print "j =", j, ", L[j] =", L[j];
j = 5 , L[j] = 4
```

`RealRootBound( f )`

`algorithm` MonStgElt *Default:* `"Hong"`

Let f be a polynomial with coefficients in some subring of the reals. This function computes an upper bound for the real roots of f. The function has an optional argument `algorithm`, that controls what algorithm is used for the computation. For now the possible values are:

- "Hong" (default): the function uses Hong's bound (see [6]).
- "local max": the function uses Vigklas' linear local-max bound (see [13]).

Example. Let us compute an upper bound for $f = x^4 - 200x^2 + 40x - 2$ using both methods. It can be checked by hand-computation that the Hong's bound for f equals $20\sqrt{2}$, while the local-max bound is precisely 20:

```
> R<x> := PolynomialRing(Rationals());
> RealRootBound(f);
28.284271247461900976033744842
> RealRootBound(f : algorithm := "local max");
20.000000000000000000000000000
```

`RealRootIntervals( f )`

Let f be a polynomial with coefficients in some subring of the reals. This function produces a sequence of pairs of rational numbers (a_i, b_i) such that every real roots of f lies in precisely one of the (closed) intervals $[a_i, b_i]$. The denominators of a_i, b_i are always powers of 2. The function uses Alesina–Galuzzi algorithm described in [1].

Warning: This function is tuned to give reasonably "nice" results and is not optimized for speed. In fact, for high degree polynomials it is much slower than the built-in function `Roots` for approximating real roots!

Example. Let us isolate real roots of the same polynomial $f = x^4 - 200x^2 + 40x - 2$ that we used in the previous example:

```
> R<x> := PolynomialRing(Integers());
> f := x^4 - 200*x^2 + 40*x - 2;
> RealRootIntervals(f);
[ <-29, 0>, <203/2048, 3277/32768>, <3277/32768, 1653/16384>,
<29/4, 29/2> ]
```

Indeed, the exact roots of f are:

$$\begin{aligned} -5\sqrt{2} - \sqrt{50+\sqrt{2}} &\in [-29, 0] \\ -5\sqrt{2} + \sqrt{50+\sqrt{2}} &\in \left[\frac{203}{2048}, \frac{3277}{32768}\right] \\ 5\sqrt{2} - \sqrt{50-\sqrt{2}} &\in \left[\frac{3277}{32768}, \frac{1653}{16384}\right] \\ 5\sqrt{2} + \sqrt{50-\sqrt{2}} &\in \left[\frac{29}{4}, \frac{29}{2}\right] \end{aligned}$$

`OrderingSeparation( K, I )`

`algorithm` MonStgElt *Default:* `"deterministic"`

`max_count` RngIntElt *Default:* `1000`

This function separates ordering of a formally real number field in the following sense. Let K be a formally real number field and $\sigma_1, \ldots, \sigma_r : K \to \mathbb{R}$ be all its real embeddings. This function constructs an element $a \in K^\times$, whose signs with respect to orderings of K are given by the sequence I, that is:

$$\operatorname{sgn} \sigma_i(a) = I[i].$$

The field K must be constructed as an absolute extension of $\mathbb{Q}$. The order of real embeddings is the one returned by Magma function `RealPlaces`.

The function accepts two optional parameters. The parameter `algorithm` controls what algorithm is used. It can have one of two possible values:

- "deterministic" (default): the function uses a deterministic algorithm described in todoKR202?.
- "randomized": the function uses a stochastic algorithm. This algorithm may sometimes provide nicer results than the deterministic one. On the other hand it may also completely fail to find a sought element.

The second parameter, called `max_count`, is meaningful only in a combination with `algorithm := "randomized"`. It determines the maximal number of iterations of the random search. If this parameter is set too law, then the function is more likely to fail to find the result.

Supports: number fields

Example. Let us consider the number field K of degree 10, that is denoted 10.10.762939453125.1 in [10].

```
> R<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(R![-1,-5,25,5,-50,-1,35,0,-10,0,1]);
```

It has 10 real embeddings $\sigma_1, \ldots, \sigma_{10}$. We will construct an element which is negative precisely in σ_2 and σ_7:

```
> r, _ := Signature(K); print r;
10
> a := OrderingSeparation(K, [1,-1,1,1,1,1,-1,1,1,1]); a;
16384*t^4 + 14336*t^3 - 21952*t^2 - 10976*t + 7203
> RealSigns(a);
[ 1, -1, 1, 1, 1, 1, -1, 1, 1, 1 ]
```

An element satisfying the same sign conditions can be obtained using Magma function `ChineseRemainderTheorem`, as follows:

```
> OK := MaximalOrder(K);
> b := K!ChineseRemainderTheorem( 1*OK, [1..r], OK!1, I); b;
1242*t^9 + 5876*t^8 - 6436*t^7 - 31694*t^6 + 11288*t^5 + 53651*t^4
- 10858*t^3 - 28786*t^2 + 6287*t + 1210
> RealSigns(b);
[ 1, -1, 1, 1, 1, 1, -1, 1, 1, 1 ]
```

The result is however much bigger.

Example. In the second example we take the field of degree 4, which is denoted 4.4.135232.1 in [10].

```
> R<x> := PolynomialRing(Rationals());
> K<theta> := NumberField(R![126, 24, -23, -2, 1]);
```

It has 4 real embeddings and we will construct an element which is negative precisely in the first two:

```
> r, _ := Signature(K); print r;
4
> a := OrderingSeparation(K, [-1,-1,1,1]); a;
-416*theta^3 + 3386*theta^2 - 7143*theta + 1593
> RealSigns(a);
[ -1, -1, 1, 1 ]
```

Let us compare the result of the deterministic algorithm above, with a randomized one. The result of the later function will differ with each invocation of the function.

```
> a := OrderingSeparation(K, [-1,-1,1,1] : algorithm := "randomized");
> a;
-24*theta + 12
> RealSigns(a);
[ -1, -1, 1, 1 ]
```

References

[1] Alberto Alesina and Massimo Galuzzi. Vincent's theorem from a modern point of view. *Rend. Circ. Mat. Palermo (2) Suppl.*, (64):179–191, 2000. Categorical studies in Italy (Perugia, 1997).

[2] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[3] John Cannon, Wieb Bosma, Claus Fieker, and Allan Steel. *Handbook Of Magma Functions*. School of Mathematics and Statistics, University of Sydney, 2015.

[4] John Cannon and Catherine Playoust. *First Steps in Magma*. School of Mathematics and Statistics, University of Sydney, 1996.

[5] Mawunyo Kofi Darkey-Mensah and Beata Rothkegel. Computing the length of sum of squares and pythagoras element in a global field, 2021.

[6] Hoon Hong. Bounds for absolute positiveness of multivariate polynomials. *J. Symbolic Comput.*, 25(5):571–585, 1998.

[7] Przemysław Koprowski. Isotropic vectors over global fields, 2021.

[8] Przemysław Koprowski and Alfred Czogała. Computing with quadratic forms over number fields. *J. Symbolic Comput.*, 89:129–145, 2018.

[9] T. Y. Lam. *Introduction to quadratic forms over fields*, volume 67 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 2005.

[10] The LMFDB Collaboration. The L-functions and modular forms database. `http://www.lmfdb.org`, 2021. [Online; accessed 9 February 2021].

[11] Denis Simon. Solving quadratic equations using reduced unimodular quadratic forms. *Math. Comp.*, 74(251):1531–1543 (electronic), 2005.

[12] Kazimierz Szymiczek. *Bilinear algebra*, volume 7 of *Algebra, Logic and Applications*. Gordon and Breach Science Publishers, Amsterdam, 1997. An introduction to the algebraic theory of quadratic forms.

[13] Panagiotis S. Vigklas. *Upper bounds on the values of the positive roots of polynomials*. PhD thesis, University of Thessaly, Volos, Greece, 2010.

8. MIT License