

Q Quadratic: PARI guide

(version 1.0)

A PARI/GP package for integral binary quadratic forms and quaternion algebras over \mathbb{Q} , with an emphasis on indefinite quadratic forms and indefinite quaternion algebras.

James Rickards
Department of Mathematics and Statistics,
McGill University, Montreal
Personal homepage
Github repository

© James Rickards 2021

Contents

1	Introduction	2
1.1	Overview of the main available methods	2
1.2	How to use the library	3
1.3	Programming style	3
1.4	How to use this manual	3
2	qq_base	4
2.1	Infinity	4
2.2	Integer vectors	4
2.3	Linear algebra	5
2.4	Lists	6
2.5	Random	8
2.6	Short vectors in lattices	9
2.7	Time	11
3	qq_bqf	11
3.1	Discriminant methods	11
3.2	Basic methods for binary quadratic forms	13
3.3	Basic methods, but specialized	17
3.4	Basic methods for indefinite quadratic forms	22
3.5	Class group and composition of forms	27
3.6	Representation of integers by forms - description tables	29
3.7	Representation of integers by forms - methods	32
3.8	Checking GP inputs	37
4	qq_bqf_int	38
4.1	Intersection Data	38
4.2	Intersection number computation	40
5	qq_geometry	43
5.1	Basic line, circle, and point operations	44
5.2	Intersection of lines and circles	49
5.3	Hyperbolic distance and area	52
5.4	Fundamental domain computation	52
5.5	Helper methods	53
6	qq_quat	54
6.1	Basic operations on elements in quaternion algebras	56
6.2	Basic operations on orders and lattices in quaternion algebras	60
6.3	Initialization methods	62
6.4	Conjugation of elements in a given order	65
6.5	Embedding quadratic orders into Eichler orders	67
6.6	Elements of norm n in an Eichler order	70
6.7	Fundamental domain methods	72

6.8	Checking methods	77
6.9	Property retrieval	78
6.10	Supporting methods	80
7	qq_quat_int	82
7.1	Intersection number based on roots	82
7.2	Intersection number based on x-linking	83
7.3	Intersection number based on fundamental domain	85
7.4	Intersection series	85
7.5	Intersection data	86
8	qq_visual	87
8.1	Histograms	87
9	Method declarations	89
9.1	qq_base	90
9.1.1	Infinity	90
9.1.2	Integer vectors	90
9.1.3	Linear algebra	90
9.1.4	Lists	90
9.1.5	Random	90
9.1.6	Short vectors in lattices	91
9.1.7	Time	91
9.2	qq_bqf	91
9.2.1	Discriminant methods	91
9.2.2	Basic methods for binary quadratic forms	91
9.2.3	Basic methods, but specialized	92
9.2.4	Basic methods for indefinite quadratic forms	93
9.2.5	Class group and composition of forms	93
9.2.6	Representation of integers by forms	94
9.2.7	Checking GP inputs	95
9.3	qq_bqf_int	95
9.3.1	Intersection Data	95
9.3.2	Intersection number computation	95
9.4	qq_geometry	95
9.4.1	Basic line, circle, and point operations	95
9.4.2	Intersection of lines and circles	96
9.4.3	Hyperbolic distance and area	97
9.4.4	Helper methods	97
9.5	qq_quat	97
9.5.1	Basic operations on elements in quaternion algebras	97
9.5.2	Basic operations on orders and lattices in quaternion algebras	98
9.5.3	Initialization methods	98
9.5.4	Conjugation of elements in a given order	99
9.5.5	Embedding quadratic orders into Eichler orders	99

9.5.6	Elements of norm n in an Eichler order	100
9.5.7	Fundamental domain methods	100
9.5.8	Checking methods	101
9.5.9	Property retrieval	101
9.5.10	Supporting methods	101
9.6	qq-quat_int	102
9.6.1	Intersection number based on roots	102
9.6.2	Intersection number based on x-linking	102
9.6.3	Intersection number based on fundamental domain	102
9.6.4	Intersection series	102
9.6.5	Intersection data	103
9.7	qq-visual	103
9.7.1	Histograms	103

References	103
-------------------	------------

1 Introduction

The roots for this library came from my thesis project, which involved studying intersection numbers of geodesics on modular and Shimura curves. To be able to do explicit computations, I wrote many GP scripts to deal with indefinite binary quadratic forms, and indefinite quaternion algebras. This library is a revised version of those scripts, rewritten in PARI ([The21]) for optimal efficiency.

The package has been designed to be easily usable with GP, with more specific and powerful methods available to PARI users. More specifically, the GP functions are all given wrappers so as to not break, and the PARI methods often allow passing in of precomputed data like the discriminant, the reduced orbit of an indefinite quadratic form, etc. If you only intend on using this library in GP, please consult the GP manual instead.

1.1 Overview of the main available methods

For integral binary quadratic forms, there are methods available to:

- Generate lists of (fundamental, coprime to a given integer n) discriminants;
- Compute the basic properties, e.g. the automorph, discriminant, reduction, and equivalence of forms;
- For indefinite forms, compute all reduced forms, the Conway river, left and right neighbours of river/reduced forms;
- Compute the narrow class group and a set of generators, as well as a reduced form for each equivalence class in the group;
- Output all integral solutions (x, y) to $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = n$ for any integers A, B, C, D, E, F, n ;
- Solve the simultaneous equations $Ax^2 + By^2 + Cz^2 + Dxy + Exz + Fyz = n_1$ and $Ux + Vy + Wz = n_2$ for any integers $A, B, C, D, E, F, U, V, W, n_1, n_2$.
- Compute the intersection number of two primitive indefinite binary quadratic form.

For quaternion algebras over \mathbb{Q} , there are methods available to:

- Initialize the algebra given the ramification, and initialize maximal/Eichler orders (with specific care given to algebras ramified at ≤ 2 finite places);
- Compute all optimal embeddings of a quadratic order into a quaternion algebra, and arrange them with respect to the class group action and their orientation;
- Compute all elements of norm n in an Eichler order (when n is coprime to the level), and compute the action of the Hecke operator T_n on optimal embeddings;
- Compute the intersection number of pairs of optimal embeddings;
- Compute the fundamental domain of unit groups of Eichler orders in indefinite algebras (Shimura curves).

The methods dealing with fundamental domains have been improved and modified to work with quaternion algebras over number fields, and with the pari “alg” methods (e.g. a quaternion algebra initialized with `algin`). If your interest is with these fundamental domains, then you should check out “Fundamental domains for Shimura curves” instead.

1.2 How to use the library

The source code and makefile are all contained in the sub-folder “Source Code”. Calling `make` will automatically move the `.so` file created up to the main folder. Some of the methods are able to call external programs (e.g. a Python script or LaTeX compiler), and the syntax assumes the user is using Linux Subsystem for Windows. If this is not the case, then either do not use the automatic parts of the methods, or modify them appropriately.

1.3 Programming style

I have gone for optimal efficiency by using the most specific methods (e.g. `addii`, passing in precomputed data, etc.) whenever possible. What this means is that it is very easy to break the library if you feed in bad inputs! However, when working with GP we really do not want this to happen, so every method available to GP has a wrapper for protection against segmentation breaks (if it is required). This wrapper is always indicated by adding the suffix “_tc” (which stands for type check) to the method, and its function is to check that the inputs are kosher and feed them on to the appropriate method. There are some methods that do not require this wrapper, and they do not come with it.

If there is a piece of “standard data” that is very useful in a method, there is typically a method that does not require passing in of this data, and another that does. This allows the user to maximize efficiency by not recomputing this standard data repeatedly. For example, if you are doing a lot of computations with a fixed quadratic form q , then you can store the discriminant of q and pass it along with q when this is available. If you are more concerned with getting it correct and not breaking your programs, then you can just use the “_tc” methods, as they require no precomputed data and are hard to break. Once you have a working program, you can revisit this to optimize for efficiency.

Many methods require passing in the precision as a long. This is always the last input, and is always denoted by `long prec`.

1.4 How to use this manual

Sections 2-8 contain detailed descriptions of every function: the input, output, and what the function does. The sections are labeled by source files, and are divided into subsections of “similar” methods. If you are seeking a function for a certain task, have a look through here.

Section 9 contains simply the method declarations, and is useful as a quick reference. Clicking the name of a method in this section will take you to its full description in Sections 2-8, and clicking on the name there will take you back to Section 9.

Methods accessible to GP are given a green background, static methods are given a blue background, and non-GP accessible and non-static methods are in yellow. The methods are generally alphabetized, with static methods appearing at the end of sections. Any non-static method that is not stack clean is given a red background, and will be appropriately noted in the description. Start by looking at the green methods, and when you want to use one check the surrounding yellow methods for the precise version you want. Unless you are modifying these methods directly, you can ignore the blue methods.

2 qq_base

This is a collection of “basic” functions and structures, which are useful in various places. Highlights include computing all square roots of an integer modulo n for general n (boosting up the PARI-implemented method for prime powers), finding all small vectors in a lattice, and methods for dealing with lists of **GENS** and **longs**.

2.1 Infinity

In dealing with the completed complex upper half plane, the projective line over \mathbb{Q} , etc., we would like to work with ∞ , but currently PARI does not support adding/dividing infinities by finite numbers. The functions here are wrappers around addition and division to allow for this.

Name:	GEN addoo
Input:	GEN a, GEN b
Input format:	a, b complex numbers or infinity
Output format:	Complex number or $\pm\infty$
Description:	Returns a+b , where the output is a if a is infinite, b if b is infinite, and a+b otherwise.

Name:	GEN divoo
Input:	GEN a, GEN b
Input format:	a, b complex numbers or infinity
Output format:	Complex number or $\pm\infty$
Description:	Returns a/b , where a/0 will return $\pm\infty$ (depending on the sign of a), and $\pm\infty/\mathbf{b}$ will return $\pm\infty$ (depending on the sign of b). Note that both 0/0 and ∞/∞ return ∞ .

2.2 Integer vectors

The following methods are typically available for **ZC**, but not for **ZV**.

Name:	GEN ZV_copy
Input:	GEN v
Input format:	v a vector with integer entries
Output format:	Vector
Description:	Returns a copy of the integral vector v .

Name:	int ZV_equal
Input:	GEN v1, GEN v2
Input format:	v1, v2 vectors with integer entries
Output format:	0 or 1
Description:	Returns 1 if v1=v2 and 0 else.

Name:	GEN ZV_Z_divexact
Input:	GEN v, GEN y
Input format:	v an integral vector, y a non-zero integer which divides all components of v
Output format:	Vector
Description:	Returns v/y.

Name:	GEN ZV_Z_mul
Input:	GEN v, GEN x
Input format:	v an integral vector, x an integer
Output format:	Vector
Description:	Returns vx.

2.3 Linear algebra

lin_intsolve is essentially just gbezout, but it outputs to a format that is useful to me.

Name:	GEN FpM_eigenvecs
Input:	GEN M, GEN p
Input format:	FpM M, prime p
Output format:	Vector of [Eigenvalue, [Eigenvectors]]
Description:	Computes and returns all eigenvalues and eigenvectors of M over Fp.

Name:	GEN lin_intsolve
Input:	GEN A, GEN B, GEN n
Input format:	Integers A, B, C
Output format:	gen_0 or $[[m_x, m_y], [x_0, y_0]]$.
Description:	Solves $Ax + By = n$ using gbezout, where the general solution is $x = x_0 + m_x t$ and $y = y_0 + m_y t$ for $t \in \mathbb{Z}$.

Name:	GEN lin_intsolve_tc
Input:	GEN A, GEN B, GEN n
Input format:	Integers A, B, C
Output format:	gen_0 or $[[m_x, m_y], [x_0, y_0]]$.
Description:	Checks that A, B, n are integral, and returns lin_intsolve(A, B, n).

Name:	GEN mat3_complete
Input:	GEN A, GEN B, GEN C
Input format:	Integers A, B, C with $\gcd(A, B, C) = 1$
Output format:	Matrix
Description:	Returns a 3x3 integer matrix with determinant 1 and first row A, B, C.

Name:	<code>GEN mat3_complete_tc</code>
Input:	<code>GEN A, GEN B, GEN C</code>
Input format:	Integers A, B, C with $\gcd(A, B, C) = 1$
Output format:	Matrix
Description:	Checks that A, B, C are relatively prime integers, and returns <code>mat3_complete(A, B, C)</code> .

2.4 Lists

There are three dynamically linked lists implemented: `clist`, `glist`, `llist`. A circular list (`clist`) `C` stores a `GEN` (accessible by `C->data`), a pointer to the next element (`C->next`), and the previous element (`C->prev`). A generic list is the same, except it does not store a pointer to the previous element. A long list is the same as a generic list, except it stores the data type `long` instead of `GEN` (and hence does not enter into the PARI stack).

Lists should be initialized by calling `(c/g/l)list *L=NULL;`, and use the following methods to work with them. If you do not call the `togvec` or `tovecsmall` methods, you should free the list using the appropriate `(c/g/l)list_free` methods.

Name:	<code>void clist_free</code>
Input:	<code>clist *l, long length</code>
Input format:	Pointer to a <code>clist</code> <code>l</code> , length of the list <code>length</code>
Output format:	-
Description:	Frees the <code>clist</code> (each data point had been initialized with <code>pari_malloc</code>). If <code>l</code> is shorter than <code>length</code> , an error will occur, and if it is longer than the remaining part of the list has not been freed.

Name:	<code>void clist_putbefore</code>
Input:	<code>clist **head_ref, GEN new_data</code>
Input format:	-
Output format:	-
Description:	Adds an element containing <code>new_data</code> before the element pointed to by <code>head_ref</code> , and updates the list start position.

Name:	<code>void clist_putafter</code>
Input:	<code>clist **head_ref, GEN new_data</code>
Input format:	-
Output format:	-
Description:	Adds an element containing <code>new_data</code> after the element pointed to by <code>head_ref</code> , and updates the list start position.

Name:	GEN clist_togvec
Input:	clist *l, long length, int dir
Input format:	length the length of l, and dir=-1 or 1.
Output format:	Vector
Description:	Returns a vector consisting of the list from l and onward of length length, and frees the list. If dir=1 we go through the list via l->next, and if dir=-1 we go through via l->prev.

Name:	void glist_free
Input:	glist *l
Input format:	Pointer to a glist l
Output format:	-
Description:	Frees the glist (each data point had been initialized with pari_malloc).

Name:	GEN glist_pop
Input:	glist **head_ref
Input format:	-
Output format:	-
Description:	Removes the last element of the list, frees this list element, and returns the data entry. This does NOT copy the return element, so it is NOT stack safe.

Name:	void glist_putstart
Input:	glist **head_ref, GEN new_data
Input format:	-
Output format:	-
Description:	Adds an element containing new_data before the element pointed to by head_ref, and updates the list start position.

Name:	GEN glist_togvec
Input:	glist *l, long length, int dir
Input format:	length the length of l, and dir=-1 or 1.
Output format:	Vector
Description:	Returns a vector consisting of the list from l and onward of length length, and frees the list. If dir=1 we fill in the return vector from index 1 to length, and if dir=-1 we go in the opposite direction.

Name:	GEN glist_togvec_append
Input:	glist *l, GEN v, long length, int dir
Input format:	length the length of l, v a vector, and dir=-1, 1.
Output format:	Vector
Description:	Appends l to the end of v, and frees l. If dir=1 we loop through l from index 1 to length, and if dir=-1 we go in the opposite direction.

Name:	<code>void llist_free</code>
Input:	<code>llist *l</code>
Input format:	Pointer to a <code>clist l</code>
Output format:	-
Description:	Frees the <code>llist</code> (each data point had been initialized with <code>pari_malloc</code>). If <code>l</code> is shorter than <code>length</code> , an error will occur, and if it is longer than the remaining part of the list has not been freed.

Name:	<code>long llist_pop</code>
Input:	<code>llist **head_ref</code>
Input format:	-
Output format:	-
Description:	Removes the last element of the list, frees this list element, and returns the data entry.

Name:	<code>void llist_putstart</code>
Input:	<code>llist **head_ref, GEN new_data</code>
Input format:	-
Output format:	-
Description:	Adds an element containing <code>new_data</code> before the element pointed to by <code>head_ref</code> , and updates the list start position.

Name:	<code>GEN llist_togvec</code>
Input:	<code>llist *l, long length, int dir</code>
Input format:	<code>length</code> the length of <code>l</code> , and <code>dir=-1</code> or <code>1</code> .
Output format:	Vector
Description:	Returns a vector consisting of the list from <code>l</code> and onward of length <code>length</code> , and frees the list. If <code>dir=1</code> we fill in the return vector from index <code>1</code> to <code>length</code> , and if <code>dir=-1</code> we go in the opposite direction.

Name:	<code>GEN llist_tovecsmall</code>
Input:	<code>llist *l, long length, int dir</code>
Input format:	<code>length</code> the length of <code>l</code> , and <code>dir=-1</code> or <code>1</code> .
Output format:	Vecsmall
Description:	Returns a Vecsmall consisting of the list from <code>l</code> and onward of length <code>length</code> , and frees the list. If <code>dir=1</code> we fill in the return vector from index <code>1</code> to <code>length</code> , and if <code>dir=-1</code> we go in the opposite direction.

2.5 Random

Methods used for producing random things.

Name:	GEN rand_elt
Input:	GEN v
Input format:	v a vector
Output format:	
Description:	Returns a random component of v.

Name:	long rand_l
Input:	long len
Input format:	len a positive integer
Output format:	long
Description:	Returns a random long from 1 to len.

2.6 Short vectors in lattices

We follow the Fincke-Pohst method ([FP85]) for finding short vectors in a lattice. The general setup is the following: A is a positive definite symmetric matrix, and we are searching for non-zero column vectors x such that $C_1 \leq x^T A x \leq C_2$. We can also add the condition of `condition=[M, n]`, where we require $x^T M x = n$ (in general, M will give an indefinite condition). If `onesign=1`, we only return one of $x, -x$ for each possible x . If `isintegral=1`, we assume that the quadratic form corresponding to A is integral.

Name:	GEN lat_smallvectors
Input:	GEN A, GEN C1, GEN C2, GEN condition, int onesign, int isintegral, int rdataonly, long prec
Input format:	See above, and rdataonly=0, 1
Output format:	Vector
Description:	Computes the small vectors. If rdataonly=1, instead returns [chol, U, perminv, condition], which can be fed into <code>lat_smallvectors_givendata</code> (useful when calling this method multiple times on the same matrix A).

Name:	GEN lat_smallvectors_givendata
Input:	GEN chol, GEN U, GEN perminv, GEN C1, GEN C2, GEN condition, int onesign, long prec
Input format:	See above
Output format:	Vector
Description:	Computes the small vectors given the input of [chol, U, perminv, condition], which come from <code>lat_smallvectors</code> . The output is the vector of solutions $[x, x^T A x]$.

Name:	GEN lat_smallvectors_tc
Input:	GEN A, GEN C1, GEN C2, int onesign, int isintegral, long prec
Input format:	See above
Output format:	Vector
Description:	Checks that A is a matrix, if C2=0 replaces (C1, C2) by (0, C1), and returns the small vectors.

Name:	GEN lat_smallvectors_cholesky
Input:	GEN Q, GEN C1, GEN C2, GEN condition, int onesign, long prec
Input format:	See above, and Q is the Cholesky decomposition of A.
Output format:	Vector
Description:	Returns the small vectors given the Cholesky decomposition. Mostly useful as a sub-method to lat_smallvectors_givendata.

Name:	GEN mat_choleskydecomp
Input:	GEN A, int rcoefs, long prec
Input format:	A a symmetric matrix, rcoefs=0, 1
Output format:	Matrix
Description:	Returns the Cholesky decomposition of A. If rcoefs=0, returns R where $R^T R = A$. If rcoefs=1, returns B, the upper triangular matrix such that $x^T A x$ is expressible as $\sum_{i=1}^n b_i i(x_i + \sum_{j=i+1}^n b_i j x_j)^2$.

Name:	GEN mat_choleskydecomp_tc
Input:	GEN A, int rcoefs, long prec
Input format:	A a symmetric matrix, rcoefs=0, 1
Output format:	Matrix
Description:	Checks that A is a square matrix and returns mat_choleskydecomp(A, rcoefs, prec).

Name:	GEN quadraticinteger
Input:	GEN A, GEN B, GEN C
Input format:	Integers A, B, C
Output format:	Vector
Description:	Returns the integral solutions to $Ax^2 + Bx + C = 0$.

Name:	int opp_gcmp
Input:	void *data, GEN x, GEN y
Input format:	GEN's x, y
Output format:	-1, 0, 1
Description:	Returns -gcmp(x, y); useful for sorting backwards.

2.7 Time

Methods for returning and printing time. Uses the C library `time.h`, as this is significantly less work than `getwalltime()`.

Name:	<code>char* returntime</code>
Input:	-
Input format:	-
Output format:	-
Description:	Returns the current time as a string.

Name:	<code>void printtime</code>
Input:	-
Input format:	-
Output format:	-
Description:	Prints the current time.

3 qq_bqf

These methods primarily deal with primitive integral homogeneous positive definite/indefinite binary quadratic forms. Such a form is represented by the vector $[A, B, C]$, which represents $AX^2 + BXY + CY^2$. Some of the basic methods support non-primitive, negative definite, or square discriminant forms (like `bqf_disc` or `bqf_trans`), but more complex ones (like `bqf_isequiv`) may not.

On the other hand, the method `bqf_reps` allows non-primitive forms, as well as negative definite and square discriminant forms. Going further, `bqf_bigreps` allows non-homogeneous binary quadratic forms (but the integral requirement is never dropped).

In this and subsequent sections, a **BQF** is an integral binary quadratic form, an **IBQF** is an indefinite BQF, a **DBQF** is a positive definite BQF, a **PIBQF/PDBQF** is a primitive indefinite/positive definite BQF respectively, and a **PBQF** is either a PIBQF or a PDBQF.

In general, a method taking in a BQF will start with `bqf_`. This is further specialized to indefinite/positive definite/square discriminant/zero discriminant forms by adding the prefixes `i/d/s/z` respectively.

3.1 Discriminant methods

These methods deal with discriminant operations that do not involve quadratic forms.

Name:	<code>GEN disclist</code>
Input:	<code>GEN D1, GEN D2, int fund, GEN cop</code>
Input format:	Integers <code>D1, D2</code> , <code>fund=0, 1</code> , <code>cop</code> an integer
Output format:	Vector
Description:	Returns the set of discriminants (non-square integers equivalent to 0, 1 modulo 4) between <code>D1</code> and <code>D2</code> inclusive. If <code>fund=1</code> , only returns fundamental discriminants, and if <code>cop≠0</code> , only returns discriminants coprime to <code>cop</code> .

Name:	GEN <code>discprimeindex</code>
Input:	GEN <code>D</code> , GEN <code>facs</code>
Input format:	Discriminant <code>D</code> , <code>facs=0</code> or the factorization of <code>D</code> (the output of <code>Z_factor</code>)
Output format:	Vector
Description:	Returns the set of primes p for which D/p^2 is a discriminant.

Name:	GEN <code>discprimeindex_tc</code>
Input:	GEN <code>D</code>
Input format:	Discriminant <code>D</code>
Output format:	Vector
Description:	Checks that <code>D</code> is a discriminant, and returns <code>discprimeindex(D, gen_0)</code> .

Name:	int <code>isdisc</code>
Input:	GEN <code>D</code>
Input format:	-
Output format:	0 or 1
Description:	Returns 1 if <code>D</code> is a discriminant and 0 else.

Name:	GEN <code>pell</code>
Input:	GEN <code>D</code>
Input format:	Positive discriminant <code>D</code>
Output format:	[<code>T</code> , <code>U</code>]
Description:	Returns the smallest solution in the positive integers to Pell's equation $T^2 - DU^2 = 4$.

Name:	GEN <code>pell_tc</code>
Input:	GEN <code>D</code>
Input format:	Positive discriminant <code>D</code>
Output format:	[<code>T</code> , <code>U</code>]
Description:	Checks that <code>D</code> is a positive discriminant, and returns <code>pell(D)</code> .

Name:	GEN <code>posreg</code>
Input:	GEN <code>D</code> , long <code>prec</code>
Input format:	Positive discriminant <code>D</code>
Output format:	Real number
Description:	Returns the positive regulator of \mathcal{O}_D , i.e. the logarithm of the fundamental unit of norm 1 in the unique order of discriminant D .

Name:	GEN <code>posreg_tc</code>
Input:	GEN <code>D</code> , long <code>prec</code>
Input format:	Positive discriminant <code>D</code>
Output format:	Real number
Description:	Checks that <code>D</code> is a positive discriminant, and returns <code>posreg(D, prec)</code> .

Name:	GEN quadroot
Input:	GEN D
Input format:	Non-square integer D
Output format:	t_QUAD
Description:	Outputs the t_QUAD w for which $w^2 = D$.

Name:	GEN quadroot_tc
Input:	GEN D
Input format:	Discriminant D
Output format:	t_QUAD
Description:	Checks that D is a discriminant and returns quadroot(D).

3.2 Basic methods for binary quadratic forms

Recall that the BQF $AX^2 + BXY + CY^2$ is represented as the vector $[A, B, C]$.

Name:	GEN bqf_automorph_tc
Input:	GEN q
Input format:	PBQF q
Output format:	Matrix
Description:	Returns the invariant automorph M of q, i.e. the $\text{PSL}(2, \mathbb{Z})$ matrix with positive trace that generates the stabilizer of q (a cyclic group of order 1, 2, 3, or ∞).

Name:	int bqf_compare
Input:	void *data, GEN q1, GEN q2
Input format:	*data=NULL, q1 and q2 BQFs
Output format:	-1, 0, 1
Description:	Lexicographically compares q1 and q2, returning -1 if $q_1 < q_2$, 0 if $q_1 = q_2$, and 1 if $q_1 > q_2$. This method is used to sort and search a set of BQFs more efficiently (with gen_sort and gen_search).

Name:	int bqf_compare_tmat
Input:	void *data, GEN d1, GEN d2
Input format:	*data=NULL, di=[qi, mi] with qi a BQF for i=1,2
Output format:	-1, 0, 1
Description:	Lexicographically compares q1 and q2, returning -1 if $q_1 < q_2$, 0 if $q_1 = q_2$, and 1 if $q_1 > q_2$. This method is used to sort and search a set of BQFs where we are also keeping track of an extra data point mi, often a transition matrix (whose value has no effect on the output of this method).

Name:	GEN bqf_disc
Input:	GEN q
Input format:	BQF q
Output format:	Integer
Description:	Returns the discriminant of q, i.e. $B^2 - 4AC$ where $q=[A, B, C]$.

Name:	GEN bqf_disc_tc
Input:	GEN q
Input format:	BQF q
Output format:	Integer
Description:	Checks that q is a BQF, and returns bqf_disc(q).

Name:	GEN bqf_isequiv
Input:	GEN q1, GEN q2, GEN rootD, int Dsign, int tmat
Input format:	PBQFs q1, q2 of the same discriminant D , rootD the real square root of D if $D > 0$ (and anything if $D < 0$), Dsign the sign of D , tmat=0, 1
Output format:	gen_0, gen_1, or a matrix
Description:	Determines if q1 and q2 are equivalent or not. If tmat=0, returns gen_0 or gen_1, and if tmat=1, returns gen_0 if not equivalent and a $SL(2, \mathbb{Z})$ transition matrix taking q1 to q2 if they are equivalent.

Name:	GEN bqf_isequiv_set
Input:	GEN q, GEN S, GEN rootD, int Dsign, int tmat
Input format:	PBQFs q and set of PBQFs S, all of the same discriminant D , rootD the real square root of D if $D > 0$ (and anything if $D < 0$), Dsign the sign of D , tmat=0, 1
Output format:	Integer or [i, M]
Description:	Determines if q and an element of S are equivalent or not. If tmat=0, returns gen_0 if not and an index i such that q is equivalent to S[i] if they are equivalent. If tmat=1, returns gen_0 if not equivalent and [i, M] if they are, where $M \circ q = S[i]$.

Name:	GEN bqf_isequiv_tc
Input:	GEN q1, GEN q2, int tmat, long prec
Input format:	q1 a PBQF, q2 a PBQF or a set of PBQFs, tmat=0, 1
Output format:	Integer or matrix or [i, M]
Description:	Checks if q1 is a PBQF, q2 is a PBQF or a set of PBQFs, and returns bqf_isequiv or bqf_isequiv_set on q1 and q2 as appropriate. Elements of q2 need not have the same discriminant as each other or q1.

Name:	<code>int bqf_isreduced</code>
Input:	GEN <code>q</code> , <code>int Dsign</code>
Input format:	<code>q</code> a PBQF of discriminant D , <code>Dsign</code> the sign of D
Output format:	0, 1
Description:	Returns 1 if <code>q</code> is reduced, and 0 if <code>q</code> is not reduced. We use the standard reduced definition when $D < 0$, and the conditions $AC < 0$ and $B > A + C $ when $D > 0$.

Name:	<code>int bqf_isreduced_tc</code>
Input:	GEN <code>q</code>
Input format:	<code>q</code> a PBQF
Output format:	0, 1
Description:	Checks that <code>q</code> is a PBQF and returns 1 if reduced, and 0 if not reduced.

Name:	<code>GEN bqf_random</code>
Input:	GEN <code>maxc</code> , <code>int type</code> , <code>int primitive</code>
Input format:	<code>maxc</code> a positive integer, <code>type</code> , <code>primitive=0, 1</code>
Output format:	BQF
Description:	Returns a random BQF of non-square discriminant with coefficient size at most <code>maxc</code> . If <code>type=-1</code> it will be positive definite, <code>type=1</code> indefinite, and <code>type=0</code> either type. If <code>primitive=1</code> the form will be primitive, otherwise it need not be.

Name:	<code>GEN bqf_random_D</code>
Input:	GEN <code>maxc</code> , GEN <code>D</code>
Input format:	<code>maxc</code> a positive integer, <code>D</code> a discriminant
Output format:	BQF
Description:	Checks that <code>maxc</code> is a positive integer and <code>D</code> is a discriminant, and returns a random primitive BQF of discriminant D (positive definite if $D < 0$).

Name:	<code>GEN bqf_red</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code> , <code>int Dsign</code> , <code>int tmat</code>
Input format:	<code>q</code> a PBQF of discriminant D , <code>rootD</code> the square root of D if $D > 0$ (and anything if $D < 0$), <code>Dsign</code> the sign of D , <code>tmat=0,1</code>
Output format:	BQF or $[q', M]$
Description:	Outputs the reduction of <code>q</code> . If <code>tmat=0</code> this is a BQF, otherwise this is $[q', M]$ where the reduction is q' and the transition matrix is M .

Name:	<code>GEN bqf_red_tc</code>
Input:	GEN <code>q</code> , <code>int tmat</code> , long <code>prec</code>
Input format:	<code>q</code> a PBQF, <code>tmat=0,1</code>
Output format:	BQF or $[q', M]$
Description:	Checks that <code>q</code> is a PBQF, and returns <code>bqf_red(q, ...)</code> .

Name:	GEN bqf_roots
Input:	GEN q, GEN D, GEN w
Input format:	BQF q of discriminant D, with $w^2 = D$ where w is a t_QUAD if D is not a square
Output format:	[r1, r2]
Description:	Returns the roots of $q(x,1)=0$, with the first root coming first. If D is not a square, these are of type t_QUAD, and otherwise they will be rational or infinite. If D=0, the roots are equal.

Name:	GEN bqf_roots_tc
Input:	GEN q
Input format:	BQF q
Output format:	[r1, r2]
Description:	Checks that q is a BQF, and returns bqf_roots(q, ...).

Name:	GEN bqf_trans
Input:	GEN q, GEN M
Input format:	BQF q, $M \in SL(2, \mathbb{Z})$
Output format:	BQF
Description:	Returns $M \circ q$.

Name:	GEN bqf_trans_tc
Input:	GEN q, GEN M
Input format:	BQF q, $M \in SL(2, \mathbb{Z})$
Output format:	BQF
Description:	Checks that q is q BQF and $M \in SL(2, \mathbb{Z})$, and returns bqf_trans(q, M).

Name:	GEN bqf_transL
Input:	GEN q, GEN n
Input format:	BQF q, integer n
Output format:	BQF
Description:	Returns $\begin{pmatrix} 1 & n \\ 0 & 1 \end{pmatrix} \circ q$.

Name:	GEN bqf_transR
Input:	GEN q, GEN n
Input format:	BQF q, integer n
Output format:	BQF
Description:	Returns $\begin{pmatrix} 1 & 0 \\ n & 1 \end{pmatrix} \circ q$.

Name:	GEN bqf_transS
Input:	GEN q
Input format:	BQF q
Output format:	BQF
Description:	Returns $\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \circ q$.

Name:	GEN bqf_trans_coprime
Input:	GEN q, GEN n
Input format:	BQF q, integer n coprime to $\gcd(q)$
Output format:	BQF
Description:	Returns a BQF equivalent to q whose first coefficient is coprime to n.

Name:	GEN bqf_trans_coprime_tc
Input:	GEN q, GEN n
Input format:	BQF q, non-zero integer n
Output format:	BQF
Description:	Checks that q is a BQF and n is coprime to $\gcd(q)$, and returns bqf_trans_coprime(q, n).

Name:	GEN ideal_tobqf
Input:	GEN numf, GEN ideal
Input format:	numf a quadratic number field, ideal an ideal in numf
Output format:	BQF
Description:	Converts the ideal to a BQF and returns it.

3.3 Basic methods, but specialized

These are the above basic methods, but specialized to the positive definite/indefinite cases.

Name:	GEN dbqf_automorph
Input:	GEN q, GEN D
Input format:	PDBQF q of discriminant D
Output format:	Matrix
Description:	Returns the invariant automorph M of q in $\text{PSL}(2, \mathbb{Z})$, which is trivial if $D < -4$, has order 2 if $D = -4$, and order 3 if $D = -3$.

Name:	GEN dbqf_isequiv
Input:	GEN q1, GEN q2
Input format:	DBQFs q1, q2 of the same discriminant
Output format:	gen.0, gen.1
Description:	Returns 1 if q1 is equivalent to q2 and 0 if not.

Name:	<code>long dbqf_isequiv_set</code>
Input:	GEN q , GEN S
Input format:	DBQF q and a set of DBQFs S , with all forms in S and q having the same discriminant
Output format:	Integer
Description:	Returns <code>gen_0</code> if q is not similar to any form in S , and an index i such that q is similar to $S[i]$ otherwise.

Name:	<code>GEN dbqf_isequiv_set_tmat</code>
Input:	GEN q , GEN S
Input format:	DBQF q and a set of DBQFs S , with all forms in S and q having the same discriminant
Output format:	<code>gen_0</code> or $[i, M]$
Description:	Returns <code>gen_0</code> if q is not similar to any form in S , and otherwise returns $[o, M]$, where q is similar to $S[i]$ with transition matrix M .

Name:	<code>GEN dbqf_isequiv_tmat</code>
Input:	GEN q_1 , GEN q_2
Input format:	DBQFs q_1 , q_2 of the same discriminant
Output format:	0, matrix
Description:	Returns 0 if q_1 is not equivalent to q_2 and a possible transition matrix if it is.

Name:	<code>GEN dbqf_red</code>
Input:	GEN q
Input format:	q a DBQF
Output format:	DBQF
Description:	Returns the reduction of q .

Name:	<code>GEN dbqf_red_tmat</code>
Input:	GEN q
Input format:	q a DBQF
Output format:	$[q', M]$
Description:	Returns $[q', M]$, where the reduction of q is q' and the transition matrix is M .

Name:	<code>GEN ibqf_automorph_D</code>
Input:	GEN q , GEN D
Input format:	q a PIBQF of discriminant D
Output format:	Matrix
Description:	Returns the invariant automorph of q , i.e. the generator with positive trace (and infinite order) of the stabilizer of q in $\text{PSL}(2, \mathbb{Z})$. This method calls <code>pell(D)</code> , so if this is already computed, use <code>ibqf_automorph_pell</code> instead.

Name:	GEN <code>ibqf_automorph_pell</code>
Input:	GEN <code>q</code> , GEN <code>qpell</code>
Input format:	<code>q</code> a PIBQF of discriminant D , where <code>qpell</code> is the output of <code>pell(D)</code>
Output format:	Matrix
Description:	Returns the invariant automorph of <code>q</code> . If you don't care about the output of <code>pell(D)</code> , then use <code>ibqf_automorph_D</code> instead.

Name:	GEN <code>ibqf_isequiv</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , GEN <code>rootD</code>
Input format:	PIBQFs <code>q1</code> , <code>q2</code> of the same discriminant D , <code>rootD</code> the real square root of D
Output format:	<code>gen_0</code> , <code>gen_1</code>
Description:	Determines if <code>q1</code> and <code>q2</code> are equivalent or not, and returns the answer.

Name:	long <code>ibqf_isequiv_set_byq</code>
Input:	GEN <code>q</code> , GEN <code>S</code> , GEN <code>rootD</code>
Input format:	PIBQFs <code>q</code> and set of PIBQFs <code>S</code> , all of the same discriminant D , <code>rootD</code> the real square root of D
Output format:	Integer
Description:	Determines if <code>q</code> and an element of <code>S</code> are equivalent or not. Returns 0 if not equivalent, and an index <code>i</code> such that <code>q</code> is equivalent to <code>S[i]</code> otherwise. Generally slower than <code>ibqf_isequiv_set_byS</code> , so not recommended for use.

Name:	long <code>ibqf_isequiv_set_byq_presorted</code>
Input:	GEN <code>qredsorted</code> , GEN <code>S</code> , GEN <code>rootD</code>
Input format:	<code>qredsorted</code> is <code>ibqf_redorbit_posonly(q, rootD)</code> , sorted with <code>gen_sort_inplace(qredsorted, NULL, &bqf_compare, NULL)</code> , <code>S</code> is a set of PIBQFs with all forms in <code>S</code> and <code>q</code> having discriminant D , and <code>rootD</code> is the positive square root of D
Output format:	Integer
Description:	<code>ibqf_isequiv_set_byq</code> where the sorted positive reduced orbit of <code>q</code> is inputted. Useful if you are making multiple calls to <code>bqf_is_equiv_set</code> with the same <code>q</code> but varying sets <code>S</code> . If this is not the case, <code>ibqf_isequiv_byS</code> is generally faster.

Name:	GEN <code>ibqf_isequiv_set_byq_tmat</code>
Input:	GEN <code>q</code> , GEN <code>S</code> , GEN <code>rootD</code>
Input format:	PIBQFs <code>q</code> and set of PIBQFs <code>S</code> , all of the same discriminant D , <code>rootD</code> the real square root of D
Output format:	<code>gen_0</code> , <code>[i, M]</code>
Description:	Determines if <code>q</code> and an element of <code>S</code> are equivalent or not. Returns <code>gen_0</code> if not equivalent, and <code>[i, M]</code> otherwise, where <code>q</code> is equivalent to <code>S[i]</code> with transition matrix <code>M</code> . Generally slower than <code>ibqf_isequiv_set_byS_tmat</code> , so not recommended for use.

Name:	GEN <code>ibqf_isequiv_set_byq_tmat_presorted</code>
Input:	GEN <code>qredsorted</code> , GEN <code>S</code> , GEN <code>rootD</code>
Input format:	<code>qredsorted</code> is <code>ibqf_redorbit_posonly_tmat(q, rootD)</code> sorted with <code>gen_sort_inplace(qredsorted, NULL, &bqf_compare_tmat, NULL)</code> , <code>S</code> is a set of PIBQFs with all forms in <code>S</code> and <code>q</code> having discriminant D , and <code>rootD</code> is the positive square root of D
Output format:	<code>gen_0</code> , <code>[i, M]</code>
Description:	<code>ibqf_isequiv_set_byq_tmat</code> where the sorted positive reduced orbit of <code>q</code> is inputted. Useful if you are making multiple calls to <code>bqf_is_equiv_set</code> with the same <code>q</code> but varying sets <code>S</code> . If this is not the case, <code>ibqf_isequiv_byS</code> is generally faster.

Name:	long <code>ibqf_isequiv_set_byS</code>
Input:	GEN <code>q</code> , GEN <code>S</code> , GEN <code>rootD</code>
Input format:	PIBQFs <code>q</code> and set of PIBQFs <code>S</code> , all of the same discriminant D , <code>rootD</code> the real square root of D
Output format:	Integer
Description:	Determines if <code>q</code> and an element of <code>S</code> are equivalent or not. Returns 0 if not equivalent, and an index <code>i</code> such that <code>q</code> is equivalent to <code>S[i]</code> otherwise. Generally faster than <code>ibqf_isequiv_set_byq</code> .

Name:	long <code>ibqf_isequiv_set_byS_presorted</code>
Input:	GEN <code>q</code> , GEN <code>Sreds</code> , GEN <code>perm</code> , GEN <code>rootD</code>
Input format:	<code>q</code> , <code>S</code> , and <code>rootD</code> as in <code>ibqf_isequiv_set_byS</code> , where the forms in <code>S</code> are reduced with <code>ibqf_red_pos</code> to get <code>Sreds</code> and sorted by <code>gen_sort_inplace(Sreds, NULL, &bqf_compare, &perm)</code>
Output format:	Integer
Description:	<code>ibqf_isequiv_set_byS</code> where we reduce <code>S</code> and sort it. Useful if you are making multiple calls to <code>bqf_is_equiv_set</code> with the same set <code>S</code> but varying forms <code>q</code> .

Name:	GEN <code>ibqf_isequiv_set_byS_tmat</code>
Input:	GEN <code>q</code> , GEN <code>S</code> , GEN <code>rootD</code>
Input format:	PIBQFs <code>q</code> and set of PIBQFs <code>S</code> , all of the same discriminant D , <code>rootD</code> the real square root of D
Output format:	<code>gen_0</code> , <code>[i, M]</code>
Description:	Determines if <code>q</code> and an element of <code>S</code> are equivalent or not. Returns <code>gen_0</code> if not equivalent, and <code>[i, M]</code> otherwise, where <code>q</code> is equivalent to <code>S[i]</code> with transition matrix <code>M</code> . Generally faster than <code>ibqf_isequiv_set_byq_tmat</code> .

Name:	GEN <code>ibqf_isequiv_set_byS_tmat_presorted</code>
Input:	GEN <code>q</code> , GEN <code>Sreds</code> , GEN <code>perm</code> , GEN <code>rootD</code>
Input format:	<code>q</code> , <code>S</code> , and <code>rootD</code> as in <code>ibqf_isequiv_set_byS</code> , where the forms in <code>S</code> are reduced with <code>ibqf_red_pos</code> to get <code>Sreds</code> and sorted by <code>gen_sort_inplace(Sreds, NULL, &bqf_compare_tmat, &perm)</code>
Output format:	<code>gen_0</code> , <code>[i, M]</code>
Description:	<code>ibqf_isequiv_set_byS_tmat</code> where we reduce <code>S</code> and sort it. Useful if you are making multiple calls to <code>bqf_is_equiv_set</code> with the same <code>q</code> but varying sets <code>S</code> .

Name:	GEN <code>ibqf_isequiv_tmat</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , GEN <code>rootD</code>
Input format:	PIBQFs <code>q1</code> , <code>q2</code> of the same discriminant D , <code>rootD</code> the real square root of D
Output format:	<code>gen_0</code> , matrix
Description:	Determines if <code>q1</code> and <code>q2</code> are equivalent or not, returning a transition matrix if they are and <code>gen_0</code> if not.

Name:	GEN <code>ibqf_red</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	BQF
Description:	Returns a reduction of <code>q</code> .

Name:	GEN <code>ibqf_red_tmat</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> a PBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	<code>[q', M]</code>
Description:	Returns <code>[q', M]</code> , where <code>q'</code> is a reduction of <code>q</code> and <code>M</code> is the transition matrix.

Name:	GEN <code>ibqf_red_pos</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	BQF
Description:	Returns a reduction of <code>q</code> with positive first coefficient.

Name:	GEN <code>ibqf_red_pos_tmat</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> a PBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	<code>[q', M]</code>
Description:	Returns <code>[q', M]</code> , where <code>q'</code> is a reduction of <code>q</code> with positive first coefficient and <code>M</code> is the transition matrix.

3.4 Basic methods for indefinite quadratic forms

Methods in this section are specific to indefinite forms. The “river” is the river of the Conway topograph; it is a periodic ordering of the forms $[A, B, C] \sim q$ with $AC < 0$. Reduced forms with $A > 0$ occur between branches pointing down and up (as we flow along the river), and reduced forms with $A < 0$ occur between branches pointing up and down.

Name:	int <code>ibqf_isrecip</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	0, 1
Description:	Returns 1 if <code>q</code> is reciprocal (<code>q</code> is similar to $-q$), and 0 else.

Name:	int <code>ibqf_isrecip_tc</code>
Input:	GEN <code>q</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF of discriminant D
Output format:	0, 1
Description:	Checks that <code>q</code> is an IBQF, and returns <code>ibqf_isrecip(q, rootD)</code> .

Name:	GEN <code>ibqf_leftnbr</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D with $AC < 0$, <code>rootD</code> the square root of D
Output format:	IBQF
Description:	Returns the left neighbour of <code>q</code> , i.e. the nearest reduced form on the river to the left of <code>q</code> .

Name:	GEN <code>ibqf_leftnbr_tmat</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D with $AC < 0$, <code>rootD</code> the square root of D
Output format:	<code>[q', M]</code>
Description:	Returns <code>[q', M]</code> , with <code>q'</code> the left neighbour of <code>q</code> , and the transition matrix is <code>M</code> .

Name:	GEN <code>ibqf_leftnbr_tc</code>
Input:	GEN <code>q</code> , int <code>tmat</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF of discriminant D with $AC < 0$, <code>tmat</code> =0, 1
Output format:	IBQF or <code>[q', M]</code>
Description:	Checks that <code>q</code> is an IBQF on the river, and returns the left neighbour. If <code>tmat</code> =0 only returns the IBQF, and if <code>tmat</code> =1 returns the form and transition matrix.

Name:	GEN <code>ibqf_leftnbr_update</code>
Input:	GEN <code>qvec</code> , GEN <code>rootD</code>
Input format:	<code>qvec</code> = <code>[q, M]</code> with <code>q</code> an IBQF of discriminant D with $AC < 0$ and $M \in \text{SL}(2, \mathbb{Z})$, <code>rootD</code> the square root of D
Output format:	<code>[q', M']</code>
Description:	Returns <code>[q', M']</code> , where <code>q'</code> is the left neighbour of <code>q</code> , the transition matrix is <code>M''</code> , and <code>M'=MM''</code> . This method makes it easier to apply the left neighbour function multiple times while keeping track of the transition matrix from our original starting point.

Name:	GEN <code>ibqf_redorbit</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns the reduced orbit of <code>q</code> .

Name:	GEN <code>ibqf_redorbit_tmat</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns the reduced orbit of <code>q</code> , where each entry is <code>[q', M]</code> , with the reduced form being <code>q'</code> and the transition matrix from <code>q</code> being <code>M</code> .

Name:	GEN <code>ibqf_redorbit_posonly</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns the reduced orbit of <code>q</code> , where we only keep the BQFs with positive first coefficient.

Name:	GEN <code>ibqf_redorbit_posonly_tmat</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns the reduced orbit with positive first coefficient of <code>q</code> , where each entry is $[q', M]$, with the reduced form being q' and the transition matrix from <code>q</code> being M .

Name:	GEN <code>ibqf_redorbit_tc</code>
Input:	GEN <code>q</code> , int <code>tmat</code> , int <code>posonly</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF, <code>tmat</code> , <code>posonly</code> =0, 1
Output format:	Vector
Description:	Returns the reduced orbit of <code>q</code> . If <code>tmat</code> =1 each entry is the pair $[q', M]$ of form and transition matrix, otherwise each entry is just the form. If <code>posonly</code> =1, we only take the reduced forms with positive first coefficient (half of the total), otherwise we take all reduced forms.

Name:	GEN <code>ibqf_rightnbr</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D with $AC < 0$, <code>rootD</code> the square root of D
Output format:	IBQF
Description:	Returns the right neighbour of <code>q</code> , i.e. the nearest reduced form on the river to the right of <code>q</code> .

Name:	GEN <code>ibqf_rightnbr_tmat</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D with $AC < 0$, <code>rootD</code> the square root of D
Output format:	$[q', M]$
Description:	Returns $[q', M]$, with q' the right neighbour of <code>q</code> , and the transition matrix is M .

Name:	GEN <code>ibqf_rightnbr_tc</code>
Input:	GEN <code>q</code> , int <code>tmat</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF of discriminant D with $AC < 0$, <code>tmat</code> =0, 1
Output format:	IBQF or [<code>q'</code> , <code>M</code>]
Description:	Checks that <code>q</code> is an IBQF on the river, and returns the right neighbour. If <code>tmat</code> =0 only returns the IBQF, and if <code>tmat</code> =1 returns the form and transition matrix.

Name:	GEN <code>ibqf_rightnbr_update</code>
Input:	GEN <code>qvec</code> , GEN <code>rootD</code>
Input format:	<code>qvec</code> =[<code>q</code> , <code>M</code>] with <code>q</code> an IBQF of discriminant D with $AC < 0$ and $M \in \text{SL}(2, \mathbb{Z})$, <code>rootD</code> the square root of D
Output format:	[<code>q'</code> , <code>M'</code>]
Description:	Returns [<code>q'</code> , <code>M'</code>], where <code>q'</code> is the right neighbour of <code>q</code> , the transition matrix is <code>M''</code> , and <code>M'</code> = <code>MM''</code> . This method makes it easier to apply the right neighbour function multiple times while keeping track of the transition matrix from our original starting point.

Name:	GEN <code>ibqf_river</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns the river sequence associated to <code>q</code> . The entry <code>gen_1</code> indicates going right, and <code>gen_0</code> indicates going left along the river.

Name:	GEN <code>ibqf_river_positions</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns [<code>Lpos</code> , <code>Rpos</code> , <code>riv</code>], where <code>riv</code> is <code>ibqf_river(q)</code> but as a <code>t_VECSMALL</code> , <code>Lpos</code> is a <code>t_VECSMALL</code> of indices <code>i</code> for which <code>riv[i]</code> =0, and <code>Rpos</code> is a <code>t_VECSMALL</code> of indices <code>i</code> for which <code>riv[i]</code> =1.

Name:	GEN <code>ibqf_river_positions_forms</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns [<code>Lpos</code> , <code>Rpos</code> , <code>rivforms</code>], where <code>rivforms</code> is the vector of forms on the river <code>riv</code> in order, <code>Lpos</code> is a <code>t_VECSMALL</code> of indices <code>i</code> for which <code>riv[i]</code> =0, and <code>Rpos</code> is a <code>t_VECSMALL</code> of indices <code>i</code> for which <code>riv[i]</code> =1.

Name:	GEN <code>ibqf_river_tc</code>
Input:	GEN <code>q</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF of discriminant D
Output format:	Vector
Description:	Checks that <code>q</code> is an IBQF and returns <code>ibqf_river(q)</code> .

Name:	GEN <code>ibqf_riverforms</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	Vector
Description:	Returns the forms on the river of <code>q</code> , where we only take the forms with first coefficient positive.

Name:	GEN <code>ibqf_riverforms_tc</code>
Input:	GEN <code>q</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF of discriminant D
Output format:	Vector
Description:	Checks that <code>q</code> is an IBQF, and returns <code>ibqf_riverforms(q)</code> .

Name:	GEN <code>ibqf_symmetricarc</code>
Input:	GEN <code>q</code> , GEN <code>D</code> , GEN <code>rootD</code> , GEN <code>qpell</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the positive square root of D , <code>qpell=pell(D)</code>
Output format:	$[z, \gamma_q(z)]$
Description:	If γ_q is the invariant automorph of q , this computes the complex number z , where z is on the root geodesic of q and $z, \gamma_q(z)$ are symmetric (they have the same imaginary part). This gives a “nice” upper half plane realization of the image of the root geodesic of q on $\text{PSL}(2, \mathbb{Z}) \backslash \mathbb{H}$ (a closed geodesic). However, if the automorph of q is somewhat large, z and $\gamma_q(z)$ will be very close to the x -axis, and this method isn’t very useful.

Name:	GEN <code>ibqf_symmetricarc_tc</code>
Input:	GEN <code>q</code> , long <code>prec</code>
Input format:	<code>q</code> an IBQF of discriminant D
Output format:	$[z, \gamma_q(z)]$
Description:	Checks that <code>q</code> is an IBQF, and returns <code>ibqf_symmetricarc(q, ...)</code> .

Name:	GEN <code>ibqf_toriver</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	IBQF
Description:	Reduces <code>q</code> to the river and returns it. Mostly useful as a supporting method for <code>ibqf_red</code> .

Name:	GEN <code>ibqf_toriver_tmat</code>
Input:	GEN <code>q</code> , GEN <code>rootD</code>
Input format:	<code>q</code> an IBQF of discriminant D , <code>rootD</code> the square root of D
Output format:	<code>[q', M]</code>
Description:	Reduces <code>q</code> to the river and returns the reduction <code>q'</code> and the transition matrix <code>M</code> . Mostly useful as a supporting method for <code>ibqf_red_tmat</code> .

Name:	GEN <code>mat_toibqf</code>
Input:	GEN <code>M</code>
Input format:	$M \in \text{SL}(2, \mathbb{Z})$
Output format:	PBQF
Description:	Output the PBQF corresponding to the equation $M(x)=x$. Typically used when <code>M</code> has determinant 1 and is hyperbolic, so that the output is a PIBQF (this method is inverse to <code>ibqf_automorph_D</code> in this case).

Name:	GEN <code>mat_toibqf_tc</code>
Input:	GEN <code>M</code>
Input format:	$M \in \text{SL}(2, \mathbb{Z})$
Output format:	PBQF
Description:	Checks that <code>M</code> is a 2x2 integral matrix, and returns <code>mat_toibqf(M)</code> . This method does not check that <code>M</code> is hyperbolic or that it has determinant 1.

3.5 Class group and composition of forms

This section deals with class group related computations. To compute the class group we take the built-in PARI methods, which cover the cases when D is fundamental and when the narrow and full class group coincide. For the remaining cases, we “boost up” the full class group to the narrow class group with `bqf_ncgp_nonfundnarrow`.

Name:	GEN <code>bqf_comp</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code>
Input format:	PBQFs <code>q1</code> , <code>q2</code> of the same discriminant
Output format:	PBQF
Description:	Returns the composition of <code>q1</code> and <code>q2</code> .

Name:	GEN <code>bqf_comp_red</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , GEN <code>rootD</code> , int <code>Dsign</code>
Input format:	PBQFs <code>q1</code> , <code>q2</code> of the same discriminant D , <code>rootD</code> the positive square root of D if $D > 0$ (and anything if $D < 0$), <code>Dsign</code> the sign of D
Output format:	PBQF
Description:	Composes <code>q1</code> , <code>q2</code> and returns an equivalent reduced form.

Name:	GEN bqf_comp_tc
Input:	GEN q1, GEN q2, int tored, long prec
Input format:	PBQFs q1, q2 of the same discriminant, tored=0, 1
Output format:	PBQF
Description:	Checks that q1, q2 have the same discriminant, and returns their composition. If tored=1 the form is reduced, otherwise it is not.

Name:	GEN bqf_idelt
Input:	GEN D
Input format:	Discriminant D
Output format:	BQF
Description:	Returns the identity element of discriminant D.

Name:	GEN bqf_ncgp
Input:	GEN D, long prec
Input format:	Discriminant D
Output format:	[n, orders, forms]
Description:	Computes and returns the narrow class group associated to D. n is the order of the group, orders=[d1, d2, ..., dk] where $d_1 \mid d_2 \mid \dots \mid d_k$ and the group is isomorphic to $\prod_{i=1}^k \frac{\mathbb{Z}}{d_i \mathbb{Z}}$, and forms is the length k vector of PBQFs corresponding to the decomposition (so forms[i] has order di).

Name:	GEN bqf_ncgp_lexic
Input:	GEN D, long prec
Input format:	Discriminant D
Output format:	[n, orders, forms]
Description:	Computes and returns the narrow class group associated to D. The output is the same as bqf_ncgp, except the third output is now a lexicographical listing of representatives of all equivalence classes of forms of discriminant D (instead of the generators).

Name:	GEN bqf_pow
Input:	GEN q, GEN n
Input format:	PBQF q, integer n
Output format:	PBQF
Description:	Returns q^n .

Name:	GEN bqf_pow_red
Input:	GEN q, GEN n, GEN rootD, int Dsign
Input format:	PBQF q of discriminant D, integer n, rootD the positive square root of D if $D > 0$ (and anything if $D < 0$), Dsign the sign of D
Output format:	PBQF
Description:	Returns a reduction of q^n .

Name:	GEN bqf_pow_tc
Input:	GEN q, GEN n, int tored, long prec
Input format:	PBQF q, integer n, tored=0, 1
Output format:	PBQF
Description:	Checks that q is a PBQF and n is an integer, and returns a form equivalent to q^n . If tored=1 the form is reduced, otherwise it is not necessarily.

Name:	GEN bqf_square
Input:	GEN q
Input format:	PBQF q
Output format:	PBQF
Description:	Returns q^2 .

Name:	GEN bqf_square_red
Input:	GEN q, GEN rootD, int Dsign
Input format:	PBQF q of discriminant D, rootD the positive square root of D if $D > 0$ (and anything if $D < 0$), Dsign the sign of D
Output format:	PBQF
Description:	Returns a reduction of q^2 .

Name:	GEN bqf_square_tc
Input:	GEN q, int tored, long prec
Input format:	PBQF q, tored=0, 1
Output format:	PBQF
Description:	Checks q is a PBQF, and returns a form equivalent to q^2 . If tored=1 the form is reduced, otherwise it is not necessarily.

Name:	GEN bqf_ncgp_nonfundnarrow
Input:	GEN cgp, GEN D, GEN rootD
Input format:	cgp=quadclassunit0(D, 0, NULL, prec), D a positive (typically non-fundamental) discriminant with norm of the fundamental unit being 1, rootD the positive square root of D
Output format:	[n, orders, forms]
Description:	With the described conditions, the narrow class group is twice the size of the class group. Since quadclassunit0 computes the class group, this method modifies the output to computing the full narrow class group, and returns it in the same format as bqf_ncgp.

3.6 Representation of integers by forms - description tables

This section deals with questions of representing integers by quadratic forms. The three main problems we solve are

- Find all integral solutions (X, Y) to $AX^2 + BXY + CY^2 = n$ (bqf_reps);

- Find all integral solutions (X, Y) to $AX^2 + BXY + CY^2 + DX + EY = n$ (`bqf_bigreps`);
- Find all integral solutions (X, Y, Z) to $AX^2 + BY^2 + CZ^2 + DXY + EXZ + FYZ = n_1$ and $UX + VY + WZ = n_2$ (`bqf_linearsolve`).

The general solution descriptions have a lot of cases, so we put the descriptions in Tables 1-3, and refer to the tables in the method descriptions.

For `bqf_reps`, let $q = [A, B, C]$ and let $d = B^2 - 4AC$. If there are no solutions the method will return `gen_0`, and otherwise it will return a vector \mathbf{v} , where

$$v = [[\text{type}, v_{extra}], v_1, v_2, \dots, v_k].$$

The types are are

$$-1=\text{all}, 0=\text{finite}, 1=\text{positive}, 2=\text{linear}.$$

Each (family of) solution(s) is given by a v_i , possibly with reference to the extra data. In this table we will only describe **half** of all solutions: we are only taking one of (X, Y) and $(-X, -Y)$. If you want all solutions without this restriction, you just have to add in these negatives.

Table 1: General solution for `bqf_reps`

Type	Conditions to appear	v_{extra}	v_i format	General solution
-1	$q = 0, n = 0$	-	-	X, Y are any integers
0	$d < 0$	-	$[x_i, y_i]$	$X = x_i$ and $Y = y_i$
	$d = \square > 0, {}^a n \neq 0$			
	$d = \boxtimes, {}^a n = 0$			
1	$d = \boxtimes > 0, n \neq 0$	M ^b	$[x_i, y_i]$	$\begin{pmatrix} X \\ Y \end{pmatrix} = M^j \begin{pmatrix} x_i \\ y_i \end{pmatrix}$ for $j \in \mathbb{Z}$
2	$d = 0, n \neq 0$	-	$[[s_i, t_i], [x_i, y_i]]$	$X = x_i + s_i U, Y = y_i + t_i U$ for $U \in \mathbb{Z}$
	$d = \square > 0, n = 0$			

^a \square means square, and \boxtimes means non-square.

^b $M \in \text{SL}(2, \mathbb{Z})$

For `bqf_bigreps`, let $q = [A, B, C, D, E]$ and let $d = B^2 - 4AC$. If there are no solutions the method will return `gen_0`, and otherwise it will return a vector \mathbf{v} , where

$$v = [[\text{type}, v_{extra}], v_1, v_2, \dots, v_k].$$

The types are are

$$-2=\text{quadratic}, -1=\text{all}, 0=\text{finite}, 1=\text{positive}, 2=\text{linear}.$$

Each (family of) solution(s) is given by a v_i , possibly with reference to the extra data.

Table 2: General solution for `bqf_bigreps`

Type	Conditions to appear	v_{extra}	v_i format	General solution
-2	$d = 0$ and condition ^a	-	$[[a_i, b_i, c_i],$ $[e_i, f_i, g_i]]$	$X = a_i U^2 + b_i U + c_i$ and $Y = e_i U^2 + f_i + g_i$ for $U \in \mathbb{Z}$
-1	$q = 0, n = 0$	-	-	X, Y are any integers
0	$d < 0$	-	$[x_i, y_i]$	$X = x_i$ and $Y = y_i$
	$d = \square > 0$, ^b some cases ^c			
1	$d = \boxtimes > 0, n \neq 0$	$M, [s_1, s_2]$ ^d	$[x_i, y_i]$ ^d	$\begin{pmatrix} X \\ Y \end{pmatrix} = M^j \begin{pmatrix} x_i \\ y_i \end{pmatrix} + \begin{pmatrix} s_1 \\ s_2 \end{pmatrix}$ for $j \in \mathbb{Z}$
2	$d = \square > 0$, ^b some cases ^c	-	$[[s_i, t_i], [x_i, y_i]]$	$x = x_i + s_i U, y = y_i + t_i U$ for $U \in \mathbb{Z}$
	$d = 0$, and condition ^e			

^a At least one of $A, B, C \neq 0$ and at least one of $D, E \neq 0$.

^b \square means square, and \boxtimes means non-square.

^c “Some cases” refers to if the translated equation has $n = 0$ or not.

^d $M \in \text{SL}(2, \mathbb{Z})$ and s_1, s_2 are *rational*; they need not be integral. Same for x_i, y_i .

^e $A = B = C = 0$ or $D = E = 0$. In this case, $s_i = s_j$ and $t_i = t_j$ for all i, j in fact.

For `bqf_linearsolve`, let $q = [A, B, C, D, E, F]$, and let $\text{lin} = [U, V, W]$. If there are no solutions the method will return `gen_0`, and otherwise it will return a vector \mathbf{v} , where

$$\mathbf{v} = [[\text{type}, v_{extra}], v_1, v_2, \dots, v_k].$$

The types are are

-2=quadratic, -1=plane, 0=finite, 1=positive, 2=linear.

Each (family of) solution(s) is given by a v_i , possibly with reference to the extra data.

Table 3: General solution for `bqf_linearsolve`

Type	v_{extra}	v_i format	General solution
-2	-	$[[x_1, x_2, x_3], [y_1, y_2, y_3], [z_1, z_2, z_3]]$	$X = x_1U^2 + x_2U + x_3,$ $Y = y_1U^2 + y_2U + y_3,$ $Z = z_1U^2 + z_2U + z_3, \text{ for } U \in \mathbb{Z}$
-1	-	$[[a_1, a_2, a_3], [b_1, b_2, b_3], [c_1, c_2, c_3]]^a$	$X = a_1U + b_1V + c_1$ $Y = a_2U + b_2V + c_2,$ $Z = a_3U + b_3V + c_3, \text{ for } U, V \in \mathbb{Z}$
0	-	$[a_i, b_i, c_i]$	$X = a_i, Y = b_i, \text{ and } Z = c_i$
1	$M, [s_1, s_2, s_3]^b$	$[a_i, b_i, c_i]^b$	$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = M^j \begin{pmatrix} a_i \\ b_i \\ c_i \end{pmatrix} + \begin{pmatrix} s_1 \\ s_2 \\ s_3 \end{pmatrix} \text{ for } j \in \mathbb{Z}$
2	-	$[[a_1, a_2, a_3], [b_1, b_2, b_3]]$	$X = a_1U + b_1,$ $Y = a_2U + b_2,$ $Z = a_3U + b_3, \text{ for } U \in \mathbb{Z}$

^a In fact, $i = 1$ necessarily (there is one plane only).

^b $M \in \text{SL}(3, \mathbb{Z})$ and s_1, s_2, s_3 are *rational*; they need not be integral. Same for a_i, b_i, c_i .

3.7 Representation of integers by forms - methods

Name:	GEN <code>bqf_bigreps</code>
Input:	GEN <code>q</code> , GEN <code>n</code> , long <code>prec</code>
Input format:	<code>q=[A, B, C, D, E]</code> length 5 integer vector, <code>n</code> integer
Output format:	<code>gen_0</code> or <code>v=[[type, data], sol1, ...]</code>
Description:	Solves $AX^2 + BXY + CY^2 + DX + EY = n$, and returns ALL solutions. If no solutions returns <code>gen_0</code> ; otherwise <code>v[1][1]</code> gives the format of the general solution in Table 2.

Name:	GEN <code>bqf_bigreps_tc</code>
Input:	GEN <code>q</code> , GEN <code>n</code> , long <code>prec</code>
Input format:	<code>q</code> length 5 integer vector, <code>n</code> integer
Output format:	<code>gen_0</code> or <code>v=[[type, data], sol1, ...]</code>
Description:	Checks that <code>q</code> , <code>n</code> have the correct type, and returns <code>bqf_bigreps(q, n, prec)</code> .

Name:	GEN bqf_linearsolve
Input:	GEN q, GEN n1, GEN lin, GEN n2, long prec
Input format:	q=[A, B, C, D, E, F] length 6 integer vector, n1 an integer, lin=[U, V, W] length 3 integer vector, n2 an integer
Output format:	gen_0 or v=[[type, data], sol1, ...]
Description:	Solves $AX^2+BY^2+CZ^2+DXY+EXY+FYZ = n1$ and $UX+VY+WZ = n2$, and returns ALL solutions. If no solutions returns gen_0; otherwise v[1][1] gives the format of the general solution in Table 3.

Name:	GEN bqf_linearsolve_tc
Input:	GEN q, GEN n1, GEN lin, GEN n2, long prec
Input format:	q length 6 integer vector, n1 an integer, lin length 3 integer vector, n2 an integer
Output format:	gen_0 or v=[[type, data], sol1, ...]
Description:	Checks that q, n1, lin, n2 have the correct type, and returns bqf_linearsolve(q, n1, lin, n2).

Name:	GEN bqf_reps
Input:	GEN q, GEN n, int proper, int half, long prec
Input format:	q=[A, B, C] length 3 integer vector, n integer, proper=0, 1, half=0, 1
Output format:	gen_0 or v=[[type, data], sol1, ...]
Description:	Solves $AX^2 + BXY + CY^2 = n$, and returns ALL solutions. If no solutions returns gen_0; otherwise v[1][1] gives the format of the general solution in Table 1. If proper=1 and the form is indefinite/definite, we only output solutions with $\gcd(x,y) = 1$ (otherwise, no restriction). If half=1, only outputs one of (the families corresponding to) (x,y) and $(-x,-y)$, and if half=0 outputs both.

Name:	GEN bqf_reps_tc
Input:	GEN q, GEN n, int proper, int half, long prec
Input format:	q length 3 integer vector, n integer, proper=0, 1, half=0, 1
Output format:	gen_0 or v=[[type, data], sol1, ...]
Description:	Checks that q, n have the correct type, and returns bqf_reps(q, n, proper, half, prec).

Name:	GEN dbqf_reps
Input:	GEN qred, GEN D, GEN n, int proper, int half
Input format:	qred=[q', M] with q' a reduced PDBQF of discriminant D and M the transition matrix, n integer, proper=0, 1, half=0, 1
Output format:	gen_0 or v=[[0], sol1, ...]
Description:	Sub-method solving bqf_reps in the definite case. Useful when you want to call bqf_reps on the same q many times.

Name:	GEN ibqf_reps
Input:	GEN qorb, GEN qautom, GEN D, GEN rootD, GEN n, int proper, int half
Input format:	For q a PIBQF, qorb is the output of iqbf_redorbit_posonly_tmat(q) sorted with bqf_compare_tmat, qautom the automorph of q of discriminant D, rootD the positive real square root of D, n an integer, proper=0, 1, half=0, 1
Output format:	gen_0 or [[0], [0, 0]] or v=[[1, M], sol1, ...]
Description:	Sub-method solving bqf_reps in the indefinite case. Useful when you want to call bqf_reps on the same q many times.

Name:	GEN sbqf_reps
Input:	GEN q, GEN D, GEN rootD, GEN n, int half
Input format:	q a primitive BQF, of positive square discriminant D, rootD the positive (integer) square root of D, n an integer, half=0, 1
Output format:	gen_0 or v=[[0/2], sol1, ...]
Description:	Sub-method solving bqf_reps in the positive square discriminant case. Very minimal savings over bqf_reps.

Name:	GEN zbqf_reps
Input:	GEN A, GEN B, GEN n, int half
Input format:	q a primitive non-trivial BQF of discriminant zero expressed as $(AX + BY)^2$ with A, B coprime, n integer, half=0, 1
Output format:	gen_0 or v=[[2], sol1, ...]
Description:	Sub-method solving bqf_reps in the discriminant zero case. Useful when you want to call bqf_reps on the same q many times.

Name:	GEN zbqf_bigreps
Input:	GEN q, GEN n
Input format:	q length 5 primitive integral and non-trivial with q[1], q[3]>0, n integer
Output format:	gen_0 or v=[[+/-2], sol1,]
Description:	Sub-method solving bqf_bigreps in the discriminant zero case. Very minimal savings over bqf_bigreps.

In the following three methods, we have first primitivized q (length 5 integer vector of non-zero discriminant d), and computed a , b for which the substitutions $X = \frac{x+a}{d}$ and $Y = \frac{y+b}{d}$ yield homogenous BQFs of discriminant d .

Name:	GEN bqf_bigreps_creatorvecfin
Input:	GEN newsols, GEN a, GEN b, GEN disc
Input format:	newsols the output of bqf_reps applied to our translated form, a, b the integers used to translate, disc the discriminant
Output format:	gen_0 or v=[[0], sol1, sol2, ...]
Description:	Takes the solutions to the translated form of type 0=finit and picks out only the ones which return to being integral.

Name:	GEN <code>bqf_bigreps_creatervecpos</code>
Input:	GEN <code>newsols</code> , GEN <code>a</code> , GEN <code>b</code> , GEN <code>disc</code>
Input format:	<code>newsols</code> the output of <code>bqf_reps</code> applied to our translated form, <code>a</code> , <code>b</code> the integers used to translate, <code>disc</code> the discriminant
Output format:	<code>gen_0</code> or <code>v=[[1, M, [s1, s2]], sol1, sol2, ...]</code>
Description:	Takes the solutions to the translated form of type 1=positive and picks out only the ones which return to being integral.

Name:	GEN <code>bqf_bigreps_creatervecclin</code>
Input:	GEN <code>newsols</code> , GEN <code>a</code> , GEN <code>b</code> , GEN <code>disc</code>
Input format:	<code>newsols</code> the output of <code>bqf_reps</code> applied to our translated form, <code>a</code> , <code>b</code> the integers used to translate, <code>disc</code> the discriminant
Output format:	<code>gen_0</code> or <code>v=[[2], sol1, sol2, ...]</code>
Description:	Takes the solutions to the translated form of type 2=linear and picks out only the ones which return to being integral.

Name:	GEN <code>bqf_reps_all</code>
Input:	GEN <code>n</code>
Input format:	<code>n</code> integer
Output format:	<code>gen_0</code> or <code>[[-1]]</code>
Description:	Solves <code>bqf_reps</code> for <code>q=0</code> .

Name:	GEN <code>bqf_reps_creatervec</code>
Input:	<code>glist *sols</code> , <code>glist *scale</code> , <code>llist *nsolslist</code> , <code>long *totnsols</code> , <code>long *count</code> , <code>int half</code>
Input format:	See C code
Output format:	Vector
Description:	This creates the return vector given the computed list of solutions to <code>bqf_reps</code> . This does not initialize the first component (the type), and is only useful internally to <code>bqf_reps</code> . The method is not stack clean, but the return is suitable for <code>gerepileupto</code> .

Name:	GEN <code>bqf_reps_creatervec_proper</code>
Input:	<code>glist *sols</code> , <code>long nsols</code> , <code>int half</code>
Input format:	See C code
Output format:	Vector
Description:	This creates the return vector given the computed list of solutions to <code>bqf_reps</code> . This does not initialize the first component (the type), and is only useful internally to <code>bqf_reps</code> . It differs to <code>bqf_reps_creatervec</code> in that it deals with the proper solutions only case, and is more efficient in this case. The method is not stack clean, but the return is suitable for <code>gerepileupto</code> .

Name:	GEN bqf_reps_makeprimitive
Input:	GEN q, GEN *n
Input format:	BQF q, pointer to integer n
Output format:	gen_0 or primitive BQF
Description:	We divide through q and n by $\gcd(q)$, update n, and return the new q. If n is no longer an integer (hence no solutions to bqf_reps), we return NULL. This clutters the stack.

Name:	GEN bqf_reps_trivial
Input:	void
Input format:	-
Output format:	[[0], [0, 0]]
Description:	Returns the trivial solution set.

Name:	void bqf_reps_updatesolutions
Input:	glist **sols, long *nsols, GEN *a, GEN *b
Input format:	See C code
Output format:	-
Description:	This method adds a new solution to the glist of solutions, and updates the relevant fields. This does not clutter the stack, but is NOT gerepile safe as the vector is created after the components. This is OK for the internal purposes of bqf_reps as the conversion from glist to vector includes a copying.

Name:	void dbqf_reps_proper
Input:	GEN qred, GEN D, GEN n, glist **sols, long *nsols, GEN f, int *terminate
Input format:	See C code
Output format:	-
Description:	This solves the proper representation case of dbqf_reps, updating the solution glist. Internal function for dbqf_reps.

Name:	void ibqf_reps_proper
Input:	GEN qorb, GEN D, GEN rootD, GEN n, glist **sols, long *nsols, GEN f, int *terminate
Input format:	See C code
Output format:	-
Description:	This solves the proper representation case of ibqf_reps, updating the solution glist. Internal function for dbqf_reps.

The following 5 methods all deal with `bqf_linearsolve`. We take a 3×3 matrix M with inverse $Minv$ such that the top row is equal to `lin`, and substitute in $[x;y;z]=M*[X;Y;Z]$. This new equation has solutions $x=n2$ and y, z described by `yzsols`. The methods bump this back to solutions for X, Y, Z , depending on the nature of the y, z solutions.

Name:	GEN <code>bqf_linearsolve_zall</code>
Input:	GEN <code>yzsols</code> , GEN <code>n2</code> , GEN <code>Minv</code>
Input format:	As above
Output format:	<code>[[-1], [[a1, a2, a3], [b1, b2, b3], [c1, c2, c3]]]</code>
Description:	As above, where the type of <code>yzsols</code> is -1, i.e. anything.

Name:	GEN <code>bqf_linearsolve_zfin</code>
Input:	GEN <code>yzsols</code> , GEN <code>n2</code> , GEN <code>Minv</code>
Input format:	As above
Output format:	<code>[[0], sol1, ...]</code>
Description:	As above, where the type of <code>yzsols</code> is 0, i.e. finite.

Name:	GEN <code>bqf_linearsolve_zlin</code>
Input:	GEN <code>yzsols</code> , GEN <code>n2</code> , GEN <code>Minv</code>
Input format:	As above
Output format:	<code>[[2], sol1, ...]</code>
Description:	As above, where the type of <code>yzsols</code> is 2, i.e. linear.

Name:	GEN <code>bqf_linearsolve_zpos</code>
Input:	GEN <code>yzsols</code> , GEN <code>n2</code> , GEN <code>Minv</code> , GEN <code>M</code>
Input format:	As above
Output format:	<code>[[1, M, [s1, s2, s3]], sol1, ...]</code>
Description:	As above, where the type of <code>yzsols</code> is 1, i.e. positive.

Name:	GEN <code>bqf_linearsolve_zquad</code>
Input:	GEN <code>yzsols</code> , GEN <code>n2</code> , GEN <code>Minv</code>
Input format:	As above
Output format:	<code>[[-2], sol1, ...]</code>
Description:	As above, where the type of <code>yzsols</code> is -2, i.e. quadratic.

3.8 Checking GP inputs

Most methods in the library are easily breakable by having bad inputs. When working in GP, we do not want to cause segmentation faults, so we define wrapper functions to check the inputs. The methods in this section are useful in making these wrapper functions.

Name:	<code>void bqf_check</code>
Input:	GEN <code>q</code>
Input format:	-
Output format:	-
Description:	Checks that <code>q</code> is a BQF, and produces an error if not. Useful for making sure that GP inputs do not break our PARI functions.

Name:	GEN bqf_checkdisc
Input:	GEN q
Input format:	-
Output format:	Integer
Description:	Checks that q is a BQF with non-square discriminant and produces an error if not, where we return the discriminant of q if it passes. Useful for making sure that GP inputs do not break our PARI functions.

Name:	void intmatrix_check
Input:	GEN mtx
Input format:	-
Output format:	-
Description:	Checks that mtx is a 2x2 integral matrix, and produces an error if not. Useful for making sure that GP inputs do not break our PARI functions.

4 qq_bqf_int

Methods in this section deal with the intersection of primitive binary quadratic forms.

4.1 Intersection Data

This section deals with data related to an intersecting pair of quadratic forms.

Name:	GEN bqf_bdelta
Input:	GEN q1, GEN q2
Input format:	q1 and q2 integral BQFs
Output format:	Integer
Description:	Let $q_i = [A_i, B_i, C_i]$, and this method returns $B_\Delta(q_1, q_2) = B_1B_2 - 2A_1C_2 - 2A_2C_1$.

Name:	GEN bqf_bdelta_tc
Input:	GEN q1, GEN q2
Input format:	q1 and q2 integral BQFs
Output format:	Integer
Description:	Checks that q1, q2 have the correct type, and returns bqf_bdelta(q1, q2).

Name:	GEN bqf_intlevel
Input:	GEN q1, GEN q2
Input format:	q1 and q2 integral BQFs
Output format:	Integer
Description:	Returns the signed intersection level of q1, q2.

Name:	GEN bqf_intlevel_tc
Input:	GEN q1, GEN q2
Input format:	q1 and q2 integral BQFs
Output format:	Integer
Description:	Checks that q1, q2 have the correct type, and returns bqf_intlevel(q1, q2).

Name:	GEN ibqf_intpairs_transtoq
Input:	GEN pairs, GEN q, GEN rootD
Input format:	pairs a set of pairs of integral IBQFs, q an IBQF that is similar to the first element of every entry in pairs, rootD the square root of the discriminant of disc(q).
Output format:	Vector of pairs [q, q']
Description:	For each pair [q1, q2] in pairs, simultaneously conjugates the pair to [q, q'], and returns the vector of transformed pairs.

Name:	GEN ibqf_intpoint
Input:	GEN q1, GEN q2, GEN location, GEN autom
Input format:	q1 and q2 IBQFs with intersecting root geodesics, location is 0, 1, or a complex point on ℓ_{q1} , autom the invariant automorph of q1 (only required if location $\neq 0, 1$).
Output format:	Imaginary t_QUAD
Description:	Outputs the upper half plane intersection point of q1, q2. If location=0, it is the intersection of q1, q2; if location=1, we translate it to the fundamental domain of $\text{PSL}(2, \mathbb{Z})$; if $\text{imag}(\text{location}) \neq 0$, then location is assumed to be a point on ℓ_{q1} . We translate the intersection point to the geodesic between location and $\gamma_{q1}(\text{location})$. If the invariant automorph is large, then we need to increase the precision to ensure accurate results.

Name:	GEN ibqf_intpoint_tc
Input:	GEN q1, GEN q2, GEN location
Input format:	q1 and q2 IBQFs with intersecting root geodesics, location is 0, 1, or a complex point on ℓ_{q1}
Output format:	Imaginary t_QUAD
Description:	Checks that q1, q2 are the correct type and that location=0, 1, or is a complex upper half plane point, and returns ibqf_intpoint(q1, q2, location, autom).

Name:	GEN bqf_iform
Input:	GEN q1, GEN q2
Input format:	q1 and q2 IBQFs
Output format:	IBQF
Description:	For $q_i=[A_i,B_i,C_i]$, this returns $[-A_1B_2 + A_2B_1, -2A_1C_2 + 2A_2C_1, -B_1C_2 + B_2C_1]$.

4.2 Intersection number computation

This section deals with the computation of the intersection number of PIBQFs.

Name:	GEN ibqf_int
Input:	GEN r1, GEN r2
Input format:	r1, r2 the output of <code>ibqf_river_positions(qi)</code>
Output format:	Integer
Description:	Computes the full intersection number of q1, q2, which correspond to the river data r1, r2.

Name:	GEN ibqf_int_tc
Input:	GEN q1, GEN q2, long prec
Input format:	q1, q2 PIBQFs
Output format:	Integer
Description:	Checks that q1, q2 are PIBQFs, and returns their full intersection number.

Name:	GEN ibqf_intrS_byriver
Input:	GEN r1, GEN r2
Input format:	r1, r2 the output of <code>ibqf_river_positions(qi)</code>
Output format:	Integer
Description:	Computes the RS-intersection number of q1, q2, which correspond to the river data r1, r2.

Name:	GEN ibqf_intrS_tc
Input:	GEN q1, GEN q2, long prec
Input format:	q1, q2 PIBQFs
Output format:	Integer
Description:	Checks that q1, q2 are PIBQFs, and returns their RS-intersection.

Name:	GEN <code>ibqf_intforms_byriver</code>
Input:	GEN <code>r1</code> , GEN <code>r2</code> , int <code>data</code> , int <code>split</code>
Input format:	<code>r1</code> , <code>r2</code> the output of <code>ibqf_river_positions(qi)</code> , <code>data</code> =0, 1, <code>split</code> =0, 1
Output format:	Vector
Description:	Outputs the intersecting forms of <code>q1</code> , <code>q2</code> of all types. If <code>data</code> =1, each entry of the output is <code>[B_delta(f1,f2), level of int, length of river overlap, f1, f2]</code> ; otherwise it is just the pair <code>[f1, f2]</code> . If <code>split</code> =0 outputs a single vector of the return data, and if <code>split</code> =1 it splits the output into <code>[[RS], [RO], [LS], [LO]]</code> intersection.

Name:	GEN <code>ibqf_intforms_tc</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , int <code>data</code> , int <code>split</code> , long <code>prec</code>
Input format:	<code>q1</code> , <code>q2</code> PIBQFs, <code>data</code> =0, 1, <code>split</code> =0, 1
Output format:	Vector
Description:	Checks the inputs and returns <code>ibqf_intforms_byriver</code> with the river data of <code>q1</code> , <code>q2</code> .

Name:	GEN <code>ibqf_intformsRS_byriver</code>
Input:	GEN <code>r1</code> , GEN <code>r2</code> , int <code>data</code>
Input format:	<code>r1</code> , <code>r2</code> the output of <code>ibqf_river_positions(qi)</code> , <code>data</code> =0, 1
Output format:	Vector
Description:	<code>ibqf_intforms_byriver</code> , except we only compute the forms for the RS-intersection.

Name:	GEN <code>ibqf_intformsRS_tc</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , int <code>data</code> , long <code>prec</code>
Input format:	<code>q1</code> , <code>q2</code> PIBQFs, <code>data</code> =0, 1
Output format:	Vector
Description:	Checks the inputs, computes the rivers, and returns <code>ibqf_intformsRS_byriver</code> .

Name:	GEN <code>ibqf_intformsRO_byriver</code>
Input:	GEN <code>r1</code> , GEN <code>r2</code> , int <code>data</code>
Input format:	<code>r1</code> , <code>r2</code> the output of <code>ibqf_river_positions(qi)</code> , <code>data</code> =0, 1
Output format:	Vector
Description:	<code>ibqf_intforms_byriver</code> , except we only compute the forms for the RO-intersection. Uses <code>Int_RO(q1, q2)=Int_RS(-q2, q1)</code> .

Name:	GEN <code>ibqf_intformsR0_tc</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , int <code>data</code> , long <code>prec</code>
Input format:	<code>q1</code> , <code>q2</code> PIBQFs, <code>data=0</code> , <code>1</code>
Output format:	Vector
Description:	Checks the inputs, computes the rivers, and returns <code>ibqf_intformsR0_byriver</code> .

Name:	GEN <code>ibqf_intformsLS_byriver</code>
Input:	GEN <code>r1</code> , GEN <code>r2</code> , int <code>data</code>
Input format:	<code>r1</code> , <code>r2</code> the output of <code>ibqf_river_positions(qi)</code> , <code>data=0</code> , <code>1</code>
Output format:	Vector
Description:	<code>ibqf_intforms_byriver</code> , except we only compute the forms for the LS-intersection. Uses <code>Int_LS(q1, q2)=Int_RS(q2, q1)</code> .

Name:	GEN <code>ibqf_intformsLS_tc</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , int <code>data</code> , long <code>prec</code>
Input format:	<code>q1</code> , <code>q2</code> PIBQFs, <code>data=0</code> , <code>1</code>
Output format:	Vector
Description:	Checks the inputs, computes the rivers, and returns <code>ibqf_intformsLS_byriver</code> .

Name:	GEN <code>ibqf_intformsLO_byriver</code>
Input:	GEN <code>r1</code> , GEN <code>r2</code> , int <code>data</code>
Input format:	<code>r1</code> , <code>r2</code> the output of <code>ibqf_river_positions(qi)</code> , <code>data=0</code> , <code>1</code>
Output format:	Vector
Description:	<code>ibqf_intforms_byriver</code> , except we only compute the forms for the LO-intersection. Uses <code>Int_LO(q1, q2)=Int_RS(q1, -q2)</code> .

Name:	GEN <code>ibqf_intformsLO_tc</code>
Input:	GEN <code>q1</code> , GEN <code>q2</code> , int <code>data</code> , long <code>prec</code>
Input format:	<code>q1</code> , <code>q2</code> PIBQFs, <code>data=0</code> , <code>1</code>
Output format:	Vector
Description:	Checks the inputs, computes the rivers, and returns <code>ibqf_intformsLO_byriver</code> .

Name:	GEN <code>ibqf_int_reverseriver</code>
Input:	GEN <code>r</code>
Input format:	<code>r</code> the output of <code>ibqf_river_positions(q)</code>
Output format:	Vector
Description:	Computes the river corresponding to $-q$, i.e. reverses the river.

Name:	GEN <code>ibqf_intRS_splitindices</code>
Input:	GEN <code>river</code> , GEN <code>ind</code>
Input format:	<code>river</code> the <code>t_VECSMALL</code> representing the river of <code>q</code> , and <code>ind</code> a sorted <code>t_VECSMALL</code> of indices of <code>river</code>
Output format:	<code>[t_VECSMALL, t_VECSMALL]</code>
Description:	Splits the indices into those which are 0 and those which are 1, adds 1 to the indices, and returns them.

Name:	void <code>ibqf_intformsRS_byriver_indices</code>
Input:	GEN <code>r1</code> , GEN <code>r2</code> , llist <code>**inds1</code> , llist <code>**inds2</code> , llist <code>**looverlap</code> , long <code>*inum</code> , int <code>data</code>
Input format:	See C code
Output format:	
Description:	The computation of the the RS-intersection forms called in <code>ibqf_intformsRS_byriver</code> .

5 qq_geometry

These methods deal with geometry, typically Euclidean or hyperbolic. They are heavily used in the computation of fundamental domains for Shimura curves.

There are five main objects in play: points, lines, line segments, circles, and circle arcs.

- Point: `p`, a complex number.
- Line:

$$l=[\text{slope}, \text{intercept}, 1]$$

If `slope` is not ∞ , the line is $y=\text{slope}*x+\text{intercept}$. If `slope`= ∞ , `intercept` is actually the $x-\text{intercept}$, and the line has equation $x=\text{intercept}$. The final 1 is to distinguish it from a circle.

- Line segment:

$$l=[\text{slope}, \text{intercept}, \text{startpt}, \text{endpt}, 0, \text{ooendptdor}, \text{dir}, 1]$$

The `slope`, `intercept`, and final 1 are the same as a line. `startpt` and `endpt` are the start and endpoints of the segment. If `dir`=1, this is the segment in the plane, and if `dir`=-1, this is the segment through the point at ∞ . If one of the endpoints is ∞ , then `dir`=0, and we instead consider `ooendptdor`. If this is 1, then the segment travels vertically upward or to the right, and if it is -1, the segment travels vertically down or to the left. This is set of 0 if neither endpoint is ∞ . The 0 is meaningless and used to match the format of circle arcs.

- Circle:

$$c=[\text{centre}, \text{radius}, 0]$$

The final 0 is to distinguish it from a line.

- Circle arc:

```
c=[centre, radius, start pt, end pt, start angle, end angle, dir, 0]
```

The arc runs along the circle defined by `centre`, `radius`, and is defined by the counterclockwise arc from `start pt` to `end pt`. The corresponding radial angles are `start angle` and `end angle`. If `dir=1`, the arc is oriented counterclockwise, whereas if `dir=-1`, it is clockwise (and so runs from `end pt` to `start pt` in a clockwise fashion). The final 0 is to distinguish it from a line segment.

Note that there is a small dichotomy between line segments and circle arcs: segments always start at the start point, whereas the start point of a circle arc defines the first point when traveling in a counterclockwise direction; if `dir=-1` the arc actually is oriented to start at `end pt`.

The main methods available include:

- Initializing lines from slope/point and two points;
- Initializing circles from centre/radius and three points;
- Computing the image of lines/segments/circles/arcs under Möbius maps;
- Computing the intersection points of pairs of lines/segments/circles/arcs.

Throughout this section we often use a variable `tol`, denoting the tolerance. When doing computations with inexact real/complex numbers, sometimes rounding errors will cause issues. The main concerns in this department include:

- Tangent circles/tangent line to a circle; we only want to have 1 intersection point, not 2 or 0;
- Determining if the endpoint of a segment/arc is on the segment/arc;
- If the image of a circle/line under a mobius map is a line, we want to correctly identify it as such (and not as a circle with a massive radius).

To this end, we declare quantities `x`, `y` to be equal if they differ by at most `tol`. The default value of `tol` is set to one quarter the precision. Of course, rounding can eventually cause unequal points to be declared as equal. If this ends up being an issue, increase the precision and make the tolerance smaller until things work.

5.1 Basic line, circle, and point operations

These functions deal with the creation of circles/lines, as well as basic operations involving one such object.

Name:	GEN <code>arc_init</code>
Input:	GEN <code>c</code> , GEN <code>p1</code> , GEN <code>p2</code> , int <code>dir</code> , long <code>prec</code>
Input format:	Circle <code>c</code> , points <code>p1</code> , <code>p2</code> on <code>c</code> , <code>dir=-1</code> , 0, 1
Output format:	Circle arc
Description:	Initializes the circle arc on the counterclockwise segment going from <code>p1</code> to <code>p2</code> on <code>c</code> , oriented counterclockwise if <code>dir=1</code> , clockwise if <code>dir=-1</code> , and unoriented if <code>dir=0</code> .

Name:	GEN arc_init_tc
Input:	GEN c, GEN p1, GEN p2, int dir, long prec
Input format:	Circle c, points p1, p2 on c, dir=-1, 0, 1
Output format:	Circle arc
Description:	Checks that c is a circle and returns arc_init(c, p1, p2, dir, prec).

Name:	GEN arc_midpoint
Input:	GEN c, GEN p1, GEN p2, GEN tol, long prec
Input format:	Circle/arc c, points p1, p2 on c
Output format:	Point
Description:	Returns the midpoint of the arc on c between p1 and p2.

Name:	GEN arc_midpoint_tc
Input:	GEN c, GEN p1, GEN p2, long prec
Input format:	Circle/arc c, points p1, p2 on c
Output format:	Point
Description:	Checks that c is a circle/arc, sets the tolerance, and calls arc_midpoint.

Name:	GEN circle_angle
Input:	GEN c1, GEN c2, GEN p, GEN tol, long prec
Input format:	Circle/arcs c1, c2, intersection point p
Output format:	Angle
Description:	Returns the angle formed by rotating the tangent line to c1 at p counter-clockwise to the tangent to c2 at p.

Name:	GEN circle_angle_tc
Input:	GEN c1, GEN c2, GEN p, long prec
Input format:	Circle/arcs c1, c2, intersection point p
Output format:	Angle
Description:	Checks that c1, c2 are circles, sets the default tolerance, and returns circle_angle(c1, c2, p, tol, prec).

Name:	GEN circle_fromcp
Input:	GEN cent, GEN p, long prec
Input format:	Points cent, p
Output format:	Circle
Description:	Initializes a circle with given centre cent that passes through a point p.

Name:	GEN <code>circle_fromppp</code>
Input:	GEN <code>p1</code> , GEN <code>p2</code> , GEN <code>p3</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Points <code>p1</code> , <code>p2</code> , <code>p3</code>
Output format:	Circle
Description:	Initializes a circle that passes through <code>p1</code> , <code>p2</code> , <code>p3</code> . If they are collinear or one of them is ∞ , then returns the corresponding line instead.

Name:	GEN <code>circle_fromppp_tc</code>
Input:	GEN <code>p1</code> , GEN <code>p2</code> , GEN <code>p3</code> , long <code>prec</code>
Input format:	Points <code>p1</code> , <code>p2</code> , <code>p3</code>
Output format:	Circle
Description:	Initializes a circle that passes through <code>p1</code> , <code>p2</code> , <code>p3</code> . If they are collinear or one of them is ∞ , then returns the corresponding line instead.

Name:	GEN <code>circle_tangentslope</code>
Input:	GEN <code>c</code> , GEN <code>p</code> , long <code>prec</code>
Input format:	Circle/arc <code>c</code> , point <code>p</code>
Output format:	$\mathbb{R} \cup \infty$
Description:	Returns the slope of the tangent line to <code>c</code> at <code>p</code> .

Name:	GEN <code>circle_tangentslope_tc</code>
Input:	GEN <code>c</code> , GEN <code>p</code> , long <code>prec</code>
Input format:	Circle/arc <code>c</code> , point <code>p</code>
Output format:	$\mathbb{R} \cup \infty$
Description:	Checks that <code>c</code> is a circle/arc, and calls <code>circle_tangentslope</code> .

Name:	GEN <code>crossratio</code>
Input:	GEN <code>a</code> , GEN <code>b</code> , GEN <code>c</code> , GEN <code>d</code>
Input format:	<code>a</code> , <code>b</code> , <code>c</code> , <code>d</code> complex numbers or infinity, with at most one being infinity
Output format:	Complex number or $\pm\infty$
Description:	Returns the crossratio $[a,b;c,d]$.

Name:	GEN <code>line_angle</code>
Input:	GEN <code>l1</code> , GEN <code>l2</code> , long <code>prec</code>
Input format:	Lines/segments <code>l1</code> , <code>l2</code>
Output format:	angle in $[0, \pi)$
Description:	Returns the angle formed by rotating <code>l1</code> counterclockwise to be parallel to <code>l2</code> .

Name:	GEN line_fromsp
Input:	GEN s, GEN p
Input format:	s real or ∞ , p point
Output format:	Line
Description:	Returns the line with slope s passing through p.

Name:	GEN line_frompp
Input:	GEN p1, GEN p2
Input format:	Points p1, p2
Output format:	Line
Description:	Returns the line passing through p1, p2.

Name:	GEN mat_eval
Input:	GEN M, GEN x
Input format:	M a 2x2 matrix and x a complex number or infinity
Output format:	Complex number or $\pm\infty$
Description:	Returns M acting on x via Mobius transformation.

Name:	GEN mat_eval_tc
Input:	GEN M, GEN x
Input format:	M a 2x2 matrix and x a complex number or infinity
Output format:	Complex number or $\pm\infty$
Description:	Checks that M is a 2x2 matrix, and returns mat_eval(M, x).

Name:	GEN midpoint
Input:	GEN p1, GEN p2
Input format:	Points p1, p2
Output format:	Point
Description:	Returns the midpoint of p1, p2.

Name:	GEN mobius
Input:	GEN M, GEN c, GEN tol, long prec
Input format:	2x2 real matrix M, circle/arc/line/segment c
Output format:	Circle/arc/line/segment
Description:	Returns Mc.

Name:	GEN mobius_tc
Input:	GEN M, GEN c, long prec
Input format:	2x2 real matrix M, circle/arc/line/segment c
Output format:	Circle/arc/line/segment
Description:	Checks that M is a 2x2 matrix, sets the tolerance, and calls mobius.

Name:	GEN mobius_arcseg
Input:	GEN M, GEN c, int isarc, GEN tol, long prec
Input format:	2x2 real matrix M, arc/segment c, isarc=1 if arc and =0 if segment
Output format:	arc/segment
Description:	mobius for arcs and segments.

Name:	GEN mobius_circle
Input:	GEN M, GEN c, GEN tol, long prec
Input format:	2x2 real matrix M, circle c
Output format:	Circle/line
Description:	mobius for circles.

Name:	GEN mobius_line
Input:	GEN M, GEN l, GEN tol, long prec
Input format:	2x2 real matrix M, line c
Output format:	Circle/line
Description:	mobius for lines.

Name:	GEN perpbis
Input:	GEN p1, GEN p2
Input format:	Points p1, p2
Output format:	Line
Description:	Returns the perpendicular bisector of p1, p2.

Name:	GEN radialangle
Input:	GEN c, GEN p, GEN tol, long prec
Input format:	Circle/arc c, point p
Output format:	Angle in $[0, 2\pi]$
Description:	Returns the angle formed between the centre of c and p.

Name:	GEN radialangle_tc
Input:	GEN c, GEN p, long prec
Input format:	Circle/arc c, point p
Output format:	Angle in $[0, 2\pi]$
Description:	Checks that c is a circle/arc, sets the default tolerance, and returns radialangle(c, p, tol, prec).

Name:	GEN slope
Input:	GEN p1, GEN p2
Input format:	Points p1, p2
Output format:	$\mathbb{R} \cup \infty$
Description:	Returns the slope of the line between p1 and p2.

5.2 Intersection of lines and circles

These functions deal with the intersections of circles/arcs/lines/segments. The main function could be `genset_int`, which can find the intersection of any pair of the above (to use the other methods you need to know that you have a line and a circle, etc.)

Name:	GEN <code>arc_int</code>
Input:	GEN <code>c1</code> , GEN <code>c2</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Arcs <code>c1</code> , <code>c2</code>
Output format:	Vector
Description:	Returns the intersection points of <code>c1</code> , <code>c2</code> .

Name:	GEN <code>arc_int_tc</code>
Input:	GEN <code>c1</code> , GEN <code>c2</code> , long <code>prec</code>
Input format:	Arcs <code>c1</code> , <code>c2</code>
Output format:	Vector
Description:	Checks that <code>c1</code> , <code>c2</code> are arcs, sets the tolerance, and calls <code>arc_int</code> .

Name:	GEN <code>arcseg_int</code>
Input:	GEN <code>c</code> , GEN <code>l</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Arc <code>c</code> , segment <code>l</code>
Output format:	Vector
Description:	Returns the intersection points of <code>c</code> , <code>l</code> .

Name:	GEN <code>arcseg_int_tc</code>
Input:	GEN <code>c</code> , GEN <code>l</code> , long <code>prec</code>
Input format:	Arc <code>c</code> , segment <code>l</code>
Output format:	Vector
Description:	Checks that <code>c</code> is an arc, <code>l</code> is a segment, sets the tolerance, and calls <code>arcseg_int</code> .

Name:	GEN <code>circle_int</code>
Input:	GEN <code>c1</code> , GEN <code>c2</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Circles <code>c1</code> , <code>c2</code>
Output format:	Vector
Description:	Returns the intersection points of <code>c1</code> , <code>c2</code> .

Name:	GEN <code>circle_int_tc</code>
Input:	GEN <code>c1</code> , GEN <code>c2</code> , long <code>prec</code>
Input format:	Circles <code>c1</code> , <code>c2</code>
Output format:	Vector
Description:	Checks that <code>c1</code> , <code>c2</code> are circles, sets the tolerance, and calls <code>circle_int</code> .

Name:	GEN <code>circleline_int</code>
Input:	GEN <code>c</code> , GEN <code>l</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Circle <code>c</code> , line <code>l</code>
Output format:	Vector
Description:	Returns the intersection points of <code>c</code> , <code>l</code> .

Name:	GEN <code>circleline_int_tc</code>
Input:	GEN <code>c</code> , GEN <code>l</code> , long <code>prec</code>
Input format:	Circle <code>c</code> , line <code>l</code>
Output format:	Vector
Description:	Checks that <code>c</code> is an circle, <code>l</code> is a line, sets the tolerance, and calls <code>circleline_int</code> .

Name:	GEN <code>genseg_int</code>
Input:	GEN <code>s1</code> , GEN <code>s2</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Circle/arc/line/segment <code>s1</code> , <code>s2</code>
Output format:	Vector
Description:	Returns the intersection points of <code>s1</code> , <code>s2</code> .

Name:	GEN <code>genseg_int_tc</code>
Input:	GEN <code>s1</code> , GEN <code>s2</code> , long <code>prec</code>
Input format:	Circle/arc/line/segment <code>s1</code> , <code>s2</code>
Output format:	Vector
Description:	Checks the input types, sets the tolerance, and calls the appropriate intersection method.

Name:	GEN <code>line_int</code>
Input:	GEN <code>l1</code> , GEN <code>l2</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Lines <code>l1</code> , <code>l2</code>
Output format:	Vector
Description:	Returns the intersection points of <code>l1</code> , <code>l2</code> .

Name:	GEN <code>line_int_tc</code>
Input:	GEN <code>l1</code> , GEN <code>l2</code> , long <code>prec</code>
Input format:	Lines <code>l1</code> , <code>l2</code>
Output format:	Vector
Description:	Checks that <code>l1</code> , <code>l2</code> are lines, sets the tolerance, and calls <code>line_int</code> .

Name:	int onarc
Input:	GEN c, GEN p, GEN tol, long prec
Input format:	Arc c, point p
Output format:	0, 1
Description:	Returns 1 if p is on the arc c, and 0 else (p is assumed to be on the circle defined by c). Accepts c to be a circle, where we return 1.

Name:	int onarc_tc
Input:	GEN c, GEN p, long prec
Input format:	Arc c, point p
Output format:	0, 1
Description:	Checks that c is an arc/circle, sets the tolerance, and calls onarc.

Name:	int onseg
Input:	GEN l, GEN p, GEN tol, long prec
Input format:	Segment l, point p
Output format:	0, 1
Description:	Returns 1 if p is on the segment l, and 0 else (p is assumed to be on the line defined by l). Accepts l to be a line, where we return 1.

Name:	int onseg_tc
Input:	GEN l, GEN p, long prec
Input format:	Segment l, point p
Output format:	0, 1
Description:	Checks that l is an segment/line, sets the tolerance, and calls onseg.

Name:	GEN seg_int
Input:	GEN l1, GEN l2, GEN tol, long prec
Input format:	Segments l1, l2
Output format:	Vector
Description:	Returns the intersection points of l1, l2.

Name:	GEN seg_int_tc
Input:	GEN l1, GEN l2, long prec
Input format:	Segments l1, l2
Output format:	Vector
Description:	Checks that l1, l2 are segments, sets the tolerance, and calls seg_int.

5.3 Hyperbolic distance and area

Name:	GEN <code>hdist</code>
Input:	GEN <code>z1</code> , GEN <code>z2</code> , long <code>prec</code>
Input format:	Upper half plane complex points <code>z1</code> , <code>z2</code>
Output format:	Distance
Description:	Returns the upper half plane hyperbolic distance between <code>z1</code> and <code>z2</code> .

Name:	GEN <code>hdist_tc</code>
Input:	GEN <code>z1</code> , GEN <code>z2</code> , long <code>prec</code>
Input format:	Upper half plane complex points <code>z1</code> , <code>z2</code>
Output format:	Distance
Description:	Checks that <code>z1</code> , <code>z2</code> are complex points in the upper half plane and returns <code>hdist(z1, z2, prec)</code> .

Name:	GEN <code>hdist_ud</code>
Input:	GEN <code>z1</code> , GEN <code>z2</code> , long <code>prec</code>
Input format:	Unit disc points <code>z1</code> , <code>z2</code>
Output format:	Distance
Description:	Returns the hyperbolic distance between <code>z1</code> , <code>z2</code> in the unit disc model.

Name:	GEN <code>hpolygon_area</code>
Input:	GEN <code>circles</code> , GEN <code>vertices</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Vectors of circles <code>circles</code> , vector of vertices <code>vertices</code>
Output format:	Positive real number or ∞
Description:	Given a hyperbolic polygon in the unit circle model, with side <code>i</code> given by <code>circles[i]</code> and the intersection of <code>circles[i]</code> , <code>circles[i+1]</code> being <code>vertices[i]</code> , this returns the area of the polygon. If there are edges on the unit circle (corresponding to <code>circles[i]=0</code>), the output is ∞ .

Name:	GEN <code>hpolygon_area_tc</code>
Input:	GEN <code>circles</code> , GEN <code>vertices</code> , long <code>prec</code>
Input format:	Vectors of circles <code>circles</code> , vector of vertices <code>vertices</code>
Output format:	Checks that <code>circles</code> , <code>vertices</code> are vectors of the same length, sets the default tolerance, and returns <code>hpolygon_area(circles, vertices, tol, prec)</code> .
Description:	

5.4 Fundamental domain computation

As mentioned earlier, the fundamental domain computation has been upgraded to work with `alginit` in a separate repository. The methods here are depreciated and are left in to work with the rest of this package.

5.5 Helper methods

These are various supporting methods.

Name:	GEN <code>anglediff</code>
Input:	GEN <code>ang</code> , GEN <code>bot</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Angles <code>ang</code> , <code>bot</code>
Output format:	$[0, 2\pi)$
Description:	Returns the angle <code>ang-bot</code> shifted to lie in the range $[0, 2\pi)$.

Name:	GEN <code>atanoo</code>
Input:	GEN <code>x</code> , long <code>prec</code>
Input format:	$x \in \mathbb{R} \cup \infty$
Output format:	Angle in $(-\pi/2, \pi/2]$
Description:	Returns $\arctan(x)$, where <code>x=oo</code> returns $\pi/2$.

Name:	GEN <code>deftol</code>
Input:	long <code>prec</code>
Input format:	
Output format:	Real number
Description:	Returns the default tolerance given the precision.

Name:	int <code>gcmp_strict</code>
Input:	void <code>*data</code> , GEN <code>x</code> , GEN <code>y</code>
Input format:	
Output format:	-1, 1
Description:	Returns <code>gcmp(x, y)</code> , except returns -1 if <code>x==y</code> . Useful for <code>gen_search</code> when you ALWAYS want to return the index to insert the piece of data.

Name:	int <code>geom_check</code>
Input:	GEN <code>c</code>
Input format:	Circle/arc/line/segment <code>c</code>
Output format:	-1, 0, 1, 2, 3
Description:	Returns 0 if <code>c</code> is a circle, 1 if a line, 2 if an arc, 3 if a segment, and -1 if none of the above.

Name:	GEN <code>shiftangle</code>
Input:	GEN <code>ang</code> , GEN <code>bot</code> , long <code>prec</code>
Input format:	Real numbers <code>ang</code> , <code>bot</code>
Output format:	$[\text{bot}, \text{bot}+2\pi)$
Description:	Shifts the angle <code>ang</code> by integer multiples of 2π until it lies in the range $[\text{bot}, \text{bot}+2\pi)$.

Name:	<code>long tolcmp</code>
Input:	GEN <code>x</code> , GEN <code>y</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Reals <code>x</code> , <code>y</code>
Output format:	-1, 0, 1
Description:	Returns -1 if $x < y$, 0 if $x = y$ up to tolerance, and 1 if $x > y$. If <code>x</code> and <code>y</code> are exact objects, will ignore the tolerance.

Name:	<code>int tolcmp_sort</code>
Input:	void <code>*data</code> , GEN <code>x</code> , GEN <code>y</code>
Input format:	<code>data</code> points to <code>[tol, VECSMALL(prec)]</code>
Output format:	-1, 0, 1
Description:	Returns <code>tolcmp(x, y, tol, prec)</code> , and is used to sort/search a list with tolerance.

Name:	<code>int toleq</code>
Input:	GEN <code>x</code> , GEN <code>y</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	Complex <code>x</code> , <code>y</code>
Output format:	0, 1
Description:	Returns 1 if $x = y$ up to tolerance, and 0 else. If <code>x</code> and <code>y</code> are exact objects, will ignore the tolerance.

6 qq_quat

This section deals with the basic function involving quaternion algebras, orders, and elements. Practically, we use the following implementations:

- The quaternion algebra (QA) $B = \left(\frac{a,b}{\mathbb{Q}} \right)$ is stored as the length 3 vector

$$[0, [p_1, \dots, p_{2r}], [a, b, -ab], \mathfrak{D}],$$

where B is ramified at p_1, p_2, \dots, p_{2r} and has discriminant \mathfrak{D} . The first entry of 0 is a placeholder to denote that the base field is \mathbb{Q} .

- An indefinite quaternion algebra is referred to as an IQA.
- An element of a quaternion algebra (Qelt) is stored as a length 4 vector.

$$[e, f, g, h] := e + fi + gj + hk.$$

- A lattice (QL) in a quaternion algebra is stored as a 4x4 matrix whose columns form a basis of the lattice.
- A quaternion order (QO) is a quaternion lattice that happens to be an order. Most methods require an initialized quaternion order (iQO), which is stored as the length 7 vector

$$[O, t, [d_1, d_2, d_3, d_4], \ell, [[p_1, e_1], \dots, [p_n, e_n]], O^{-1}, [b_1, b_2, b_3]].$$

- O is the QL that generates the order;
 - t is the type of the order, which is 0 if maximal, 1 if Eichler and non-maximal, and -1 otherwise;
 - d_i is the maximal denominator of the i^{th} coefficient of an element of the order (in particular, d_1 is 1 or 2 necessarily);
 - $\ell = p_1^{e_1} \cdots p_n^{e_n}$ is the level of the order;
 - The rank three Z -module formed by the elements of trace 0 in O is generated by b_1, b_2, b_3 .
- An Eichler order is denoted as EQO, and an initialized Eichler order is iEQO.

The “standard” functions available include:

- Initialize a quaternion algebra B from the set of primes ramifying, or from a, b ;
- Standard element operations, e.g. multiplication, conjugation, powering, reduced norm, etc.
- Initializing an order based on a set of generators;
- Returning a maximal order/Eichler order of a given level in B ;
- Computing all superorders of a given index to the order O ;
- Computing the left/right orders of a lattice;
- Computing fundamental domains of Eichler orders in indefinite quaternion algebras, as well as paths of closed geodesics.

Furthermore, there is a focus on computing with optimal embeddings. An embedding (Qemb) of the quadratic order of discriminant D (\mathcal{O}_D) into the quaternion order O is just a ring homomorphism $\phi : \mathcal{O}_D \rightarrow O$. It is optimal if it does not extend to an embedding of a larger order. Choosing an optimal embedding amounts to picking the element

$$\phi \left(\frac{p_D + \sqrt{D}}{2} \right),$$

i.e. an element $x \in O$ for which $x^2 - p_D x + \frac{p_D - D}{4} = 0$, where $p_D \in \{0, 1\}$ is the parity of D . Most of the time, we store optimal embeddings via this element x .

Two optimal embeddings are declared to be equivalent if they are related by conjugation by an element of norm 1 in O . Two optimal embeddings are said to have the same orientation if they are locally equivalent everywhere. If O is Eichler and B is indefinite, there are finitely many orientations, and the set of equivalence classes of optimal embeddings of the same orientation can be identified with the narrow class group $\text{Cl}^+(D)$ after choosing a basepoint.

A non-rational element $x \in O$ with separable minimal polynomial (guaranteed if B has ramification) will correspond to a unique optimal embedding of a quadratic order, called the associated embedding. Sometimes we allow passing of such an x .

In general, we will use the variable \mathbf{Q} to denote a quaternion algebra, \mathbf{ord} to denote a quaternion order, and \mathbf{order} to denote an initialized quaternion order.

6.1 Basic operations on elements in quaternion algebras

The “usual” operations, e.g. multiplication, conjugation, reduced norm, etc.

Name:	GEN qa_conj
Input:	GEN x
Input format:	Qelt x
Output format:	Qelt
Description:	Returns the conjugate of x. Note that a QA is not inputted.

Name:	GEN qa_conj_tc
Input:	GEN x
Input format:	Qelt x
Output format:	Qelt
Description:	Checks that x is a Qelt, and returns qa_conj(x).

Name:	GEN qa_conjby
Input:	GEN Q, GEN x, GEN y
Input format:	QA Q, Qelts x, y with y invertible
Output format:	Qelt
Description:	Returns xyx^{-1} .

Name:	GEN qa_conjby_tc
Input:	GEN Q, GEN x, GEN y
Input format:	QA Q, Qelts x, y with y invertible
Output format:	Qelt
Description:	Checks the types of Q, x, y and returns qa_conjby(Q, x, y).

Name:	GEN qa_inv
Input:	GEN Q, GEN x
Input format:	QA Q, invertible Qelt x
Output format:	Qelt
Description:	Returns the inverse of x.

Name:	GEN qa_inv_tc
Input:	GEN Q, GEN x
Input format:	QA Q, invertible Qelt x
Output format:	Qelt
Description:	Checks the types of Q, x and returns qa_inv(Q, x).

Name:	GEN qa_m2rembed
Input:	GEN Q, GEN x
Input format:	IQA Q, Qelt x
Output format:	2x2 t_MAT of t_QUADs
Description:	Returns the image of x under the standard embedding of Q into $M_2(\mathbb{R})$ (assumes that $a > 0$).

Name:	GEN qa_m2rembed_tc
Input:	GEN Q, GEN x
Input format:	IQA Q, Qelt x
Output format:	2x2 t_MAT of t_QUADs
Description:	Checks the types of Q, x and returns qa_m2rembed(Q, x).

Name:	GEN qa_minpoly
Input:	GEN Q, GEN x
Input format:	QA Q, Qelt x
Output format:	Vector
Description:	Returns the minimal polynomial of x. The format is 1, b, c for $x^2 + bx + c$, and [1, b] for $x + b$.

Name:	GEN qa_minpoly_tc
Input:	GEN Q, GEN x
Input format:	QA Q, Qelt x
Output format:	Vector
Description:	Checks the types of Q, x, and returns qa_minpoly(Q, x).

Name:	GEN qa_mul
Input:	GEN Q, GEN x, GEN y
Input format:	QA Q, Qelts x, y
Output format:	Qelt
Description:	Returns xy.

Name:	GEN qa_mul_tc
Input:	GEN Q, GEN x, GEN y
Input format:	QA Q, Qelts x, y
Output format:	Qelt
Description:	Checks the types of Q, x, y, and returns qa_mul(Q, x, y).

Name:	GEN qa_mulsets
Input:	GEN Q, GEN S1, GEN S2
Input format:	QA Q, vectors of Qelts S1, S2
Output format:	Vector of Qelts
Description:	Returns the vector of elements of the form $s1 \cdot s2$ for s_i in S_i .

Name:	GEN <code>qa_mulvec</code>
Input:	GEN <code>Q</code> , GEN <code>L</code>
Input format:	<code>QA</code> <code>Q</code> , vector of <code>Qelts</code> <code>L</code>
Output format:	<code>Qelt</code>
Description:	Returns the product $L[1] \cdot L[2] \cdots L[n]$.

Name:	GEN <code>qa_mulvec_tc</code>
Input:	GEN <code>Q</code> , GEN <code>L</code>
Input format:	<code>QA</code> <code>Q</code> , vector of <code>Qelts</code> <code>L</code>
Output format:	<code>Qelt</code>
Description:	Checks the types of <code>Q</code> , <code>L</code> and returns <code>qa_mulvec(Q, L)</code> .

Name:	GEN <code>qa_mulvecindices</code>
Input:	GEN <code>Q</code> , GEN <code>L</code> , GEN <code>indices</code>
Input format:	<code>QA</code> <code>Q</code> , vector of <code>Qelts</code> <code>L</code> , vecsmall <code>indices</code>
Output format:	<code>Qelt</code>
Description:	Returns the product $L[\text{indices}[1]] \cdot L[\text{indices}[2]] \cdots L[\text{indices}[n]]$.

Name:	GEN <code>qa_mulvecindices_tc</code>
Input:	GEN <code>Q</code> , GEN <code>L</code> , GEN <code>indices</code>
Input format:	<code>QA</code> <code>Q</code> , vector of <code>Qelts</code> <code>L</code> , vector/vecsmall <code>indices</code>
Output format:	<code>Qelt</code>
Description:	Checks the types of <code>Q</code> , <code>L</code> , sets <code>indices</code> to be a vecsmall if it is not, and returns <code>qa_mulvecindices(Q, L, indices)</code> .

Name:	GEN <code>qa_norm</code>
Input:	GEN <code>Q</code> , GEN <code>x</code>
Input format:	<code>QA</code> <code>Q</code> , <code>Qelt</code> <code>x</code>
Output format:	Rational
Description:	Returns the reduced norm of <code>x</code> .

Name:	GEN <code>qa_norm_tc</code>
Input:	GEN <code>Q</code> , GEN <code>x</code>
Input format:	<code>QA</code> <code>Q</code> , <code>Qelt</code> <code>x</code>
Output format:	Rational
Description:	Checks the types of <code>Q</code> , <code>x</code> , and returns <code>qa_norm(Q, x)</code> .

Name:	GEN <code>qa_pow</code>
Input:	GEN <code>Q</code> , GEN <code>x</code> , GEN <code>n</code>
Input format:	<code>QA</code> <code>Q</code> , <code>Qelt</code> <code>x</code> , integer <code>n</code>
Output format:	<code>Qelt</code>
Description:	Returns x^n .

Name:	GEN <code>qa_pow_tc</code>
Input:	GEN <code>Q</code> , GEN <code>x</code> , GEN <code>n</code>
Input format:	<code>QA</code> <code>Q</code> , <code>Qelt</code> <code>x</code> , integer <code>n</code>
Output format:	<code>Qelt</code>
Description:	Checks the types of <code>Q</code> , <code>x</code> , <code>n</code> , and returns <code>qa_pow(Q, x, n)</code> .

Name:	GEN <code>qa_roots</code>
Input:	GEN <code>Q</code> , GEN <code>x</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>Qelt</code> <code>x</code>
Output format:	Length 2 vector
Description:	Returns the roots of <code>x</code> under the standard embedding into $M_2(\mathbb{R})$, first root first.

Name:	GEN <code>qa_roots_tc</code>
Input:	GEN <code>Q</code> , GEN <code>x</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>Qelt</code> <code>x</code>
Output format:	Length 2 vector
Description:	Checks the types of <code>Q</code> , <code>x</code> and returns <code>qa_roots(Q, x, prec)</code> .

Name:	GEN <code>qa_square</code>
Input:	GEN <code>Q</code> , GEN <code>x</code>
Input format:	<code>QA</code> <code>Q</code> , <code>Qelt</code> <code>x</code>
Output format:	<code>Qelt</code>
Description:	Returns x^2 .

Name:	GEN <code>qa_square_tc</code>
Input:	GEN <code>Q</code> , GEN <code>x</code>
Input format:	<code>QA</code> <code>Q</code> , <code>Qelt</code> <code>x</code>
Output format:	<code>Qelt</code>
Description:	Checks the types of <code>Q</code> , <code>x</code> , and returns <code>qa_square(Q, x)</code> .

Name:	GEN <code>qa_trace</code>
Input:	GEN <code>x</code>
Input format:	<code>Qelt</code> <code>x</code>
Output format:	<code>Qelt</code>
Description:	Returns the reduced trace of <code>x</code> . Note that a <code>QA</code> is not inputted.

Name:	GEN <code>qa_trace_tc</code>
Input:	GEN <code>x</code>
Input format:	<code>Qelt</code> <code>x</code>
Output format:	<code>Qelt</code>
Description:	Checks that <code>x</code> is a <code>qelt</code> , and returns <code>qa_trace(x)</code> .

6.2 Basic operations on orders and lattices in quaternion algebras

Name:	int qa_isinorder
Input:	GEN Q, GEN ordinv, GEN x
Input format:	QA Q, inverse of a QO ordinv, Qelt x
Output format:	0 or 1
Description:	Checks if x is in the order specified by the inverse of ordinv, and returns 1 if so.

Name:	int qa_isinorder_tc
Input:	GEN Q, GEN ord, GEN x
Input format:	QA Q, QO ord, Qelt x
Output format:	0 or 1
Description:	Checks the inputs, and returns qa_isinorder(Q, ord ⁻¹ , x).

Name:	int qa_isorder
Input:	GEN Q, GEN ord, GEN ordinv
Input format:	QA Q, QO ord, ordinv the inverse of ord
Output format:	0 or 1
Description:	Checks if ord is an order, and returns 1 if so.

Name:	int qa_isorder_tc
Input:	GEN Q, GEN ord
Input format:	QA Q, QO ord
Output format:	0 or 1
Description:	Checks the inputs, and returns qa_isorder(Q, ord, ord ⁻¹).

Name:	GEN qa_leftorder
Input:	GEN Q, GEN L, GEN Linv
Input format:	QA Q, QL L, Linv=L ⁻¹
Output format:	QO
Description:	Returns the left order associated to L, i.e. the set of $x \in Q$ such that $xL \subseteq L$.

Name:	GEN qa_leftorder_tc
Input:	GEN Q, GEN L
Input format:	QA Q, QL L
Output format:	QO
Description:	Checks the inputs, and returns qa_leftorder(Q, L, L ⁻¹).

Name:	GEN qa_rightorder
Input:	GEN Q, GEN L, GEN Linv
Input format:	QA Q, QL L, Linv=L ⁻¹
Output format:	QO
Description:	Returns the right order associated to L, i.e. the set of $x \in Q$ such that $Lx \subseteq L$.

Name:	GEN qa_rightorder_tc
Input:	GEN Q, GEN L
Input format:	QA Q, QL L
Output format:	QO
Description:	Checks the inputs, and returns qa_rightorder(Q, L, L ⁻¹).

Name:	GEN qa_ord_conj
Input:	GEN Q, GEN ord, GEN c
Input format:	QA Q, QO ord, invertible Qelt c
Output format:	QO
Description:	Returns the order $c \cdot \text{ord} \cdot c^{-1}$.

Name:	GEN qa_ord_conj_tc
Input:	GEN Q, GEN ord, GEN c
Input format:	QA Q, (i)QO ord, invertible Qelt c
Output format:	QO
Description:	Checks the inputs, and returns qa_ord_conj(Q, ord, c).

Name:	GEN qa_ord_disc
Input:	GEN Q, GEN ord
Input format:	QA Q, QO ord
Output format:	Integer
Description:	Returns the discriminant of ord.

Name:	GEN qa_ord_disc_tc
Input:	GEN Q, GEN ord
Input format:	QA Q, (i)QO ord
Output format:	Integer
Description:	Checks the inputs, and returns qa_ord_disc(Q, ord).

Name:	GEN qa_ord_normform
Input:	GEN Q, GEN ord
Input format:	QA Q, QO ord
Output format:	4x4 matrix
Description:	Returns the matrix M such that $v^T M v = \text{nrd}(x)$, where $x = v[1]\text{ord}[1] + \dots + v[4]\text{ord}[4]$ (ord[i] is the i th column).

Name:	GEN qa_ord_type
Input:	GEN Q, GEN ord, GEN level
Input format:	QA Q, QO ord of level level
Output format:	-1, 0, 1
Description:	Returns the type of the order, which is 0 if maximal, 1 if Eichler but non-maximal, and -1 if non-Eichler.

Name:	GEN qa_superorders
Input:	GEN Q, GEN ord, GEN n
Input format:	QA Q, QO ord, integer n
Output format:	Vector of QOs
Description:	Returns all quaternion orders \mathcal{O} containing ord such that the quotient has size n.

Name:	GEN qa_superorders_prime
Input:	GEN Q, GEN ord, GEN ordinv, GEN n
Input format:	QA Q, QO ord, ordinv=ord ⁻¹ , prime number n
Output format:	Vector of QOs
Description:	Returns all quaternion orders \mathcal{O} containing ord such that the quotient has size n.

Name:	GEN qa_superorders_tc
Input:	GEN Q, GEN ord, GEN n
Input format:	QA Q, (i)QO ord, integer n
Output format:	Vector of QOs
Description:	Checks the inputs and returns qa_superorders(Q, ord, n).

6.3 Initialization methods

Name:	GEN qa_eichlerorder
Input:	GEN Q, GEN l, GEN maxord
Input format:	QA Q, positive integer l, QO maxord
Output format:	iEQO
Description:	Returns an initialized Eichler order of level l inside maxord.

Name:	GEN qa_eichlerorder_tc
Input:	GEN Q, GEN l, GEN maxord
Input format:	QA Q, positive integer l, (i)QO maxord or maxord=0
Output format:	iEQO
Description:	Checks the inputs, takes maxord to be a maximal order of Q if passed as 0, and returns qa_eichlerorder(Q, l, maxord).

Name:	GEN qa_maximalorder
Input:	GEN Q, GEN baseord
Input format:	QA Q, QO baseord
Output format:	iQO
Description:	Returns a maximal order containing baseord.

Name:	GEN qa_maximalorder_tc
Input:	GEN Q, GEN baseord
Input format:	QA Q, (i)QO baseord or baseord=0
Output format:	iQO
Description:	Checks the inputs, sets baseord if passed as 0, and returns qa_maximalorder(Q, baseord).

Name:	GEN qa_ord_init
Input:	GEN Q, GEN ord
Input format:	QA Q, QO ord
Output format:	iQO
Description:	Returns the initialized order corresponding to ord.

Name:	GEN qa_ord_init_tc
Input:	GEN Q, GEN ord
Input format:	QA Q, QO ord
Output format:	iQO
Description:	Checks the inputs, and returns qa_ord_init(Q, ord).

Name:	GEN qa_ord_init_trace0basis
Input:	GEN Q, GEN ord, GEN maxds
Input format:	QA Q, QO ord, length 3 vector maxds
Output format:	Length 3 vector
Description:	If the maximal denominator of the i^{th} coefficient of an element of ord is maxds[i], this returns a basis for the trace 0 elements of ord.

Name:	GEN qa_init_ab
Input:	GEN a, GEN b
Input format:	Non-zero integers a, b
Output format:	QA
Description:	Returns the quaternion algebra $\left(\frac{a,b}{\mathbb{Q}}\right)$.

Name:	GEN <code>qa_init_ab_tc</code>
Input:	GEN <code>a</code> , GEN <code>b</code>
Input format:	Non-zero integers <code>a</code> , <code>b</code>
Output format:	QA
Description:	Checks <code>a</code> , <code>b</code> and returns <code>qa_init_ab(a, b)</code> .

Name:	GEN <code>qa_init_primes</code>
Input:	GEN <code>pset</code> , int <code>type</code>
Input format:	Sorted vector of primes <code>pset</code> of even length, <code>type</code> =-1, 1
Output format:	QA
Description:	Returns the quaternion algebra ramified at primes of <code>pset</code> . <code>type</code> must be passed as 1 if this is indefinite, and as -1 if definite (with the infinite prime being included in <code>pset</code>).

Name:	GEN <code>qa_init_primes_tc</code>
Input:	GEN <code>pset</code>
Input format:	Vector of primes <code>pset</code>
Output format:	QA
Description:	Sorts the vector <code>pset</code> , adds ∞ if of odd length and not present, determines the type (definite/indefinite), and returns <code>qa_init_primes(pset, type)</code> .

Name:	GEN <code>qa_init_m2z</code>
Input:	
Input format:	
Output format:	QA
Description:	Returns the quaternion algebra over \mathbb{Q} ramified nowhere.

Name:	GEN <code>qa_init_2primes</code>
Input:	GEN <code>p</code> , GEN <code>q</code>
Input format:	Distinct primes <code>p</code> , <code>q</code>
Output format:	[QA, iQO]
Description:	Returns the quaternion algebra ramified at <code>p</code> , <code>q</code> and a maximal order.

Name:	GEN <code>qa_init_2primes_tc</code>
Input:	GEN <code>p</code> , GEN <code>q</code>
Input format:	Distinct primes <code>p</code> , <code>q</code>
Output format:	[QA, iQO]
Description:	Checks that <code>p</code> , <code>q</code> are distinct primes, and returns <code>qa_init_2primes(p, q)</code> .

Name:	GEN qa_ram_fromab
Input:	GEN a, GEN b
Input format:	Integers a, b
Output format:	Vector
Description:	Returns the sorted set of primes ramifying in the quaternion algebra (a, b/Q).

Name:	GEN qa_ram_fromab_tc
Input:	GEN a, GEN b
Input format:	Integers a, b
Output format:	Vector
Description:	Checks that a, b are integers, and returns qa_ram_fromab(a, b).

6.4 Conjugation of elements in a given order

Name:	GEN qa_conjbasis
Input:	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2, int orient
Input format:	QA Q, QO ord, ordinv=ord ⁻¹ , non-rational conjugate Qelts e1, e2, orient=0, 1
Output format:	0 or [v1, v2]
Description:	Returns a (length 2) basis for the set of $x \in Q$ for which $x \cdot e1 = e2 \cdot x$. If e1, e2 are rational or not conjugate, returns 0. If orient=1, orients the output so that $v2 \overline{v1} = A + B e2$ with $B > 0$.

Name:	GEN qa_conjbasis_tc
Input:	GEN Q, GEN ord, GEN e1, GEN e2, int orient
Input format:	QA Q, (i)QO ord, non-rational conjugate Qelts e1, e2, orient=0, 1
Output format:	0 or [v1, v2]
Description:	Checks the inputs, and returns qa_conjbasis(Q, ord, ord ⁻¹ , e1, e2, orient).

Name:	GEN qa_conjbasis_orient
Input:	GEN Q, GEN ord, GEN v1, GEN v2, GEN e2
Input format:	QA Q, QO ord, Qelts v1, v2, e2
Output format:	[v1', v2']
Description:	Orients the output of qa_conjbasis so that $v2' \overline{v1'} = A + B e2$ with $B > 0$.

Name:	GEN qa_conjqf
Input:	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2
Input format:	QA Q, QO ord, ordinv=ord ⁻¹ , non-rational conjugate Qelts e1, e2
Output format:	0 or [q, v1, v2]
Description:	Computes the BQF associated to e1, e2, where [v1, v2] is the output from conjbasis with orient=1, and q is the BQF coming from nrd(X.v1+Y.v2).

Name:	GEN qa_conjqf_tc
Input:	GEN Q, GEN ord, GEN e1, GEN e2
Input format:	QA Q, (i)QO ord, non-rational conjugate Qelts e1, e2
Output format:	0 or [q, v1, v2]
Description:	Checks the inputs, and returns qa_conjqf(Q, ord, ord ⁻¹ , e1, e2).

Name:	GEN qa_conjnorm
Input:	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2, GEN n, int retconelt, long prec
Input format:	QA Q, QO ord, ordinv=ord ⁻¹ , non-rational conjugate Qelts e1, e2, integer n, retconelt=0, 1
Output format:	0, 1 or Qelt
Description:	Checks if there is an invertible element $x \in \text{ord}$ with $\text{nrd}(x)=n$ and $x \cdot e1 \cdot x^{-1} = e2$, and returns the determination. If retconelt=1, returns the element.

Name:	GEN qa_conjnorm_tc
Input:	GEN Q, GEN ord, GEN e1, GEN e2, GEN n, int retconelt, long prec
Input format:	QA Q, (i)QO ord, non-rational conjugate Qelts e1, e2, integer n, retconelt=0, 1
Output format:	0, 1 or qelt
Description:	Checks the inputs, and returns qa_conjnorm(Q, ord, ord\text{tinv} , e1, e2, n, retconelt, prec).

Name:	GEN qa_simulconj
Input:	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2, GEN f1, GEN f2, long prec
Input format:	QA Q, QO ord, ordinv=ord ⁻¹ , simultaneously conjugate pairs of Qelts (e1, e2) and (f1, f2)
Output format:	0 or Qelt
Description:	If the pairs (e1, e2) and (f1, f2) are simultaneously conjugate with e1, e2, e1e2 all being non-rational (equivalent to the minimal polynomials of e1, e2, e1e2 and f1, f2, f1f2 being equal), then the conjugation space is 1-dimensional. This method returns a generator for this space intersected with ord, and 0 if they are not simultaneously conjugate.

Name:	GEN <code>qa_simulconj_tc</code>
Input:	GEN <code>Q</code> , GEN <code>ord</code> , GEN <code>e1</code> , GEN <code>e2</code> , GEN <code>f1</code> , GEN <code>f2</code> , long <code>prec</code>
Input format:	<code>QA</code> <code>Q</code> , (<code>i</code>) <code>QO</code> <code>ord</code> , simultaneously conjugate pairs of Qelts (<code>e1</code> , <code>e2</code>) and (<code>f1</code> , <code>f2</code>)
Output format:	0 or Qelt
Description:	Checks the inputs, and returns <code>qa_simulconj(Q, ord, ord⁻¹, e1, e2, f1, f2, prec)</code> .

6.5 Embedding quadratic orders into Eichler orders

Name:	GEN <code>qa_associatedemb</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>emb</code> , GEN <code>D</code>
Input format:	<code>QA</code> <code>Q</code> , <code>iQO</code> <code>order</code> , <code>Qelt</code> <code>emb</code> , integer <code>D</code>
Output format:	[<code>emb'</code> , <code>D'</code>]
Description:	Computes the unique optimal embedding associated to <code>order</code> and <code>emb</code> (i.e. is an optimal embedding into <code>order</code> and agrees with <code>emb</code> where both defined). <code>emb</code> is assumed to be the image of $(A + \sqrt{D})/2$, where <code>D</code> may be passed in as 0. The output is the pair consisting of the associated embedding and its discriminant.

Name:	GEN <code>qa_associatedemb_tc</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>emb</code> , GEN <code>D</code>
Input format:	<code>QA</code> <code>Q</code> , <code>iQO</code> <code>order</code> , <code>Qelt</code> <code>emb</code> , integer <code>D</code>
Output format:	[<code>emb'</code> , <code>D'</code>]
Description:	Checks the inputs, and returns <code>qa_associatedemb(Q, order, emb, D)</code> .

Name:	GEN <code>qa_embed</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>D</code> , GEN <code>nembeds</code> , GEN <code>rpell</code> , long <code>prec</code>
Input format:	<code>QA</code> <code>Q</code> , <code>iQO</code> <code>order</code> , discriminant <code>D</code> , positive integer <code>nembeds</code> , <code>rpell</code> =0 or the output of <code>pell(D)</code>
Output format:	Vector
Description:	Finds and returns <code>nembeds</code> non-equivalent optimal embeddings of the order of discriminant <code>D</code> into <code>order</code> . Will return the images of $(p_D + \sqrt{D})/2$ if <code>rpell</code> =0. Otherwise, <code>rpell</code> =[<code>T</code> , <code>U</code>], and will return the images of $(T + U\sqrt{D})/2$, which has norm 1 if <code>rpell</code> = <code>pell(D)</code> . This method does not check that it is possible to find <code>nembeds</code> non-equivalent embeddings, so if you cannot, it will never end (and eventually the memory will run out).

Name:	GEN qa_embed_tc
Input:	GEN Q, GEN order, GEN D, GEN nembeds, int rpell, long prec
Input format:	QA Q, iQO order, discriminant D, integer nembeds, rpell=0, 1
Output format:	Vector
Description:	Checks the inputs, and returns qa_embed(Q, ...). If nembeds=0, this computes the total number of non-equivalent optimal embeddings and feeds this into qa_embed. If rpell=1, this returns the images of the fundamental unit, and otherwise it returns the images of $(p_D + \sqrt{D})/2$.

Name:	int qa_embed_isnewoptimal
Input:	GEN Q, GEN ord, GEN ordinv, GEN D, GEN Dmod2, GEN dfacs, GEN emb, GEN gcdf1g1h1, GEN embs, long pos, long prec
Input format:	QA Q, QO ord, ordinv=ord ⁻¹ , discriminant D, Dmod2=D modulo 2, dfacs the vector of primes dividing D/D^{fund} , Qemb emb, gcdf1g1h1=gcd(f1, g1, h1) (see source code for qa_embed), embs a list of optimal Qembs of discriminant D of length pos-1
Output format:	0 or 1
Description:	Returns 1 if the embedding emb is optimal and not conjugate to one of the embeddings in embs, and 0 otherwise. Supporting method to qa_embed.

Name:	GEN qa_embeddablediscs
Input:	GEN Q, GEN order, GEN d1, GEN d2, int fund, GEN cop
Input format:	IQA Q, iEQO order, integers D1, D2, fund=0, 1, integer cop
Output format:	Vector
Description:	Returns the vector of discriminants D with $d1 \leq D \leq d2$ for which there exists optimal embeddings of D into order. If fund=1, only returns fundamental discriminants. If cop \neq 0, only returns discriminants coprime to cop.

Name:	GEN qa_embeddablediscs_tc
Input:	GEN Q, GEN order, GEN d1, GEN d2, int fund, GEN cop
Input format:	IQA Q, iEQO order, integers D1, D2, fund=0, 1, integer cop
Output format:	Vector
Description:	Checks the inputs, and returns qa_embeddablediscs(Q, order, d1, d2, fund, cop).

Name:	GEN qa_numemb
Input:	GEN Q, GEN order, GEN D, GEN narclno
Input format:	IQA Q, iEQO order, discriminant D, positive integer narclno
Output format:	[m, n, v1, v2, v3]
Description:	Returns data associated to the number of optimal embeddings of D into order. m is the total number of optimal embeddings, n is the number of a fixed orientation (i.e. $h^+(D)$), v1=[x] with x being the number of orientations at ∞ , v2=[x1, ..., xr] with Q[1]=[p1, ..., pr] and there are xi orientations (local embeddings) at the prime pi ramifying in Q, and v3=[y1, ..., ys] where the s distinct primes q1, q2, ..., qs divide the level of order and yi is the number of orientations at the prime qi. If you just want to check for non-zeros, pass in narclno=1; the corresponding m, n values will be incorrect, but will be non-zero if and only if an optimal embedding exists.

Name:	GEN qa_numemb_tc
Input:	GEN Q, GEN order, GEN D, GEN narclno, long prec
Input format:	IQA Q, iEQO order, discriminant D, nonnegative integer narclno
Output format:	[m, n, v1, v2, v3]
Description:	Checks the inputs, presets narclno if passed as 0, and returns qa_numemb(Q, order, D, narclno).

Name:	GEN qa_ordiffer
Input:	GEN Q, GEN order, GEN e1, GEN e2, GEN D
Input format:	IQA Q, iEQO order, Qembs e1, e2, discriminant D
Output format:	Vector
Description:	Returns the vector of primes for which the optimal embeddings e1, e2 of discriminant D differ in orientation at.

Name:	GEN qa_ordiffer_tc
Input:	GEN Q, GEN order, GEN e1, GEN e2, GEN D
Input format:	IQA Q, iEQO order, Qembs e1, e2, discriminant D
Output format:	Vector
Description:	Checks the inputs and returns qa_ordiffer(Q, order, e1, e2, D).

Name:	GEN qa_orinfinite
Input:	GEN Q, GEN emb, GEN D, long prec
Input format:	IQA Q, Qemb [emb], discriminant D
Output format:	-1, 1
Description:	Returns the orientation of emb at ∞ .

Name:	GEN <code>qa_orinfinite_tc</code>
Input:	GEN <code>Q</code> , GEN <code>emb</code> , GEN <code>D</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>Qemb</code> [<code>emb</code>], discriminant <code>D</code>
Output format:	-1, 1
Description:	Checks the inputs and returns <code>qa_orinfinite(Q, emb, D, prec)</code> .

Name:	GEN <code>qa_sortedembed</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>D</code> , GEN <code>rpell</code> , GEN <code>ncgp</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>IEQO</code> <code>order</code> , discriminant <code>D</code> , <code>rpell=0</code> or <code>pell(D)</code> , <code>ncgp=bqf_ncgp_lexic(D, prec)</code>
Output format:	0 or matrix
Description:	Computes all optimal embeddings of <code>D</code> into <code>order</code> , and returns the sorted output. The output is <code>N x 2</code> matrix, with the entries in the second column being $h^+(D)$ optimal embeddings, sorted according to the order of the forms in <code>ncgp</code> . The first column entries denote the sets of primes for which the orientations of embeddings in that row differ to the embeddings of the first row. The ordering of embeddings also respects the action of Atkin-Lehner elements (except for the case that primes dividing the level of <code>order</code> also divide <code>D</code>).

Name:	GEN <code>qa_sortedembed_tc</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>D</code> , int <code>rpell</code> , GEN <code>ncgp</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>IEQO</code> <code>order</code> , discriminant <code>D</code> , <code>rpell=0</code> , 1, <code>ncgp=0</code> or <code>bqf_ncgp_lexic(D, prec)</code>
Output format:	0 or matrix
Description:	Checks the inputs and returns <code>qa_sortedembed(Q, order, ...)</code> . If <code>rpell=1</code> and/or <code>ncgp=0</code> , this method will preset them.

Name:	int <code>qa_embedor_compare</code>
Input:	void <code>*data</code> , GEN <code>pair1</code> , GEN <code>pair2</code>
Input format:	<code>pair1</code> , <code>pair2</code> length 2 vectors
Output format:	0, 1
Description:	This compares two pairs [<code>e1</code> , <code>or1</code>], [<code>e2</code> , <code>or2</code>]: first by longest vector <code>ori</code> , then lexicographically by <code>ori</code> .

6.6 Elements of norm n in an Eichler order

Let n be coprime to the level of the Eichler order O , and write

$$\Theta(n) := O_{N=1} \backslash O_{N=n} = \bigcup_{i=1}^M O_{N=1} \pi_i,$$

the set of element of reduced norm n modulo left multiplication by the set of elements of reduced norm 1. These methods compute $\Theta(n)$, as well as the action of the Hecke operator T_n on optimal embeddings.

Name:	GEN <code>qa_orbitrepsprime</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>p</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>ieQO</code> <code>order</code> , prime <code>p</code>
Output format:	Vector
Description:	Returns representatives for $\Theta(p)$ when p is coprime to the level of <code>order</code> .

Name:	int <code>qa_orbitrepsprime_checksol</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>T</code> , GEN <code>W</code> , GEN <code>*reps</code> , long <code>*place</code> , GEN <code>sol</code>
Input format:	
Output format:	
Description:	Helper method to <code>qa_orbitrepsprime</code> that updates the vector of solutions and returns 1 if we are done.

Name:	GEN <code>qa_orbitrepsprimepower</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>p</code> , GEN <code>e</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>ieQO</code> <code>order</code> , prime <code>p</code> , nonnegative integer <code>e</code>
Output format:	Vector
Description:	Returns representatives for $\Theta(p^e)$ when p is coprime to the level of <code>order</code> .

Name:	GEN <code>qa_orbitreps</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>n</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>ieQO</code> <code>order</code> , positive integer <code>n</code>
Output format:	Vector
Description:	Returns representatives for $\Theta(n)$ when n is coprime to the level of <code>order</code> .

Name:	GEN <code>qa_orbitreps_tc</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>n</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>ieQO</code> <code>order</code> , positive integer <code>n</code>
Output format:	Vector
Description:	Checks the inputs, and returns representatives for $\Theta(n)$ when n is coprime to the level of <code>order</code> .

Name:	GEN <code>qa_orbitrepsrange</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>n</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>ieQO</code> <code>order</code> , positive integer <code>n</code>
Output format:	Vector
Description:	Returns representatives for $\Theta(i)$ for all $i \leq n$ coprime to the level of <code>order</code> .

Name:	GEN qa_orbitrepsrange_tc
Input:	GEN Q, GEN order, GEN n, long prec
Input format:	IQA Q, iEQO order, positive integer n
Output format:	Vector
Description:	Checks the inputs, and returns representatives for $\Theta(i)$ for all $i \leq n$ coprime to the level of order.

Name:	GEN qa_hecke
Input:	GEN Q, GEN order, GEN n, GEN emb, GEN D, long prec
Input format:	IQA Q, iEQO order, positive integer n, Qemb emb of positive discriminant D
Output format:	Vector of [m, emb']
Description:	Returns $T_n[\text{emb}]$, where n must be coprime to the level of the order. Each return element corresponds to an optimal embedding emb' of multiplicity m .

Name:	GEN qa_hecke_tc
Input:	GEN Q, GEN order, GEN n, GEN emb, long prec
Input format:	IQA Q, iEQO order, positive integer n, Qemb emb of positive discriminant
Output format:	Vector of [m, emb']
Description:	Checks the inputs, and returns qa_hecke(Q, order, n, emb, D, prec).

6.7 Fundamental domain methods

An algorithm to compute the fundamental domain of a Fuchsian group is described in a paper of Voight ([Voi09]). While we generally follow this process for the geometric part of it, we replace the enumeration of elements by adapting the probabilistic enumeration of Page in [Pag15]. As before, we work with the unit disc model for hyperbolic space, and we map from the upper half plane to it via the point p (which is sent to the origin). Anytime a method calls for `*data`, this must point to an indefinite quaternion algebra \mathbb{Q} . See the source code for descriptions of the format of normalized boundaries/fundamental domains.

Name:	GEN qa_fundamentaldomain
Input:	GEN Q, GEN order, GEN p, int dispprogress, GEN ANRdata, GEN tol, long prec
Input format:	IQA Q, iEQO order, upper half plane point p, dispprogress=0, 1, ANRdata=0 or a length 5 vector
Output format:	Fundamental domain
Description:	Returns the fundamental domain associated to order. If ANRdata is non-zero, it corresponds to the constants [A, N, R, 1+nu, epsilon] as in [Pag15]. Any non-zero values will be used as the constants in the enumeration, with the zero values still being automatically set. If dispprogress=1, we print the progress of the method to the screen during the computation.

Name:	GEN qa_fundamentaldomain_tc
Input:	GEN Q, GEN order, GEN p, int dispprogress, GEN ANRdata, long prec
Input format:	IQA Q, iEQO order, upper half plane point p, dispprogress=0, 1, ANRdata=0 or a length 5 vector
Output format:	Fundamental domain
Description:	Checks the input, sets $p=1/2$ if passed as 0, and returns qa_fundamentaldomain(Q, order, ...).

Name:	GEN qa_invradqf
Input:	GEN Q, GEN order, GEN mats, GEN z, long prec
Input format:	IQA Q, iQO order, transition data mats, unit disc point z
Output format:	4x4 Matrix
Description:	Returns the quadratic form invrad as a matrix (if a basis of order is v_1, v_2, v_3, v_4 , then this is $\text{invrad}(e_1 \cdot v_1 + \dots + e_4 \cdot v_4)$, with variables e_1, e_2, e_3, e_4). Invrad is defined on page 477 of JV09) as the sum of the reciprocal of the Euclidean radius of the isometric circle plus the reduced norm, and is a positive definite form.

Name:	GEN qa_isometriccircle
Input:	GEN Q, GEN x, GEN mats, GEN tol, long prec
Input format:	IQA Q, Qelt x of norm 1, transition data mats
Output format:	[x, mat, circ]
Description:	Finds the isometric circle circ of x with respect to mats. The image of x in PSU(1, 1) is mat.

Name:	GEN qa_isometriccircle_tc
Input:	GEN Q, GEN x, GEN p, long prec
Input format:	IQA Q, Qelt x of norm 1, upper half plane point p
Output format:	[x, mat, circ]
Description:	Checks the inputs and returns qa_isometriccircle(Q, x, ...).

Name:	GEN qa_fdarea
Input:	GEN Q, GEN order, long prec
Input format:	IQA Q, iEQO order
Output format:	Real
Description:	Returns the hyperbolic area of the fundamental domain associated to order.

Name:	GEN qa_fdarea_tc
Input:	GEN Q, GEN order, long prec
Input format:	IQA Q, iEQO order
Output format:	Real
Description:	Checks the inputs and returns qa_fdarea(Q, order, prec).

Name:	GEN qa_fdm2rembed
Input:	GEN *data, GEN x, long prec
Input format:	data points to a QA Q, Qelt x of norm 1
Output format:	
Description:	qa_m2rembed, but in the format required for the geometry package.

Name:	GEN qa_fdinv
Input:	GEN *data, GEN x
Input format:	data points to a QA Q, invertible Qelt x
Output format:	
Description:	qa_inv, but in the format required for the geometry package.

Name:	GEN qa_fdmul
Input:	GEN *data, GEN x, GEN y
Input format:	data points to a QA Q, Qelts x, y
Output format:	
Description:	qa_mul, but in the format required for the geometry package.

Name:	int qa_istriv
Input:	GEN *data, GEN x
Input format:	data points to a QA Q, Qelt x
Output format:	
Description:	Returns 1 if x is equal to ± 1 in Q.

Name:	GEN qa_normalizedbasis
Input:	GEN Q, GEN G, GEN mats, GEN U, GEN tol, long prec
Input format:	IQA Q, vector of Qelts G of norm 1, transition data mats, normalized boundary U or 0
Output format:	Normalized boundary
Description:	Returns the normalized basis associated to the union of U and G.

Name:	GEN qa_normalizedbasis_tc
Input:	GEN Q, GEN G, GEN p, long prec
Input format:	IQA Q, vector of Qelts G of norm 1, upper half plane point or normalized boundary p
Output format:	Normalized boundary
Description:	If p is a point, returns the normalized basis associated to U with respect to p. If p is a normalized boundary, returns the normalized basis associated to G union p.

Name:	GEN qa_normalizedboundary
Input:	GEN Q, GEN G, GEN mats, GEN tol, long prec
Input format:	IQA Q, vector of Qelts G of norm 1, transition data mats
Output format:	Normalized boundary
Description:	Returns the normalized boundary associated G.

Name:	GEN qa_normalizedboundary_tc
Input:	GEN Q, GEN G, GEN p, long prec
Input format:	IQA Q, vector of Qelts G of norm 1, upper half plane point p
Output format:	Normalized boundary
Description:	Checks the inputs and returns the normalized boundary of G with respect to p.

Name:	void qa_printisometriccircles
Input:	GEN Q, GEN L, char *filename, GEN mats, GEN tol, long prec
Input format:	IQA Q, vector of Qelts L of norm 1, string filename, transition data mats
Output format:	
Description:	Computes the isometric circles of L, and prints the circles to "fdoms/filename.dat".

Name:	void qa_printisometriccircles_tc
Input:	GEN Q, GEN L, GEN p, char *filename, int view, long prec
Input format:	IQA Q, vector of Qelts L of norm 1, upper half plane point p, string filename, view=0, 1
Output format:	
Description:	Computes the isometric circles of L with respect to p, and prints the circles to "fdoms/filename.dat". If view=1, runs the code to display the circles (on Windows subsystem for Linux only).

Name:	GEN qa_reduceelt
Input:	GEN Q, GEN G, GEN x, GEN z, GEN p, GEN tol, long prec
Input format:	IQA Q, vector of Qelts G of norm 1, Qelt x of norm 1, unit disc point z, upper half plane point p
Output format:	[gammabar, delta, decomp]
Description:	Reduces the element x with respect to G and z. In otherwords, $d(\text{gammabar} \cdot z, 0) \leq d(g \cdot \text{gammabar} \cdot z, 0)$ for all g in G , where $\text{gammabar} = \text{delta} \cdot x$ and decomp is the vecsmall of indices of G used to produce delta.

Name:	GEN <code>qa_reduceelt_normbound</code>
Input:	GEN Q, GEN U, GEN x, GEN z, GEN <code>tol</code> , long <code>prec</code>
Input format:	<code>IQA</code> Q, <code>normalized boundary</code> U, <code>Qelt</code> x of norm 1, unit disc point z
Output format:	[<code>gammabar</code> , <code>delta</code> , <code>decomp</code>]
Description:	As <code>qa_reduceelt</code> , but we pass in the normalized boundary formed by G instead. This method is much faster than <code>qa_reduceelt</code> .

Name:	GEN <code>qa_reduceelt_tc</code>
Input:	GEN Q, GEN G, GEN x, GEN z, GEN p, long <code>prec</code>
Input format:	<code>IQA</code> Q, vector of <code>Qelts</code> G of norm 1 OR <code>normalized boundary</code> G, <code>Qelt</code> x of norm 1, unit disc point z, upper half plane point p
Output format:	[<code>gammabar</code> , <code>delta</code> , <code>decomp</code>]
Description:	Returns the reduction of x with respect to G and z, as in <code>qa_reduceelt(_normbound)</code> .

Name:	GEN <code>qa_rootgeodesic_fd</code>
Input:	GEN Q, GEN U, GEN g, GEN <code>tol</code> , long <code>prec</code>
Input format:	<code>IQA</code> Q, <code>normalized boundary</code> U, <code>Qelt</code> g of norm 1
Output format:	[<code>elts</code> , <code>arcs</code> , <code>sides hit</code> , <code>sides left</code>]
Description:	Computes the image of the root geodesic of x in U. The <code>elts</code> are the elements whose unit disc root geodesics correspond to the consecutive sides, <code>arcs</code> are the corresponding arcs, <code>sides hit</code> are the indices of the sides the geodesic hits, and <code>sides left</code> are the indices of the sides the geodesic leaves from.

Name:	GEN <code>qa_rootgeodesic_fd_tc</code>
Input:	GEN Q, GEN U, GEN g, long <code>prec</code>
Input format:	<code>IQA</code> Q, <code>normalized boundary</code> U, <code>Qelt</code> g of norm 1
Output format:	[<code>elts</code> , <code>arcs</code> , <code>sides hit</code> , <code>sides left</code>]
Description:	Checks the inputs and returns <code>qa_rootgeodesic_fd(Q, ...)</code> .

Name:	GEN <code>qa_smallnorm1elts_invrad</code>
Input:	GEN Q, GEN order, GEN C1, GEN C2, GEN invrad, long <code>prec</code>
Input format:	<code>IQA</code> Q, <code>iQO</code> order, reals C1, C2, 4x4 matrix invrad
Output format:	Vector of <code>Qelts</code>
Description:	Returns the norm 1 elements of order for which $C1 < \text{invrad}(x) \leq C2$.

Name:	GEN <code>qa_smallnorm1elts_tc</code>
Input:	GEN Q, GEN order, GEN p, GEN z, GEN C1, GEN C2, long <code>prec</code>
Input format:	<code>IQA</code> Q, <code>iQO</code> order, upper half plane point p, reals C1, C2
Output format:	Vector of <code>Qelts</code>
Description:	Returns the norm 1 elements of order for which $C1 < \text{invrad}(x) \leq C2$. If $p=0$, sets $p=1/2$, and if $C2=0$, then sets $(C1, C2)=(0, C1)$.

Name:	GEN <code>qa_topsu</code>
Input:	GEN <code>Q</code> , GEN <code>g</code> , GEN <code>p</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>Qelt</code> <code>g</code> of norm 1, upper half plane point <code>p</code>
Output format:	Matrix
Description:	Returns the image of <code>g</code> in $\text{PSU}(1,1)$.

Name:	GEN <code>qa_topsu_mat</code>
Input:	GEN <code>Q</code> , GEN <code>g</code> , GEN <code>mats</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>Qelt</code> <code>g</code> of norm 1, <code>transition data</code> <code>mats</code>
Output format:	Matrix
Description:	Returns the image of <code>g</code> in $\text{PSU}(1,1)$.

Name:	GEN <code>qa_topsu_tc</code>
Input:	GEN <code>Q</code> , GEN <code>g</code> , GEN <code>p</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>Qelt</code> <code>g</code> of norm 1, upper half plane point <code>p</code>
Output format:	Matrix
Description:	Checks the inputs and returns the image of <code>g</code> in $\text{PSU}(1,1)$.

6.8 Checking methods

Methods here are mostly used to check the inputs and avoid segmentation errors in GP (generally not useful in PARI).

Name:	void <code>qa_check</code>
Input:	GEN <code>Q</code>
Input format:	<code>QA</code> <code>Q</code>
Output format:	-
Description:	Raises an error if <code>Q</code> is not a vector of the correct length or the sub-vector of <code>[a, b, -ab]</code> is not a vector of length 3.

Name:	void <code>qa_indefcheck</code>
Input:	GEN <code>Q</code>
Input format:	<code>IQA</code> <code>Q</code>
Output format:	-
Description:	Raises an error if <code>Q</code> is not an indefinite quaternion algebra.

Name:	void <code>qa_eltcheck</code>
Input:	GEN <code>x</code>
Input format:	<code>Qelt</code> <code>x</code>
Output format:	-
Description:	Raises an error if <code>x</code> is not a length 4 vector.

Name:	GEN <code>qa_ordcheck</code>
Input:	GEN <code>ord</code>
Input format:	(i) <code>QO</code> <code>ord</code>
Output format:	<code>QO</code>
Description:	Checks that <code>ord</code> is a quaternion order (possibly initialized), and returns the (uninitialized) order (which is either <code>ord</code> or in <code>ord</code> , and NOT a copy of it).

Name:	void <code>qa_ordeichlercheck</code>
Input:	GEN <code>order</code>
Input format:	<code>iEQO</code> <code>order</code>
Output format:	-
Description:	Raises an error if <code>order</code> is not an initialized Eichler order.

Name:	void <code>QM_check</code>
Input:	GEN <code>M</code>
Input format:	matrix <code>M</code>
Output format:	-
Description:	Raises an error if <code>M</code> is not a rational matrix.

6.9 Property retrieval

Quaternion algebras/orders store a fair amount of precomputed information, and these methods retrieve this info. None of it is stack clean, since it is always a reference to the object in the input. It is equally fine to call `gel(Q, 1)`, etc., but this makes the code a bit more readable and resistant to a change of input format.

Name:	GEN <code>qa_getnf</code>
Input:	GEN <code>Q</code>
Input format:	<code>QA</code> <code>Q</code>
Output format:	Number field
Description:	From <code>Q</code> , retrieve the number field.

Name:	GEN <code>qa_getpram</code>
Input:	GEN <code>Q</code>
Input format:	<code>QA</code> <code>Q</code>
Output format:	Vector
Description:	From <code>Q</code> , retrieve the ramifying primes.

Name:	GEN <code>qa_getabvec</code>
Input:	GEN <code>Q</code>
Input format:	<code>QA</code> <code>Q</code>
Output format:	Vector
Description:	From <code>Q</code> , retrieve the vector <code>[a, b, -ab]</code> .

Name:	GEN qa_geta
Input:	GEN Q
Input format:	QA Q
Output format:	Integer
Description:	From Q, retrieve a.

Name:	GEN qa_getb
Input:	GEN Q
Input format:	QA Q
Output format:	Integer
Description:	From Q, retrieve b.

Name:	GEN qa_getmab
Input:	GEN Q
Input format:	QA Q
Output format:	Integer
Description:	From Q, retrieve $-ab$.

Name:	GEN qa_getpramprod
Input:	GEN Q
Input format:	QA Q
Output format:	Integer
Description:	From Q, retrieve the product of the ramifying primes.

Name:	GEN qa_getord
Input:	GEN order
Input format:	iQO order
Output format:	QO
Description:	From order, get ord.

Name:	GEN qa_getordtype
Input:	GEN order
Input format:	iQO order
Output format:	-1, 0, 1
Description:	From order, get the type (Eichler/maximal/other).

Name:	GEN qa_getordmaxd
Input:	GEN order
Input format:	iQO order
Output format:	Length 4 vector
Description:	From order get the maximal denominators of the coefficients.

Name:	GEN <code>qa_getordlevel</code>
Input:	GEN <code>order</code>
Input format:	<code>iQ0</code> <code>order</code>
Output format:	Integer
Description:	From <code>order</code> get the level.

Name:	GEN <code>qa_getordlevelpfac</code>
Input:	GEN <code>order</code>
Input format:	<code>iQ0</code> <code>order</code>
Output format:	<code>nx2</code> matrix
Description:	From <code>order</code> get the prime factorization of the level.

Name:	GEN <code>qa_getordinv</code>
Input:	GEN <code>order</code>
Input format:	<code>iQ0</code> <code>order</code>
Output format:	QO^{-1}
Description:	From <code>order</code> get the inverse of <code>ord</code> .

Name:	GEN <code>qa_getordtrace0basis</code>
Input:	GEN <code>order</code>
Input format:	<code>iQ0</code> <code>order</code>
Output format:	Length 3 vector
Description:	From <code>order</code> get the basis of trace 0 elements.

6.10 Supporting methods

Name:	int <code>cmp_data</code>
Input:	void <code>*data</code> , GEN <code>x</code> , GEN <code>y</code>
Input format:	<code>x</code> , <code>y</code> any GENs
Output format:	-1, 0, 1
Description:	Calls and returns <code>gcmp(x, y)</code> . Use this to have <code>gcmp</code> in <code>gen_sort</code> methods (like <code>gen_sort_uniq</code>).

Name:	GEN <code>module_intersect</code>
Input:	GEN <code>A</code> , GEN <code>B</code>
Input format:	QM <code>A</code> and <code>B</code>
Output format:	0 or matrix
Description:	Given \mathbb{Z} -modules spanned by the columns of <code>A</code> , <code>B</code> , this finds and returns their intersection (as a matrix with columns forming a \mathbb{Z} -basis, of 0 if trivial intersection).

Name:	GEN module_intersect_tc
Input:	GEN A, GEN B
Input format:	QM A and B
Output format:	0 or matrix
Description:	Checks that A, B are rational matrices, and returns <code>module_intersect(A, B)</code> .

Name:	GEN prime_ksearch
Input:	GEN relations, GEN extra
Input format:	<code>relations=[[p_1,s_1],...,[p_k,s_k]]</code> with p_i distinct integers and $s_i=-1, 1$, <code>extra=0</code> or <code>[n, c]</code>
Output format:	Prime number
Description:	Searches for a prime p such that $\text{kronecker}(p, p_i)=s_i$ for each i , and $p \equiv c \pmod{n}$ (if this is not 0). If the inputs are inconsistent and there is NO solution, this will not terminate.

Name:	GEN prime_ksearch_tc
Input:	GEN relations, GEN extra
Input format:	<code>relations=[[p_1,s_1],...,[p_k,s_k]]</code> with p_i distinct integers and $s_i=-1, 1$, <code>extra=0</code> or <code>[n, c]</code>
Output format:	Prime number
Description:	Checks that the inputs are in the correct format, and returns <code>prime_ksearch(relations, extra)</code> . Does NOT check that the inputs are consistent.

Name:	GEN QM_hnf
Input:	GEN M
Input format:	QM M
Output format:	QM
Description:	Returns the Hermite normal form of the rational matrix M with respect to the columns.

Name:	GEN QM_hnf_tc
Input:	GEN M
Input format:	QM M
Output format:	QM
Description:	Checks that M is a QM, and returns <code>QM_hnf(M)</code> .

Name:	<code>int Q_issquareall</code>
Input:	GEN <code>x</code> , GEN <code>*sqrtx</code>
Input format:	Rational <code>x</code> , pointer <code>sqrtx</code>
Output format:	0, 1
Description:	Returns 1 if <code>x</code> is a rational square, and 0 if not. If <code>x</code> is a rational square, sets <code>sqrtx</code> to its square root.

Name:	GEN <code>powerset</code>
Input:	GEN <code>L</code>
Input format:	Vector <code>L</code>
Output format:	Vector
Description:	Returns the powerset of <code>L</code> .

Name:	GEN <code>powerset_tc</code>
Input:	GEN <code>L</code>
Input format:	Vector <code>L</code>
Output format:	Vector
Description:	Checks that <code>L</code> is a vector, and returns <code>powerset(L)</code> .

Name:	GEN <code>vecratio</code>
Input:	GEN <code>v1</code> , GEN <code>v2</code>
Input format:	Vectors <code>v1</code> , <code>v2</code>
Output format:	Number
Description:	Assuming <code>v1</code> , <code>v2</code> are in the same one dimensional linear subspace, this returns <code>v1/v2</code> . If <code>v1=0</code> , returns 0, and if <code>v2=0</code> , returns ∞ .

Name:	GEN <code>vecratio_tc</code>
Input:	GEN <code>v1</code> , GEN <code>v2</code>
Input format:	Vectors <code>v1</code> , <code>v2</code>
Output format:	Number
Description:	Checks that <code>v1</code> , <code>v2</code> are vectors of the same length, and returns <code>vecratio(v1, v2)</code> .

7 qq_quat_int

Methods in this section deal with the computation of intersection numbers associated to optimal embeddings of positive discriminants in Eichler orders of indefinite quaternion algebras. See [Ric21] for more details.

7.1 Intersection number based on roots

This computation of the intersection number relies on very little theory and setup. It is good when the solution to Pell's equation for D_1 or D_2 is relatively small. There are also two static methods: GEN

`qa_inum_roots_f1bds` and `GEN qa_inum_roots_ghsearch`, which are used to find bounds for part of the computation. We will not describe them here: see the code for more details.

Name:	<code>GEN qa_inum_roots</code>
Input:	<code>GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, int data, long prec</code>
Input format:	<code>IQA Q, iEQO order, Qembs e1, e2</code> of discriminants <code>D1, D2, data=0, 1</code>
Output format:	<code>[pairs]</code> or <code>[[pairs], [[signed level, x]]</code>
Description:	Computes the intersection number of <code>e1, e2</code> via the roots method. If <code>data=0</code> , returns the set of pairs giving all non-simultaneously equivalent intersections. If <code>data=1</code> , then first element of the output is the set of pairs. If the i^{th} pair is x -linked with signed level ℓ , then the i^{th} entry of the second element of the output is <code>[x, ℓ]</code> .

Name:	<code>GEN qa_inum_roots_tc</code>
Input:	<code>GEN Q, GEN order, GEN e1, GEN e2, int data, long prec</code>
Input format:	<code>IQA Q, iEQO order, Qembs e1, e2, data=0, 1</code>
Output format:	<code>[pairs]</code> or <code>[[pairs], [[signed level, x]]</code>
Description:	Checks the inputs, and calls <code>qa_inum_roots</code> . The embeddings <code>e1, e2</code> need not be the image of $\frac{p_{D_i} + \sqrt{D_i}}{2}$ nor do they need to be optimal; this method replaces them with the corresponding optimal embedding.

7.2 Intersection number based on x-linking

This computation of the intersection number relies on the theory of x -linking, and the method `bqf_linearsolve`. While `qa_inum_roots` may be sometimes slightly faster when `D1=5, 8`, this method is overall much faster, and does not suffer from the Pell's equation shenanigans.

Name:	<code>GEN qa_inum_x</code>
Input:	<code>GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, int data, long prec</code>
Input format:	<code>IQA Q, iEQO order, Qembs e1, e2</code> of discriminants <code>D1, D2, data=0, 1</code>
Output format:	<code>[pairs]</code> or <code>[[pairs], [[signed level, x]]</code>
Description:	Computes the intersection number of <code>e1, e2</code> via the x -linking method. If <code>data=0</code> , returns the set of pairs giving all non-simultaneously equivalent intersections. If <code>data=1</code> , then first element of the output is the set of pairs. If the i^{th} pair is x -linked with signed level ℓ , then the i^{th} entry of the second element of the output is <code>[x, ℓ]</code> .

Name:	GEN qa_inum_x_tc
Input:	GEN Q, GEN order, GEN e1, GEN e2, int data, long prec
Input format:	IQA Q, iEQO order, Qembs e1, e2, data=0, 1
Output format:	[pairs] or [[pairs], [[signed level, x]]
Description:	Checks the inputs, and calls qa_inum_x. The embeddings e1, e2 need not be the image of $\frac{p_{D_i} + \sqrt{D_i}}{2}$ nor do they need to be optimal; this method replaces them with the corresponding optimal embedding.

Name:	GEN qa_xlink
Input:	GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, GEN x, long prec
Input format:	IQA Q, iEQO order, Qembs e1, e2 of discriminants D1, D2, integer x
Output format:	[pairs]
Description:	Computes all x-linking of e1, e2, and returns the set of x-linked pairs individually equivalent to e1, e2 but all non-simultaneously equivalent to each other.

Name:	GEN qa_xlink_tc
Input:	GEN Q, GEN order, GEN e1, GEN e2, GEN x, long prec
Input format:	IQA Q, iEQO order, Qembs e1, e2, integer x
Output format:	[pairs]
Description:	Checks the inputs, and calls qa_xlink. The embeddings e1, e2 need not be the image of $\frac{p_{D_i} + \sqrt{D_i}}{2}$ nor do they need to be optimal; this method replaces them with the corresponding optimal embedding.

Name:	GEN qa_xposs
Input:	GEN pset, GEN Psetprod, GEN D1, GEN D2, GEN xmin, GEN xmax
Input format:	pset even length vector of finite primes, Psetprod the product of pset, discriminants D1, D2, integers xmin, xmax
Output format:	Vector
Description:	Returns the set of x's in [xmin, xmax] for which there exists x-linked embeddings (not necessarily optimal) in the indefinite quaternion algebra ramified at Pset. If xmin and xmax are passed as 0, the method returns the x's in the range $[0, \sqrt{D_1 D_2})$.

Name:	GEN qa_xposs_tc
Input:	GEN Qorpset, GEN D1, GEN D2, GEN xmin, GEN xmax
Input format:	Qorpset even length vector of finite primes OR an IQA, discriminants D1, D2, integers xmin, xmax
Output format:	Vector
Description:	Checks the inputs and calls qa_xposs.

7.3 Intersection number based on fundamental domain

This computation of the intersection number relies upon a computed fundamental domain, and tracing out the root geodesics. It is by far the fastest computation, assuming that the fundamental domain has been pre-computed. Furthermore, the coefficients of the resulting pairs are small.

Name:	GEN <code>qa_inum_fd_givengeod</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>U</code> , GEN <code>geod1</code> , GEN <code>geod2</code> , GEN <code>pell1</code> , GEN <code>pell2</code> , GEN <code>D1D2</code> , int <code>data</code> , GEN <code>tol</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>iEQO</code> <code>order</code> , <code>fundamental domain</code> <code>U</code> , geodesics <code>geod1</code> , <code>geod2</code> that are the output of <code>qa_rootgeodesic_fd</code> on optimal embeddings of discriminants <code>D1</code> , <code>D2</code> , <code>pell_i=pell(D_i)</code> for $i=1, 2$, <code>D1D2=D1·D2</code> , <code>data=0, 1</code>
Output format:	<code>[pairs]</code> or <code>[[pairs], [[signed level, x]]</code>
Description:	Computes the intersection number of <code>geod1</code> , <code>geod2</code> via the fundamental domain method. If <code>data=0</code> , returns the set of pairs giving all non-simultaneously equivalent intersections. If <code>data=1</code> , then first element of the output is the set of pairs. If the i^{th} pair is x -linked with signed level ℓ , then the i^{th} entry of the second element of the output is <code>[x, ℓ]</code> .

Name:	GEN <code>qa_inum_fd_tc</code>
Input:	GEN <code>Q</code> , GEN <code>order</code> , GEN <code>U</code> , GEN <code>e1</code> , GEN <code>e2</code> , int <code>data</code> , long <code>prec</code>
Input format:	<code>IQA</code> <code>Q</code> , <code>iEQO</code> <code>order</code> , <code>fundamental domain</code> <code>U</code> , <code>Qembs</code> <code>e1</code> , <code>e2</code> , <code>data=0, 1</code>
Output format:	<code>[pairs]</code> or <code>[[pairs], [[signed level, x]]</code>
Description:	Finds the root geodesics of <code>e1</code> , <code>e2</code> , and calls <code>qa_inum_fd_given_geod</code> .

Name:	int <code>sides_int_indices</code>
Input:	long <code>i1</code> , long <code>i2</code> , long <code>j1</code> , long <code>j2</code>
Input format:	
Output format:	<code>-1, 0, 1</code>
Description:	In a fundamental domain, take geodesics going between sides $i1 \leq i2$ and $j1, j2$. This returns 1 if they are guaranteed to intersect, -1 if they do not, and 0 if this is not enough info (which is the case iff $\{i1, i2, j1, j2\}$ has at most 3 distinct elements).

7.4 Intersection series

These methods deal with computing the intersection series as described in Chapter 8 of [Ric21].

Name:	GEN qa_inumseries
Input:	GEN Q, GEN order, GEN U, GEN e1, GEN e2, GEN D1, GEN D2, GEN N, int type, long prec
Input format:	IQA Q, iEQO order, fundamental domain U, Qembs e1, e2 of positive discriminants D1, D2, positive integer N, type=0, 1, prime
Output format:	Vector
Description:	Returns the intersection series up to exponent N, i.e. the sum of the intersection of e1 with $T_n[e2]$ for all $n \leq N$. The i^{th} entry of the return is the coefficient of q^i in the series. The variable type determines which intersection series: 0 means unsigned, 1 means signed, and q where $q > 1$ is a prime means the q-weighted series.

Name:	GEN qa_inumseries_tc
Input:	GEN Q, GEN order, GEN U, GEN e1, GEN e2, GEN N, int type, long prec
Input format:	IQA Q, iEQO order, fundamental domain U, Qembs e1, e2 of positive discriminants, positive integer N, type=0, 1, prime
Output format:	Vector
Description:	Checks the inputs and returns qa_inumseries(...).

7.5 Intersection data

These methods deal with the computation of data associated to intersection, for example the signed level.

Name:	GEN qa_intlevel
Input:	GEN Q, GEN order, GEN e1, GEN e2, GEN D1D2, long prec
Input format:	QA Q, iEQO order, Qembs e1, e2 of discriminants D1, D2, D1D2=D1*D2
Output format:	[signed level, x]
Description:	If e1, e2 represent optimal embeddings ϕ_1, ϕ_2 , let $z = \phi_1(\sqrt{D_1})\phi_2(\sqrt{D_2})$. Then z has trace $2x$ and corresponds to an optimal embedding of discriminant $\frac{x^2 - D_1 D_2}{\ell^2}$, where ℓ is the level. This returns the pair $[\pm \ell, x]$, where the \pm is the sign of the intersection (or 1 if $x^2 > D_1 D_2$ and there is no intersection).

Name:	GEN qa_intlevel_tc
Input:	GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, long prec
Input format:	QA Q, iEQO order, Qembs e1, e2 of discriminants D1, D2
Output format:	[signed level, x]
Description:	Checks the inputs and calls qa_intlevel. D1, D2 can be passed as 0, and they will be automatically set.

Name:	GEN <code>qa_inum_fromdata</code>
Input:	GEN <code>data</code> , int <code>type</code>
Input format:	Intersection data <code>Data</code> , <code>type=0</code> , <code>1</code> , prime
Output format:	Integer
Description:	Given the data associated to an intersection (the second output of an intersection method with <code>data</code>), returns the corresponding intersection number. <code>type=0</code> means unsigned, <code>1</code> means signed, and <code>q</code> where $q > 1$ is a prime means <code>q</code> -weighted.

8 qq_visual

These methods deal with the visualization of data. At the moment, they only include methods to bin data for histograms, and display the data.

8.1 Histograms

Since users will likely want to adjust the histograms created with these methods, this part of the package is best used in GP. See [Ric20] for a detailed description on how to use these methods in GP. In library mode, the most likely methods one would use are `hist_make`, `hist_tobins`, and possibly `hist_scale`. Since many of the user-supplied variable names are shared among methods, we describe them first, rather than repeat them in each method description.

- GEN `data`: The sorted (increasing order) raw data that you want to make a histogram with. Should be real numbers, but does not need to be of type `t_REAL`.
- GEN `histdata`: The output of any of the functions in this section returning a GEN, storing information about the histogram. Typically will not be created or altered by the user. The format is a length 8 vector, with entries (all translated into GENs):

`minx, maxx, nbins, scale, imagename, autofile, plotoptions, open.`

- GEN `minx`, GEN `maxx`: The boundaries of the bins and hence the histogram.
- GEN `nbins`: The number of bins
- int `compilenew`: set to 1 if the LaTeX document specified by `autofile` has NOT yet been written, and needs to be written. If the document already exists, set this to 0 so as to not overwrite it.
- int `open`: set to 1 if you want the PDF to automatically open, and 0 otherwise. This only works on the Linux subsystem for Windows, so please set to 0 if this is not the case.
- int `scale`: if 0, the histogram y-axis will be the the absolute counts for each bin. If 1, scales the counts so the entire graph has area 1. It is useful to scale the histogram if you are trying to fit the data to a function.
- long `prec`: the precision.

- `char *autofile`: The file name for the LaTeX file, without the `.tex` suffix. This should be found in the folder `“/images/build”`.
- `char *imagename`: The name of the tikz image created.
- `char *plotoptions`: Set to NULL if you want everything done automatically, or have created the LaTeX document yourself. If set to non-null, inserts the character string between `“\begin{axis}”` and `“\end{axis}”`. Thus it is useful to completely customize how you want the histogram to look like.

Note that the created PDF document will reside in the subfolder `“/images”`, and the LaTeX document and all the build files will reside in the subfolder `“/images/build”` (this helps keep the clutter of files sequestered). The methods will create these folders if they do not exist yet.

Name:	<code>void hist_autocompile</code>
Input:	<code>GEN minx, GEN maxx, char *imagename, char *autofile, char *plotoptions, int open</code>
Input format:	See bullets near top of Section 8.1
Output format:	
Description:	Writes the LaTeX document <code>autofile</code> automatically, using <code>plotoptions</code> if it is non-NULL.

Name:	<code>void hist_compile</code>
Input:	<code>char *imagename, char *autoname, int open</code>
Input format:	See bullets near top of Section 8.1
Output format:	
Description:	Compiles the LaTeX document <code>autofile</code> , and opens it if <code>open=1</code> .

Name:	<code>GEN hist_make</code>
Input:	<code>GEN data, char *imagename, char *autofile, int compileneu, char *plotoptions, int open, long prec</code>
Input format:	See bullets near top of Section 8.1
Output format:	Vector
Description:	Initiates the making of the histogram with default bins and <code>minx, maxx</code> . Should be called at most once per histogram.

Name:	<code>GEN hist_tobins</code>
Input:	<code>GEN data, GEN minx, GEN maxx, GEN nbins, int toscale, int compileneu, char *imagename, char *autofile, char *plotoptions, int open, long prec</code>
Input format:	See bullets near top of Section 8.1
Output format:	Vector
Description:	This method does the binning of the data according to the inputs, and calls the appropriate submethods to make and compile the LaTeX document. Should be called at most once per histogram.

Name:	GEN hist_tobins_defaultbins
Input:	GEN data, GEN minx, GEN maxx, int toscale, int compilenew, char *imagename, char *autofile, char *plotoptions, int open, long prec
Input format:	See bullets near top of Section 8.1
Output format:	Vector
Description:	Finds the default number of bins, according to the Freedman-Diaconis rule of bin width= $2IQR/(n^{1/3})$ (n data points), and calls hist_tobins.

Name:	GEN hist_rebin
Input:	GEN data, GEN histdata, GEN nbins, long prec
Input format:	See bullets near top of Section 8.1
Output format:	Vector
Description:	Remakes the histogram according to the new nbins.

Name:	void hist_recompile
Input:	GEN histdata
Input format:	See bullets near top of Section 8.1
Output format:	
Description:	Recompiles the LaTeX document. Used when the LaTeX document was modified manually (hence it is unlikely to be useful in library mode).

Name:	GEN hist_rerange
Input:	GEN data, GEN histdata, GEN minx, GEN maxx, long prec
Input format:	See bullets near top of Section 8.1
Output format:	Vector
Description:	Remakes the histogram according to the new range.

Name:	GEN hist_rescale
Input:	GEN data, GEN histdata, int scale, long prec
Input format:	See bullets near top of Section 8.1
Output format:	Vector
Description:	Remakes the histogram by either scaling it or not.

9 Method declarations

Methods in this section are divided into subsections by the files, and into subsubsections by their general function. They will appear approximately alphabetically in each subsubsection, with the static methods always appearing at the bottom. Clicking on a method name will bring you to its full description in the previous sections.

9.1 qq_base

9.1.1 Infinity

GEN	addoo	GEN a, GEN b
GEN	divoo	GEN a, GEN b

9.1.2 Integer vectors

GEN	ZV_copy	GEN v
GEN	ZV_Z_divexact	GEN v, GEN y
GEN	ZV_Z_mul	GEN v, GEN x
int	ZV_equal	GEN v1, GEN v2

9.1.3 Linear algebra

GEN	FpM_eigenvecs	GEN M, GEN p
GEN	lin_intsolve	GEN A, GEN B, GEN n
GEN	lin_intsolve_tc	GEN A, GEN B, GEN n
GEN	mat3_complete	GEN A, GEN B, GEN C
GEN	mat3_complete_tc	GEN A, GEN B, GEN C

9.1.4 Lists

void	clist_free	clist *l, long length
void	clist_putafter	clist **head_ref, GEN new_data
void	clist_putbefore	clist **head_ref, GEN new_data
GEN	clist_togvec	clist *l, long length, int dir
void	glist_free	glist *l
GEN	glist_pop	glist **head_ref
void	glist_putstart	glist **head_ref, GEN new_data
GEN	glist_togvec	glist *l, long length, int dir
GEN	glist_togvec_append	glist *l, GEN v, long length, int dir
void	llist_free	llist *l
long	llist_pop	llist **head_ref
void	llist_putstart	llist **head_ref, long new_data
GEN	llist_togvec	llist *l, long length, int dir
GEN	llist_tovecsmall	llist *l, long length, int dir

9.1.5 Random

GEN	rand_elt	GEN v
long	rand_l	long len

9.1.6 Short vectors in lattices

GEN	lat_smallvectors	GEN A, GEN C1, GEN C2, GEN condition, int onesign, int isintegral, int rdataonly, long prec
GEN	lat_smallvectors_givendata	GEN chol, GEN U, GEN perminv, GEN C1, GEN C2, GEN condition, int onesign, long prec
GEN	lat_smallvectors_tc	GEN A, GEN C1, GEN C2, int onesign, int isintegral, long prec
GEN	lat_smallvectors_cholesky	GEN Q, GEN C1, GEN C2, GEN condition, int onesign, long prec
GEN	mat_choleskydecomp	GEN A, int rcoefs, long prec
GEN	mat_choleskydecomp_tc	GEN A, int rcoefs, long prec
GEN	quadraticinteger	GEN A, GEN B, GEN C
int	opp_gcmp	void *data, GEN x, GEN y

9.1.7 Time

void	printtime	void
char*	returntime	void

9.2 qq_bqf

9.2.1 Discriminant methods

GEN	disclist	GEN D1, GEN D2, int fund, GEN cop
GEN	discprimeindex	GEN D, GEN facs
GEN	discprimeindex_tc	GEN D
int	isdisc	GEN D
GEN	pell	GEN D
GEN	pell_tc	GEN D
GEN	posreg	GEN D, long prec
GEN	posreg_tc	GEN D, long prec
GEN	quadroot	GEN D
GEN	quadroot_tc	GEN D

9.2.2 Basic methods for binary quadratic forms

GEN	bqf_automorph_tc	GEN q
int	bqf_compare	void *data, GEN q1, GEN q2
int	bqf_compare_tmat	void *data, GEN d1, GEN d2
GEN	bqf_disc	GEN q
GEN	bqf_disc_tc	GEN q
GEN	bqf_isequiv	GEN q1, GEN q2, GEN rootD, int Dsign, int tmat
GEN	bqf_isequiv_set	GEN q, GEN S, GEN rootD, int Dsign, int tmat

GEN	bqf_isequiv_tc	GEN q1, GEN q2, int tmat, long prec
int	bqf_isreduced	GEN q, int Dsign
int	bqf_isreduced_tc	GEN q
GEN	bqf_random	GEN maxc, int type, int primitive
GEN	bqf_random_D	GEN maxc, GEN D
GEN	bqf_red	GEN q, GEN rootD, int Dsign, int tmat
GEN	bqf_red_tc	GEN q, int tmat, long prec
GEN	bqf_roots	GEN q, GEN D, GEN w
GEN	bqf_roots_tc	GEN q
GEN	bqf_trans	GEN q, GEN M
GEN	bqf_trans_tc	GEN q, GEN M
GEN	bqf_transL	GEN q, GEN n
GEN	bqf_transR	GEN q, GEN n
GEN	bqf_transS	GEN q
GEN	bqf_trans_coprime	GEN q, GEN n
GEN	bqf_trans_coprime_tc	GEN q, GEN n
GEN	ideal_tobqf	GEN numf, GEN ideal

9.2.3 Basic methods, but specialized

GEN	dbqf_automorph	GEN q, GEN D
GEN	dbqf_isequiv	GEN q1, GEN q2
long	dbqf_isequiv_set	GEN q, GEN S
GEN	dbqf_isequiv_set_tmat	GEN q, GEN S
GEN	dbqf_isequiv_tmat	GEN q1, GEN q2
GEN	dbqf_red	GEN q
GEN	dbqf_red_tmat	GEN q
GEN	ibqf_automorph_D	GEN q, GEN D
GEN	ibqf_automorph_pell	GEN q, GEN qpell
GEN	ibqf_isequiv	GEN q1, GEN q2, GEN rootD
long	ibqf_isequiv_set_byq	GEN q, GEN S, GEN rootD
long	ibqf_isequiv_set_byq_presorted	GEN qredssorted, GEN S, GEN rootD
GEN	ibqf_isequiv_set_byq_tmat	GEN q, GEN S, GEN rootD
GEN	ibqf_isequiv_set_byq_tmat_presorted	GEN qredssorted, GEN S, GEN rootD
long	ibqf_isequiv_set_byS	GEN q, GEN S, GEN rootD
long	ibqf_isequiv_set_byS_presorted	GEN q, GEN Sreds, GEN perm, GEN rootD
GEN	ibqf_isequiv_set_byS_tmat	GEN q, GEN S, GEN rootD
GEN	ibqf_isequiv_set_byS_tmat_presorted	GEN q, GEN Sreds, GEN perm, GEN rootD
GEN	ibqf_isequiv_tmat	GEN q1, GEN q2, GEN rootD
GEN	ibqf_red	GEN q, GEN rootD
GEN	ibqf_red_tmat	GEN q, GEN rootD

GEN	ibqf_red_pos	GEN q, GEN rootD
GEN	ibqf_red_pos_tmat	GEN q, GEN rootD

9.2.4 Basic methods for indefinite quadratic forms

int	ibqf_isrecip	GEN q, GEN rootD
int	ibqf_isrecip_tc	GEN q, long prec
GEN	ibqf_leftnbr	GEN q, GEN rootD
GEN	ibqf_leftnbr_tmat	GEN q, GEN rootD
GEN	ibqf_leftnbr_tc	GEN q, int tmat, long prec
GEN	ibqf_leftnbr_update	GEN qvec, GEN rootD
GEN	ibqf_redorbit	GEN q, GEN rootD
GEN	ibqf_redorbit_tmat	GEN q, GEN rootD
GEN	ibqf_redorbit_posonly	GEN q, GEN rootD
GEN	ibqf_redorbit_posonly_tmat	GEN q, GEN rootD
GEN	ibqf_redorbit_tc	GEN q, int tmat, int posonly, long prec
GEN	ibqf_rightnbr	GEN q, GEN rootD
GEN	ibqf_rightnbr_tmat	GEN q, GEN rootD
GEN	ibqf_rightnbr_tc	GEN q, int tmat, long prec
GEN	ibqf_rightnbr_update	GEN qvec, GEN rootD
GEN	ibqf_river	GEN q, GEN rootD
GEN	ibqf_river_positions	GEN q, GEN rootD
GEN	ibqf_river_positions_forms	GEN q, GEN rootD
GEN	ibqf_river_tc	GEN q, long prec
GEN	ibqf_riverforms	GEN q, GEN rootD
GEN	ibqf_riverforms_tc	GEN q, long prec
GEN	ibqf_symmetricarc	GEN q, GEN D, GEN rootD, GEN qpell, long prec
GEN	ibqf_symmetricarc_tc	GEN q, long prec
GEN	ibqf_toriver	GEN q, GEN rootD
GEN	ibqf_toriver_tmat	GEN q, GEN rootD
GEN	mat_toibqf	GEN M
GEN	mat_toibqf_tc	GEN M

9.2.5 Class group and composition of forms

GEN	bqf_comp	GEN q1, GEN q2
GEN	bqf_comp_red	GEN q1, GEN q2, GEN rootD, int Dsign
GEN	bqf_comp_tc	GEN q1, GEN q2, int tored, long prec
GEN	bqf_idelt	GEN D
GEN	bqf_ncgp	GEN D, long prec
GEN	bqf_ncgp_lexic	GEN D, long prec
GEN	bqf_pow	GEN q, GEN n
GEN	bqf_pow_red	GEN q, GEN n, GEN rootD, int Dsign
GEN	bqf_pow_tc	GEN q, GEN n, int tored, long prec
GEN	bqf_square	GEN q

GEN	bqf_square_red	GEN q, GEN rootD, int Dsign
GEN	bqf_square_tc	GEN q, int tored, long prec
GEN	bqf_ncgp_nonfundnarrow	GEN cgp, GEN D, GEN rootD

9.2.6 Representation of integers by forms

GEN	bqf_bigreps	GEN q, GEN n, long prec
GEN	bqf_bigreps_tc	GEN q, GEN n, long prec
GEN	bqf_linearsolve	GEN q, GEN n1, GEN lin, GEN n2, long prec
GEN	bqf_linearsolve_tc	GEN q, GEN n1, GEN lin, GEN n2, long prec
GEN	bqf_reps	GEN q, GEN n, int proper, int half, long prec
GEN	bqf_reps_tc	GEN q, GEN n, int proper, int half, long prec
GEN	dbqf_reps	GEN qred, GEN D, GEN n, int proper, int half
GEN	ibqf_reps	GEN qorb, GEN qautom, GEN D, GEN rootD, GEN n, int proper, int half
GEN	sbqf_reps	GEN q, GEN D, GEN rootD, GEN n, int half
GEN	zbqf_reps	GEN A, GEN B, GEN n, int half
GEN	zbqf_bigreps	GEN q, GEN n
GEN	bqf_bigreps_creatervecfin	GEN newsols, GEN a, GEN b, GEN disc
GEN	bqf_bigreps_creatervecpos	GEN newsols, GEN a, GEN b, GEN disc
GEN	bqf_bigreps_creatervecclin	GEN newsols, GEN a, GEN b, GEN disc
GEN	bqf_reps_all	GEN n
GEN	bqf_reps_creatervec	glist *sols, glist *scale, llist *nsolslist, long *totnsols, long *count, int half
GEN	bqf_reps_creatervec_proper	glist *sols, long nsols, int half
GEN	bqf_reps_makeprimitive	GEN q, GEN *n
GEN	bqf_reps_trivial	void
void	bqf_reps_updatesolutions	glist **sols, long *nsols, GEN *a, GEN *b
void	dbqf_reps_proper	GEN qred, GEN D, GEN n, glist **sols, long *nsols, GEN f, int *terminate
void	ibqf_reps_proper	GEN qorb, GEN D, GEN rootD, GEN n, glist **sols, long *nsols, GEN f, int *terminate
GEN	bqf_linearsolve_zall	GEN yzsols, GEN n2, GEN Minv
GEN	bqf_linearsolve_zfin	GEN yzsols, GEN n2, GEN Minv
GEN	bqf_linearsolve_zlin	GEN yzsols, GEN n2, GEN Minv
GEN	bqf_linearsolve_zpos	GEN yzsols, GEN n2, GEN Minv, GEN M
GEN	bqf_linearsolve_zquad	GEN yzsols, GEN n2, GEN Minv

9.2.7 Checking GP inputs

void	bqf_check	GEN q
GEN	bqf_checkdisc	GEN q
void	intmatrix_check	GEN mtx

9.3 qq_bqf_int

9.3.1 Intersection Data

GEN	bqf_bdelta	GEN q1, GEN q2
GEN	bqf_bdelta_tc	GEN q1, GEN q2
GEN	bqf_intlevel	GEN q1, GEN q2
GEN	bqf_intlevel_tc	GEN q1, GEN q2
GEN	ibqf_intpairs_transtoq	GEN pairs, GEN q, GEN rootD
GEN	ibqf_intpoint	GEN q1, GEN q2, GEN location, GEN autom
GEN	ibqf_intpoint_tc	GEN q1, GEN q2, GEN location
GEN	bqf_iiform	GEN q1, GEN q2

9.3.2 Intersection number computation

GEN	ibqf_int	GEN r1, GEN r2
GEN	ibqf_int_tc	GEN q1, GEN q2, long prec
GEN	ibqf_intrs_byriver	GEN r1, GEN r2
GEN	ibqf_intrs_tc	GEN q1, GEN q2, long prec
GEN	ibqf_intforms_byriver	GEN r1, GEN r2, int data, int split
GEN	ibqf_intforms_tc	GEN q1, GEN q2, int data, int split, long prec
GEN	ibqf_intformsRS_byriver	GEN r1, GEN r2, int data
GEN	ibqf_intformsRS_tc	GEN q1, GEN q2, int data, long prec
GEN	ibqf_intformsRO_byriver	GEN r1, GEN r2, int data
GEN	ibqf_intformsRO_tc	GEN q1, GEN q2, int data, long prec
GEN	ibqf_intformsLS_byriver	GEN r1, GEN r2, int data
GEN	ibqf_intformsLS_tc	GEN q1, GEN q2, int data, long prec
GEN	ibqf_intformsLO_byriver	GEN r1, GEN r2, int data
GEN	ibqf_intformsLO_tc	GEN q1, GEN q2, int data, long prec
GEN	ibqf_int_reverseriver	GEN r
GEN	ibqf_intrs_splitindices	GEN river, GEN ind
void	ibqf_intformsRS_byriver_ indices	GEN r1, GEN r2, llist **inds1, llist **inds2, llist **looverlap, long *inum, int data

9.4 qq_geometry

9.4.1 Basic line, circle, and point operations

GEN	arc_init	GEN c, GEN p1, GEN p2, int dir, long prec
GEN	arc_init_tc	GEN c, GEN p1, GEN p2, int dir, long prec

GEN	arc_midpoint	GEN c, GEN p1, GEN p2, GEN tol, long prec
GEN	arc_midpoint_tc	GEN c, GEN p1, GEN p2, long prec
GEN	circle_angle	GEN c1, GEN c2, GEN p, GEN tol, long prec
GEN	circle_angle_tc	GEN c1, GEN c2, GEN p, long prec
GEN	circle_fromcp	GEN cent, GEN p, long prec
GEN	circle_fromppp	GEN p1, GEN p2, GEN p3, GEN tol, long prec
GEN	circle_fromppp_tc	GEN p1, GEN p2, GEN p3, long prec
GEN	circle_tangentslope	GEN c, GEN p, long prec
GEN	circle_tangentslope_tc	GEN c, GEN p, long prec
GEN	crossratio	GEN a, GEN b, GEN c, GEN d
GEN	line_angle	GEN l1, GEN l2, long prec
GEN	line_fromsp	GEN s, GEN p
GEN	line_frompp	GEN p1, GEN p2
GEN	mat_eval	GEN M, GEN x
GEN	mat_eval_tc	GEN M, GEN x
GEN	midpoint	GEN p1, GEN p2
GEN	mobius	GEN M, GEN c, GEN tol, long prec
GEN	mobius_tc	GEN M, GEN c, long prec
GEN	mobius_arcseg	GEN M, GEN c, int isarc, GEN tol, long prec
GEN	mobius_circle	GEN M, GEN c, GEN tol, long prec
GEN	mobius_line	GEN M, GEN l, GEN tol, long prec
GEN	perpbis	GEN p1, GEN p2
GEN	radialangle	GEN c, GEN p, GEN tol, long prec
GEN	radialangle_tc	GEN c, GEN p, long prec
GEN	slope	GEN p1, GEN p2

9.4.2 Intersection of lines and circles

GEN	arc_int	GEN c1, GEN c2, GEN tol, long prec
GEN	arc_int_tc	GEN c1, GEN c2, long prec
GEN	arcseg_int	GEN c, GEN l, GEN tol, long prec
GEN	arcseg_int_tc	GEN c, GEN l, long prec
GEN	circle_int	GEN c1, GEN c2, GEN tol, long prec
GEN	circle_int_tc	GEN c1, GEN c2, long prec
GEN	circleline_int	GEN c, GEN l, GEN tol, long prec
GEN	circleline_int_tc	GEN c, GEN l, long prec
GEN	gensseg_int	GEN s1, GEN s2, GEN tol, long prec
GEN	gensseg_int_tc	GEN s1, GEN s2, long prec
GEN	line_int	GEN l1, GEN l2, GEN tol, long prec
GEN	line_int_tc	GEN l1, GEN l2, long prec
int	onarc	GEN c, GEN p, GEN tol, long prec
int	onarc_tc	GEN c, GEN p, long prec
int	onseg	GEN l, GEN p, GEN tol, long prec
int	onseg_tc	GEN l, GEN p, long prec

GEN	seg_int	GEN l1, GEN l2, GEN tol, long prec
GEN	seg_int_tc	GEN l1, GEN l2, long prec

9.4.3 Hyperbolic distance and area

GEN	hdist	GEN z1, GEN z2, long prec
GEN	hdist_tc	GEN z1, GEN z2, long prec
GEN	hdist_ud	GEN z1, GEN z2, long prec
GEN	hpolygon_area	GEN circles, GEN vertices, GEN tol, long prec
GEN	hpolygon_area_tc	GEN circles, GEN vertices, long prec

9.4.4 Helper methods

GEN	anglediff	GEN ang, GEN bot, GEN tol, long prec
GEN	atanoo	GEN x, long prec
GEN	deftol	long prec
int	gcmp_strict	void *data, GEN x, GEN y
int	geom_check	GEN c
GEN	shiftangle	GEN ang, GEN bot, long prec
long	tolcmp	GEN x, GEN y, GEN tol, long prec
int	tolcmp_sort	void *data, GEN x, GEN y
int	toleq	GEN x, GEN y, GEN tol, long prec

9.5 qq-quat

9.5.1 Basic operations on elements in quaternion algebras

GEN	qa_conj	GEN x
GEN	qa_conj_tc	GEN x
GEN	qa_conjby	GEN Q, GEN x, GEN y
GEN	qa_conjby_tc	GEN Q, GEN x, GEN y
GEN	qa_inv	GEN Q, GEN x
GEN	qa_inv_tc	GEN Q, GEN x
GEN	qa_m2rembed	GEN Q, GEN x
GEN	qa_m2rembed_tc	GEN Q, GEN x
GEN	qa_minpoly	GEN Q, GEN x
GEN	qa_minpoly_tc	GEN Q, GEN x
GEN	qa_mul	GEN Q, GEN x, GEN y
GEN	qa_mul_tc	GEN Q, GEN x, GEN y
GEN	qa_mulsets	GEN Q, GEN S1, GEN S2
GEN	qa_mulvec	GEN Q, GEN L
GEN	qa_mulvec_tc	GEN Q, GEN L
GEN	qa_mulvecindices	GEN Q, GEN L, GEN indices
GEN	qa_mulvecindices_tc	GEN Q, GEN L, GEN indices
GEN	qa_norm	GEN Q, GEN x

GEN	qa_norm_tc	GEN Q, GEN x
GEN	qa_pow	GEN Q, GEN x, GEN n
GEN	qa_pow_tc	GEN Q, GEN x, GEN n
GEN	qa_roots	GEN Q, GEN x, long prec
GEN	qa_roots_tc	GEN Q, GEN x, long prec
GEN	qa_square	GEN Q, GEN x
GEN	qa_square_tc	GEN Q, GEN x
GEN	qa_trace	GEN x
GEN	qa_trace_tc	GEN x

9.5.2 Basic operations on orders and lattices in quaternion algebras

int	qa_isinorder	GEN Q, GEN ordinv, GEN x
int	qa_isinorder_tc	GEN Q, GEN ord, GEN x
int	qa_isorder	GEN Q, GEN ord, GEN ordinv
int	qa_isorder_tc	GEN Q, GEN ord
GEN	qa_leftorder	GEN Q, GEN L, GEN Linv
GEN	qa_leftorder_tc	GEN Q, GEN L
GEN	qa_rightorder	GEN Q, GEN L, GEN Linv
GEN	qa_rightorder_tc	GEN Q, GEN L
GEN	qa_ord_conj	GEN Q, GEN ord, GEN c
GEN	qa_ord_conj_tc	GEN Q, GEN ord, GEN c
GEN	qa_ord_disc	GEN Q, GEN ord
GEN	qa_ord_disc_tc	GEN Q, GEN ord
GEN	qa_ord_normform	GEN Q, GEN ord
GEN	qa_ord_type	GEN Q, GEN ord, GEN level
GEN	qa_superorders	GEN Q, GEN ord, GEN n
GEN	qa_superorders_prime	GEN Q, GEN ord, GEN ordinv, GEN n
GEN	qa_superorders_tc	GEN Q, GEN ord, GEN n

9.5.3 Initialization methods

GEN	qa_eichlerorder	GEN Q, GEN l, GEN maxord
GEN	qa_eichlerorder_tc	GEN Q, GEN l, GEN maxord
GEN	qa_maximalorder	GEN Q, GEN baseord
GEN	qa_maximalorder_tc	GEN Q, GEN baseord
GEN	qa_ord_init	GEN Q, GEN ord
GEN	qa_ord_init_tc	GEN Q, GEN ord
GEN	qa_ord_init_traceObasis	GEN Q, GEN ord, GEN maxds
GEN	qa_init_ab	GEN a, GEN b
GEN	qa_init_ab_tc	GEN a, GEN b
GEN	qa_init_primes	GEN pset, int type
GEN	qa_init_primes_tc	GEN pset
GEN	qa_init_m2z	
GEN	qa_init_2primes	GEN p, GEN q
GEN	qa_init_2primes_tc	GEN p, GEN q

GEN	qa_ram_fromab	GEN a, GEN b
GEN	qa_ram_fromab_tc	GEN a, GEN b

9.5.4 Conjugation of elements in a given order

GEN	qa_conjbasis	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2, int orient
GEN	qa_conjbasis_tc	GEN Q, GEN ord, GEN e1, GEN e2, int orient
GEN	qa_conjbasis_orient	GEN Q, GEN ord, GEN v1, GEN v2, GEN e2
GEN	qa_conjqf	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2
GEN	qa_conjqf_tc	GEN Q, GEN ord, GEN e1, GEN e2
GEN	qa_conjnorm	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2, GEN n, int retconelt, long prec
GEN	qa_conjnorm_tc	GEN Q, GEN ord, GEN e1, GEN e2, GEN n, int retconelt, long prec
GEN	qa_simulconj	GEN Q, GEN ord, GEN ordinv, GEN e1, GEN e2, GEN f1, GEN f2, long prec
GEN	qa_simulconj_tc	GEN Q, GEN ord, GEN e1, GEN e2, GEN f1, GEN f2, long prec

9.5.5 Embedding quadratic orders into Eichler orders

GEN	qa_associatedemb	GEN Q, GEN order, GEN emb, GEN D
GEN	qa_associatedemb_tc	GEN Q, GEN order, GEN emb, GEN D
GEN	qa_embed	GEN Q, GEN order, GEN D, GEN nembeds, GEN rpell, long prec
GEN	qa_embed_tc	GEN Q, GEN order, GEN D, GEN nembeds, int rpell, long prec
int	qa_embed_isnewoptimal	GEN Q, GEN ord, GEN ordinv, GEN D, GEN Dmod2, GEN dfacs, GEN emb, GEN gcdf1g1h1, GEN embs, long pos, long prec
GEN	qa_embeddablediscs	GEN Q, GEN order, GEN d1, GEN d2, int fund, GEN cop
GEN	qa_embeddablediscs_tc	GEN Q, GEN order, GEN d1, GEN d2, int fund, GEN cop
GEN	qa_numemb	GEN Q, GEN order, GEN D, GEN narclno
GEN	qa_numemb_tc	GEN Q, GEN order, GEN D, GEN narclno, long prec
GEN	qa_ordiffer	GEN Q, GEN order, GEN e1, GEN e2, GEN D
GEN	qa_ordiffer_tc	GEN Q, GEN order, GEN e1, GEN e2, GEN D
GEN	qa_orinfinite	GEN Q, GEN emb, GEN D, long prec
GEN	qa_orinfinite_tc	GEN Q, GEN emb, GEN D, long prec
GEN	qa_sortedembed	GEN Q, GEN order, GEN D, GEN rpell, GEN ncgp, long prec

GEN	qa_sortedembed_tc	GEN Q, GEN order, GEN D, int rpell, GEN ncgp, long prec
int	qa_embedor_compare	void *data, GEN pair1, GEN pair2

9.5.6 Elements of norm n in an Eichler order

GEN	qa_orbitrepsprime	GEN Q, GEN order, GEN p, long prec
int	qa_orbitrepsprime_checksol	GEN Q, GEN order, GEN T, GEN W, GEN *reps, long *place, GEN sol
GEN	qa_orbitrepsprimepower	GEN Q, GEN order, GEN p, GEN e, long prec
GEN	qa_orbitreps	GEN Q, GEN order, GEN n, long prec
GEN	qa_orbitreps_tc	GEN Q, GEN order, GEN n, long prec
GEN	qa_orbitrepsrange	GEN Q, GEN order, GEN n, long prec
GEN	qa_orbitrepsrange_tc	GEN Q, GEN order, GEN n, long prec
GEN	qa_hecke	GEN Q, GEN order, GEN n, GEN emb, GEN D, long prec
GEN	qa_hecke_tc	GEN Q, GEN order, GEN n, GEN emb, long prec

9.5.7 Fundamental domain methods

GEN	qa_fundamentaldomain	GEN Q, GEN order, GEN p, int dispprogress, GEN ANRdata, GEN tol, long prec
GEN	qa_fundamentaldomain_tc	GEN Q, GEN order, GEN p, int dispprogress, GEN ANRdata, long prec
GEN	qa_invradqf	GEN Q, GEN order, GEN mats, GEN z, long prec
GEN	qa_isometriccircle	GEN Q, GEN x, GEN mats, GEN tol, long prec
GEN	qa_isometriccircle_tc	GEN Q, GEN x, GEN p, long prec
GEN	qa_fdarea	GEN Q, GEN order, long prec
GEN	qa_fdarea_tc	GEN Q, GEN order, long prec
GEN	qa_fdm2rembed	GEN *data, GEN x, long prec
GEN	qa_fdm2inv	GEN *data, GEN x
GEN	qa_fdm2mul	GEN *data, GEN x, GEN y
int	qa_istriv	GEN *data, GEN x
GEN	qa_normalizedbasis	GEN Q, GEN G, GEN mats, GEN U, GEN tol, long prec
GEN	qa_normalizedbasis_tc	GEN Q, GEN G, GEN p, long prec
GEN	qa_normalizedboundary	GEN Q, GEN G, GEN mats, GEN tol, long prec
GEN	qa_normalizedboundary_tc	GEN Q, GEN G, GEN p, long prec
void	qa_printisometriccircles	GEN Q, GEN L, char *filename, GEN mats, GEN tol, long prec
void	qa_printisometriccircles_tc	GEN Q, GEN L, GEN p, char *filename, int view, long prec

GEN	qa_reduceelt	GEN Q, GEN G, GEN x, GEN z, GEN p, GEN tol, long prec
GEN	qa_reduceelt_normbound	GEN Q, GEN U, GEN x, GEN z, GEN tol, long prec
GEN	qa_reduceelt_tc	GEN Q, GEN G, GEN x, GEN z, GEN p, long prec
GEN	qa_rootgeodesic_fd	GEN Q, GEN U, GEN g, GEN tol, long prec
GEN	qa_rootgeodesic_fd_tc	GEN Q, GEN U, GEN g, long prec
GEN	qa_smallnorm1elts_invrad	GEN Q, GEN order, GEN C1, GEN C2, GEN invrad, long prec
GEN	qa_smallnorm1elts_tc	GEN Q, GEN order, GEN p, GEN z, GEN C1, GEN C2, long prec
GEN	qa_topsu	GEN Q, GEN g, GEN p, long prec
GEN	qa_topsu_mat	GEN Q, GEN g, GEN mats, long prec
GEN	qa_topsu_tc	GEN Q, GEN g, GEN p, long prec

9.5.8 Checking methods

void	qa_check	GEN Q
void	qa_indefcheck	GEN Q
void	qa_eltcheck	GEN x
GEN	qa_ordcheck	GEN ord
void	qa_ordeichlercheck	GEN order
void	QM_check	GEN M

9.5.9 Property retrieval

GEN	qa_getnf	GEN Q
GEN	qa_getpram	GEN Q
GEN	qa_getabvec	GEN Q
GEN	qa_geta	GEN Q
GEN	qa_getb	GEN Q
GEN	qa_getmab	GEN Q
GEN	qa_getpramprod	GEN Q
GEN	qa_getord	GEN order
GEN	qa_getordtype	GEN order
GEN	qa_getordmaxd	GEN order
GEN	qa_getordlevel	GEN order
GEN	qa_getordlevelpfac	GEN order
GEN	qa_getordinv	GEN order
GEN	qa_getordtrace0basis	GEN order

9.5.10 Supporting methods

int	cmp_data	void *data, GEN x, GEN y
GEN	module_intersect	GEN A, GEN B

GEN	module_intersect_tc	GEN A, GEN B
GEN	prime_ksearch	GEN relations, GEN extra
GEN	prime_ksearch_tc	GEN relations, GEN extra
GEN	QM_hnf	GEN M
GEN	QM_hnf_tc	GEN M
int	Q_issquareall	GEN x, GEN *sqrtx
GEN	powerset	GEN L
GEN	powerset_tc	GEN L
GEN	vecratio	GEN v1, GEN v2
GEN	vecratio_tc	GEN v1, GEN v2

9.6 qq-quat_int

9.6.1 Intersection number based on roots

GEN	qa_inum_roots	GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, int data, long prec
GEN	qa_inum_roots_tc	GEN Q, GEN order, GEN e1, GEN e2, int data, long prec

9.6.2 Intersection number based on x-linking

GEN	qa_inum_x	GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, int data, long prec
GEN	qa_inum_x_tc	GEN Q, GEN order, GEN e1, GEN e2, int data, long prec
GEN	qa_xlink	GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, GEN x, long prec
GEN	qa_xlink_tc	GEN Q, GEN order, GEN e1, GEN e2, GEN x, long prec
GEN	qa_xposs	GEN pset, GEN Psetprod, GEN D1, GEN D2, GEN xmin, GEN xmax
GEN	qa_xposs_tc	GEN Qorpsset, GEN D1, GEN D2, GEN xmin, GEN xmax

9.6.3 Intersection number based on fundamental domain

GEN	qa_inum_fd_givengeod	GEN Q, GEN order, GEN U, GEN geod1, GEN geod2, GEN pell1, GEN pell2, GEN D1D2, int data, GEN tol, long prec
GEN	qa_inum_fd_tc	GEN Q, GEN order, GEN U, GEN e1, GEN e2, int data, long prec
int	sides_int_indices	long i1, long i2, long j1, long j2

9.6.4 Intersection series

GEN	qa_inumseries	GEN Q, GEN order, GEN U, GEN e1, GEN e2, GEN D1, GEN D2, GEN N, int type, long prec
-----	---------------	--

GEN	qa_inumseries_tc	GEN Q, GEN order, GEN U, GEN e1, GEN e2, GEN N, int type, long prec
-----	------------------	---

9.6.5 Intersection data

GEN	qa_intlevel	GEN Q, GEN order, GEN e1, GEN e2, GEN D1D2, long prec
GEN	qa_intlevel_tc	GEN Q, GEN order, GEN e1, GEN e2, GEN D1, GEN D2, long prec
GEN	qa_inum_fromdata	GEN data, int type

9.7 qq_visual

9.7.1 Histograms

void	hist_autocompile	GEN minx, GEN maxx, char *imagenname, char *autofile, char *plotoptions, int open
void	hist_compile	char *imagenname, char *autoname, int open
GEN	hist_make	GEN data, char *imagenname, char *autofile, int compileneu, char *plotoptions, int open, long prec
GEN	hist_tobins	GEN data, GEN minx, GEN maxx, GEN nbins, int toscale, int compileneu, char *imagenname, char *autofile, char *plotoptions, int open, long prec
GEN	hist_tobins_defaultbins	GEN data, GEN minx, GEN maxx, int toscale, int compileneu, char *imagenname, char *autofile, char *plotoptions, int open, long prec
GEN	hist_rebin	GEN data, GEN histdata, GEN nbins, long prec
void	hist_recompile	GEN histdata
GEN	hist_rerange	GEN data, GEN histdata, GEN minx, GEN maxx, long prec
GEN	hist_rescale	GEN data, GEN histdata, int scale, long prec

References

- [FP85] U. Fincke and M. Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Math. Comp.*, 44(170):463–471, 1985.
- [Pag15] Aurel Page. Computing arithmetic Kleinian groups. *Math. Comp.*, 84(295):2361–2390, 2015.
- [Ric20] James Rickards. *Q Quadratic: GP guide*, 2020. available from <https://github.com/JamesRickards-Canada/Q-Quadratic>.

- [Ric21] James Rickards. *Intersections of closed geodesics on Shimura curves*. PhD thesis, McGill University, Spring 2021.
- [The21] The PARI Group, Univ. Bordeaux. *PARI/GP version 2.13.1*, 2021. available from <http://pari.math.u-bordeaux.fr/>.
- [Voi09] John Voight. Computing fundamental domains for Fuchsian groups. *J. Théor. Nombres Bordeaux*, 21(2):469–491, 2009.