$\mathbb{Q}$ Quadratic: GP 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

Contents

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.

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 equvalence 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 `alginit`). 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

This library will only work on a Linux-based system, so if you are using Windows, then you can use Linux Subsystem for Windows. Furthermore, it will only work with a PARI version that is identical to the one used to compile it. If you are using an older version of PARI/GP (or I fail to update it when a new version is released), then you will need to recompile the source code to make it work.

The only essential files are **libqquadratic.so**, **qquadratic.gp**, and **fdviewer.py**. Move them to the same folder, and call "gp qquadratic" to install the methods! If you are looking for "on the go" help with methods, addhelp files have been created for all GP-accessible methods. Call "?base", "?bqf", etc. ("?" followed by the part after the underscore of each source file) to access the list of sub-topics, and "?method" to get a description of the method "method".

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 do not use the automatic parts of the methods.

1.3 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 2-3, and clicking on the name there will take you back to Section 9.

In each method, optional arguments are given inside curly braces, and the default value is given (for example, `{flag=1}` means `flag` is optional and is defaulted to `1`).

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/GP does not support adding/dividing infinities by finite numbers. The functions here are wrappers around addition and division to allow for this.

Name:	`addoo`
Input:	`a, b`
Input format:	`a, b` complex numbers or infinity
Output format:	Complex number or ±`oo`
Description:	Returns `a+b`, where the output is `a` if `a` is infinite, `b` if `b` is infinite, and `a+b` otherwise.

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

2.2 Linear algebra

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

Name:	`lin_intsolve`
Input:	`A, B, n`
Input format:	Integers `A, B, C`
Output format:	0 or $[[m_x, m_y], [x_0, y_0]]$.
Description:	Solves $Ax+By = n$ using `gbezout`, where the general solution is $x = x_0+m_xt$ and $y = y_0 + m_xt$ for $t \in \mathbb{Z}$. If there are no solutions or `A=B=0`, returns 0.

Name:	`mat3_complete`
Input:	`A, B, 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`.

2.3 Short vectors in lattices

We follow the Fincke-Pohst method ([FP85]) for finding short vectors in a lattice.

Name:	`lat_smallvectors`
Input:	`A, C1, {C2=0}, {onesign=1}, {isintegral=0}`
Input format:	Symmetric positive definite matrix `A`, non-negative real numbers `C1, C2`, `onesign` and `integral` are `0` or `1`
Output format:	Vector of `[x,` x^TAx`]`
Description:	Finds all non-zero small vectors of the lattice specified by `A`, i.e. all `x` for which $C_1 \leq x^TAx \leq C_2$ (if `C2=0`, we instead search for $x^TAx \leq C_1$). If `onesign=1`, only output one of `x, -x` for each solution `x`. If the norms are always integral, (entries of A are half integers and integers on the diagonal), then pass `isintegral=1` and the values of the output are fixed to being exact integers.

Name:	`mat_choleskydecomp`
Input:	`A, {rcoefs=1}`
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 $x^T Ax$ is expressible as $\sum_{i=1}^{n} b_i i(x_i + \sum_{j=i+1}^{n} b_i jxj)^2$.

2.4 Time

Name:	`printtime`
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 $AX^2 + BXY + CY^2$ is represented by the vector $[A, B, C]$. 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.

3.1 Discriminant methods

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

Name:	`disclist`
Input:	`D1, D2, {fund=0}, {cop=0}`
Input format:	Integers `D1`, `D2`, `fund=0, 1`, `cop` an integer
Output format:	Vector
Description:	Returns the set of discriminants (non-square integers equivalent to `0, 1` modulo 4) between `D1` and `D2` inclusive. If `fund=1`, only returns fundamental discriminants, and if `cop`$\neq$`0`, only returns discriminants coprime to `cop`.

Name:	`discprimeindex`
Input:	`D`
Input format:	Discriminant `D`
Output format:	Vector
Description:	Returns the set of primes p for which D/p^2 is a discriminant.

Name:	`isdisc`
Input:	`D`
Input format:	-
Output format:	`0` or `1`
Description:	Returns `1` if `D` is a discriminant and `0` else.

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

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

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

Name:	bqf_disc
Input:	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:	bqf_isequiv
Input:	q1, q2, {tmat=0}
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:	Tests if q is equivalent to q2 or a BQF in q2 (when q2 is a set). If q2 is a BQF, returns 1 if equivalent and 0 if not, unless tmat=1 where we return a transition matrix taking q1 to q2. If q2 is a set of BQFs, if tmat=0 returns an index i for which q1 is equivalent to q2[i], and 0 if no such index exists. If tmat=1, instead returns [i, M] where M is the transition matrix taking q1 to q2[i].

Name:	bqf_isreduced
Input:	q
Input format:	q a PBQF
Output format:	0, 1
Description:	Returns 1 if q is reduced, and 0 is q 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:	bqf_random
Input:	maxc, {type=0}, {primitive=1}
Input format:	maxc a positive integer, type, primitive=0, 1
Output format:	BQF
Description:	Returns a random BQF of non-square discriminant with coefficient size at most maxc. If type=-1 it will be positive definite, type=1 indefinite, and type=0 either type. If primitive=1 the form will be primitive, otherwise it need not be.

Name:	bqf_random_D
Input:	maxc, D
Input format:	maxc a positive integer, D a discriminant
Output format:	BQF
Description:	Returns a random primitive BQF of discriminant D (positive definite if $D < 0$).

Name:	`bqf_red`
Input:	`q, {tmat=0}`
Input format:	`q` a PBQF, `tmat=0,1`
Output format:	BQF or `[q', M]`
Description:	Returns the reduction of `q`. If `tmat=0` this is a BQF, otherwise this is `[q', M]` where the reduction is `q'` and the transition matrix is `M`.

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

Name:	`bqf_trans_coprime`
Input:	`q, n`
Input format:	BQF `q`, non-zero integer `n`
Output format:	BQF
Description:	Returns a BQF equivalent to `q` whose first coefficient is coprime to `n`.

Name:	`ideal_tobqf`
Input:	`numf, 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 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:	`ibqf_isrecip`
Input:	`q`
Input format:	IBQF `q`
Output format:	`0, 1`
Description:	Returns `1` if `q` is reciprocal (`q` is similar to `-q`), and `0` else.

Name:	`ibqf_leftnbr`
Input:	`q, {tmat=0}`
Input format:	IBQF `q=[A, B, C]` with $AC < 0$, `tmat=0, 1`
Output format:	IBQF or `[q', M]`
Description:	Returns the left neighbour of `q`, i.e. the nearest reduced form on the river to the left of `q`. If `tmat=0` only returns the IBQF, and if `tmat=1` returns the form and transition matrix.

Name:	`ibqf_redorbit`
Input:	`q, {tmat=0}, {posonly=0}`
Input format:	IBQF `q`, `tmat, posonly=0, 1`
Output format:	Vector
Description:	Returns the reduced orbit of `q`. If `tmat=1` each entry is the pair `[q', M]` of form and transition matrix, otherwise each entry is just the form. If `posonly=1`, we only take the reduced forms with positive first coefficient (half of the total), otherwise we take all reduced forms.

Name:	`ibqf_rightnbr`
Input:	`q, {tmat=0}`
Input format:	IBQF `q=[A, B, C]` with $AC < 0$, `tmat=0, 1`
Output format:	IBQF or `[q', M]`
Description:	Returns the right neighbour of `q`, i.e. the nearest reduced form on the river to the right of `q`. If `tmat=0` only returns the IBQF, and if `tmat=1` returns the form and transition matrix.

Name:	`ibqf_river`
Input:	`q`
Input format:	IBQF `q`
Output format:	Vector
Description:	Returns the river sequence associated to `q`. The entry 1 indicates going right, and 0 indicates going left along the river.

Name:	`ibqf_riverforms`
Input:	`q`
Input format:	IBQF `q`
Output format:	Vector
Description:	Returns the forms on the river of `q` in the order they appear, where we only take the forms with first coefficient positive.

Name:	`ibqf_symmetricarc`
Input:	`q`
Input format:	IBQF `q`
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 $\mathrm{PSL}(2,\mathbb{Z})\backslash\mathbb{H}$ (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:	`mat_toibqf`
Input:	`M`
Input format:	$M \in \mathrm{SL}(2,\mathbb{Z})$
Output format:	PBQF
Description:	Returns the PBQF corresponding to the equation `M(x)=x`. Typically used when `M` has determinant `1` and is hyperbolic, so that the output is a PIBQF (this method is inverse to `bqf_automorph` in this case).

3.4 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:	`bqf_comp`
Input:	`q1, q2, {tored=1}`
Input format:	PBQFs `q1, q2` of the same discriminant, `tored=0, 1`
Output format:	PBQF
Description:	Returns the composition of `q1` and `q2`, where we reduce it if `tored=1`.

Name:	`bqf_ncgp`
Input:	`D`
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 $d_1 \mid d_2 \mid \cdots \mid d_k$ 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:	`bqf_ncgp_lexic`
Input:	`D`
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`: starting with the identity element, and the component with the highest order moves first.

Name:	`bqf_pow`
Input:	`q, n, {tored=1}`
Input format:	PBQF `q`, integer `n`, `tored=0, 1`
Output format:	PBQF
Description:	Returns a form equivalent to q^n, reduced if `tored=1`.

Name:	`bqf_square`
Input:	`q, {tored=1}`
Input format:	PBQF `q`, `tored=0, 1`
Output format:	PBQF
Description:	Returns q^2, reduced if `tored=1`.

3.5 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 0, and otherwise it will return a vector `v`, where

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

The types are are

-1=all, 0=finite, 1=positive, 2=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]	$[x_i, y_i]$	$\left(\begin{smallmatrix} X \\ Y \end{smallmatrix}\right) = M^j \left(\begin{smallmatrix} x_i \\ y_i \end{smallmatrix}\right)$ 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] $M \in \mathrm{SL}(2, \mathbb{Z})$

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 0, and otherwise it will return a vector `v`, where

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

The types are are

-2=quadratic, -1=all, 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 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] $M \in \mathrm{SL}(2, \mathbb{Z})$ 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 $\mathrm{lin} = [U, V, W]$. If there are no solutions the method will return 0, and otherwise it will return a vector `v`, where

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

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

[a] In fact, $i = 1$ necessarily (there is one plane only).
[b] $M \in \mathrm{SL}(3, \mathbb{Z})$ and s_1, s_2, s_3 are *rational*; they need not be integral. Same for a_i, b_i, c_i.

3.6 Representation of integers by forms - methods

Name:	`bqf_bigreps`
Input:	`q, n`
Input format:	`q=[A, B, C, D, E]` integral vector, `n` integer
Output format:	0 or `v=[[type, data], sol1, ...]`
Description:	Solves $AX^2 + BXY + CY^2 + DX + EY = n$, and returns ALL solutions. If no solutions returns 0; otherwise `v[1][1]` gives the format of the general solution in Table 2.

Name:	`bqf_linearsolve`
Input:	`q, n1, lin, n2`
Input format:	`q=[A, B, C, D, E, F]` integer vector, `n1` an integer, `lin=[U, V, W]` integer vector, `n2` an integer
Output format:	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 0; otherwise `v[1][1]` gives the format of the general solution in Table 3.

Name:	`bqf_reps`
Input:	`q, n, {proper=0}, {half=1}`
Input format:	`q=[A, B, C]` integer vector, `n` integer, `proper=0, 1`, `half=0, 1`
Output format:	0 or `v=[[type, data], sol1, ...]`
Description:	Solves $AX^2+BXY+CY^2=n$, and returns ALL solutions. If no solutions returns 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.

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:	`bqf_bdelta`
Input:	`q1, q2`
Input format:	`q1` and `q2` integral BQFs
Output format:	Integer
Description:	Returns $B_\Delta(q_1,q_2)=B_1B_2-2A_1C_2-2A_2C_1$, where $q_i=[A_i,B_i,C_i]$.

Name:	`bqf_intlevel`
Input:	`q1, q2`
Input format:	`q1` and `q2` integral BQFs
Output format:	Integer
Description:	Returns the signed intersection level of `q1, q2`, i.e. if $q_i=[A_i,B_i,C_i]$, then this is $\text{sign}(-A_1B_2+A_2B_1)\cdot\gcd(-A_1B_2+A_2B_1,-2A_1C_2+2A_2C_1,-B_1C_2+B_2C_1)$.

Name:	`ibqf_intpoint`
Input:	`q1, q2, {location=0}`
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:	Outputs a point $\mathrm{PSL}(2,\mathbb{Z})$ equivalent to 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 $\mathrm{PSL}(2,\mathbb{Z})$; if the imaginary part of `location` is non-zero, then `location` is assumed to be a point on ℓ_{q_1}. We translate the intersection point to the geodesic between `location` and γ_{q_1}`(location)`. If the invariant automorph is large, then one must increase the precision to ensure accurate results.

4.2 Intersection number computation

Name:	`ibqf_int`
Input:	`q1, q2`
Input format:	`q1, q2` PIBQFs
Output format:	Integer
Description:	Returns the full intersection number of `q1, q2`.

Name:	`ibqf_intRS`
Input:	`q1, q2`
Input format:	`q1, q2` PIBQFs
Output format:	Integer
Description:	Computes the RS-intersection number of `q1, q2`

Name:	`ibqf_intforms`
Input:	`q1, q2, {data=0}, {split=0}`
Input format:	`q1, q2` PIBQFs, `data=0, 1`, `split=0, 1`
Output format:	Vector
Description:	Returns the intersecting forms of `q1, q2` of all types. If `data=1`, each entry of the output is `[`B_Δ`(f1, f2), level of int, length of river overlap, f1, f2]`; otherwise it is just the pair `[f1, f2]`. If `split=0` outputs a single vector of the return data, and if `split=1`, it splits the output into `[[RS], [RO], [LS], [LO]]` intersection.

Name:	`ibqf_intformsRS`
Input:	`q1, q2, {data=0}`
Input format:	`q1, q2` PIBQFs, `data=0, 1`
Output format:	Vector
Description:	Returns the RS intersection of `q1` and `q2` as a vector of non-simultaneously equivalent intersecting forms. If `data=1`, each output entry is instead `[`B_Δ`(f1, f2), level of int, length of river overlap, f1, f2]`.

Name:	`ibqf_intformsRO`
Input:	`q1, q2, {data=0}`
Input format:	`q1, q2` PIBQFs, `data=0, 1`
Output format:	Vector
Description:	Returns the RO intersection of `q1` and `q2` as a vector of non-simultaneously equivalent intersecting forms. If `data=1`, each output entry is instead `[`B_Δ`(f1, f2), level of int, length of river overlap, f1, f2]`.

Name:	`ibqf_intformsLS`
Input:	`q1, q2, {data=0}`
Input format:	`q1, q2` PIBQFs, `data=0, 1`
Output format:	Vector
Description:	Returns the LS intersection of `q1` and `q2` as a vector of non-simultaneously equivalent intersecting forms. If `data=1`, each output entry is instead `[`B_Δ`(f1, f2), level of int, length of river overlap, f1, f2]`.

Name:	`ibqf_intformsLO`
Input:	`q1, q2, {data=0}`
Input format:	`q1, q2` PIBQFs, `data=0, 1`
Output format:	Vector
Description:	Returns the LO intersection of `q1` and `q2` as a vector of non-simultaneously equivalent intersecting forms. If `data=1`, each output entry is instead `[`B_Δ`(f1, f2), level of int, length of river overlap, f1, f2]`.

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=[slope, intercept, 1]
```

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

- Line segment:

```
l=[slope, intercept, startpt, endpt, 0, ooendptor, 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=[centre, 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 form 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.

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 one quarter the precision. Of course, this can eventually cause unequal points to be declared as equal. If this ends up being an issue, increase the precision.

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:	`arc_init`
Input:	`c, p1, p2, {dir=0}`
Input format:	Circle `c`, points `p1, p2` on `c`, `dir=-1, 0, 1`
Output format:	Circle arc
Description:	Initializes the circle arc on the counterclockwise segment going from `p1` to `p2` on `c`, oriented counterclockwise if `dir=1`, clockwise if `dir=-1`, and unoriented of `dir=0`.

Name:	`arc_midpoint`
Input:	`c, p1, p2`
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:	`circle_angle`
Input:	`c1, c2, p`
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` counterclockwise to the tangent to `c2` at `p`.

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

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

Name:	`circle_tangentslope`
Input:	`c, p`
Input format:	Circle/arc `c`, pop
Output format:	$\mathbb{R} \cup \infty$
Description:	Returns the slope of the tangent line to `c` at `p`.

Name:	`crossratio`
Input:	`a, b, c, d`
Input format:	`a`, `b`, `c`, `d` complex numbers or infinity, with at most one being infinity
Output format:	Complex number or ±`oo`
Description:	Returns the crossratio `[a,b;c,d]`.

Name:	`line_angle`
Input:	`l1, l2`
Input format:	Lines/segments `l1, l2`
Output format:	angle in `[0, Π)`
Description:	Returns the angle formed by rotating `l1` counterclockwise to be parallel to `l2`.

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

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

Name:	`mat_eval`
Input:	`M, x`
Input format:	`M` a 2x2 matrix and `x` a complex number or infinity
Output format:	Complex number or ±`oo`
Description:	Returns `M` acting on `x` via Mobius transformation.

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

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

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

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

Name:	`slope`
Input:	`p1, 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 fould 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:	`arc_int`
Input:	`c1, c2`
Input format:	Arcs `c1, c2`
Output format:	Vector
Description:	Returns the intersection points of `c1, c2`.

Name:	`arcseg_int`
Input:	`c, l`
Input format:	Arc `c`, segment `l`
Output format:	Vector
Description:	Returns the intersection points of `c, l`.

Name:	`circle_int`
Input:	`c1, c2`
Input format:	Circles `c1, c2`
Output format:	Vector
Description:	Returns the intersection points of `c1, c2`.

Name:	`circleline_int`
Input:	`c, l`
Input format:	Circle `c`, line `l`
Output format:	Vector
Description:	Returns the intersection points of `c, l`.

Name:	`genseg_int`
Input:	`s1, s2`
Input format:	Circle/arc/line/segment `s1, s2`
Output format:	Vector
Description:	Returns the intersection points of `s1, s2`.

Name:	`line_int`
Input:	`l1, l2`
Input format:	Lines `l1, l2`
Output format:	Vector
Description:	Returns the intersection points of `l1, l2`.

Name:	`onarc`
Input:	`c, p`
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:	`onseg`
Input:	`l, p`
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:	`seg_int`
Input:	`l1, l2`
Input format:	Segments `l1, l2`
Output format:	Vector
Description:	Returns the intersection points of `l1, l2`.

5.3 Hyperbolic distance and area

Name:	`hdist`
Input:	`z1, z2`
Input format:	Upper half plane complex points `z1, z2`
Output format:	Distance
Description:	Returns the upper half plane hyperbolic distance between `z1` and `z2`.

Name:	`hdist_ud`
Input:	`z1, z2`
Input format:	Unit disc points `z1, z2`
Output format:	Distance
Description:	Returns the hyperbolic distance between `z1, z2` in the unit disc model.

Name:	`hpolygon_area`
Input:	`circles, vertices`
Input format:	Vectors of circles `circles`, vector of vertices `vertices`
Output format:	Given a hyperbolic polygon in the unit circle model, with side `i` given by `circles[i]` and the intersection of `circles[i], circles[i+1]` being `vertices[i]`, this returns the area of the polygon. If there are edges on the unit circle (corresponding to `circles[i]=0`), the output is ∞.
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 Visualization of fundamental domains

The Python script “fdviewer.py” provides a way to plot the fundamental domains and geodesics. You must have a version of Python 3 with matplotlib ([Hun07]) in order to use this.

Name:	`python_printarcs`
Input:	`arcs, filename, {view=0}, {extrainput=NULL}`
Input format:	A set of arcs `arcs`, string `filename`, `view=0, 1`, `extrainput=NULL` or a string
Output format:	
Description:	Prints the arcs in `arcs` to the file “fdoms/filename.dat”. The file name CANNOT start with the letter “f”. If you are running Linux Subsystem for Windows, you can pass `view=1` to automatically call the plot viewer. This will also pass on the extra information from `extrainput`, which are the filenames of other arcs/fundamental domains (must all be .dat files and in the fdoms folder).

Name:	`python_plotviewer`
Input:	`S`
Input format:	String S
Output format:	
Description:	Only use this with Linux Subsystem for Windows. This calls "fdviewer.py S", hence `S` should be stored as a list of filenames separated by spaces (stored in the sub-folder fdoms).

Name:	`python_printfdom`
Input:	`U, filename`
Input format:	Fundamental domain U, string filename
Output format:	
Description:	Prints the fundamental domain `U` to the file "fdoms/filename.dat", ready for the plot viewer. The file name MUST start with "f".

5.6 Helper methods

These are various supporting methods.

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

Name:	`shiftangle`
Input:	`ang, bot`
Input format:	Real numbers `ang, bot`
Output format:	`[bot, bot+`2π`)]`
Description:	Shifts the angle `ang` by integer multiples of 2π until it lies in the range `[bot, bot+`2π`)]`.

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

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

  where B is ramified at $p_1, p_2, \ldots, p_{2r}$ 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 $\phi : \mathcal{O}_D : O$. 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 $\mathrm{Cl}^+(D)$ 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 `Q` to denote a quaternion algebra, `ord` to denote a quaternion order, and `order` to denote an initialized quaternion order.

6.1 Basic operations on elements in quaternion algebras

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

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

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

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

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

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

Name:	qa_mulvecindices
Input:	Q, L, indices
Input format:	QA Q, vector of Qelts L, vector/vecsmall indices
Output format:	Qelt
Description:	Returns the product L[indices[1]]·L[indices[2]]···L[indices[n]].

Name:	qa_norm
Input:	Q, x
Input format:	QA Q, Qelt x
Output format:	t_INT
Description:	Returns the reduced norm of x.

Name:	qa_pow
Input:	Q, x, n
Input format:	QA Q, Qelt x, integer n
Output format:	qelt
Description:	Returns x^n.

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

Name:	qa_square
Input:	Q, x
Input format:	QA Q, Qelt x
Output format:	Qelt
Description:	Returns x^2.

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

6.2 Basic operations on orders and lattices in quaternion algebras

Name:	qa_isinorder
Input:	Q, ord, x
Input format:	QA Q, QO ord, Qelt x
Output format:	0 or 1
Description:	Checks if x is in ord, and returns 1 if so.

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

Name:	qa_leftorder
Input:	Q, L
Input format:	QA Q, QL L
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:	qa_rightorder
Input:	Q, L
Input format:	QA Q, QL L
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:	qa_ord_conj
Input:	Q, ord, c
Input format:	QA Q, (i)QO ord, invertible Qelt c
Output format:	QO
Description:	Returns the order c·ord·c^-1.

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

Name:	qa_superorders
Input:	Q, ord, n
Input format:	QA Q, (i)QO ord, integer n
Output format:	Vector of QOs
Description:	Returns all quaternion orders O containing ord such that the quotient has size n.

6.3 Initialization methods

Name:	qa_eichlerorder
Input:	Q, l, {maxord=0}
Input format:	QA Q, positive integer l, (i)QO maxord or maxord=0
Output format:	iEQO
Description:	Returns an initialized Eichler order of level l, which is contained inside maxord if maxord is non-zero.

Name:	qa_maximalorder
Input:	Q, {baseord=0}
Input format:	QA Q, (i)QO baseord or baseord=0
Output format:	iQO
Description:	Returns a maximal order of Q, which contains baseord if baseord is non-zero.

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

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

Name:	`qa_init_primes`
Input:	`pset`
Input format:	Vector of primes `pset`
Output format:	QA
Description:	Returns the quaternion algebra ramified at primes of `pset`. The prime ∞ will be automatically added if the list has odd length and it is not already present (in which case an error will be raised).

Name:	`qa_init_2primes`
Input:	`p, q`
Input format:	Distinct primes `p, q`
Output format:	[IQA, iQO]
Description:	Returns the quaternion algebra ramified at `p, q` and a maximal order.

Name:	`qa_ram_fromab`
Input:	`a, b`
Input format:	Integers `a, b`
Output format:	Vector
Description:	Returns the sorted set of primes ramifying in the quaternion algebra `(a, b/`$\mathbb{Q}$`)`.

6.4 Conjugation of elements in a given order

Name:	`qa_conjbasis`
Input:	`Q, ord, e1, e2, {orient=0}`
Input format:	QA `Q`, (i)QO `ord`, 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`$\overline{\texttt{v1}}$`=A+Be2` with `B`>`0`.

Name:	`qa_conjqf`
Input:	`Q, ord, e1, e2`
Input format:	QA `Q`, (i)QO `ord`, 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`$\cdot$`v1+Y`$\cdot$`v2)`.

Name:	qa_conjnorm
Input:	`Q, ord, e1, e2, n, {retconelt=0}`
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 if there is an invertible element `x`∈`ord` with nrd`(x)=n` and `x·e1·x`$^{-1}$`=e2`, and returns the determination. If `retconelt=1`, returns the element.

Name:	qa_simulconj
Input:	`Q, ord, e1, e2, f1, f2`
Input format:	`QA` `Q`, (i)`QO` `ord`, 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.

6.5 Embedding quadratic orders into Eichler orders

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

Name:	qa_embed
Input:	`Q, order, D, {nembeds=0}, {rpell=0}`
Input format:	`QA` `Q`, `iQO` `order`, discriminant `D`, integer `nembeds`, `rpell=0, 1`
Output format:	Vector
Description:	Finds and returns `nembeds` non-equivalent optimal embeddings of the order of discriminant `D` into `order`. If `nembeds=0`, this sets `nembeds` to the total number of non-equivalent optimal embeddings into `order`. Will return the images of $(p_D+\sqrt{D})/2$ if `rpell=0`, and will return the images of the fundamental units otherwise. This method does not check that it is possible to find `nembeds` non-equivalent embeddings, so if you cannot, it will never end (and eventually the memory will run out).

Name:	`qa_embeddablediscs`
Input:	`Q, order, d1, d2, {fund=0}, {cop=0}`
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`≤`D`≤`d2` for which there exists optimal embeddings of `D` into `order`. If `fund=1`, only returns fundamental discriminants. If `cop`≠`0`, only returns discriminants coprime to `cop`.

Name:	`qa_numemb`
Input:	`Q, order, D, {narclno=0}`
Input format:	`IQA` `Q`, `iEQO` `order`, discriminant `D`, nonnegative 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-zeroness, 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. If you don't know the narrow class number, pass it in as `0`, and it will be automatically set.

Name:	`qa_ordiffer`
Input:	`Q, order, e1, e2, {D=0}`
Input format:	`IQA` `Q`, `iEQO` `order`, `Qembs` `e1, e2` of 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 (`D` is automatically set if passed as `0`).

Name:	`qa_orinfinite`
Input:	`Q, emb, {D=0}`
Input format:	`IQA` `Q`, `Qemb` `emb`, discriminant `D`
Output format:	`-1, 1`
Description:	Returns the orientation of `emb` at ∞ (`D` is automatically set if passed as `0`).

Name:	`qa_sortedembed`
Input:	`Q, order, D, {rpell=0}, {ncgp=0}`
Input format:	`IQA` `Q`, `iEQO` `order`, discriminant `D`, `rpell=0, 1`, `ncgp=0` or `bqf_ncgp_lexic(D, prec)`
Output format:	`0` or matrix
Description:	Computes all optimal embeddings of `D` into `order`, and returns the sorted output. The output is `N x 2` matrix, with the entries in the second column being `h`$^+$`(D)` optimal embeddings, sorted according to the order of the forms in `ncgp`. 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 `order` also divide `D`). If `rpell=1`, returns the images of the fundamental unit. If `ncgp=0`, this method will compute it.

6.6 Elements of norm n in an Eichler order

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

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

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:	`qa_orbitreps`
Input:	`Q, order, n`
Input format:	`IQA` `Q`, `iEQO` `order`, positive integer `n`
Output format:	Vector
Description:	Returns representatives for $\Theta(n)$ when n is coprime to the level of `order`.

Name:	`qa_orbitrepsrange`
Input:	`Q, order, n`
Input format:	`IQA` `Q`, `iEQO` `order`, positive integer `n`
Output format:	Vector
Description:	Returns representatives for $\Theta(i)$ for all $i \leq n$ coprime to the level of `order`.

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

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). See the source code for descriptions of the format of normalized boundaries/fundamental domains.

Name:	`qa_fundamentaldomain`
Input:	`Q, order, {p=0}, {dispprogress=0}, {ANRdata=0}`
Input format:	`IQA` `Q`, `iEQO` `order`, upper half plane point or `0` `p`, `dispprogress=0, 1`, `ANRdata=0` or a length 5 vector
Output format:	Fundamental domain
Description:	Returns the fundamental domain associated to `order`. If `p=0`, we set `p=I/2`. 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:	`qa_isometriccircle`
Input:	`Q, x, p`
Input format:	`IQA` `Q`, `Qelt` `x` of norm `1`, upper half plane point `p`
Output format:	`[x, mat, circ]`
Description:	Finds the isometric circle `circ` of `x` with respect to `mats`. The image of `x` in $\mathrm{PSU}(1,1)$ is `mat`.

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

Name:	`qa_normalizedbasis`
Input:	`Q, G, p`
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:	`qa_normalizedboundary`
Input:	`Q, G, p`
Input format:	`IQA` `Q`, vector of `Qelts` `G` of norm 1, upper half plane point `p`
Output format:	Normalized boundary
Description:	Returns the normalized boundary of `G` with respect to `p`.

Name:	`qa_printisometriccircles`
Input:	`Q, L, p, filename, {view=0}`
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:	`qa_reduceelt`
Input:	`Q, G, x, {z=0}, {p=0}`
Input format:	`IQA` `Q`, vector of `Qelts` `G` of norm 1 OR `normalized boundary` `G`, `Qelt` `x` of norm 1, unit disc point `z`, upper half plane point `p`
Output format:	`[gammabar, delta, decomp]`
Description:	Returns the reduction of `x` with respect to `G` and `z`. In otherwords, `d(gammabar·z, 0)<=d(g·gammabar·z, 0)` for all `g` in `G`, where `gammabar=delta·x` and `decomp` is the vecsmall of indices of `G` used to produce `delta`. If `G` is a normalized boundary, this is much faster.

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

Name:	`qa_smallnorm1elts`
Input:	`Q, order, p, z, C1, {C2=0}`
Input format:	`IQA` `Q`, `iQO` `order`, upper half plane point `p`, reals `C1, C2`
Output format:	Vector of Qelts
Description:	Returns the norm 1 elements of `order` for which `C1<invrad(x)<=C2`. If `p=0`, sets `p=I/2`, and if `C2=0`, then sets `(C1, C2)=(0, C1)`.

Name:	`qa_topsu`
Input:	`Q, g, p`
Input format:	`IQA` `Q`, `Qelt` `g` of norm 1, upper half plane point `p`
Output format:	Matrix
Description:	Returns the image of `g` in $\mathrm{PSU}(1,1)$.

6.8 Supporting methods

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

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

Name:	`QM_hnf`
Input:	`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:	`powerset`
Input:	`L`
Input format:	Vector `L`
Output format:	Vector
Description:	Returns the powerset of `L`.

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

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 algberas. 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.

Name:	`qa_inum_roots`
Input:	`Q, order, e1, e2, {data=0}`
Input format:	`IQA` `Q`, `iEQO` `order`, `Qembs` `e1, e2`, `data=0, 1`
Output format:	`[pairs]` or `[[pairs], [[signed level, x]]`
Description:	Computes the intersection number of `e1, e2` via the roots method. 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. If `data=0`, returns the set of pairs giving all non-simultaneously equivalent intersections. If `data=1`, then first element of the output is the set of pairs. If the $\mathtt{i}^{\text{th}}$ pair is `x`-linked with signed level ℓ, then the $\mathtt{i}^{\text{th}}$ entry of the second element of the output is `[x,` ℓ`]`.

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:	`qa_inum_x`
Input:	`Q, order, e1, e2, {data=1}`
Input format:	`IQA` `Q`, `iEQO` `order`, `Qembs` `e1, e2`, `data=0, 1`
Output format:	`[pairs]` or `[[pairs], [[signed level, x]]`
Description:	Computes the intersection number of `e1, e2` via the `x`-linking method. 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. If `data=0`, returns the set of pairs giving all non-simultaneously equivalent intersections. If `data=1`, then first element of the output is the set of pairs. If the $\mathtt{i}^{\text{th}}$ pair is `x`-linked with signed level ℓ, then the $\mathtt{i}^{\text{th}}$ entry of the second element of the output is `[x,` ℓ`]`.

Name:	`qa_xlink`
Input:	`Q, order, e1, e2, x`
Input format:	`IQA` `Q`, `iEQO` `order`, `Qembs` `e1, e2`, 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. 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:	`qa_xposs`
Input:	`Qorpset, D1, D2, {xmin=0}, {xmax=0}`
Input format:	`Qorpset` even length vector of finite primes OR an IQA, 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 `Qorpset`/the indefinite quaternion algebra ramified at `Qorpset`. If `xmin` and `xmax` are passed as `0`, the method returns the `x`'s in the range `[0,` $\sqrt{D_1D_2}$).

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:	`qa_inum_fd_tc`
Input:	`Q, order, U, e1, e2, {data=1}`
Input format:	`IQA` `Q`, `iEQO` `order`, `fundamental domain` `U`, `Qembs` `e1, e2`, `data=0, 1`
Output format:	`[pairs]` or `[[pairs], [[signed level, x]]`
Description:	Computes the intersection number of `e1, e2` via the fundamental domain method. If `data=0`, returns the set of pairs giving all non-simultaneously equivalent intersections. If `data=1`, 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 `[x,` ℓ`]`.

7.4 Intersection series

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

Name:	`qa_inumseries`
Input:	`Q, order, U, e1, e2, N, {type=1}`
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:	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$^{\text{th}}$ 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.

7.5 Intersection data

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

Name:	`qa_intlevel`
Input:	`Q, order, e1, e2, {D1=0}, {D2=0}`
Input format:	`QA` `Q`, `iEQO` `order`, `Qembs` `e1, e2` of discriminants `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 $\frac{x^2-D_1D_2}{\ell^2}$, where ℓ is the level. This returns the pair $[\pm\ell, x]$, where the $\pm$ is the sign of the intersection (or `1` if $x^2 > D_1D_2$ and there is no intersection). `D1, D2` can be passed as `0`, and they will be automatically set.

8 qq_visual

These methods deal with the visualization of data. At the moment, they only include methods to create histograms.

8.1 Histograms

Given some data, calling `hist_make` will automatically bin the data, write a LaTeX document displaying the histogram, compile it, and (optionally) open it. The automatic opening will only work with the Linux subsystem for Windows; I don't think it will work on Linux directly. The PDF document will reside in the subfolder "/images", and the LaTeX document and all the build files will reside in the subfolder "/images/build" (which are automatically created if they do not yet exist).

The LaTeX document this program writes uses pgfplots and externalize, so that the outputted histogram can easily be inserted into other documents. It uses very basic options for labeling the axes and the figure, and for a "finished product" that is suitable for a research paper, the user will want to make adjustments. Furthermore, the automatic binning of the data may not make optimal choices. As such, there is an array of options to adjust the output:

- When calling `hist_make`, the user can specify their own LaTeX document to compile with. This document should be placed in "/images/build" to work correctly.
- When calling `hist_make`, the user can specify options to be added between "\begin{axis}" and "\end{axis}", with the rest of the document being automatically created. This allows them to tailor the look of the histogram, as well as adding a trendline, etc.
- To change the number of bins of an already created histogram, call `hist_rebin`;
- to change the range of $x-$values used for binning, call `hist_rerange`;
- to change between absolute and relative counts ($y-$axis being the absolute count, or the scaled version giving the histogram an area of 1 respectively), call `hist_rescale`;
- to recompile the pdf after making manual changes to the LaTeX document, call `hist_recompile`.

The length 8 vector returned by all methods except `hist_recompile` (which returns nothing), is used to adjust the histogram. The exact format is:

$$\begin{aligned}&[\text{minimum } x-\text{value, maximum } x-\text{value, number of bins, is scaled, image name,} \\ &\qquad \text{LaTeX file name, plot options, } \texttt{open}]\end{aligned} \tag{8.1}$$

Name:	`hist_make`
Input:	`data, imagename, autofile, {compilenew=0}, {plotoptions=NULL}, {open=0}`
Input format:	`data` a sorted vector of real numbers, `imagename` and `autofile` strings, {`compilenew=0, 1`}, {`ploptions` either a string or `NULL`}, {`open=0, 1`}
Output format:	See Equation 8.1
Description:	First, the data is binned automatically. If `compilenew=0`, the LaTeX document `autofile`.tex is compiled. Otherwise, this method writes this file before compiling it. The image is named `imagename`, and if `plotoptions` is non-`NULL`, this string is placed between "\begin{axis}" and "\end{axis}" in `autofile`.tex. Finally, if `open=1`, the pdf is opened (only works with Linux subsystem for Windows. The returned value is used to modify the histogram, e.g. changing the bins, scaling it, and changing the range.

Name:	`hist_rebin`
Input:	`data, histdata, nbins`
Input format:	`data` the sorted list of data, `histdata` the histogram data as in Equation 8.1, `nbins` positive integer
Output format:	See Equation 8.1
Description:	Remakes the histogram with the new number of bins, `nbins`.

Name:	`hist_recompile`
Input:	`histdata`
Input format:	`histdata` the histogram data as in Equation 8.1
Output format:	
Description:	Recompiles the LaTeX document; used when you modify the LaTeX document manually.

Name:	`hist_rerange`
Input:	`data, histdata, minx, maxx`
Input format:	`data` the sorted list of data, `histdata` the histogram data as in Equation 8.1, `minx` and `maxx` real numbers
Output format:	See Equation 8.1
Description:	Remakes the histogram according to the new range `[minx, maxx]`.

Name:	`hist_rescale`
Input:	`data, histdata, scale`
Input format:	`data` the sorted list of data, `histdata` the histogram data as in Equation 8.1, `scale=0, 1`
Output format:	See Equation 8.1
Description:	If `scale=1`, scales the $y-$axis so the total area is 1, and if `scale=0`, scales it so that the $y-$axis is the actual count.

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. Clicking on a method name will bring you to its full description in the previous sections.

9.1 qq_base

9.1.1 Infinity

`addoo`	`a, b`
`divoo`	`a, b`

9.1.2 Linear algebra

`lin_intsolve`	`A, B, n`

mat3_complete	A, B, C

9.1.3 Short vectors in lattices

lat_smallvectors	A, C1, {C2=0}, {onesign=1}, {isintegral=0}
mat_choleskydecomp	A, {rcoefs=1}

9.1.4 Time

printtime

9.2 qq_bqf

9.2.1 Discriminant methods

disclist	D1, D2, {fund=0}, {cop=0}
discprimeindex	D
isdisc	D
pell	D
posreg	D
quadroot	D

9.2.2 Basic methods for binary quadratic forms

bqf_automorph	q
bqf_disc	q
bqf_isequiv	q1, q2, {tmat=0}
bqf_isreduced	q
bqf_random	maxc, {type=0}, {primitive=1}
bqf_random_D	maxc, D
bqf_red	q, {tmat=0}
bqf_roots	q
bqf_trans	q, M
bqf_trans_coprime	q, n
ideal_tobqf	numf, ideal

9.2.3 Basic methods for indefinite quadratic forms

ibqf_isrecip	q
ibqf_leftnbr	q, {tmat=0}
ibqf_redorbit	q, {tmat=0}, {posonly=0}
ibqf_rightnbr	q, {tmat=0}
ibqf_river	q
ibqf_riverforms	q
ibqf_symmetricarc	q
mat_toibqf	M

9.2.4 Class group and composition of forms

bqf_comp	q1, q2, {tored=1}
bqf_ncgp	D
bqf_ncgp_lexic	D
bqf_pow	q, n, {tored=1}
bqf_square	q, {tored=1}

9.2.5 Representation of integers by forms

bqf_bigreps	q, n
bqf_linearsolve	q, n1, lin, n2
bqf_reps	q, n, {proper=0}, {half=1}

9.3 qq_bqf_int

9.3.1 Intersection Data

bqf_bdelta	q1, q2
bqf_intlevel	q1, q2
ibqf_intpoint	q1, q2, {location=0}

9.3.2 Intersection number computation

ibqf_int	q1, q2
ibqf_intRS	q1, q2
ibqf_intforms	q1, q2, {data=0}, {split=0}
ibqf_intformsRS	q1, q2, {data=0}
ibqf_intformsRO	q1, q2, {data=0}
ibqf_intformsLS	q1, q2, {data=0}
ibqf_intformsLO	q1, q2, {data=0}

9.4 qq_geometry

9.4.1 Basic line, circle, and point operations

arc_init	c, p1, p2, {dir=0}
arc_midpoint	c, p1, p2
circle_angle	c1, c2, p
circle_fromcp	cent, p
circle_fromppp	p1, p2, p3
circle_tangentslope	c, p
crossratio	a, b, c, d
line_angle	l1, l2
line_fromsp	s, p
line_frompp	p1, p2
mat_eval	M, x
midpoint	p1, p2

mobius	M, c
perpbis	p1, p2
radialangle	c, p
slope	p1, p2

9.4.2 Intersection of lines and circles

arc_int	c1, c2
arcseg_int	c, l
circle_int	c1, c2
circleline_int	c, l
genseg_int	s1, s2
line_int	l1, l2
onarc	c, p
onseg	l, p
seg_int	l1, l2

9.4.3 Hyperbolic distance and area

hdist	z1, z2
hdist_ud	z1, z2
hpolygon_area	circles, vertices

9.4.4 Visualization of fundamental domains

python_printarcs	arcs, filename, {view=0}, {extrainput=NULL}
python_plotviewer	S
python_printfdom	U, filename

9.4.5 Helper methods

atanoo	x
shiftangle	ang, bot

9.5 qq_quat

9.5.1 Basic operations on elements in quaternion algebras

qa_conj	x
qa_conjby	Q, x, y
qa_inv	Q, x
qa_m2rembed	Q, x
qa_minpoly	Q, x
qa_mul	Q, x, y
qa_mulvec	Q, L
qa_mulvecindices	Q, L, indices
qa_norm	Q, x

qa_pow	Q, x, n
qa_roots	Q, x
qa_square	Q, x
qa_trace	x

9.5.2 Basic operations on orders and lattices in quaternion algebras

qa_isinorder	Q, ord, x
qa_isorder	Q, ord
qa_leftorder	Q, L
qa_rightorder	Q, L
qa_ord_conj	Q, ord, c
qa_ord_disc	Q, ord
qa_superorders	Q, ord, n

9.5.3 Initialization methods

qa_eichlerorder	Q, l, {maxord=0}
qa_maximalorder	Q, {baseord=0}
qa_ord_init	Q, ord
qa_init_ab	a, b
qa_init_primes	pset
qa_init_2primes	p, q
qa_ram_fromab	a, b

9.5.4 Conjugation of elements in a given order

qa_conjbasis	Q, ord, e1, e2, {orient=0}
qa_conjqf	Q, ord, e1, e2
qa_conjnorm	Q, ord, e1, e2, n, {retconelt=0}
qa_simulconj	Q, ord, e1, e2, f1, f2

9.5.5 Embedding quadratic orders into Eichler orders

qa_associatedemb	Q, order, emb, {D=0}
qa_embed	Q, order, D, {nembeds=0}, {rpell=0}
qa_embeddablediscs	Q, order, d1, d2, {fund=0}, {cop=0}
qa_numemb	Q, order, D, {narclno=0}
qa_ordiffer	Q, order, e1, e2, {D=0}
qa_orinfinite	Q, emb, {D=0}
qa_sortedembed	Q, order, D, {rpell=0}, {ncgp=0}

9.5.6 Elements of norm n in an Eichler order

qa_orbitreps	Q, order, n
qa_orbitrepsrange	Q, order, n
qa_hecke	Q, order, n, emb

9.5.7 Fundamental domain methods

qa_fundamentaldomain	Q, order, {p=0}, {dispprogress=0}, {ANRdata=0}
qa_isometriccircle	Q, x, p
qa_fdarea	Q, order
qa_normalizedbasis	Q, G, p
qa_normalizedboundary	Q, G, p
qa_printisometriccircles	Q, L, p, filename, {view=0}
qa_reduceelt	Q, G, x, {z=0}, {p=0}
qa_rootgeodesic_fd	Q, U, g
qa_smallnorm1elts	Q, order, p, z, C1, {C2=0}
qa_topsu	Q, g, p

9.5.8 Supporting methods

module_intersect	A, B
prime_ksearch	relations, {extra=0}
QM_hnf	M
powerset	L
vecratio	v1, v2

9.6 qq_quat_int

9.6.1 Intersection number based on roots

qa_inum_roots	Q, order, e1, e2, {data=1}

9.6.2 Intersection number based on x-linking

qa_inum_x	Q, order, e1, e2, {data=1}
qa_xlink	Q, order, e1, e2, x
qa_xposs	Qorpset, D1, D2, {xmin=0}, {xmax=0}

9.6.3 Intersection number based on fundamental domain

qa_inum_fd_tc	Q, order, U, e1, e2, {data=1}

9.6.4 Intersection series

qa_inumseries	Q, order, U, e1, e2, N, {type=1}

9.6.5 Intersection data

qa_intlevel	Q, order, e1, e2, {D1=0}, {D2=0}

9.7 qq_visual

9.7.1 Histograms

`hist_make`	`data, imagename, autofile, {compilenew=0}, {plotoptions=NULL}, {open=0}`
`hist_rebin`	`data, histdata, nbins`
`hist_recompile`	`histdata`
`hist_rerange`	`data, histdata, minx, maxx`
`hist_rescale`	`data, histdata, scale`

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.

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[Pag15] Aurel Page. Computing arithmetic Kleinian groups. *Math. Comp.*, 84(295):2361–2390, 2015.

[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.