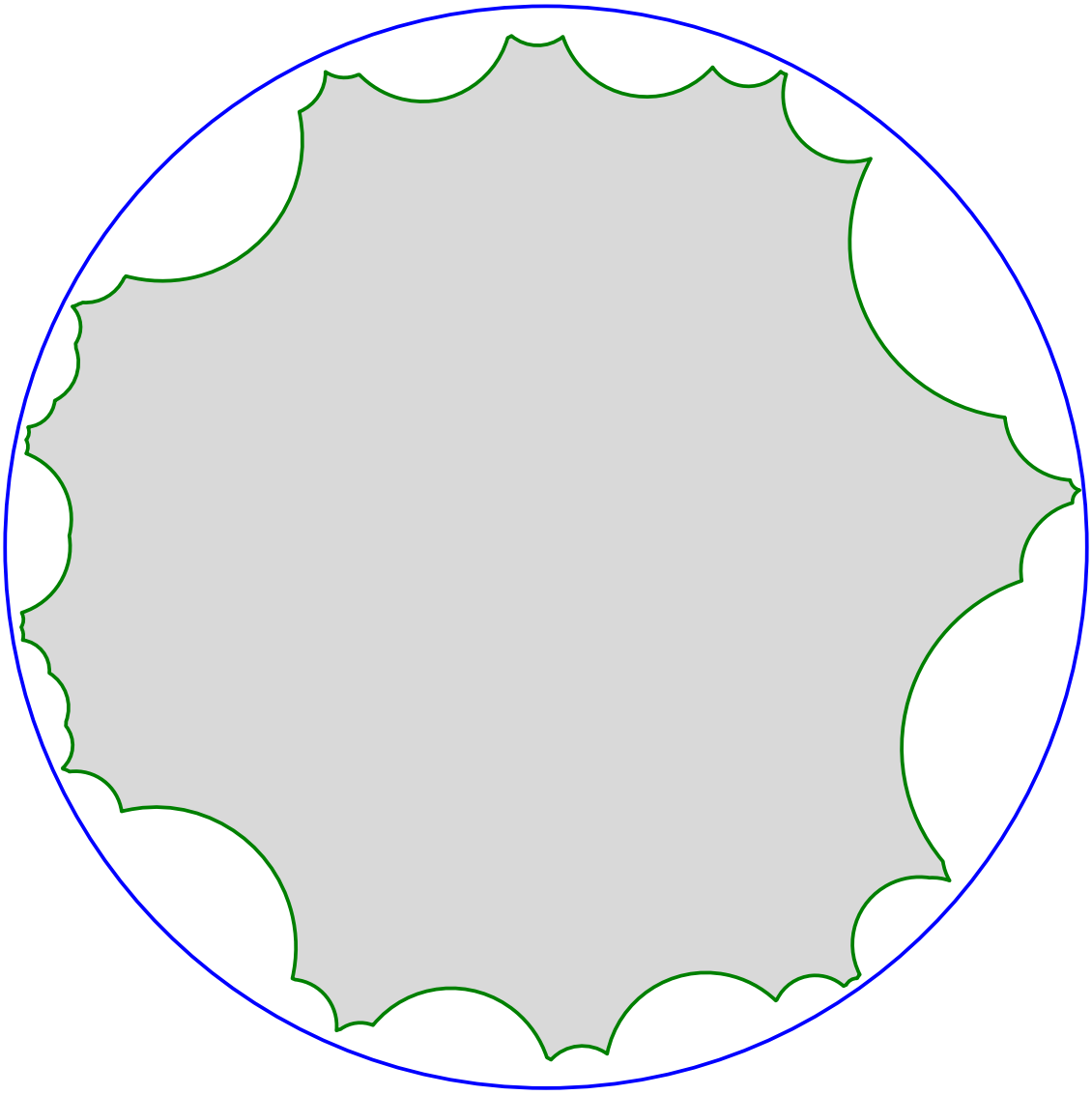
Fundamental Domains for Congruence Arithmetic Fuchsian Groups in PARI/GP - User's Manual



James Rickards
University of Colorado Boulder
E-mail: james.rickards@colorado.edu
Website GitHub

Contents

1 Introduction

Quaternion algebras over number fields are implemented as part of the algebras package in PARI/GP. This guide is intended as an introduction to how to initialize and use quaternion algebras in PARI/GP, as well as how to compute fundamental domains for arithmetic Fuchsian groups with the additional package [Ric23].

1.1 Timings

Table 1: Running times of the PARI versus the Magma implementation.

$\deg(F)$	$\mathrm{disc}(F)$	$N(\mathfrak{D})$	Area	t(MAGMA) (s)	t(PARI) (s)	$\approx \frac{\text{t(MAGMA)}}{\text{t(PARI)}}$
1	1	390	100.530	79.900	0.028	2900
1	1	2145	1005.310	$>$ 16 days	0.388	$>$3500000
1	1	2021	2023.186	$>$ 20 days	0.976	$>$1700000
1	1	5111	5051.681	$>$ 30 days	3.095	$>$830000
2	5	3724	542.867	17860.740	0.501	36000
2	44	283	2067.168	$>$ 16 days	5.230	$>$260000
3	81	1509	350.462	4910.460	0.861	5700
3	257	423	770.737	8721.000	2.486	3500
4	14656	17	469.145	7768.180	2.852	2700
4	1957	2511	1633.628	642055.790	24.637	26000
5	24217	415	343.481	17697.460	3.499	5000
5	240881	35	829.380	154800.980	12.873	12000
6	371293	131	198.968	4064.760	2.657	1500
6	300125	491	542.448	23059.450	14.103	1600
7	83934569	1	138.230	454.880	3.892	120
7	119292949	1	238.761	5418.350	7.150	760
8	309593125	19	422.230	4017.320	34.929	120
8	456768125	11	423.068	4852.140	43.681	110

The first fundamental domains for arithmetic Fuchsian groups were computed by hand for some small examples, where it was feasible to enumerate enough elements and solve the geometry. For instance, see Example G of [Kat92] or Section 6 of [Mac08].

The first result making the computation practical in a general situation is due to John Voight in [Voi09], with an accompanying package in Magma [BCP97]. Aurel Page then improved the enumeration step and generalized the geometry to work for arithmetic Kleinian groups in [Pag15], and this was again implemented in Magma. The current package is based on my paper [Ric22], which takes the general algorithm set out by Voight, improves on the geometric algorithms, takes the enumeration from Page, and fine tunes the constants to produce greatly improved running times. We take advantage of the speed of C by writing directly in C using the PARI library, [PAR24]. We also generalize the input slightly to allow for any group lying between the norm 1 and totally positive normalizer groups of an Eichler order.

In order to demonstrate the improvements, we make a table comparing the running times of John Voight's original code in Magma versus our improved code. We computed the fundamental domains for O^1 in 18 examples, where O is a maximal order in a quaternion algebra A over a totally real number field F, that is split at a unique real place. In Table 1, we note the degree and discriminant of F, the norm to $\mathbb{Q}$ of the discriminant of the algebra $\mathfrak{D}$, the hyperbolic area of the fundamental domain, and the running times. The Magma code was run once, and the PARI code was run 200 times, with the average time displayed (due to the probabilistic nature of the algorithm). All computations were run on the Euclid server at CU Boulder (with a Intel Xeon Silver 4314 CPU at 2.40GHz), running Magma version 2.21-2, and PARI/GP version 2.16.2 (development 29192-76aa21cfb). Note that the PARI/GP code also computes the signature

and presentation, and this is included in the timing.

1.2 Installation prerequisites

You need to be running PARI/GP on a Linux-based system. In particular, on Windows, you need to run it through Windows Subsystem for Linux (WSL). The Windows binaries that are offered will not be able to run this code. For more information on how to set that up, see the tutorial I wrote: https://pari.math.u-bordeaux.fr/PDF/PARIwithWindows.pdf. If you do run it through WSL, make sure you `git clone` from WSL and not Windows! Otherwise, Windows line endings (carriage returns) may be added to files, causing issues.

This package has been tested on version 2.15.2, 2.15.4, and 2.16.2, and should generally work on anything after 2.15.0. It may also work on 2.13.x, but this is not suggested, as there are various quaternion algebra bugs that are unfixed.

1.3 Installation instructions

To build this package, follow these instructions:

1. Open a terminal, git clone this repository, and navigate into the folder.
2. You need to know where the version of PARI/GP you want to use is installed, in particular, the file "pari.cfg". The default location is `/usr/local/lib/pari/pari.cfg`, but this may change based on your Linux distro, or if your copy of PARI/GP was installed with SageMath. If you think it is installed in the default location, you can simply call `make`, which will build the project.
3. Otherwise, call `make setup` to search for "pari.cfg". By default, the program searches in `/usr`, but there is a chance it is not installed there (this sometimes happens on a remote server. For example, on CoCalc, it is in `/ext/sage/VERSION/local/lib/pari/pari.cfg`). In this case, you can supply an alternate location to search.
4. If the program finds a potential match, it will ask you to confirm which file is correct, and save it to `paricfg_loc.txt`. If you installed SageMath and PARI/GP separately, or have multiple versions of PARI/GP, then you may find multiple copies in this search.
5. Once this step completes, a call to `make` will compile the project!
6. Since the location of "pari.cfg" is saved, further updates to the code (e.g. with `git pull`) can be re-built by simply calling `make`. The only time you will need to go through `make setup` again would be if you want to work with a PARI/GP installed in a different place.
7. A call to `make clean` will remove any library files created.

Now that the project is built, you can start it by calling `gp fdom` to open a gp session and load the methods. Help can be accessed in-program with `?fdom`, or by consulting this manual.

1.4 Troubleshooting

If you are having problems installing or using the package, please get in touch! Here are some common problems you may run into:

- **No "pari.cfg" file found:** no matches were found in the search location. Try asking to search in `/`, which searches everywhere. This will be slow, but is guaranteed to find the correct files, if they exist.

- **Wrong version:** Maybe you found the libraries, but there were warnings with `make`. The likely cause is the version of PARI/GP you found was too old. If there are multiple copies of PARI/GP on your computer, then perhaps you chose the wrong one! Check the shared object file created: it will be called `libfdom-X-Y-Z.so`, where `X.Y.Z` is the version of PARI/GP corresponding to "pari.cfg". If this does not match the version you are using, then you found the wrong one!
- **Miscellaneous:** when you compile with `make`, object files (.o) are created. However, if the underlying code did not change, then nothing will happen when you call `make` again. If you change the version of PARI/GP you are working with (or perhaps, trying different installations), then you should call `make clean` in between to clear out these object files. Otherwise, recompiling will do nothing!

1.5 Testing

There are two testing methods included, with one moderately faster than the other (the slower of which should take between 3 and 30 seconds, depending on your setup). Call `./testing/fdom-fast` and `./testing/fdom-slow` from the main folder to run them. If there are problems, this will be noted on the screen, and you can check out the exact issues in the files `./testing/fdom-fast.diff` and `./testing/fdom-slow.diff`.

1.6 Warnings

The current algorithms on the `klein` branch are updated versions of what was described in [Ric22]. Some of the main differences are:

- We use the Klein model instead of the unit disc, as this makes the geometry easier and more efficient. The main downside is occasionally we need to use more precision, as points are shoved closer to the boundary.
- The basic geometric algorithms are essentially unchanged, but the implementation was further optimized.
- The enumeration strategy is also mostly unchanged, but instead of enumerating small vectors with the Fincke-Pohst algorithm [FP85], we use Schnorr-Euchner linear pruning, alongside a minor implementation speedup described in Appendix B of [GNR10].
- We now support finding the fundamental domain for a general subgroup between O^1 and $N^+_{A^\times}(\mathrm{O})$.
- The naming conventions and formatting of data structures has been updated from the `paper` branch, which should not be used except to directly test details related to the paper [Ric22].

There are plans to integrate this code into the main PARI/GP program (no timeline on completion of this). In order to do so, some names and conventions will likely change. While I do not intend any more major changes to the code on GitHub, the final integration into PARI/GP could look fairly different. Therefore you should be aware that you may need to eventually update your code, or restrict it to certain pre-integration versions of PARI/GP. Since there is no timeline on this step, I would also not suggest waiting for this to happen, as delays are inevitable.

1.7 Acknowledgements

This project has been a monumental task, spanning over three years, and two complete rewrites. My hope is that it will be very useful for a variety of projects, so if you end up using this package, I would appreciate knowing how you are using it!

A list of people I wish to thank includes:

- Mike Lipnowski, who suggested that I might be able to use fundamental domains to compute intersection numbers of closed geodesics, pointing out that there was a Magma package to compute them. All of my thesis work was in PARI/GP, so I decided to have a go at porting the code over. Little did I know what it would become!
- John Voight, who helped shed a lot of light on quaternion algebra questions and shared insights from his original version of the code.
- Aurel Page, who shared insights from his package to compute fundamental domains for arithmetic Kleinian groups, as well as gave a lot of implementation related suggestions.
- Bill Allombert, who greatly improved the implementation by poring over many early versions of the code.
- The PARI/GP group in general, who have been invaluable at the online/in person ateliers and over the e-mail threads.

2 Quaternion Algebras in PARI/GP

2.1 Initializing a quaternion algebra

Let F be a number field, and $a, b \in F^\times$. The quaternion algebra $\left(\frac{a,b}{F}\right)$ is the $F-$vector space with basis $1, i, j, k$, with multiplication defined by

$$i^2 = a, \quad j^2 = b, \quad k = ij = -ji.$$

2.1.1 Initialize by (a, b, F)

In gp, it is easy to initialize the algebra given a, b, for example:

```
F=nfinit(y^2-y+1);
A=alginit(F, [y, y-7]);
```

initializes $A = \left(\frac{y,y-7}{F}\right)$, where $F = \mathbb{Q}(y)$ with $y^2 - y + 1 = 0$.

Warnings:

- The variable you use in defining F must be *lower* in priority than the variable used to define A, which is `'x` by default (can be changed by adding a third input to `alginit`). I suggest using `'y` for F.
- The input a must NOT be a square in F. The reason for this is PARI/GP lets $L = F(\sqrt{a})$, and stores an element as $[e, f]$, representing $e + fj$ for $e, f \in L$ (see Section 2.2 for more on this). There is currently no way around this limitation, so if a happens to be a square, then you need to swap a, b or modify them suitably.
- Both a and b must not only be integral, but have integral coefficients! If you initialize A by ramification, then this extra limitation is no longer present. This will hopefully be fixed in the future.

2.1.2 Initialize by ramification

Let $\mathrm{Pl}(F)$ be the set of places of F, and for $v \in \mathrm{Pl}(F)$, we say that A is ramified at v if

$$A_v = A \otimes_F F_V$$

is a division ring. Otherwise, A is split/unramified at v.

Every quaternion algebra over F is ramified at a finite even-cardinality set of places. Furthermore, every such set of places corresponds to a unique isomorphism class of quaternion algebras over F. We can initialize

a quaternion algebra by specifying the set of ramification places, though it is a little more complicated. All complex places are split, so we only need to specify the splitting behaviour on a set of finite primes, as well as the infinite real places.

Given the number field F, the command `F.roots` gives the roots of the defining polynomial, with the r_1 real roots coming first. When you specify the infinite ramification, you give an r_1-length vector of 0's and 1's, where 1 =ramification, and the order corresponds to the ordering of `F.roots`.

For the finite ramification, you supply two vectors: a vector of prime ideals, and a same-length vector of 0's and 1's, where again, 1 =ramification. The total number of 1's across both vectors must be even. For example:

```
F=nfinit(y^5-y^4-3*y^2+1); /*3 real places, approximately -0.539, 0.564, 1.817*/
I1=idealprimedec(F, 2)[1]; /*A prime ideal lying above 2*/
I2=idealprimedec(F, 7)[1]; /*A prime ideal lying above 7*/
I3=idealprimedec(F, 11)[1]; /*A prime ideal lying above 11*/
pfin=[I1, I2, I3];
ramfin=[1, 0, 1]; /*Ramification at I1 and I3*/
raminf=[1, 1, 0]; /*Ramification at the first two infinite places*/
A=alginit(F, [2, [pfin, ramfin], raminf]); /*Initialize by ramification.*/
```

This code initializes the quaternion algebra over F ramified at `I1`, `I3`, and the first two infinite places. The ideal `I2` was of no use, and was included to demonstrate that you may include extraneous places in the initialization. The input of 2 is to specify that quaternion algebras have degree 2 (`alginit` can create more general central simple algebras).

Warnings:

- As before, the variable used in F must have lower priority than the variable in A.
- In version 2.13.3 and earlier, there was a rare bug that could occur. If F had at least 3 real places and A was unramified at all finite places, then the only choice of `raminf` that worked was `[1,1,...,1,0,0,...,0]`, i.e. all the ramified real places came first. All other infinite ramification choices (e.g. `[1, 0, 1]`, `[0, 1, 1]`, etc.) would encounter an infinite loop. This bug has been fixed in version 2.13.4 and beyond.

2.1.3 Retrieving a, b

Calling `algab(A)` retrieves the pair (a, b) that defined the quaternion algebra. Note that this is a function installed with the fundamental domain package, and is not built into PARI/GP itself.

2.2 Elements of quaternion algebras

If you have never used the algebras package before, then this is likely the most confusing part. PARI/GP uses two main representations of elements, the "algebraic" and the "basis" representations, and *neither* is the traditional $[1, i, j, k]$ basis representation! In the package [Ric23], I have included methods to translate elements to and from the traditional representation.

The nomenclature of the translation methods are:

`algalgtobasis, algbasistoalg, alg1ijktoalg, algbasisto1ijk,`

etc. (the first two methods are in PARI/GP, and the last two are in the extra package).

2.2.1 Algebraic representation

Assume that F was initialized with the variable `'y`, and $A = \left(\frac{a,b}{F}\right)$ with the default variable `'x`. The algebra stores the splitting field $L = F(\sqrt{a})$ using $x = \sqrt{a}$, and the algebraic representation of an element α is a

length 2 column vector:

$$\alpha = [u, v]\sim \text{ means } \alpha = u + jv, \text{ where } u, v \in L.$$

Note that the j is on the other side of v to the North American convention! In particular, if $u = e + fi$ ($i = \sqrt{a} = x$), and $v = g + hi$, then

$$\alpha = e + fi + j(g + hi) = e + fi + gj - hk.$$

Using one of the methods to translate an element to the algebraic representation will typically produce an element with a lot of `Mod`s, as this is how PARI/GP likes to store elements of number fields. For example,

```
F=nfinit(y^2-3);
A=alginit(F, [y, 2*y-3]);
alpha=algbasistoalg(A, [1, 1, 0, 0, 1, 0, -1, 1]~);
```

produces the element

```
[Mod(Mod(1/6*y + 3/2, y^2 - 3)*x + Mod(y + 1, y^2 - 3), x^2 + Mod(-y, y^2 - 3)), Mod(Mod
    (1/6*y + 1/2, y^2 - 3)*x + Mod(1/3*y + 2, y^2 - 3), x^2 + Mod(-y, y^2 - 3))]~.
```

Use `liftall` to eliminate all the modului, and get the much simpler looking
`[(1/6*y + 3/2)*x + (y + 1), (1/6*y + 1/2)*x + (1/3*y + 2)]~`.

2.2.2 Basis representation

A quaternion algebra A comes with a "natural order": let $L = F(\sqrt{a})$, and then $\mathcal{O}_L \oplus j\mathcal{O}_L$ is an order (as b was necessarily integral). By taking a $\mathbb{Z}-$basis of $\mathcal{O}_L$, we obtain a $\mathbb{Z}-$basis of this order. When you initialize an order, PARI/GP also computes a maximal order O_0 which contains the natural order. You can also choose to have `alginit` not compute a maximal order, in which case O_0 stores the natural order. If you don't need the maximal order and are working with an extremely large algebra, then this is a good idea.

If $n = [F : \mathbb{Q}]$, then this basis has length $4n$, and we store an element of A as a length $4n$ column vector of coefficients. The basis representation is the most common form outputted by algebra methods. You can retrieve the basis of O_0 in terms of the natural basis by calling `algbasis(A)`, where the columns are the coefficients.

2.2.3 1ijk representation

This representation is *not* built into PARI, but instead is in the Fundamental Domains package. The element $\alpha \in \left(\frac{a,b}{F}\right)$ is represented as a 4-dimensional vector `[e, f, g, h]`, where $e, f, g, h \in F$ and

$$\alpha = e + fi + gj + hk.$$

This representation is only there to help input data from a problem or output data into a more palatable format. You should only use this representation at the start or end of a computation, as the PARI/GP library does not handle it.

2.2.4 Basic operations on elements

The normal `+,*,/,^` symbols do not work on elements of quaternion algebras. Instead, you should use

```
algadd(A, elt1, elt2);
algsub(A, elt1, elt2);
algneg(A, elt1); /*Only useful for the algebraic representation, as -elt1 works
    otherwise.*/
algmul(A, elt1, elt2);
algsqr(A, elt1);
```

```
algpow(A, elt1, -10);
alginv(A, elt1);
algdivr(A, elt1, elt2); /*Returns elt1*elt2^(-1).*/
algdivl(A, elt1, elt2); /*Returns elt1^(-1)*elt2.*/
algnorm(A, elt1); /*Reduced norm of elt1, an element of F.*/
algnorm(A, elt1, 1); /*Absolute norm of elt1, an element of Q.*/
algtrace(A, elt1); /*Reduced trace of elt1, an element of F.*/
algtrace(A, elt1, 1); /*Absolute trace of elt1, an element of Q.*/
```

The elements can be in either the algebraic or basis representations (mixed is fine). The output will be in basis form, unless all input elements were in algebraic form.

3 Fundamental domain background

For a longer text, see the books by Beardon [Bea95], Katok [Kat92], or Section IV of Voight [Voi21]. See also the papers by Voight [Voi09] or myself [Ric22].

3.1 Setup

Let F be a totally real number field, and let $A = \left(\frac{a,b}{F}\right)$ be split at a *unique* real place. If $\sigma_1, \sigma_2, \ldots, \sigma_n$ are the embeddings of $F \to \mathbb{R}$, then this is equivalent to

$$\sigma_i(a) < 0 \text{ and } \sigma_i(b) < 0 \text{ for exactly } n-1 \text{ choices of } i.$$

Let O be an order in A, and define

$$N_{A^\times}(\mathrm{O}) := \{x \in A^\times : x\mathrm{O}x^{-1} = \mathrm{O}\}$$

to be the normalizer of O (see Section 3.3 and Section 28.9 of [Voi21] for more on this group). Define

$$\mathrm{O}^1 = \{x \in \mathrm{O} : \mathrm{nrd}(x) = 1\}$$

to be the set of elements of norm 1. Let $\tilde{\Gamma}$ be any group for which

$$\mathrm{O}^1 \leq \tilde{\Gamma} \leq N^+_{A^\times}(\mathrm{O}),$$

where the $+$ indicates that we only keep the elements x with $\mathrm{nrd}(x)$ being totally positive.

Given any element $x \in \tilde{\Gamma}$, the unique split real place maps x to an element of $\mathrm{Mat}(2,\mathbb{R})$ with positive determinant, which can be uniquely scaled to live in $\mathrm{SL}(2,\mathbb{R})$. By quotienting by $\{\pm 1\}$, the image of $\tilde{\Gamma}$ in $\mathrm{PSL}(2,\mathbb{R})$ is denoted by Γ.

This group Γ is a discrete subgroup of $\mathrm{PSL}(2,\mathbb{R})$, and is called a (congruence) arithmetic Fuchsian group. The package can compute a fundamental (Dirichlet) domain for Γ when O is an Eichler order (it can work in some instances for non-Eichler orders, but this behaviour is not guaranteed).

3.2 Dirichlet domains

The group $\mathrm{PSL}(2,\mathbb{R})$ acts on the upper half plane $\mathbb{H}$ by Möbius maps. Let $p \in \mathbb{H}$ have trivial stabilizer under the action of a discrete subgroup Γ, and define

$$D(p) := \{z \in \mathbb{H} : d(z,p) \leq d(gz,p) \text{ for all } g \in \Gamma\},$$

which forms a fundamental domain for $\Gamma\backslash\mathbb{H}$, and is known as a Dirichlet domain. It is a connected region whose boundary is a closed hyperbolic polygon with finitely many sides. Each side is a subset of the *isometric circle* $I(g)$ of an element $g \in \Gamma$, defined by the following equation:

$$I(g) := \{z \in \mathbb{H} : d(p,z) = d(p,gz)\},$$

where $d(,)$ denotes hyperbolic distance. When we say "the set of elements forming the boundary of a Dirichlet domain", we are referring to the set of elements whose isometric circles form the boundary.

A Dirichlet domain also has a notion of a side pairing: we say that sides S_1 and S_2 are paired if there exists a $g \in \Gamma$ such that $gS_1 = S_2$. Since $gI(g) = I(g^{-1})$, it will follow that paired sides will correspond to $I(g)$ and $I(g^{-1})$, although this is not sufficient. The element g will pair up their isometric circles, but we need to pair up the actual side, which is a finite length subset. In particular the vertices need to be paired with each other. In a Dirichlet domain, all sides will be paired.

Remark 3.1. In literature, a side paired with itself is often split into two sides by the midpoint, which are then paired with each other. Computationally we store this side as being paired with itself, but if you are looking at results involving counting the number of sides of the fundamental domain, then you need to account for this splitting.

3.3 Structure of the normalizer group

The main reference is Proposition 28.9.17 of [Voi21], which gives the split exact sequence:

$$1 \to (\mathrm{Cl}(\mathcal{O}_F)[2]_{\uparrow \mathrm{O}} \to N_{A^\times}(\mathrm{O})/(F^\times \mathrm{O}^\times) \to \mathrm{AL}(\mathrm{O}) \to 1$$

While we will not describe all terms in this sequence, we will talk about $\mathrm{AL}(\mathrm{O})$ briefly. If O is an Eichler order with $\mathrm{discrd}(\mathrm{O}) = \mathfrak{N}$), then $\mathrm{AL}(\mathrm{O})$ can be naturally identified with a subgroup of

$$\prod_{\mathfrak{p} | \mathfrak{N}} \mathbb{Z}/2\mathbb{Z}.$$

If Ω is the set of ramified places of A and $\mathrm{Cl}_\Omega(\mathcal{O}_F)^2$ is trivial, then this is an isomorphism.

We will focus on 4 main groups $\tilde{\Gamma}$ for which $\mathrm{O}^1 \leq \tilde{\Gamma} \leq N^+_{A^\times}(\mathrm{O})$:

- O^1, the norm 1 group.
- $U(\mathrm{O})^+$, the elements of O with totally positive unit norm.
- $\mathrm{AL}(\mathrm{O})^+$, the elements of $\mathrm{AL}(\mathrm{O})$ with totally positive norm.
- $N^+_{A^\times}(\mathrm{O})$, the full totally positive norm normalizer group.

Our strategy hinges on *always* computing the fundamental domain for O^1, and then adding additional elements to the generating set. In general, the quotient $N^+_{A^\times}(\mathrm{O})/(F^\times \mathrm{O}^1)$ is an abelian 2-group, $\tilde{\Gamma}/(F^\times \mathrm{O}^1)$ is also an abelian 2-group. If it is generated by the elements $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathrm{O}$, let $n_i = \mathrm{nrd}(\alpha_i)$ for $1 \leq i \leq k$ be the norms of the elements. If we can find elements $\alpha_i' \in N^+_{A^\times}(\mathrm{O})$ with norm n_i, then Γ will be generated by O^1 and the set of α_i'.

4 Computing fundamental domains in PARI/GP

When you compute a fundamental domain, the signature, presentation, and elliptic elements are all computed and stored at the same time.

4.1 Finding fields and algebras

To begin, we need to specify a totally real number field F. These can be found in a few places, including:

- On the LMFDB [LMF23], where you can search for number fields with a variety of properties.
- On John Voight's website, where tables of fields with small discriminant are listed;

When you initialize this in PARI/GP (with the command `nfinit(...)`, where ellipsis denotes a monic irreducible polynomial), don't forget that you **cannot** use the variable `x` if you want to work with an algebra.

If you have a specific pair (a, b) in mind or know the ramification that you want, then you can now initialize a quaternion algebra as in Sections 2.1.1 and 2.1.2. Alternatively, you can search for Fuchsian groups over F whose area lies in a given range with the function `afuchlist`, for example:

```
afuchlist(F, 100); /*Area up to 100, split at the first oo place*/
afuchlist(F, [50, 80]); /*Area between 50 and 80, split at the first oo place*/
afuchlist(F, 30, 3); /*Area up to 30, split at the third oo place*/
```

Each return element is a triple `[[a, b], area, ram]`, where `ram` is the multiset of finite primes lying above the finite ramification of `A=alginit(F, [a, b])`.

4.2 Eichler orders

While Eichler orders are not yet supported in PARI/GP, Aurel Page provided a short gp script to create them, which is stored in "eichler.gp". If `I` is an ideal (in any format) of `F`, then an Eichler order with this level is created with

```
Or = algeichlerorder(A, I);
```

If you want to save a certain order between sessions (not necessarily Eichler), then storing the matrix representation is not a good idea. The maximal order computed with `alginit(...)` is probabilistic and can change depending on the random seed, meaning this ourder would change.

One solution is to use `algordertoalgorder` and `algorderalgtoorder` to covert between storing a basis as a vector of elements in the algebraic representation, and in this matrix format.

4.3 Specifying Gamma: generic case

The default case is computing a fundamental domain for O^1, which can be accomplished by

```
X=afuchinit(A); /*For the stored maximal order*/
X=afuchinit(A, Or); /*For an order, assumed to be Eichler.*/
```

If you want to compute one of the other three main groups as specified in Section 3.3, this can be accomplished by setting the *type* via

```
X=afuchinit(A, , 1); /*For U(maximal order)*/
X=afuchinit(A, Or, 2); /*For AL(Or)*/
X=afuchinit(A, , 3); /*For N_{A^x}(maximal order)^+*/
```

There is one final flag, which you can set to 0 to not initialize the fundamental domain and related data (default is 1, which computes the fundamental domain, signature, presentation, and elliptic elements):

```
X=afuchinit(A, , , 0); /*Create the Fuchsian group, do not compute the fundamental
     domain, signature, etc.*/
X=afuchinit(A, , 1, 1); /*Identical to X=afuchinit(A, , 1)*/
```

If you initialize `X` with `flag=0`, then you can add the fundamental domain and related data with `X=afuchmakefdom(X)`.

Remark 4.1. Working with type 2 or 3 requires computing one element of each of the target norms w. If $\mathrm{Nm}_{F/\mathbb{Q}}(w)$ is rather large, finding such an element may take awhile, possibly even longer than the computation of the original fundamental domain!

4.4 Specifying Gamma: general subgroup

If you want something a little finer than the 4 generic fundamental domain groups, you can specify any group $\mathrm{O}^1 \leq \tilde{\Gamma} \leq N_{A^\times}^+(\mathrm{O})$. To begin, you must initialize `X` with type 3 and compute a fundamental domain. You should then view the possible norms of elements:

```
X=afuchinit(A, Or, 3);
norms=afuchnormalizernorms(X);
```

The vector `norms` has length 3, with

- `norms[1]` storing totally positive units;
- `norms[2]` storing totally positive norms from AL(O);
- `norms[3]` storing totally positive norms coming from the rest of $N_{A^\times}^+(\mathrm{O})$.

If the total number of such norms is n, then

$$\frac{N_{A^\times}^+(\mathrm{O})}{F^\times \mathrm{O}^1} \simeq \left(\frac{\mathbb{Z}}{2\mathbb{Z}}\right)^n.$$

The group $\tilde{\Gamma}$ can be represented by a generating set, where each element is a column vector of 0's and 1's representing a product from the concatenation of the components of `norms`. By storing these generators in a matrix `M` (whose columns form the generating set), we can call

```
X1=afuchnewtype(X, M)
```

to compute the new fundamental domain! This should be very quick now, as all of the required elements have been found, it is a matter of running the normalized basis algorithm once.

4.5 Other options

4.5.1 Viewing progress

If you want to see the progress of a computation, you can call `setdebug(alg, 1)` to have the program display partial results as we go. This will also display if more precision was required and the algebra was recomputed.

4.5.2 Changing p

Since we work in the Klein model, a choice of centre p was required to map the upper half plane to the disc. The point p must have trivial stabilizer in Γ, and the default choice is $p = \pi/8 + i/2$. If, for whatever reason, you wish to change p, you can do so by calling `afuchchangep(X, p)`. This will update the structure `X`, and recompute the stored fundamental domain and presentation if they were already initialized.

4.5.3 More precision

You can increase the precision that `X` is computed with by calling `X=afuchmoreprec(X, 3)`, which increments the precision by 3 steps.

5 Computations using the fundamental domain

Once you have a fundamental domain, you can compute many things, including the signature, a group presentation with a minimal set of generators, closed geodesics, or a visualization of it.

For the rest of the section, we assume that we have an arithmetic Fuchsian group initialized with `X=afuchinit(...)`.

5.1 Basic data

You should typically not access the components of `X` directly. Instead, use some of the following commands:

- `afuchalg(X)` returns the stored algebra. This can be useful as sometimes it was automatically recomputed with more precision.
- `afucharea(X)` returns the computed area of the fundamental domain, which may differ slightly to the theoretical computation due to floating point rounding errors.
- `afuchelliptic(X)` returns the vector of elliptic elements, whose orders are respectively listed in the signature.
- `afuchelts(X)` returns the elements corresponding to the sides of the fundamental domain. They generate the group Γ, but are not a minimal set of generators.
- `afuchnormalizernorms(X)`: returns a generating set of norms of elements in the totally positive normalizer. If we initialized `X` with type 3, this simply recalls the info. Else, it computes it.
- `afuchorder(X)` returns the stored order. If you take the order to be the stored maximal order, this will just be the identity matrix.
- `afuchsides(X)` returns the sides of the fundamental domain. The format of a side is $[a, b, r]$, where the side has equation $ax + by = 1$ in the Klein model, and $(x - a)^2 + (y - b)^2 = r^2 = a^2 + b^2 - 1$ in the unit disc model.
- `afuchspair(X)` returns the side pairing of the fundamental domain. It is a Vecsmall S, where $S[i] = j$ means side i is paired with side j (and vice versa).
- `afuchvertices(X)` returns the vertices of the fundamental domain, where the $i - 1$ and i^{th} entriex correspond to the i^{th} side. This defaults to being in the Klein model, but calling `afuchvertices(X, 1)` converts them to the unit disc model.

5.2 Signature

Call `afuchsignature(X)` to retrieve the signature. The format is $[g, V, s]$, where g is the genus, $V = [m_1, m_2, \ldots, m_t]$ are the orders (≥ 2) of the elliptic cycles, and s is the number of parabolic cycles. In particular, there exists a group presentation of Γ_{O} in the following format:

- The group is generated by $a_1, a_2, \ldots, a_g, b_1, b_2, \ldots, b_g, g_1, g_2, \ldots, g_{t+s}$;
- The g_i satisfy the relations $g_i^{m_i} = 1$ for $1 \leq i \leq t$;
- We have the relation
$$[a_1, b_1][a_2, b_2] \cdots [a_g, b_g] g_1 g_2 \cdots g_{t+s} = 1,$$
where $[x, y] = xyx^{-1}y^{-1}$ is the commutator.

This is a minimal presentation if $t + s = 0$. If $t + s > 0$, then removing g_{t+s} makes it a minimal presentation.

5.3 Presentation

Call `P=afuchpresentation(X)` to retrieve the presentation. The output is a length 2 vector, where:

- $P[1]$ is the list of elements generating Γ_{O}, given in basis representation;
- $P[2]$ is the vector of relations;

Let $P[1] = \{g_1, g_2, \ldots, g_k\}$, and then the relation $[a_1, a_2, \ldots, a_i]$ means

$$1 = g_{a_1}^{\mathrm{sign}(a_1)} g_{a_2}^{\mathrm{sign}(a_2)} \cdots g_{a_i}^{\mathrm{sign}(a_i)}.$$

For example, the relation $[1, 4, -5, -5, 3]$ corresponds to

$$1 = g_1 g_4 g_5^{-2} g_3.$$

Each element of $P[2]$ is a Vecsmall. Note that this presentation is currently *not* in the format of Section 5.2. This functionality may eventually be added.

5.4 Elements as words in the presentation

Given an element $g \in \tilde{\Gamma}$ and a presentation P, you can compute g as a word in P with the command `w=afuchword(X, g)`. The output is a Vecsmall such that

$$g = P[1][w[1]] \cdot P[1][w[2]] \cdots P[1][w[k]] \text{ in } \Gamma.$$

In particular, if you carry out this multiplication, your result might not equal g! However, their quotient will be in $F^\times$. If you are working in O^1, then it will be $\pm g$.

5.5 Closed geodesics

Given a primitive hyperbolic element $g \in \tilde{\Gamma}$, there is a corresponding closed geodesic on $\Gamma_{\mathrm{O}} \backslash \mathbb{H}$. We can compute this geodesic with `afuchgeodesic(X, g)`. The return is a vector of vectors of the form

$$[g, s1, s2, v1, v2, [a, b, c]],$$

representing the segments of the geodesic inside the fundamental domain in order. Specifically, we have

- g is the element whose geodesic gives the current segment;
- the segment runs from vertex $v1$ on side $s1$ to vertex $v2$ on side $s2$;
- the equation of the segment is $ax + by = c$, where $c = 0$ or $c = 1$.

6 Visualization

6.1 LaTeX

Call `afuchfdom_latex(U, filename)` to print the domain to the file "plots/build/filename.tex". In order to compile the file, you need the document class standalone, as well as the pgf package.

The command `afuchfdom_latex` includes 4 further options (in order):

- `model`, which is 1 by default. 0 uses the Klein model, 1 uses the unit disc model, and 2 will eventually use the upper half plane model.
- `boundcircle`, which is 1 by default. Changing it to 0 will not print the bounding circle.
- `compile`, which is 1 by default. If you are working with Windows Subsystem for Linux (WSL), this will compile the LaTeX document, and move the pdf up one folder to "plots/filename.pdf". If you do not use WSL or have pdflatex installed, set this to 0.
- `open`, which is 1 by default. If we compiled the picture, this also opens it. If you do not use WSL, set this to 0.

6.2 Python

We require the packages matplotlib and numpy to run "fdomviewer.py", which contains the code to visualize a fundamental domain and geodesics. To write a fundamental domain to a Python-readable file, call `afuchfdom_python(X, filename)`, which prints the requisite data to "fdoms/filename.dat".

You can also print geodesics to a file using `afuchgeodesic_python(X, g, "geodesic1")`. You can pass either `g` or the output of `afuchgeodesic(X, g)` to this function.

To open the application, from the terminal call `py fdomviewer.py filenames`, where you must list the fundamental domain before any geodesics that appear (multiple geodesics can be put on the same picture, though it is suggested to put no more than 5). If you are working in WSL, you can also call `fdomviewer(filenames)`.

In addition to the normal matplotlib commands, you can:

- click on a side to highlight it in red, as well as the paired side in blue. You can use the right/left arrow keys to change sides (as well as clicking on a new side);
- click on a geodesic to highlight it, highlight the next side in orange, and put an arrow in the middle specifying the orientation. You can use the up/down arrow keys to move between consecutive segments of the geodesic;
- Press "t" to hide the text box with information;
- Press "m" to hide the bounding axes;
- Press "c" to hide the bounding circle.
- Press "a" to animate the drawing of the currently selected geodesic.

7 Further development

There are many possibilities for further development using the code (Hilbert modular forms, cohomology, etc.). If you are interested writing code for such applications, do get in touch! I have tried to make an effort to make it accessible with this user's manual and extensive comments in the C code, but I am sure that the documentation is not complete.

References

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[Bea95] Alan F. Beardon. *The geometry of discrete groups*, volume 91 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1995. Corrected reprint of the 1983 original.

[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.

[GNR10] Nicolas Gama, Phong Q. Nguyen, and Oded Regev. Lattice enumeration using extreme pruning. In *Advances in cryptology—EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Comput. Sci.*, pages 257–278. Springer, Berlin, 2010.

[Kat92] Svetlana Katok. *Fuchsian groups*. Chicago Lectures in Mathematics. University of Chicago Press, Chicago, IL, 1992.

[LMF23] The LMFDB Collaboration. The L-functions and modular forms database. http://www.lmfdb.org, 2023. [Online].

[Mac08] Melissa L. Macasieb. Derived arithmetic Fuchsian groups of genus two. *Experiment. Math.*, 17(3):347–369, 2008.

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

[PAR24] The PARI Group, Univ. Bordeaux. *PARI/GP version* `2.16.2`, 2024. available from https://pari.math.u-bordeaux.fr/.

[Ric22] James Rickards. Improved computation of fundamental domains for arithmetic Fuchsian groups. *Math. Comp.*, 91(338):2929–2954, 2022.

[Ric23] James Rickards. Fundamental domains for Shimura curves. https://github.com/JamesRickards-Canada/Fundamental-Domains-for-Shimura-curves, 2023.

[Voi09] John Voight. Computing fundamental domains for Fuchsian groups. *J. Théor. Nombres Bordeaux*, 21(2):469–491, 2009.

[Voi21] John Voight. *Quaternion algebras*, volume 288 of *Graduate Texts in Mathematics*. Springer, Cham, [2021] ©2021.