3-MANIFOLDS, TRIANGULATIONS, AND VOLUME COMPUTATIONS PROJECT DESCRIPTION

CHRISTIAN ZICKERT

1. In brief

Goals: Below is a summary of our goals.

- Understand the basics of triangulations, and how they can be used to compute invariants such as the set of representation volumes.
- Get acquainted with various mathematics software to be used throughout the project.
- Write new software tools to investigate a conjecture stating that all representation volumes are linear combinations of hyperbolic volumes.
- Make all software, databases, and findings of our investigation publically available.

Software: We will use the software listed below. You are strongly encouraged (expected!) to install the software and play around with it before we begin.

SnapPy: Software for computing invariants of hyperbolic manifolds. It includes a census of triangulated hyperbolic manifolds that can be created from 8 simplices or less, and also a census of link complents. Available at http://www.math.uic.edu/t3m/SnapPy/. Check out the video tutorials.

Snap: Software for number theoretic invariants of hyperbolic manifolds. Available at http://unhyperbolic.org/snap/.

Pari/GP: General software for number theory, available at http://pari.math.u-bordeaux.fr/.

Sage: A unified interface between various math software. Both Pari/GP and SnapPy can be used with Sage. Available at http://www.sagemath.org/

Theory: Much of the theory can be found in the references below. We only have 8 weeks, so don't expect to understand all of the theory. To do research one often does not need to know all aspects of the theory, only the part that is relevant for the problem at hand!

[1]: Basic theory of hyperbolic manifolds. Chapter E.5-ii, deals with Thurston's gluing equations.

[6]: A very readable description of how to triangulate a link complement.

[4]: Chapter 0 gives a brief summary of the number theory needed. We will also need some results from Chapter 3 about number fields associated to hyperbolic manifolds.

[2]: A simple introduction to Ptolemy coordinates with many examples.

2. Background, examples, and plan of action

By the end of the 8 weeks, the goal is to have built a comprehensive database of shape fields, and of examples of Neumann's conjecture. All results will be made publically available at http://ptolemy.unhyperbolic.org/, and you will become participants of CURVE, http://curve.unhyperbolic.org/index.html.

It is only very recently that the proper software tools for studying representation volumes have been developed. You will thus be doing cutting edge research. The better prepared you are, the more we will achieve, so I strongly recommend that you start working before we officially begin. The programming exercises can be done without fully understanding the theory. The most important ones are Exercises 2.34 and 2.44. See Section 2.6 for general SnapPy tips.

2.1. **Basics of triangulations.** Every 3-manifold is homeomorphic to one obtained by gluing together simplices. A 3-manifold can thus be encoded by specifying the faces that are glued and the permutation of the gluing (see Figure 1). The manifold pictured is homeomorphic to the complement of the figure 8 knot, and in general, one has a simple algorithm for triangulating a link complement [6]. The software Plink, which is part of SnapPy, allows you to draw a link and create the corresponding triangulation file. A triangulation file looks like in http://regina.sourceforge.net/docs/foreign.html#import-snappea.

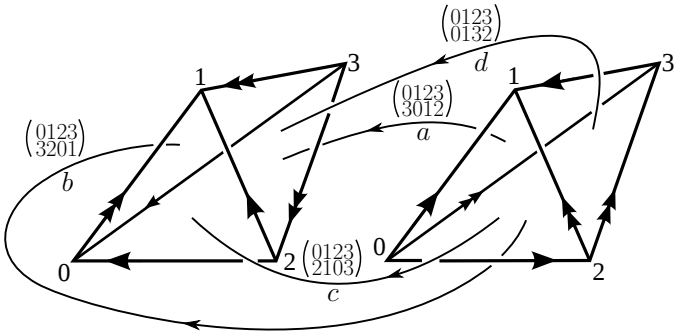

Figure 1. Triangulation of the census manfold m004.

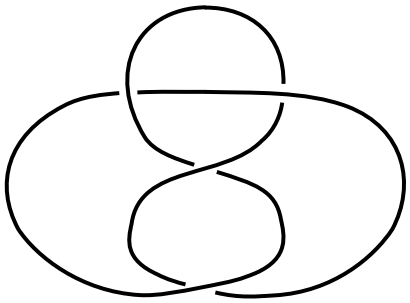

Figure 2. The figure 8 knot.

2.1.1. *Censuses.* SnapPy has several different censuses of 3-manifolds (see http://www.math.uic.edu/t3m/SnapPy/censuses.html). We shall mainly use the following:

- `OrientableCuspedCensus`: Hyperbolic manifolds with 2-5, 6, 7, and 8 simplices; typical names: m049, s509, v2079, t12313.
- `LinkExteriors`: Knots with at most 11 crossings and links with at most 10 crossings; typical names: 5_3 and 10^2_71.
- `HTLinkExteriors`: Knots and links with up to 14 crossings; typical names: K10a122, K11n54, L13a2110, L14n71 (K for knot, L for link, a for alternating, n for non-alternating).

Example 2.1. Here is how to load a census manifold in SnapPy, and to save the triangulation to a file.

```
SnapPy
>>> M=Manifold("m019")  \\ Loads the census manifold m019 and calls it M.
>>> M.save('/some_directory/m019.trig') \\ saves the triangulation to a file.
```

Exercise 2.2. Create triangulation files for each manifold of each census, and save them in appropriate directories (write a script, don't do it one manifold at a time).

2.2. **Gluing equations and volume.** The gluing equations are polynomial equations designed to construct a hyperbolic structure on a triangulated 3-manifold M. The idea is that a hyperbolic structure on a simplex is determined by a single complex number, and for the hyperbolics structures to glue together to a hyperbolic structure on M, the complex numbers must satisfy a system of polynomial equations. The gluing equations are desribed in [1, E.5-ii]. They have the form

$$\prod_{i=1}^{n} z_i^{A_{ji}}(1-z_i)^{B_{ji}} = \varepsilon_j, \tag{2.1}$$

where the z_i's are variables (called *shape coordinates*, *shapes*, or *cross-ratios*), n is the number of simplices, ε_j is a sign, and $[A|B]$ is a matrix with $2n$ columns.

Example 2.3.

```
SnapPy
>>> M=Manifold("m004") # Loads the manifold m004
>>> M.gluing_equations() # Displays the gluing equation matrix.
[([2, -1], [-1, 2], 1),
 ([-2, 1], [1, -2], 1),
 ([1, 0], [0, 1], 1),
 ([0, -2], [0, 4], 1)]
```

This means that the gluing equations are:

$$\begin{aligned} z_1^2(1-z_1)^{-1}z_2^{-1}(1-z_2)^2 &= 1 \\ z_1^{-2}(1-z_1)^{1}z_2^{1}(1-z_2)^{-2} &= 1 \\ z_1^2(1-z_1)^{0}z_2^{0}(1-z_2)^{1} &= 1 \\ z_1^0(1-z_1)^{0}z_2^{-2}(1-z_2)^{4} &= 1. \end{aligned} \tag{2.2}$$

Remark 2.4. Each solution to the gluing equations gives rise to a homomorphism (called a representation)

$$\pi_1(M) \to \mathrm{PSL}(2,\mathbb{C}), \tag{2.3}$$

where $\pi_1(M)$ is the fundamental group of M.

Definition 2.5. A solution to the gluing equations where all shapes z_i are positive is called a *geometric solution*.

Theorem 2.6. *A geometric solution determines a (complete) hyperbolic structure on M. Furthermore, we have*

$$\mathrm{Vol}(M) = \sum_{i=1}^{n} D(z_i), \tag{2.4}$$

where D is the function defined by

$$D(z) = \mathrm{Im}(\mathrm{Li}_2(z)) + \arg(1-z)\log|z|, \qquad \mathrm{Li}_2(z) = \int_0^1 \frac{\log(1-tz)}{t}dt. \tag{2.5}$$

□

Remark 2.7. If a geometric solution exists, it is unique. This is a consequence of a result known as *Mostow rigidity*, which states that (complete) hyperbolic structures are unique in dimension 3 and higher.

Example 2.8.

```
SnapPy
>>> M=Manifold('m023');
>>> M.volume()  # Computes the volume of M.
2.9441064867
```

Exercise 2.9. Create a file containing all manifolds in `OrientableCuspedCensus` and their volumes.

Exercise 2.10. Which link (in `LinkExteriors` and `HTLinkExteriors`) has the largest volume?

The definition below is motivated by (2.4).

Definition 2.11. For a solution $z_1, \dots, z_n$ to the gluing equations (corresponding to a representation ρ, c.f. Remark 2.4) one can define the volume (of ρ) as $\sum_{i=1}^{n} D(z_i)$.

Example 2.12. One easily checks that the gluing equations (2.2) are equivalent to the equations

$$z_1 = z_2, \qquad z_1^2 - z_1 + 1 = 0. \tag{2.6}$$

We thus have two solutions

$$z_1 = z_2 = \frac{1+\sqrt{-3}}{2}, \qquad z_1 = z_2 = \frac{1-\sqrt{-3}}{2}, \tag{2.7}$$

The first solution is geometric, so by by Theorem 2.6, we have

$$\mathrm{Vol}(m004) = 2D\left(\frac{1+\sqrt{-3}}{2}\right) = 2.02988321282... \tag{2.8}$$

Similarly, the other solution has volume $2D(\frac{1-\sqrt{-3}}{2}) = -2.02988321282...$

Example 2.13. Let's compute the shapes for the geometric solution to the gluing equations of the knot 7_2 (see http://katlas.math.toronto.edu/wiki/7_2 for a picture.)

```
SnapPy
>>> M=Manifold('7_2')
>>> M.tetrahedra_shapes(part='rect')
[0.97968392714 + 0.59056955984*I,  # value of z_1
 0.25132270106 + 0.45131497073*I,  # value of z_2
 0.0581813774 + 1.6912791495*I,    # value of z_3
 1.1636911715 + 0.5641856323*I]    # value of z_4
>>> M.volume()
3.3317442316
```

You should check that the gluing equations really are satisfied. You can double check that the volume is correct e.g. using Pari/GP as follows:

```
Pari/GP
>>> D(z)=imag(dilog(z))+arg(1-z)*log(abs(z));  # Defines the function D
>>> v_1=D(0.97968392714 + 0.59056955984*I);
>>> v_2=D(0.25132270106 + 0.45131497073*I);
>>> v_3=D(0.0581813774 + 1.6912791495*I);
>>> v_4=D(1.1636911715 + 0.5641856323*I);
>>> v_1+v_2+v_3+v_4
3.3317442316447718071859581687299307457
```

Exercise 2.14. Use SnapPy to write down the gluing equations for the knot complement 5_2. Solve these by hand (you should get 3 solutions). Compute the volume of each of them using Pari/GP (you should get 0 and $\pm \mathrm{Vol}(5_2)$).

Remark 2.15. Solving the gluing equations is difficult. SnapPy tries to find a geometric solution numerically, and computes the volume using (2.4). It neither computes the other solutions, nor their volumes. To do this we need *Ptolemy coordinates*.

2.3. **Ptolemy coordinates.** The Ptolemy coordinates are another way of computing representations, which were invented recently by Garoufalidis-Thurston-Zickert [3]. Every solution to the gluing equations corresponds to a solution to the Ptolemy equations and vice versa. One obtains a system of equations by assigning a variable to each edge of each simplex, and an equation to each simplex. Each equation has the form

$$c_{03}c_{12} + c_{01}c_{23} = c_{02}c_{13}, \tag{2.9}$$

where the c_{ij}'s are the variables assigned to the edges (see Figure 3). We also stipulate that if two edges are identified in the manifold, the Ptolemy coordinates must be equal up to a sign depending on the vertex ordering. One can always assume that one of the Ptolemy coordinates is 1. We refer to the set of Ptolemy equations as the *Ptolemy variety*.

The Ptolemy coordinates have several advantages over the shape coordinates including

- Equations are easy to write down and are always homogeneous of degree 2.
- Solutions are easy to compute.
- Computations can be done for other solutions than the geometric one using the *Ptolemy module* http://ptolemy.unhyperbolic.org/ (part of SnapPy).
- The Ptolemy coordinates generalize to representations in $\mathrm{SL}(n, \mathbb{C})$ (the shape coordinates also generalize, but we shall not need this).

The disadvantage is that they are less geometric, but let's not care about that.

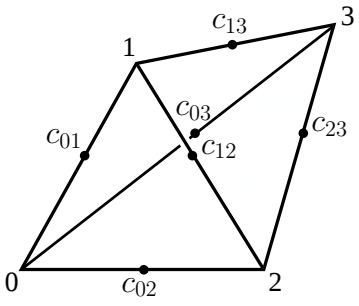

FIGURE 3. Ptolemy coordinates on a simplex.

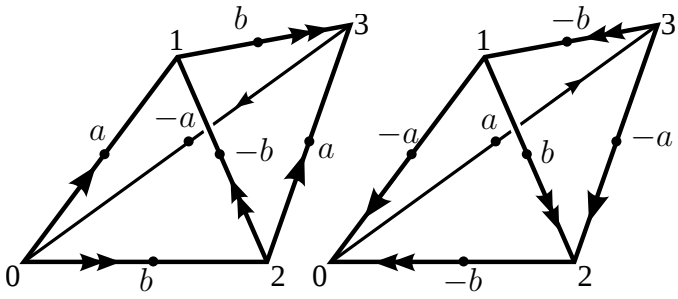

FIGURE 4. Triangulation of m003 with Ptolemy coordinates.

Example 2.16. For the triangulation in Figure 4, we have

$$ab + a^2 - b^2 = 0, \qquad ab + a^2 - b^2 = 0, \qquad b = 1, \tag{2.10}$$

which is equivalent to the system

$$b = 1, \qquad a^2 + a - 1 = 0. \tag{2.11}$$

Hence, there are two solutions, both defined over $\mathbb{Q}(\sqrt{5})$.

We refer to [2, Sec 3] for an explanation of the theory, and for more examples.

Remark 2.17. Typically (although not always), the solutions come in families (called components) such that for each component, each variable (shape or Ptolemy coordinate) has the form $q_i(x)$, where $q_i(x)$ is a polynomial, and x is a root of an irreducible polynomial p. In particular, each component determines a *number field* (see Section 2.4).

The Ptolemy variety is stratified by so-called *obstruction classes* (we will not attempt to understand these). We illustrate the Ptolemy variety, and how to obtain components of solutions, by an example.

Example 2.18.

```
SnapPy
>>> M=Manifold('m098');
>>> M.num_tetrahedra()
5 # M has 5 simplices.
>>> PtVars=M.ptolemy_variety(2,'all'); # All obstruction classes
>>> len(PtVars)  # Length of a list
2 # 2 obstruction classes.
>>> Var0=PtVars[0];Var1=PtVars[1];
>>> for e in Var0.equations:
...     print e # Prints Ptolemy equations for Obstruction class 0.
c_0011_0 * c_0011_2 - c_0011_0 * c_0101_0 + c_0101_0^2
c_0011_0 * c_0011_2 - c_0011_0 * c_0101_0 + c_0101_0^2
```

```
c_0011_0 * c_0101_3 + c_0011_2^2 - c_0101_0^2
c_0011_2^2 + c_0101_0 * c_0101_4 - c_0101_3^2
- c_0011_2 * c_0101_3 - c_0101_3^2 + c_0101_4^2
- 1 + c_0011_0
>>> for e in Var1.equations:
...     print e # Prints Ptolemy equations for Obstruction class 1.
c_0011_0 * c_0011_2 + c_0011_0 * c_0101_0 - c_0101_0^2
c_0011_0 * c_0011_2 + c_0011_0 * c_0101_0 - c_0101_0^2
c_0011_0 * c_0101_3 + c_0011_2^2 - c_0101_0^2
c_0011_2^2 + c_0101_0 * c_0101_4 - c_0101_3^2
- c_0011_2 * c_0101_3 - c_0101_3^2 + c_0101_4^2
- 1 + c_0011_0
```

If you have Sage installed, you can compute the solutions.

```
>>> Sols0=Var0.compute_solutions()
>>> len(Sols0)
2 # 2 components of solutions
>>> Sols0_0=Sols0[0];Sols0_1=Sols0[1];
>>> Sols0_0
{'c_0011_0': 1,   # value of the coordinate c_0011_0 is 1
 'c_0011_1': -1,
 'c_0011_2': -1,
 'c_0011_3': 1,
 'c_0011_4': -1,
 'c_0101_0': Mod(x, x^2 - x - 1),
 'c_0101_1': Mod(-x, x^2 - x - 1),
 'c_0101_2': Mod(-x, x^2 - x - 1),
 'c_0101_3': Mod(x, x^2 - x - 1),
 'c_0101_4': 1,
 'c_0110_0': Mod(-x, x^2 - x - 1),
 .  #  We shall not display them all
```

Note that each variable is a polynomial (here either 1, -1, x or $-x$) in x, where x is any zero of the irreducible polynomial $p(x) = x^2 - x - 1$. Since $\deg(p) = 2$, this component has two solutions. One can compute the polynomials p as follows:

```
>>> Sols0_0.number_field()
x^2 - x - 1 # 2 solutions
>>> Sols0_1.number_field()
x^6 - x^5 - 29*x^4 - 50*x^3 - 11*x^2 - 8*x - 1 # 6 solutions
>>> Sols0_1['c_0011_2']
Mod(63/2477*x^5 - 53/2477*x^4 - 2032/2477*x^3 - 3158/2477*x^2 +
3917/2477*x + 1140/2477, x^6 - x^5 - 29*x^4 - 50*x^3
- 11*x^2 - 8*x - 1) # The value of c_0011_2 in terms of a root of p.
```

You can also load the solutions from a precomputed database.

```
>>> Sols1=Var1.retrieve_solutions()
```

```
>>> len(Sols1)
1 # 1 component in obstruction class 1
>>> Sols1_0=Sols1[0]
>>> Sols1_0.number_field()
... x^8 - 3*x^7 + 4*x^6 + 5*x^5 - 16*x^4 - 4*x^3 + 10*x^2 + 6*x + 1
```

As mentioned earlier, each solution gives rise to a solution to the gluing equations. One can thus compute both shapes (cross-ratios) and volumes.

```
>>> Sols0_0.cross_ratios()
{'z_0000_0': Mod(x - 1, x^2 - x - 1), # value of shape 0 in terms of p.
 'z_0000_1': Mod(x - 1, x^2 - x - 1), # value of shape 1 in terms of p.
 'z_0000_2': Mod(x, x^2 - x - 1),
 'z_0000_3': Mod(x, x^2 - x - 1),
 'z_0000_4': Mod(-x + 2, x^2 - x - 1),
. # some more stuff that we shall not need.
     . }
>>> Sols1_0.volume_numerical()
[-2.30697456855596 E-15,
 2.07693795813790 E-16,
 2.19377776286726 E-15,
 2.67324665421574 E-14,
 1.92953212086467,
 -1.92953212086467,
 -3.50891718707666,
 3.50891718707666]
```

Often, one is primarily interested in the polynomial defining each component, and the set of volumes.

Example 2.19. This example summarizes the previous one.

```
SnapPy
>>> M=Manifold('m098');
>>> AllSols=M.ptolemy_variety(2,'all').retrieve_solutions();
>>> AllSols.number_field() # Gives a nested list
[[x^2 - x - 1, x^6 - x^5 - 29*x^4 - 50*x^3 - 11*x^2 - 8*x - 1],
 [x^8 - 3*x^7 + 4*x^6 + 5*x^5 - 16*x^4 - 4*x^3 + 10*x^2 + 6*x + 1]]
>>> AllVols=AllSols.volume_numerical(drop_negative_vols=True)
[[[5.81775636930214 E-15, 1.88266550875941 E-14],
 [-5.81964550493112 E-15,
  -1.54512291991702 E-15,
  7.59661101682800 E-15,
  4.79701633678885 E-15,
  2.99994042065713]],
 [[-2.30697456855596 E-15,
   2.07693795813790 E-16,
   2.19377776286726 E-15,
```

```
   2.67324665421574 E-14,
   1.92953212086467,
   3.50891718707666]]]
>>> AllVols.flatten(depth=2); # gives a single list
```

Remark 2.20. Real zeros of p give volume 0, and complex conjugate pairs of zeros give volumes differing by a sign.

2.3.1. *Representations in* $\mathrm{PGL}(n,\mathbb{C})$. One also has Ptolemy coordinates for representations in $\mathrm{PGL}(n,\mathbb{C})$ for $n > 2$. Note that this can be very slow.

Example 2.21.

```
SnapPy
>>> M=Manifold('m015');
>>> AllSols=M.ptolemy_variety(3,'all').retrieve_solutions().flatten(depth=2);
>>> len(AllSols)
5  # 5 components of solutions.
>>> AllSols.number_field()
[x^3 + 3*x^2 + 2*x - 1,
 x^4 + x^3 + x^2 - 24*x + 16,
 x^6 - 1/4*x^5 + 1/8*x^4 - 17/64*x^3 + 9/64*x^2 - 1/32*x + 1/64,
 x^4 + 4*x^3 + 5*x^2 + 2*x + 1,
 x^6 + 3*x^5 + 10*x^4 - 5*x^3 - 2*x^2 - x + 1]
>>> AllSols.volume_numerical(drop_negative_vols=True);
[[3.58997794036132 E-38, 11.3124883533231],
 [0.E-37, 0.E-37, -1.11671963328117 E-37, 1.11671963328117 E-37],
 [6.33266664249925,
  5.87747175411144 E-39,
  -5.87747175411144 E-39,
  6.33266664249925],
 [-2.35098870164458 E-38,
  2.35098870164458 E-38,
  4.40810381558358 E-38,
  -4.40810381558358 E-38],
 [3.17729327860032,
  -7.61132592157431 E-37,
  7.61132592157431 E-37,
  3.17729327860032]]
```

2.4. **Number fields and Neumann's conjecture.** This section contains the heart of the project.

Definition 2.22. A *number field* is a finite field extension of $\mathbb{Q}$. A *concrete number field* is a number field, which is a subfield of $\mathbb{C}$.

Every number field F is isomorphic to $\mathbb{Q}[x]/\langle p\rangle$, where p is an irreducible polynomial, and every root α of p determines an embedding of F in $\mathbb{C}$ with image equal to the concrete number field $\mathbb{Q}(\alpha)$.

Remark 2.23. A number field is not uniquely determined by p. For example, x^2-5 and x^2+x-1 both determine the number field $\mathbb{Q}(\sqrt{5})$. Given an arbitrary irreducible polynomial, PariGP can compute a "simpler" polynomial defining the same number field.

```
Pari/GP
>>> polredabs(x^4+4*x^3-2*x^2+x+1)
x^4 - x^3 - 2*x + 1 # the two polynomials define the same number field.
```

The simplified polynomial is "canonical" in the sense that p and q define isomorphic number fields if and only if the reduced polynomials are the same. It is not truely canonical, though. There are other simplification algorithms giving different polynomials, which are also "canonical" in the above sense.

Theorem 2.24. *If $z_1,\dots,z_n$ is the geometric solutions to the gluing equation, the field $\mathbb{Q}(z_1,\dots,z_n)$ is a number field.*

As the examples, 2.19 and 2.21 show, this is typically true for other solutions as well, but not always.

Example 2.25.

```
SnapPy
>>> Manifold('m135').ptolemy_variety(2,4).retrieve_solutions()
[NonZeroDimensionalComponent(dimension = 1)]
 # Obstruction class 4 has a component not defined over a number field.
```

Definition 2.26. The *concrete* number field corresponding to the geometric solution is called *the shape field*.

Example 2.27.

```
SnapPy
>>> M=Manifold('9_5');
>>> M.save('TriangulationFiles/%s.trig' % M.name()); # saves files as 9_5.trig
Snap
>>> read file TriangulationFiles/9_5.trig
>>> compute shape
Shape field: x^11-x^10+6*x^9-5*x^8+12*x^7-8*x^6+8*x^5-3*x^4+x^3+x^2+2*x-1
[1,5] -2835434699687 R(-5) = 0.239882509-1.50375813*I
  # Root of geometric solution = 0.239882509-1.50375813*I.
```

Remark 2.28. Unlike the Ptolemy module, Snap computes numerically, and approximates with an exact solution. If the shape field has high degree, one needs to set the degree and the precision. We shall only care about low degree fields. Snap internally uses the `polredabs` command (see Remark 2.23, so if two polynomials are the same, the number fields are isomorphic (although, they may differ as concrete number fields, c.f. Exercise 2.33).

Remark 2.29. Unfortunately, many functionalities of Snap are not yet part of SnapPy, and Snap does not come with a Python interface. To do scripting you thus need to open Snap as its own process, and write your own regular expressions to parse the output. I trust that you can figure this out on your own. Alternatively, one can use Sage, but this is

much slower and not recommended (see http://www.math.uic.edu/t3m/SnapPy/snap.html). Note that Sage uses a different simplification algorithm, so the polynomial given will not in general agree with the one coming from Snap.

Conjecture 2.30 (Neumann)**.** Every concrete number field which is not a subfield of $\mathbb{R}$ is the shape field of a hyperbolic manifold.

Exercise 2.31. Compute the shape fields of all census manifolds.

Exercise 2.32. Show that the only manifolds in `OrientableCuspedCensus` with shape field $\mathbb{Q}(\sqrt{-2})$ are

$$v2787, \qquad v2788, \qquad v2789.$$

In contrast, there are many manifolds with shape field $\mathbb{Q}(\sqrt{-1})$ and $\mathbb{Q}(\sqrt{-3})$.

Exercise 2.33. Find an example of 2 census manifolds, where the shape fields are equal as abstract number fields, but distinct as concrete number fields (i.e. examples where the polynomials are the same, but the roots determining the geometric solution are different).

Exercise 2.34. Make a list of number fields (sorted by degree of p) that arise as shape fields of census manifolds (ignore fields of degree larger than 8). The list should contain the set of all manifolds with a given field, and should distinguish between distinct concrete number fields. How many fields are there of degree 2, 3, 4, etc.?

Conjecture 2.35 (Neumann's conjecture)**.** The volume of a representation is an integral linear combination of volumes of hyperbolic manifolds. More generally, the volume of a representation defined over a number field F is an integral linear combination of volumes of hyperbolic manifolds with shape field contained in F.

Remark 2.36. This wild conjecture was formulated only recently, so there is currently not much literature about it (see [5] and [3]).

Example 2.37. `SnapPy`

```
>>> M=Manifold('10_155')
>>> M.ptolemy_variety(2,'all').retrieve_solutions().number_field()
[[x^4 - 5*x^3 + 3*x^2 - 3*x + 5],
 [x^4 + 3*x^3 + 2*x^2 - 3*x + 1,
  x^4 + x^3 + 2*x^2 - x + 1,
  x^4 + x^3 - 3*x^2 - 6*x - 4,
  x^4 + x^3 - x^2 - x + 1]]
>>> Sols=M.ptolemy_variety(2,'all').retrieve_solutions();
>>> Vols=Sols.volume_numerical(drop_negative_vols=True)
[[[1.32099016885775 E-15, -1.23348777488109 E-14, 2.527418477316091]],
 [[4.05976642563861, 4.05976642563862],
  [4.88975905279037 E-38,
   -4.88975905279037 E-38,
   -1.00437481596434 E-37,
   1.00437481596434 E-37],
  [-4.05078705499027 E-15,
```

```
   -4.05078705499027 E-15,
   -2.77371389268652 E-38,
    1.65452736260517 E-38],
  [1.13100876173487, 9.250541613012 09]]]
```

The 4 non-zero volumes thus come from representations defined over the fields given by

$$x^4 - 5x^3 + 3x^2 - 3x + 5, \quad x^4 + 3x^3 + 2x^2 - 3x + 1, \quad x^4 + x^3 - x^2 - x + 1.$$

To compare the number fields with the shape fields returned by Snap, we need to simplify:

```
Pari/GP
>>> polredabs(x^4 - 5*x^3 + 3*x^2 - 3*x + 5)
x^4 - x^3 + x^2 + x - 1)
>>> polredabs(x^4 + 3*x^3 + 2*x^2 - 3*x + 1)
x^4 - x^3 - x^2 - 2*x + 4
>>> polredabs(x^4 + x^3 - x^2 - x + 1)
x^4 - x^3 - x^2 + x + 1
```

Using the tools explained above, one can check (do this!) that the manifolds m155 and v1181 have shape fields defined by $p_1(x) = x^4 - x^3 + x^2 + x - 1$, and that we have

$$\mathrm{Vol}(m155) = 3.79112771597413..., \qquad \mathrm{Vol}(v1181) = 5.05483695463218...$$

It is now easy to check that

$$2.52741847731609 \approx 2\,\mathrm{Vol}(v1181) - 2\,\mathrm{Vol}(m155)$$

in agreement with Neumann's conjecture.

There are no manifolds in `OrientableCuspedCensus` with shape field defined by

$$p_2(x) = x^4 - x^3 - x^2 - 2x + 4.$$

However, the number field determined by $p_2(x)$ has 3 proper subfields:

```
Pari/GP
>>> nf=nfinit(x^4 - x^3 - x^2 - 2*x + 4) \\ initializes number field for p_2
>>> nfsubfields(nf)
[[x - 1, 1], [x^2 - x + 1, -1/2*x^3 - 1/2*x^2 + 1/2*x + 2],
 [x^2 - x + 2, x^3 - 2], [x^2 - x - 5, -1/2*x^3 + 1/2*x^2 + 3/2*x + 1],
 [x^4 - x^3 - x^2 - 2*x + 4, x]]
  # 3 proper subfields (the first and last are Q and the field itself).
```

From this it follows that the number field equals $\mathbb{Q}(\sqrt{-3}, \sqrt{-7})$. Using Example 2.12 we observe that

$$4.05976642563861 \approx 2\,\mathrm{Vol}(m004).$$

Next, one checks (do this!) that e.g. 10_155 and v3461 have shape fields defined by

$$p_3(x) = x^4 - x^3 - x^2 + x + 1,$$

and that we have

$$1.13100876173487 = 3\,\mathrm{Vol}(10_1 55) - 4\,\mathrm{Vol}(v3461). \tag{2.12}$$

Finally, the last volume is just the volume of 10_155 itself.

2.5. **Looking systematically for examples.** One of the major goals of the project is to write new software tools that systematically searches for examples of Neumann's conjecture as in Example 2.37. The main obstacles in doing so are:

(1) There are a lot more hyperbolic manifolds than those in the censuses, so a volume might be realized by manifolds that are not census manifolds.
Solution: Create a large new census of manifolds with shape field of small degree (see Section 2.5.1).
(2) A representation may be defined over a number field F of large degree. If so, it is unlikely that even after enlarging the search space, that we can find enough hyperbolic manifolds with shape field contained in F.
Solution: Restrict to representations with small degree (see Section 2.5.2).
(3) The set of volumes is huge, so searching among all manifolds is completely infeasible.
Solution: Restrict to manifolds with shape field contained in F, and use tools available in Pari/GP (see Section 2.5.3).

2.5.1. *Dehn surgery.* One can obtain new manifolds from the census manifolds by a process called Dehn surgery. We shall not attempt to understand this process. Just think of it as a way of creating new manifolds from old ones. Each census manifold has a number of cusps and we refer to the process as "filling the cusps".

Example 2.38. `SnapPy`

```
>>> M=Manifold('m084');N=Manifold('m292');
>>> [M.num_cusps(),N.num_cusps()]
[1,2] # M has 1 cusps, N has 2 cusps
>>> M
m084(0,0)
>>> N
m292(0,0)(0,0)
```

The number of cusps is the number of parantheses with zeros. For each cusp, the zeros can be replaced by a pair (p, q) of relatively prime integers.

```
>>> N.dehn_fill([(1,4),(2,3)])
>>> N
m292(1,4)(2,3)
>>> N.volume()
3.1472791248
>>> N.dehnfill([(1,2),(0,0)]) # leaves second cusp unfilled.
```

Each Dehn filling gives rise to a new manifold. If all cusps have been filled we say that the manifold is closed. The SnapPy census `OrientableClosedCensus` is a list of closed manifolds with small volumes obtained by fillings. We want instead a census of manifolds with small shape fields.

Exercise 2.39. Consider the manifold m015. Create triangulation files for all possible Dehn surgeries with coefficients (p, q) relatively prime with $-16 \leq p \leq 16$, $1 \leq q \leq 16$. How many of these have shape field of degree ≤ 8.

Exercise 2.40. Create a large census of manifolds with shape fields of degree ≤ 8 obtained by Dehn surgeries on census manifolds. In practice one rarely gets low degree shape fields if the Dehn surgery coefficients are larger than, say 16. This will require a lot of CPU time, so we will have computers working on this 24/7.

2.5.2. *Representations with low degree number fields.* The second obstacle is easy to overcome. We just restrict to the representations that are defined over number fields of degree ≤ 12.

Exercise 2.41. Create a list of all representations of census manifolds defined over number fields of degree ≤ 12 (using the `retrieve.solutions` command). Do this also for representations in $\mathrm{PGL}(3,\mathbb{C})$ (see Example 2.21).

2.5.3. *Searching for dependence relations.* Pari/GP has built in methods for searching for dependence relations.

Example 2.42. `Pari/GP`

```
>>> x0=1;x1=sqrt(2);x2=sqrt(3);x3=sqrt(6);
>>> x=0.60039830998148144005591988409615526336;
>>> lindep([x0,x1,x2,x3,x])
[-1, 1, -2, 1, 1]~ # -x0+x1-2*x2+x3+x=0
>>> x=Pi;
>>> lindep([x0,x1,x2,x3,x])
[78989, -355797, 589666, -1195843, 742317]~
  # Gives an approximation of Pi as a rational
    linear combination of x0,x1,x2, and x3.
```

If the coefficients are large, like in the second case, the approximation is unlikely to be exact.

Example 2.43. Let's find the linear combination in the last part of Example 2.37:

```
SnapPy
>>> M=Manifold('10_155');
>>> pari.set_real_precision(100); # sets the precision to 100 digits
>>> sols1=M.ptolemy_variety(2,1).retrieve_solutions();
>>> sols1[3].volume_numerical()[0]
1.13100876173486977651917745105789780814590229893942915435
63805851904023773604010518177718959080422
>>> sols1[3].volume_numerical()[3]
9.25054161301209877668879788525406009567941675918182749049
55509127318200478019922822735400234168577 80
>>> ManifoldHP(v3461).volume()  # HP stands for high precision.
6.65515401932535663838680405117607061972308699465151332928275367 0
Pari/GP
>>> lindep([1.13100876173486977651917745105789780814590229893942915435
... 63805851904023773604010518177718959080422 2,9.25054161301209
    87766887978852540600956794167591818274904955509127318200478
    01992282273540023416857780,6.65515401932535663838680405117 6
```

```
    07061972308699465151332928275367 0])
[-1, 3, -4]~
```

This is a very strong indication (but of course no proof), that (2.12) holds.

2.5.4. *Strategy of investigation.* Our strategy for investigating Neumann's conjecture is the following:

(1) Create a large census C of manifolds with shape fields of small degree. Compute their volume to high precision.
(2) For each manifold M for which the Ptolemy varieties have been computed (all manifolds in `OrientableCuspedCensus`, almost all manifolds in `LinkExteriors` and a lot of manifolds in `HTLinkExteriors` carry out the following steps:
    - For all small degree representations, compute all volumes to high precision.
    - Simplify each polynomial, and check if the simplified polynomial p appears the census C. If it does, run `lindep` on the set of manifolds with shape field given by p (many manifolds will have the same volume; ignore duplicates).
    - If it does not, check if F has any subfields in C. If so, run `lindep` on those. If not, continue with the next volume.
(3) Create a list of all examples found this way.

Exercise 2.44. Implement the above.

Remark 2.45. A systematic search like this has never been done before, so with a bit of luck we will discover lots of new and interesting patterns!!

2.6. General SnapPy tips.

2.6.1. *Tab completion and help.* Tab completion gives a list of available commands.

Example 2.46. `SnapPy`

```
>>> M=Manifold('m004');
>>> M. # type tab
M.DT_code
M.LE
M.browse
M.canonize
M.chern_simons
M.clear_cache
M.complex_volume
M.copy
M.cover
M.cover_info
. # We shall not display all
```

To get an explanation for the various commands available, type ?

```
>>> M.dehn_fill? # Gives explanation.
>>> PtVars=M.ptolemy_variety(2,'all');
>>> Var0=PtVars[0]
>>> Var0? # Gives explanation.
```

2.6.2. *Basic scripting syntax.* SnapPy is a Python interpreter, and thus allows for easy scripting.

Example 2.47. `SnapPy`

```
>>> for M in OrientableCuspedCensus(tets=6,cusps=2):
...     if M.volume()<5:
          print (M,M.volume())
(s441(0,0)(0,0), 4.7517019655)
(s443(0,0)(0,0), 4.75170196552)
(s503(0,0)(0,0), 4.8937641326)
(s506(0,0)(0,0), 4.8937641326)
(s548(0,0)(0,0), 4.97677029426)
(s549(0,0)(0,0), 4.9767702943)
```

2.6.3. *Precision.* One can get higher precision using `ManifoldHP`.

Example 2.48. `>>> M=ManifoldHP('m032');`

```
>>> M.volume()
3.16396322888314398399101471597315448481278767151811526658292244 2
>>> M.dehn_fill([(2,3)]
>>> M.volume()
2.75862016077889095791779156611872730924997028423146230258039607 8
```

When working with the Ptolemy module, one changes the precision with the command `pari.set_precision(n)`, where n is the number of digits (see Example 2.43).

References

[1] Riccardo Benedetti and Carlo Petronio. *Lectures on hyperbolic geometry*. Universitext. Springer-Verlag, Berlin, 1992.

[2] Stavros Garoufalidis, Matthias Goerner, and Christian K. Zickert. The Ptolemy field of 3-manifold representations. *arXiv:1401.5542*, Preprint 2014.

[3] Stavros Garoufalidis, Dylan P. Thurston, and Christian K. Zickert. The complex volume of $\mathrm{SL}(n,\mathbb{C})$-representations of 3-manifolds. *ArXiv:math.GT/1111.2828*, 2011.

[4] Colin Maclachlan and Alan W. Reid. *The arithmetic of hyperbolic 3-manifolds*, volume 219 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2003.

[5] Walter D. Neumann. Realizing arithmetic invariants of hyperbolic 3-manifolds. In *Interactions between hyperbolic geometry, quantum topology and number theory*, volume 541 of *Contemp. Math.*, pages 233–246. Amer. Math. Soc., Providence, RI, 2011.

[6] Jeffrey R. Weeks. Computation of hyperbolic structures in knot theory, 2003.