Computational tools in 3-dimensional topology

Nathan Dunfield, University of Illinois, Urbana-Champaign

I will demonstrate a wide variety of tools for studying Dehn surgery, hyperbolic geometry, Floer homology, foliations, character varieties, and Heegaard splittings, all of which can be used together via Python/SageMath via the computop/sage Docker image.

Dehn surgeon school: Nathan's HW 1 Wednesday, July 17, 2019.

References and links for the second computational session, which demonstrated a wide variety of tools for studying Dehn surgery, hyperbolic geometry, Floer homology, foliations, character varieties, and Heegaard splittings, all of which can be used together via Python/SageMath via the computop/sage Docker image.

- http://snappy.computop.org
- http://www.sagemath.org
- http://bitbucket.org/t3m/sagedocker
- http://doi.org/10.7910/DVN/LCYXPO
- http://github.com/bzhan/bfh_python
- http://regina-normal.github.io

1. Get SageMath and SnapPy working together on your laptop, for example using the computop Docker image. Alternatively, from any of the physical ICERM terminals you can access it via http://icerm2.icerm.brown.edu:8888.

2. You can get a knot of 14 or fewer crossings in SnapPy by doing:

   ```
   knots = snappy.HTLinkExteriors(cusps=1)
   E = knots.random()
   ```

   Use verified computation as described here: http://snappy.computop.org/verify.html to prove it is hyperbolic and to compute its volume to a provably correct 250 decimal places. By Mostow rigidity this number is an invariant of the knot exterior and hence of the knot itself. (There are a handful of non-hyperbolic links in this range, so you're very unlike to pick one of them and so be unable to complete this problem!)

3. Look at the documentation for HTLinkExteriors by typing

   ```
   ?snappy.HTLinkExteriors
   ```

   to figure out how to pick a random **10 crossing** knot. Download the software of http://doi.org/10.7910/DVN/LCYXPO and use it to find coorientable taut/Reebless foliations on at least one Dehn surgery of your random knot.

4. Python programming practice:

   Use http://snappy.computop.org/spherogram.html to write a Python function to produce a link projection of the $(a_1, a_2, \ldots, a_k)$ pretzel link. For the $(-2, 3, 7)$ pretzel knot, write a procedure that searches for the two slopes of the two len space Dehn surgeries discovered by Fintushel-Stern. Use Regina to determine which lens spaces these are. Can you find lens space surgeries on other pretzel knots?

5. Look at the list of software that is part of the computop Docker image. See if you can compute something interesting with one of them.

6. The webpage http://computop.org lists a wide variety of computational tools in low-dimensional topology. Find one that is relevant to your own work and try to get it working in your Docker container.

Proving manifolds are hyperbolic

This is a Jupyter notebook, which works similar to a Maple or Mathematica notebook.

```
In [1]: import snappy
```

You can mix code and text, even with math(s): $\int_0^\infty x^{-2} dx$

```
In [2]: len(snappy.HTLinkExteriors)

Out[2]: 180510
```

```
In [3]: M = snappy.HTLinkExteriors.random()
```

```
In [4]: M

Out[4]: L14n11157(0,0)(0,0)
```

```
In [5]: M.volume()

Out[5]: 18.0675611176150
```

```
In [6]: M.solution_type()

Out[6]: 'all tetrahedra positively oriented'
```

```
In [7]: M.verify_hyperbolicity()

Out[7]: (True,
         [-0.0621537131329? + 1.0178073903282?*I,
          0.059774970118? + 0.978853929655l?*I,
          0.638466496795? + 1.441708925408?*I,
          0.2577526846089? + 0.6777228769149?*I,
          0.4689231434336? + 0.5089036951641?*I,
          0.662460312241? + 1.314626609150?*I,
          0.5818380652715? + 1.0991958451076?*I,
          0.2732769626431? + 0.3330550329362?*I,
          0.1249879363912? + 0.6734001962976?*I,
          0.5177383714016? + 0.2092928551311?*I,
          0.3815308239748? + 1.1424738781077?*I,
          0.958838864608? + 1.108858231676?*I,
          0.6613225634146? + 1.3500175082759?*I,
          0.3806174629844? + 0.7043673148659?*I,
          0.4366921328557? + 0.4496161851162?*I,
          0.1660936567574? + 0.828126772153?*I,
          0.1748238444990? + 0.6968732647716?*I,
          0.7228861202326? + 0.4363507298890?*I,
          0.0848890117025? + 0.6343178679268?*I,
          1.037125892189? + 1.633085431964?*I])
```

```
In [8]: M.volume(bits_prec=1000, verified=True)

Out[8]: 18.0675611176149961411408981133339043646217755355843719538025393513960312011793159823325710 25
        5285639238724197637284294104181338622212115159576620581758596883915793254233307662138915460 51
        5612954509070158900618191910322711966601686160578994891327545133128366729157696121705031621 38
        2436415483973951424?
```

Foliations and Floer homology for fun and profit

First, let's find some foliations using the software available here: https://doi.org/10.7910/DVN/LCYXPO (https://doi.org/10.7910/DVN/LCYXPO)

```
In [1]: import snappy, foliar
```

First, we build the (-2, 3, 7) pretzel knot programmatically.

```
In [2]: RT = snappy.RationalTangle
        P = (RT(-1/2) + RT(1/3) + RT(1/7)).numerator_closure()
        E = P.exterior()
        E.identify()

Out[2]: [m016(0,0), K3_1(0,0), K12n242(0,0)]
```

```
In [3]: E.dehn_fill((2, 0))
        covers = E.covers(2)
        len(covers)

Out[3]: 1
```

```
In [4]: C = covers[0]
        C.volume()

Out[4]: 0.000000000000000
```

After looking at the README file for this software, we search for a taut foliation and find one.

```
In [5]: eo = foliar.first_foliation(C, 5, 25)
```

```
In [6]: eo

Out[6]: <foliar.edge_orient.EdgeOrientation object at 0x7f12ed5ea2d0>
```

```
In [7]: eo.gives_foliation()

Out[7]: True
```

Now, let's compute some Floer homology using https://github.com/bzhan/bfh_python (https://github.com/bzhan/bfh_python)

```
In [8]: import sys
        sys.path.append('bfh_python')
        import braid
```

First, we find by hand a bridge/plat presentation for $P(-2,3,7)$ in BHF's notation, which is based on Artin generators of the braid group. The error in my talk was that the Morse diagram was not actually a bridge diagram even though SnapPy claimed it was; this bug will be fixed in the next release.

```
In [9]:  # Pairing of strands at bottom and top of the plat.

         pairing = [6, 3, 2, 5, 4, 1]

         # The braid

         word = 2*[-1] + 3*[3] + 7*[5]
         word

Out[9]:  [-1, -1, 3, 3, 3, 5, 5, 5, 5, 5, 5, 5]
```

```
In [10]: bp = braid.BridgePresentation("P(-2, 3, 7)", pairing, word, pairing)
```

```
In [11]: %time bp.getHFByLocalDA()

         1 2 3 4 7 8 11 12 15 14 10 13 16 15 11 14 17 16 14 17 20 19 17 20 23 22 20 2
         3 26 25 23 26 29 28 26 29CPU times: user 2min 7s, sys: 180 ms, total: 2min 8
         s
         Wall time: 2min 8s

Out[11]: Chain complex.
         d(g198) = 0
         d(g18) = 0
         d(g90) = 0
```

Finally, use https://regina-normal.github.io/ (https://regina-normal.github.io/) to identify the Seifert fibered space C.

```
In [12]: import regina
```

```
In [13]: T = C.filled_triangulation()
         R = regina.Triangulation3(T._to_string())
         R.isHaken()

Out[13]: False
```

```
In [14]: R.countTetrahedra()

Out[14]: 7
```

```
In [15]: regina.Census.lookup(R).first().name()

Out[15]: 'SFS [S2: (2,1) (3,1) (7,-6)] : #1'
```