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'
```