Khovanov homology with mod 2 coefficients in `Python`

A coding supplement to the article
Two-fold quasi-alternating links, Khovanov homology and instanton homology
by Christopher Scaduto & Matthew Stoffregen

This document explains the following Python scripts, available online [sca]:

- `kh_mod2.py`, reduced Khovanov homology with mod 2 coefficients
- `kh_twisted.py`, *Twisted Khovanov homology* for a two-fold marked link
- `kh_dotted.py`, *Dotted diagram homology* of a two-fold marked diagram

These Python modules were written to produce data for the TQA paper [SS16]. The first file, `kh_mod2.py`, computes the reduced Khovanov homology of a link with mod 2 coefficients. Although much less efficient and more bare-bones than other available Khovanov homology programs, we include the code here for two reasons. (1) The code is on the shorter side, and is in an easily understood language (Python), so it may be useful for learning purposes. (2) It is rather easy to modify and should be useful for quickly coding variations of Khovanov homology. The second and third files `kh_twisted.py` and `kh_dotted.py` are in fact modifications of the first file, and are what we needed to produce examples for our paper.

We remark that any conventions for Khovanov homology or terms unknown to the reader that are not explained here can be found in our paper [SS16].

1 `kh_mod2.py`

To help the interested user navigate and modify the code, we explain what's in `kh_mod2.py`. The first function in the file, `homology(complex)`, computes the homology of a mod 2 chain complex. The algorithm is the standard "gaussian elimination" algorithm, and is described as follows (from the file):

```
# This function computes the homology of a chain complex
# with coefficients in the field with 2 elements.
#
# The input, ‘complex’, is a dictionary, written as follows:
# complex = [[a, gr(a), [a_1, ..., a_n]], [b, gr(b), [...]], ...].
# We assume gr(a) takes values from 0 to some positive integer.
#
# The notation here is as follows:
# complex.......a chain complex over the field with 2 elements
# a, b, ........generators of the chain complex
# gr(a).........the grading of the generator a
# a_1, a_2, ....the differential takes a to a_1 + ...  + a_n.
#
# The algorithm ("gaussian elimination"):
# If the differential is zero, return the complex.
# Otherwise, choose an a with non-zero differential.
# If d(a) = a_1 + ...  + a_n, choose some a_i, say a_1.
# For each generator b with d(b) containing a_1, and
# each c in d(a) not a_1, do the following:  if c is in d(b),
# remove it from d(b); otherwise, add c to d(b).
# Then delete a and a_1 from the complex.  Repeat!
```

For example, a complex that has two generators, `‘a’` and `‘b’`, in gradings 1 and 0, respectively, and a nonzero differential from `‘a’` to `‘b’`, is represented by `[[‘a’, 1, [‘b’]], [‘b’, 0, []]]`. In a Python shell running our module we apply the function `homology` to this complex, and get zero homology back, as expected:

```
>>> complex = [[‘a’, 1, [‘b’]], [‘b’, 0, []]]
>>> homology(complex)
[]
```

The shell returned `[]`, which is an empty set. Note that our generators `‘a’` and `‘b’` are strings, but they can be of any type (as long as they are not equal to each other). If instead we want the differential to be zero between `‘a’` and `‘b’`, then we get:

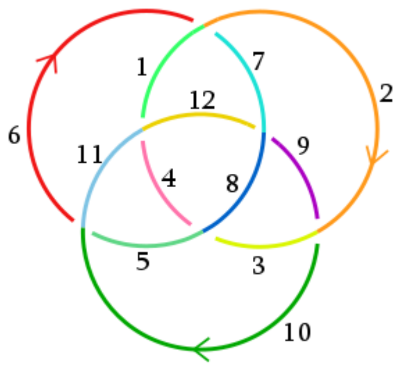
```
>>> complex = [[‘a’, 1, []], [‘b’, 0, []]]
>>> homology(complex)
[[‘a’, 1, []], [‘b’, 0, []]]
```

Returned is the original complex, which is the homology, having set the differential equal to zero. The next function, `betti(complex)`, takes in a complex as above and returns the dimensions of the homology in its different gradings, i.e. the betti numbers of the complex. What is returned is of the form `[[i, b_i], ...  ]`, which means that the dimension in grading `i` is equal to `b_i`. We remark that a “grading” need not be an integer; in the application of Khovanov homology, the symbol `i` will record two numbers `[q,h]`, the quantum and homological gradings. In any case, in our simple (integer-graded) example above, we may compute:

```
>>> complex = [[‘a’, 1, []], [‘b’, 0, []]]
>>> betti(complex)
[[1,1], [0,1]]
```

Thus the nonzero betti numbers are in gradings 0 and 1, and are both equal to 1.

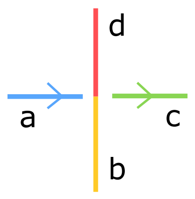
Now we move on to creating the Khovanov complex. For this we need a way of encoding a link diagram. We use *Planar Diagram* (PD) notation. If you know the name of the link you want to use, then this presentation is available on either the Knot Atlas [BNM] or KnotInfo [CLa] and LinkInfo [CLb], although the latter pair of references' syntax is closer to what we use. We first orient the link, and order its components. We define an *arc* to be an edge of the 4-valent graph obtained from the diagram by forgetting the over and under crossing information. Beginning at the first component, label the arcs `1,2,3,...`, and so forth, up to, say, `k`. It does not matter which arc is labelled first, but we make sure to label them around the component in the direction of our orientation. Then, label the arcs of the second component `k,k+1,...`, and so forth. Here's an example (the link L6n1):



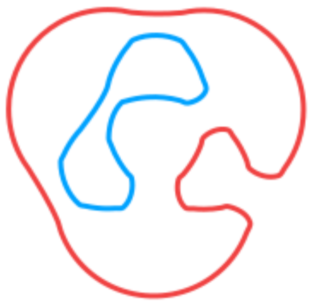
Each arc has been colored differently. Now, the PD notation for this diagram is given by:

```
D = [[6,1,7,2],[12,8,9,7],[4,12,1,11],[5,11,6,10],[3,8,4,5],[9,3,10,2]]
```

This particular diagram, as indicated, is included in the file under the variable name `D`. The notation is explained as follows. Each 4-tuple in the above list corresponds to a crossing. The 4-tuple `[a,b,c,d]` means that the arcs appearing in counterclockwise order at the crossing are `a,b,c,d`, and that arc `a` is proceeding *under* the crossing (here we use the orientation). The picture is:



In constructing the Khovanov complex, the first thing we need to do is take complete resolutions of our diagram. Suppose our diagram has n crossings. For each vector `v` which is an n-tuple of 0's and 1's, we can perform a 0-smoothing and a 1-smoothing at the crossings to obtain a disjoint union of planar circles. For example, if we use the diagram above (with crossings as ordered in the PD notation) and `v = [1,1,0,0,0,1]`, then we get:



We can write each circle as a union of the arcs. Thus any complete resolution may be encoded as a partition of the set of arcs. The resolution with two circles depicted above may be written from the prior PD notation as follows:

```
        [[6,2,9,8,3,10], [1,7,12,4,5,11]]
```

This should give the reader enough background to understand the rest of the code in the file `kh_mod2.py`. The function `redcomp(D)` returns a chain complex for the reduced Khovanov homology. Thus to compute the reduced mod two Khovanov homology of the link represented by the above PD notation, we may run the command `homology(redcomp(D))`. The output will be (in general) a long list: the elements of the chain complex are subsets of resolutions written as we just explained, with the resolution vertex in $\{0,1\}^{\#\text{ crossings}}$ carried in the notation. To extract just the (bigraded) betti numbers of the reduced Khovanov homology, we run instead the following:

```
>>> betti(redcomp(D))
[[[0, 0], 2], [[2, 0], 1], [[2, 1], 1], [[4, 2], 1], [[8, 4], 1]]
```

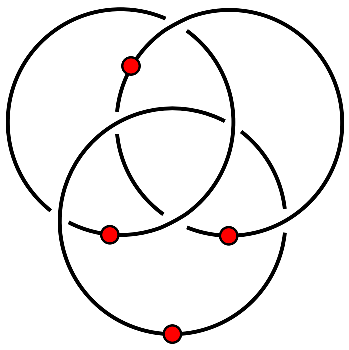
The output should be read as saying that in (quantum, homological)-bigrading $(0,0)$, the rank is 2, while in bigradings $(2,0),(2,1),(4,2),(8,4)$ the homology has rank 1. In particular, the total dimension of the (reduced) Khovanov homology of L6n1 is equal to 6. We remark that our quantum grading is twice the quantum grading of another common convention in the subject.

2 kh_twisted.py

The file `kh_twisted.py` computes the (reduced) twisted Khovanov homology of a two-fold marked link. This is denote $\mathrm{Kh}(L,\omega)$ in our paper [SS16], where L is the link and ω is the marking data. To compute this invariant, one starts with a marked diagram $(D,\check{\omega})$ representing (L,ω). Here D is a diagram for L and $\check{\omega}$ is a collection of dots on the set of arcs of D. Thus $\check{\omega}$ can be encoded by a vector in $\{0,1\}^{\#\text{ arcs}}$ and in the file `kh_twisted.py`, this vector is denoted by `f`. Thus a valid two-fold marking for the diagram `D` from above is:

```
f_D = [1,0,1,0,1,0,0,0,0,1,0,0]
```

Thus there is one dot on arcs numbered 1, 3, 5, 10, and no dots on all other arcs. The pair `D`, `f_D` thus represents the two-fold marked diagram for L6n1 depicted as follows:



This marked diagram appears in Figure 14 of [SS16]. Now the function `tcomplex(D,f_D)` produces the unreduced twisted Khovanov chain complex for a general marked diagram, while `tredcomp(D,f_D)` returns the *reduced* twisted Khovanov chain complex. We no longer have a bigraded complex, but a $\mathbb{Z}$-graded complex, for which the grading is denoted δ. This grading requires the computation of the signature and nullity of the link, which we do not include – thus our complex is only correctly graded up to an overall shift. (See the comments in the code for how to fix the absolute grading.) To compute the twisted Khovanov homology of the above marked link, we run:

```
>>> betti(tredcomp(D,f_D))
[[0.0, 4]]
```

The output says that the (reduced) twisted Khovanov homology of L6n1 with above marking is of rank 4 and in δ-grading 0. Note that in this case, the signature and nullity of L6n1 are zero, and so the grading computed by our code needs no correction shift.

Through our examples we have seen that the mod two reduced Khovanov homology of L6n1 is of rank 6, while for the above two-fold marked L6n1, the mod two twisted Khovanov homology is of rank 4. The determinant of L6n1 is 4. Thus L6n1 with the marking `f_D` is “thin” for twisted Khovanov homology, while L6n1 with no marking is not.

3 `kh_dotted.py`

Finally, we have a script that computes the “dotted diagram” homology of a two-fold marked diagram $(D, \breve{\omega})$. This is denoted $\mathrm{Hd}(D, \breve{\omega})$ in the paper [SS16], and is in general only an invariant of the diagram $(D, \breve{\omega})$, not the two-fold marked link (L, ω) that it represents. Recall that this homology is defined as the E^2-page of a double-complex spectral sequence that converges to $\mathrm{Kh}(L, \omega)$: it is obtained by first taking the vertical homology of the twisted complex, and then the induced horizontal homology. The grading is an integer grading that is defined as is the δ-grading for $\mathrm{Kh}(L, \omega)$ in the previous section.

We only briefly describe the contents of this file, as it is similar to the previous one. The function `dcomplex(D,f)` constructs an unreduced version of the dotted diagram complex, while `dredcomp(D,f)` gives the complex that computes $\mathrm{Hd}(D, \breve{\omega})$. Thus the following

```
>>> betti(dredcomp(D,f_D))
[[0.0, 4]]
```

says that the dotted diagram homology of the above marked diagram for L6n1 is of rank 4. Indeed, in this case, the spectral sequence from the dotted diagram homology to the twisted Khovanov homology collapses, and both are thin.

The data of Table 1 from [SS16] is computed with the use of these three scripts. There was only some additional code, not included here, that helped automate the computations – code, for example, to run through enough two-fold markings of a given diagram. We mention also that the data of Figure 14 was computed using `kh_dotted.py`.

References

[BNM] Dror Bar-Natan and Scott Morrison. *KnotAtlas*. `http://katlas.org/`.

[CLa] Jae Choon Cha and Charles Livingston. *KnotInfo*. `http://www.indiana.edu/~knotinfo/`.

[CLb] Jae Choon Cha and Charles Livingston. *LinkInfo*. `http://www.indiana.edu/~linkinfo/`.

[sca] `https://www.ic.sunysb.edu/Faculty/cscaduto/code/`.

[SS16] Christopher Scaduto and Matthew Stoffregen. *Two-fold quasi-alternating links, Khovanov homology and instanton homology.* arXiv:1605.05394, 2016.