# DOCUMENTATION FOR SATELLITES PYTHON CODE

DAREN CHEN, IAN ZEMKE, AND HUGO ZHOU

## 1. Overview

The code is written in a Jupyter Notebook in python. You will need to install `sympy` and `numpy`.

## 2. Classes

We now enumerate the main classes in the program. (There are a number of extra classes which are used internally for the code).

### 2.1. `CFK`.

This class encodes the data of the full knot Floer complex, viewed as a free, finitely generated chain complex over the ring $R = \mathbb{F}[W, Z]$. Let `C` be an instance of this class. The main attributes are the following:

(1) `C.gens` This is a list of length 2 lists. An entry of `C.gens` corresponds to an $R$-basis element. The first component is the $\mathrm{gr}_{\mathbf{w}}$-grading, while the second is the $\mathrm{gr}_{\mathbf{z}}$ grading. Example `[[-2,0],[-1,-1],[0,-2]]` means we have three generators, $x_0, x_1, x_2$ which have $(\mathrm{gr}_{\mathbf{w}}, \mathrm{gr}_{\mathbf{z}})$-bigradings $(-2, 0)$, $(-1, -1)$ and $(0, -2)$, respectively.

(2) `C.diff` this is the differential of underlying chain complex, encoded as a class of type `mor`. See the `mor` package below for more information on this. To print the differential, we type `C.diff.matrix`.

It is initialized with `CFK(gens, diff)`

Some helpful functions:

(1) `C.tau()` returns the $\tau$ invariant of a `CFK` object.

(2) `C.Vs(n)` returns $V_n(K)$ of a `CFK` object `C`.

(3) `CFK_reduced(C)` returns a reduced model (homotopy equivalent but with differential in the ideal $(W, Z)$) of `C`.

### 2.2. `XK`.

This class is contains the data of a type-$D$ module $\mathcal{X}_n(K)^{\mathcal{K}}$ when $K$ is a knot in an integer homology 3-sphere.

It is initalized by typing `XK(gens_0,gens_1,diff_0,diff_1, v_map,h_map)`

(1) `gens_0` is a list of length 2 lists (as in `CFK`).

(2) `gens_1` is a list of integers. Each integer is the Maslov grading of a generator of $CF^-(Y)$.

(3) `diff_0` is an object of class `mor` which is the differential on idempotent 0.

(4) `diff_1` is an object of class `mor` which is the differential on idempotent 1.

(5) `v_map` is the $v$ map from the mapping cone formula. It is encoded as an object of class `mor`. The input `h_map` is similar. These

2.3. `h2_func`. The class `h2_func` is a helpful class for computations involving the $H$-function of a 2-component link.

There is an initializer which uses just the values of the $H$-function in a given range, but the most useful functions are

(1) `Alex_to_h2(linking, A,A1,A2)` Here, `linking` is the linking number, while `A` is the Alexander polynomial of L and `A1` and `A2` are the Alexander polynomials of the components $K_1, K_2 \subseteq L$. This function has polynomials input as lists of tuples. `A` is a list of triples `[a,s1,s2]` corresponding to a monomial $at_1^{s_1}t_2^{s_2}$. We assume these are normalized as in the article. `A1` and `A2` are lists of integers, so `[-1,0,1]` corresponds to $t^{-1} - t + t^1$. `A1` and `A2` do not need to be normalized, though `A` does.
(2) `Alex_to_h2_poly(linking, A, A1,A2)` is a more user friendly method for inputting Alexander polynomials. Here, we input the polynomials as sympy polynomials. We input `A1` and `A2` as polynomials in a variable `T`, so the trefoil would be `1-T+T^2` (the overall sign is not important for these two). Negative powers are not allowed. For `A`, the polynomial is input in variables `U` and `V`, which correspond to $t_1^{1/2}$ and $t_2^{1/2}$ (we do this because fractional powers seem to not be handled well by sympy). Almost no normalization is required: we only renormalize up to $\pm 1$. If the wrong sign is input, the program may fail.

A particularly entertaining function is the `print` function. If `H` is an instance of this class, then `H.print()` prints the values of the h-function in some default range. `H.print([A,B],[C,D])` prints the H-function in the given range. The code should accept either integers, or half integers. Half integers should be rationals (e.g. `rational(1/2)`).

An important function is `h2_to_KXR(H)` which has input an object of type `h2_func` and outputs the `KXR` object build from the algorithm in the paper. (I.e. this computes the corresponding bimodule ${}_{\mathcal{K}}\mathcal{X}(K)^R$).

2.4. `KXR`. This class encodes the data of the bimodules ${}_{\mathcal{K}}\mathcal{X}(L)^R$ in the paper. The most practical constructor is from an `h2_func` object, as described in the previous section.

If `X` is an `KXR` object, then we can build a finite truncation of the complex $\mathbb{X}(P, K, n)^R$ from the article via the command `satellite(K,X,n)`. Typically one also wants to subsequently reduce this, which can be accomplished by subsequently applying the function `CFK_reduced`.

2.5. `mor`. This class is the main tool for encoding a map between spaces. We view $f$ as a map between finite dimensional vector spaces $C$ and $D$ (or finitely generated, free $R$-modules, $R = \mathbb{F}[W, Z]$, which have a $(\mathrm{gr}_{\mathbf{w}}, \mathrm{gr}_{\mathbf{z}}$-bigrading) Here are the attributes of a morphism $f$.

(1) `f.co_rank` This is the rank of the codomain of $f$.
(2) `f.matrix` This is a list of sets.
(3) `f.rank` is the length of `f.matrix`.

Example: if `f.rank=2` and `f.co_rank=3`, then we might have `f.matrix=[set(),{1,2}]` . This would correspond to the linear map $f$ from $\mathrm{Span}(x_0, x_1)$ to $\mathrm{Span}(y_0, y_1, y_2)$ with $f(x_0) = 0$ and $f(x_1) = y_1 + y_2$.

To build a morphism, the command is `mor(matrix,co_rank)`. If one enters `mor(matrix)` then `co_rank` is set to `len(matrix)` (i.e. we build a square matrix).

## 3. Helpful basic functions

**3.1. Cabling.** If `K` is an `XK` object, and `p,q` are coprime (possibly negative) integers, then the cabling complex can be computed using the command `XK_cable(K,p,q)`.

**3.2. Whitehead and Mazur links.** There is a family of patterns considered in the paper $P_m$, where $P_1$ is the Whitehead pattern, $P_2$ is the Mazur pattern, and $P_j$ ($j \geq 2$). If `K` is an `XK` complex, then we can compute the satellites via the command `XK_Mazur(K,m,n)`.

Department of Mathematics, California Institute of Technology, Pasadena, CA, USA
*Email address*: darenc@caltech.edu

Department of Mathematics, University of Oregon, Eugene, OR, USA
*Email address*: izemke@uoregon.edu

Department of Mathematics, University of Michigan, Ann Arbor, MI, USA
*Email address*: hugozhou@umich.edu