Knoto-ID

Topological invariants of open curves
using the concept of knotoids

User guide

Version 1.1.1 (May 1, 2018)

Julien Dorier[1,3], Dimos Goundaroulis[2,3]

[1] Vital-IT, SIB Swiss Institute of Bioinformatics, Switzerland.
[2] SIB Swiss Institute of Bioinformatics, Switzerland.
[3] Center for Integrative Genomics, University of Lausanne, Switzerland.

Contents

1 Introduction

The backbone of most proteins forms an open curve. To study their entanglement, a common strategy consists in searching for the presence of knots in their backbones using topological invariants. However, this approach requires to close the curve into a loop, which alters its geometry. *Knoto-ID* allows evaluating the entanglement of open curves without the need to close them, using the recent concept of knotoids [1, 2] which is a generalization of classical knot theory to open curves. *Knoto-ID* can analyse the global topology of the full chain as well as the local topology by exhaustively studying all subchains or only determining the knotted core. The use of *Knoto-ID* is not limited to proteins, it can be used to analyse any open curve in 3D space such as chromosomes, synthetic polymers, random walks, etc.
Knoto-ID is a collection of command line tools with three executables (`polynomial_invariant` , `knotted_core` and `convert_diagram`) and five helper scripts (`pdb_to_xyz.R` , `plot_3D_curve.R` , `plot_projection_map.R` , `plot_knotted_core.R` and `plot_diagram.R`):

- `polynomial_invariant` computes the following polynomial invariants: classical Jones polynomial for knots (closed curves), Jones polynomial for knotoids (open curves projected on a sphere) and Turaev loop bracket for knotoids (open curves projected on a plane). `polynomial_invariant` can output lists of polynomials obtained with multiple projection directions, which can be used to generate projection maps similar to those presented in [3]. A simple helper script `plot_projection_map.R` is included with *Knoto-ID* to create such projection maps using the output of `polynomial_invariant` .
- `knotted_core` computes the knotted core of an open or closed curve. It can also be used to evaluate the dominant invariant polynomials for all subchains of the input curve. This data can be used to generate fingerprint matrices [4, 5, 3] (for open curves) or disk matrices [6] (for closed curves). A simple script `plot_knotted_core.R` is included with *Knoto-ID* to create such fingerprint or disk matrices using the output produced by `knotted_core` .
- `convert_diagram` convert diagrams from/to PD codes, extended Gauss codes and piecewise linear curve (xyz). In particular, the xyz output format can be used to draw the diagram with the helper script `plot_diagram.R` which is included with *Knoto-ID*.
- `pdb_to_xyz.R` is a simple helper script to convert the backbone of proteins given in pdb format [7] to piecewise linear curves in xyz format that can be used as input by *Knoto-ID*.
- `plot_3D_curve.R` is a simple helper script to convert piecewise linear curves from xyz format to webGL format [8] that can then be viewed interactively in compatible web browsers.

This document starts with a tutorial (section "2 Tutorial") where examples of use cases are shown. More details on the executables and their arguments, as well as a description of the file formats are given in section "3 Command reference". Mathematical concepts, definitions and choices of conventions are given in section "4 Theory".

1.1 Credit

If you use this software for a publication, please cite:
J. Dorier, D. Goundaroulis, F. Benedetti and A. Stasiak, "Knoto-ID: a tool to study the entanglement of open protein chains using the concept of knotoids", Bioinformatics (2018).

1.2 Installation

Decompress Knoto-ID:

```
$ tar -xf Knoto-ID-<version>.tar.gz
```

Go in the Knoto-ID directory

```
$ cd Knoto-ID-<version>/
```

Knoto-ID is organized into four directories:

- `bin/` : executables `polynomial_invariant` , `knotted_core` and `convert_diagram` .
- `scripts/` : scripts `plot_projection_map.R` , `plot_knotted_core.R` and `plot_diagram.R` .

- `doc/` : documentation (this document).
- `examples/` : input files used in this document.

To use the scripts `pdb_to_xyz.R` , `plot_3D_curve.R` , `plot_projection_map.R` , `plot_knotted_core.R` and `plot_diagram.R` , `R` (version>=3.1) [9] must be installed along with the following packages

- `optparse` [10].
- `ggplot2` (version>=2.2.0) [11].
- `RColorBrewer` [12].
- `reshape2` [13].
- `geometry` [14].
- `geosphere` [15].
- `rgl` [16] (only required for `plot_projection_map.R` with option `--output-3D` and `plot_3D_curve.R`).
- `rmarkdown` [17] (only required for `plot_projection_map.R` with option `--output-3D` and `plot_3D_curve.R`).
- `Rpdb` [18] (only required for `pdb_to_xyz.R`).

These package can installed by entering the following command in `R`

```
install.packages(c("optparse","ggplot2","RColorBrewer","reshape2",
                   "geometry","geosphere","rgl","rmarkdown","Rpdb"))
```

In addition, to use the webGL output format (`plot_projection_map.R` with option `--output-3D` and `plot_3D_curve.R`), `pandoc` [19] must also be installed.

Tested configurations:

- Mac OS X 10.12.6: R 3.3.2, optparse 1.4.4, ggplot2 2.2.1, RColorBrewer 1.1-2, reshape2 1.4.2, geometry 0.3-6, geosphere 1.5-7, rgl 0.98.1, rmarkdown 1.9, Rpdb 2.3, pandoc 2.1.1.
- Mac OS X 10.13.3: R 3.4.3, optparse 1.4.4, ggplot2 2.2.1, RColorBrewer 1.1-2, reshape2 1.4.3, geometry 0.3-6, geosphere 1.5-7, rgl 0.99.9, rmarkdown 1.8, Rpdb 2.3, pandoc 1.12.4.2.
- Ubuntu 16.04.3 LTS: R 3.2.3, optparse 1.4.4, ggplot2 2.2.1, RColorBrewer 1.1-2, reshape2 1.4.3, geometry 0.3-6, geosphere 1.5-7, rgl 0.99.9, Rpdb 2.3, pandoc 1.16.0.2.
- Fedora 25: R 3.4.2, optparse 1.4.4, ggplot2 2.2.1, RColorBrewer 1.1-2, reshape2 1.4.3, geometry 0.3-6, geosphere 1.5-7, rgl 0.99.9, rmarkdown 1.8, Rpdb 2.3, pandoc 1.17.0.3.
- Windows 10: R 3.4.3, optparse 1.4.4, ggplot2 2.2.1, RColorBrewer 1.1-2, reshape2 1.4.3, geometry 0.3-6, geosphere 1.5-8, rgl 0.99.9, rmarkdown 1.9, Rpdb 2.3, pandoc 2.1.2.

1.3 Windows users

This user guide was written for Linux and Mac OS X users. However, *Knoto-ID* can also be run on Windows using the *Windows PowerShell* (preferred) or the *Windows Command Prompt*. If you use *Knoto-ID* on Windows, please read the following comments:

- **`pandoc` installation.** `pandoc` should not only be available on the command line (e.g. `pandoc --version` should return information on the installed version) but also be found by `R`. To check if `R` can find `pandoc`, start `R` and type

  ```
  library(rmarkdown)
  pandoc_available()
  ```

  It should return `TRUE` . When testing *Knoto-ID* on Windows 10, we noticed that `R` was able to find `pandoc` only when `pandoc` was installed for all users.

- **Line continuation symbol.** In this document, we use the symbol `\` to break lines of code that should be entered as a single line. This symbol is properly interpreted in Linux and Mac OS X, but not in Windows. Windows users should not type this symbol but should manually concatenate consecutive lines separated by `\`. For example:

```
$ bin/polynomial_invariant --projection="1,0,0" \
  examples/3_1R.xyz
```

  should be entered as

```
$ bin/polynomial_invariant --projection="1,0,0" examples/3_1R.xyz
```

- `R` **scripts.** On Windows, `R` scripts cannot be executed directly but must be passed as an argument to the `Rscript` executable distributed with `R`. If it is not on the path, first locate the `Rscript` executable. For example:

```
C:\Program Files\R\R-3.4.3\bin\x64\Rscript.exe
```

  To run examples using `R` scripts in this document, insert the full path to `Rscript` at the beginning of the line. For example, to run

```
$ scripts/plot_diagram.R --output=out.pdf diagram.xyz
```

  type

```
$ C:\'Program Files\R\R-3.4.3\bin\Rscript.exe' scripts/plot_diagram.R --output=out.pdf diagram.xyz
```

  Please note that if the path contains space characters, as in this example, it must be enclosed between quotes.

- **Paths.** In Windows, paths are written using backslashes `\`, while Linux and Mac OS X (and this document) use forward slashes `/`. For example, for Linux and Mac OS X

```
$ bin/polynomial_invariant --projection="1,0,0" examples/3_1R.xyz
```

  and for windows

```
$ bin\polynomial_invariant --projection="1,0,0" examples\3_1R.xyz
```

  The *Windows PowerShell* accepts both notation but the *Command Prompt* does not.

- **Line endings.** Windows uses different line endings for text files than Linux and Mac OS X. Windows uses carriage return and line feed `\r \n` while Linux and Mac OS X use only line feed `\n`. This is not a problem for *Knoto-ID* as it can read files with both line endings. However, all text files distributed with *Knoto-ID* use Linux/Mac OS X line endings and may not be properly displayed in Windows (e.g. *Notepad*). Please use a text editor that supports both line endings (such as atom[1], notepad++[2], emacs[3], vim[4], ...)

1.4 Copyright

1.5 Licensing

[1] https://atom.io/
[2] https://notepad-plus-plus.org/
[3] https://www.gnu.org/software/emacs/
[4] https://www.vim.org/

2 Tutorial

2.1 Polynomial invariants

2.1.1 Knotoids

Knotoids on the sphere (Jones polynomial for knotoids). The program `polynomial_invariant` is used to evaluate the polynomial invariant of a piecewise linear curve. For knotoids on the sphere, the polynomial invariant is the Jones polynomial for knotoids. To print a list of all possible options

```
$ bin/polynomial_invariant --help
```

By default, the input curve is considered as open and the method of analysis is the knotoids approach. To evaluate the the polynomial invariant of the open curve given in the file `examples/3_1R.xyz` [5]:

```
$ bin/polynomial_invariant examples/3_1R.xyz
```

When `polynomial_invariant` starts, it outputs information on its progression to standard error:

```
seed: 1522146804
polynomial invariant: Jones polynomial for knotoids.
Loading input curve
3D curve has 112 vertices
projection: 0.504062,0.753096,0.42281
Simplifying 3D curve
3D curve has 8 vertices
Evaluating diagram
diagram has 3 crossings
Simplifying diagram
diagram has 3 crossings
Simplifying diagram with random Reidemeister moves III (max 100000 moves)
diagram has 3 crossings
Final simplifying diagram
diagram has 3 crossings
```

After completion, the polynomial invariant (Jones polynomial for knotoids) is written to standard output:

```
Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

The exact output may change depending on the seed used to initialize the random number generator. In particular, the polynomial invariant may be different as the polynomial invariant of an open curve depends on the choice of projection direction. To obtain reproducible results, the seed can be set with `--seed`. A specific projection direction can also be set using `--projection`:

```
$ bin/polynomial_invariant --projection="1,0,0" examples/3_1R.xyz
```

Knotoids on the plane (Turaev loop bracket). By default, the polynomial invariant is evaluated after projecting the input curve on a sphere[6]. To project the curve onto a plane instead, use option `--planar`. The polynomial invariant used in this case is the Turaev loop bracket polynomial:

```
$ bin/polynomial_invariant --planar examples/3_1R.xyz
```

The polynomial will be written to standard output:

```
Polynomial:  - A^(-16) + A^(-12) - A^(-8) - A^(-6)*v
```

Knotoid diagram. To evaluate the polynomial invariant, the input curve is first projected (on a sphere or on a plane) to obtain a knotoid diagram. To save this diagram, specify the output file using option `--output-diagram`. To output to standard output instead, use the special filename "stdout":

[5] in xyz file format, see section "3.5.1 Piecewise linear curve (xyz)" for more information on the file format.
[6] see section "4 Theory" for more details.

```
$ bin/polynomial_invariant --output-diagram=stdout examples/3_1R.xyz
```

This will output both the PD code for the knotoid diagram[7] and the polynomial invariant to standard output:

```
PD[
X[3,1,4,0],
X[1,5,2,4],
X[5,3,6,2]
];
Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

In this case (knotoids on a sphere) the polynomial invariant is the Jones polynomial for knotoids. Note that the polynomial invariant can also be written to file using the option `--output`. For example

```
$ bin/polynomial_invariant --output-diagram=diagram.txt --output=polynomial.txt examples/3_1R.xyz
```

will write the polynomial invariant to file `polynomial.txt` and the knotoid diagram to file `diagram.txt`
Option `--output-diagram-format` can be used to specify the output format for the knotoid diagram:

```
$ bin/polynomial_invariant --output-diagram=stdout --output-diagram-format=gauss examples/3_1R.xyz
```

will output the extended Gauss code for the knotoid diagram[8] and the polynomial invariant to standard output:

```
-1 2 -3 1 -2 3 +++
Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

Drawing knotoid diagrams To draw knotoid diagrams such as those obtained with `polynomial_invariant` using option `--output-diagram`, one should use the program `convert_diagram` to convert the diagram to xyz format for knot(oid) diagram[9] and the script `scripts/plot_diagram.R` [10] to draw the diagram.
For example, to draw the knotoid diagram `examples/3_1L_diagram_open_PD.txt`:

```
$ bin/convert_diagram --input-format=pd --output-format=xyz --output=diagram.xyz \
  examples/3_1L_diagram_open_PD.txt
```

Options `--input-format=pd` and `--output-format=xyz` must be used to specify input and output format respectively. The resulting diagram in xyz format can be then used as input for the script `scripts/plot_diagram.R`

```
$ scripts/plot_diagram.R --output=diagram.pdf diagram.xyz
```

The resulting diagram is shown in Figure 1 (left). Option `--labels` can be used to add arc labels (Figure 1 right):

```
$ scripts/plot_diagram.R --output=diagram.pdf --labels diagram.xyz
```

Option `--line-width` can be used to change the thickness of curve and improve readability (Figure 2):

```
$ scripts/plot_diagram.R --output=diagram.pdf --line-width=5 diagram.xyz
```

To draw a planar knotoid diagram (i.e. to correctly place arcs marked as touching the outside region), the option `--planar` must be used. For example, to draw the planar knotoid diagram
`examples/3_1L_diagram_open_planar_PD.txt`:

```
$ bin/convert_diagram --input-format=pd --output-format=xyz --output=diagram.xyz \
  --planar examples/3_1L_diagram_open_planar_PD.txt
$ scripts/plot_diagram.R --output=diagram.pdf diagram.xyz
```

Note that using option `--planar` with an input diagram that does not contain inside/outside information will result in an error.

[7] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[8] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.
[9] see section "3.5.10 xyz format for knot(oid) diagrams" for more information on the file format.
[10] To use this script, `R` [9] must be installed with packages `optparse` [10] and `ggplot2` [11].

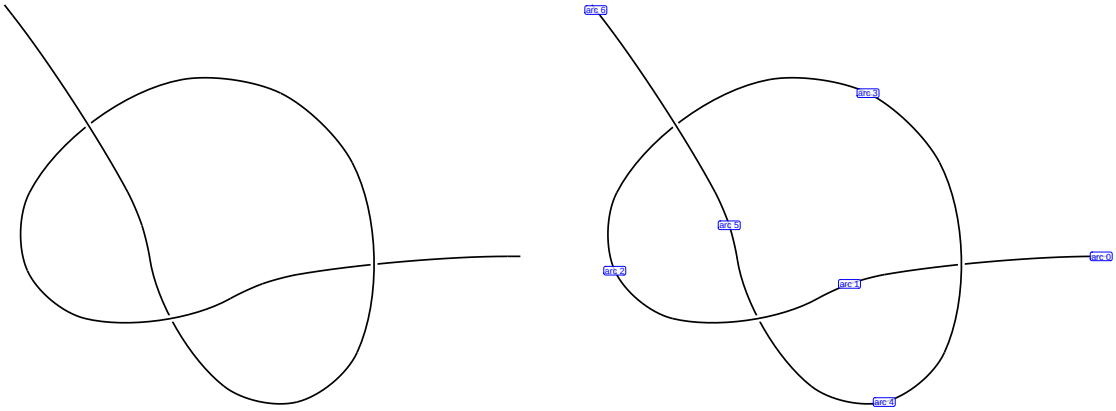

Figure 1: Graphical representation of the knotoid diagram `examples/3_1L_diagram_open_PD.txt` without (left) and with (right) arc labels.

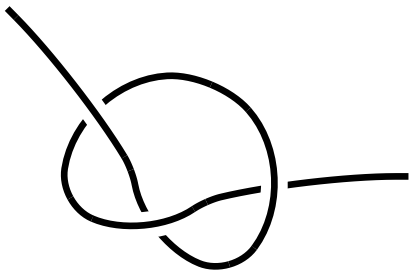

Figure 2: Graphical representation of the knotoid diagram `examples/3_1L_diagram_open_PD.txt` using `--line-width=5` .

To use a diagram in extended Gauss code format as input, use `--input-format=gauss` :

```
$ bin/convert_diagram --input-format=gauss --output-format=xyz --output=diagram.xyz \
  examples/3_1L_diagram_planar_Gauss.txt
$ scripts/plot_diagram.R --output=diagram.pdf diagram.xyz
```

Note that the extended Gauss format does not distinguish knots (closed diagrams) from knot-type knotoids (open diagrams). Option `--closure-method` must be used to specify whether the diagram is open (`--closure-method=open`) or closed (`--closure-method=direct` or `rays`).

`convert_diagram` also accepts piecewise linear curve in xyz format as input. To draw the knotoid diagram obtained by projecting the curve `examples/3_1R_supercoiled.xyz` along the direction $(0, 1, 0)$, change `--input-format` to `xyz` :

```
$ bin/convert_diagram --input-format=xyz --output-format=xyz --output=diagram.xyz \
  --planar --projection="0,1,0" examples/3_1R_supercoiled.xyz
$ scripts/plot_diagram.R --output=diagram.pdf diagram.xyz
```

The resulting `diagram.pdf` is shown in Figure 3.

While `convert_diagram` usually produces reasonably good results for knot diagrams, it may also produce unbalanced

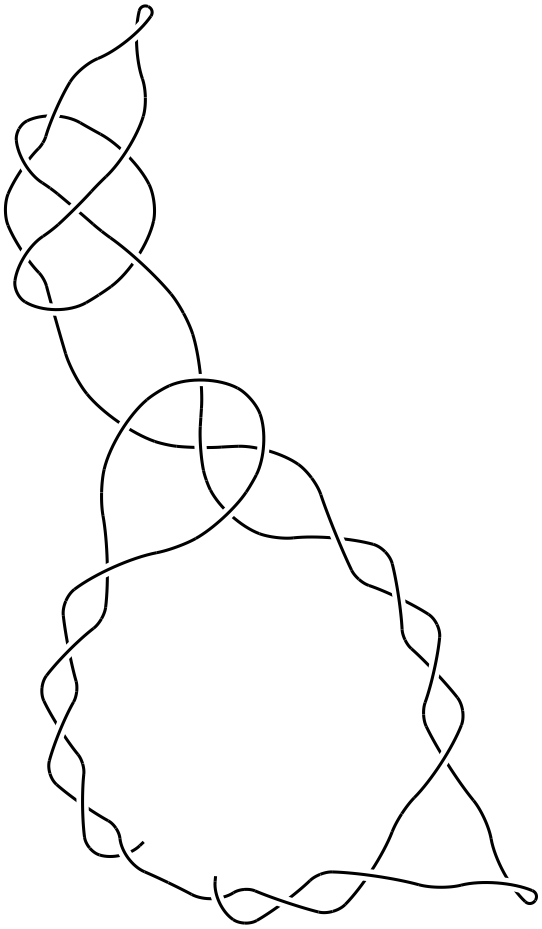

Figure 3: Graphical representation of the planar knotoid diagram obtained by projecting the curve `examples/3_1R_supercoiled.xyz` along the direction $(0, 1, 0)$.

diagrams, with arc lenght differing by several orders of magnitude. This problem is particularly important when working with planar diagrams (using `--planar`), in particular when the diagram is enclosed within an external loop, as in the knotoid diagram given in `examples/Unbalanced_diagram_open.txt`.

```
$ bin/convert_diagram --input-format=pd --output-format=xyz --output=diagram.xyz \
  --planar examples/Unbalanced_diagram_open.txt
```

Running this command produces an error and writes the following message to standard error:

```
*******************************************************
ERROR: Unbalanced circle packing.
Use option --nb-iterations-relaxation to relax the knot(oid) diagram.
Or use option --force to save the unbalanced diagram.
*******************************************************
```

To ignore the error and write the output file `diagram.xyz`, use `--force`:

```
$ bin/convert_diagram --input-format=pd --output-format=xyz --output=diagram.xyz \
  --planar --force examples/Unbalanced_diagram_open.txt
```

A warning is still written to standard error

```
*******************************************************
WARNING: Unbalanced circle packing.
Use option --nb-iterations-relaxation to relax the knot(oid) diagram.
*******************************************************
```

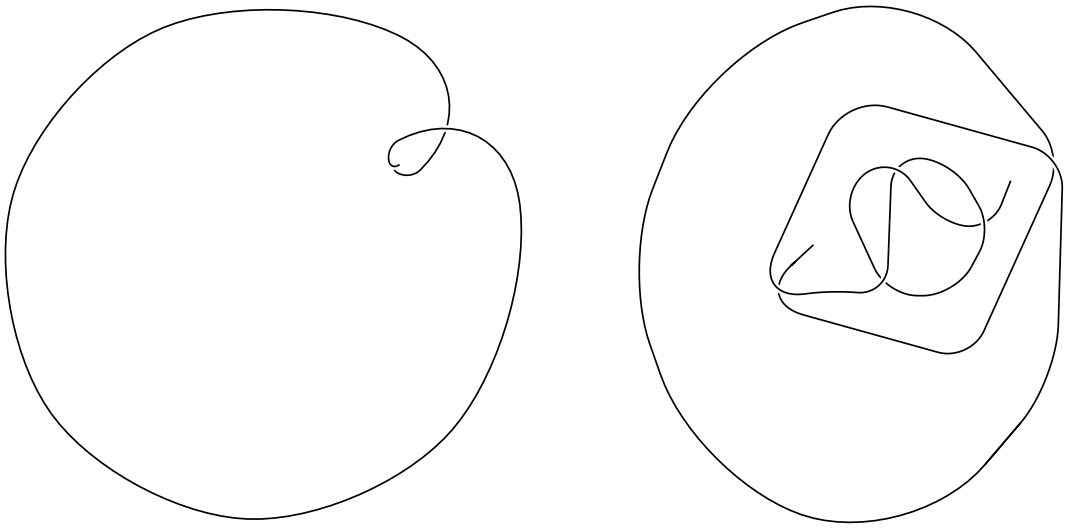

Figure 4: Graphical representation of the knotoid diagram `examples/Unbalanced_diagram_open.txt` without (left) and with (right) relaxation.

but the output file `diagram.xyz` is created and the script `plot_diagram.R` can then be used to draw the knotoid diagram:

```
$ scripts/plot_diagram.R --output=diagram.pdf diagram.xyz
```

The resulting diagram is shown in Figure 4 left. While the knotoid diagram has 5 crossings, this unbalanced graphical representation suggests that it has only one crossing, and one endpoint seems to be touching one arc. In reality, all "missing" crossings are in the region of the endpoints, but to see them one would have to zoom strongly on this region (and decrease the thickness of the lines).
To improve the readability of the diagram, `convert_diagram` can try to "relax" the diagram using simulated annealing. To use it, specify a positive number of iterions with `--nb-iterations-relaxation`:

```
$ bin/convert_diagram --input-format=pd --output-format=xyz --output=diagram.xyz \
  --planar  --nb-iterations-relaxation=10000 examples/Unbalanced_diagram_open.txt
$ scripts/plot_diagram.R --output=diagram.pdf diagram.xyz
```

The resulting diagram is shown in Figure 4 right. Admittedly, the diagram is not very pleasing aesthetically, but the topology of the knotoid diagram is clearly visible.

Multiple projections. The polynomial invariant of an open curve depends on the choice of projection direction. To avoid this dependency on the projection direction, one can evaluate the polynomial invariant for multiple projection directions, and consider the distribution of the polynomials. This can be done with the option `--nb-projections`.

```
$ bin/polynomial_invariant --nb-projections=100 examples/3_1R.xyz
```

The resulting distribution of polynomials will be written to standard output (or in the file specified with `--output`):

```
#frequency  polynomial
0.92     - A^(-16) + A^(-12) + A^(-4)
0.08     - A^(-10) + A^(-6) + A^(-4)
```

The output is in tab separated format with two columns[11]. The second column contains the polynomial and the first column contains the fraction of projections that gave the corresponding polynomial. The exact output may change since the projections are chosen randomly[12].
Instead of using random projections, it is possible to use a predefined list of projections with the option `--projections-list`. This list must contain at least three columns corresponding to x, y and z coordinates of the projection directions, and optionally a fourth column corresponding to a weight[13]. To evaluate the distribution of polynomials obtained with the

[11] see section "3.5.4 Distribution of polynomials" for more information on the file format.
[12] with uniform distribution on the surface of the sphere.
[13] see section "3.5.6 List of projections" for more information on the file format.

list of projections given in file `examples/projections_list_100.txt` :

```
$ bin/polynomial_invariant --projections-list=examples/projections_list_100.txt examples/3_1R.xyz
```

Note that when the list of projections contains a fourth column with weights, the fraction of projections giving a specific polynomial is replaced by a weighted fraction.
It is also possible to get the corresponding knotoids diagrams when using multiple projections (only 10 projections are used here to keep the output short):

```
$ bin/polynomial_invariant --output-diagram=diagrams.txt --nb-projection=10 examples/3_1R.xyz
```

The resulting file `diagrams.txt` is in tab separated format with 5 columns[14]. Each row corresponds to one projection, with x,y,z coordinates of the projection direction in columns 1 to 3, polynomial in column 4 and PD code of the knotoid diagram in column 5:

```
$ cat diagrams.txt
```

```
#projection.x  projection.y  projection.z  polynomial                  PD_code
0.483595       -0.85801      0.173072      - A^(-16) + A^(-12) + A^(-4)  PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
0.521928       -0.578265     0.627058      - A^(-16) + A^(-12) + A^(-4)  PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
0.898137       0.241849      0.367231      - A^(-16) + A^(-12) + A^(-4)  PD[X[7,0,8,1],X[4,2,5,1],X[2,6,3,5],X[6,4,7,3]];
-0.624015      0.596182      0.505146      - A^(-10) + A^(-6) + A^(-4)   PD[X[0,3,1,2],X[3,2,4,1]];
0.134038       -0.988518     -0.0697575    - A^(-16) + A^(-12) + A^(-4)  PD[X[5,1,6,0],X[1,5,2,4],X[2,8,3,7],X[6,4,7,3]];
0.377609       0.924851      0.0454074     - A^(-10) + A^(-6) + A^(-4)   PD[X[3,1,4,0],X[5,2,6,1],X[2,5,3,4]];
-0.720043      0.186829      0.668306      - A^(-16) + A^(-12) + A^(-4)  PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
-0.767046      -0.420252     -0.484798     - A^(-16) + A^(-12) + A^(-4)  PD[X[0,4,1,3],X[4,2,5,1],X[2,6,3,5]];
0.290762       0.956792      -0.00267033   - A^(-16) + A^(-12) + A^(-4)  PD[X[0,6,1,5],X[4,2,5,1],X[7,3,8,2],X[3,7,4,6]];
0.0476456      0.685232      -0.726764     - A^(-16) + A^(-12) + A^(-4)  PD[X[0,4,1,3],X[4,2,5,1],X[2,6,3,5]];
```

Note that PD codes can be replaced by extended Gauss codes using option `--output-diagram-format=gauss` . This file can then be used to generate a projection map similar to those presented in [3, 20]. A simple `R` script [9] is included with *Knoto-ID* to plot the projection map, using voronoi tessellation on the sphere[15]:

```
$ scripts/plot_projection_map.R --output=projection_map.png diagrams.txt
```

The resulting projection map is shown in Figure 5, together with a more detailed projection map obtained with 10000 projections. The script `plot_projection_map.R` can also be used to create a 3D globe (in webGL format [8]) with option `--output-3D` :

```
$ scripts/plot_projection_map.R --output-3D=projection_map.html diagrams.txt
```

The resulting 3D projection map can then be opened in a web browser. Using option `--curve-3D` , a piecewise linear curve can be added next to the 3D globe:

```
$ scripts/plot_projection_map.R --output-3D=projection_map.html \
  --curve-3D=examples/3_1R.xyz diagrams.txt
```

A screenshot is shown in Figure 6.

Using a knotoid diagram as input. By default, `polynomial_invariant` takes a 3D piecewise linear curve as input. However, it is possible to use a knotoid diagram as input using the option `--input-format=pd`[16] or `--input-format=gauss`[17]. To load the PD code for a knotoid diagram:

```
$ bin/polynomial_invariant --input-format=pd examples/3_1L_diagram_open_PD.txt
```

Note that even if the input diagram contains inside/outside information[18], it will be considered as projected on a sphere:

```
$ bin/polynomial_invariant --input-format=pd examples/3_1L_diagram_open_planar_PD.txt
```

[14] see section "3.5.5 List of projections with polynomials and knot(oid) diagrams" for more information on the file format.
[15] To use this script, `R` [9] must be installed with packages `optparse` [10], `ggplot2` [11], `RColorBrewer` [12], `geometry` [14], `geosphere` [15] and `rgl` [16]. To use option `--output-3D` , pandoc [19] must also be installed.
[16] in KnotTheory file format, see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[17] in KnotTheory file format, see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.
[18] i.e. `r[]` , see section "3.5.2 Knot(oid) diagrams (PD code)" for more details.

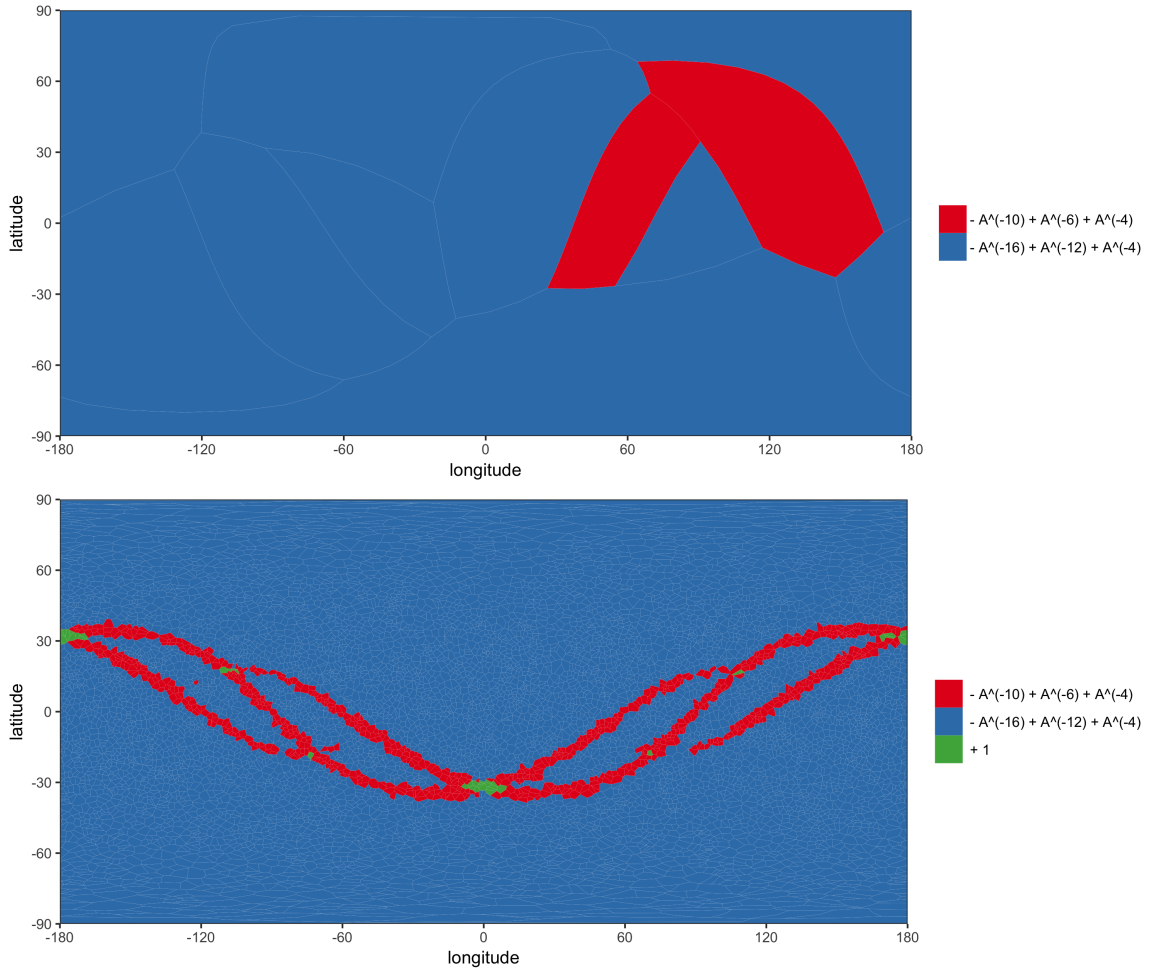

Figure 5: Projection maps generated with the script `plot_projection_map.R` using 10 (top) and 10000 projections (bottom). For a projection direction (x, y, z), longitude (λ) and latitude (δ) are defined as $x = \cos(\delta)\cos(\lambda)$, $y = \cos(\delta)\sin(\lambda)$, $z = \sin(\delta)$. Color corresponds to polynomial. Since the voronoi tesselation is performed on the sphere, voronoi facets are separated by arcs of great circles, which do not correspond to straight line after equirectangular projection.

To interpret the diagram as projected on a plane, one need to pass the option `--planar`:

```
$ bin/polynomial_invariant --input-format=pd --planar examples/3_1L_diagram_open_planar_PD.txt
```

Using option `--planar` with an input diagram that does not contain inside/outside information will result in an error.
To load a knotoid diagram in extended Gauss code format:

```
$ bin/polynomial_invariant --input-format=gauss examples/3_1L_diagram_Gauss.txt
```

The extended Gauss format does not distinguish knots (closed diagrams) from knot-type knotoids (open diagrams). Option `--closure-method` is used to specify whether the diagram is open (`--closure-method=open`) or closed (`--closure-method=direct` or `rays`).
By default, even if the input diagram contain inside/outside information[19], the diagram will be considered as projected on a sphere:

```
$ bin/polynomial_invariant --input-format=gauss examples/3_1L_diagram_planar_Gauss.txt
```

To interpret the diagram as projected on a plane, one need to pass the option `--planar`:

```
$ bin/polynomial_invariant --input-format=gauss --planar examples/3_1L_diagram_planar_Gauss.txt
```

[19] i.e. `r[]` , see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more details.

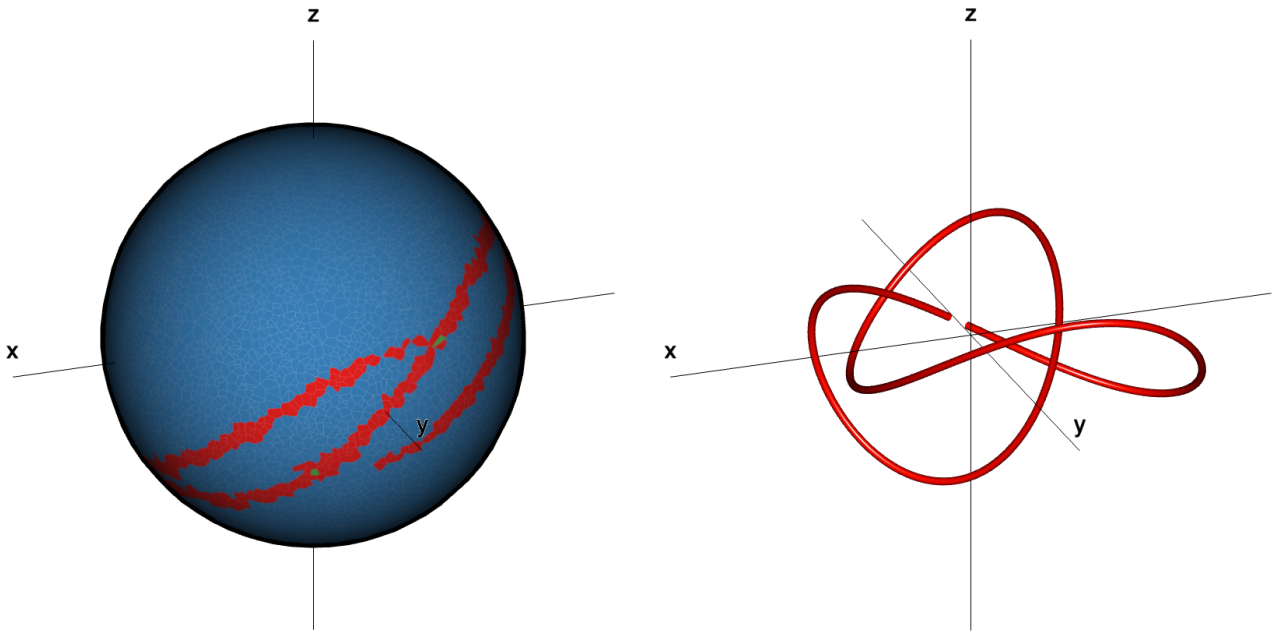

Figure 6: 3D projection map and piecewise linear curve generated with the script `plot_projection_map.R` using 10000 projections. Color corresponds to polynomial.

When loading a diagram in PD code or an extended Gauss code format, `polynomial_invariant` checks that the diagram is valid using the method proposed by Vijayan and Wigderson [21]. For instance, when loading the invalid diagram (non planar) from file `examples/Invalid_diagram_open_PD.txt`

```
$ bin/polynomial_invariant --input-format=pd examples/Invalid_diagram_open_PD.txt
```

`polynomial_invariant` fails with the following error message

```
Loading input diagram
************************************
ERROR knot(oid) diagram not valid!
************************************
```

Finally, note that all options concerning the projection (`--projection` , `--projections-list` and `--nb-projections`) are ignored when using a knot(oid) diagram as input while option `--closure-method` is ignored when using PD code format (`--input-format=pd`).

2.1.2 Knots

To evaluate the classical Jones polynomial of a closed curve, one has to specify what method should be used to close the input curve. Two closure methods are currently implemented: "direct" and "rays"[20]. The closure method can be specified with the option `--closure-method=direct` or `--closure-method=rays` . Note that the option `--closure-method` accepts a third argurment `--closure-method=open` (selected by default) which leaves the curve open (discussed in section "2.1.1 Knotoids").
For example:

```
$ bin/polynomial_invariant --closure-method=direct examples/3_1R.xyz
```

will evaluate the polynomial invariant (classical Jones polynomial) of the closed curve obtained by adding a straight line from last to first point of the input curve, and output the result to standard output:

```
Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

Using the `rays` closure method, which closes the curve by extending two parallel rays along the projection direction

[20] see section "4.3 Knotoids and curves in space" for more details.

from each of the endpoints of the curve and connects them once they pierce the sphere that encloses the curve[21]:

```
$ bin/polynomial_invariant --closure-method=rays examples/3_1R.xyz
```

gives the following polynomial

```
Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

When using `rays` closure method, the resulting polynomial depends on the projection direction, which defines the direction of the two rays used to close the curve. To evaluate the distribution of polynomials obtained on multiple random projection directions[22]:

```
$ bin/polynomial_invariant --closure-method=rays --nb-projections=100 examples/3_1R.xyz
```

The resulting distribution is written to standard output:

```
#frequency  polynomial
0.93        - A^(-16) + A^(-12) + A^(-4)
0.07        + 1
```

Note that the combination of the `rays` closure method with multiple directions of projections corresponds to the uniform (or stochastic) closure technique.
All options discussed previously in the context of knotoids can also be used with knots, except `--planar` which is ignored for closed curve.

2.1.3 Mapping polynomials to knot(oid) names.

The directory `examples/` contains two files mapping polynomials to knot and knotoid names[23]. File `examples/knotoid_names.txt` should be used when working with knotoids (`--closure-method=open`), while file `examples/knot_names.txt` [24] should be used with knots (`--closure-method=direct` or `rays`). These files can be loaded using the option `--names-db` . When using `--names-db` with knotoids

```
$ bin/polynomial_invariant --names-db=examples/knotoid_names.txt examples/3_1R.xyz
```

The standard output will have an additional field with "Knotoid type":

```
Knotoid type: k3.1    Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

Please note that the file `knotoid_names.txt` is a work in progress and is given only as an example.
Similarly when using a closed curve (knot) and `--names-db=examples/knot_names.txt`

```
$ bin/polynomial_invariant --closure-method=direct --names-db=examples/knot_names.txt examples/3_1R.xyz
```

The standard output will have an additional field with "Knot type":

```
Knot type: 3_1R    Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

The polynomial invariants used here are not complete invariants, meaning that some knot types are not distinguished. As a consequence, multiple lines in the file `examples/knot_names.txt` have the same polynomial. When this file is loaded, all knot types with the same polynomial are concatenated (separated with a `|`). For example, 5_2R has the same polynomial as 11n_57L and 12n_0475L. Evaluating the classical Jones polynomial of `examples/5_2R.xyz` using direct closure

```
$ bin/polynomial_invariant --names-db=examples/knot_names.txt examples/5_2R.xyz --closure-method=direct
```

will generate the following in the standard output

```
Knot type: 5_2R|11n_57L|12n_0475L  Polynomial:  - A^(-24) + A^(-20) - A^(-16) + 2*A^(-12) - A^(-8) + A^(-4)
```

which means that 5_2R, 11n_57L and 12n_0475L were found with the same polynomial in the file

[21] see section "4.3 Knotoids and curves in space" for more details.
[22] with uniform distribution on the surface of the sphere.
[23] see section "3.5.7 List of knot(oid) names" for more information on the file format.
[24] This file was created using "The Rolfsen Knot Table" from The Knot Atlas (katlas.org).

`examples/knot_names.txt`. In some cases, knowing the number of crossings in the diagram may help to determine the knot type. This information is given in standard error along with other information:

```
seed: 1496327362
polynomial invariant: classical Jones polynomial.
Loading names data base
Loading input curve
3D curve has 111 vertices
projection: -0.972927,-0.122399,-0.196038
Simplifying 3D curve
3D curve has 8 vertices
Evaluating diagram
diagram has 10 crossings
Simplifying diagram
diagram has 8 crossings
Simplifying diagram with random reidemeister moves III (max 100000 moves)
diagram has 6 crossings
Final simplifying diagram
diagram has 6 crossings
```

The last line is of particular interest as it contain the number of crossings in the knot diagram obtained after all simplifications. Since this diagram has 6 crossings and both 11n_57L and 12n_0475L have more crossings, it can only correspond to 5_2R. Note that in general the number of crossing obtained after simplification by `polynomial_invariant` is only an upper bound to the minimal number of crossing.
Finally, note that if a polynomial invariant is not found in the knot(oid) names database specified with `--names-db`, it will take the special knot type `UNKNOWN`.

2.2 Knotted core

2.2.1 Using open curves as input

The program `knotted_core` evaluates the knotted core of a piecewise linear curve. To print a list of all possible options

```
$ bin/knotted_core --help
```

Similarly to `polynomial_invariant`, the input curve is considered as open by default. For example

```
$ bin/knotted_core --output=knotted_core.txt examples/3_1R_supercoiled.xyz
```

will search for the knotted core of the (open) input curve `examples/3_1R_supercoiled.xyz`. The resulting knotted core is written to the file `knotted_core.txt`:

```
$ cat knotted_core.txt
```

```
#index_first  index_last  length  frequency  polynomial
23            52          30      0.55       - A^(-16) + A^(-12) + A^(-4)
```

The output is in a simple tab separated format with each row corresponding to a subchain (knotted core). In this example the knotted core is the subchain obtained by keeping points of the input curve `examples/3_1R_supercoiled.xyz` with indices 23 to 52 (zero based indexing). The dominant polynomial of the knotted core is $-A^{-16} + A^{-12} + A^{-4}$ and it appeared in 55% of the projections[25]. Note that the exact output may change since the dominant knotoid type of each subchain is evaluated using random sampling of a finite number projections (20 by default). In particular, the output could contain more than one line if more than one subchain have the same minimal length and same polynomial.
The script `plot_3D_curve.R`[26] distributed with *Knoto-ID* can be used to plot the input curve given in file `examples/3_1R_supercoiled.xyz`, highlight the knotted core saved in `knotted_core.txt`, and output to webGL format [8]:

[25] see section "3.5.9 List of subchains" for more information on the file format.
[26] To use this script, `R` [9] must be installed with packages `optparse` [10] and `rgl` [16].

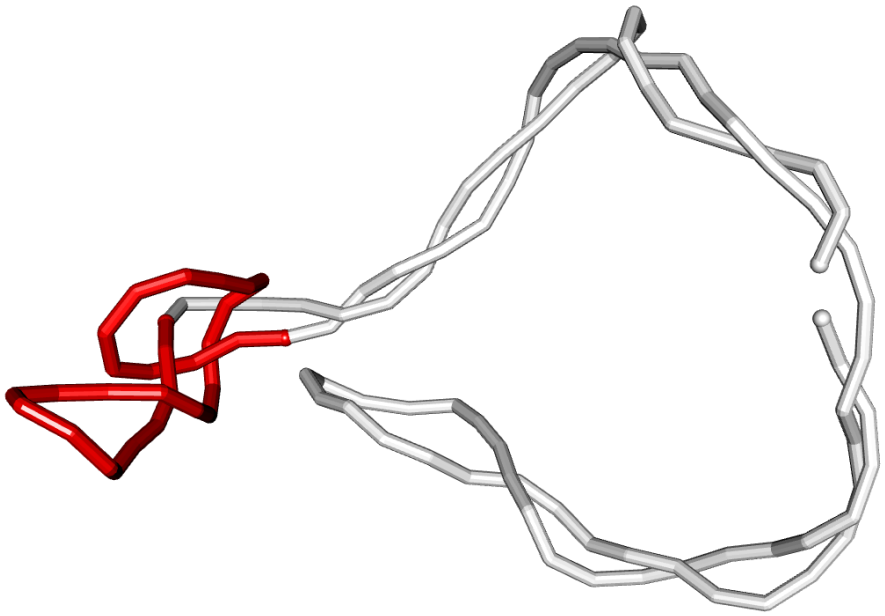

Figure 7: Piecewise linear curve given in file `examples/3_1R_supercoiled.xyz` with knotted core highlighted in red. Figure created with script `plot_3D_curve.R`

```
$ scripts/plot_3D_curve.R --subchain=knotted_core.txt --output=knotted_core.html \
  examples/3_1R_supercoiled.xyz
```

The resulting file `knotted_core.html` can then be opened in a web browser. A screenshot is shown in Figure 7.

Closure method (subchains). In this last example, the polynomial invariant of each subchain was evaluated without closing the subchain (using knotoids). To evaluate the polynomial invariant of closed subchain instead (classical Jones polynomial for knots), one has to specify a closure method with the option `--closure-method=direct` or `--closure-method=rays` [27]. To evaluate the knotted core using `rays` closure method for the subchains:

```
$ bin/knotted_core --closure-method=rays examples/3_1R_supercoiled.xyz
```

The knotted core is written to standard output:

```
#index_first  index_last  length  frequency  polynomial
23            51          29      0.65       - A^(-16) + A^(-12) + A^(-4)
```

To use `direct` closure of the subchains:

```
$ bin/knotted_core --closure-method=direct examples/3_1R_supercoiled.xyz
```

Note that with the `direct` closure method, the polynomial does not depend on the projection. Therefore only one projection is used and the dominant knot type appears with frequency 1 (100%):

```
#index_first  index_last  length  frequency  polynomial
20            49          30      1          - A^(-16) + A^(-12) + A^(-4)
```

When using open subchains (`--closure-method=open`), the polynomial invariant is evaluated after projecting the input curve on a sphere. To project on a plane, use option `--planar` :

```
$ bin/knotted_core --closure-method=open --planar examples/3_1R_supercoiled.xyz
```

Mapping polynomials to knot(oid) names. As with `polynomial_invariant` , option `--names-db` can be used to load a file mapping polynomials to knot or knotoid names

[27] see section "4.3 Knotoids and curves in space" for more details.

```
$ bin/knotted_core --names-db=examples/knotoid_names.txt examples/3_1R_supercoiled.xyz
```

The output contains an additional column `knotoid_type` (or `Knot` if subchains are closed)

```
#index_first  index_last  length  frequency  knotoid_type  polynomial
23            52          30      0.55       k3.1          - A^(-16) + A^(-12) + A^(-4)
```

File `examples/knotoid_names.txt` should be used with open subchains (`--closure-method=open` , by default). When working with closed subchains (`--closure-method=direct` or `rays`), use file `examples/knot_names.txt` instead.

Fingerprint matrices. By default, `knotted_core` outputs only the knotted core to the standard output. Option `--output` can be used to save the knotted core to a file. While searching for the knotted core, the program computes the dominant polynomial of several subchains. These can be saved to a file using `--output-search` Finally, using option `--output-all` , the dominant polynomial is computed for every subchain and saved to a file.

```
$ bin/knotted_core --output-all=all.txt --output-search=search.txt --output=knotted-core.txt \
  --names-db=examples/knotoid_names.txt examples/3_1R_supercoiled.xyz
```

The resulting files can then be used to produce a knotoid fingerprint matrix [4, 5, 3] using a simple `R` script [9] included with *Knoto-ID*

```
$ scripts/plot_knotted_core.R --output=fingerprint_matrix.png all.txt
```

Note that this script requires `R` [9] with packages `optparse` [10], `ggplot2` [11], `reshape2` [13] and `RColorBrewer` [12]. The knotted core as well as the search path can be added to the plot using options `--knotted-core` and `--search-path` (see Figure 8):

```
$ scripts/plot_knotted_core.R --output=fingerprint_matrix.png \
  --knotted-core=knotted-core.txt --search-path=search.txt all.txt
```

It is also possible to plot only the search path by replacing the file `all.txt` by `search.txt` :

```
$ scripts/plot_knotted_core.R --output=fingerprint_matrix.png \
  --knotted-core=knotted-core.txt search.txt
```

Projections. By default `knotted_core` uses 20 projections[28] to estimate the dominant knot(oid). The number of random projection can be changed with `--nb-projections` . For example, to use 100 projections:

```
$ bin/knotted_core --nb-projections=100 examples/3_1R_supercoiled.xyz
```

Alternatively, a predefined list of projections can be specified with option `--projections-list` . For example to use the list of projections given in file `examples/projections_list_100.txt` :

```
$ bin/knotted_core --projections-list=examples/projections_list_100.txt examples/3_1R_supercoiled.xyz
```

The number of projections has an impact on computational cost (proportional to the number of projections) but also on the precision of the results. Figure 9 presents fingerprint matrices of the curve `examples/3_1R_supercoiled.xyz` obtained with different number of projections. The fingerprint matrix obtained with `--nb-projections=1` is very noisy, with multiple spurious dominant knotoid types. With `--nb-projections=10` the matrix already gives a good approximation at a reasonable computational cost. With `--nb-projections=100` the level of noise is strongly decreased and the all spurious knotoid types disappear.

2.2.2 Using closed curves as input

By default, the input curve is considered as open. To close the input curve with a straight line from last to first point, use option `--cyclic-input` :

[28] randomly chosen with uniform distribution on the surface of the sphere.

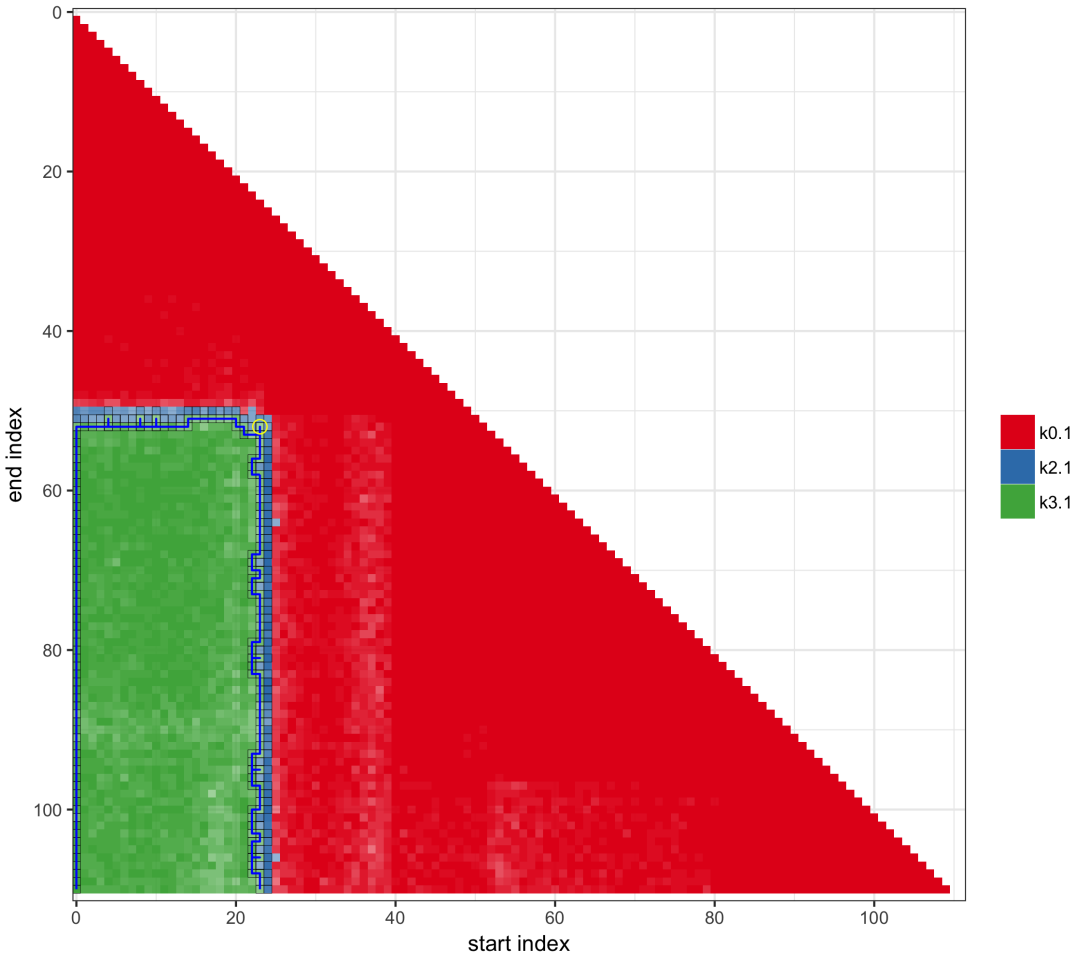

Figure 8: The knotoid fingerprint matrix of the curve `examples/3_1R_supercoiled.xyz`. Each entry in this matrix with coordinates x and y corresponds to a subchain having start index x and end index y. Color corresponds to the dominant knotoid type of the subchain and transparency corresponds to its frequency. The knotted core is shown with yellow circle. All subchains evaluated while searching for the knotted core are shown with a black border. In addition, the blue line shows consecutive subchains with same polynomial as the input curve that were found during the search.

```
$ bin/knotted_core --cyclic-input examples/3_1R_supercoiled.xyz
```

When using option `--cyclic-input`, the input curve is closed with direct closure[29]. Therefore, this option should be used with caution and one should make sure that endpoints are close enough so that direct closure does not disturb the topology of the input curve. Note that the direct closure method used to close the input curve (`--cyclic-input`) is completely independent from the choice of closure method for the subchains (`--closure-method`).
Closing the input curve modifies the set of possible subchains that are considered when searching for the knotted core. With open input curve, subchains were not allowed to contain endpoints of the input curve (except at their endpoint). When an input curve is closed, the first and last points are not endpoints anymore and can appear anywhere in the subchains (see Figure 10).
With closed input curves, one should be careful when interpreting the output of `knotted_core`. Indeed, in the output of `knotted_core` [30], subchains are defined by two points of the input curve (`index_first` and `index_last`). When the first point has an index i_1 lower than the last point i_2, the subchain consist of all points with indices: $\{i_1, i_1+1, i_1+2, \cdots, i_2\}$. However, when the first point has an index i_1 higher than the last point i_2, the subchain consist of all points with index larger or equal to the index of the first point up to the last point of the input curve, and then continue with the first point of the input curve up to i_2, i.e. for an input curve with N points (using zero-based indexing): $\{i_1, i_1+1, i_1+2, \cdots, N-1, 0, 1, 2, \cdots, i_2\}$. For example, the output

[29] closed with a straight line from last to first point of the curve.
[30] see section "3.5.9 List of subchains" for more information on the file format.

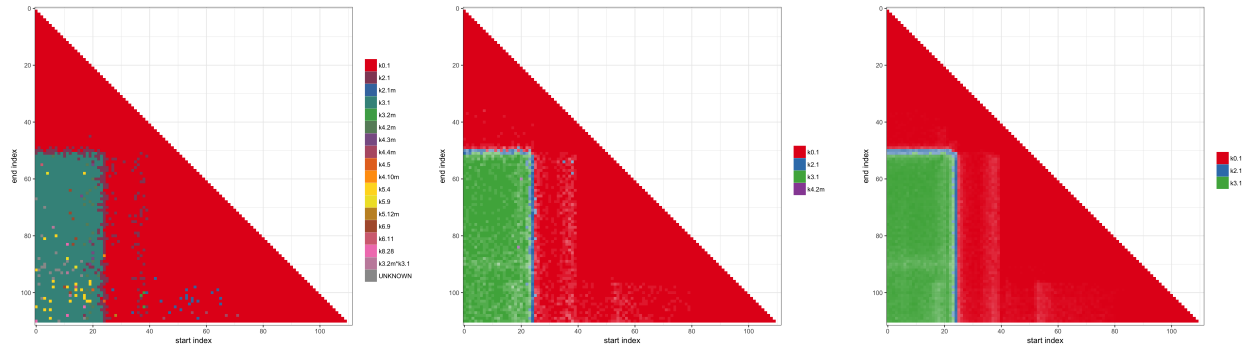

Figure 9: The knotoid fingerprint matrix of the curve `examples/3_1R_supercoiled.xyz` obtained with 1 projection (left), 10 projections (middle) and 100 projections (right).

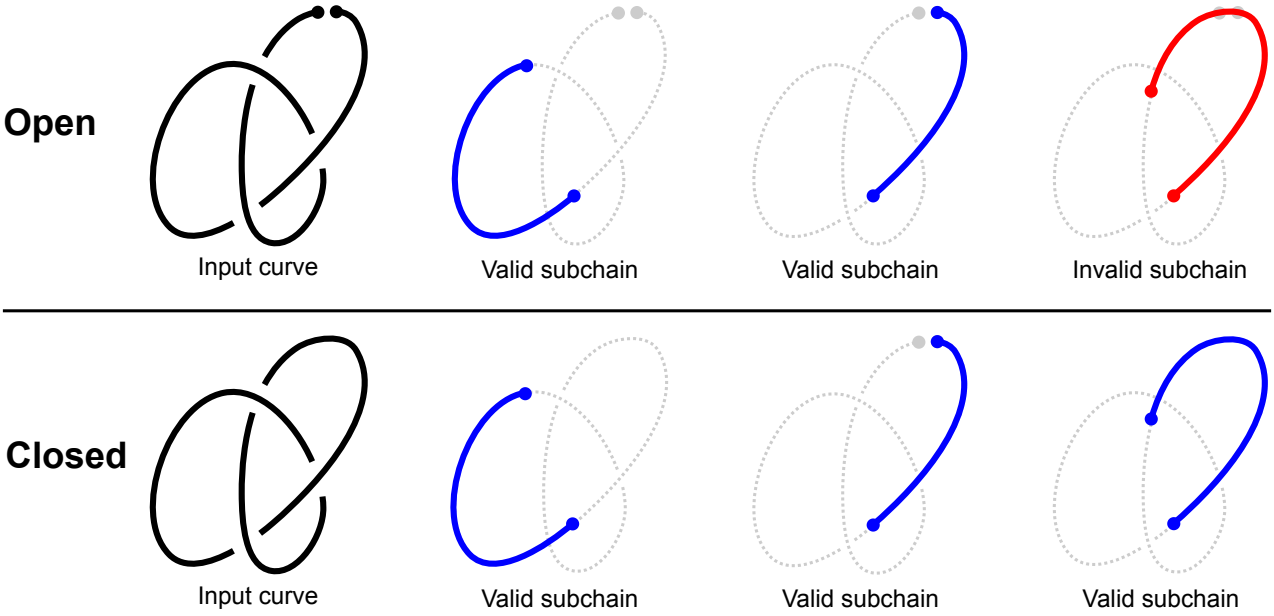

Figure 10: With open input curve (upper panel), subchains are not allowed to contain endpoints of the input curve (except at their endpoint). With closed input curve (lower panel), all subchains are allowed. Example of valid subchains are shown in blue, and invalid subchain in red.

```
#index_first  index_last  length  frequency  polynomial
3             8           6       0.5        - A^(-16) + A^(-12) + A^(-4)
```

corresponds to the subchain with points $\{3, 4, 5, 6, 7, 8\}$ of the input curve, while

```
#index_first  index_last  length  frequency  polynomial
109           3           6       0.5        - A^(-16) + A^(-12) + A^(-4)
```

corresponds to the subchain with points $\{109, 110, 0, 1, 2, 3\}$ of the input curve (which has 111 points).
All options discussed when using open input curve can also be used with closed curve. In particular, the option `--closure-method`, which defines how each subchain should be closed (or kept open), can be used independently from the choice of closure for the input curve (`--cyclic-input`). For example, to evaluate the knotted core of the closed input curve `examples/3_1R_supercoiled.xyz` based on the evaluation of the polynomial invariant of open subchains projected on the sphere (Jones polynomial for knotoids):

```
$ bin/knotted_core --output-all=all.txt --output-search=search.txt --output=knotted-core.txt \
  --names-db=examples/knotoid_names.txt --nb-projections=100 --cyclic-input \
  examples/3_1R_supercoiled.xyz
```

The resulting knotted core is saved in the file `knotted-core.txt`, while the dominant knotoid types of all possible subchains and of all subchains evaluated during the search are saved in files `all.txt` and `search.txt` respectively. The resulting files can be used to produce a disk matrix [6] with `plot_knotted_core.R` and option `--cyclic`

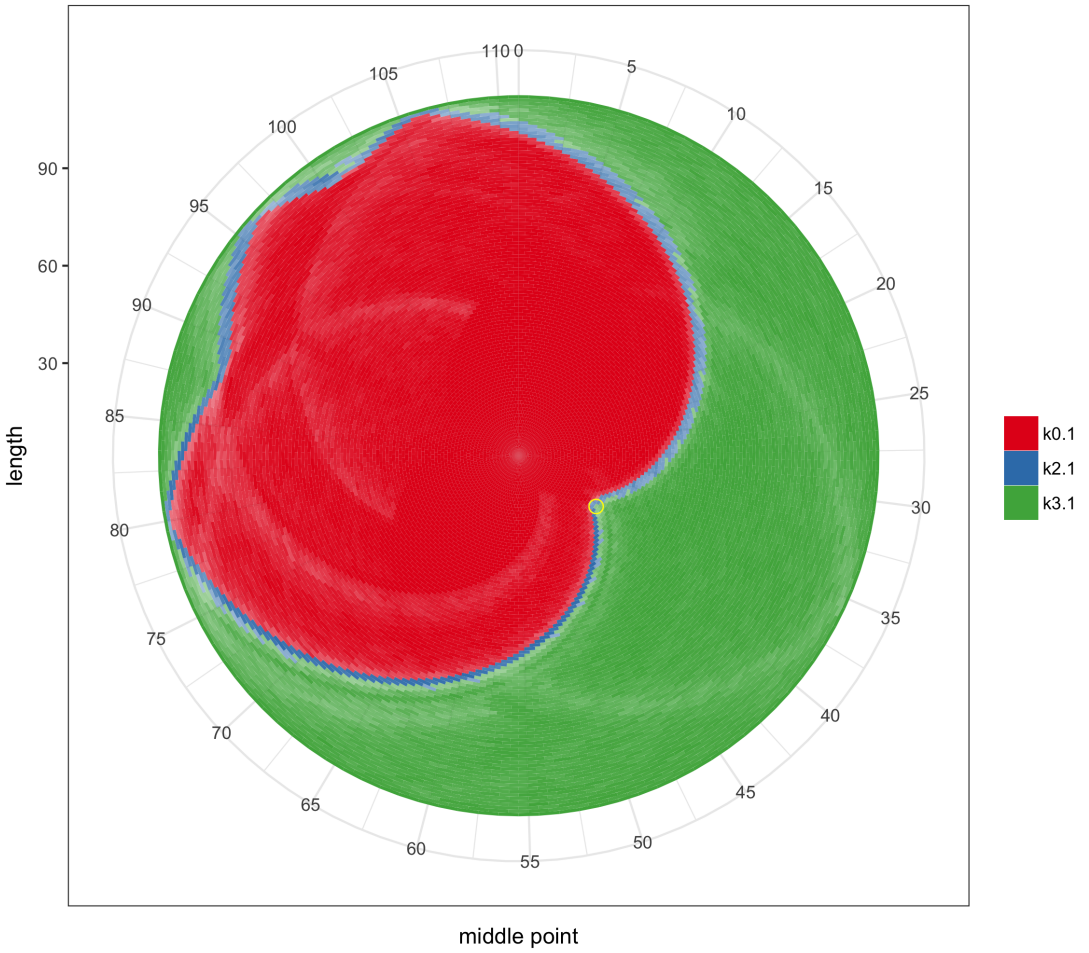

Figure 11: The disk matrix of the closed curve `examples/3_1R_supercoiled.xyz`. Radial coordinate corresponds to the length of the subchain and angular coordinate to the midpoint of the subchain. Color corresponds to dominant knotoid type of the subchain and transparency to its frequency. The knotted core is shown with yellow circle.

(see Figure 11):

```
$ scripts/plot_knotted_core.R --cyclic --output=disk_matrix.png \
  --knotted-core=knotted-core.txt all.txt
```

2.3 Real life example: protein 3KZN

In this section we apply the concepts discussed above on a real life example.

2.3.1 Knotoids

We start by downloading the protein structure that we want to analyze from the Protein database (PDB) [7]. In this example we shall use the protein 3KZN [22] that is known to form a deep trefoil knot.

```
$ wget https://files.rcsb.org/download/3KZN.pdb
```

Note for Windows users: with the *Windows PowerShell*, `wget` needs the additional option `-OutFile 3KZN.pdb`. Alternatively, the file `https://files.rcsb.org/download/3KZN.pdb` can be simply downloaded using a web browser.
The next step is to extract from the .pdb file the xyz-coordinates of the C_α atoms, that make up the protein backbone and store them into a file with xyz file format[31]. Note that a protein molecule may include more than one chains in its conformation so one has to be careful when extracting coordinates. In this example we work with chain A. Note that many proteins that are deposited in the PDB have gaps in their conformations. If this is the case, the gaps are connected with a straight line. One should be careful when dealing with such situations since connecting the endpoints of a gap with a straight line may alter the topology of the chain. For further details on the .pdb file format, the reader should refer to the documentation at http://www.wwpdb.org/documentation/file-format. The simple script `pdb_to_xyz.R` [32] distributed with *Knoto-ID* can be used to convert the pdb file `3KZN.pdb` to xyz format and save the output to `examples/3KZN_chain_A.xyz`:

```
$ scripts/pdb_to_xyz.R --output=examples/3KZN_chain_A.xyz 3KZN.pdb
```

We can evaluate the Jones polynomial for knotoids on the sphere of this protein chain by calling the command:

```
$ bin/polynomial_invariant examples/3KZN_chain_A.xyz
```

which generates the following output to standard error:

```
seed: 1501677085
polynomial invariant: Jones polynomial for knotoids.
Loading input curve
3D curve has 331 vertices
projection: -0.443782,-0.690045,-0.571748
Simplifying 3D curve
3D curve has 13 vertices
Evaluating diagram
diagram has 8 crossings
Simplifying diagram
diagram has 5 crossings
Simplifying diagram with random reidemeister moves III (max 100000 moves)
diagram has 5 crossings
Final simplifying diagram
diagram has 5 crossings
```

As we can see, the output includes information, amongst other things, on the random projection that was used and an upper bound on the crossing number of the diagram. After completion, the Jones polynomial for knotoids of the chain is written to standard output:

```
Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

For the moment, we don't have any information on the knotoid type that corresponds to this polynomial. However, this can be solved easily by using the option `--names-db` to specify a knotoid names database:

```
$ bin/polynomial_invariant --projection="-0.443782,-0.690045,-0.571748" \
  --names-db=examples/knotoid_names.txt examples/3KZN_chain_A.xyz
```

We have used the option `--projection` in order to use a fixed projection for our chain. In particular, we used the projection that was randomly chosen in the previous run. The option `--names-db=examples/knotoid_names.txt`

[31] see section "3.5.1 Piecewise linear curve (xyz)" for more information on the file format.
[32] To use this script, `R` [9] must be installed with packages `optparse` [10] and `Rpdb` [18].

allows the loading of the knotoids names database[33]. Please note that the file `knotoid_names.txt` is a work in progress and is given only as an example. The standard output now takes the following form:

```
Knotoid type: k3.1      Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

This means that this particular projection of the protein chain corresponds to a knot-type knotoid with 3 crossings. As mentioned in section "4 Theory", the chain is considered lying inside a large enough sphere. Each point of the sphere indicates a projection direction towards an oriented surface that lies outside the sphere that encloses the chain. There are two options for the oriented surface: a sphere, which is the default option and is the one that was used so far in this example, as well as the 2D plane. By adding the option `--planar`, we evaluate the knotoid type of the protein chain using planar knotoids.

```
$ bin/polynomial_invariant --projection="-0.443782,-0.690045,-0.571748" \
  --names-db=examples/knotoid_names.txt --planar examples/3KZN_chain_A.xyz
```

In this case the knotoid type of the protein chain remains the same upon evaluation using planar knotoids. Indeed

```
Knotoid type: k3.1      Polynomial:  - A^(-16) + A^(-12) + A^(-4)
```

Note that this is not the general case. There could be protein chains that have a planar knotoid type different from the knotoid type on the sphere.
In the example runs that we presented so far, the results were just printed to standard output. By adding the option `--output=FILENAME`, we can ask the program to print the results into a file. Moreover, by including `--output-diagram=FILENAME` the program creates a separate file that contains the corresponding knotoid diagram in PD format[34]:

```
$ bin/polynomial_invariant --projection="-0.443782,-0.690045,-0.571748" \
  --names-db=examples/knotoid_names.txt --planar --output-diagram=3KZN_diagram.txt \
  --output=3KZN_polynomial.txt examples/3KZN_chain_A.xyz
```

Note that option `--output-diagram-format=gauss` can be used to output extended Gauss codes[35] instead of PD codes for the knotoid diagrams.
Up until this point, we have dealt only with a single projection of the protein chain. In order to find the dominant knotoid type of a chain, we have to consider all of its projections and, subsequently, sample them in a uniform way. By adding the option `--nb-projections=N` the program samples N random projections with uniform distribution on the surface of the sphere. Below we give an example with 100 random projections:

```
$ bin/polynomial_invariant --nb-projections=100 --names-db=examples/knotoid_names.txt \
  --planar examples/3KZN_chain_A.xyz
```

The standard output includes a histogram of frequency of appearance of knotoid types. The knotoid with the highest frequency of appearance is the dominant knotoid type of the chain:

```
#frequency  knotoid_type  polynomial
0.59        k3.1          - A^(-16) + A^(-12) + A^(-4)
0.06        k9.12         - A^(-30)*v - 2*A^(-28) - A^(-28)*v - 2*A^(-26) + A^(-26)*v + A^(-24) + A^(-24)*v + ...
0.06        k1.1m*k3.1    + A^(-14)*v + A^(-12) - A^(-10)*v - A^(-8) - A^(-2)*v - 1
0.04        k4.2m         - A^(-18) - A^(-16) + A^(-14) + 2*A^(-12) - A^(-8) + A^(-4)
0.03        k2.2*k3.1     - A^(-18) - A^(-16) - A^(-16)*v + A^(-12) + A^(-12)*v + A^(-10) + A^(-6) + A^(-4) + ...
0.03        k5.3m         + A^(-26) - 2*A^(-22) - A^(-20) + A^(-18) + A^(-16) - A^(-14) + A^(-10) + A^(-8)
0.03        k2.5*k3.1     + A^(-20) - A^(-16) - A^(-16)*v^(2) - A^(-14)*v + A^(-12)*v^(2) + A^(-10)*v - A^(-8)...
0.03        k3.19m*k3.1   + A^(-14) + A^(-14)*v + A^(-12) + A^(-12)*v - A^(-10)*v - A^(-8) - A^(-8)*v - A^(-6)...
0.02        k8.28         - A^(-12) - A^(-10) + 2*A^(-8) + 2*A^(-6) - 2*A^(-4) - 2*A^(-2) + 2 + 3*A^(2) - A^(4...
0.02        k9.15         + A^(-22)*v + 2*A^(-20) + A^(-18) - A^(-18)*v - 3*A^(-16) - 2*A^(-14) + A^(-12) + A^...
0.02        k1.1*k3.1     + A^(-20) + A^(-18)*v - A^(-16) - A^(-14)*v - A^(-8) - A^(-6)*v
0.02        k2.3m*k3.1    + A^(-14)*v - 2*A^(-10)*v - A^(-8) + A^(-6)*v + A^(-4) - A^(-2)*v + A^(2)*v + A^(4)
0.01        k7.25         - A^(-24)*v - 2*A^(-22) - A^(-20) + A^(-20)*v + 2*A^(-18) + A^(-16) + A^(-12)*v + 2*...
0.01        k6.9          - A^(-18) - A^(-16) + 2*A^(-14) + 2*A^(-12) - 2*A^(-10) - 2*A^(-8) + 2*A^(-6) + 3*A^...
0.01        k8.42         - A^(-14) - A^(-12) - A^(-12)*v - A^(-10) - A^(-10)*v + A^(-8)*v + 2*A^(-6) + A^(-6)...
0.01        k7.31         - A^(-14) - 2*A^(-12) - A^(-10)*v + 3*A^(-8) + 2*A^(-6) + 2*A^(-6)*v - A^(-2) - 2*A^...
0.01        k5.4          - A^(-12) - A^(-10) + A^(-8) + 2*A^(-6) - A^(-2) + 1 + A^(2) - A^(6)
```

In this example, the dominant knotoid type for the protein 3KZN is the knot-type knotoid $k3.1$ with frequency of appearance 59%. Note that the exact frequency may change depending on the 100 random projections chosen. If

[33] see section "3.5.7 List of knot(oid) names" for more information on the file format.
[34] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[35] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.

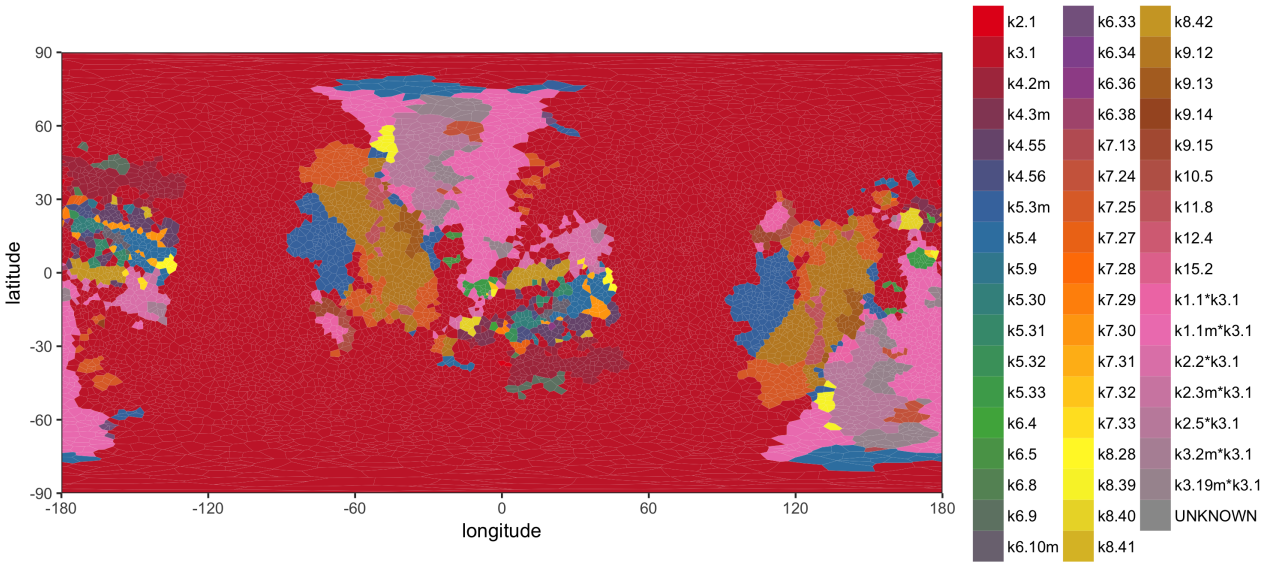

Figure 12: Projection map for the protein 3KZN (obtained with 10000 projections). For a projection direction (x, y, z), longitude (λ) and latitude (δ) are defined as $x = \cos(\delta)\cos(\lambda)$, $y = \cos(\delta)\sin(\lambda)$, $z = \sin(\delta)$. Colors correspond to planar knotoid types. Since the voronoi tesselation is performed on the sphere, voronoi facets are separated by arcs of great circles, which do not correspond to straight line after equirectangular projection.

multiple knotoid types have the same polynomial invariant[36], all knotoid types are concatenated with a `|` separator. We can also use a predetermined list of projections. In the following example we use the option `--projections-list` in order to specify a list of 100 fixed projection directions uniformly distributed on the surface of a sphere, weighted by the surface of their corresponding Voronoi cells[37].

```
$ bin/polynomial_invariant --projections-list=examples/projections_list_100.txt \
  --names-db=examples/knotoid_names.txt --planar examples/3KZN_chain_A.xyz
```

Using option `--output-diagram=diagrams.txt`, the list of projection directions with corresponding polynomials and knotoid types is saved to file `diagrams.txt`. Although 100 projection can be sufficient to obtain a first approximation of the projection map, better results can be achieved using a larger number of projections:

```
$ bin/polynomial_invariant --nb-projections=10000 --names-db=examples/knotoid_names.txt \
  --planar --output-diagram=diagrams.txt examples/3KZN_chain_A.xyz
```

The script `plot_projection_map.R` can then be used to plot the projection map using voronoi tesselation:

```
$ scripts/plot_projection_map.R --output=projection_map.png diagrams.txt
```

The resulting projection map is shown in Figure 12. The script `plot_projection_map.R` can also be used to create a 3D globe (webGL) with the backbone of the protein using options `--output-3D` and `--curve-3D`:

```
$ scripts/plot_projection_map.R --output=projection_map.png --output-3D=projection_map.html \
  --curve-3D=examples/3KZN_chain_A.xyz diagrams.txt
```

A screenshot is shown in Figure 13.

2.3.2 Knots

In order to analyze an open protein chain using knots, one works in an analogous way. Since the protein chain is an open 3D curve, we have to specify what method should be used to close the curve. At the moment the program supports two closure methods: "direct" and "rays"[38]. These can be specified by adding the option `--closure-method=direct`

[36] in the knotoid names database specified with `--names-db`.
[37] see section "3.5.6 List of projections" for a description of the file format.
[38] more details on these methods can be found in section "4.3 Knotoids and curves in space".

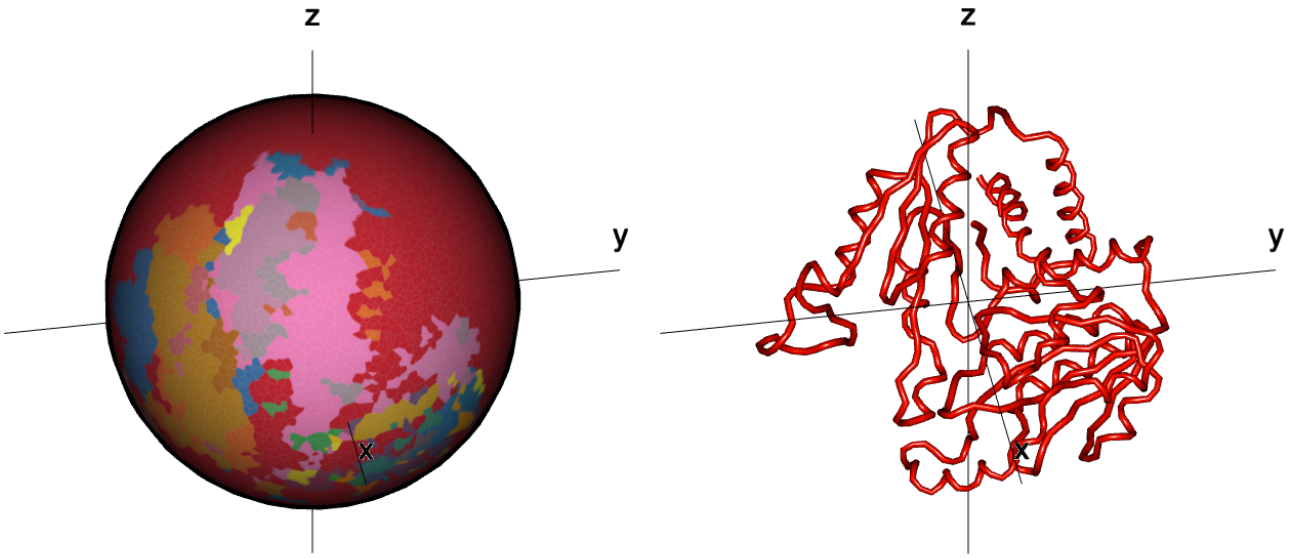

Figure 13: 3D projection map and backbone of the protein 3KZN, generated with the script `plot_projection_map.R` using 10000 projections. Color corresponds to polynomial.

for the former and `--closure-method=rays` for the latter.

```
$ bin/polynomial_invariant --closure-method=rays --names-db=examples/knot_names.txt \
  examples/3KZN_chain_A.xyz
```

Note that the file used for the knot names database (`examples/knot_names.txt` [39]) is not the same as the file used for the knotoid names database (`examples/knotoid_names.txt`).
The choice of projection directions can be specified with `--nb-projections` or `--projections-list` as discussed above. However, since the `direct` closure (`--closure-method=direct`) does not depend on the direction of projection, using multiple projections should only be used with `--closure-method=rays` .

2.3.3 Subchains, knotted cores and fingerprint matrices

We would like now to analyze the subchains of 3KZN's backbone using the `knotted_core` program. We start by calling

```
$ bin/knotted_core examples/3KZN_chain_A.xyz
```

The program will search for the knotted core, which is the shortest subchain obtained by progressively altering the length of the input curve by 1 point without changing the dominant knotoid type in the process. The dominant knotoid type of each subchain is evaluated by projecting on a sphere (using `--nb-projections=20` by default). During the run, information on the progression will be written to standard error and after completion, the knotted core will be written to standard output

```
#index_first  index_last  length  frequency  polynomial
168           252         85      0.5        - A^(-16) + A^(-12) + A^(-4)
170           254         85      0.45       - A^(-16) + A^(-12) + A^(-4)
```

Each line corresponds to a possible knotted core. Here, two subchains have the same the minimal length. Note that this list of possible knotted cores may change due to the random sampling of the 20 projection directions. The first knotted core is the subchain with starting index 168 (using zero-based indexing) and ending index 252 and it has a dominant polynomial invariant $-A^{-16} + A^{-12} + A^{-4}$ that appears in 50% of the projections[40]. The second solution starts at index 170, ends at index 254 and has the same dominant polynomial invariant that appears in 45% of the projection.
In analogy with the case explained above, we can specify the number of random projections that we want our sample

[39] This file was created using "The Rolfsen Knot Table" from The Knot Atlas (katlas.org).
[40] see section "3.5.9 List of subchains" for more information on the file format.

to include (`--nb-projections` or `--projections-list`), the knot or knotoid names (`--names-db`), the option of evaluating the diagrams on the plane (`--planar`), the closure method (`--closure-method`) and so on. For example, the following finds the knotted core of 3KZN using a set of predetermined projection directions and also loads the knotoid names database:

```
$ bin/knotted_core --projections-list=examples/projections_list_100.txt \
  --names-db=examples/knotoid_names.txt examples/3KZN_chain_A.xyz
```

When the option `--names-db` is used, the output has an additional column with the knot(oid) type corresponding to the dominant polynomial invariant (here k3.1):

```
#index_first  index_last  length  frequency  knotoid_type  polynomial
170           252         83      0.260337   k3.1          - A^(-16) + A^(-12) + A^(-4)
```

We proceed and ask `knotted_core` to produce all data required to generate the fingerprint matrix of 3KZN (using options `--output-all` , `--output-search` and `--output`). WARNING: evaluating all subchains (option `--output-all`) can be very slow[41].

```
$ bin/knotted_core --names-db=examples/knotoid_names.txt \
  --output-all=3KZN_all.txt --output-search=3KZN_search.txt \
  --output=3KZN_knotted_core.txt --timeout=1 examples/3KZN_chain_A.xyz
```

The option `--timeout=1` was used to interrupt the evaluation of the polynomial after 1 second. Indeed, for a few subchains and projections, the resulting diagram has a large number of crossings and the evaluation of the polynomial invariant may be very slow. Note that if the evaluation of a polynomial invariant takes more than 1 second, the knot type and polynomial invariant are set to `TIMEOUT` . This special value is treated as any other polynomial invariant and knot type and will appear in the output if it is the dominant knot type.
Subsequently we call the `R` script that creates the fingerprint matrix:

```
$ scripts/plot_knotted_core.R --output=3KZN_fingerprint_matrix.png 3KZN_all.txt
```

To overlay the knotted core(s) (yellow circle):

```
$ scripts/plot_knotted_core.R --output=3KZN_fingerprint_matrix.png \
  --knotted-core=3KZN_knotted_core.txt 3KZN_all.txt
```

The resulting fingerprint matrix is shown in Figure 14. The fingerprint matrix is dominated by k0.1, k2.1 and k3.1 knotoids. However, because of the random sampling of a small number of projection directions used to estimate the dominant knotoid type (`--nb-projections=20` by default), the figure has a high level of noise and several other knotoid types appear in the regions where the frequency of the dominant knotoid type is low. To decrease the level of noise, one possibility is to increase the number of projections. Figure 15 presents the same fingerprint matrix as in Figure 14 but obtained with `--nb-projections=1000` . As expected, increasing the number of projections strongly decreases the level of noise, and only three dominant knotoid types remain (k0.1, k2.1 and k3.1). In addition, a unique and shorter knotted core could be found (starting index 170, ending index 253, appearing in 31% of the projections). Obviously, increasing the number of projection not only increases the accuracy of the resulting fingerprint matrix, but it also increases the computational cost which can quickly become prohibitively high. Therefore, it may be worth keeping the number of projection low to obtain a first approximation of the fingerprint matrix, which may be sufficient for most applications (compare Figure 14 and Figure 15).
Finally, we ask `knotted_core` to produce the disk matrix of 3KZN. Since the endpoints of the chain are very close, there is no problem with connecting them with a straight line and to consider the input curve as circular (option `--cyclic-input`). WARNING: evaluating all subchains (option `--output-all`) can be very slow[42].

```
$ bin/knotted_core --cyclic-input --closure-method=rays --names-db=examples/knot_names.txt \
  --output-all=3KZN_all.txt --output-search=3KZN_search.txt \
  --output=3KZN_knotted_core.txt --timeout=1 examples/3KZN_chain_A.xyz
```

A disk matrix can be produced using `plot_knotted_core.R` with option `--cyclic` . WARNING: drawing heatmaps in polar coordinate using `R` package `ggplot2` [11] can be very slow[43].

[41] On a MacBook Pro from 2011 with 2.4 GHz Intel Core i5 CPU, it takes approximately 15 minutes to execute this command.
[42] On a MacBook Pro from 2011 with 2.4 GHz Intel Core i5 CPU, it takes approximately 50 minutes to execute this command.
[43] On a MacBook Pro from 2011 with 2.4 GHz Intel Core i5 CPU, it takes approximately 20 minutes to execute this command.

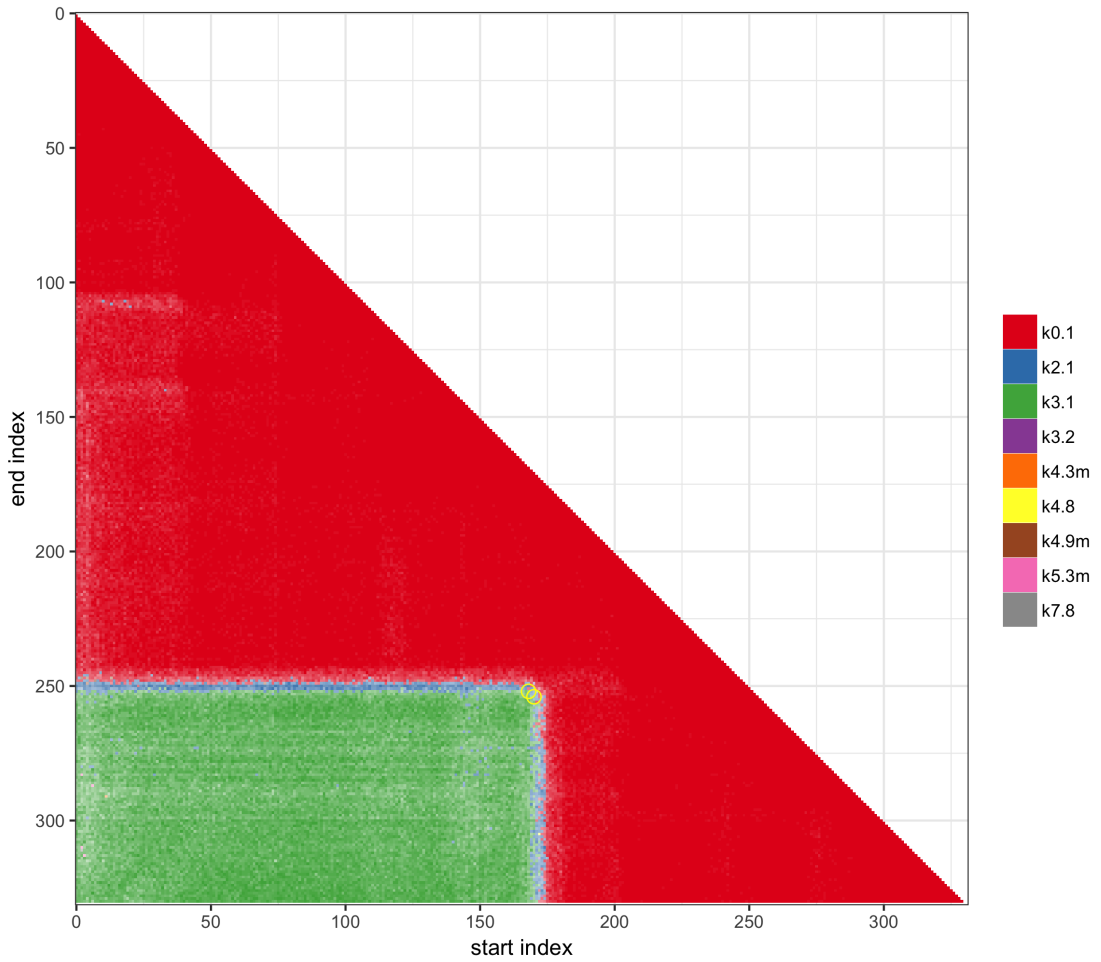

Figure 14: The knotoid fingerprint matrix for the protein 3KZN. This proteins forms a deep trefoil knotoid (bottom left corner corresponds to the full chain). Color corresponds to dominant knotoid type of the subchain and transparency corresponds to its frequency. Yellow circles are centered on knotted core(s).

```
$ scripts/plot_knotted_core.R --cyclic --output=3KZN_circular_matrix.png \
  --knotted-core=3KZN_knotted_core.txt 3KZN_all.txt
```

The resulting disk matrix is shown in Figure 16

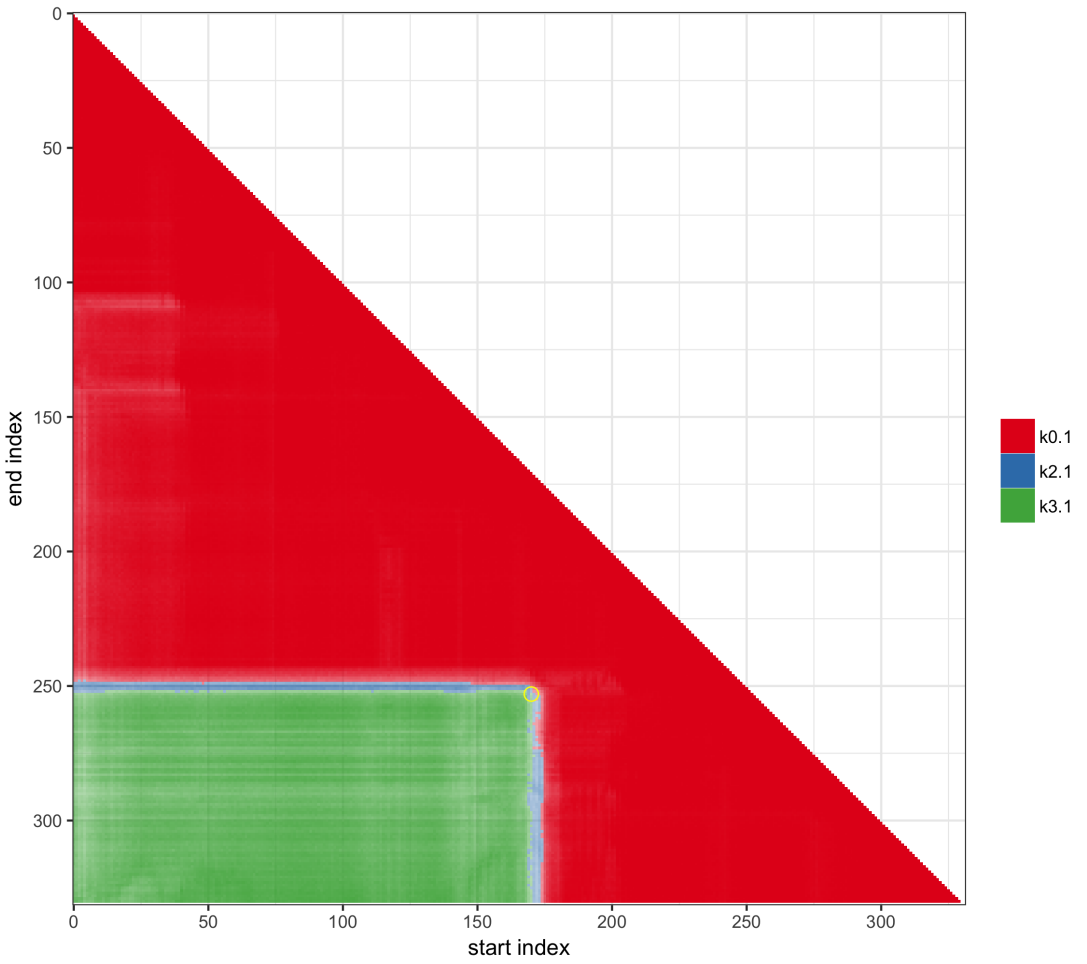

Figure 15: The knotoid fingerprint matrix for the protein 3KZN obtained with `--nb-projections=1000`. A yellow circle is centered on the knotted core.

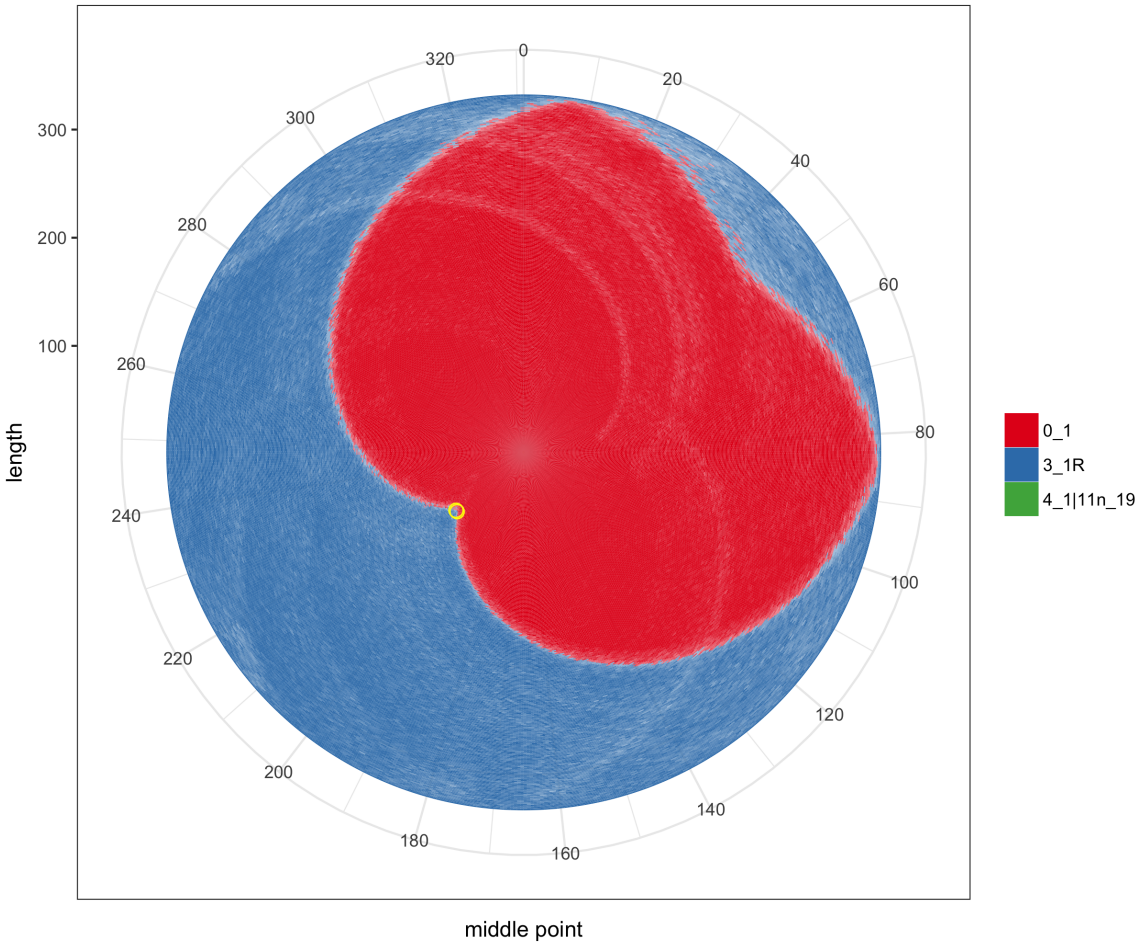

Figure 16: The disk matrix for the protein 3KZN with uniform closure (i.e. `--closure-method=rays` in multiple directions). This proteins forms a right handed trefoil knot (all points on the enclosing circle corresponds to the full chain). Color corresponds to dominant knot type of the subchain and transparency corresponds to its frequency. Yellow circles are centered around knotted core(s).

3 Command reference

3.1 Polynomial invariant

3.1.1 Usage

```
$ bin/polynomial_invariant [options] FILENAME
```

Load a piecewise linear curve from file `FILENAME`[44]. To read from standard input instead, use file name `stdin` or `-`. By default, the curve is open (knotoid), but it can be closed (knot) by specifying a closure method with option `--closure-method`. The input curve is then simplified using a 3D triangle elimination method[45].
Evaluate the knot(oid) diagram obtained by projecting the curve along a randomly chosen projection direction (with uniform distribution on the surface of the sphere) or along the direction specified with option `--projection`.
Alternatively, if `--input-format=pd` or `--input-format=gauss`, load directly the PD or extended Gauss code for a knot(oid) diagram from file `FILENAME`.
Check that the knot(oid) diagram is valid using the method proposed by Vijayan and Wigderson [21].
Simplify the knot(oid) diagram with Reidemeister moves[46].
Evaluate the polynomial invariant for the knot(oid) diagram on the surface of a sphere (by default) or on a plane (with option `--planar`).
Output the polynomial invariant, and optionally the knot(oid) diagram[47] (with option `--output-diagram`).
If options `--nb-projections` or `--projections-list` are used, the above procedure is repeated for all projections, and the output is a distribution of polynomials.
The polynomial invariant is the classical Jones polynomial for knots [23] when the curve in closed (`--closure-method=direct` or `rays`), the Jones polynomial for knotoids [1, 2] when the curve is open and the knotoid diagram is on the surface of a sphere (`--closure-method=open`, without `--planar`), and the Turaev loop bracket polynomial for knotoid [1] when the curve is open and the knotoid diagram is on the surface of a plane (`--closure-method=open` and `--planar`).

3.1.2 Options

`-h`, `--help`
Print usage.

`-V`, `--version`
Print *Knoto-ID* version.

`-s SEED`, `--seed=SEED`
Initialize the random number generator with seed `SEED`. If not specified, the seed is taken from the current time.

`-F FORMAT`, `--input-format=FORMAT`
Specify input format. Possible values for `FORMAT`:

- `xyz` piecewise linear curve in xyz file format[48].
- `pd` PD code for a knot(oid) diagram in KnotTheory file format[49]. With input format `pd`, options `--projection`, `--nb-projections`, `--projections-list` and `--closure-method` are ignored.
- `gauss` extended Gauss code for a knot(oid) diagram[50]. Use option `--closure-method=open` to specify that the diagram is open and `--closure-method=direct` or `rays` to specify that the diagram is closed. With input format `gauss`, options `--projection`, `--nb-projections` and `--projections-list` are ignored.

Default: `xyz`.

[44] in xyz file format, see section "3.5.1 Piecewise linear curve (xyz)" for more information on the file format.
[45] see section "3.4.1 Curve simplification using triangle elimination" for more details.
[46] see section "3.4.2 Knot(oid) diagram simplification" for more details.
[47] Output the simplified diagram.
[48] see section "3.5.1 Piecewise linear curve (xyz)" for more information on the file format.
[49] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[50] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.

`-o FILENAME`, `--output=FILENAME`

Output the polynomial or distribution of polynomials to file `FILENAME`. To write to standard output, use file name `stdout` or `-`.
Default `stdout`.

`--output-diagram=FILENAME`

Output the knot(oid) diagram or list diagrams to file `FILENAME` using format specified with `--output-diagram-format`.

To write to standard output, use file name `stdout` or `-`.

`--output-diagram-format=FORMAT`

Specify output format for knot(oid) diagrams. Possible values for `FORMAT`:

- `pd` PD code for a knot(oid) diagram in KnotTheory file format[51].
- `gauss` extended Gauss code for a knot(oid) diagram[52].

Default: `pd`.

`-m METHOD`, `--closure-method=METHOD`

Specify how to close the input curve. Possible values for `METHOD`:

- `open` the curve is open (knotoid).
- `direct` connect last point to first point by a straight line.
- `rays` close by extending two parallel rays along the projection direction, each originating from one of the endpoints of the curve and connecting them outside the sphere that encloses the curve[53].

Default: `open`.

`-p`, `--planar`

Evaluate polynomial invariant for the knot(oid) diagram on a plane. If not specified, evaluate polynomial invariant for the knot(oid) diagram on the surface of a sphere. This option is only relevant for open curves (`--closure-method=open`).

`--nb-moves-III=NMOVES`

Use `NMOVES` iterations to simplify the knot(oid) diagram using Reidemeister move III[54].
Default `100000`.

`--projection="X,Y,Z"`

Project the input curve along projection direction (X,Y,Z). If not specified, use randomly chosen projection direction (with uniform distribution on the surface of the sphere).

`-N NPROJ`, `--nb-projections=NPROJ`

Choose `NPROJ` random projection directions (with uniform distribution on the surface of the sphere) and evaluate the corresponding polynomial invariant for each projection. Note:

- Instead of a unique polynomial, the output specified with `--output` will contain the distribution of polynomials[55].
- The output specified with `--output-diagram` will contain a list of projections with polynomials and knot(oid) diagrams[56].
- For cyclic curves with closure method `direct` the polynomial invariant (classical Jones polynomial) does not depend on the projection. To avoid useless computations `NPROJ` will be set to 1.
- Option `--projection` is ignored.

[51] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[52] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.
[53] see section "4.3 Knotoids and curves in space" for more details.
[54] see section "3.4.2 Knot(oid) diagram simplification" for more details.
[55] see section "3.5.4 Distribution of polynomials" for more information on the file format.
[56] see section "3.5.5 List of projections with polynomials and knot(oid) diagrams" for more information on the file format.

`--projections-list=FILENAME`

Load a list of projection directions from `FILENAME` [57] and evaluate the corresponding polynomial invariant for each projection. Note:

- Instead of a unique polynomial, the output specified with `--output` will contain the distribution of polynomials[58].
- The output specified with `--output-diagram` will contain a list of projections with polynomials and knot(oid) diagrams[59].
- For cyclic curves with closure method `direct` the polynomial invariant (classical Jones polynomial) does not depend on the projection. To avoid useless computations, only one projection direction will be used.
- Options `--nb-projections` and `--projection` are ignored.

`-n FILENAME , --names-db=FILENAME`

Load a list of polynomials with corresponding knot(oid) names from file `FILENAME` [60].

`--timeout=TIMEOUT`

Abort evaluation of polynomial invariant after `TIMEOUT` seconds (integer number). Note: if the evaluation of the polynomial invariant is aborted, the polynomial is replaced by `TIMEOUT` .

3.2 Knotted core

3.2.1 Usage

```
$ bin/knotted_core [options] FILENAME
```

Load a piecewise linear curve from file `FILENAME` [61] and evaluate the knotted core[62]. To read from standard input instead, use file name `stdin` or `-` . By default, the curve is considered open, but it can be closed with option `--cyclic-input` .
Output the knotted core, and optionally all subchains tested when searching the knotted core (with option `--output-search`) as well as all possible subchains of the input curve (with option `--output-all`).
Note that the knotted core corresponds to the "top-down" knotted core discussed by Tubiana and coauthors [24].

3.2.2 Options

`-h , --help`

Print usage.

`-V , --version`

Print *Knoto-ID* version.

`-s SEED , --seed=SEED`

Initialize the random number generator with seed `SEED` . If not specified, the seed is taken from the current time.

`-o FILENAME , --output=FILENAME`

Output the knotted core to file `FILENAME` [63]. To write to standard output, use file name `stdout` or `-` . Default `stdout` .

`--output-search=FILENAME`

Output all subchains tested when searching for the knotted core to file `FILENAME` [64]. To write to standard output, use file name `stdout` or `-` .

[57] see section "3.5.6 List of projections" for more information on the file format.
[58] see section "3.5.4 Distribution of polynomials" for more information on the file format.
[59] see section "3.5.5 List of projections with polynomials and knot(oid) diagrams" for more information on the file format.
[60] see section "3.5.7 List of knot(oid) names" for more information on the file format.
[61] in xyz file format, see section "3.5.1 Piecewise linear curve (xyz)" for more information on the file format.
[62] see section "3.4.4 Knotted core" for more information on the file format.
[63] see section "3.5.9 List of subchains" for more information on the file format.
[64] see section "3.5.9 List of subchains" for more information on the file format.

`--output-all=FILENAME`

Output all possible subchains of the input curve to file `FILENAME`[65]. To write to standard output, use file name `stdout` or `-`. Warning: evaluating the polynomial invariants for all subchains of the input curve may be slow.

`-C`, `--cyclic-input`

Close the input curve by connecting its last point to its first point by a straight line.

`-m METHOD`, `--closure-method=METHOD`

Specify how to close each subchain. Possible values for `METHOD`:

- `open` the subchain is open (knotoid).
- `direct` connect last point to first point by a straight line.
- `rays` close by extending two parallel rays along the projection direction, each originating from one of the endpoints of the curve and connecting them outside the sphere that encloses the curve[66].

Default: `open`.

`-p`, `--planar`

For each subchain, evaluate its polynomial invariant for the knot(oid) diagram on a plane. If not specified, evaluate polynomial invariant for the knot(oid) diagram on the surface of a sphere. This option is only relevant for open curves (`--closure-method=open`).

`--nb-moves-III=NMOVES`

Use `NMOVES` iterations to simplify the knot(oid) diagram using Reidemeister move III[67].
Default `10`.

`-N NPROJ`, `--nb-projections=NPROJ`

Use `NPROJ` random projection directions (with uniform distribution on the surface of the sphere) to evaluate the dominant polynomial invariant of each subchain. For cyclic subchains with closure method `direct` the polynomial invariant (classical Jones polynomial) does not depend on the projection. To avoid useless computations `NPROJ` will be set to 1.
Default: `20`.

`--projections-list=FILENAME`

Evaluate the dominant polynomial using the list of projection directions in file `FILENAME`[68]. Note:

- For cyclic subchains with closure method `direct` the polynomial invariant (classical Jones polynomial) does not depend on the projection. To avoid useless computations, only one projection direction will be used.
- Options `--nb-projections` is ignored.

`-n FILENAME`, `--names-db=FILENAME`

Load a list of polynomials with corresponding knot(oid) names from file `FILENAME`[69].

`--timeout=TIMEOUT`

Abort evaluation of polynomial invariant after `TIMEOUT` seconds (integer number). Note: if the evaluation of the polynomial invariant is aborted, the polynomial is replaced by `TIMEOUT`.

3.3 Convert diagram

3.3.1 Usage

```
$ bin/convert_diagram [options] FILENAME
```

Load a knot(oid) diagram file `FILENAME`. To read from standard input instead, use file name `stdin` or `-`. Depending on `--input-format`, different types of files can be opened:

[65] see section "3.5.9 List of subchains" for more information on the file format.
[66] see section "4.3 Knotoids and curves in space" for more details.
[67] see section "3.4.2 Knot(oid) diagram simplification" for more details.
[68] see section "3.5.6 List of projections" for more information on the file format.
[69] see section "3.5.7 List of knot(oid) names" for more information on the file format.

- If `--input-format=pd` (by default), load the diagram in PD code format [70].
- If `--input-format=gauss`, load the diagram in extended Gauss code format[71]. Option `--closure-method` is used to specify whether the diagram is open (default) or closed.
- If `--input-format=xyz`, load a piecewise linear curve in xyz format[72]. By default, the curve is open (knotoid), but it can be closed (knot) by specifying a closure method with option `--closure-method`. If option `--3D-reduction` is given, the input curve is simplified using a 3D triangle elimination method[73].

  A knot(oid) diagram is then obtained by projecting the curve along a randomly chosen projection direction (with uniform distribution on the surface of the sphere) or along the direction specified with option `--projection`.

Check that the knot(oid) diagram is valid using the method proposed by Vijayan and Wigderson [21].
If option `--simplify-diagram` is given, the knot(oid) diagram is simplified using Reidemeister moves. Finally, the resulting diagram is saved to file specified by option `--output`, with format specified by `--output-format`.
Notes:

- If the input is in xyz format or contains information on which arcs touch the outside region, option `--planar` can be used to keep this information in the output.
- To output in "xyz for knot(oid) diagram" format, `convert_diagram` uses a circle packing algorithm[74] to draw the diagram in the x-y plane (with z storing information on what stand is above/below in each crossing). While this algorithm usually produces good results for knot diagrams (see Figures 1 and 3), it may also produce poorly balanced results, with circle radii (or arc lengths) differing by several orders of magnitude. This problem of unbalanced circle packing is particularly strong for planar knotoid diagrams, in particular when the diagram is enclosed within an external loop. To mitigate this problem, it is possible to use a simulated annealing algorithm to "relax" the diagram (with option `--nb-iterations-relaxation`). By default, if the resulting diagram is not balanced (i.e. ratio of largest arc length to shortest arc length is larger than 20), `convert_diagram` will quit with an error without writing the resulting diagram (to avoid producing unreadable and potentially misleading output). To ignore this test and write the diagram even if it is unbalanced, use option `--force`.

3.3.2 Options

`-h`, `--help`
Print usage.

`-V`, `--version`
Print *Knoto-ID* version.

`-s SEED`, `--seed=SEED`
Initialize the random number generator with seed `SEED`. If not specified, the seed is taken from the current time.

`-p`, `--planar`
Use information on what arcs are touching the outside region.

WARNING: exit with error if the input file does not contain this information.

`-F FORMAT`, `--input-format=FORMAT`
Specify input format. Possible values for `FORMAT`:

- `xyz` piecewise linear curve in xyz file format[75].
- `pd` PD code for a knot(oid) diagram in KnotTheory file format[76]. With input format `pd`, options `--projection` and `--closure-method` are ignored. If the input file contains multiple PD codes, only the first one will be used.

[70] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[71] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.
[72] see section "3.5.1 Piecewise linear curve (xyz)" for more information on the file format.
[73] see section "3.4.1 Curve simplification using triangle elimination" for more details.
[74] see section "3.4.5 Circle packing" for more information.
[75] see section "3.5.1 Piecewise linear curve (xyz)" for more information on the file format.
[76] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.

- `gauss` extended Gauss code for a knot(oid) diagram[77]. Use option `--closure-method=open` to specify that the diagram is open and `--closure-method=direct` or `rays` to specify that the diagram is closed. With input format `gauss`, option `--projection` is ignored. If the input file contains multiple Gauss codes, only the first one will be used.

Default: `xyz`.

`--output-format=FORMAT`

Specify output format. Possible values for `FORMAT`:

- `pd` PD code for knot(oid) diagrams[78].
- `gauss` extended Gauss code for knot(oid) diagrams[79].
- `xyz` xyz format for knot(oid) diagrams[80].

Default: `pd`.

`-o FILENAME, --output=FILENAME`

Output the knot(oid) diagram to file `FILENAME` using format specified with `--output-format`. To write to standard output, use file name `stdout` or `-`.
Default `stdout`.

`--simplify-diagram`

Simplify the knot(oid) diagram using Reidemeister moves.

`--nb-moves-III=NMOVES`

Use `NMOVES` iterations to simplify the knot(oid) diagram using Reidemeister move III[81].
Default `100000`.

Options for xyz input format:

`-m METHOD, --closure-method=METHOD`

Specify how to close the input curve. Possible values for `METHOD`:

- `open` the curve is open (knotoid).
- `direct` connect last point to first point by a straight line.
- `rays` close by extending two parallel rays along the projection direction, each originating from one of the endpoints of the curve and connecting them outside the sphere that encloses the curve[82].

Only used with `--input-format=xyz` and `gauss`.
Default: `open`.

`--projection="X,Y,Z"`

Project the input curve along projection direction (X,Y,Z). If not specified, use randomly chosen projection direction (with uniform distribution on the surface of the sphere). Only used with `--input-format=xyz`.

`--3D-reduction`

Simplify the input curve using a 3D triangle elimination method[83]. Only used with `--input-format=xyz`.

Options for xyz output format:

`--force`

Output the knot(oid) diagram without checking if it is balanced (i.e. same order of magnitude for all arc lengths). Without `--force`, quit without saving the diagram if it is not balanced.

`--nb-iterations-relaxation=N`

After circle packing, relax the knot(oid) diagram with N iterations of simulated annealing.
Default: `0`.

[77] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.
[78] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[79] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.
[80] see section "3.5.10 xyz format for knot(oid) diagrams" for more information on the file format.
[81] see section "3.4.2 Knot(oid) diagram simplification" for more details.
[82] see section "4.3 Knotoids and curves in space" for more details.
[83] see section "3.4.1 Curve simplification using triangle elimination" for more details.

3.4 Algorithms

In the following the main algorithms used in *Knoto-ID* are briefly described.

3.4.1 Curve simplification using triangle elimination

Evaluating a polynomial invariant can be very challenging since the run time scales exponentially with the number of crossings in the knot(oid) diagram. Therefore, one needs to reduce as much as possible the number of crossings. One way to achieve this goal consists in simplifying the 3D curve used as input, without modifying its topology, before evaluating its knot(oid) diagram. To do this, we use a triangle elimination method based on the KMT algorithm [25] [26]: For each triplet of sequential points along the curve, if the triangle formed by the three points is not crossed by any segment of the curve, the middle point is removed and the first and last point of the triplet are directly connected by a straight line. This operation is repeated iteratively until no point of the curve can be removed. When the input curve is open (knotoid), we first introduce two infinite lines that pass through the endpoints of the curve, parallel to the direction of projection. The same algorithm is then used with a small modification: a point is removed only if the triangle is not crossed by any segment of the curve nor by any infinite line.

3.4.2 Knot(oid) diagram simplification

The next step of simplification is performed directly on the knot(oid) diagram.

Simplification with type I and II Reidemeister moves. The knot(oid) diagram is simplified by iteratively applying type I and II Reidemeister moves[84] that decrease the number of crossings until no more crossings can be removed.

Random shuffling with type III Reidemeister. Among all possible type III Reidemeister moves that can be applied to the knot(oid) diagram, one move is randomly chosen and applied to the diagram. This operation is followed by a simplification with type I and II Reidemeister moves described previously. The combination of Random shuffling with type III Reidemeister followed by a simplification with type I and II Reidemeister moves is then repeated multiple times (specified with `--nb-moves-III`).
While the simplification method can strongly reduce the complexity of the knot(oid) diagram, and therefore the computational time, it introduces an overhead that can become non-negligible for relatively simple diagrams. Therefore it may be worth adjusting the number of iterations with `--nb-moves-III` to find the optimal balance between computational effort spent in the simplification of the knot(oid) diagram and in the evaluation of the polynomial invariant.

3.4.3 Polynomial invariant

The polynomial invariants are evaluated by recursively applying the skein relations discussed in section "4.4 Computing polynomial invariants" to all crossings of the knot(oid) diagram. At each step of the recursion, the resulting diagram is simplified by type I and II Reidemeister moves whenever possible, thus reducing the number of iteration in the recursion. Although the run time of this algorithm still scales exponentially with the number of crossings in the diagram, the simplification by type I and II Reidemeister moves reduces the execution time by a factor that increase exponentially with the number of crossings.

3.4.4 Knotted core

The knotted core is defined as the shortest subchain obtained by progressively altering the length of the whole curve by 1 point without changing the knot(oid) type in the process. When the input curve is open, the knotted core can be easily visualized on the fingerprint matrix (Figure 17) as the point of the path connected region that include the full curve (lower left corner) and is closer to the diagonal of the matrix (yellow dot in Figure 17). When the curve is closed, it can be visualized on the disk matrix (Figure 18) as the point of the path connected region that include the full curve (outer circle) and is closer to the origin (yellow dot in Figure 18). In both cases, the algorithm starts with the full curve (lower left corner in the fingerprint matrix, any point on the outer ring in the disk matrix) and reduces the length of the curve by removing one point at a time from the end of the curve (i.e. decreasing end index with constant start index) while the resulting subchain has the same polynomial as the full curve (green region in Figures 17 and 18). Once a subchain has a different polynomial than the full curve (reaching the border of the green region in Figures 17 and 18), the algorithm starts to follow the boundary of the path connected region that include the full curve, by adding or removing one point at a time from the ends of the subchain, until the boundary has been fully visited. The knotted core is then obtained as the shortest subchain(s) found during this search, with same polynomial as the full curve.

[84] see Figure 32 in section "3.5.7 List of knot(oid) names".

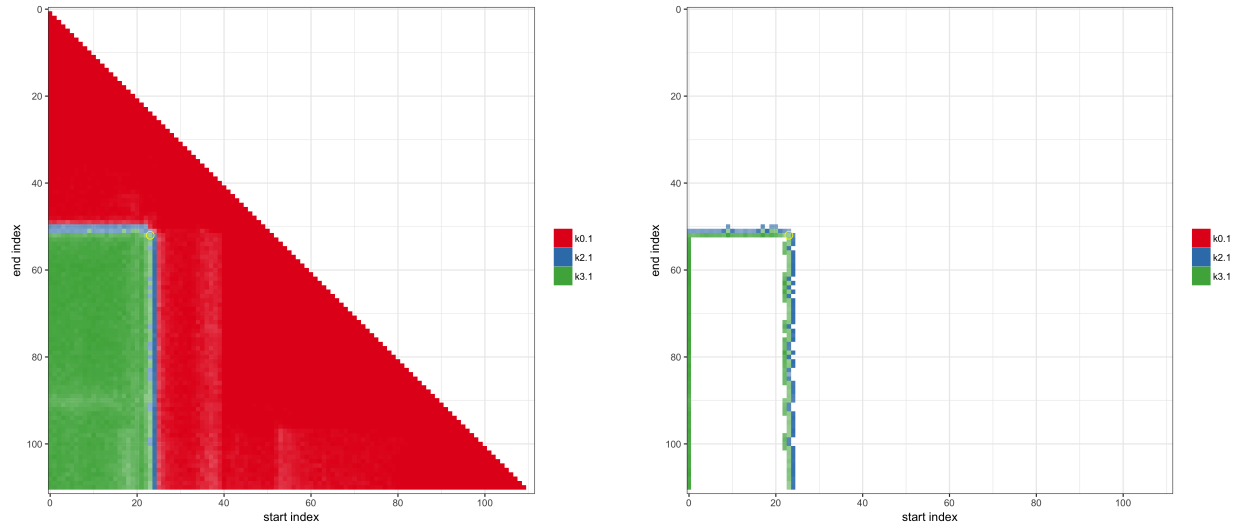

Figure 17: Left: Fingerprint matrix for the curve `examples/3_1R_supercoiled.xyz`. The knotted core is shown with a yellow circle. Right: Search path to find the knotted core.

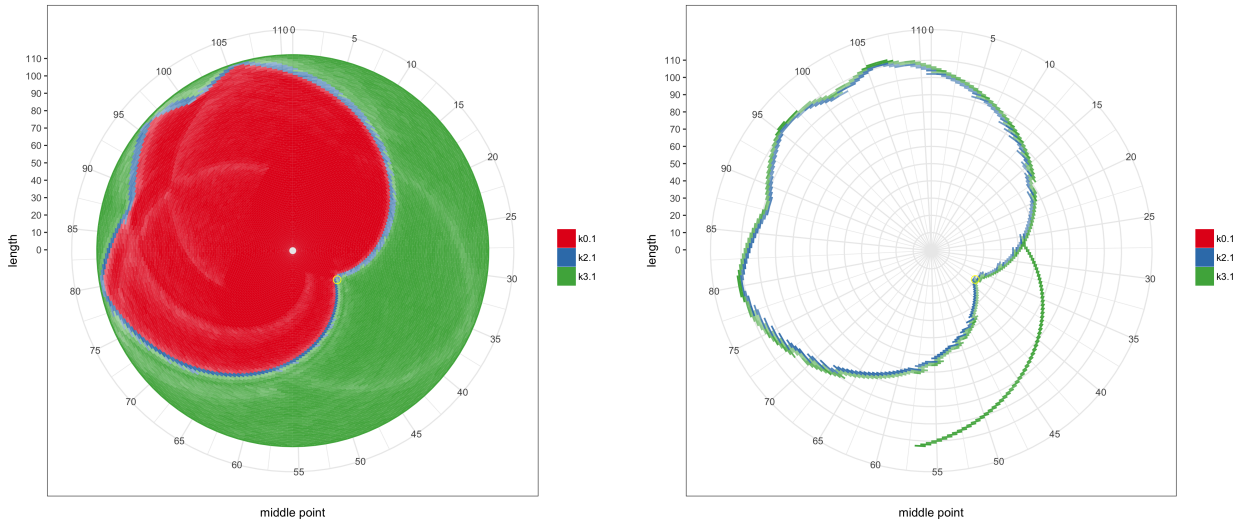

Figure 18: Left: Disk matrix for the curve `examples/3_1R_supercoiled.xyz`. The knotted core is shown with a yellow circle. Right: Search path to find the knotted core.

While the full fingerprint and disk matrices were shown to illustrate the algorithm, `knotted_core` does not compute them to find the knotted core. It only evaluates the minimal number of subchains needed to find and follow the boundary of the path connected region that include the full curve. The actual search path is shown in the right panels of Figures 17 and 18.

3.4.5 Circle packing

To output knot(oid) diagrams in xyz format, we use the circle packing algorithm described by Collins and Stephenson [27]. Our implementation is based on the python implementation by David Eppstein [85]. To decrease the sensitivity to errors due to finite precision arithmetic, we used a breadth first search to place circle centers instead of the original depth first search.

3.5 File formats

3.5.1 Piecewise linear curve (xyz)

Tab (or space) separated input format with 3 columns corresponding to x, y and z coordinates. Each row correspond to a point of the curve. Consecutive points (rows) will be connected by a straight line. All points should be distinct.

[85] http://www.ics.uci.edu/~eppstein/PADS/CirclePack.py

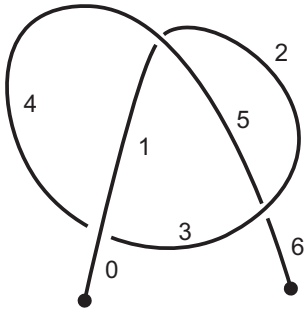

Figure 19: Labeled knotoid diagram corresponding to the PD code `PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]]`.

In particular, first and last point should not be equal. To define a closed curve with last point connected to first point by a straight line, use option `--closure-method=direct`. The following example defines a curve with 4 points with coordinates $(0,0,0)$, $(1,0,0)$, $(1,1,0)$ and $(0,1,0)$

```
0 0 0
1 0 0
1 1 0
0 1 0
```

3.5.2 Knot(oid) diagrams (PD code)

Input and output format used to encode knot or knotoid diagrams based on the Planar Diagram format (or PD code) from the Mathematica package KnotTheory [28] (see http://katlas.org/wiki/Planar_Diagrams). PD codes provide a presentation of a knotoid or a knot diagram that can be easily handled by a computer. Start by labelling the arcs of the diagram in the following way. If we are dealing with a knot diagram, we choose a random starting point on the diagram and we go around labelling the arcs with natural numbers (including zero). The labels increase in number each time we pass a crossing. The crossings are presented as symbols $\mathrm{X}[\mathrm{a},\mathrm{b},\mathrm{c},\mathrm{d}]$ where $\mathrm{a},\mathrm{b},\mathrm{c},\mathrm{d}$ are the labels of the edges that make the particular crossing, starting from the lower incoming edge and proceeding counterclockwise. Note that a,c correspond to the underpassing arc and b,d to the overpassing arc. Moreover, we always have that $\mathrm{a}<\mathrm{c}$ while there is no such restriction for the overpassing arc. If we are dealing with a knotoid diagram on the sphere, we start from the tail of the knotoid and we go around repeating the same process as with the case of knots. Let's have a look at the following example (the corresponding labeled knotoid diagram is shown in Figure 19):

$$\mathrm{PD}[\mathrm{X}[\,3,\overbrace{1,4,0}^{\text{overpass}}\,],\ \mathrm{X}[1,5,2,4],\ \mathrm{X}[5,3,6,2]]$$
$$\underbrace{}_{\text{underpass}}$$

A PD code for a knotoid diagram projected on the plane is almost identical to a regular PD code, except for the fact that it includes also the information of which arcs touch the outside region of the diagram. If a is such an arc, we indicate it by $\mathrm{r}[\mathrm{a}]$ in the PD code. For example, the following PD code corresponds to the labeled knotoid diagram on the plane of Figure 20

$$\mathrm{PD}[\mathrm{X}[2,\mathrm{r}[1],\mathrm{r}[3],0],\ \mathrm{X}[\mathrm{r}[1],4,2,\mathrm{r}[3]]]$$

Example of PD code for a knotoid diagram on the sphere (Figure 21 left):

```
PD[
X[0,3,1,4],
X[4,1,5,2],
X[2,5,3,6]
];
```

It can also be written one line

```
PD[X[0,3,1,4],X[4,1,5,2],X[2,5,3,6]];
```

Example of PD code for a knotoid diagram on the plane (Figure 21 right):

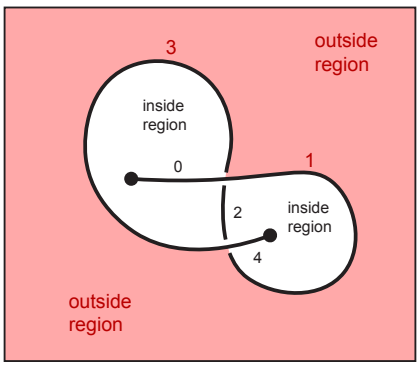

Figure 20: Labeled planar knotoid diagram corresponding to the PD code `PD[X[2,r[1],r[3],0],X[r[1],4,2,r[3]]]`.

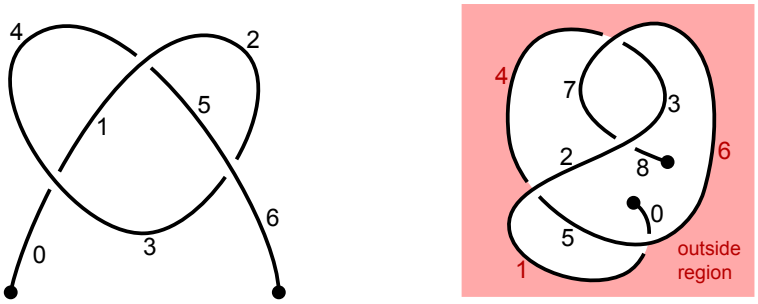

Figure 21: Left: Labeled knotoid diagram corresponding to the PD code `PD[X[0,3,1,4],X[4,1,5,2],X[2,5,3,6]]`. Right: Labeled knotoid diagram on the plane corresponding to the PD code `PD[X[0,5,r[1],r[6]],X[r[4],r[1],5,2],X[7,2,8,3],X[3,r[6],r[4],7]]`.

```
PD[
X[0,5,r[1],r[6]],
X[r[4],r[1],5,2],
X[7,2,8,3],
X[3,r[6],r[4],7]
];
```

and in one line

```
PD[X[0,5,r[1],r[6]],X[r[4],r[1],5,2],X[7,2,8,3],X[3,r[6],r[4],7]];
```

Example of PD code for a knot diagram:

```
PD[
X[0,3,1,4],
X[4,1,5,2],
X[2,5,3,0]
];
```

and in one line

```
PD[X[0,3,1,4],X[4,1,5,2],X[2,5,3,0]];
```

Knoto-ID also accept files with multiple diagrams. An example with two knotoid diagrams on the sphere:

```
PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
PD[X[0,3,1,4],X[4,1,5,2],X[2,5,3,6]];
```

or simply concatenated in one line

```
PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];PD[X[0,3,1,4],X[4,1,5,2],X[2,5,3,6]];
```

The semicolons are not necessary and the following is also a valid list of diagrams

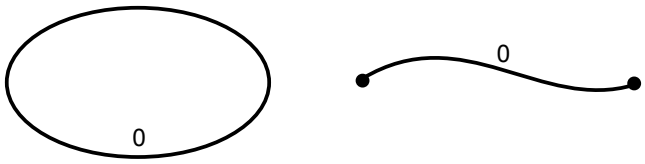

Figure 22: Left: Labeled knot diagram corresponding to the empty PD code `PD[]` . Right: Labeled knotoid diagram corresponding to the empty PD code `PD[]` .

Figure 23: A positive crossing with sign $+1$ (a) and a negative crossing with sign -1 (b).

```
PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]]PD[X[0,3,1,4],X[4,1,5,2],X[2,5,3,6]]
```

Finally, diagrams without crossings are allowed (Figure 22):

```
PD[]
```

3.5.3 Knot(oid) diagrams (Extended Gauss code)

Input and output format used to encode knot or knotoid diagrams based on the extended Gauss codes [29,2]. Extended gauss codes allows to encode knots and knotoid diagrams on the sphere and have the following form:

$$\text{Gauss code} = \underbrace{\text{Crossings}}_{\text{first part}} \quad \underbrace{\text{Signs}}_{\text{second part}}$$

The **first part** is a sequence of the crossings of the diagram that is obtained by travelling around it, starting from one end and proceeding to the other, and labeling the crossings as we meet them (using strictly increasing non-negative integers). If we meet an undercrossing we note the crossing with a "-" sign, while if we meet an overcrossing we note it with a "+" sign. Note that each crossing is met twice in this trip and so each crossing will appear with both signs in the sequence. Therefore the length of this sequence is $2n$, where n is the number of crossings. The case of knots is similar. The only difference is that we start the trip around the diagram from any arbitrary point. Note that different choices of starting points may yield different codes however, all codes will represent the same diagram.

The **second part** is a sequence of the signs of each of the crossings of the diagram (Figure 23). Without it, the Gauss code represents a diagram up to its mirror image. The length of this sequence is n.

For example, the knotoid diagram in Figure 24 (left) corresponds to the following extended Gauss code:

```
-1 2 -3 1 -2 3 ---
```

Another example is the knotoid diagram shown in Figure 25, which corresponds to the following extended Gauss code:

```
1 -2 3 -1 -4 -5 2 -3 5 4 +++-+
```

Except for pure knotoids, it is not possible to know whether a diagram is open or closed based only on its extended Gauss code. For example, both the knotoid diagram and the knot diagram in Figure 24 correspond to the same extended Gauss code. Therefore, when loading a diagram from a Gauss code, one has to additionally specify whether it is open or closed using option `--closure-method=open` (open diagram) or `--closure-method=direct` or `rays` (closed diagram).
This form of extended Gauss codes cannot be used for planar diagrams as the information on which arcs touch the outside region is missing. For example, both knotoids diagrams in Figure 26 correspond to the same extended Gauss code `-1 2 -3 1 -2 3 ---` .
To overcome this limitation, Knoto-ID accepts a modified version of the extended Gauss code, which allows the

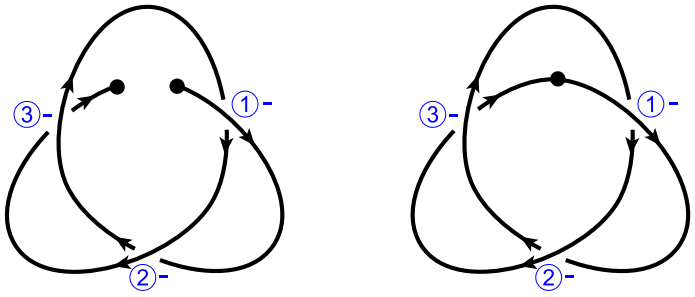

Figure 24: A knotoid diagram on the sphere and a knot diagram corresponding to the extended Gauss code `-1 2 -3 1 -2 3 ---`. Crossing labels and signs are shown in blue.

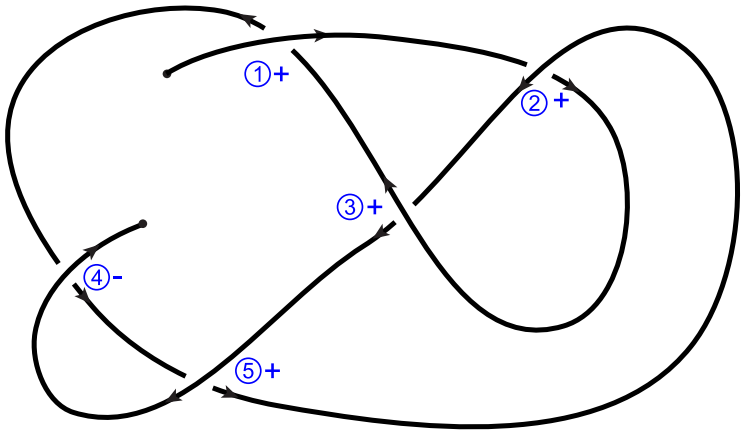

Figure 25: A labeled knotoid diagram corresponding to the extended Gauss code `1 -2 3 -1 -4 -5 2 -3 5 4 +++-+`. Crossing labels and signs are shown in blue.

encoding of planar and spherical knotoid diagrams as well as knot diagrams. Such a code has the following form:

$$\text{Gauss code} = \underbrace{\text{Crossings}}_{\text{first part}} \quad \underbrace{\text{Signs}}_{\text{second part}} \quad \underbrace{\text{Outside arcs}}_{\text{third part}}$$

The **first part** and **second part** encode information on the crossings, as described previously.

The **third part** allows the proper encoding of a planar knotoid diagram. It consists in a list of arc labels corresponding to the arcs of the diagram that are adjacent to the outside region of the diagram. This part has no fixed length. Note that arcs must be labeled with integer numbers, starting from 0 for the first arc and increasing by steps of 1 for each subsequent arc that is met when traveling along the curve.

For example, the knotoid diagram in Figure 27 corresponds to the following extended Gauss code:

```
1 -2 3 -1 -4 -5 2 -3 5 4 +++-+ 0 3 4 5 8 10
```

Notes:

- For diagrams without crossings (Figure 22), the special keyword `empty_diagram` is used.
- When specifying whether a diagram is open or closed (with `--closure-method`) care must be taken to avoid unrealisable diagrams. Two typical sources of mismatch between diagram topology and the choice of closure are:
  - Closing a pure knotoid diagram. For example, the knotoid diagram in Figure 20 with extended Gauss code `1 -2 -1 2 ++ 1 3` cannot be closed without creating additional crossings. Note that *Knoto-ID* may read such a Gauss code without detecting that it does not correspond to realisable knot(oid) diagram. However this will lead to unpredictable output.
  - Closing a planar knotoid diagram with its last endpoint in the outside region. For example, in the knotoid diagram shown in Figure 27, the last endpoint is outside and arc 10 appears in the extended Gauss code

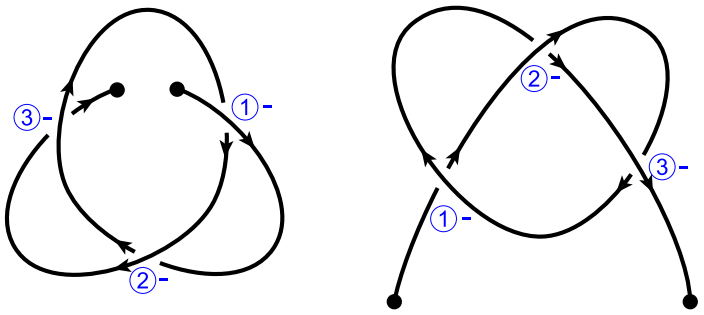

Figure 26: Two knotoid diagrams corresponding to the extended Gauss code `-1 2 -3 1 -2 3 ---`. Crossing labels and signs are shown in blue.

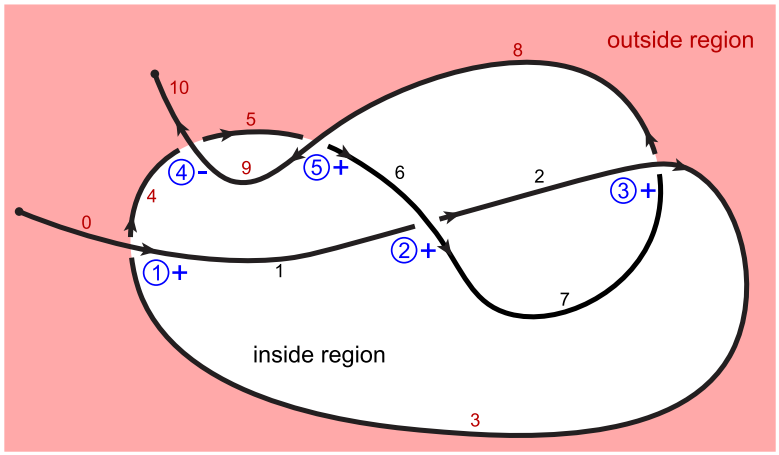

Figure 27: Labeled knotoid diagram corresponding to the extended Gauss code `1 -2 3 -1 -4 -5 2 -3 5 4 +++-+ 0 3 4 5 8 10`. Crossing labels and signs are shown in blue. Arc labels are shown in red for arcs adjacent to the outside region and black for internal arcs.

`1 -2 3 -1 -4 -5 2 -3 5 4 +++-+ 0 3 4 5 8 10`. If we ask *Knoto-ID* to consider this diagram closed, it will complain that arc 10 does not exist (arc 10 is merged arc 0 when closing the diagram).

- While the labeling of arcs is very strict (starting from 0, increasing by steps of 1 when following the curve), labeling of nodes is more flexible. The only requirements are that node labels are non negative integer, and that the label increase when following the curve and meeting a crossing for the time. All the following Gauss code correspond to the diagram shown in Figure 27 (although with different labeling of the crossings):

```
1 -2 3 -1 -4 -5 2 -3 5 4 +++-+ 0 3 4 5 8 10
```

```
0 -1 2 -0 -3 -4 1 -2 4 3 +++-+ 0 3 4 5 8 10
```

but also

```
0 -5 7 -0 -10 -12 5 -7 12 10 +++-+ 0 3 4 5 8 10
```

Note that *Knoto-ID* relabels the crossings when loading a diagram from an extended Gauss code. Therefore the crossing labels in output may not correspond to the labels used in input.

- When reading an extended Gauss code, *Knoto-ID* replaces the following characters by blank space: `[`, `]`, `(`, `)`, `{`, `}`, `,` and `;`. As a consequence, the input format is quite flexible and all the following expressions are valid extended Gauss codes representing the knotoid diagram in Figure 26:

```
{1,-2,3,-1,-4,-5,2,-3,5,4} {+++-+} {0,3,4,5,8,10}
```

```
(1 -2 3 -1 -4 -5 2 -3 5 4) (+ + + - +) (0 3 4 5 8 10)
```

```
1,-2,3,-1,-4,-5,2,-3,5,4,+,+,+,-,+,0,3,4,5,8,10
```

Knoto-ID also accept files with multiple diagrams, with one extended Gauss code per line. An example with two diagrams:

```
1 -2 3 -1 2 -3 +++
-1 2 -3 1 -2 3 ---
```

3.5.4 Distribution of polynomials

Output format used to characterize the distribution of polynomials obtained with multiple projections. This is a tab separated format with 2 mandatory columns (frequency and polynomial) and 1 optional column (knot or knotoid type). The first row is a header (starting with #) specifying the label of each column. In each subsequent row, the entry in column frequency is the fraction of all projections that gave the corresponding polynomial, followed optionally by the knot or knotoid type corresponding to the polynomial. Note that if weights were associated with the projections[86], the column frequency contains the weighted fraction.
Example, without knot type:

```
#frequency  polynomial
0.9         - A^(-16) + A^(-12) + A^(-4)
0.1         + 1
```

Example, with knot type:

```
#frequency  knot_type  polynomial
0.7         3_1R       - A^(-16) + A^(-12) + A^(-4)
0.3         0_1        + 1
```

3.5.5 List of projections with polynomials and knot(oid) diagrams

Tab separated output format with 5 columns (x, y, z coordinates of the projection direction, polynomial invariant and knot(oid) diagram) and 1 optional column (knot or knotoid type). The first row is a header (starting with #) specifying the label of each column. With the optional column for the knot(oid) type, each subsequent row corresponds to a single projection direction (columns 1 to 3) together with the knot(oid) type (column 4), the polynomial (column 5) and PD code[87] or extended Gauss code[88] for the knot(oid) diagram (column 6) obtained from this projection.
Without the optional column for the knot(oid) type, the polynomial is given in column 4 and the PD code or extended Gauss code for the knot(oid) diagram is in column 5.
Example (without knot(oid) names):

```
#projection.x  projection.y  projection.z  polynomial                  PD_code
0.483595       -0.85801      0.173072      - A^(-16) + A^(-12) + A^(-4)  PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
0.521928       -0.578265     0.627058      - A^(-16) + A^(-12) + A^(-4)  PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
0.898137       0.241849      0.367231      - A^(-16) + A^(-12) + A^(-4)  PD[X[7,0,8,1],X[4,2,5,1],X[2,6,3,5],X[6,4,7,3]];
-0.624015      0.596182      0.505146      - A^(-10) + A^(-6) + A^(-4)   PD[X[0,3,1,2],X[3,2,4,1]];
0.134038       -0.988518     -0.0697575    - A^(-16) + A^(-12) + A^(-4)  PD[X[5,1,6,0],X[1,5,2,4],X[2,8,3,7],X[6,4,7,3]];
```

Example (with knot(oid) names):

```
#projection.x projection.y projection.z knotoid_type polynomial                  PD_code
0.483595      -0.85801     0.173072     k3.1o       - A^(-16) + A^(-12) + A^(-4)  PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
0.521928      -0.578265    0.627058     k3.1o       - A^(-16) + A^(-12) + A^(-4)  PD[X[3,1,4,0],X[1,5,2,4],X[5,3,6,2]];
0.898137      0.241849     0.367231     k3.1o       - A^(-16) + A^(-12) + A^(-4)  PD[X[7,0,8,1],X[4,2,5,1],X[2,6,3,5],X[6,4,7,3]];
-0.624015     0.596182     0.505146     k2.1        - A^(-10) + A^(-6) + A^(-4)   PD[X[0,3,1,2],X[3,2,4,1]];
0.134038      -0.988518    -0.0697575   k3.1o       - A^(-16) + A^(-12) + A^(-4)  PD[X[5,1,6,0],X[1,5,2,4],X[2,8,3,7],X[6,4,7,3]];
```

Example with extended Gauss code instead of PD code:

```
#projection.x projection.y projection.z knotoid_type polynomial                  gauss_code
0.483595      -0.85801     0.173072     k3.1o       - A^(-16) + A^(-12) + A^(-4)  1 -2 3 -1 2 -3 +++
0.521928      -0.578265    0.627058     k3.1o       - A^(-16) + A^(-12) + A^(-4)  1 -2 3 -1 2 -3 +++
0.898137      0.241849     0.367231     k3.1o       - A^(-16) + A^(-12) + A^(-4)  1 2 -3 4 -2 3 -4 -1 -+++
-0.624015     0.596182     0.505146     k2.1        - A^(-10) + A^(-6) + A^(-4)   -1 2 1 -2 ++
0.134038      -0.988518    -0.0697575   k3.1o       - A^(-16) + A^(-12) + A^(-4)  1 -2 -3 4 2 -1 -4 3 ++++
```

[86] see section "3.5.6 List of projections" for more details.
[87] see section "3.5.2 Knot(oid) diagrams (PD code)" for more information on the file format.
[88] see section "3.5.3 Knot(oid) diagrams (Extended Gauss code)" for more information on the file format.

3.5.6 List of projections

Tab (or space) separated input format with 3 mandatory columns (x, y, z coordinates of the projection direction) and 1 optional column (weight) The first row can optionally contain a header starting with #. Each subsequent row defines one projection direction with an optional weight (set to 1 if missing). The weight is used when evaluating the distribution of polynomials (the fraction of projections is replaced by a weighted fraction of projections)
Example:

```
#projection.x  projection.y  projection.z  weight
0              0             1             1
0              1             0             1
1              0             0             1
0              0             -1            1
0              -1            0             1
-1             0             0             1
```

3.5.7 List of knot(oid) names

Tab separated input format with 2 columns used to specify the correspondence between polynomials and knot(oid) names. Each row contains a knot or knotoid name in the first column and the corresponding polynomial in the second column. While any string are accepted as a knot or knotoid name (as long as it does not contain a tab character), the polynomials must conform to the "Polynomial" format described in section "3.5.8 Polynomials".
Two rows may contain the same polynomial with a different knot(oid) names. All knot(oid) names corresponding to the same polynomial will be concatenated with a `|` separator. For example, knots 6_2R and 12n_0025L have the same classical Jones polynomial $-1 + A^{-20} - 2A^{-16} + 2A^{-12} - 2A^{-8} + 2A^{-4} + A^4$ and the corresponding knot name will be `6_2R|12n_0025L`.
Example of list of knot names:

```
0_1     1
3_1R    -A^(-16) + A^(-12) + A^(-4)
3_1L    A^4 + A^12 - A^16
4_1     1 + A^(-8) - A^(-4) - A^4 + A^8
5_1R    -A^(-28) + A^(-24) - A^(-20) + A^(-16) + A^(-8)
5_1L    A^8 + A^16 - A^20 + A^24 - A^28
5_2R    -A^(-24) + A^(-20) - A^(-16) + 2*A^(-12) - A^(-8) + A^(-4)
5_2L    A^4 - A^8 + 2*A^12 - A^16 + A^20 - A^24
6_1R    2 + A^(-16) - A^(-12) + A^(-8) - 2*A^(-4) - A^4 + A^8
6_1L    2 + A^(-8) - A^(-4) - 2*A^4 + A^8 - A^12 + A^16
6_2R    -1 + A^(-20) - 2*A^(-16) + 2*A^(-12) - 2*A^(-8) + 2*A^(-4) + A^4
6_2L    -1 + A^(-4) + 2*A^4 - 2*A^8 + 2*A^12 - 2*A^16 + A^20
6_3     3 - A^(-12) + 2*A^(-8) - 2*A^(-4) - 2*A^4 + 2*A^8 - A^12
```

3.5.8 Polynomials

Polynomials are used both as input[89] and output[90].
Classical Jones polynomial and Jones polynomial for knotoids are univariate Laurent polynomials in variable A with integer coefficients. Turaev loop bracket polynomial is a bivariate Laurent polynomial in variables A and v with integer coefficients. See section "4.4 Computing polynomial invariants" for more details.
For the file format, we consider the more general case of bivariate Laurent polynomial with integer coefficients. It should be written as a sum of monomials:

$$\sum_{n,m} C_{n,m} A^n v^m \text{ with } n, m, C_{n,m} \in \mathbb{Z}$$

Each monomial must be written using operators `*` for multiplication and `^` for exponentiation:

```
2*A^2*v^8
```

For clarity, exponents can be written between brackets

[89] section "3.5.7 List of knot(oid) names"
[90] sections "3.5.4 Distribution of polynomials", "3.5.5 List of projections with polynomials and knot(oid) diagrams".

```
2*A^(-4)*v^(2)
```

but it is not mandatory:

```
2*A^-4*v^2
```

will also be interpreted as $2A^{-4}v^2$.
Terms with exponent 0 can be replaced by 1. For example `2*A^(-4)*v^(0)` can be replaced by `2*A^(-4)` and `2*A^(0)*v^(0)` can be replaced by `2`.
Exponent 1 can be removed. For example `2*v^(1)` can be replaced by `2*v`.
Coefficient 1 can be removed for terms with non-zero exponent. For example `1*A^(-4)*v^(8)` can be replaced by `A^(-4)*v^(8)` and `1*A^(-4)` can be replaced by `A^(-4)`.
Finally, monomials must be combined with addition and subtraction operators (`+` and `-`):

```
1 + A^(-8) - A^(-4) - A^4 + A^8
```

The first monomial can be prefixed by a unary sign operator

```
-A^(-16) + A^(-12) + A^(-4)
```

Note that the combination of the binary addition or substraction operators with unary sign operator is not allowed: `+ -A^2`, `A^2 + -v^4` or `A^2 - -v^4` will all generate an error.
Space character is allowed and can be used to improve readability but it is ignored.
Examples of valid polynomials:

```
1
```

```
-A^(-4) - 4*A^(-2)*v
```

```
-A^2*v - A^4
```

```
-A^-10 + A^-6 + A^-4
```

```
1 + A^(-2) + v + A^(2)
```

```
+A^(-8) + A^(-6)*v^(-4) - A^(-2)*v
```

3.5.9 List of subchains

Output format used by `knotted_core` program to output the dominant knot(oid) of subchains of the input curve. This is a tab separated format with 5 mandatory columns (first and last indices of the subchain, length of the subchain, frequency and polynomial) and 1 optional column (Knot or knotoid type). The first row is a header (starting with #) specifying the label of each column. Each subsequent row corresponds to one subchain, defined by its first and last indices (using zero-based indexing) as well as the length of the subchain (number of points). In addition, the corresponding dominant polynomial is given, together with its frequency (fraction of all projections that have the corresponding polynomial). Optionally, the Knot or knotoid type corresponding to the polynomial can also be given. Each subchain is defined by a first index and a length. Although not necessary to define the subchain, its last index is also given for completeness. Given a first index i_1, a length l and a last index $i_2 := i_1 + l - 1 \bmod N$, the subchain is defined by the ordered set of points of the input curve with indices:

$$\{i_1 \bmod N, (i_1+1) \bmod N, \cdots, (i_1+l-1) \bmod N\}$$

where N is the number of points in the input curve (See Figures 28 and 29).
Note: by definition, $l > N$ corresponds to the full input curve closed with direct closure.
Example for an open chain, without knot(oid) type:

```
#index_first  index_last  length  frequency  polynomial
11            51          41      0.7        - A^(-10) + A^(-6) + A^(-4)
12            52          41      0.75       - A^(-16) + A^(-12) + A^(-4)
13            52          40      0.85       - A^(-16) + A^(-12) + A^(-4)
14            52          39      0.75       - A^(-16) + A^(-12) + A^(-4)
```

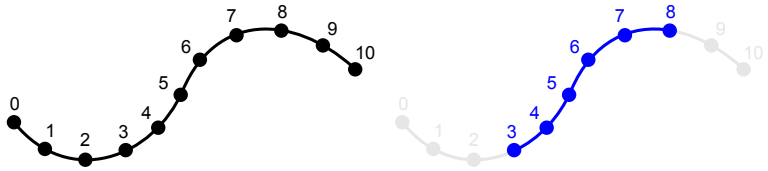

Figure 28: Left: open input curve with $N = 11$ points. Right: a subchain with first index $i_1 = 3$, length $l = 6$ and last index $i_2 = 8$ (blue).

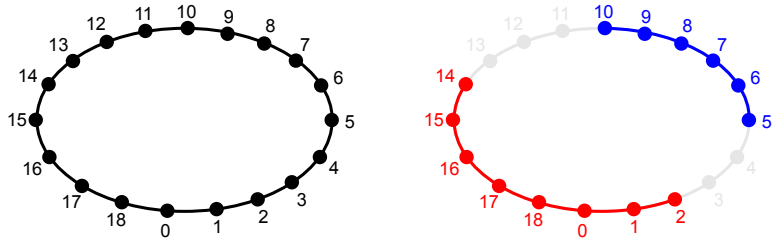

Figure 29: left: closed input curve with $N = 19$ points. Right: a subchain with first index $i_1 = 5$, length $l = 6$ and last index $i_2 = 10$ (blue), and a subchain with first index $i_1 = 14$, length $l = 8$ and last index $i_2 = 2$ (red).

Example for a closed chain (with $N = 111$ points), with knot type:

#index_first	index_last	length	frequency	knot_type	polynomial
22	110	89	0.8	3_1R	- A^(-16) + A^(-12) + A^(-4)
22	0	90	0.5	3_1R	- A^(-16) + A^(-12) + A^(-4)
23	1	90	0.6	3_1R	- A^(-16) + A^(-12) + A^(-4)
22	79	58	0.8	3_1R	- A^(-16) + A^(-12) + A^(-4)
23	23	112	1	3_1R	- A^(-16) + A^(-12) + A^(-4)

Note that the length of last subchain is bigger than the number of input points, therefore the last subchain corresponds to the full input curve (with direct closure).

3.5.10 xyz format for knot(oid) diagrams

Tab (or space) separated output format with 4 columns corresponding to x, y, z coordinates and a label. The first row is a header (starting with #) specifying the label of each column. Each subsequent row corresponds to a point of the curve. Consecutive points (rows) will be connected by a straight line. Note that consecutive lines may have the same x, y, z coordinates. This format is used by `convert_diagram` to output planar represenations of knot(oid) diagrams in the xy plane. The z coordinate ranges from -1 to +1 and is used to specify which arc is above ($z > 0$) or below ($z < 0$) at each crossing.
The following example corresponds to a knot diagram with two arcs and one crossing:

```
#x y z label
0.728017   0.387851      0.698611   Arc_0
0.960894   0.511916      0.358889   Arc_0
1.02482    0.255915      0          Arc_0
1.02482    0.255915      0          Arc_0
1.08874    -8.68623e-05  -0.358889  Arc_0
0.824876   -6.58107e-05  -0.698611  Arc_0
0.824876   -6.58107e-05  -0.698611  Crossing_0
0          0             -1         Crossing_0
-0.824876  3.29053e-05   -0.698611  Crossing_0
-0.824876  3.29053e-05   -0.698611  Arc_1
-1.08874   4.34311e-05   -0.358889  Arc_1
-1.02481   -0.255956     0          Arc_1
-1.02481   -0.255956     0          Arc_1
-0.960873  -0.511954     0.358889   Arc_1
-0.728001  -0.38788      0.698611   Arc_1
-0.728001  -0.38788      0.698611   Crossing_0
0          0             1          Crossing_0
0.728017   0.387851      0.698611   Crossing_0
```

4 Theory

4.1 Knots

A knot is a smooth embedding of a circle into $\mathbb{R}^3$, considered up to continuous deformations. To be more precise, a knot is a closed curve in 3-dimensional Euclidean space that does not intersect itself anywhere and can be continuously deformed as if it was made of rubber [30]. Knots are usually studied through their diagrams, which are abstract and schematized pictures of a knot composed of curves in the plane that cross transversely in 4-fold vertices. Each vertex is equipped with extra structure in the form of a deleted segment that indicates the under crossing line and they are called *crossings* of the knot. With this information, one is always able to reconstruct the knot in 3-space (see Fig. 30).

Figure 30: A crossing (a) and a knot diagram (b).

Two knots are equivalent, that is they can be deformed to one another, if and only if their diagrams can be transformed to one another by a finite sequence of three elementary diagram moves called the Reidemeister moves (see Fig. 32) along with planar isotopy. Planar isotopy is a motion of the diagram in the plane that preserves the underlying structure. These tools work efficiently well for distinguishing knots with small number of crossings, however, for knots with higher number of crossings (see Fig. 31) more sophisticated tools are required.

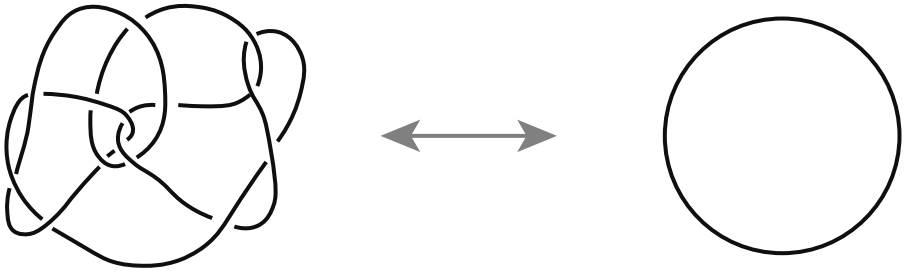

Figure 31: Louis Kauffman's unknot. This rather complicated knot is actually equivalent to the trivial knot.

Such tools are the so-called knot invariants and are functions defined on the set of all knots and links that assign the same value to equivalent knots or links. In this work we focus on the Jones polynomial [23] and its extension to the case of equivalence classes of open ended diagrams called knotoids [1, 2].

4.2 Knotoids

Consider that we started drawing a knot diagram and at some point in this process we decided to leave the diagram open, i.e. not to connect the end points. The question now is if this allows the definition of open knot diagrams. It is not difficult to see that if we apply planar isotopy and the Reidemeister moves as described above to such a diagram, eventually we will always manage to unknot it. However, if we require the Reidemeister moves to be applied only at local neighbourhoods of the diagram that don't involve the endpoints we are getting closer to the notion of open knotting. Additionally, if we forbid the endpoints to slide over or under the rest of the diagram (see Fig. 33), then we have achieved our goal.

What we have just described is a *knotoid diagram*. Knotoid diagrams generalize the notion of a 1-1 tangle (or a long knot) since they allow the endpoints to be in different regions of the diagram, and thus they provide a rigorous definition for open knots. Their equivalence classes under the forbidden moves as well as the Reidemeister moves away from the endpoints are called *knotoids* (see Fig. 34). Knotoids were first introduced by V. Turaev in [1] and have been studied further by L. Kauffman and N. Gügümcü in [2]. Formally, a knotoid is defined as the generic immersion of the closed unit interval in the interior of the surface whose only singularities are transversal double points endowed with over/undercrossing data. In analogy to the case of knots, the double points of the knotoid diagram are also called crossings. The images of 0 and 1 under this immersion are called the *tail* and the *head* of the knotoid diagram,

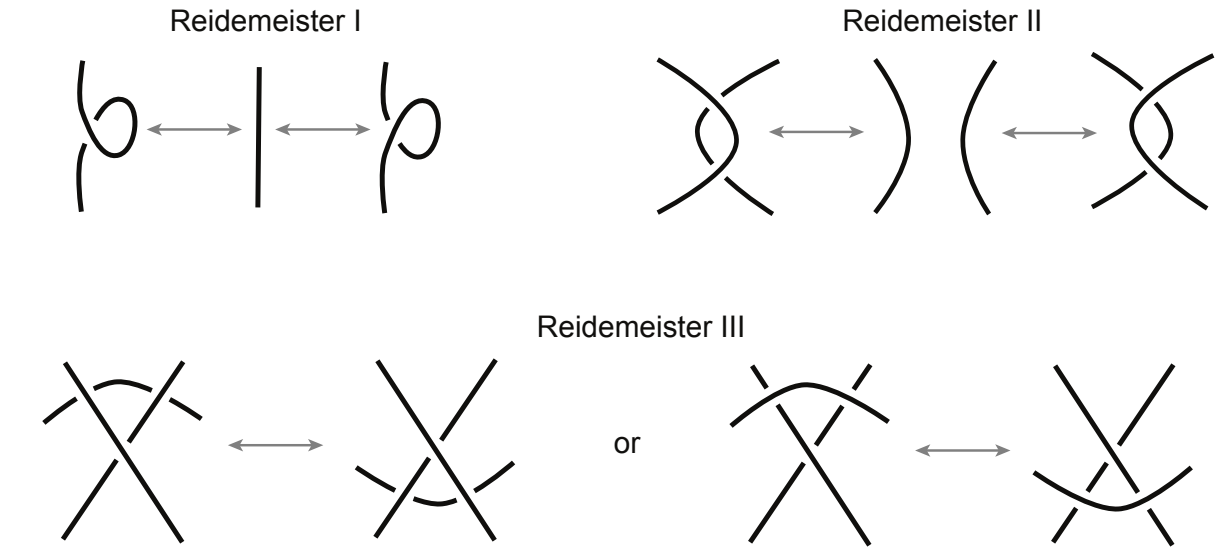

Figure 32: The three Reidemeister moves.

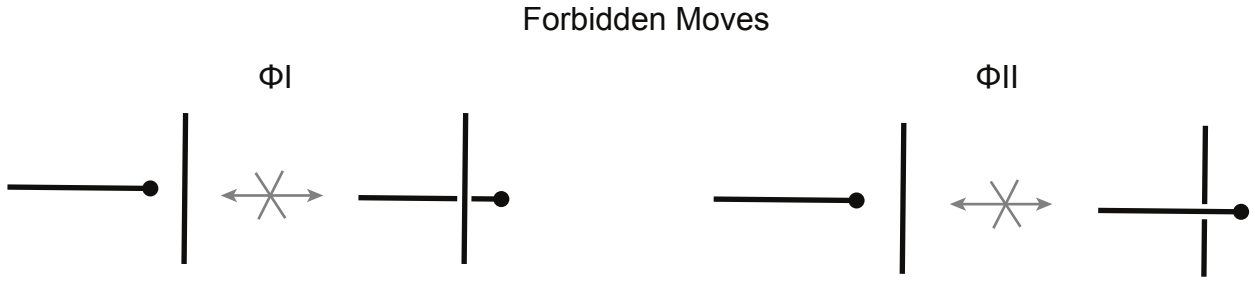

Figure 33: The two forbidden moves.

respectively, they are distinct from each other and from the double points. Every knotoid diagram comes with an orientation the goes from the tail to the head [1]. Knotoids that have both ends in the same region of the diagram are called *knot-type* knotoids while knotoids that have their end points in different regions of the diagram are called *proper* knotoids.

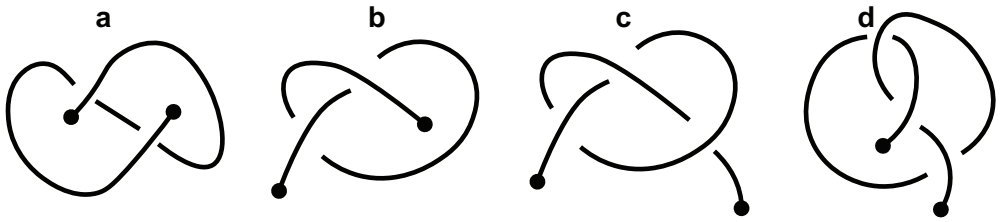

Figure 34: Examples of knotoids. The diagram (c) corresponds to a knot-type knotoid while (a), (b) and (d) to proper knotoids.

Knotoids are defined on an oriented surface but they are usually studied on the surface of the 2-sphere S^2 but their definition can also be extended to the plane $\mathbb{R}^2$. We shall call this class of knotoids *planar*. There are pairs non-isotopic planar knotoids that become isotopic once we consider them as knotoids on S^2. For example, in Figure 34, diagrams (a) and (b) are not equivalent as planar knotoids but they become equivalent once they are considered in S^2. This is because on the sphere one has the freedom to move arcs using isotopy towards/over the poles and around the sphere in order to simplify the diagram, something that is not possible while working on the plane.
One can recover a knot diagram from a knotoid diagram k by embedding an arc α that passes everywhere under k and connect the endpoints of k to α. The arc is called a *shortcut* for k and the way of closure is called the *underpass* closure (See Figure 35). The shortcut is unique for every knotoid diagram, up to isotopy. In an analogous way one can define the *overpass closure* [1, 2]. On the other hand, every knot may be presented by a knotoid diagram by cutting out an underpassing arc [1].

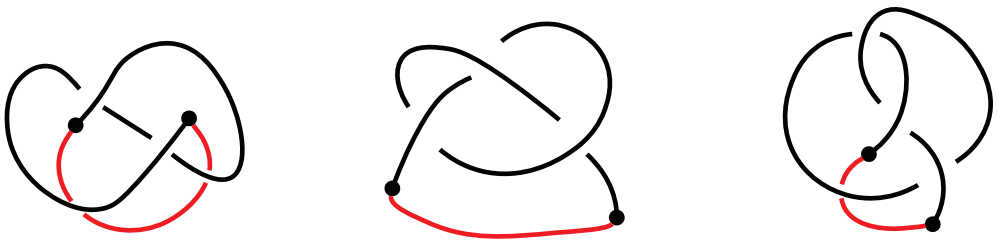

Figure 35: Knotoids to knots using the underpass closure. The embedded arc α is shown in red.

4.3 Knotoids and curves in space

Consider an embedded open curve in space. We would like to know if it is entangled or not. One may suggest to project the curve on a plane and study its projection as a knotoid diagram and they would be partially right. However, how can we be sure that by accidentally perturbing a bit the curve we don't mess with its topology? In order to avoid situations like this we first introduce two infinite lines that pass through the endpoints of the curve. In this way all manipulations of the curve in space with respect to these two lines preserve the topology of the curve [2]. Finally, the curve is projected on an orientable surface (e.g. a sphere, a plane, etc.), and thus a knotoid diagram is obtained. Note now that different choices of projection planes will possibly lead to different knotoid diagrams. For this reason, in order to study the topology of an embedded open curve we work as follows. We assume that the curve lies inside a large enough sphere (a radius of twice the length of the curve will do fine for most of the cases). Each point of the sphere corresponds to a vector that points towards an orientable surface that lies outside of the sphere (see Figure 36). For each choice of the projection surface we introduce the infinite parallel lines and we take the projection of the curve together with the information of over/under crossing arc at each double point. The knotoid type is then evaluated using an invariant for knotoids. Thus, the entanglement of the embedded open curve is a probability distribution that we can approximate by sampling the sphere [3].

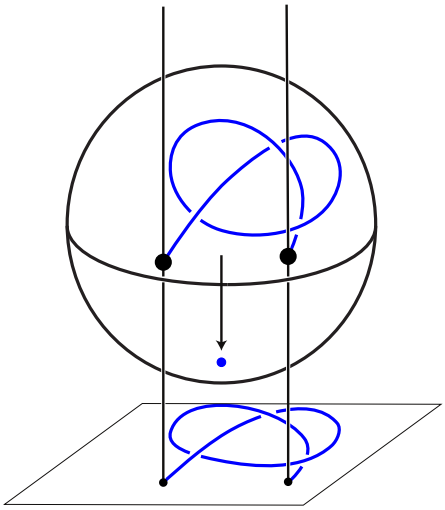

Figure 36: The curve is placed inside a large sphere. The blue dot indicates a vector that points towards an oriented surface of projection (plane or sphere). The two infinite parallel lines are introduced and the curve is projected on the chosen oriented surface.

The equivalent of the over/underpassing closure for the case of embedded open curves is most probably the uniform (or stochastic) closure technique [31, 5, 32–34]. Here the curve is placed again inside a large ball, only this time each point of the sphere corresponds to a closure direction. The closure is achieved by extending two parallel rays, each originating from one of the endpoints of the curve, towards that the chosen closure direction and they are connected outside of the sphere (see Figure 35). This method is, as hinted by its name, also probabilistic and so the knot-type of the core is a probability distribution of all knot-types obtained by closing over all possible directions that are defined by the points of the sphere. There is however the option to close the open curve using an arc that directly connects the endpoints [26, 35] (direct closure method). This method is computationally faster but it may interfere with the topology of the chain by introducing or removing crossings. In contrast to the direct closure technique, the stochastic closure provides more detailed overview but it is computationally more demanding as one is required to sample a probability distribution in order to understand the topology of the studied object.

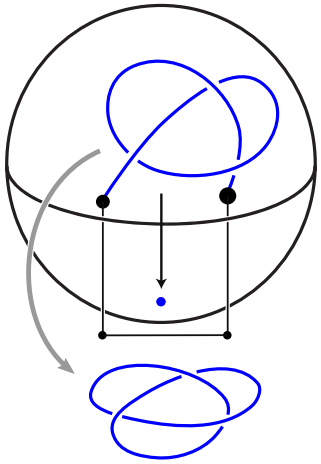

Figure 37: The uniform (or stochastic) closure technique. Two parallel rays are extended towards the closure direction indicated by the blue point and they are connected outside of the sphere. The resulting knot may be evaluated using a knot invariant.

4.4 Computing polynomial invariants

In this section we will present the Kauffman bracket [36] and how it gives rise to the classical Jones polynomial for knots [23], the Jones polynomial for knotoids [1, 2] and the Turaev loop bracket polynomial for planar knotoids [1]. Consider a knot K and observe that a each crossing $\times$ of K can be smoothed in two different ways. The first way is to smooth it horizontally $=$ and the second is to smooth it vertically $)($. If K has n crossings then there are 2^n possible smoothings of K. With these in mind we define the Kauffman bracket as the 1-variable polynomial $\langle K \rangle \in \mathbb{Z}[A, A^{-1}]$ that satisfies the following axioms:

$$\langle \times \rangle = A\langle = \rangle + A^{-1} \langle)(\rangle \tag{1}$$
$$\langle K \sqcup \bigcirc \rangle = \left(-A^2 - A^{-2}\right) \langle K \rangle \tag{2}$$
$$\langle \bigcirc \rangle = 1 \tag{3}$$

The first axiom gives the smoothing rule on a local region of the knot diagram. This means that all three diagrams in 9 are identical everywhere except in the region that is shown inside the brackets. The second axiom tells us that a disjoint circle from the rest of the diagram multiplies the diagram by $(-A^2 - A^{-2})$. The last axiom is the basis of the inductive process. Note that the Kauffman bracket polynomial is invariant under the second and the third Reidemeister moves but not under the first Reidemeister move. In order to establish invariance under the first move, we choose an orientation for the knot diagram, i.e. we assign a direction indicated by arrows on its arcs. The crossings in an oriented diagram are given signs of ± 1 (see Fig. 38)

Figure 38: A positive crossing with sign $+1$ (a) and a negative crossing with sign -1 (b).

The sum of signs of all crossings of a knot diagram K is called *the writhe* of K, $\mathrm{wr}(K)$, and it will allow us to make the Kauffman bracket invariant under the first Reidemeister move. Indeed, by considering the following normalization of the Kauffman bracket, we have achieved our goal.

$$f_K(A) = (-A^3)^{-\mathrm{wr}(K)} \langle K \rangle, \tag{4}$$

where $\langle K \rangle$ is the Kauffman bracket polynomial of K. It turns out the $f_K(A)$ coincides with the classical Jones polynomial.
One may extend the definition of the Kauffman bracket and, in turn of the classical Jones Polynomial, to the case of knotoids. The rules are more or less the same, with some minor tweaks depending on whether we want compute the invariant for a knotoid in S^2 or a planar knotoid. Knotoids are open ended diagrams and therefore when one smoothes all crossings of the diagram, in the end they will always end up with a number of disjoint circles and a

single long segment. We choose the long segment to be the basis of the inductive procedure and so the axioms for a knotoid diagram k become as follows:

$$\langle \times \rangle = A\langle \asymp \rangle + A^{-1} \langle)(\rangle \tag{5}$$

$$\langle k \sqcup \bigcirc \rangle = \left(-A^2 - A^{-2}\right) \langle k \rangle \tag{6}$$

$$\langle \sim \rangle = 1 \tag{7}$$

The invariance unde the first Reidemeister move is taken care by the same normalization as with the case of knots.

$$J_k(A) = (-A^3)^{-\mathrm{wr}(k)} \langle k \rangle, \tag{8}$$

where $\langle k \rangle$ is the Kauffman bracket polynomial of a knotoid diagram k and $\mathrm{wr}(k)$ is the writhe of k. Equation 8 is the extension of the classical Jones for the case of knotoids in S^2. The case of planar knotoids requires some more explanation. As mentioned above when one smoothes all crossings of a knotoid diagram, they will end up with a number of disjoint circles and one long segment. When we work on the surface of a 2-sphere, it doesn't matter whether the long segment rests inside a circle or not because, even if it does, we can always take it out by moving an arc of the circle around the surface of the sphere. If we work with planar knotoids, we cannot do this and so we want to keep track of this information, i.e. if the long segments is inside a circle or not. For this reason, we add an extra variable to the Kauffman bracket polynomial that counts the number of circles that enclose the long segment in each possible smoothing of the knotoid diagram k.

$$\langle\langle \times \rangle\rangle = A\langle\langle \asymp \rangle\rangle + A^{-1} \langle\langle)(\rangle\rangle \tag{9}$$

$$\langle\langle K \sqcup \bigcirc \rangle\rangle = \left(-A^2 - A^{-2}\right) \langle\langle K \rangle\rangle \tag{10}$$

$$\langle\langle \; \overset{k}{\bigcirc\!\!\sim} \; \rangle\rangle = v^k \tag{11}$$

$$\langle\langle \sim \rangle\rangle = 1 \tag{12}$$

This version of the bracket is a Laurent polynomial in $\mathbb{Z}[A, A^{-1}, v]$. The invariance under the first Reidemeister move is given again by the same normalization:

$$\widehat{J}_k(A, v) = (-A^3)^{-\mathrm{wr}(k)} \langle\langle k \rangle\rangle \tag{13}$$

Equation 13 comprises the Turaev loop bracket polynomial, which is the extension of the classical Jones polynomial to the case of planar knotoids.

4.5 Notation Conventions

Knots have been tabulated in terms of this number of crossings [37, 30]. The standard notation is in the form X_Y or X_Y, where X is the number of crossings of the knot diagram in question and Y corresponds to the position of the knot in the table among knots with the same number of crossings. For example, 5_2 or 5_2 denotes the second knot in the knot table that has five crossings. An analogous notation for knotoids is being implemented here which is of the form: $kX.Y$, where X is the number of crossings of the knotoid diagram in question and Y corresponds to the position of the knot in the table among knotoids with the same number of crossings. An "m" is added at the end of a knotoid name when its mirror reflection is considered. The mirror reflection transforms a knotoid into a knotoid represented by the same diagrams with overpasses changed to underpasses and vice versa [1]. For example $k1.1m$ is the mirror reflection of the knotoid $k1.1$. Additionally, we use * to denote the product of two knotoids, for instance $k2.1$*$k3.1$ is the product between a $k2.1$ and a $k3.1$ knotoid. Note that at the moment only partial tables for knotoids exist [38]. Throughout the development of this software we have identified many new knotoids and expanded the list.

4.6 Knotted proteins, slipknots and knotted cores

It is a long standing belief that folded configurations of proteins should provide an insight on the folding pathways that the backbone follows to reach its native state [39, 40]. In principle, proteins tend to avoid folding into configurations that involve non-trivial topological features such as knots. At a first glance, it may appear as if nature selects with a bias against the formation of knots. However, their existence [26] tells us that this may not be the case as knots have been observed to contribute in the stability as well as the function of a protein.
A fingerprint matrix is a triangular matrix where each entry (i, j) corresponds to a subchain with starting index i and ending index j, while it carries also the information of the dominant knotoid or knot type of its corresponding subchain and it is encoded using a colour scale. Moreover, the whole chain corresponds to the lower left part of the matrix and,

therefore, the fingerpring matrix provides a detailed overview of the global as well as the local topology of an open chain. The knotted core is defined as the shortest subchain obtained by progressively altering the length of the whole chain by 1 point without changing the knot(oid) type in the process. Visually, the knotted core is the point of the path connected region that includes the full chain and is closer to the diagonal of the fingerprint matrix. In Figure 15 we see the knotoid fingerprint for the protein 3KZN. Note that this definition of the knotted core corresponds to the "top-down" knotted core discussed by Tubiana and coauthors [24].
In open chains the ending index has to be greater than the starting index which leads to the empty upper part of the matrix and thus to the triangular shape of the fingerprint. On the other hand, in closed chains there is not such a restriction [6, 41]. Thus disk matrices provide an ideal representation of the topology closed chains. In this case we use polar coordinates (r, θ) to read the matrices, where r corresponds to the length of the subchain and θ corresponds to the midpoint of the subchain (see Figure 16). In analogy to the fingerprint matrix, the knotted core is the point of the path connected region that includes the full chain and is closer to the center of the disk matrix.
Apart from knotoids and knots, slipknots also seem to provide valuable information on protein folding [4]. Slipknotting appears when a part of the protein backbone forms a knot and then it doubles back so that a non-minimal diagram of the unknot is formed [4]. Slipknots can be also generalized to the case of knotoids, where the only difference lies in the fact that the chain may also correspond to a non-minimal non-trivial knotoid. In order to detect slipknots, one has to study all possible subchains of the initial chain. Each subchain is evaluated for the knot or the knotoid type it posseses and the results are summarized in the fingerprint [4, 5, 3] or disk matrix [6]. A slipknot can then be visualized as a region of the matrix that corresponds to a non-trivial knot(oid) and is disconnected from the region containing the full chain.

References

[1] Vladimir Turaev. Knotoids. *Osaka J. Math.*, 49(1):195–223, 2012.

[2] Neslihan Gügümcü and Louis H Kauffman. New invariants of knotoids. *European Journal of Combinatorics*, 65:186 – 229, 2017.

[3] Dimos Goundaroulis, Julien Dorier, Fabrizio Benedetti, and Andrzej Stasiak. Studies of global and local entanglements of individual protein chains using the concept of knotoids. *Scientific Reports*, 7:6309, 2017.

[4] Neil P. King, Eric O. Yeates, and Todd O. Yeates. Identification of rare slipknots in proteins and their implications for stability and folding. *Journal of Molecular Biology*, 373(1):153–166, 2007.

[5] Joanna I Sułkowska, Eric J Rawdon, Kenneth C Millett, Jose N Onuchic, and Andrzej Stasiak. Conservation of complex knotting and slipknotting patterns in proteins. *Proceedings of the National Academy of Sciences*, 109(26):E1715–E1723, 2012.

[6] Eric J Rawdon, Kenneth C Millett, and Andrzej Stasiak. Subknots in ideal knots, random knots, and knotted proteins. *Scientific Reports*, 5:8928, 2015.

[7] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[8] https://www.khronos.org/webgl/.

[9] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. https://www.R-project.org/.

[10] Trevor L Davis. Some documentation, examples ported from Allen Day's getopt package. Some documentation from the optparse Python module by the Python Software Foundation. Contributions from Steve Lianoglou, Jim Nikelski, Kirill Müller, Peter Humburg, and Rich FitzJohn. *optparse: Command Line Option Parser*, 2017. https://CRAN.R-project.org/package=optparse.

[11] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. http://ggplot2.org.

[12] Erich Neuwirth. *RColorBrewer: ColorBrewer Palettes*, 2014. https://CRAN.R-project.org/package=RColorBrewer.

[13] Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. http://www.jstatsoft.org/v21/i12/.

[14] Kai Habel, Raoul Grasman, Robert B. Gramacy, Andreas Stahel, and David C. Sterratt. *geometry: Mesh Generation and Surface Tesselation*, 2015. https://CRAN.R-project.org/package=geometry.

[15] Robert J. Hijmans. *geosphere: Spherical Trigonometry*, 2017. https://CRAN.R-project.org/package=geosphere.

[16] Daniel Adler, Duncan Murdoch, and others. *rgl: 3D Visualization Using OpenGL*, 2017. https://CRAN.R-project.org/package=rgl.

[17] JJ Allaire, Yihui Xie, Jonathan McPherson, Javier Luraschi, Kevin Ushey, Aron Atkins, Hadley Wickham, Joe Cheng, and Winston Chang. *rmarkdown: Dynamic Documents for R*, 2017. https://CRAN.R-project.org/package=rmarkdown.

[18] Julien Idé. *Rpdb: Read, Write, Visualize and Manipulate PDB Files*, 2017. https://CRAN.R-project.org/package=Rpdb.

[19] *Pandoc, a universal document converter*. http://pandoc.org/.

[20] Dimos Goundaroulis, Neslihan Gügümcü, Sofia Lambropoulou, Julien Dorier, Andrzej Stasiak, and Louis H Kauffman. Topological Models for Open-Knotted Protein Chains Using the Concepts of Knotoids and Bonded Knotoids. *Polymers*, 9(9):444, 2017.

[21] Gopalakrishnan Vijayan and Avi Wigderson. Planarity of Edge Ordered Graphs. Technical Report 307, Department of Electrical Engineering and Computer Science, Princeton University, 1982.

[22] Dashuang Shi, Xiaolin Yu, Lauren Roth, Hiroki Morizono, Mendel Tuchman, and Norma M. Allewell. Structures of n-acetylornithine transcarbamoylase from xanthomonas campestris complexed with substrates and substrate analogs imply mechanisms for substrate binding and catalysis. *Proteins: Structure, Function, and Bioinformatics*, 64(2):532–542, 2006.

[23] Vaughan F. R. Jones. Hecke algebra representations of braid groups and link polynomials. *Annals of Mathematics*, 126(2):335–388, 1987.

[24] Luca Tubiana, Enzo Orlandini, and Cristian Micheletti. Probing the entanglement and locating knots in ring polymers: A comparative study of different arc closure schemes. *Progress of Theoretical Physics Supplement*, 191:192–204, 2011.

[25] Kleanthes Koniaris and M. Muthukumar. Self-entanglement in ring polymers. *The Journal of Chemical Physics*, 95(4):2873–2881, 1991.

[26] William R Taylor. A deeply knotted protein structure and how it might fold. *Nature*, 406(6798):916, 2000.

[27] Charles R. Collins and Kenneth Stephenson. A circle packing algorithm. *Computational Geometry*, 25(3):233–256, 2003.

[28] Dror Bar-Natan, Scott Morrison, and et al. The Knot Atlas. http://katlas.org.

[29] Bo˘ stjan Gabrov˘ sek. Tabulation of prime knots in lens spacess. *Mediterranean Journal of Mathematics*, 88(14):24, 2017.

[30] Colin C. Adams. *The Knot Book*. W.H. Freeman, 1994.

[31] Marc L Mansfield. Are there knots in proteins? *Nature Structural & Molecular Biology*, 1(4):213–214, 1994.

[32] Rhonald C Lua and Alexander Y Grosberg. Statistics of knots, geometry of conformations, and evolution of proteins. *PLoS computational biology*, 2(5):e45, 2006.

[33] Kenneth Millett, Akos Dobay, and Andrzej Stasiak. Linear random knots and their scaling behavior. *Macromolecules*, 38(2):601–606, 2005.

[34] Michal Jamroz, Wanda Niemyska, Eric J. Rawdon, Andrzej Stasiak, Kenneth C. Millett, Piotr Sułkowski, and Joanna I. Sulkowska. Knotprot: a database of proteins with knots and slipknots. *Nucleic Acids Research*, 43(D1):D306–D314, 2015.

[35] Peter Virnau, Leonid A Mirny, and Mehran Kardar. Intricate knots in proteins: Function and evolution. *PLoS computational biology*, 2(9):e122, 2006.

[36] Louis H Kauffman. New invariants in the theory of knots. *American Mathematical Monthly*, 95(3):195–242, 1988.

[37] Dale Rolfsen. *Knots and links*. Publish or Perish Press, 1976.

[38] Andrew bartholomew's mathematics page: Knotoids. http://www.layer8.co.uk/maths/knotoids/index.htm.

[39] Gordon M. Crippen. Topology of globular proteins. *Journal of Theoretical Biology*, 45(2):327–338, 1974.

[40] Michael L. Connolly, I. D. Kuntz, and Gordon M. Crippen. Linked and threaded loops in proteins. *Biopolymers*, 19(6):1167–1182, 1980.

[41] David A B Hyde, Joshua Henrich, Eric J Rawdon, and Kenneth C Millett. Knotting fingerprints resolve knot complexity and knotting pathways in ideal knots. *Journal of Physics: Condensed Matter*, 27(35):354112, 2015.