

Persistent Cohomology and Circle-valued coordinates

Dmitriy Morozov Vin de Silva
Mikael Vejdemo-Johansson

July 9, 2009

Coordinatization

Essentially

It's all about finding *coordinate function* on a dataset $X \subseteq \mathbb{R}^d$.
Preferably few coordinates - cognitive tools.

Coordinatization

Essentially

It's all about finding *coordinate function* on a dataset $X \subseteq \mathbb{R}^d$.
Preferably few coordinates - cognitive tools.

Classically

- Linear coordinatization: Find maps $X \rightarrow \mathbb{R}$, concentrating information.
- Principal Component Analysis; Projection pursuit

Coordinationization

Essentially

It's all about finding *coordinate function* on a dataset $X \subseteq \mathbb{R}^d$.
Preferably few coordinates - cognitive tools.

Classically

- Linear coordinationization: Find maps $X \rightarrow \mathbb{R}$, concentrating information.
- Principal Component Analysis; Projection pursuit

Recently

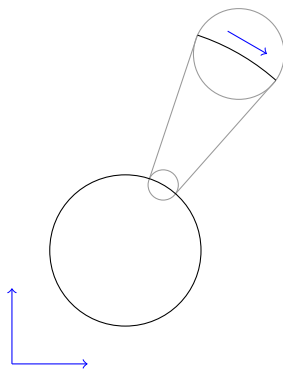
- Nonlinear methods: drop expectation that for f coordinate:
 $f(\lambda x + \mu y) = \lambda f(x) + \mu f(y)$.
- MDS, Kernel methods, Locally linear methods

Problematic cases

Some shapes take up too many coordinates.

Problematic cases

Some shapes take up too many coordinates.



Locally 1-dimensional.

Globally 2 coordinates needed to describe all points.

The shape doesn't fit in $\mathbb{R}$.

How can we fix this?

Circle-valued coordinates

- Use $S^1 = [0, 1]/(0 \sim 1)$ as additional coordinate space
- Fixes the circle
- Fixes the torus
- Occurs naturally:
 - Phase coordinates for waves
 - Angle coordinates for directions

Outline

Problem remains: how do we find circle-valued coordinates?

Circle-valued coordinates and cohomology

Problem remains: how do we find circle-valued coordinates?

Persistent cohomology

- Degree one cohomology equivalent to circle-valued maps
- Persistence picks out relevant features from noise
- Once a feature-rich parameter has been found, we can work in ordinary (non-persistent) cohomology theories

The algorithm we use is a variation on the Persistence algorithm. We introduce simplices one after the other, and reduce a cumulative coboundary matrix.

From cohomology to circle-valued coordinatizations

Use canonical isomorphism

$$H^1(X; \mathbb{Z}) \cong [X, S^1]$$

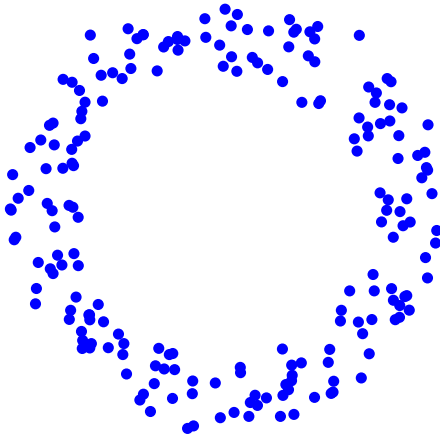
Issues

- Easy to compute: Modular cohomology, coefficients in $\mathbb{F}_p$ for small primes p .
Need for the isomorphism: Integer-valued cohomology.
Smoothness: Integer cohomology gives constant values on all vertices, and wraps edges in the complex around the target circle.
- Numerical stability of cohomology computation and of the smoothing operations.

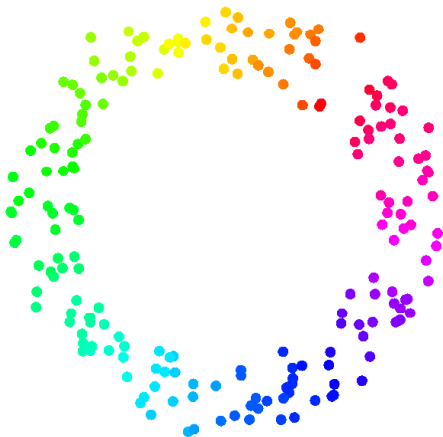
- Integral 1-cocycle: integer weighted edge graph.
- Coordinates found by edge traversal, increasing by edge weights.
- Each cohomologous cocycle guaranteed by cocycle condition to give compatible values, mod 1.0, to each vertex.
- Application straight on integral cocycle yields value 0 at each vertex.
- Given ζ integral cocycle, we wish to find cohomologous cocycle z such that the edges are small.
- Hence, we wish to find x such that $\zeta + \partial x$ has minimal L_2 -norm.
- This is a well-known optimization problem. We use the LSQR algorithm.

Outline

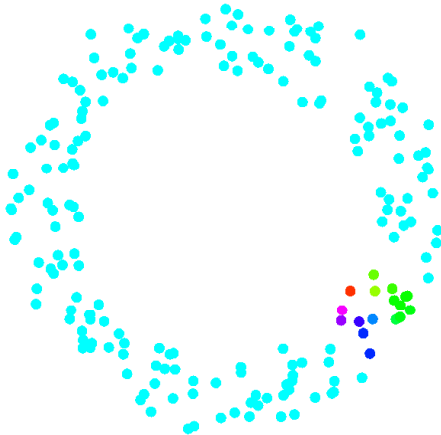
Parametrized circles



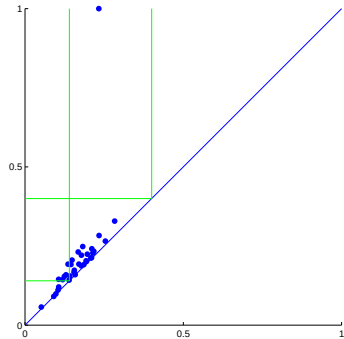
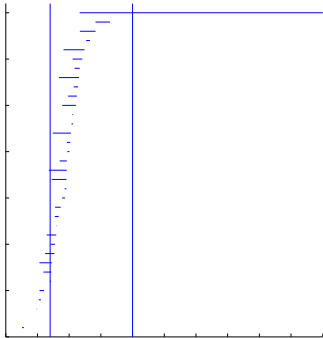
Parametrized circles



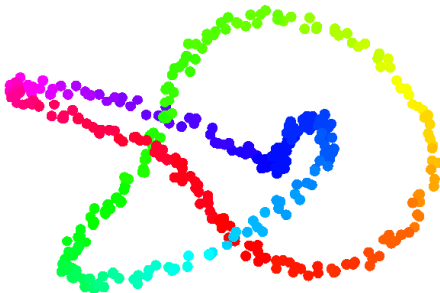
Parametrized circles

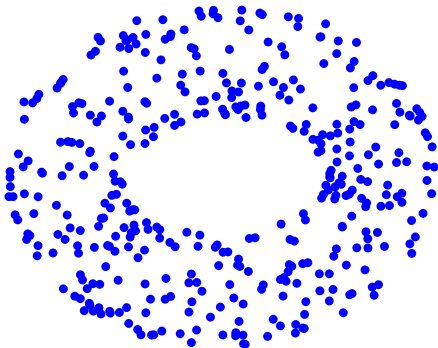


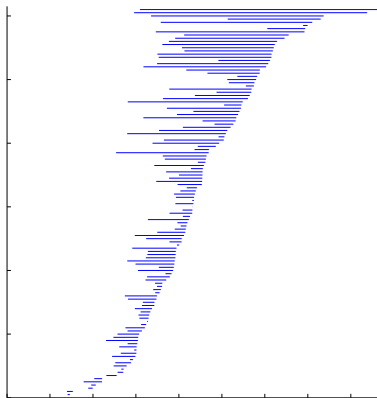
Parametrized circles

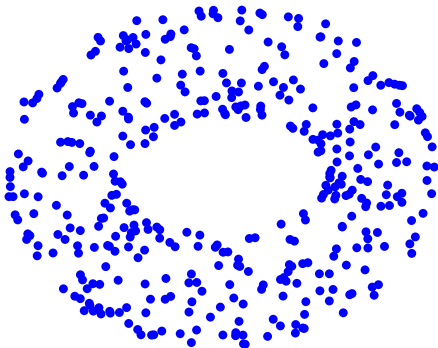


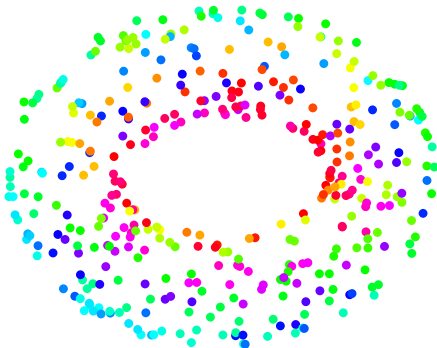
Knots



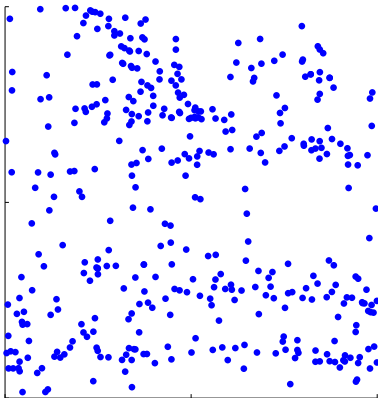




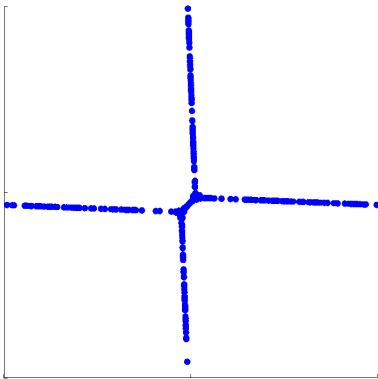




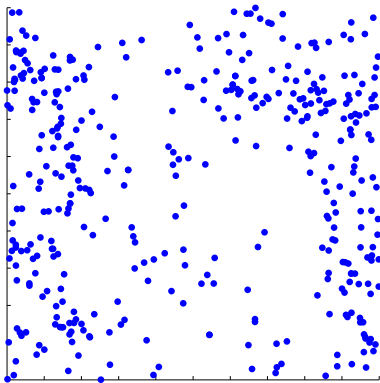




Correlation plot for this torus parametrization

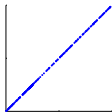
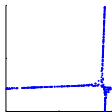
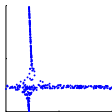
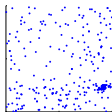


Correlation plot for a wedge of two circles

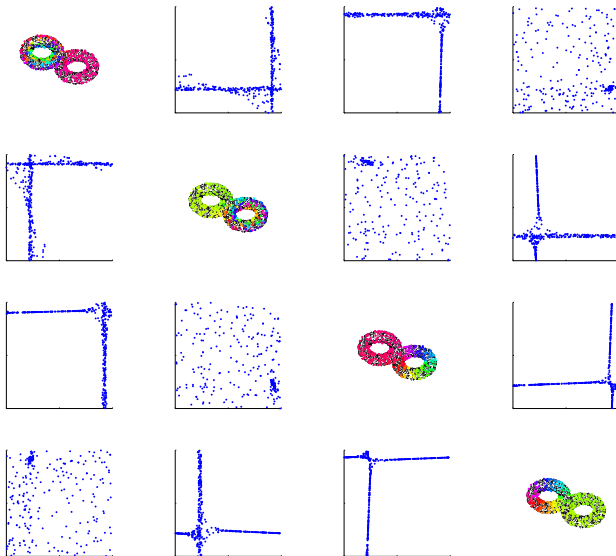


Correlation plot for the elliptic curve given by $y^2z - x^3 - xz^2 = 0$ in $\mathbb{CP}^2$. Metric is given by $d(p, q) = \tan^{-1}(p_x \overline{q_x} + p_y \overline{q_y} + p_z \overline{q_z})$

Pop quiz



Pop quiz – the Double Torus



Persistent cocycles

Given a total filtration order of simplices $\sigma_1, \dots, \sigma_m$, appearing at times $\varepsilon_1, \dots, \varepsilon_m$, for each $0 \leq k \leq m$, we maintain

- A set I_k of indices corresponding to “live” cocycles
- A list of cocycles α_i for $i \in I_k$. Each α_i involves only cocycles appearing no earlier than ε_i .

Persistent cocycles: update step

The algorithm starts with both of these empty. The update, when the simplex σ_k is introduced works by:

- 1 Compute the coboundaries $d\alpha_i = c_i[\sigma_k]$ within the complex $\mathbb{X}_k = \bigcup_{i=1}^k \sigma_i$.

Persistent cocycles: update step

The algorithm starts with both of these empty. The update, when the simplex σ_k is introduced works by:

- ① Compute the coboundaries $d\alpha_i = c_i[\sigma_k]$ within the complex $\mathbb{X}_k = \bigcup_{i=1}^k \sigma_i$.
- ②
 - ① If all $c_i = 0$, then σ_k starts a new cocycle.
 $l_k = l_{k-1} \cup \{k\}$ and $\alpha_k = [\sigma_k]$.
 - ② Otherwise, one cocycle dies at this point. Let j be the largest index such that $c_j \neq 0$.
 $l_k = l_{k-1} \setminus \{j\}$ and for all α_i , we update with $\alpha_i = \alpha_i - (c_i/c_j)\alpha_j$.
The interval $[\varepsilon_j, \varepsilon_k)$ is returned.

Persistent cocycles: update step

The algorithm starts with both of these empty. The update, when the simplex σ_k is introduced works by:

- ① Compute the coboundaries $d\alpha_i = c_i[\sigma_k]$ within the complex $\mathbb{X}_k = \bigcup_{i=1}^k \sigma_i$.
- ②
 - ① If all $c_i = 0$, then σ_k starts a new cocycle.
 $l_k = l_{k-1} \cup \{k\}$ and $\alpha_k = [\sigma_k]$.
 - ② Otherwise, one cocycle dies at this point. Let j be the largest index such that $c_j \neq 0$.
 $l_k = l_{k-1} \setminus \{j\}$ and for all α_i , we update with
 $\alpha_i = \alpha_i - (c_i/c_j)\alpha_j$.
The interval $[\varepsilon_j, \varepsilon_k)$ is returned.

At the end of this, the infinite intervals are given by $[\varepsilon_i, \infty)$ for all $i \in l_m$, and their cocycles are given by the α_i .

An example

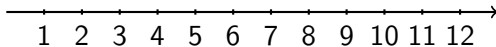
$$I_0 = []$$

$$\alpha_* = \{ \}$$

β_1



β_0



An example

$$I_1 = [1]$$

$$\alpha_* = \{[\sigma_1]\}$$

Coboundary is 0

1
•

β_1



β_0



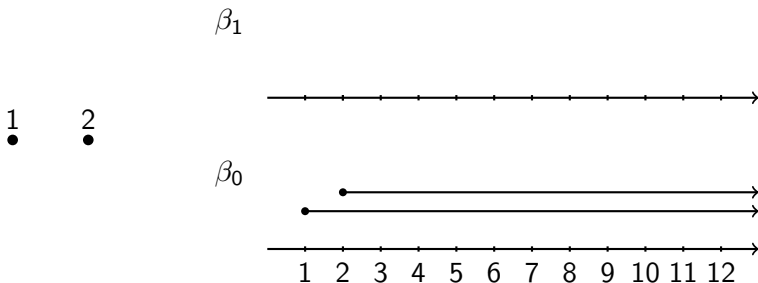
1 2 3 4 5 6 7 8 9 10 11 12

An example

$$I_2 = [1, 2]$$

$$\alpha_* = \{[\sigma_1], [\sigma_2]\}$$

Coboundary is 0

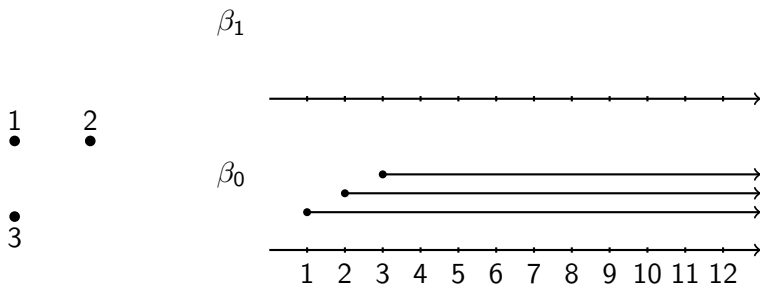


An example

$$I_3 = [1, 2, 3]$$

$$\alpha_* = \{[\sigma_1], [\sigma_2], [\sigma_3]\}$$

Coboundary is 0

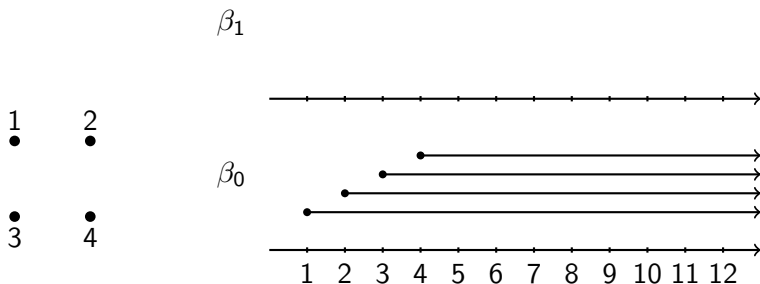


An example

$$I_4 = [1, 2, 3, 4]$$

$$\alpha_* = \{[\sigma_1], [\sigma_2], [\sigma_3], [\sigma_4]\}$$

Coboundary is 0

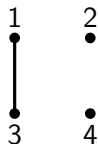


An example

$$I_4 = [1, 2, 4]$$

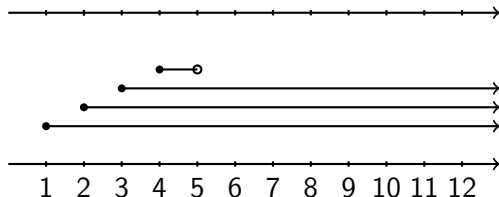
$$\alpha_* = \{[\sigma_1 + \sigma_3], [\sigma_2], [\sigma_4]\}$$

Coboundary is: $-d\alpha_1 = d\alpha_3 = [\sigma_5]$. 3 dies.



β_1

β_0

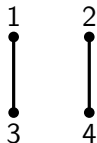


An example

$$I_4 = [1, 2]$$

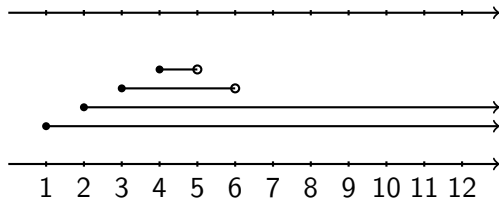
$$\alpha_* = \{[\sigma_1 + \sigma_3], [\sigma_2 + \sigma_4]\}$$

Coboundary is: $-d\alpha_2 = d\alpha_4 = [\sigma_6]$. 4 dies.



β_1

β_0

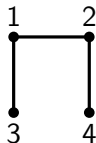


An example

$$I_4 = [1]$$

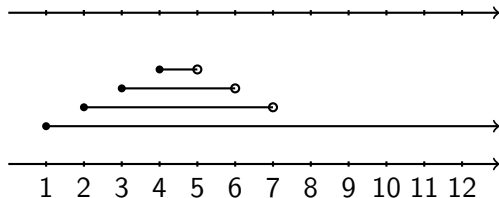
$$\alpha_* = \{[\sigma_1 + \sigma_2 + \sigma_3 + \sigma_4]\}$$

Coboundary is: $-d\alpha_1 = d\alpha_2 = [\sigma_7]$. 2 dies.



β_1

β_0

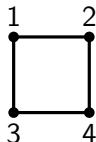


An example

$$I_4 = [1, 8]$$

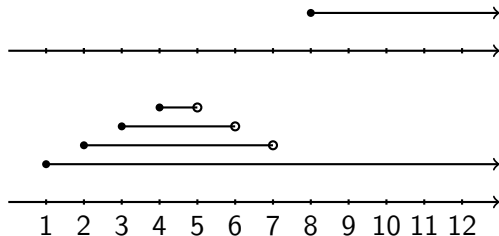
$$\alpha_* = \{[\sigma_1 + \sigma_2 + \sigma_3 + \sigma_4], [\sigma_8]\}$$

Coboundary is 0



β_1

β_0

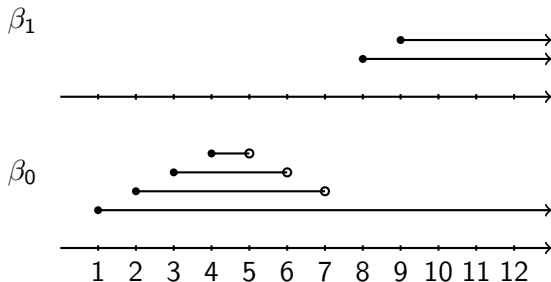
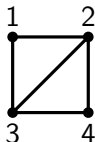


An example

$$I_4 = [1, 8, 9]$$

$$\alpha_* = \{[\sigma_1 + \sigma_2 + \sigma_3 + \sigma_4], [\sigma_8], [\sigma_9]\}$$

Coboundary is 0

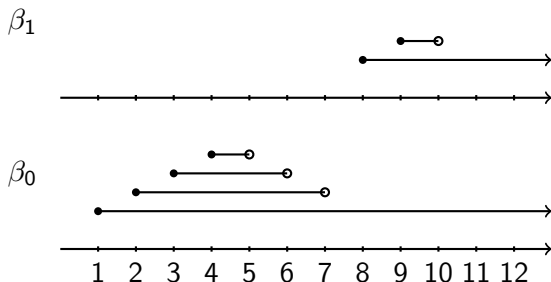
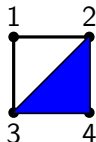


An example

$$I_4 = [1, 8]$$

$$\alpha_* = \{[\sigma_1 + \sigma_2 + \sigma_3 + \sigma_4], [\sigma_8 - \sigma_9]\}$$

Coboundary is $d\alpha_8 = d\alpha_9 = [\sigma_{10}]$. 9 dies.

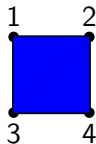


An example

$$I_4 = [1]$$

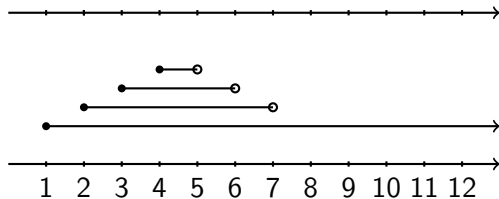
$$\alpha_* = \{[\sigma_1 + \sigma_2 + \sigma_3 + \sigma_4]\}$$

Coboundary is $d\alpha_8 = [\sigma_{11}]$. 8 dies.



β_1

β_0



The persistent cohomology algorithm is implemented in two places:

jPlex

- `http://comptop.stanford.edu/programs/jplex`
- Java based. Matlab integration.
- Will do cohomology in the next release.

Dionysus

- `http://www.mrzv.org/software/dionysus/`
- C++ based. Has Python bindings.
- Still very much under development.
- Orders of magnitude faster than jPlex on these examples.

Timings for Dionysus

Example	# data points	# simplices	total time	time/size ($\mu\text{s}/\text{spx}$)
Noisy circle	200	23 475	0.10s	4.26
Torus knot	400	36 936	0.16s	4.33
Wedge of 2 circles	400	76 763	0.36s	4.69
2 disjoint circles	400	45 809	0.20s	4.37
Torus	400	61 522	0.29s	4.71
Elliptic curve torus	400	44 184	0.14s	3.17
Double torus	3 120	764 878	5.28s	6.90

Acknowledgements

Thanks are due for this to:

- Pomona College
- Vin de Silva, Dmitriy Morozov – my collaborators
- Gunnar Carlsson
- ONR, DARPA-TDA, Pomona College and Stanford University – funding