パーシステントホモロジーの計算ソフトウェアと発展的話題

大林一平

WPI-AIMR, Tohoku University

Feb. 28, 2017

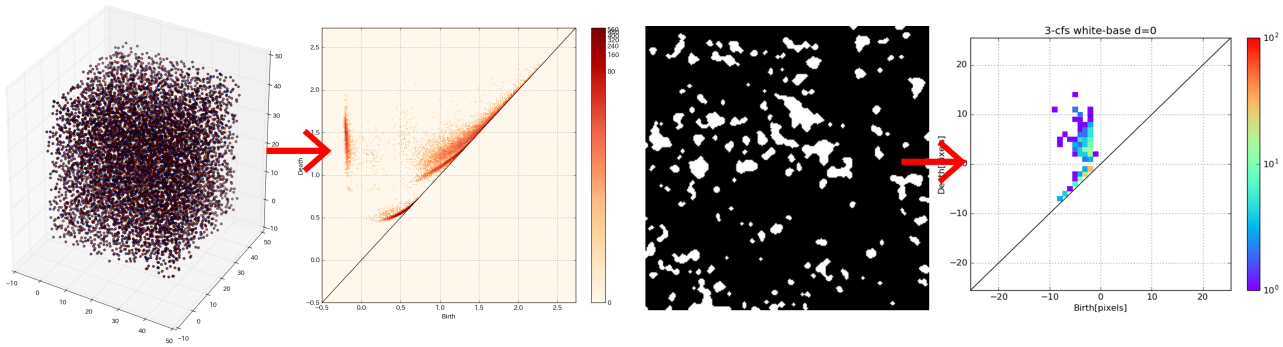
Outline

- パーシステントホモロジーの計算ソフトウェア
  - パーシステント図の計算の概要
  - ソフトウェアのレビュー
  - パーシステントホモロジー計算体験セッション
  - homcloud (我々が開発しているソフトウェア) デモ
- パーシステント図の逆問題について
  - 逆問題について
  - 理論とアルゴリズム

計算の概要

パーシステンスホモロジーの理論から，何か「幾何的なデータの増大列 (フィルトレーションと呼ぶ)」があればパーシステント図は計算可能であるが，一般的に次のようなデータを考えることが多い．

- ポイントクラウド (2 次元，3 次元の上の点の集合)
- 抽象的な距離空間上の点の集合と，各点の間の距離
- n 次元の画像 (グレイスケール，二値画像)
  - 3 次元ではボクセルデータと呼ばれる



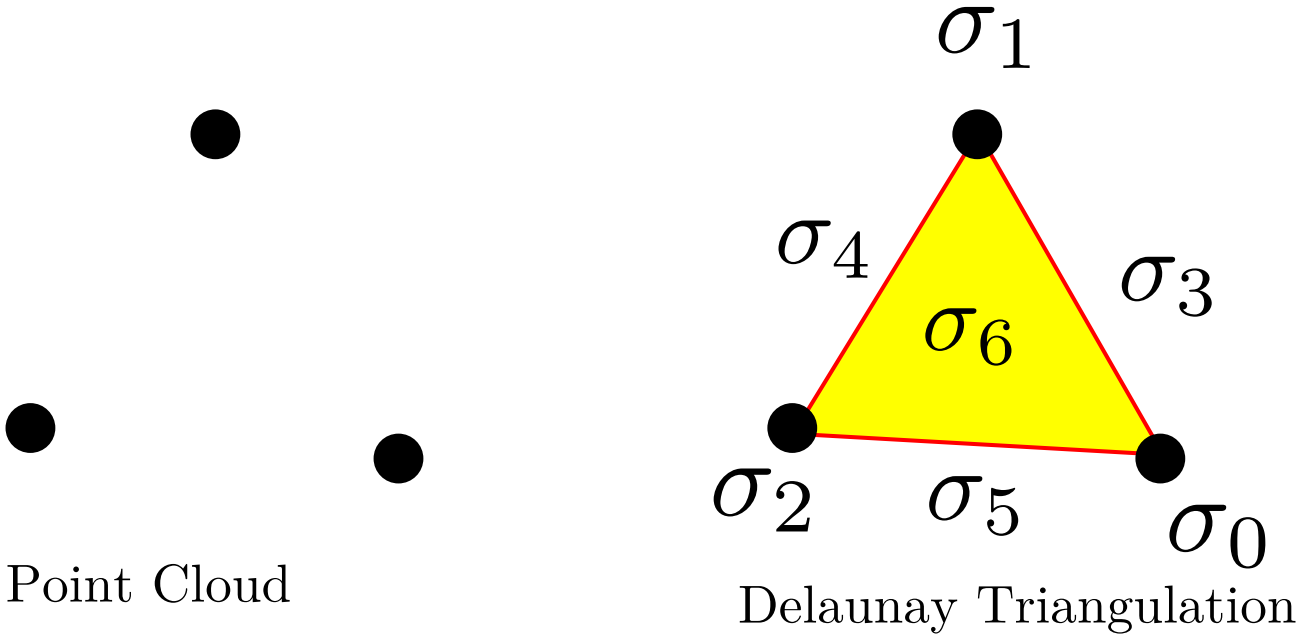
ポイントクラウド

2 次元，3 次元の上の点の集合

- 3 次元の表面データ
  - 3D キャプチャ
  - 3D スキャナ
- MD シミュレーションや RMC などで得られる原子配置データ

このようなデータの場合，「アルファ複体」と呼ばれるものを経由して計算する．

アルファ複体



アルファ複体の

- ドロネー三角形分割
- すべての「単体 (頂点，辺，三角形，三角錐)」の「Birth time」
- 「単体」の間の関係
  - $\partial\sigma_6 = \sigma_3 + \sigma_4 + \sigma_5$, $\partial\sigma_3 = \sigma_1 + \sigma_0$, ...

ドロネー複体の計算は計算幾何学の基本的問題．例えば cgal というソフトウェアで計算できる．

抽象的な距離空間のデータ

- 遺伝子のデータ (2 つの遺伝子の間に「距離」を定義する標準的な方法がある)
  - 遺伝子の遠い近いは判定できる
  - ただ，3 次元などに配置するのは難しい
- 非常に高次元の空間上のデータ
  - この場合，アルファ複体は実用的ではないので距離だけ考える
- 例えば 2 つの文章の「距離」なども自然言語処理で実用的に使われている

このような場合は「Vietris-Rips フィルトレーション」を用いる

画像について: グレイスケール画像や二値画像

- スカラー場のデータでも同様のことはできる
  - 温度の空間分布など

グレイスケール画像であれば，二値化の閾値を徐々に変化させることで，領域の増大列が作れる→パーシステント図が計算できる.
二値画像データでも，領域を広げる/狭める操作を考えることで領域の増大列が作りパーシステント図が計算できる.

Example: Gray scale image (values: from 0 to 7)

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 0

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 1

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 2

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 3

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 4

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 5

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 6

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

Threshold: 7

7	5	5	3	4	1	1	1	1	1	2
4	5	2	2	4	6	0	0	0	0	1
3	2	1	4	5	0	2	3	2	0	0
2	2	3	3	5	0	4	7	1	1	0
2	3	4	5	5	0	2	5	6	1	0
4	4	5	0	0	0	0	0	0	0	0
1	1	0	0	1	1	1	2	2	1	0
0	1	0	1	1	2	1	2	2	1	0
2	1	0	0	0	0	0	1	1	0	0
3	2	1	1	1	1	0	0	0	4	4

$D_0 : (0, \infty), (0, 1), (1, 4)$
$D_1 : (1, 2), (1, 2), (2, 7), (4, 6)$

出力 (可視化) について

有限個の birth-death pair をどう可視化するかは広く二通りの方法がある.

- バーコード
- パーシステント図←この講義ではこちらを使います

計算の概要:

1. 入力データの準備
   - ポイントクラウド
   - 画像 (「場」のデータ)
2. フィルトレーションを計算する
   - アルファフィルトレーション (ポイントクラウド)
   - Vietris-Rips フィルトレーション (高次元ポイントクラウドや距離データ)
   - Cubical filtration (画像など)
3. フィルトレーションからパーシステントホモロジーの計算をする (つまり birth-death pair を計算する)
4. 計算結果を可視化

Software

レビュー論文

様々なパーシステントホモロジーのソフトウェアについてレビューをしている

http://arxiv.org/abs/1506.08903

特に性能のベンチマークが有用

Software

- Perseus
- JavaPlex
- Dipha
- GUDHI

Perseus

- `http://www.sas.upenn.edu/~vnanda/perseus/`
- Developed by V. Nanda in upenn
- 性能はよくない
- 手軽には使える

JavaPlex

- `http://appliedtopology.github.io/javaplex/`
- Developed by Computational Topology workgroup at Stanford University
- Java で書かれている
- matlab の例などもある
- 歴史あるソフト
- 性能はそこそこ
- Java で使いたければ良いかも

Dipha

- `https://github.com/DIPHA/dipha`
- Developed by IST Austria
  - Jan Reininghaus
  - Ulrich Bauer
  - Michael Kerber
- 高速
- 並列計算をサポート
- Vietris-Rips, Cubical, 単体複体のフィルトレーションをサポート
  - アルファフィルトレーションは自分で+αの実装が必要

GUDHI

- https://project.inria.fr/gudhi/
- Developed by INRIA
- これも高速
- $+\alpha$ の機能がいくつか
  - アルファフィルトレーションの構築など

Homcloud

- 我々が開発しているソフトウェア
- パーシステントホモロジーの計算は dipha に丸投げ
- どちらかというと UI や使いやすさに重点
- 近日一般公開予定 (夏ごろに部分的に公開, funding などお金の関係で完全な公開は大分先, 使いたければ要相談)
- Windows 版を鋭意開発中

以上のソフトはそんなに UI がユーザに優しくないので使いこなすのは大変かもしれない
ここでは体験用に Web アプリケーションを準備したのでそれを使いましょう.

体験セッション

使い方:

- Range: の所でパーシステント図のヒストグラムの左右の幅と bin の数を指定
  - 幅はデータの空間スケールの二乗を指定
- Dim で次元，Degree でホモロジーの次数を指定
- エディットボックスで点の座標を指定
- 「submit」で計算

注意:

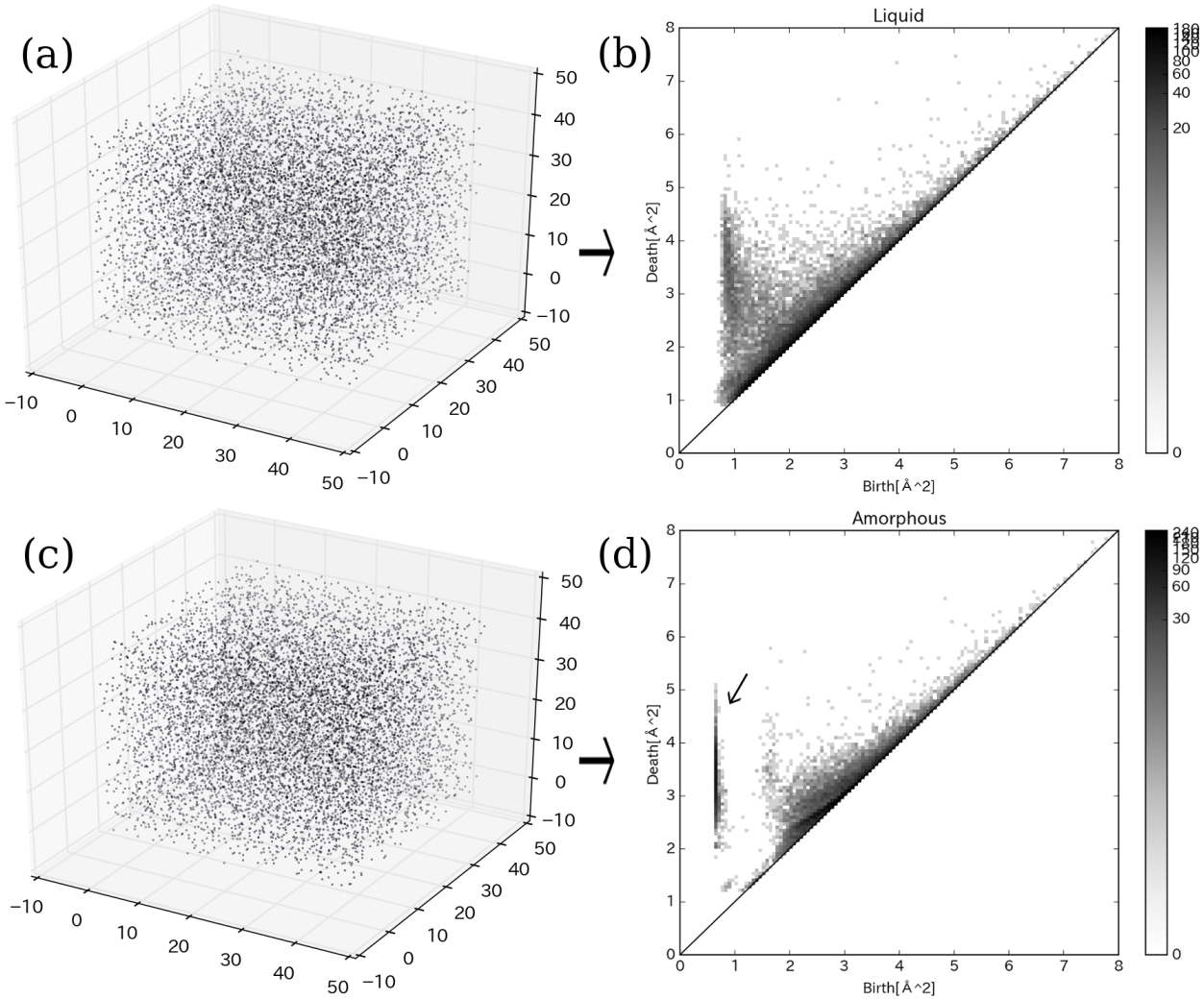
- 点の個数が「次元+1」以上ないとうまく動かない
- 何かエラーが生じたときは講師まで

Homcloud デモ

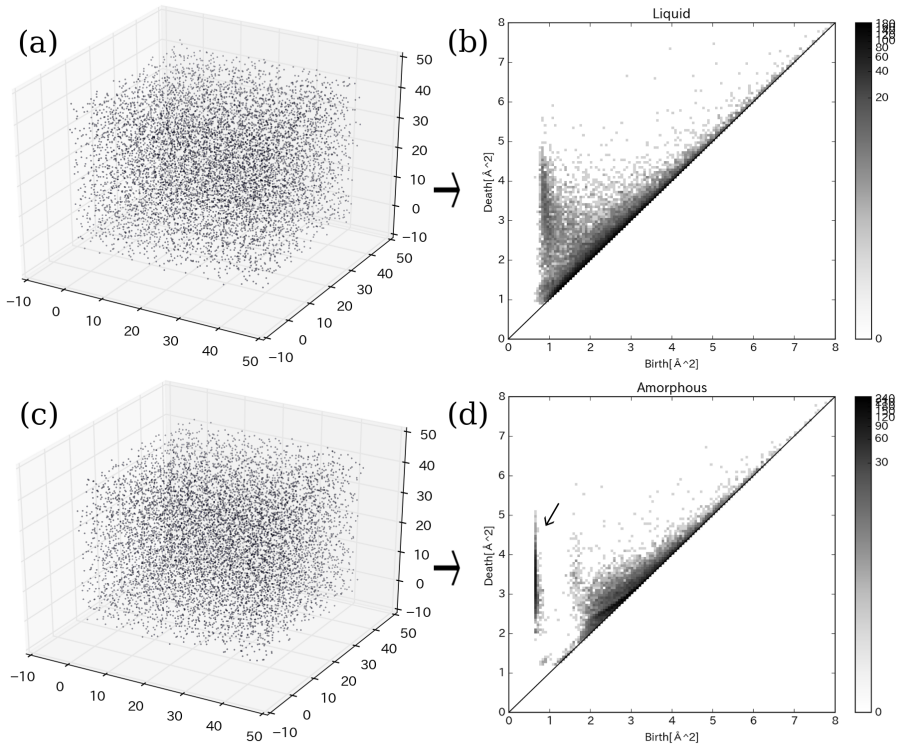
ここでデモをします．Web アプリケーションではできないことも紹介.

パーシステント図の逆問題

液体シリカとアモルファスシリカの原子配置から計算したPD(1次)



アモルファスシリカのほうには明らかに特徴的構造が見える．これの「源」を調べることで液体シリカとアモルファスシリカの幾何構造の違いがわかるはず．

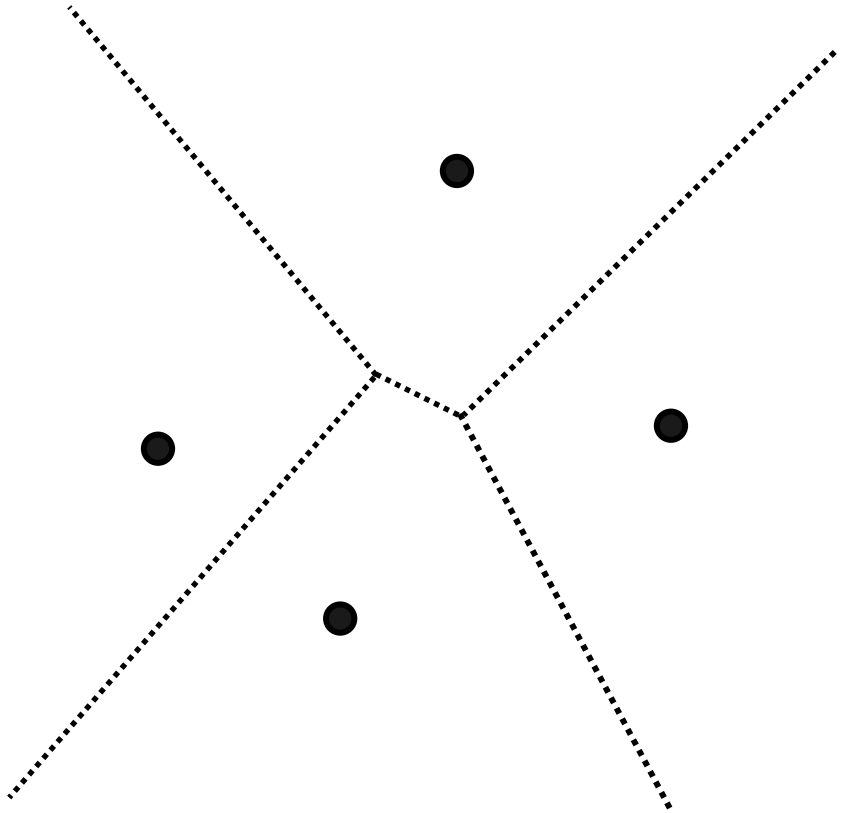


つまり問題は「パーシステント図から元のデータの情報を得たい」.
こういう類の問題を「パーシステント図の逆問題」と呼ぶことにする.

Persistent homology

逆問題をちゃんと定義して解くためにはある程度理論的背景を抑える必要がある

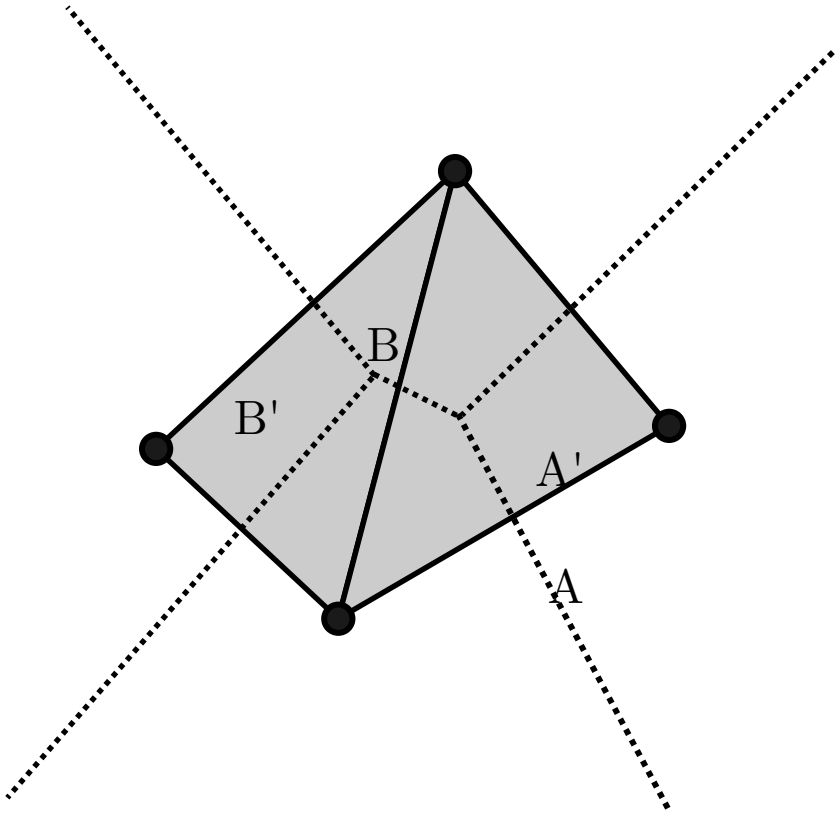
ボロノイ図



Point Cloudのどの点に一番近いか、で領域を分割

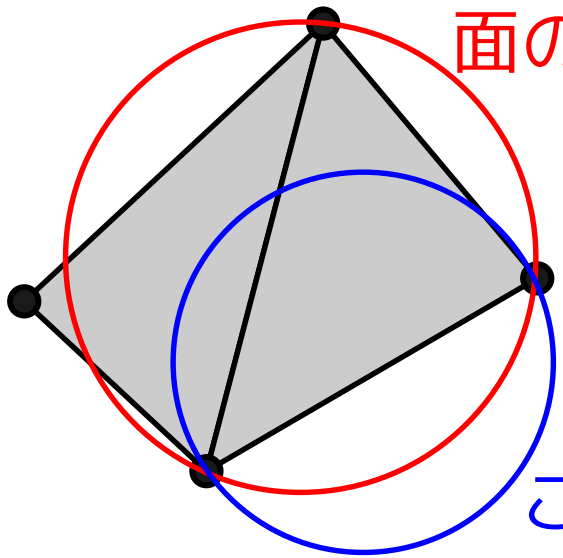
分割の教会は2点の垂直二等分線になる

ドロネー図 (実線,灰色の面)



ボロノイ図で、2つの領域の境界が接するときに、対応する点に辺を引く。3つの領域が接するとき、そこに面を置く。例えばAの点線とA'の辺が対応し、Bの点線三叉路とB'の面が対応する。3次元でも同様(この場合は三角錐まで考える;)。

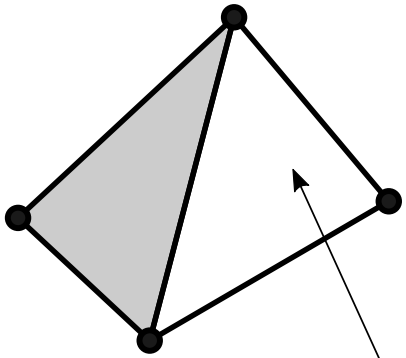
アルファ複体



各辺、各面、各三角錐（3次元）に「生成時間」を定める。これは、その辺を覆うような円（3次元だと球）で最小半径なものの半径、と決める。通常は外接円半径だが、鈍角三角形だと一番長い辺の長さの半分となる。

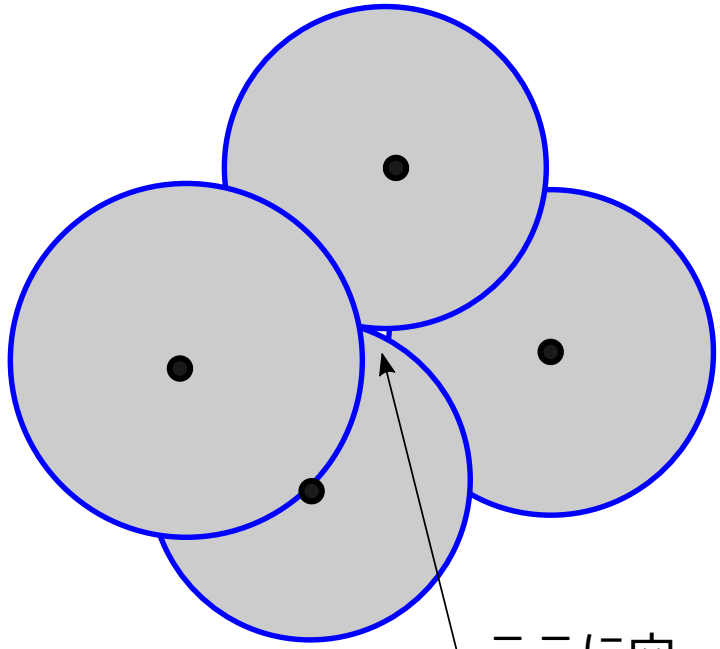
円を貼り付けること

生成時間がr以下のものだけ



ここに穴

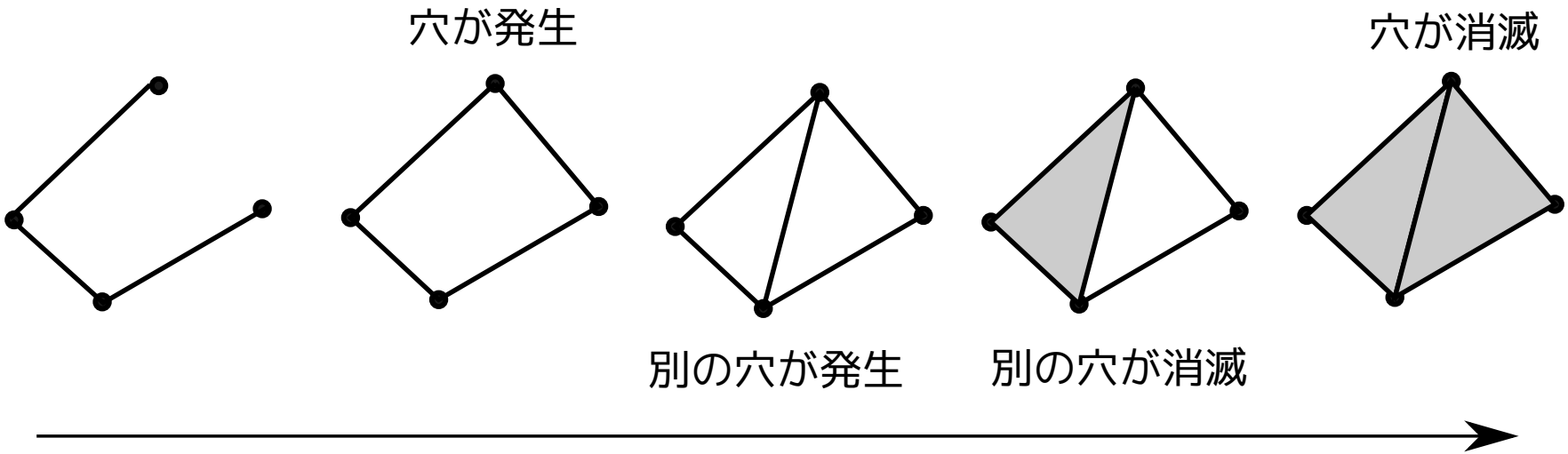
半径rの円



ここに穴

実は半径rの円を貼り付けることと、生成時間がr以下だけの辺、面、三角錐に注目することは「トポロジー的には」同じである（これは穴や空洞の個数は完全に一致するという意味、大きさは違う）。そのため、半径rを大きくしていくのと生成時間を大きくしていくことは同じもの。

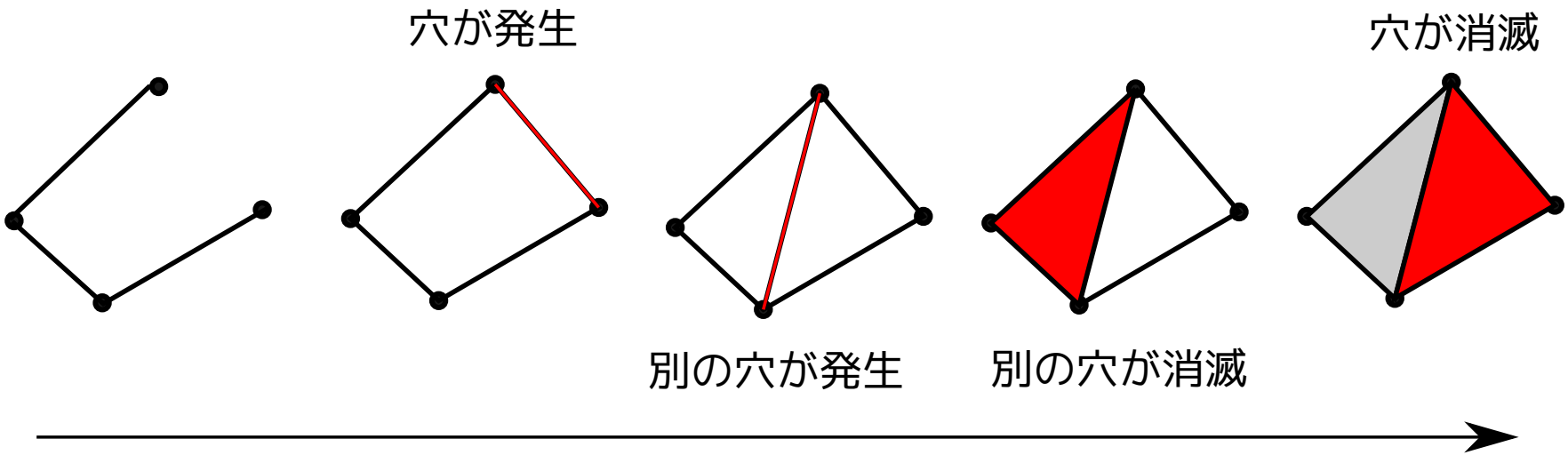
アルファフィルトレーションと パーシステントホモロジー



このように生成時間の閾値を徐々に大きくすることは貼り付けた円を徐々に大きくすることと同じ

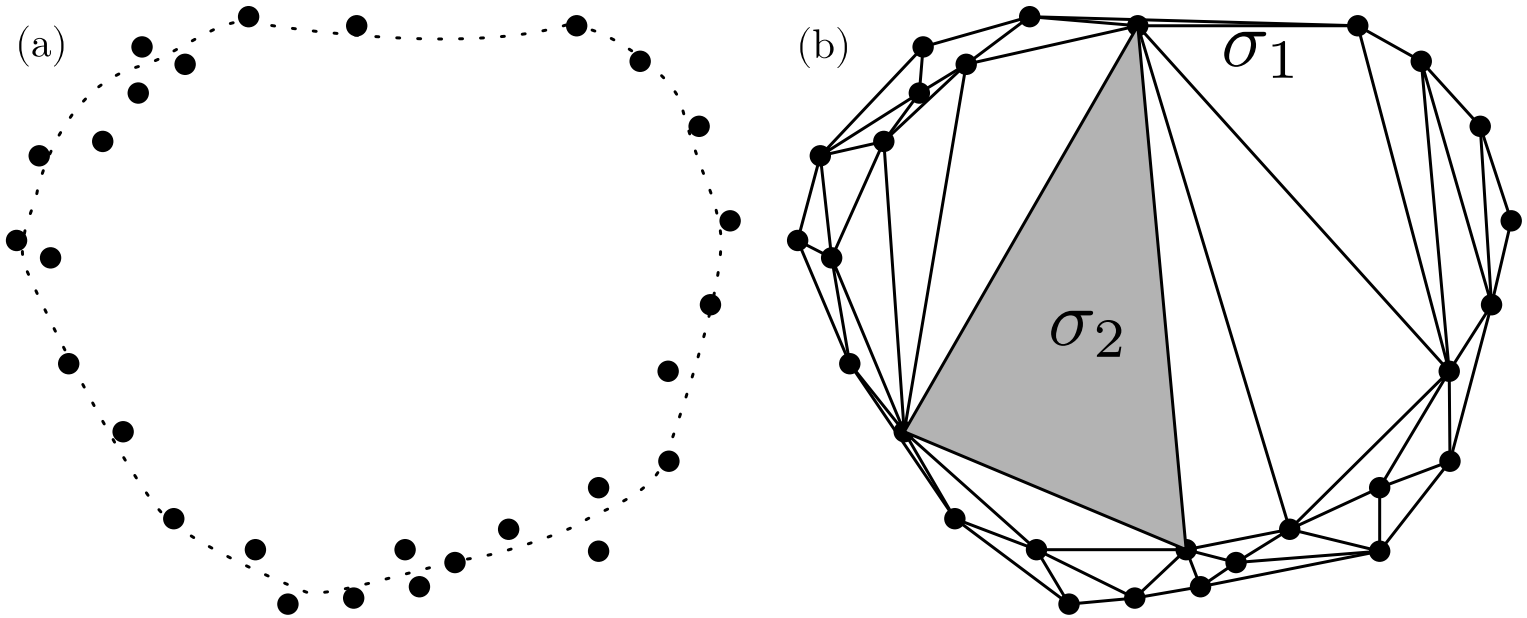
birth/death simplices

Birth Simplex と Death Simplex



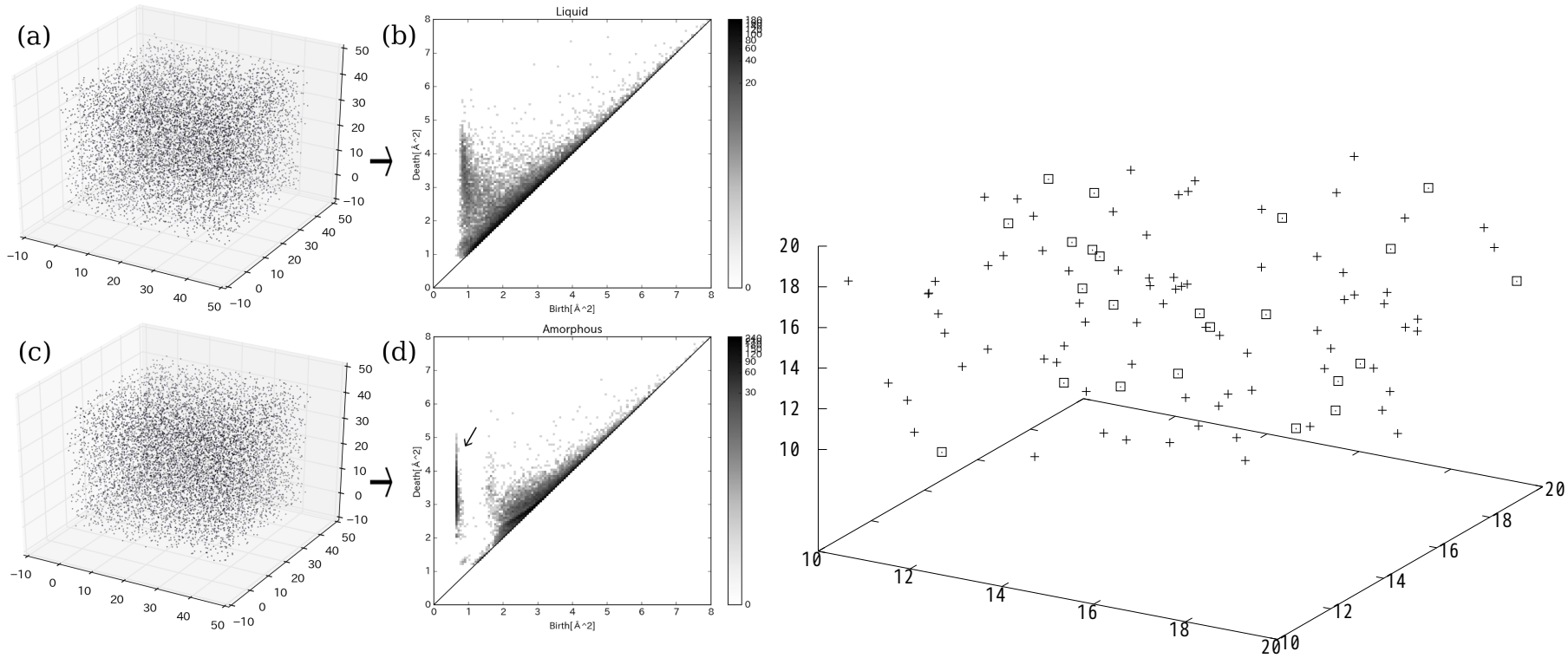
上の赤で書いた辺、面のように穴の「生成」「消滅」のきっかけと
なるものがある。これがbirth simplex、death simplexである。
円を貼り付ける場合で考えると、「穴が発生する」場合には2つの円が接し、
「穴が消滅する」場合には3つの円が交わるようになる、のに対応する

ではこれの意味は？



σ_1 が birth simplex で σ_2 が death simplex.

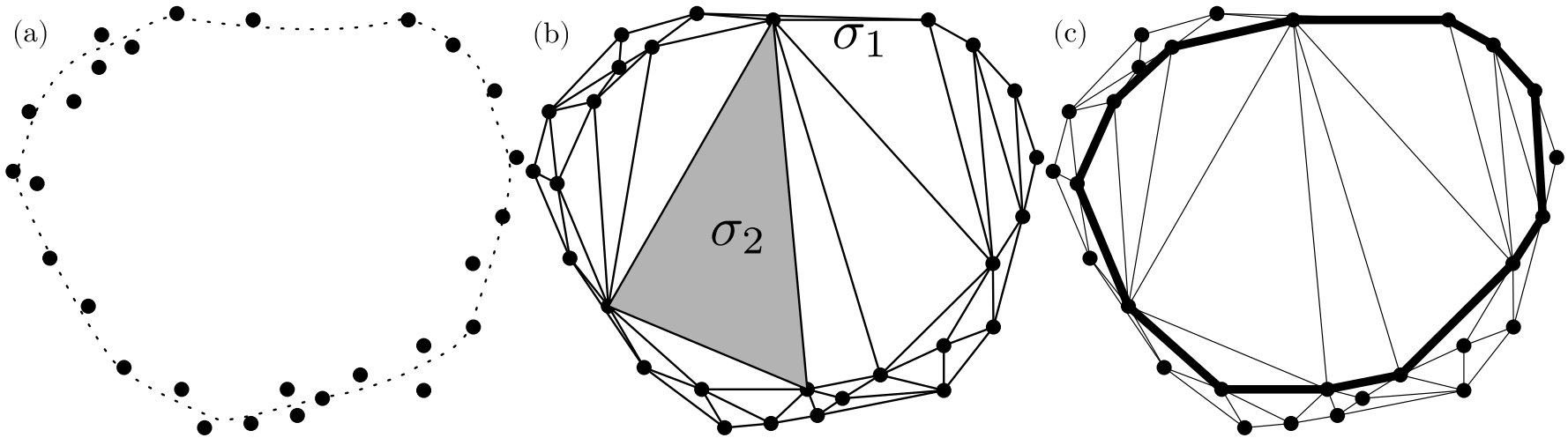
Example of death simplices



これは homcloud で計算できる.

Optimal cycles

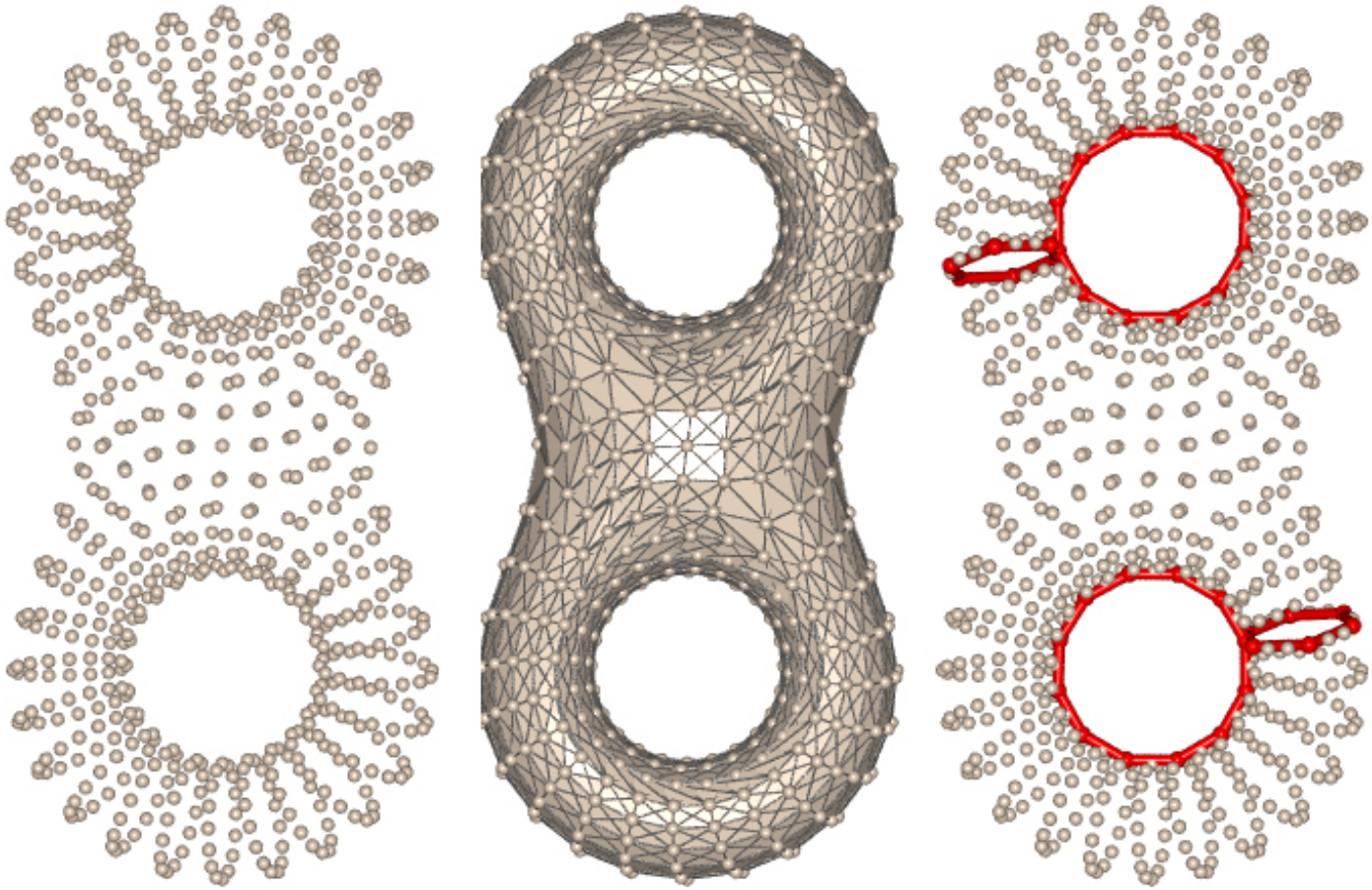
Optimal cycle



ここでは birth/death simplices を考えたが，(c) のような情報が得られるとより便利．これは *optimal cycle* と呼ばれる．

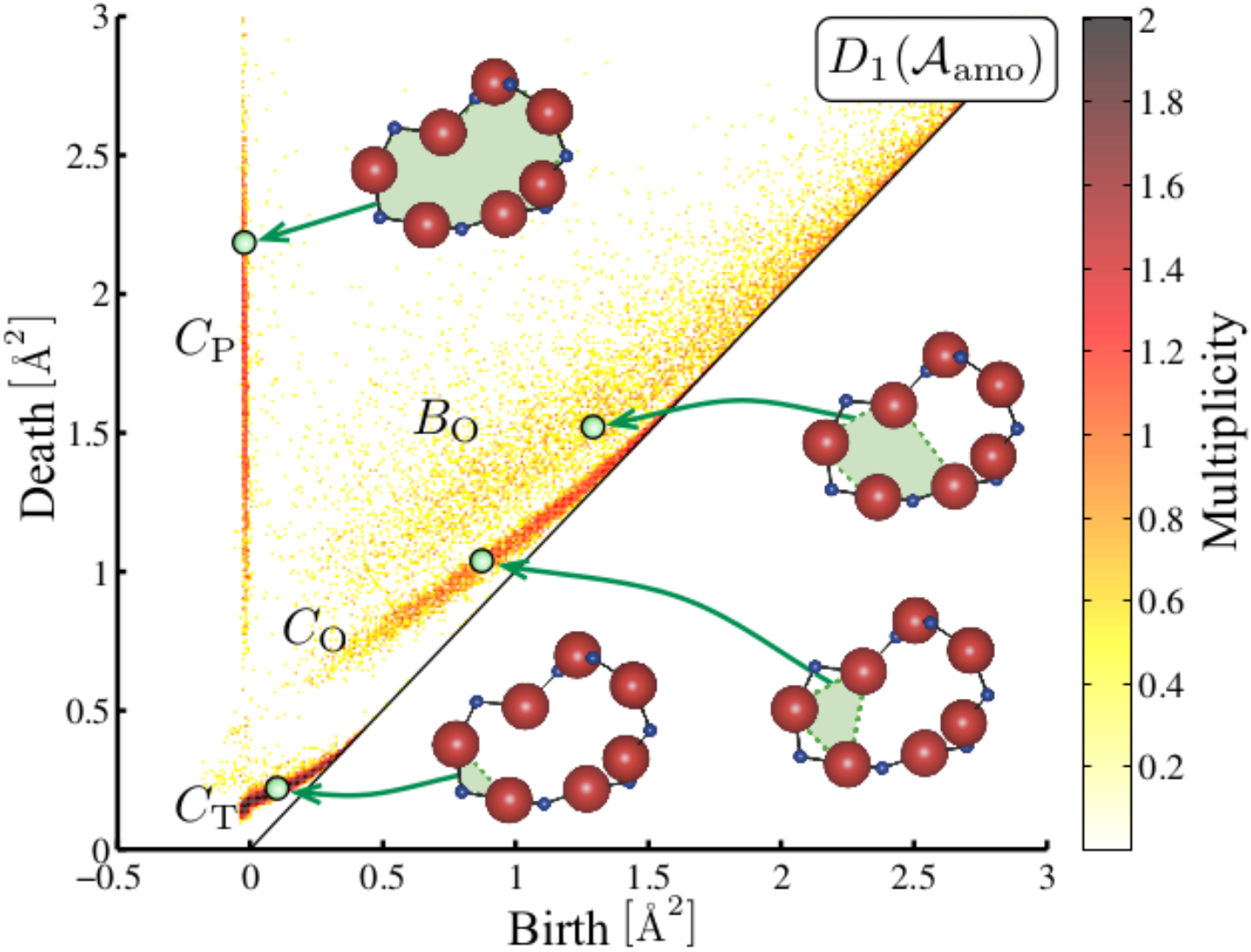
計算には離散最適化問題を解く必要があるため，かなりの計算時間がかかる問題があるが，より多くの情報を持っている利点がある．

Example



(by the computational geometry group in Ohio state)

Example

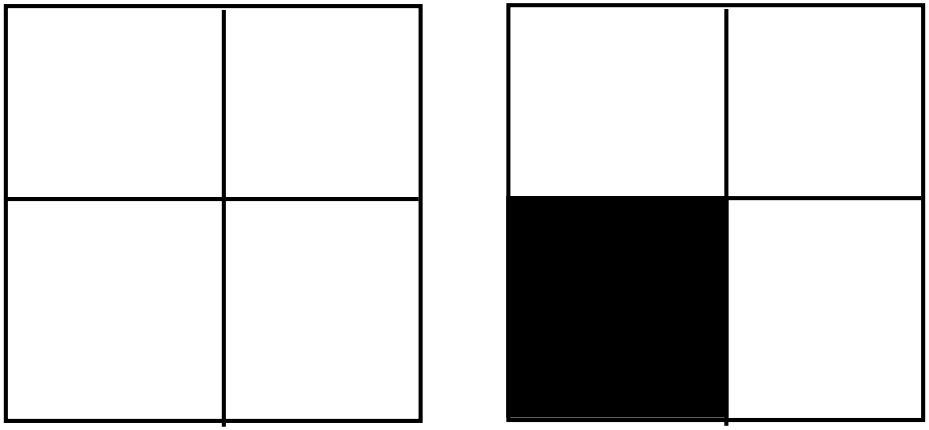

$D_1(\mathcal{A}_{\mathrm{amo}})$
C_{P}
B_{O}
C_{O}
C_{T}
Death [Å^2]
Birth [Å^2]
Multiplicity
3
2.5
2
1.5
1
0.5
0
−0.5
0
0.5
1
1.5
2
2.5
3
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2

理論的背景

Q: 係数体

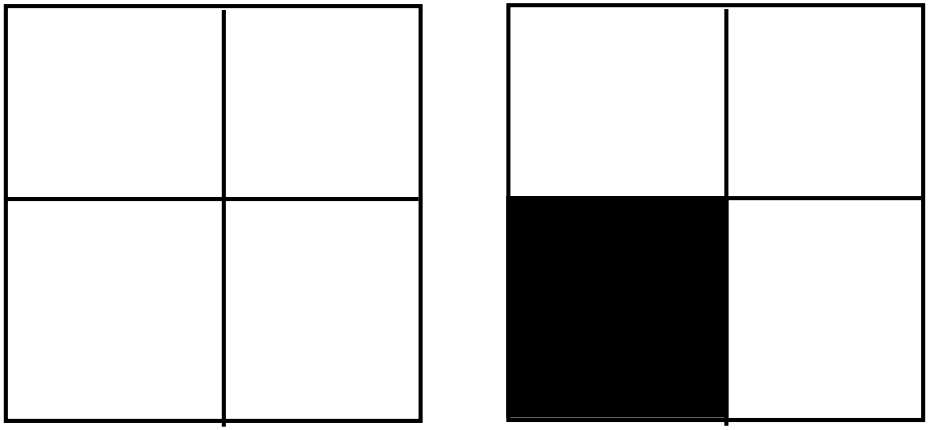
- ホモロジーというのはチェイン複体 $\{C_\ell\}_{\ell=0}^{L}$ とその上の境界作用素 $\partial_\ell : C_\ell \to C_{\ell-1}$ で定義される
- サイクル: $Z_\ell = \ker \partial_\ell$
- バウンダリ: $B_\ell = \mathrm{im}\, \partial_{\ell+1}$
- ℓ-th ホモロジー加群 (ベクトル空間): $H_\ell = Z_\ell / B_\ell$

なにこれ？



- 右図と左図の穴の個数を数える

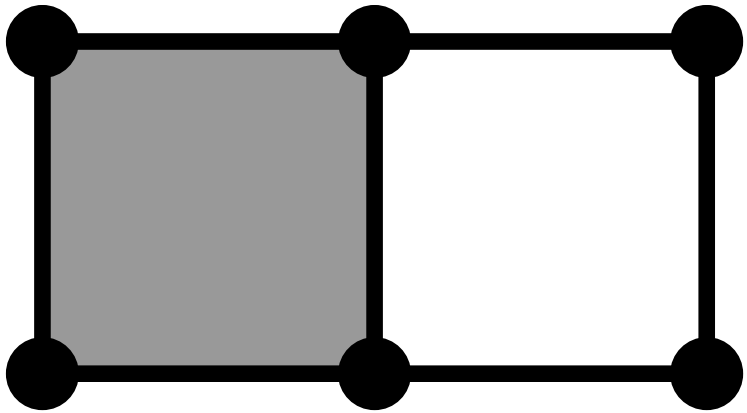
なにこれ？



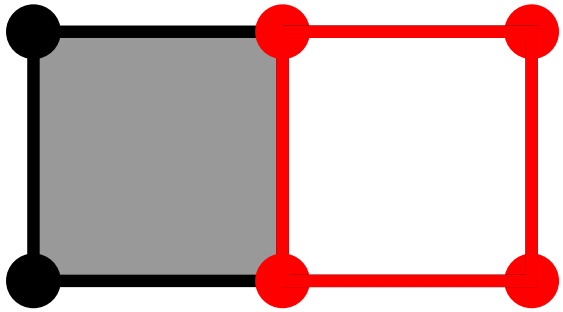
- 右図と左図の穴の個数を数える
- 穴の個数というのはベクトル空間の次元

1次元の穴を定義してみよう

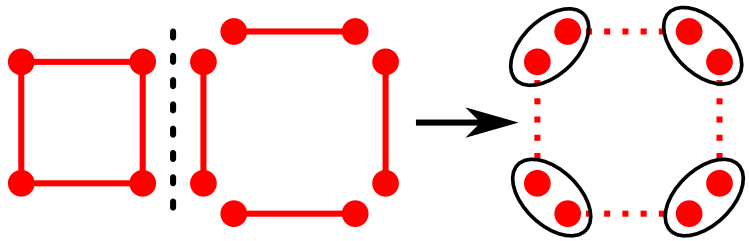
以下の図には，頂点が6個，辺が7個，面が1個あり，穴は1個である．
ここで突然であるが，係数が$\mathbb{Z}/2\mathbb{Z}$で基底が頂点/辺/面であるベクトル空間をC_0, C_1, C_2と置く．係数が$\mathbb{Z}/2\mathbb{Z}$だと同じものを2つ足すと0になる．



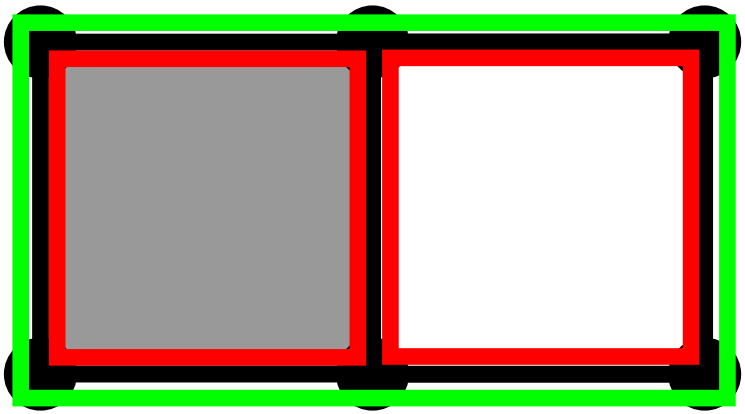
穴とは？ →ループではないか.



線形写像 $\partial_1 : C_1 \to C_0$ を，
∂_1(辺) = (辺の一方の頂点) + (辺のもう一方の頂点) とすると，∂_1(ループ) $= 0$ となっている.

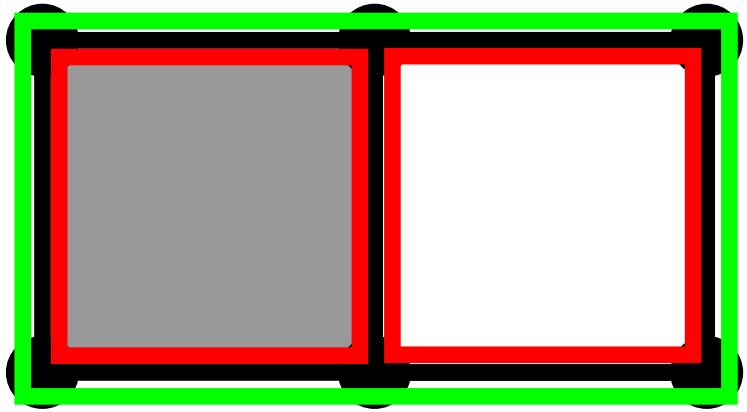


つまりループとは $\ker\partial_1$ で表されるのでは？ ループの個数とは $\dim\ker\partial_1$ では？

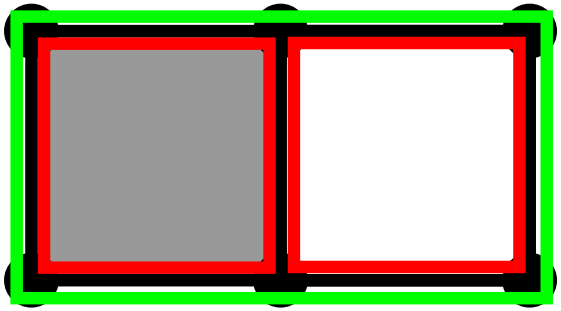


緑のループは２つの赤のループの和ということに注意すると $\dim \ker \partial_1 = 2$

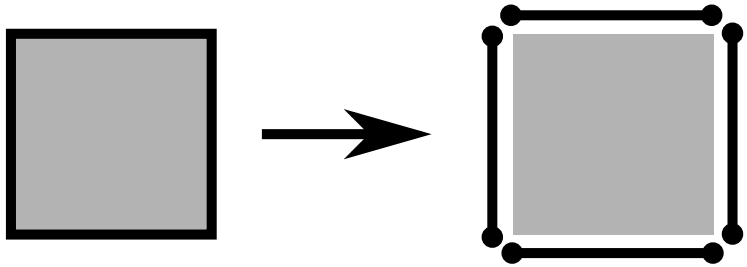
この図形ではループの一方は埋まっているため穴ではない.



→埋まっているループとそうでないループを区別する必要がある.



線形写像 $\partial_2 : C_2 \to C_1$ を，$\partial_2(\text{面}) = \sum(\text{面の各辺})$ で定義すると，$\partial_2(A)$ の像がループになっている．



→これが埋まっているループだろう．→つまり埋まっているループの個数は $\dim \mathrm{im}\partial_2$.

結局，穴の個数は

$$\dim \ker \partial_1 - \dim \mathrm{im} \partial_2$$

と言える．で，これは行列の rank を計算すればよい．
行列の rank は掃き出し法で計算機で計算できる!

2次元の穴の場合

この場合は3次元データを考える必要がある．頂点，辺，面に加えてキューブを考える．そして，

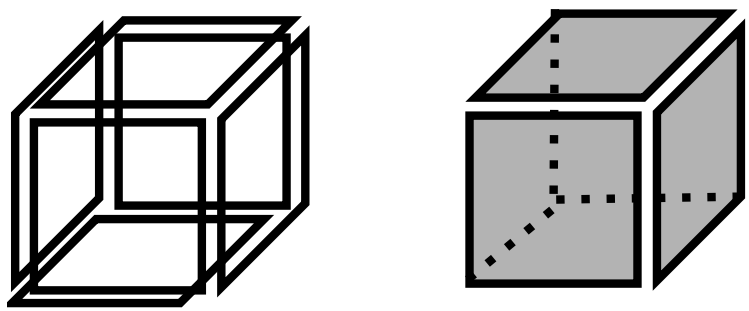
$$\partial_3(\text{キューブ}) = \sum \text{キューブの各面}$$

$$\partial_2(\text{面}) = \sum(\text{面の各辺})$$

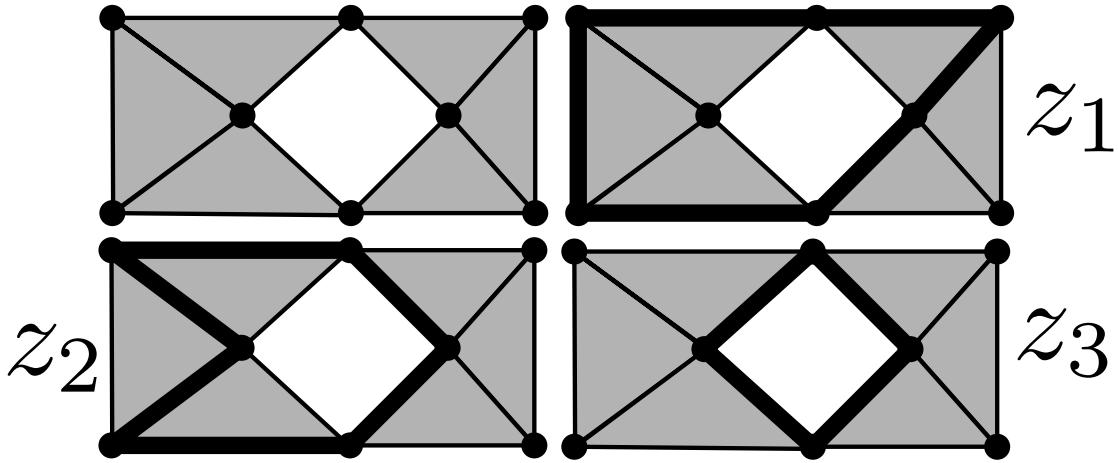
とすると，先程と同様にして

$$\dim \ker \partial_2 - \dim \mathrm{im} \partial_3$$

が2次元の穴の個数.

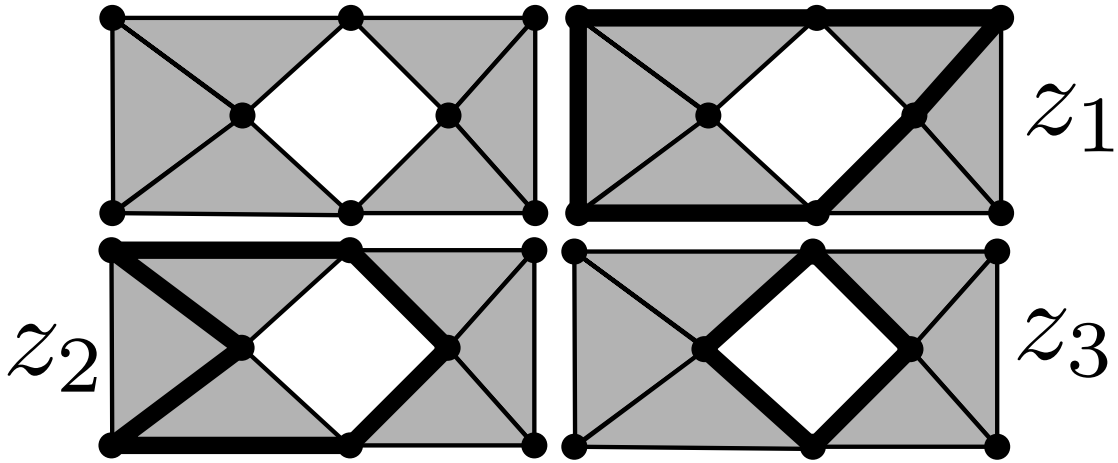


$Q = \mathbb{Z}_2$:



$H_1 \cong \mathbb{Z}_2$ and $H_1 = \langle [z] \rangle$.
z が穴の情報を持っている，が $[z] = [z_1] = [z_2] = [z_3]$ となり，z_1, z_2, z_3 はホモロジーのレベルでは同じ．でも z_3 が一番良さそう．何故？

$Q = \mathbb{Z}_2$:



$H_1 \cong \mathbb{Z}_2$ and $H_1 = \langle [z] \rangle$.
z が穴の情報を持っている，が $[z] = [z_1] = [z_2] = [z_3]$ となり，z_1, z_2, z_3 はホモロジーのレベルでは同じ．でも z_3 が一番良さそう．何故？
ループの長さが重要!

ホモロジーの世界では $z = \sum_i k_i \sigma_i$, ループの長さというのは非零である k_i. の個数と同じ．そこでこれを $\|z\|_1$ と書くと，次の問題を考えるとよい結果が得られそう

$$\begin{aligned} &\text{minimize } \|z + \partial y\|_1 \\ &\text{where } y \in C_{\ell+1} \end{aligned}$$

for fixed $z \in C_\ell$.

$$\text{minimize} \|z + \partial y\|_1$$
$$\text{where } y \in C_{\ell+1}$$

この問題は線形計画法と呼ばれる．ただ，普通線形計画法は $\mathbb{R}$ や $\mathbb{Z}$ で考える．そこで $\mathbb{R}$ や $\mathbb{Z}$ を $\mathbb{Z}_2$ のかわりに使おう．

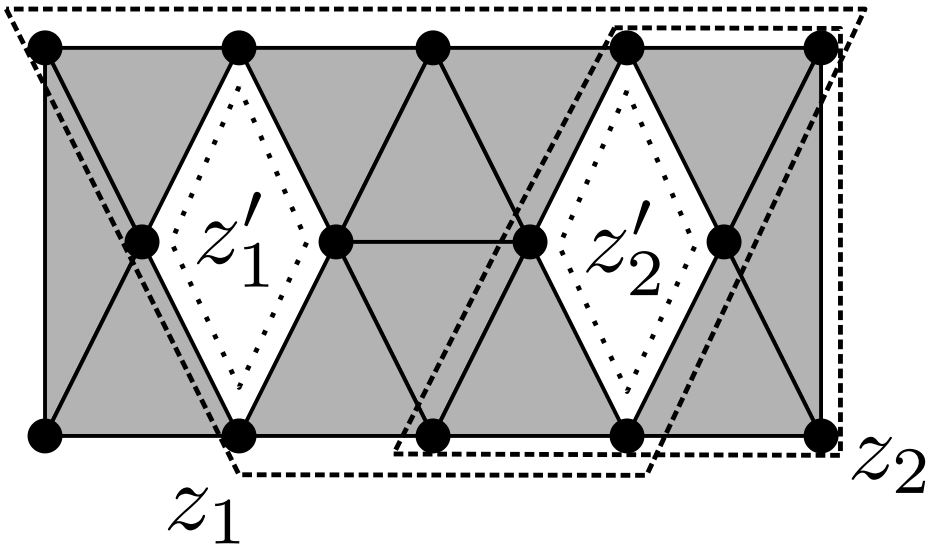
$\mathbb{R}$ だったら高速に計算できる．でも

- $\|\cdot\|_1$ の意味が変わってしまう
- 解釈が難しい

そこで $0-1$ 線形計画法というものを使うとまだ解釈しやすい結果が得られる．つまり係数を 0, 1, -1 に制限するのである．ただこうすると計算は大変になる．

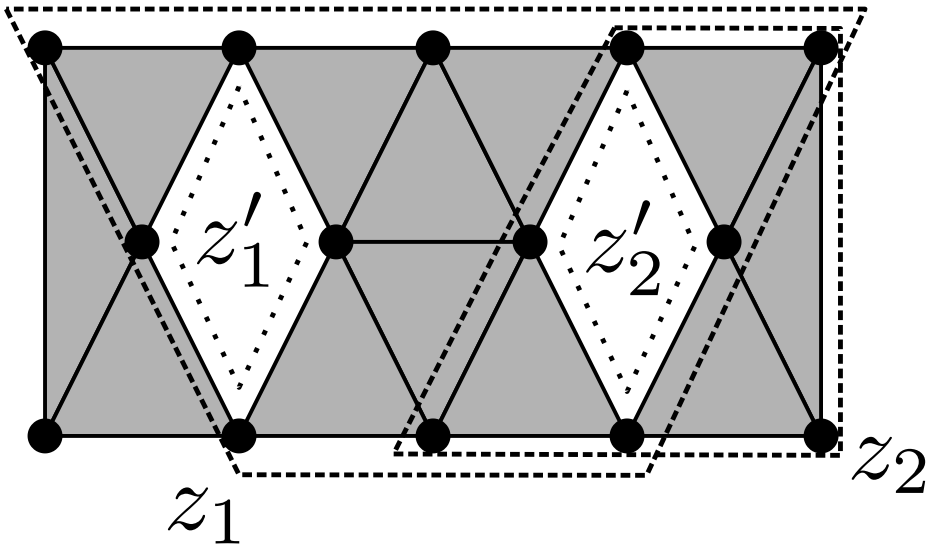
$$\begin{aligned} &\text{minimize} \|z + \partial y\|_1 \\ &\text{where } y \in C_{\ell+1} \end{aligned}$$

The case of two holes (このへんはスキップ)



$H_1 = \langle [z_1], [z_2] \rangle$. We want to find z_1' and z_2' from z_1 and z_2.

The case of two holes (このへんはスキップ)



$H_1 = \langle [z_1], [z_2] \rangle$. We want to find z_1' and z_2' from z_1 and z_2.
Note that $[z_1'] = [z_1] + [z_2]$, $[z_2'] = [z_2]$, we can find z_1' by filling in the hole represented by $[z_2]$.

This corresponds to $Z_1/(B_1 \oplus \langle z_2 \rangle)$. And we formalize the problem as follows:

$$\begin{aligned} &\text{minimize } \|x\|_1 \\ &\text{subject to } x = z_1 + \partial y + kz_2 \\ &\text{where } x \in C_1, y \in C_2, k \in Q \end{aligned}$$

We can also solve the problem using linear programming. We can also find z_2' from z_2 and z_1'.

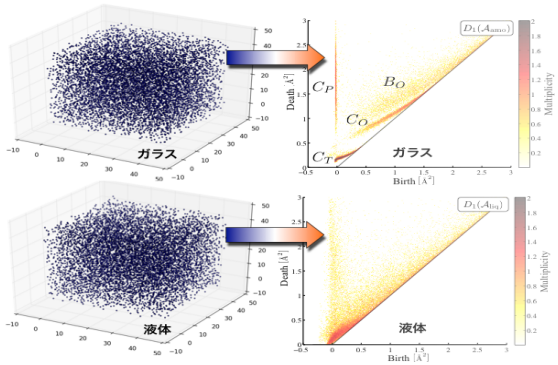
Note

- 今のところ計算ソフトウェアは公開されていない
  - 最適化のソフトウェアライセンスの問題

Continuation

Inverse Problem

ここでは別の形の「逆問題」を考える．通常パーシステント図をポイントクラウドから計算してその幾何的特徴を調べる．



では逆向きの計算はできないだろうか？ つまりパーシステント図からポイントクラウドを構成できないだろうか？ こういうことが実現できればポイントクラウドをパーシステント図の視点から設計できるのではないか？

この問題は解が一意ではない．そのため適当な制約を付け加えることで適切な問題にする．

基準となるポイントクラウドを一つ用意し，
その基準に最も近いポイントクラウドで，
逆問題の解となるものを探す

この基準となるポイントクラウドは *initial point cloud* と呼ぶ

Persistence map

問題の数学的な定式化のため，「persistence map」を以下のように定義する．

ポイントクラウド $P = \{u_1, \cdots, u_M \in \mathbb{R}^L\}$ から ℓ 次のパーシステント図 $D_\ell = \{(b_i, d_i)\} \cup \{(b_i, +\infty)\}$

$f : \mathbb{R}^n \to \mathbb{R}^m$ への対応を多次元の写像として定義

$$u = (u_{11}, \cdots, u_{1L}, u_{21}, \cdots, u_{2L}, \cdots u_{ML}) \in \mathbb{R}^n$$
$$\mapsto (b_1, d_1, \cdots, b_s, d_s, b_{s+1}, \cdots, b_{s+t}) \in \mathbb{R}^m$$

where $n = LM$, $\quad m = 2s + t$.

すると問題は与えられた v に対し $f(u) = v$ を解け，という問題となる

これは「擬逆行列」を用いたニュートン法で定義できる.

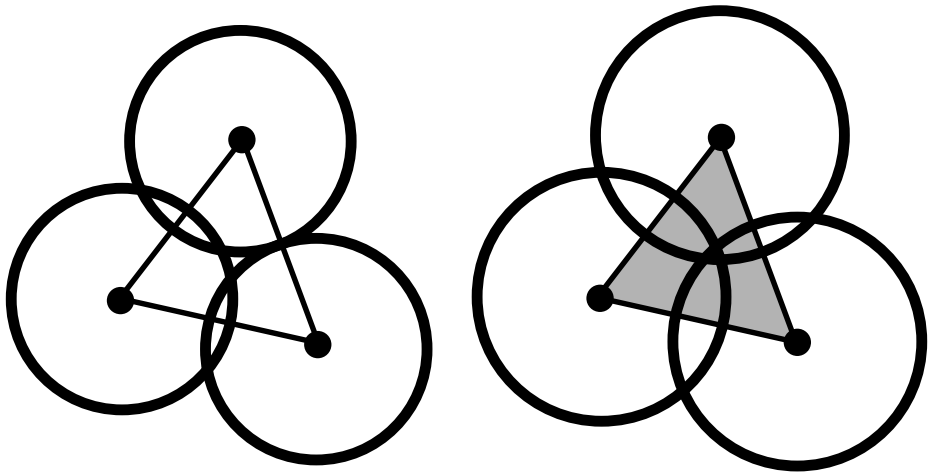
ここで問題となるのがfはちゃんと定義できるのか？という問題: 特に

- 微分できるか？
- パーシステント図の点の個数は変化するのでは？

という問題がある．これは，以下のようにしてある程度解決できる．

- ポイントクラウドがGeneral positionと呼ばれる良い条件を持つ
- 対角線付近のbirth-death pairはこの定義に含めない

- 実は，birth death pair が増えたり減ったりするのは対角線付近のみであることが数学的にわかっている．そこで対角線の周辺を無視すれば個数は (そう簡単には) 変化しない
- 微分可能性は以下の図からわかる



すると定式化は以下の通り

$$\text{minimize } \left\| u - u^* \right\| \text{, subject to } f(u) = v^t$$

u^* が initial point cloud で，v^t が目標となるパーシステント図.

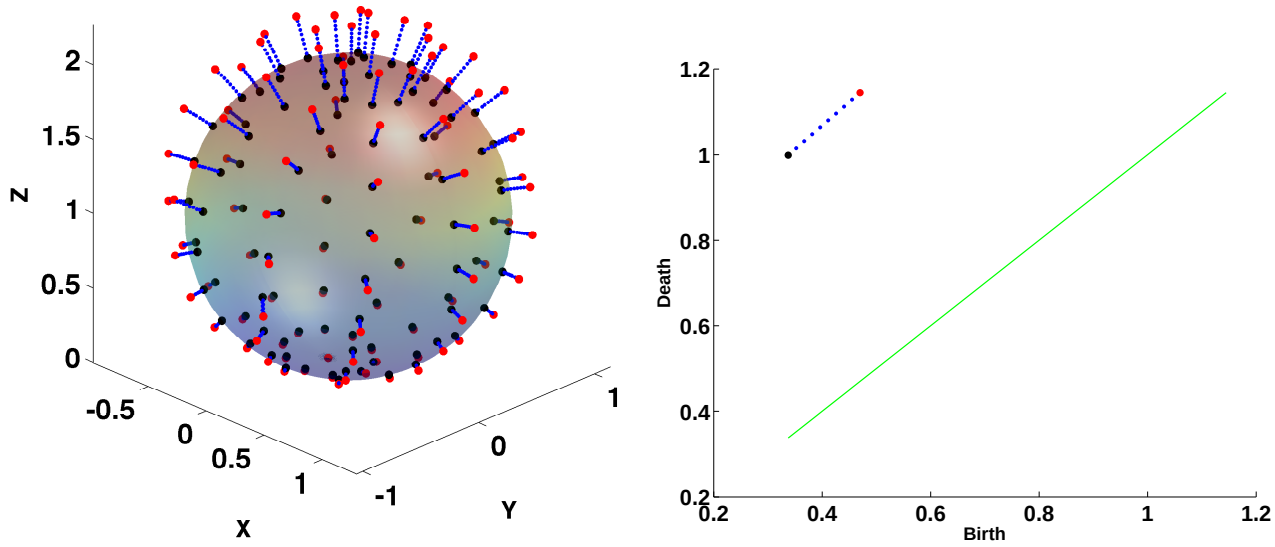
すると定式化は以下の通り

$$\text{minimize } \left\| u - u^* \right\| \text{, subject to } f(u) = v^t$$

u^* が initial point cloud で，v^t が目標となるパーシステント図.
この問題は「擬逆行列」を用いたニュートン法で計算できる．ニュートン法の初期点を u^* にすればよいのである.

Example

Initial point cloud: 左図の黒点
initial point cloud のパーシステント図: 右図の黒点
目標のパーシステント図: 右図の赤点
得られた解: 左図の赤点



逆問題まとめ

- パーシステント図→元のデータという向きの問題も重要
- 三つのアイデア
  - birth/death simplices
  - optimal cycles
  - continuation

参考文献, URL など

- Perseus: `http://www.sas.upenn.edu/~vnanda/perseus/`
- JavaPlex: `http://appliedtopology.github.io/javaplex/`
- Dipha: `https://github.com/DIPHA/dipha`
- Gudhi: `https://project.inria.fr/gudhi/`
- ripser: `https://github.com/Ripser/ripser`

- Optimal cycle: Dey, T. K., Hirani, A. N., and Krishnamoorthy, B., Optimal homologous cycles, total unimodularity and linear programming, SIAM Journal on Computing, **40**(4) (2011), 1026–1044.
- Optimal cycle for persistent homology: Escolar, E. G., and Hiraoka, Y., Optimal Cycles for Persistent Homology Via Linear Programming, Optimization in the Real World: Toward Solving Real-World Optimization Problems, Springer Japan (2016), 79–96.
- Continuation: Gameiro, M., Hiraoka, Y., and Obayashi, I., Continuation of point clouds via persistence diagrams, Physica D, **334** (2016), 118–132