

横浜国立大学大学院環境情報学府

情報環境専攻数理科学プログラム

野間研究室 18HJ045 道用悠太朗

代数曲線のブローアップのみを用いた
特異点解消コンピュータプログラム

目次

- ▶ 特異点解消とは
- ▶ 研究動機
- ▶ 特異点解消のアルゴリズムとデータ構造

特異点解消 - RESOLUTION

特異点解消とは？

特異点解消 - RESOLUTION

特異点解消とは？

特異点の解消を考える

特異点解消 - RESOLUTION

特異点解消とは？

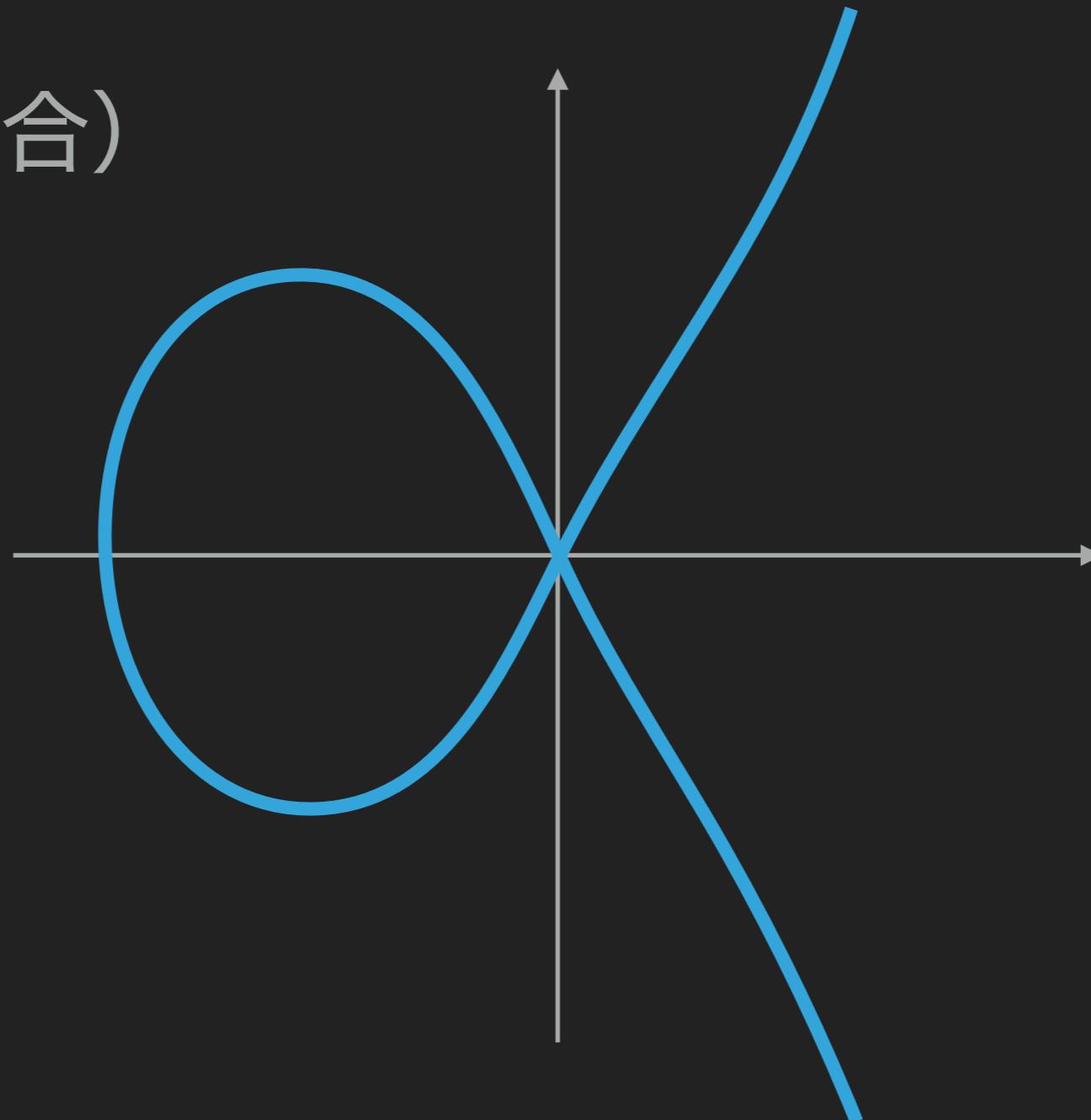
特異点の解消を考える

技術的特異点（シングュラリティ）

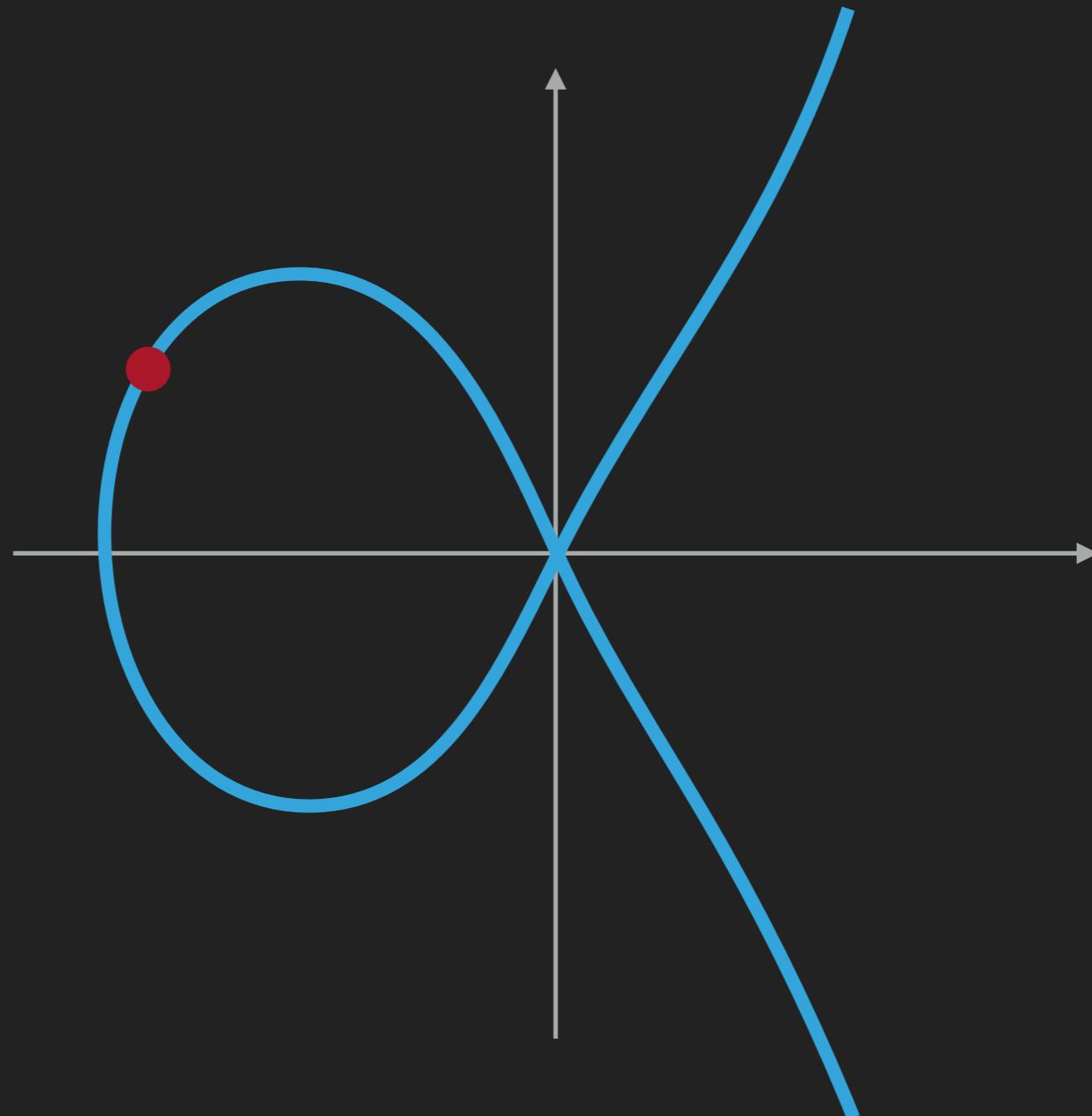


特異点解消 - RESOLUTION

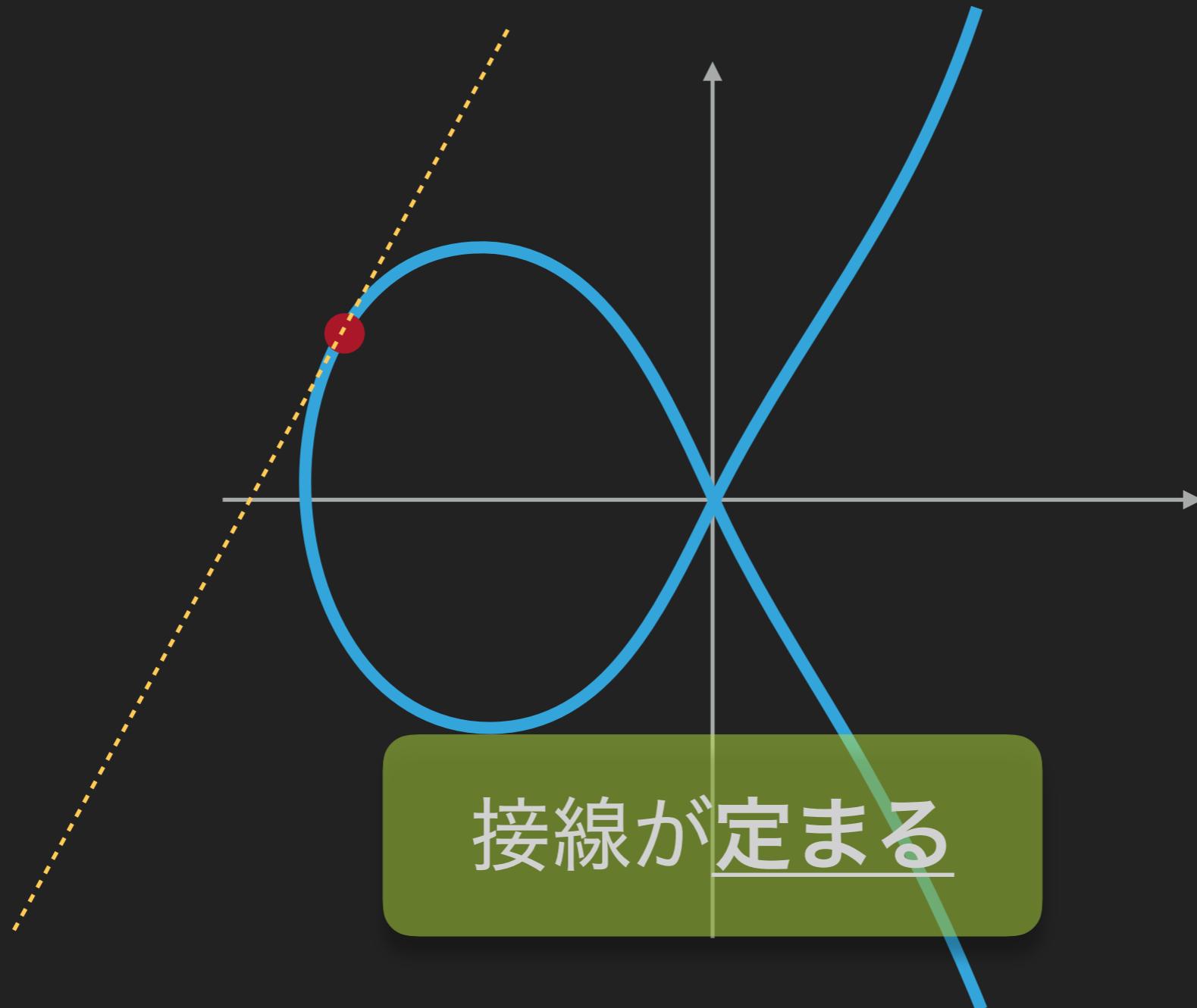
(曲線の場合)



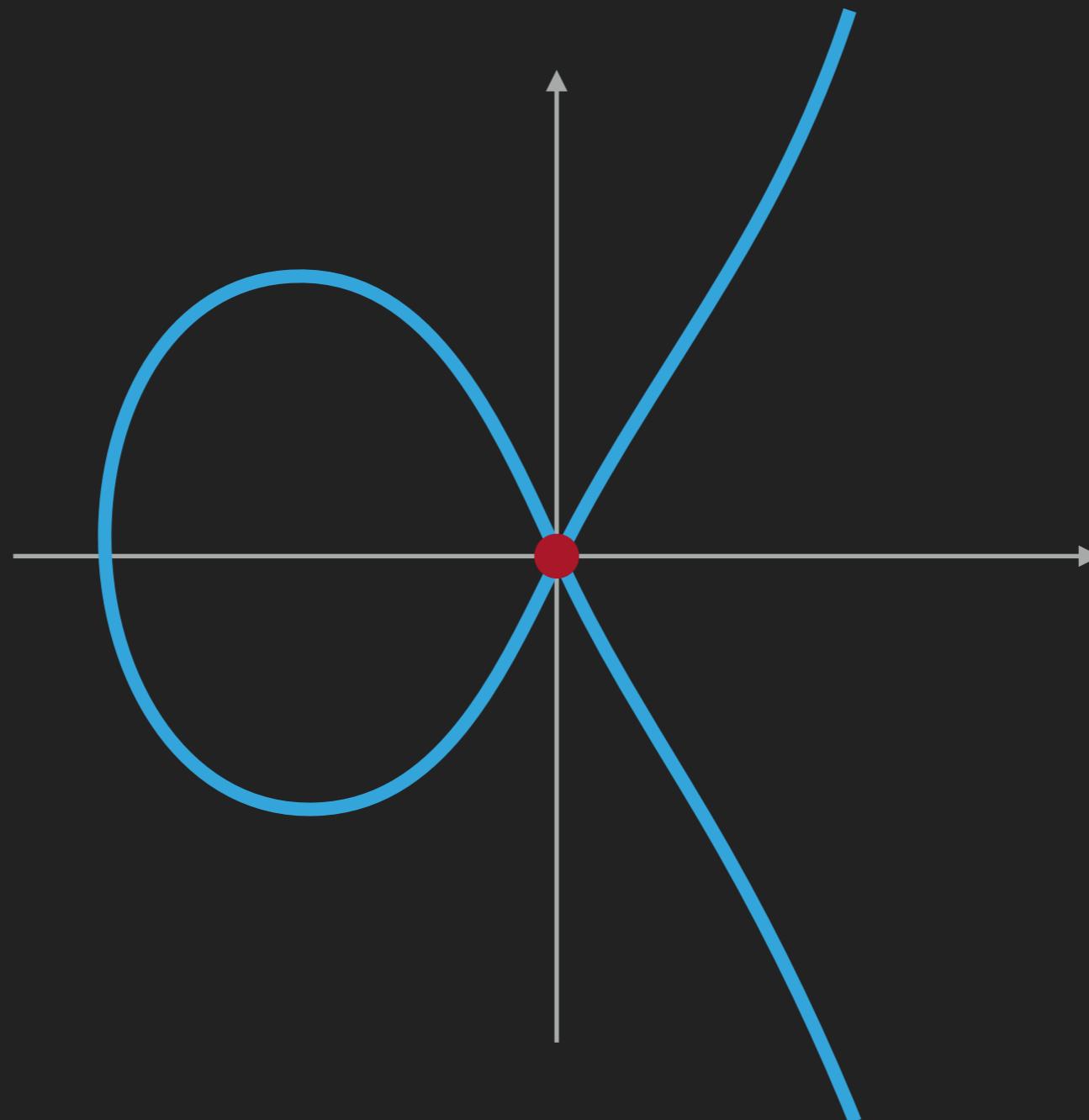
特異点解消 - RESOLUTION



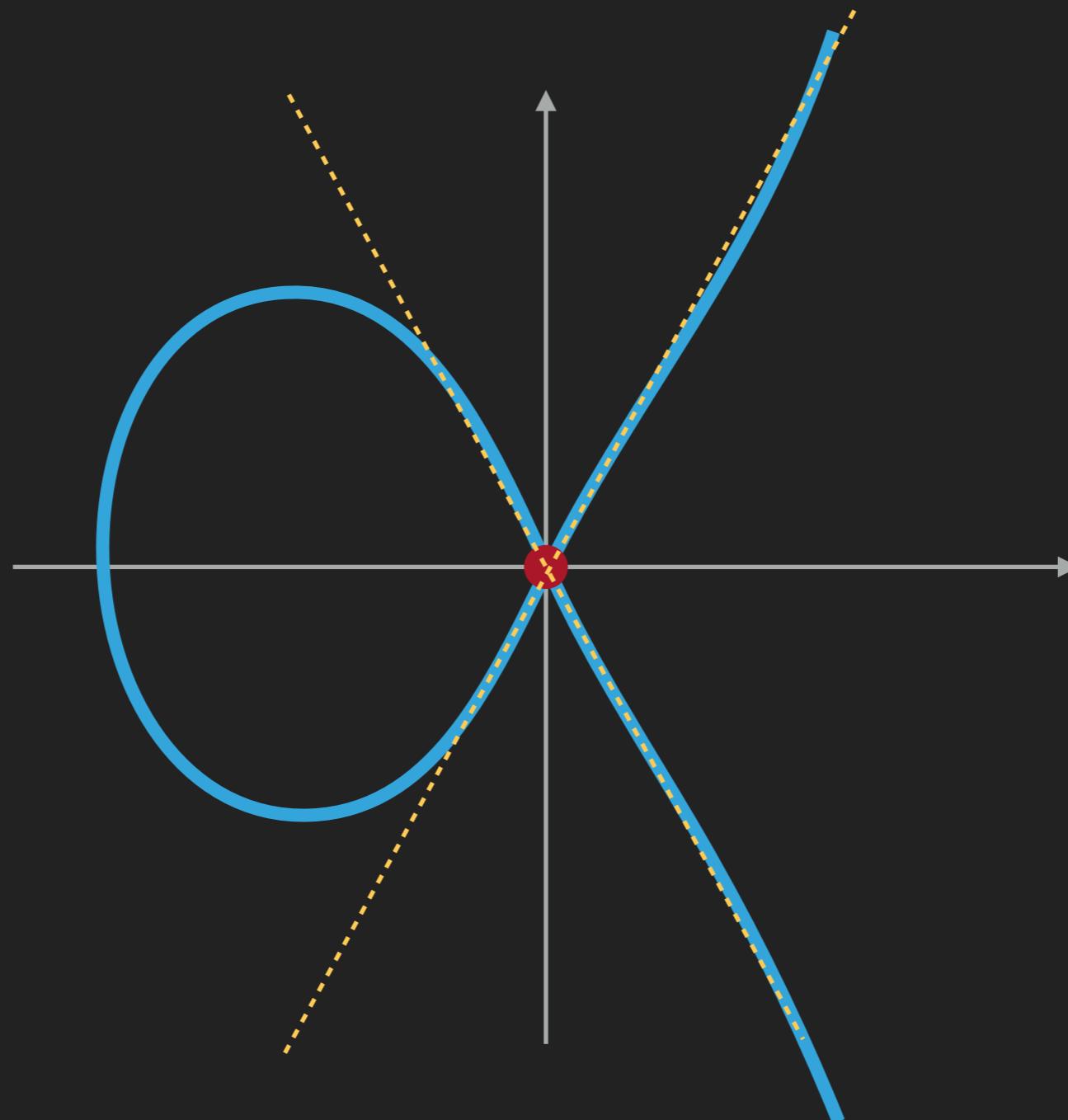
特異点解消 - RESOLUTION



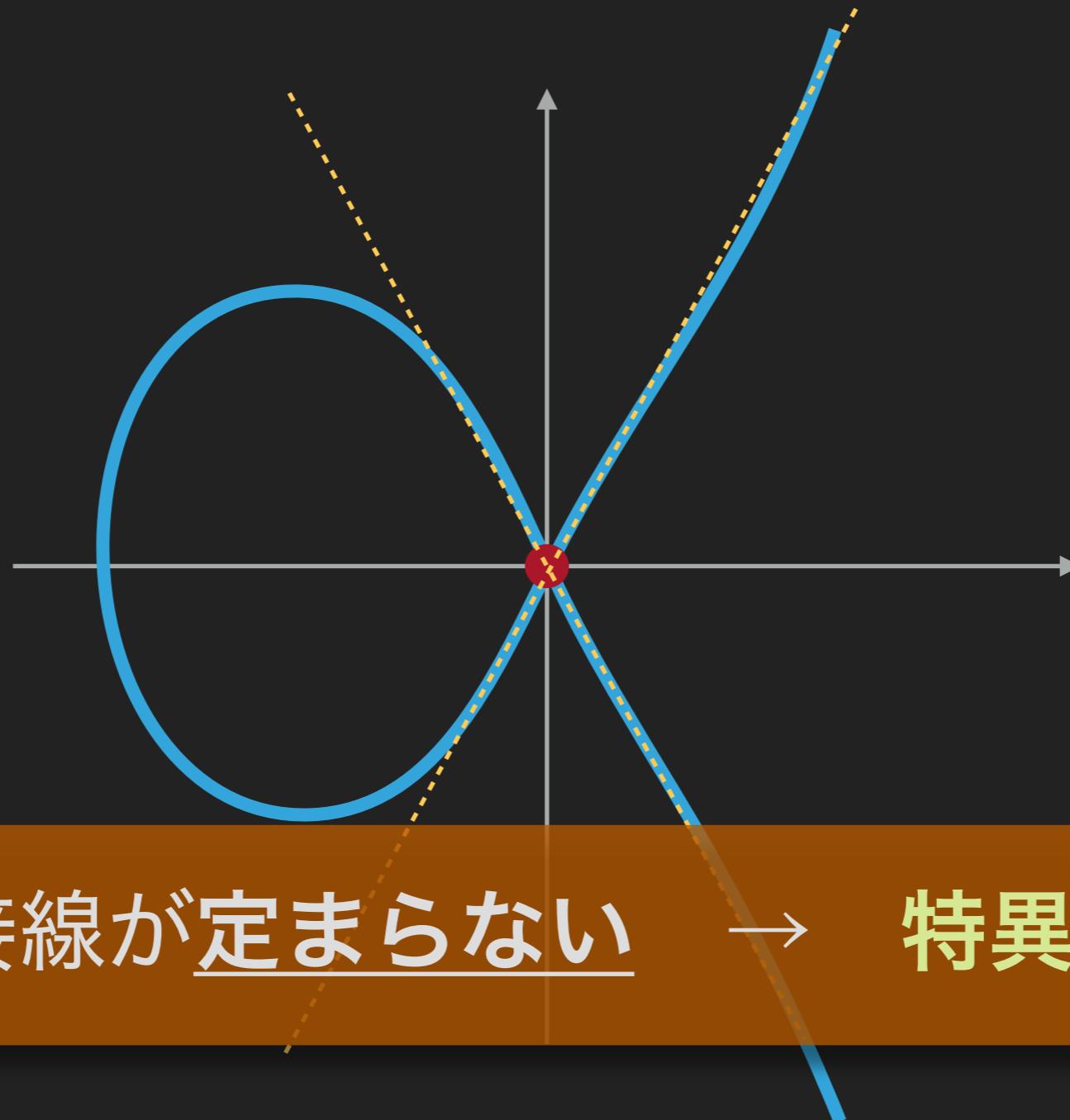
特異点解消 - RESOLUTION



特異点解消 - RESOLUTION



特異点解消 - RESOLUTION



接線が定まらない

→ 特異点

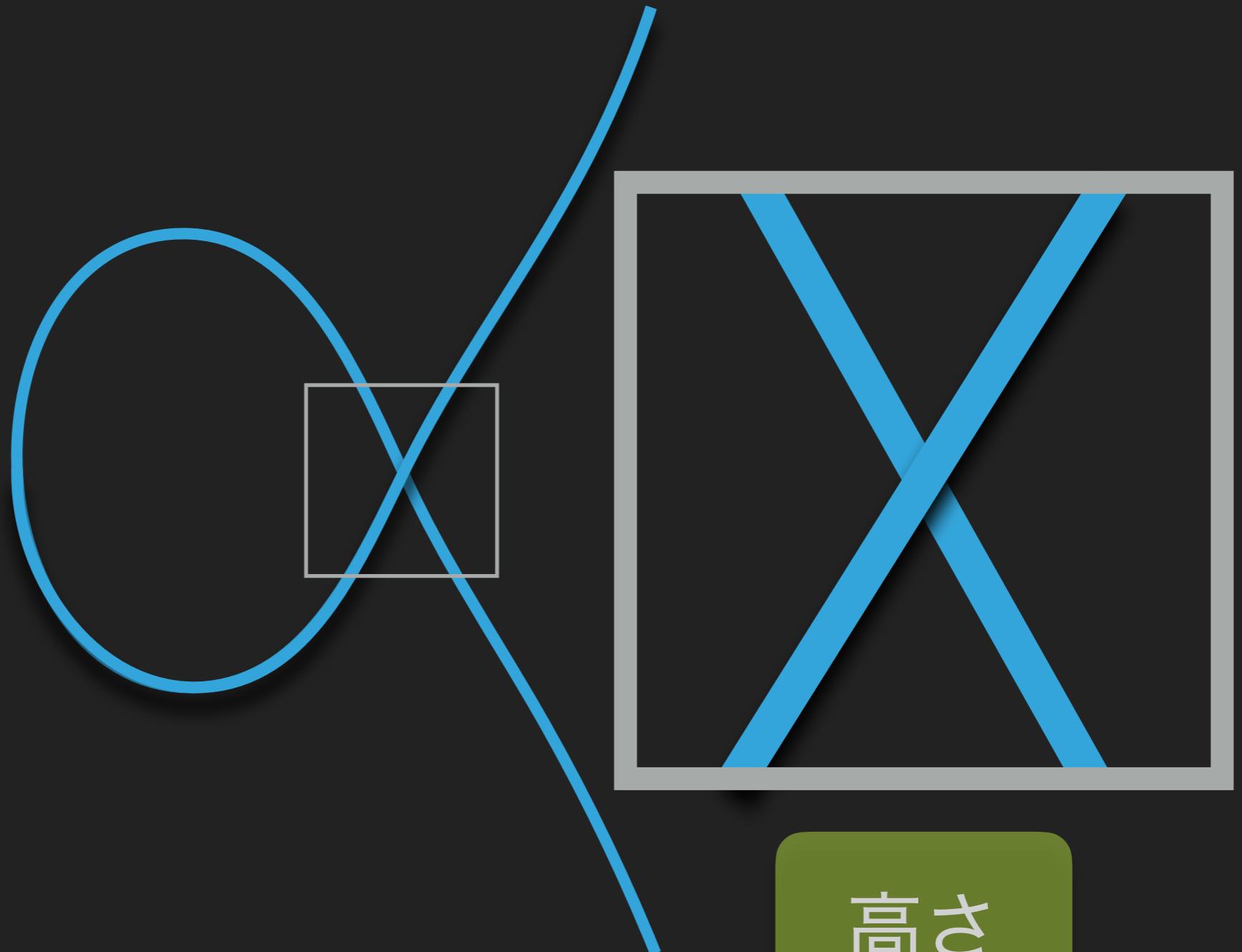
特異点解消 - RESOLUTION



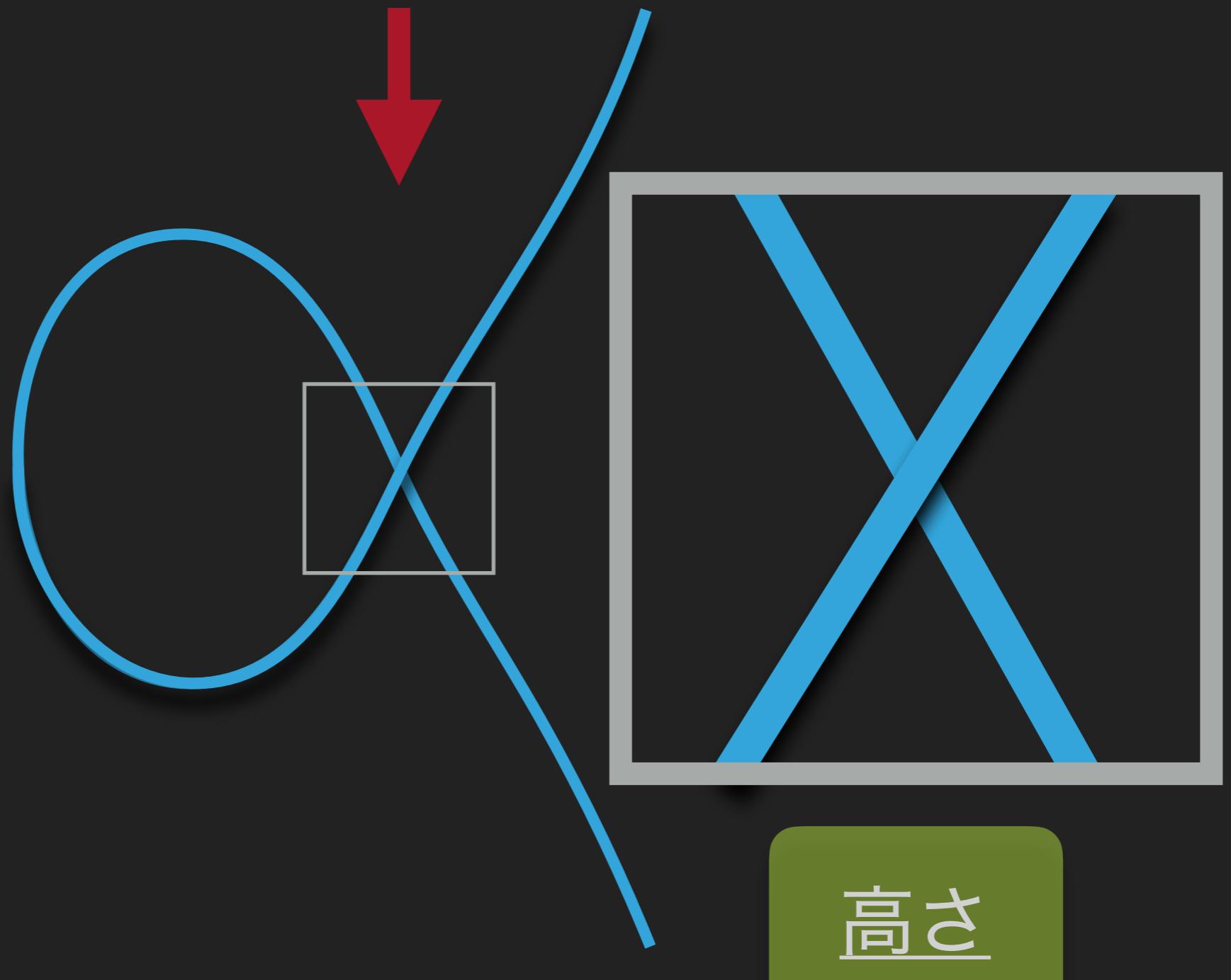
特異点解消 - RESOLUTION



特異点解消 - RESOLUTION

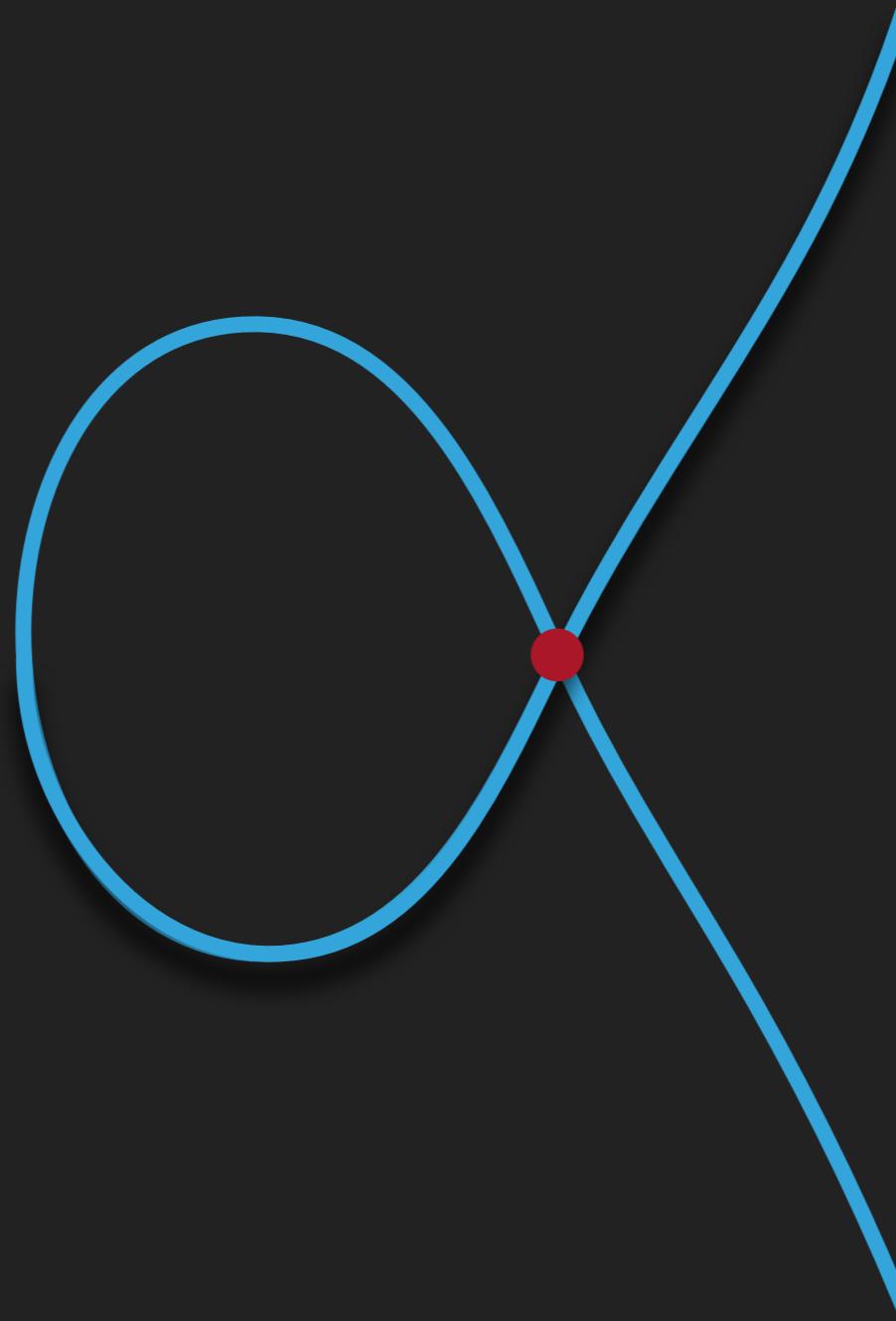


特異点解消 - RESOLUTION

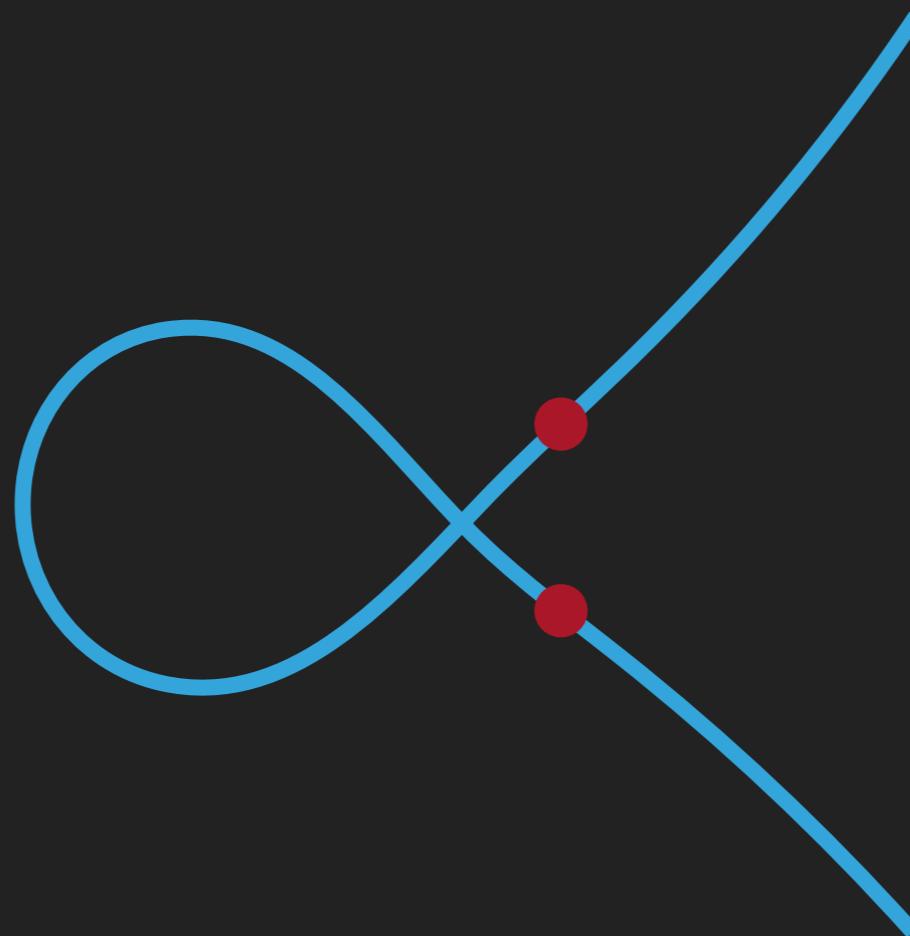


高さ

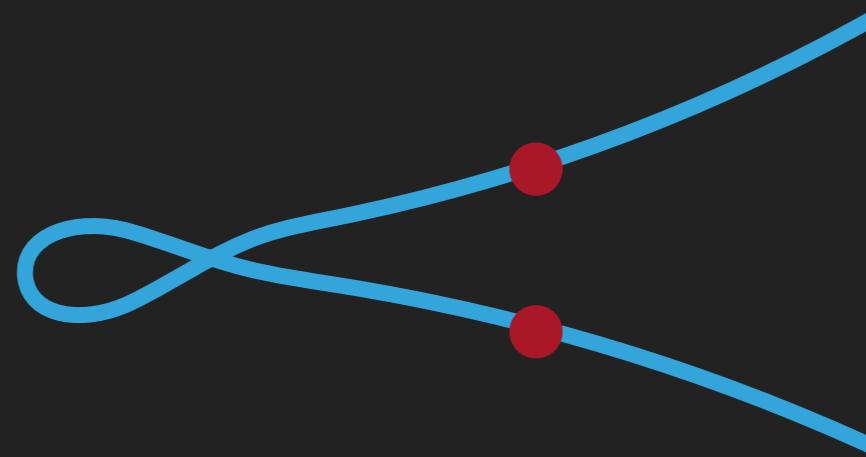
特異点解消 - RESOLUTION



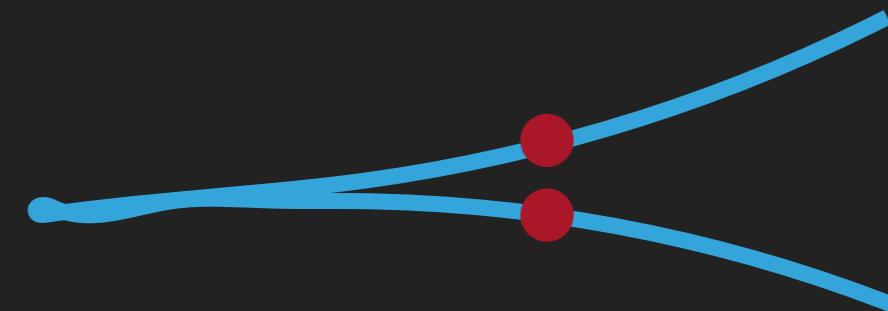
特異点解消 - RESOLUTION



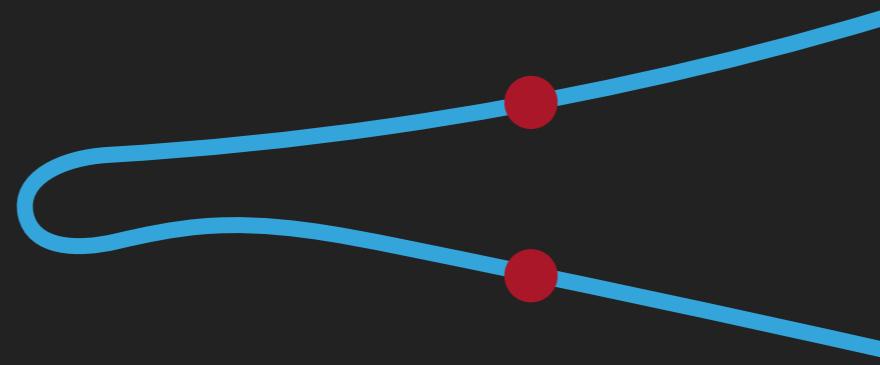
特異点解消 - RESOLUTION



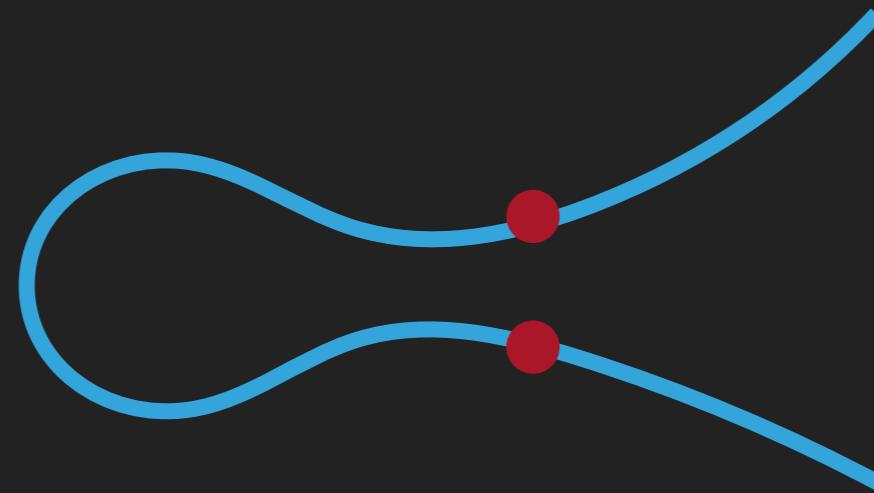
特異点解消 - RESOLUTION



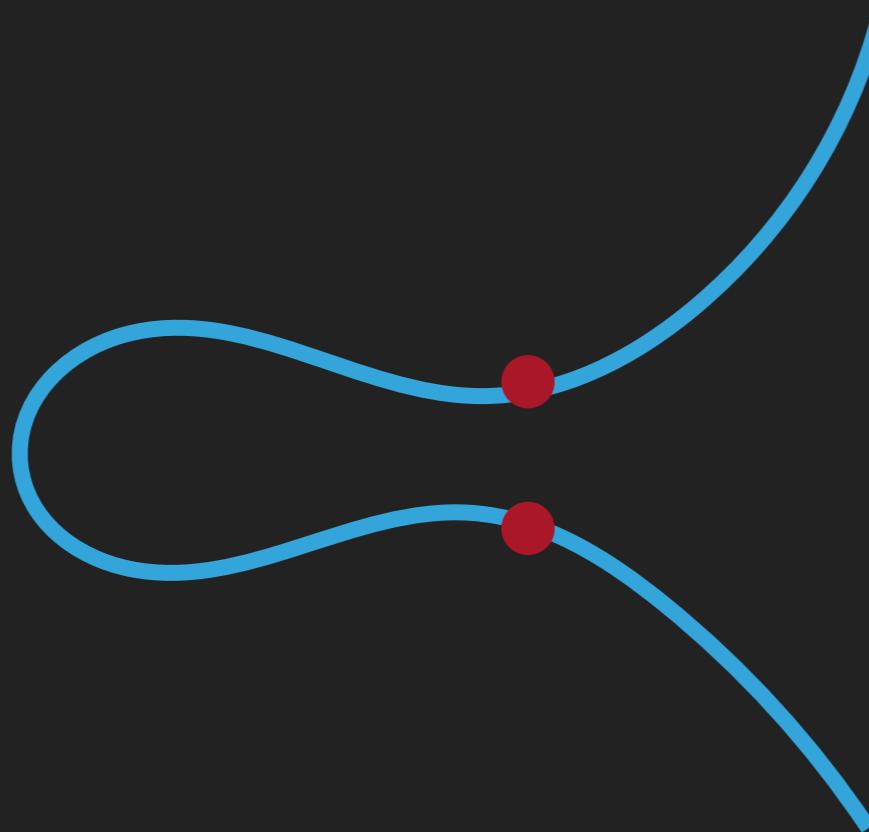
特異点解消 - RESOLUTION



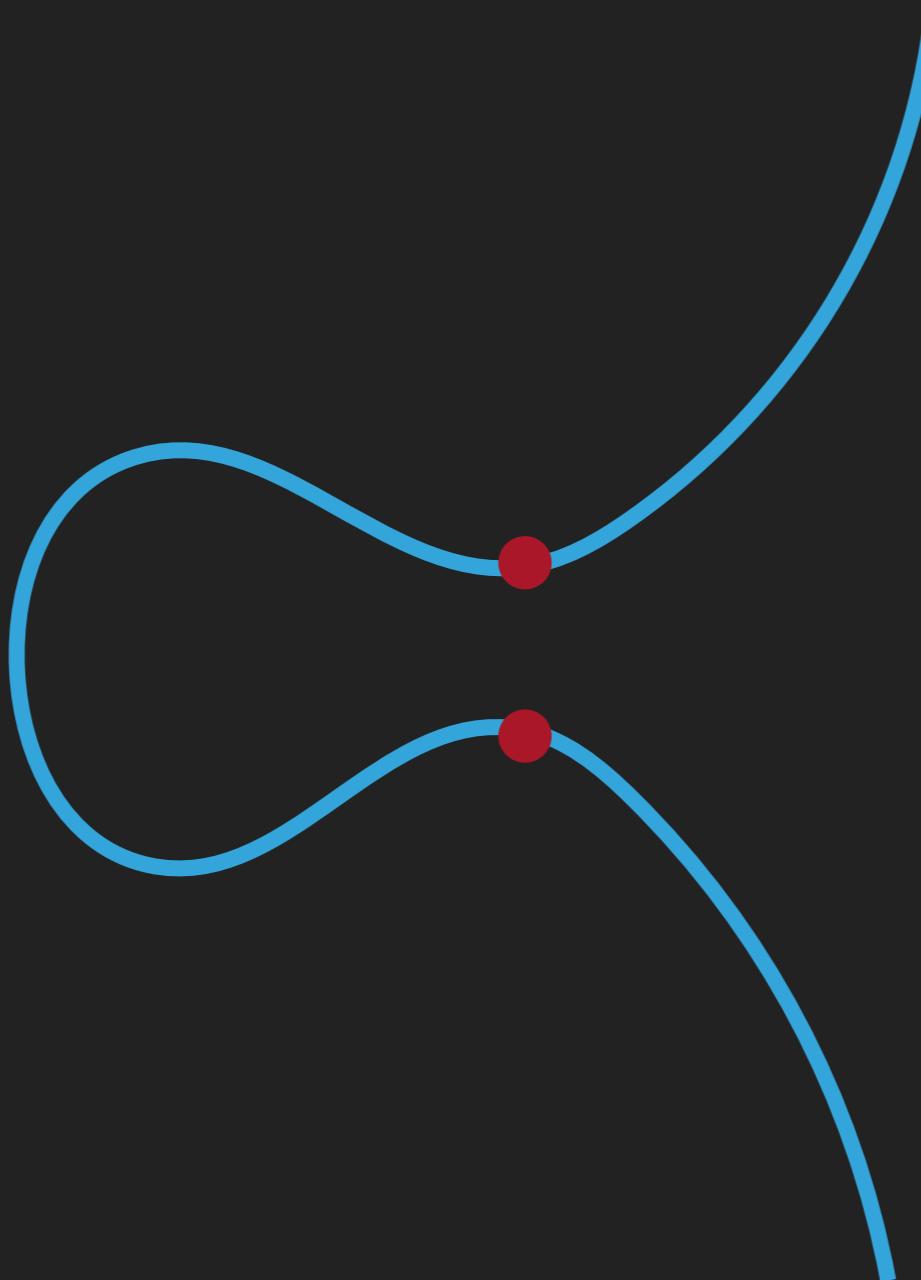
特異点解消 - RESOLUTION



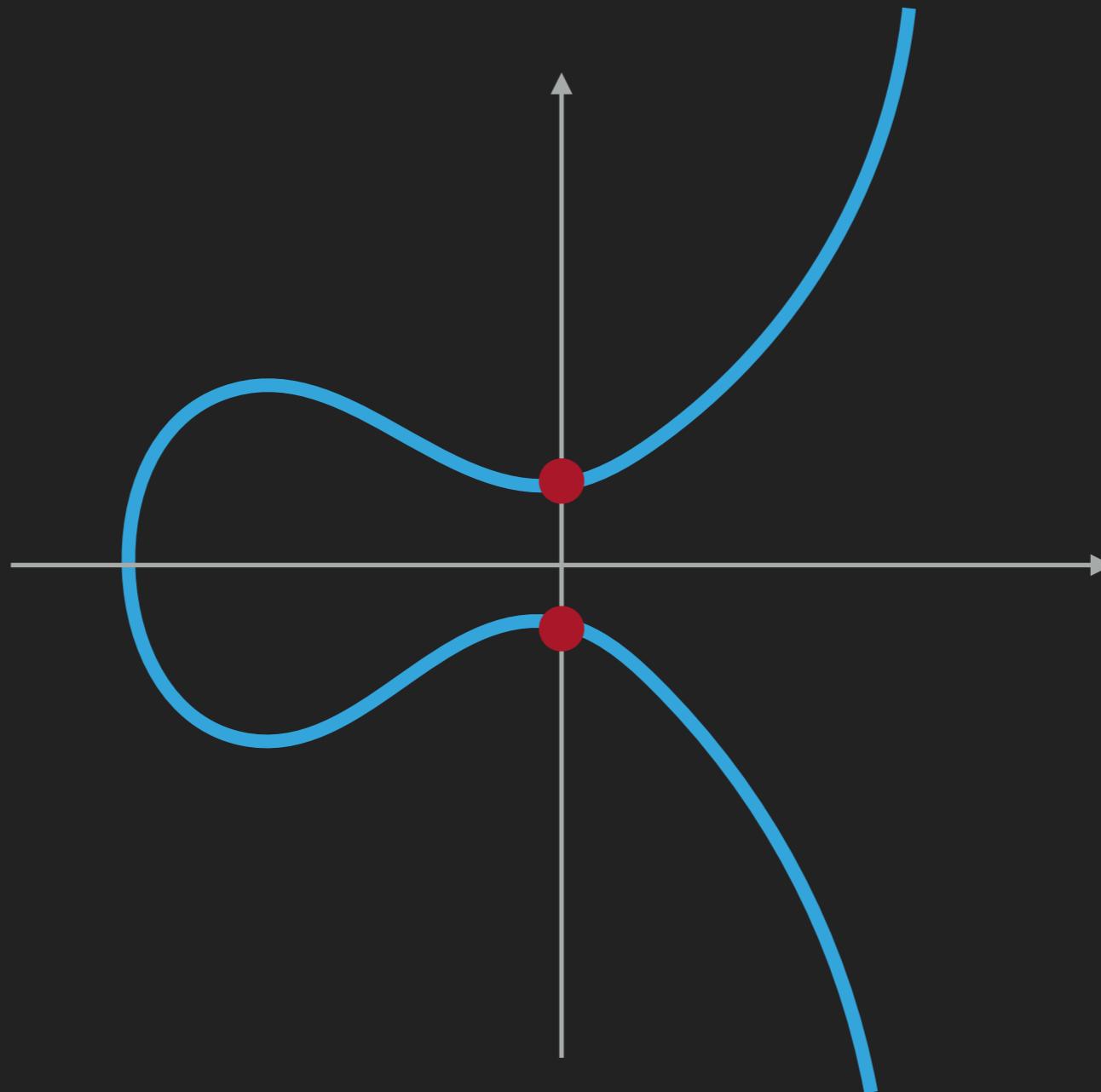
特異点解消 - RESOLUTION



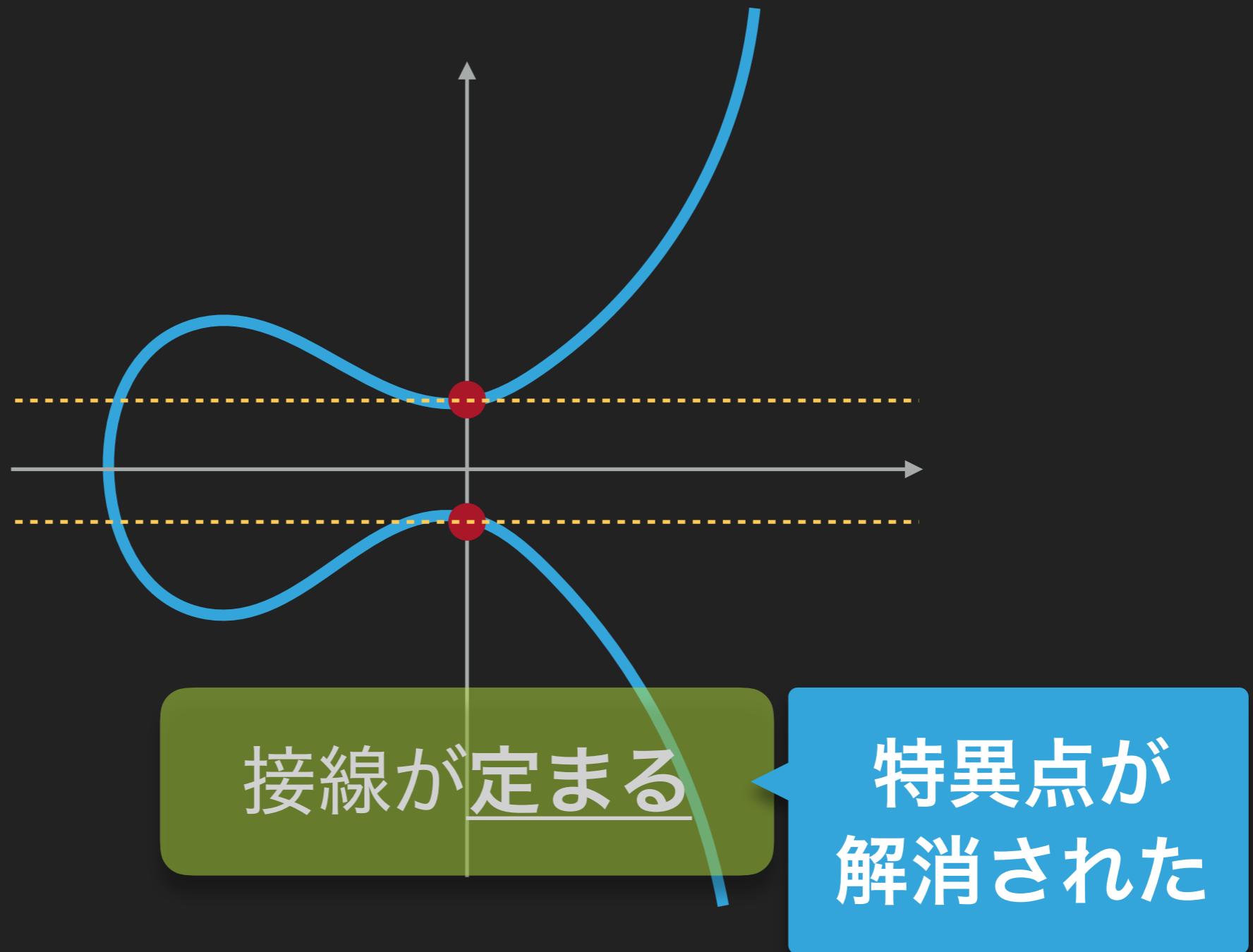
特異点解消 - RESOLUTION



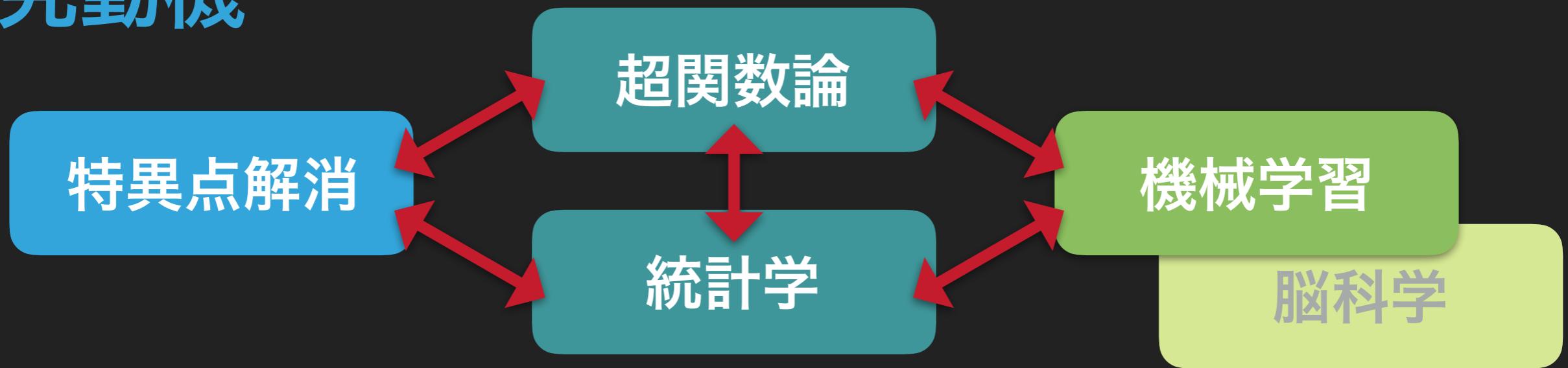
特異点解消 - RESOLUTION



特異点解消 - RESOLUTION



研究動機



情報科学・統計学との強い関連性

「ブローアップは（手計算が）煩雑」

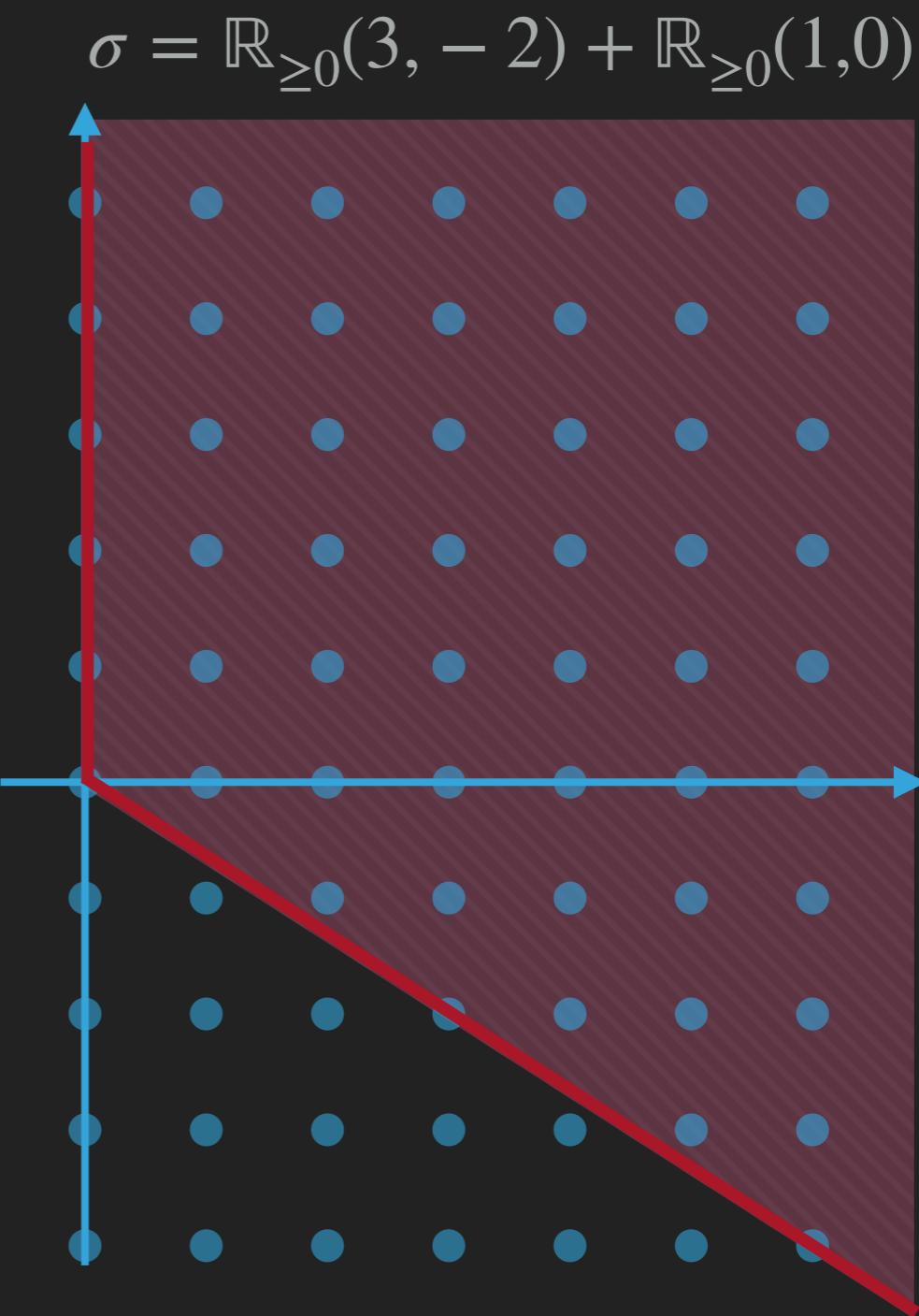
研究動機

	ブローアップ のみを用いる方法	ブローアップ 以外も用いる方法
代数幾何 との対応	○ (代数幾何的)	△ (算術的)
実装の 困難性	?	○
計算量	大きい	小さい

研究動機

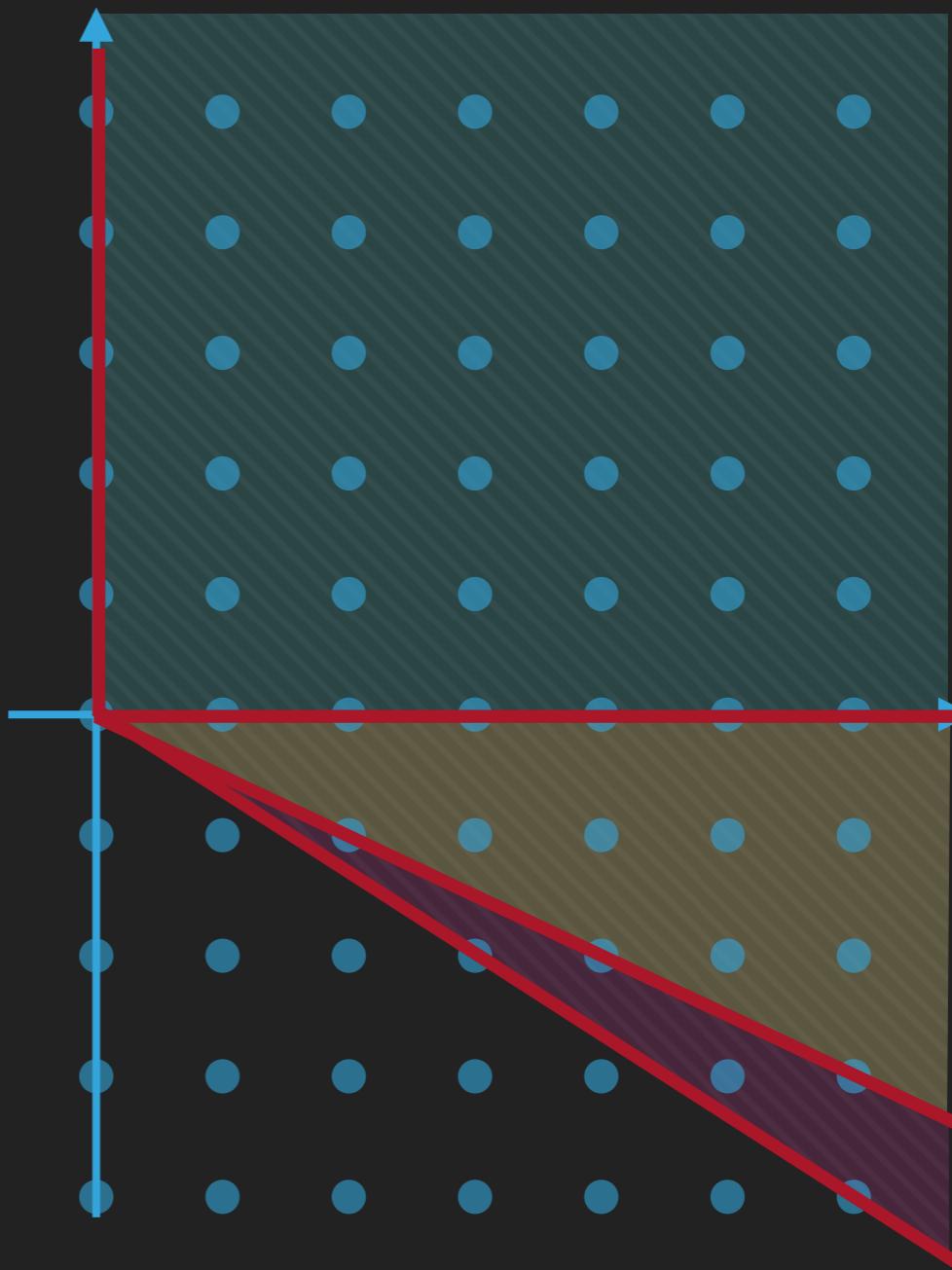
曲面 $y^3 - xz = 0$

特異点を持つ



研究動機

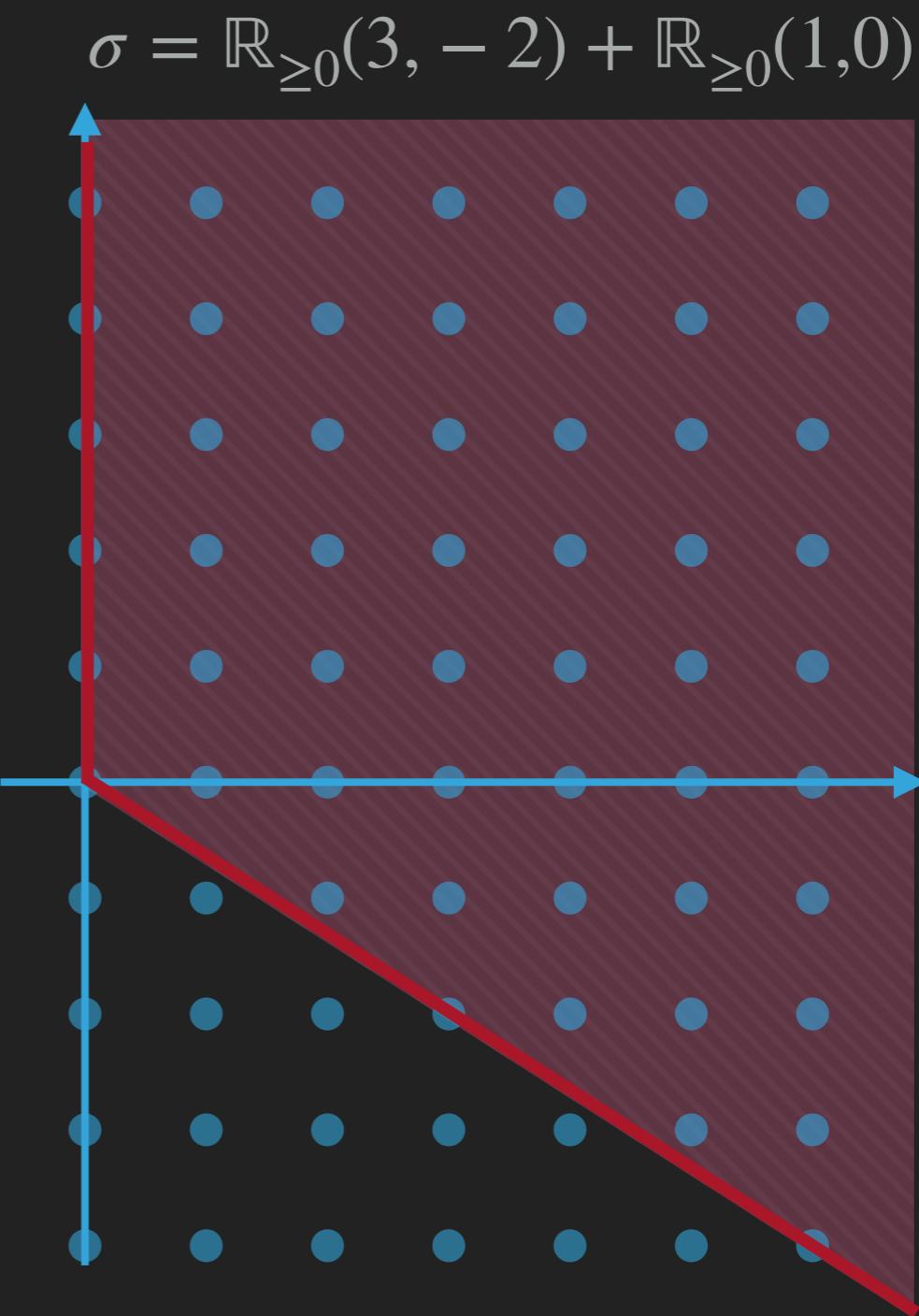
$y^3 - xz = 0$ の特異点解消



研究動機

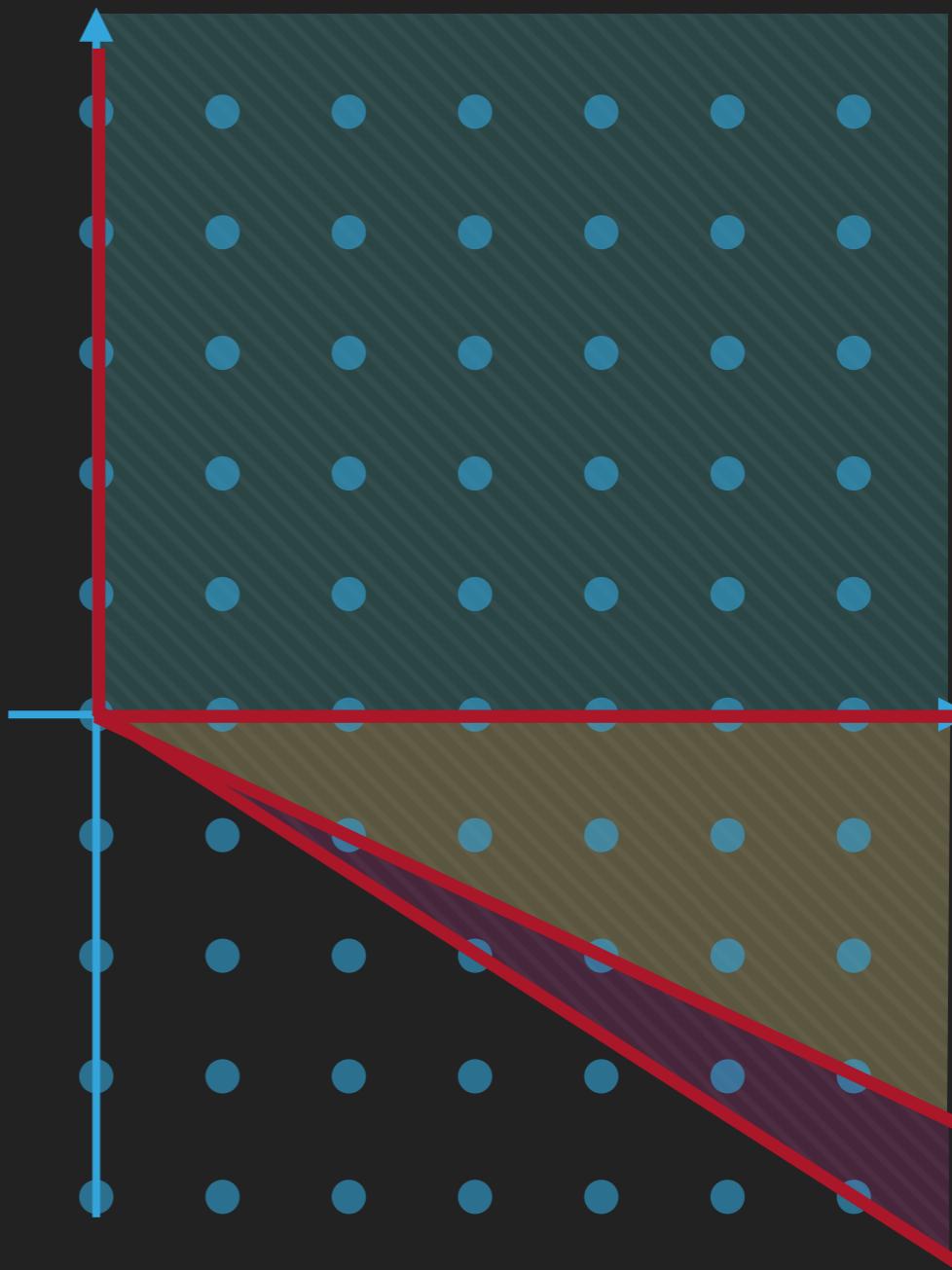
曲面 $y^3 - xz = 0$

特異点を持つ



研究動機

$y^3 - xz = 0$ の特異点解消



研究動機

よく分からぬ



研究動機

なぜ よく分からない のか？

- ▶ 図形がどのように変化しているのか？
- ▶ 図形のどのような性質を用いているのか？
- ▶ その対応は本当に正しいのか？

研究動機 (再掲)

	ブローアップ のみを用いる方法	ブローアップ 以外も用いる方法
代数幾何 との対応	○ (代数幾何的)	△ (算術的)
実装の 困難性	?	○
計算量	大きい	小さい

研究動機

ブローアップ
のみを用いる方法

ブローアップ
以外も用いる方法

代数幾何
との対応

(代数幾何的)



(算術的)

実装の
困難性

代数幾何の対象（多項式・開集合など）を
可能な限りモデル化して計算機で扱う

計算量

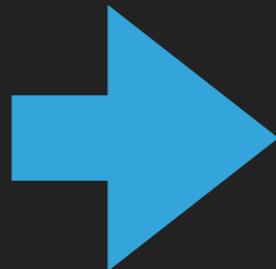
抽象的なものをどのような手法で
モデル化するか？データ構造は？

プログラム

入力

$$f = x^a - y^b$$

(ただし $\text{GCD}(a, b) = 1$)



出力

計算結果・双対グラフ

- ブローアップのみを使用
- 正規交差を持つまで行う

実行例

曲線 $C : x^3 - y^2 = 0$ の特異点解消

resolution.py

```
masters_thesis.tex resolution.py blowing_up.py abs1sj.tex exceptional_curve.py
```

1
2
3 Resolution of Curve Singularities
4 2019 by dafuyafu
5
6
7 from sympy import *
8 from blowing_up import blowing_up as bu
9
10 if __name__ == '__main__':
11 x = symbols('x')
12 y = symbols('y')
13 f =
14 # f = x ** 3 - (x - y) ** 2 # 2,3-cusp with co-tr
15 # f = x ** 25 - y ** 19 # 19,25-cusp
16 # f = x ** 3 + x ** 2 - y ** 2 # node
17 sing = solve([f, diff(f, x), diff(f, y)])
18 print("f = " + str(f))
19 print("Sing V(f) = " + str(sing))
20 print("")
21 bu.blowing_up(f)

Line 13, Column 9

master [2]

Tab Size: 4

Python

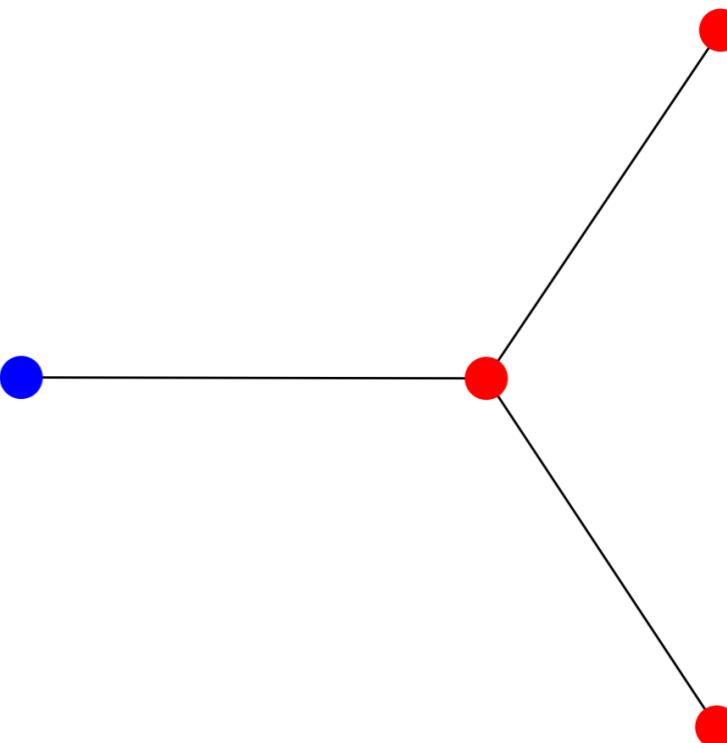


出力結果

```

Sing V(f) = [{x: 0, y: 0}]
resolution of Curve Singularities
019 by dafuyafu
on AA(x, y/x):
    f_x = x - y**2 : nonsingular
    Exc: V(x): not normal crossing
sympy.simplify:
coordinate translation up as bu
    {x: 0, y: 0} --> {x: 0, y: 0}
name f_x = x - y**2 :
    on AA(x, y/x**2):
        f_x = -x*y**2 + 1 : nonsingular
        = x ** -y ** 2
        f = x * y ** 2 # 3-cusp with co-tr
        f = on AA(x**2/y, y/x):
        f = x * f_y = x - y : nonsingular
        sing = soExc(V(y):f, not, normal crossing)
        print("f coordinate translation")
        print("Sing V(f) = {y: 0, x: 0} --> {x: 0, y: 0}")
        print("") f_y = x - y
        u.blowing_up(f)
        on AA(x**2/y, y**2/x**3):
            f_x = 1 - y : nonsingular
            Exc: V(x): normal crossing
        on AA(x**3/y**2, y/x):
            f_y = x - 1 : nonsingular
            Exc: V(y): normal crossing
        on AA(x/y, y):
            f_y = x**3*y - 1 : nonsingular
            Exc: V(y): no crossing
ExceptionalCurve({x, AA(x, y/x)}, {y, AA(x/y, y)}, {x, AA(x**3/y**2, y/x)})
ExceptionalCurve({x, AA(x, y/x**2)}, {y, AA(x**2/y, y/x)}, {y, AA(x**2/y, y**2/x**3)})
ExceptionalCurve({x, AA(x**2/y, y**2/x**3)}, {y, AA(x**3/y**2, y/x)})
NonsingularStrictTransform({1 - y, AA(x**2/y, y**2/x**3)}, {x - 1, AA(x**3/y**2, y/x)})

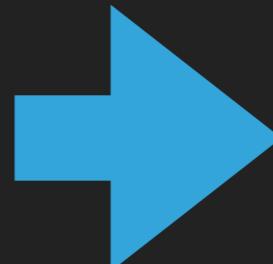
```



プログラム（再掲）

入力

$$f = x^a - y^b$$



出力

計算結果・双対グラフ

プログラム

入力

$$f = x^a - y^b$$

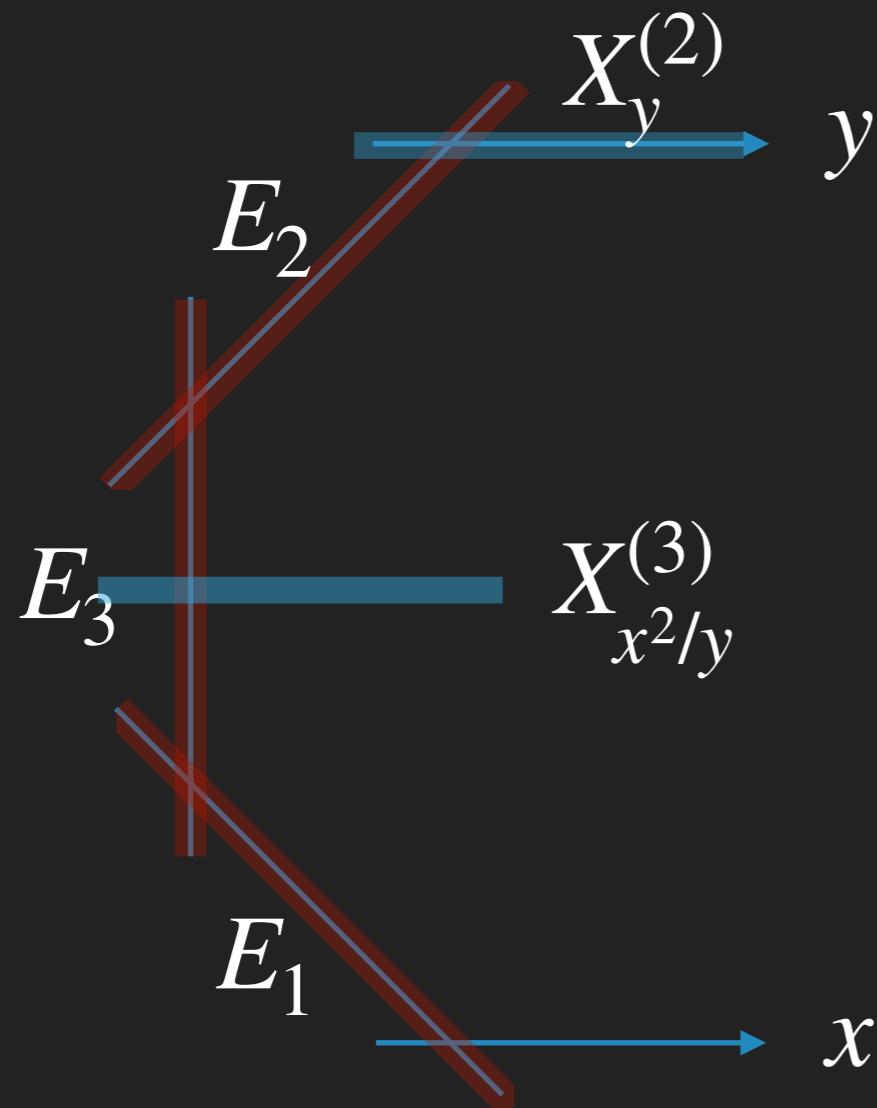
出力

計算結果・双対グラフ

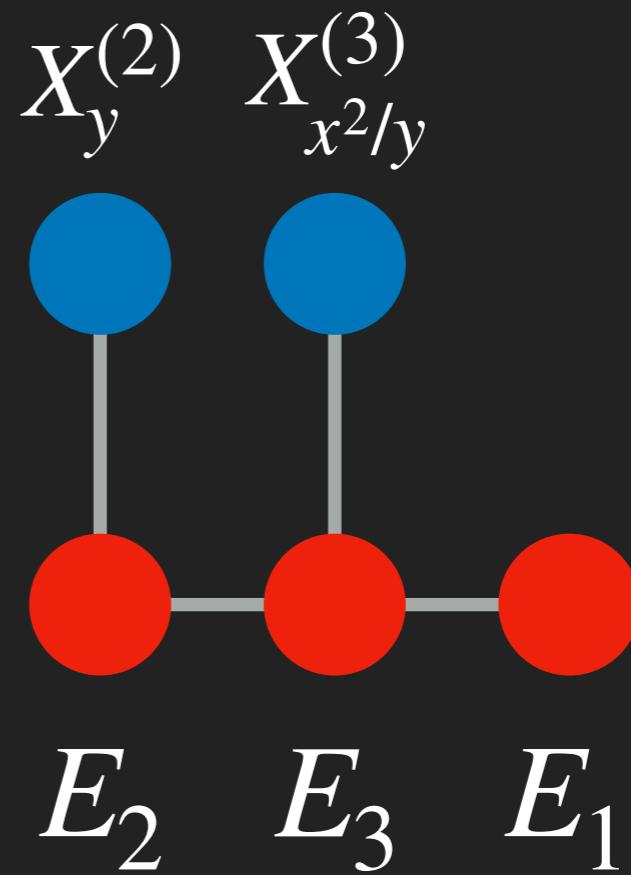
どのようにして
双対グラフを構成するか？

例 $x^3 + xy^3 = 0$

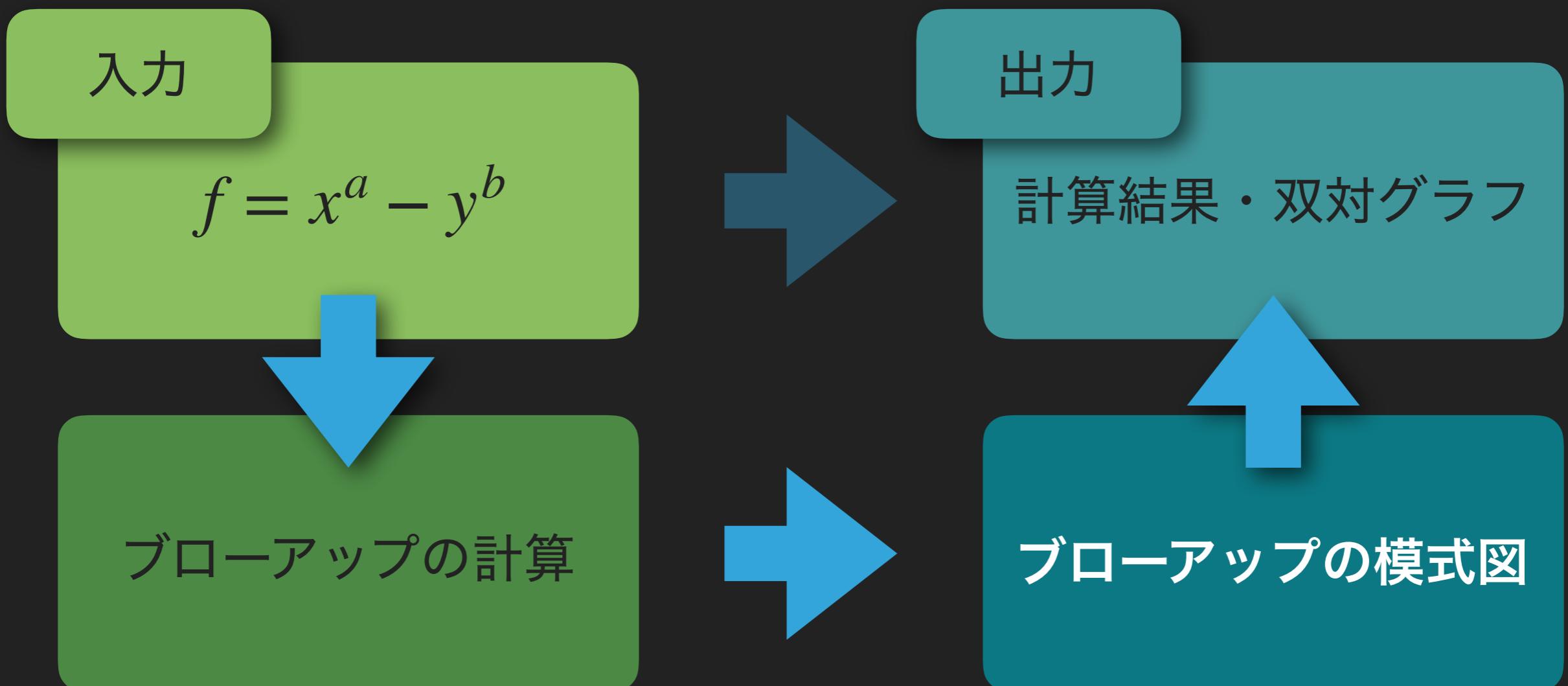
ブローアップの模式図



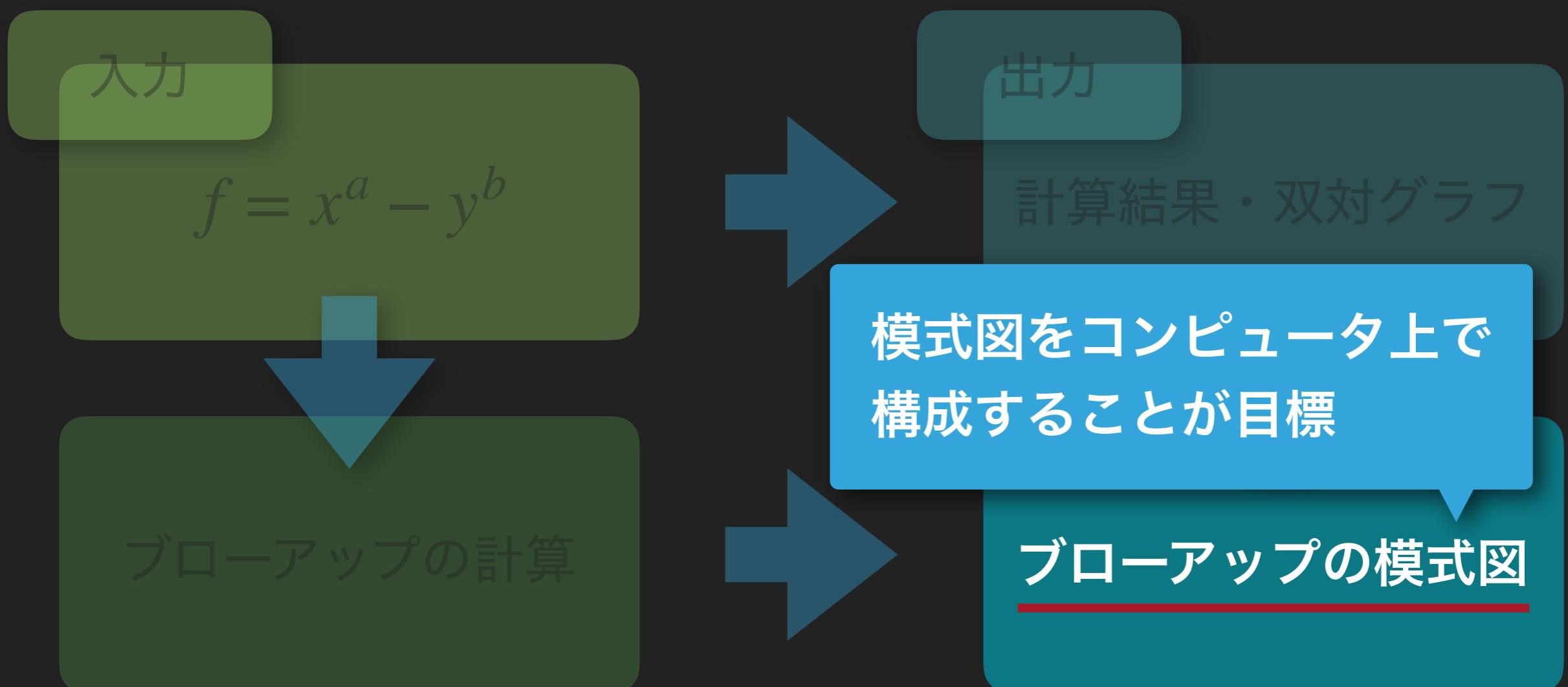
特異点解消の双対グラフ



プログラム



プログラム



プログラム

- ▶ Python
- ▶ SymPy (Python ライブライ)
代数計算ライブラリ. 多項式や有理関数などのデータ構造および
それらのGCDや展開, 既約分解, 求根などのメソッドをサポート.
- ▶ NetworkX (Python ライブライ)
グラフ理論ライブラリ. 双対グラフの出力に使用.

プログラム

ブローアップ
のみを用いる方法

ブローアップ
以外も用いる方法

代数幾何
との対応



実装の
困難性

代数幾何の対象 (多項式・開集合など) を
可能な限りモデル化して計算機で扱う

計算量



データ構造

- ▶ **class AffineOpen**

アフィン開集合を表すクラス

- ▶ **axis = (axis_1, axis_2)**

軸のラベルを保持するタプル

- ▶ **glued = {}**

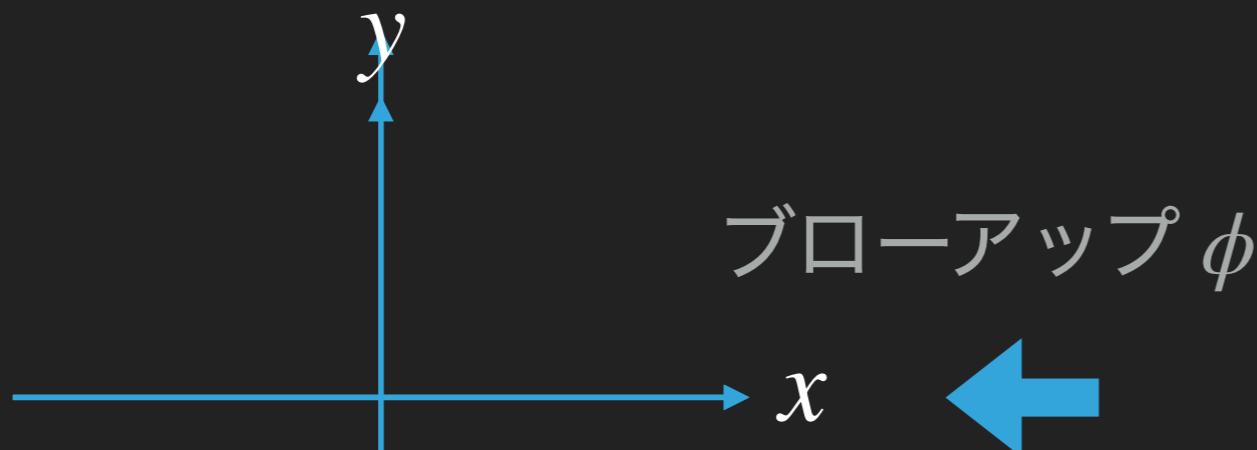
貼り合わされたアフィン開集合とその軸を保持する辞書

データ構造

- ▶ `class AffineOpen`
アフィン開集合を表すクラス
- ▶ `glue()`
アフィン開集合を貼り合わせるメソッド
- ▶ `detach()`
貼り合わさったアフィン開集合を切り離すメソッド
- ▶ `__eq__()`
2つのアフィン開集合が同型かどうかを判定するメソッド

データ構造

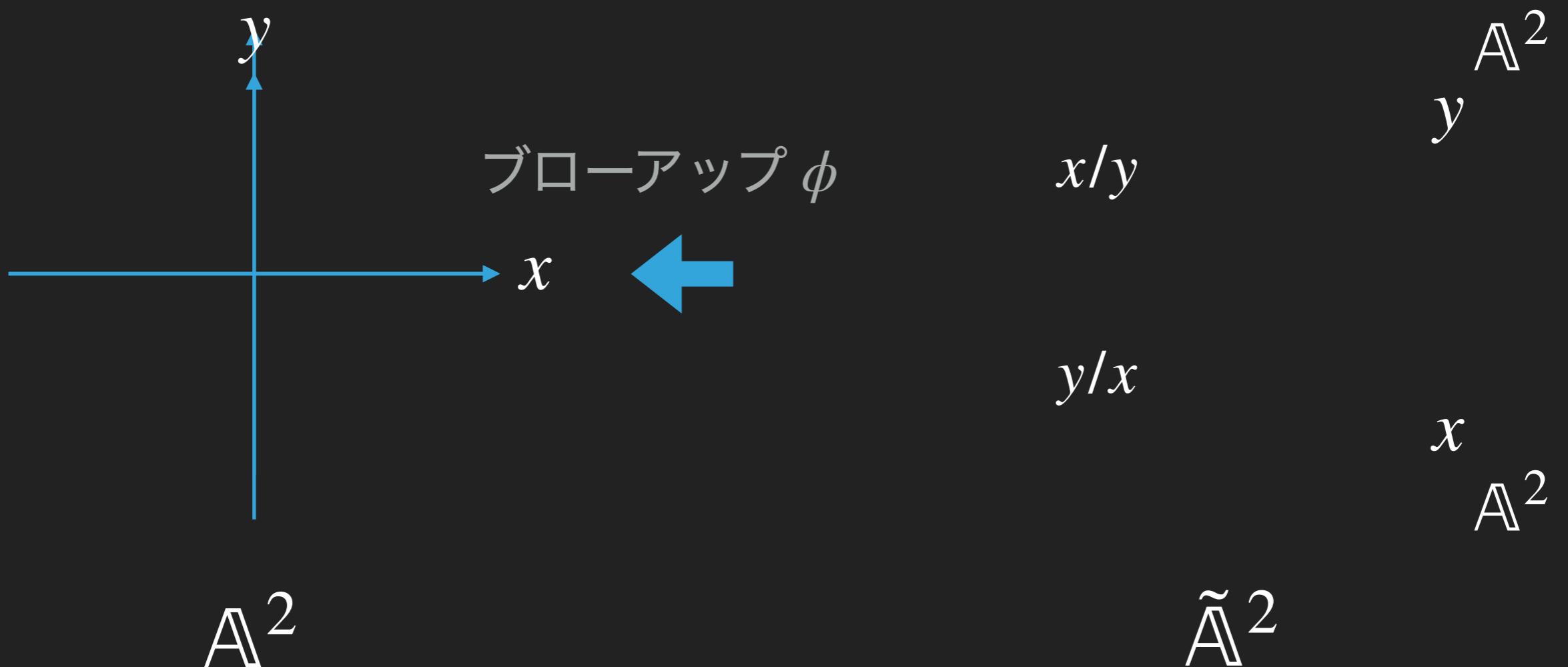
▶ class AffineOpen



\mathbb{A}^2

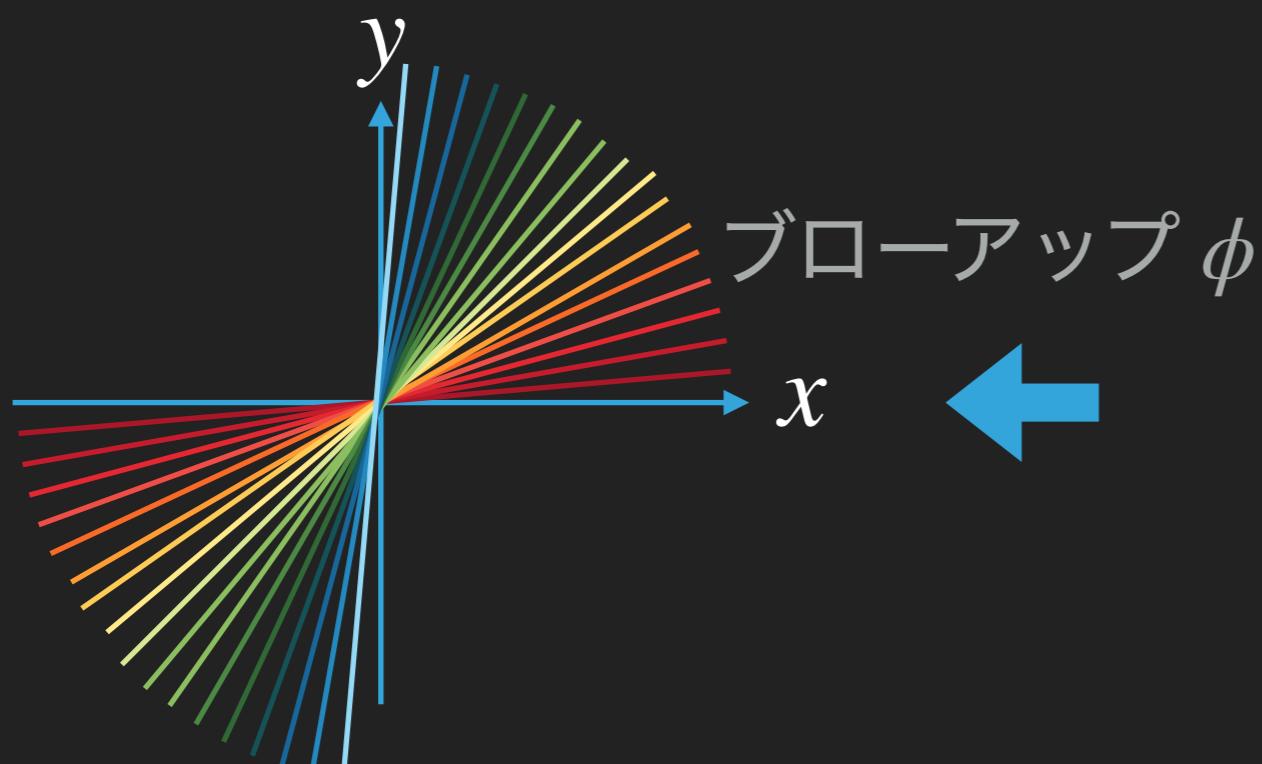
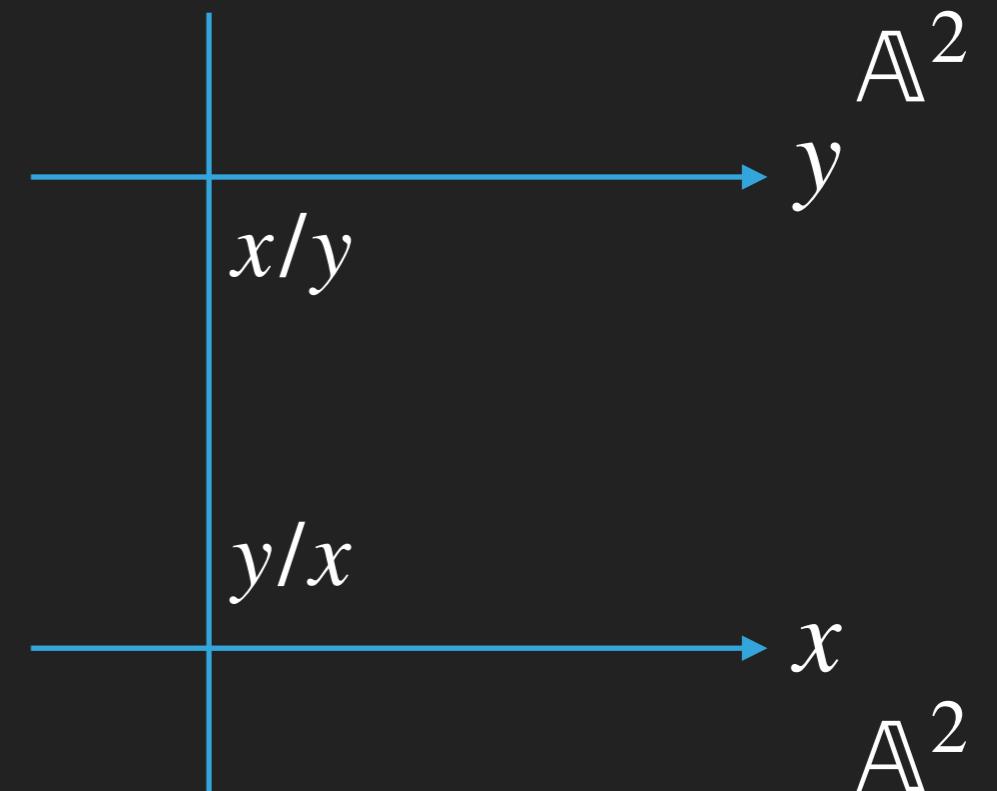
データ構造

▶ class AffineOpen



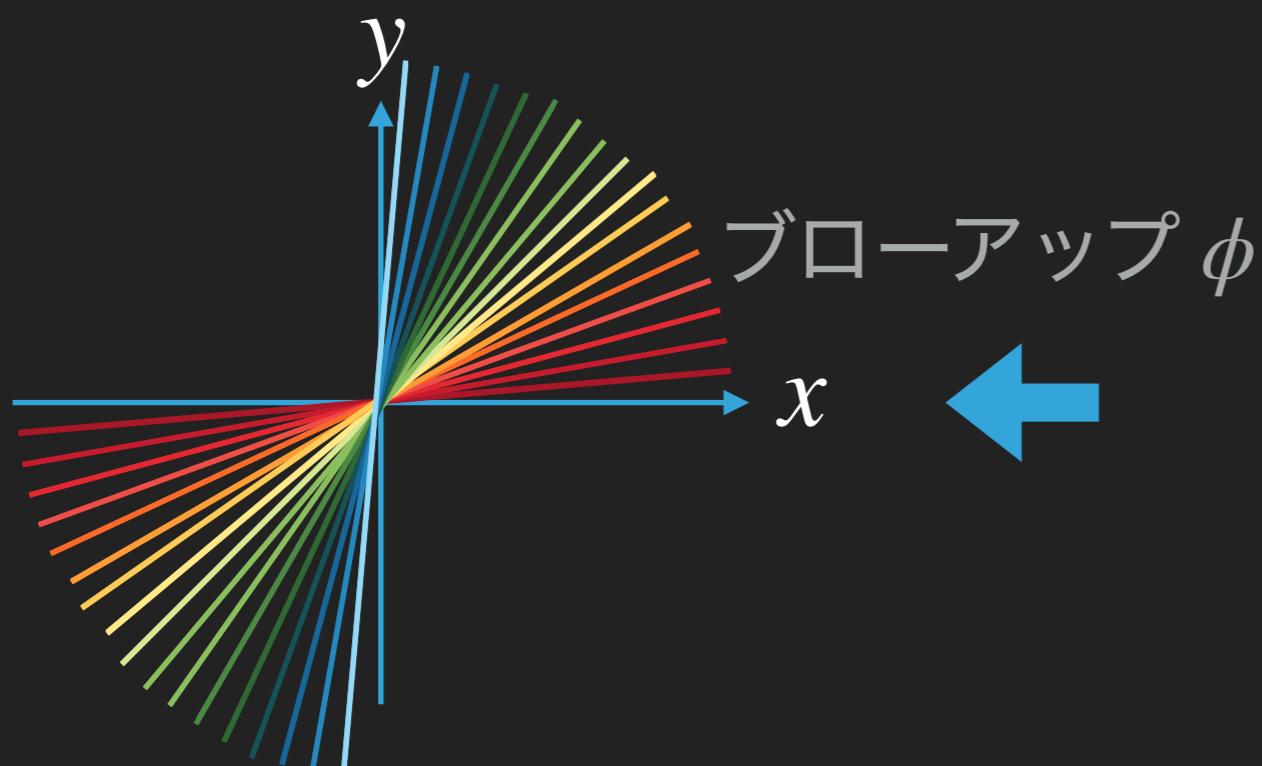
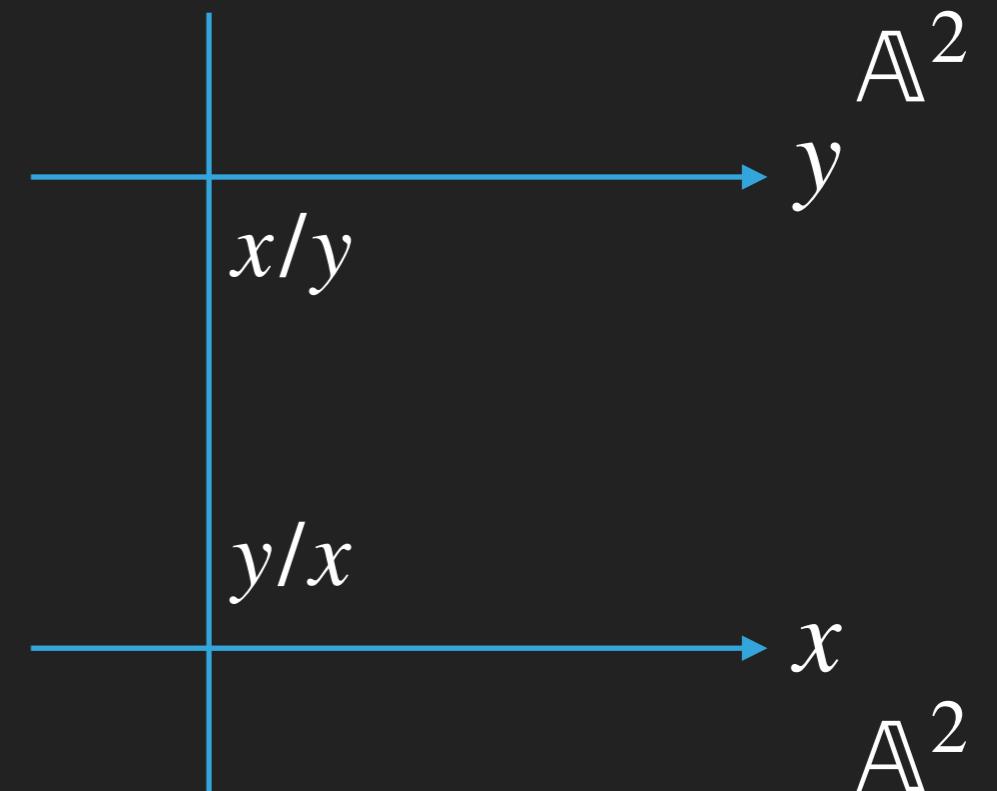
データ構造

▶ class AffineOpen

 \mathbb{A}^2  $\tilde{\mathbb{A}}^2$

データ構造

▶ class AffineOpen

 \mathbb{A}^2  $\tilde{\mathbb{A}}^2$

データ構造

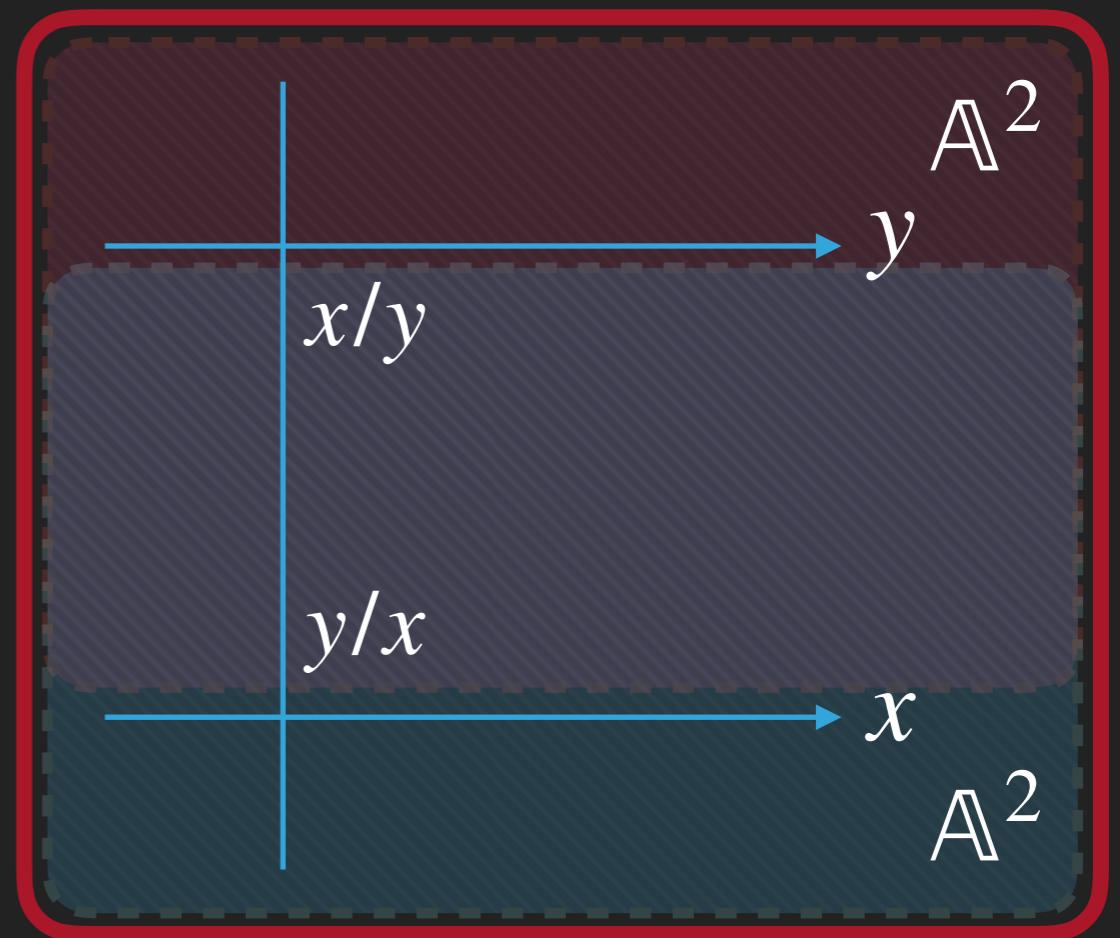
▶ class AffineOpen



ブローアップ ϕ



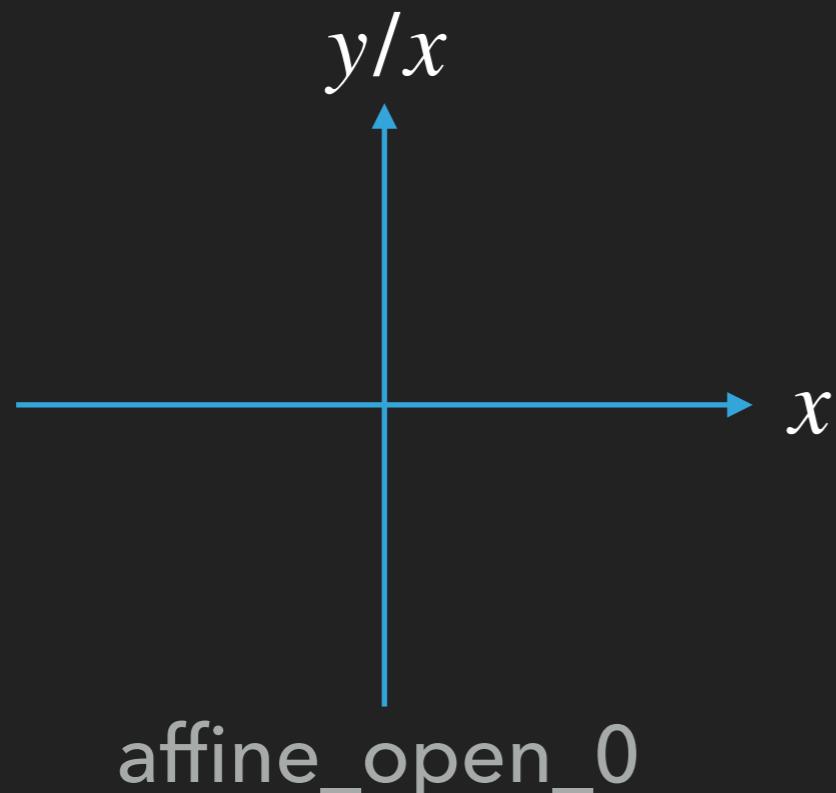
\mathbb{A}^2



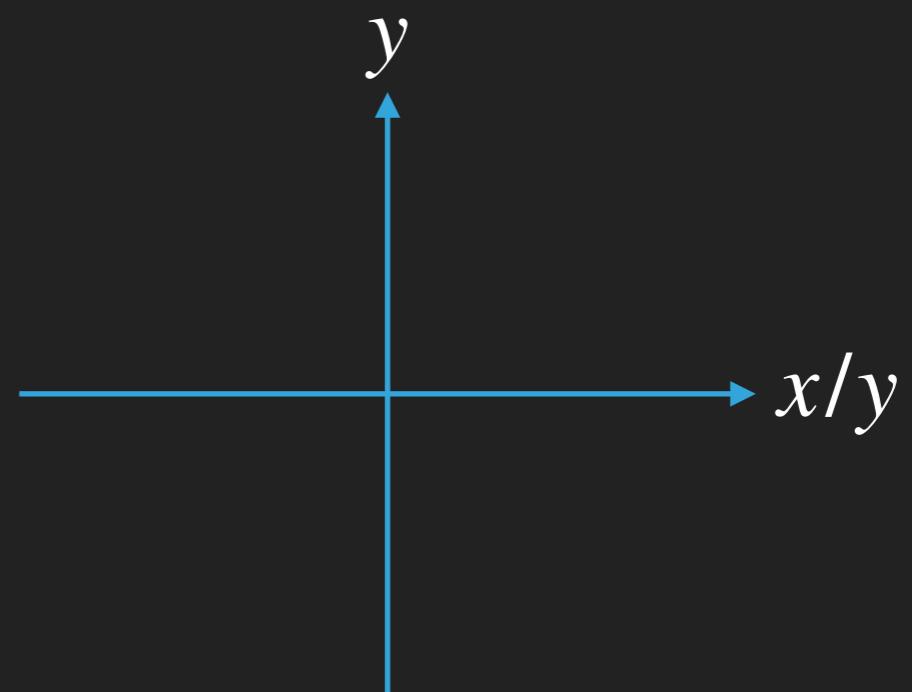
$\tilde{\mathbb{A}}^2$

データ構造

▶ class AffineOpen



affine_open_0

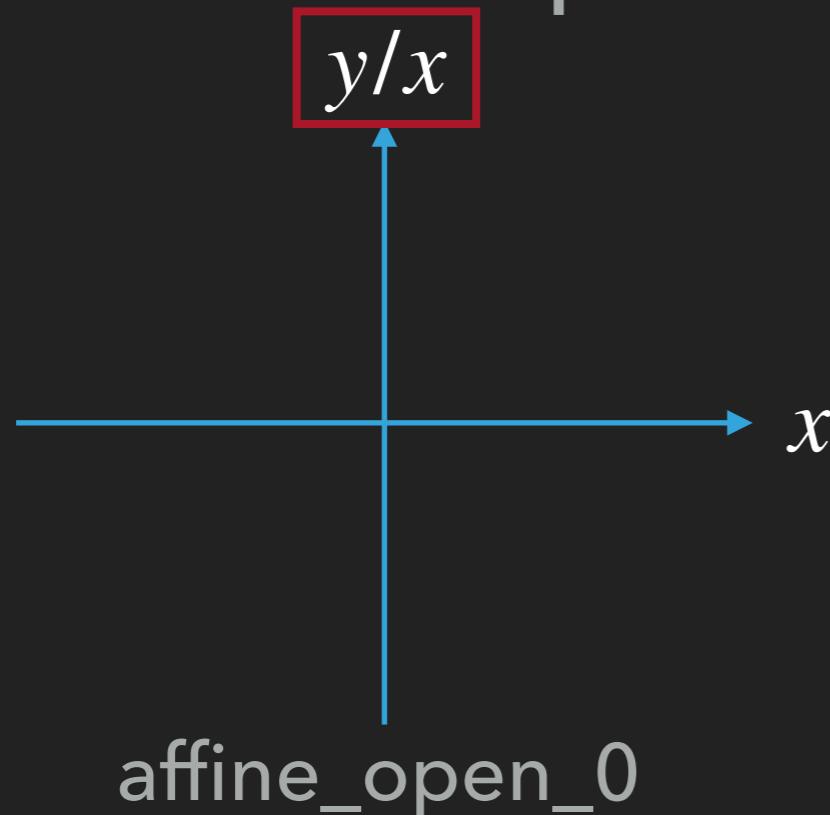


affine_open_1

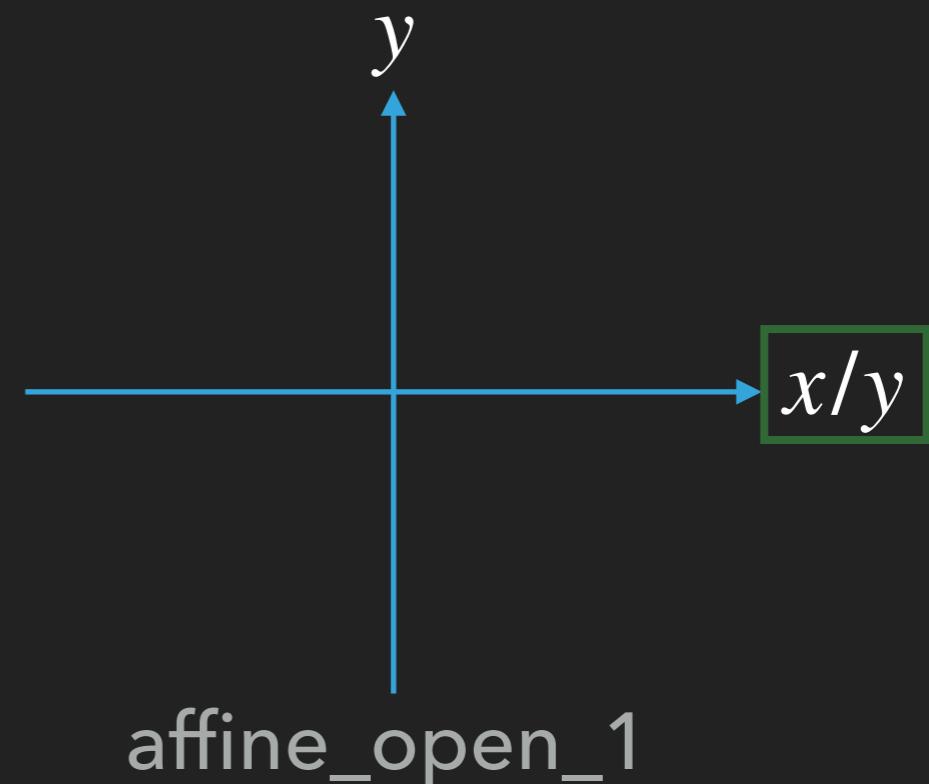
```
>>> affine_open_0.glue(1, affine_open_1, 0)
>>> affine_open_1.glue(0, affine_open_0, 1)
```

データ構造

▶ class AffineOpen



affine_open_0

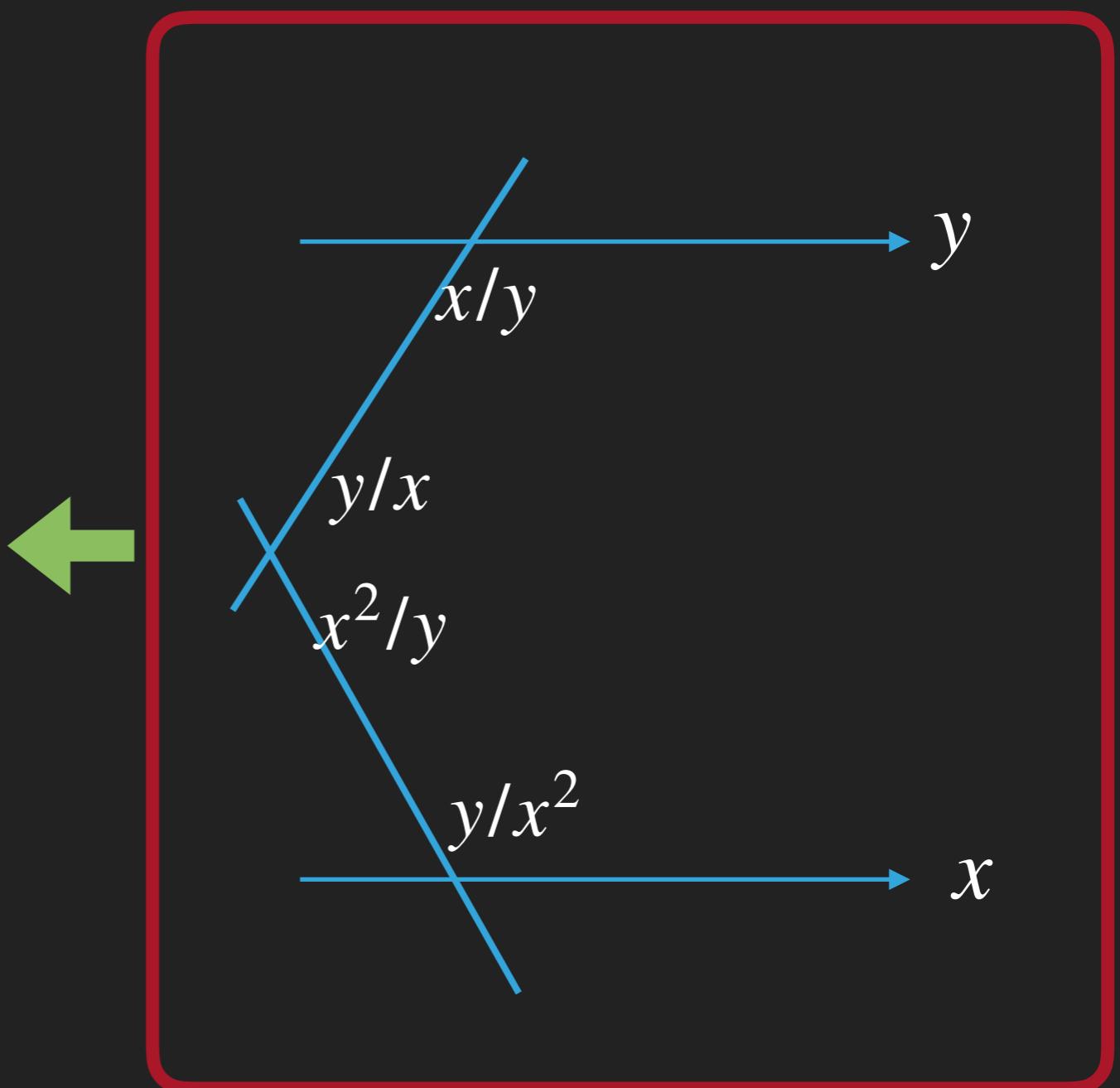
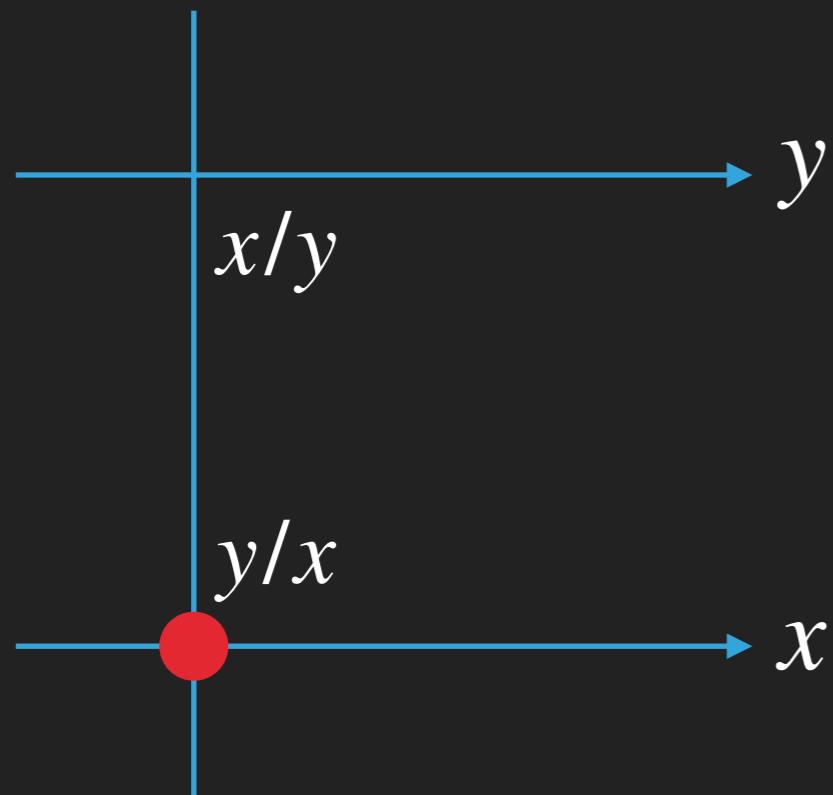


affine_open_1

```
affine_open_0.glued = [1: {'affine': affine_open_1, 'axis': 0}]  
affine_open_1.glued = [0: {'affine': affine_open_0, 'axis': 1}]
```

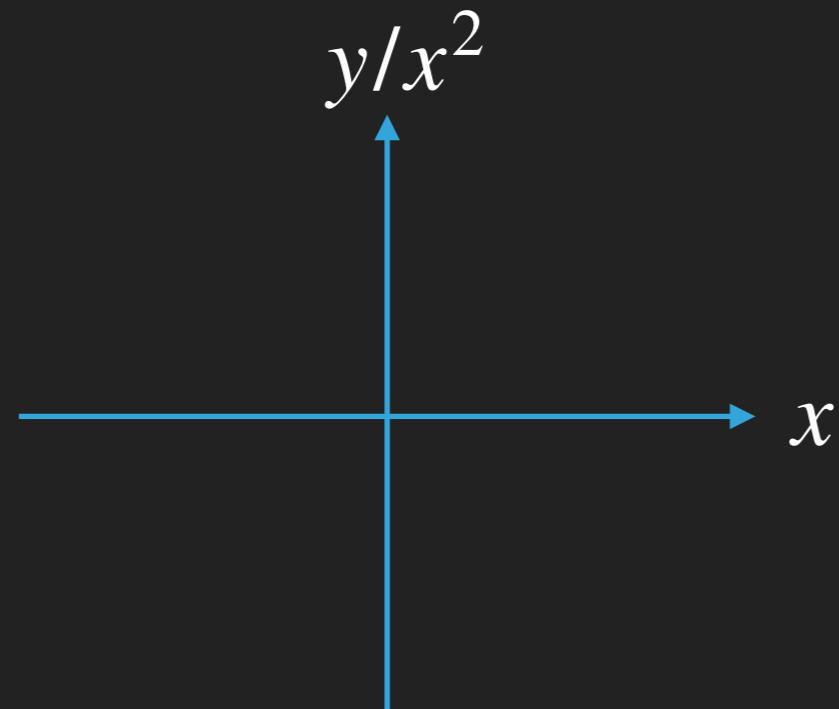
データ構造

▶ class AffineOpen

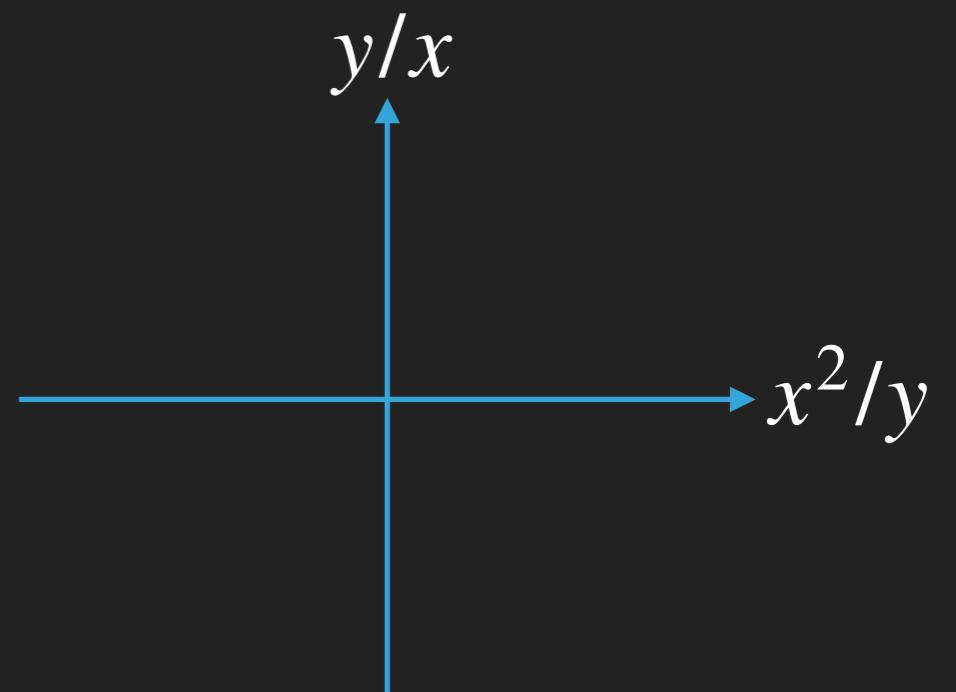


データ構造

▶ class AffineOpen



affine_open_2

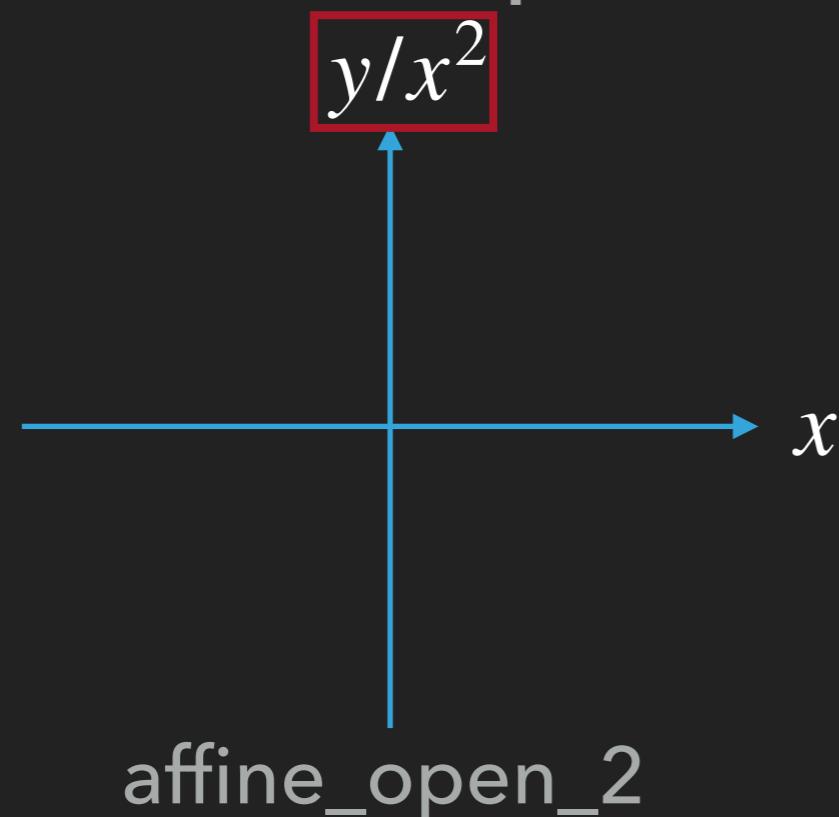


affine_open_3

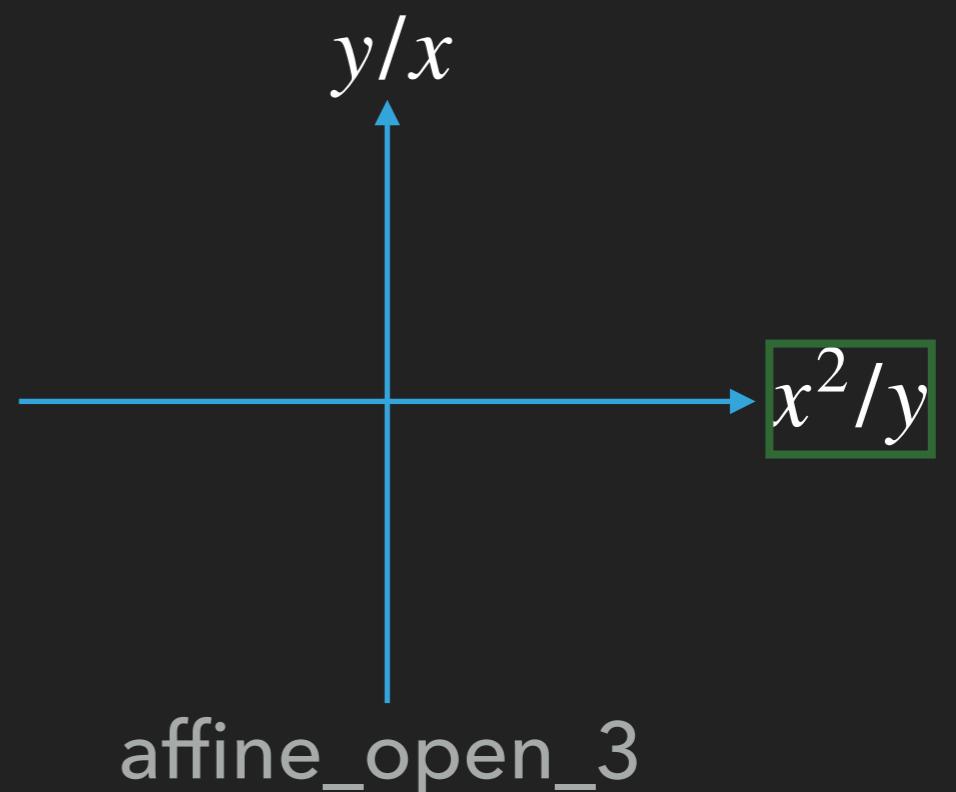
```
>>> affine_open_2.glue(1, affine_open_3, 0)
>>> affine_open_3.glue(0, affine_open_2, 1)
```

データ構造

▶ class AffineOpen



affine_open_2

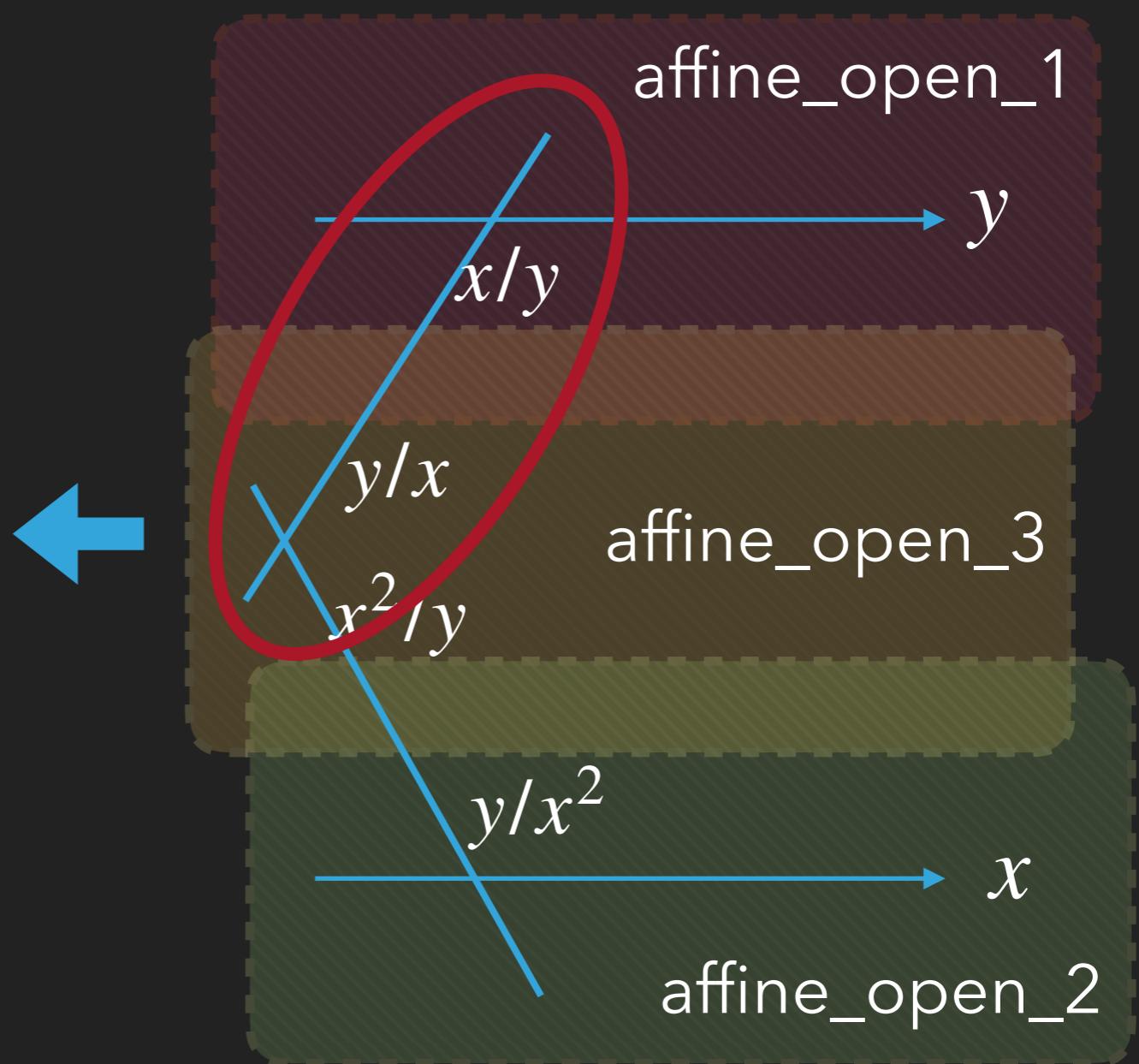
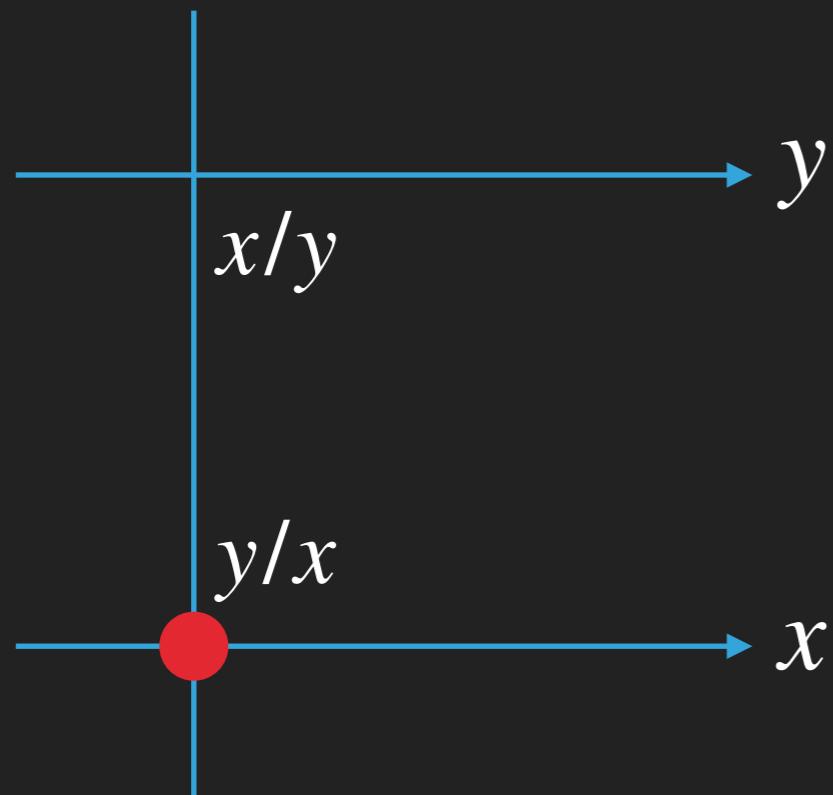


affine_open_3

```
affine_open_2.glued = [1: {'affine': affine_open_3, 'axis': 0}]  
affine_open_3.glued = [0: {'affine': affine_open_2, 'axis': 1}]
```

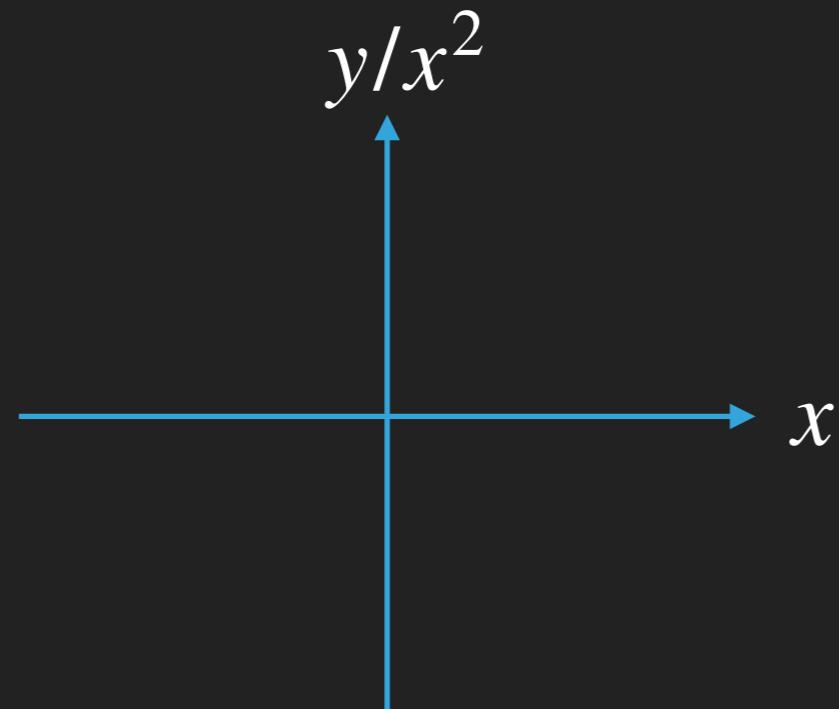
データ構造 (再掲)

▶ class AffineOpen



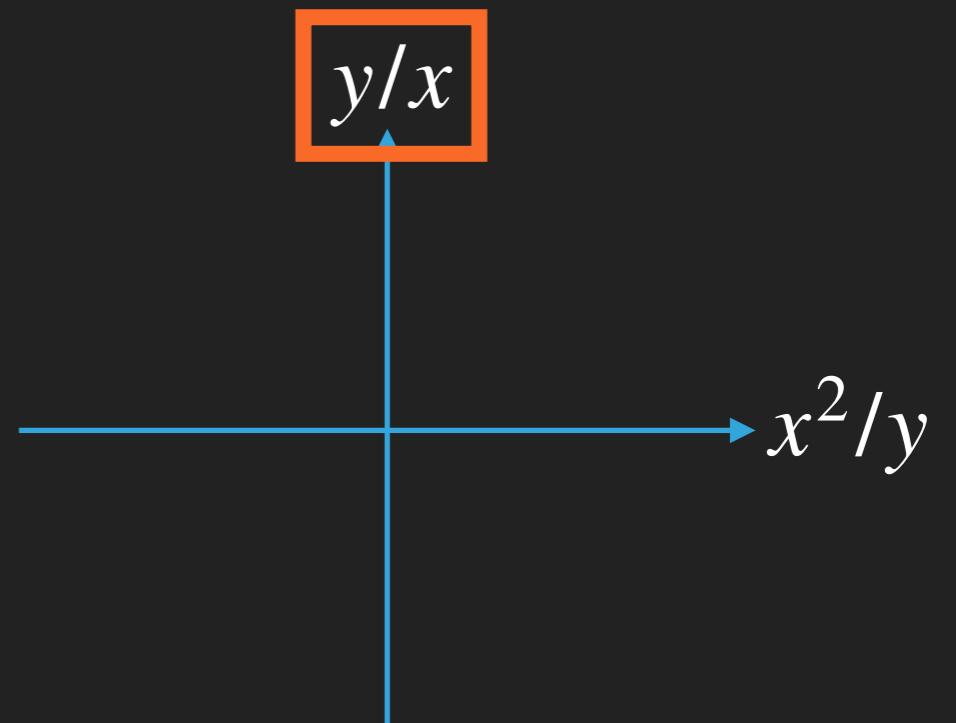
データ構造

▶ class AffineOpen



affine_open_2

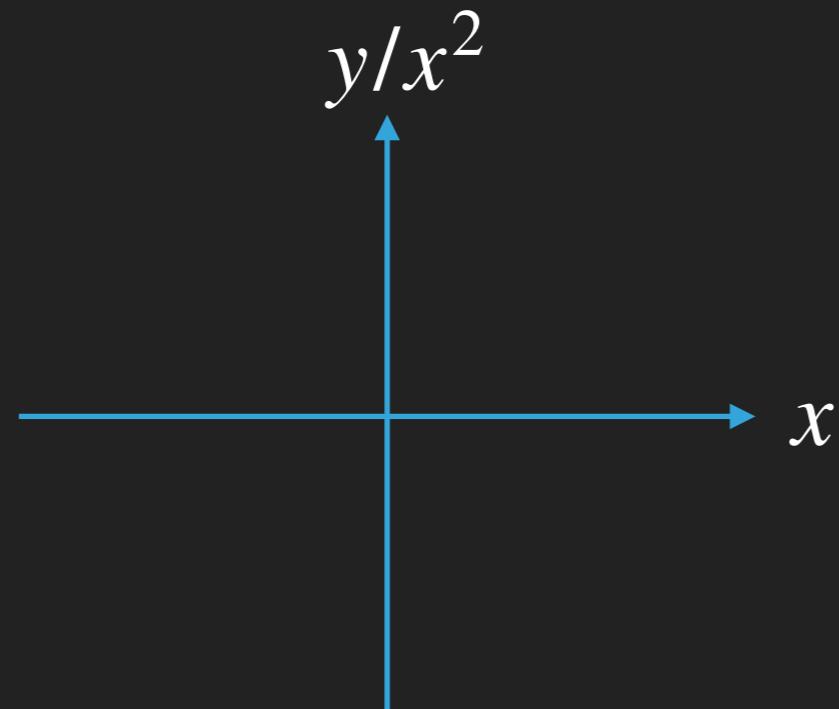
```
>>> affine_open_3.glue(1, affine_open_1, 0)
```



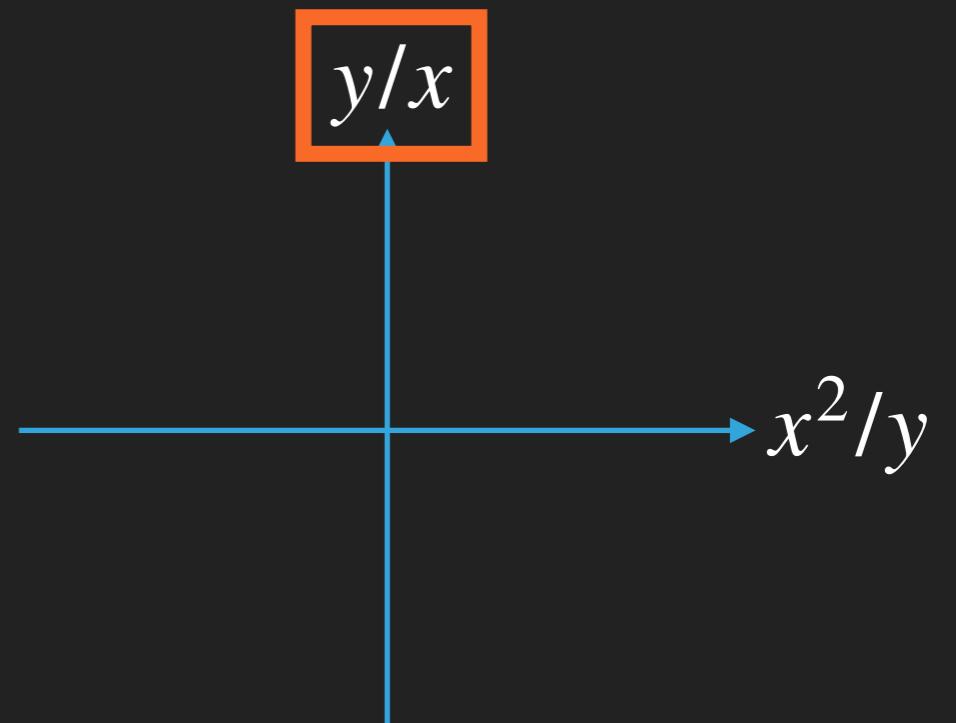
affine_open_3

データ構造

▶ class AffineOpen



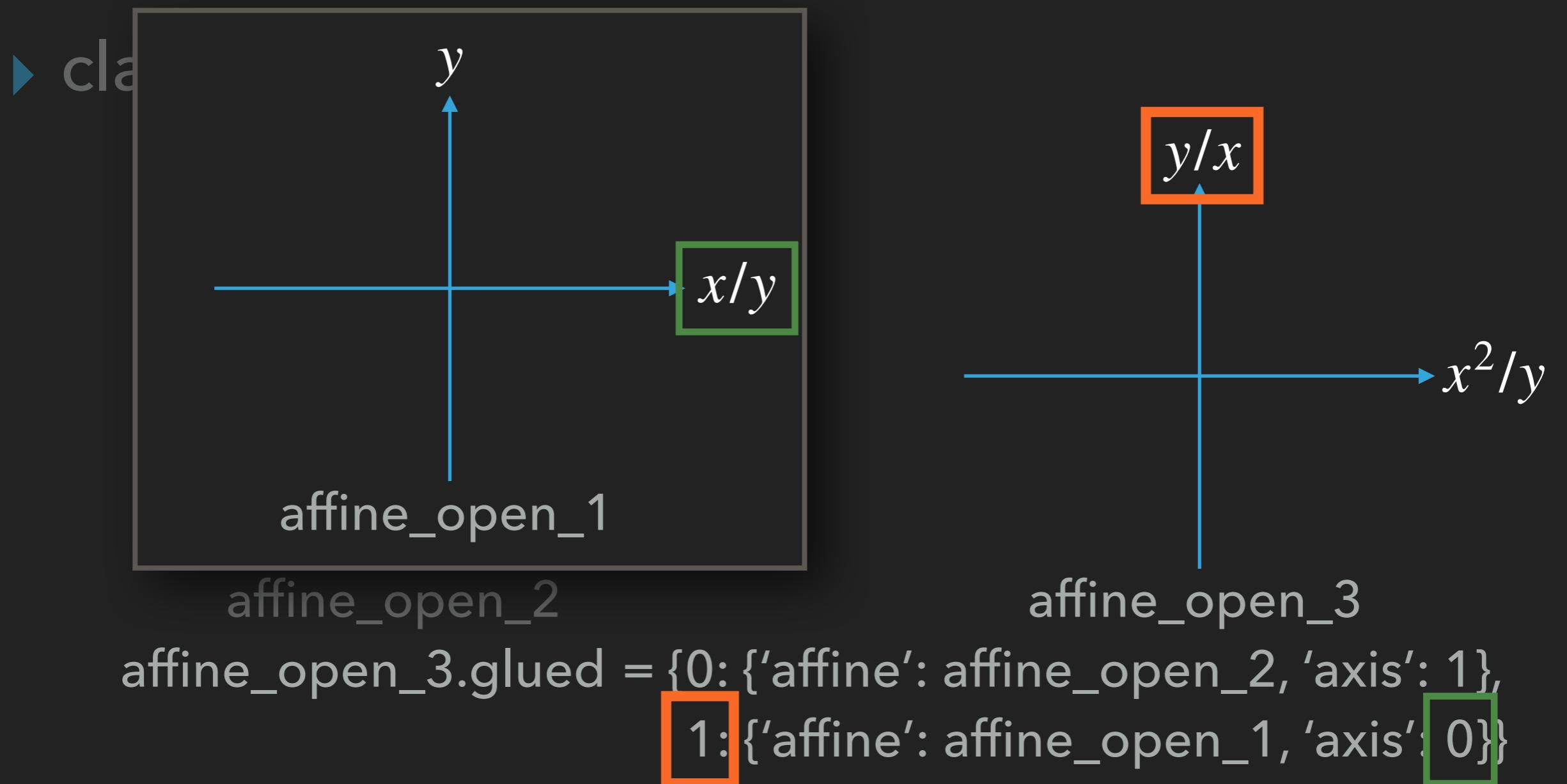
affine_open_2



affine_open_3

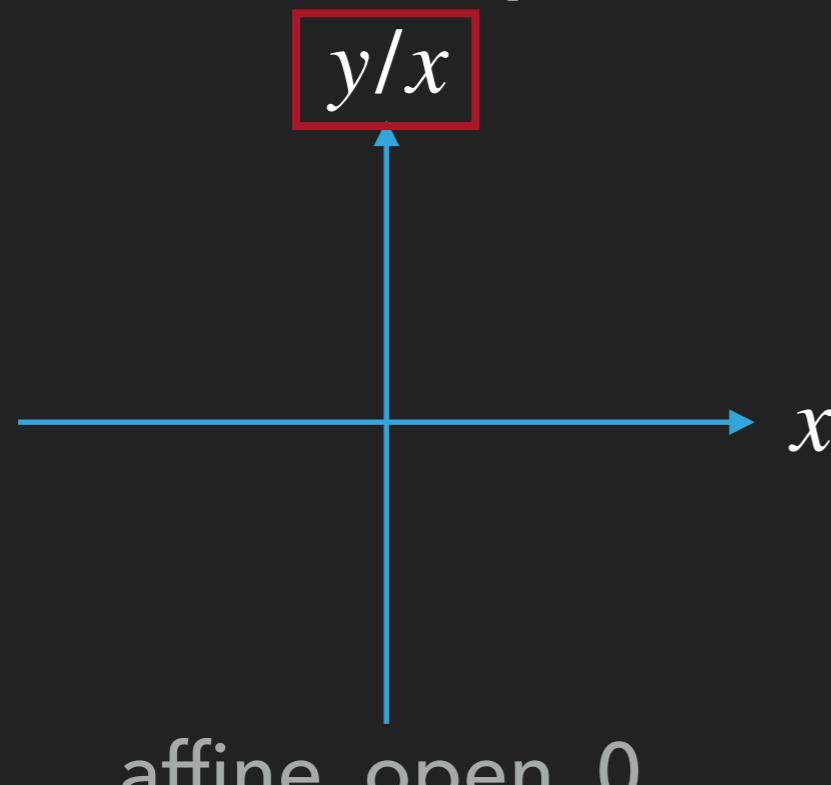
```
affine_open_3.glued = {0: {'affine': affine_open_2, 'axis': 1},  
                      1: {'affine': affine_open_1, 'axis': 0}}
```

データ構造



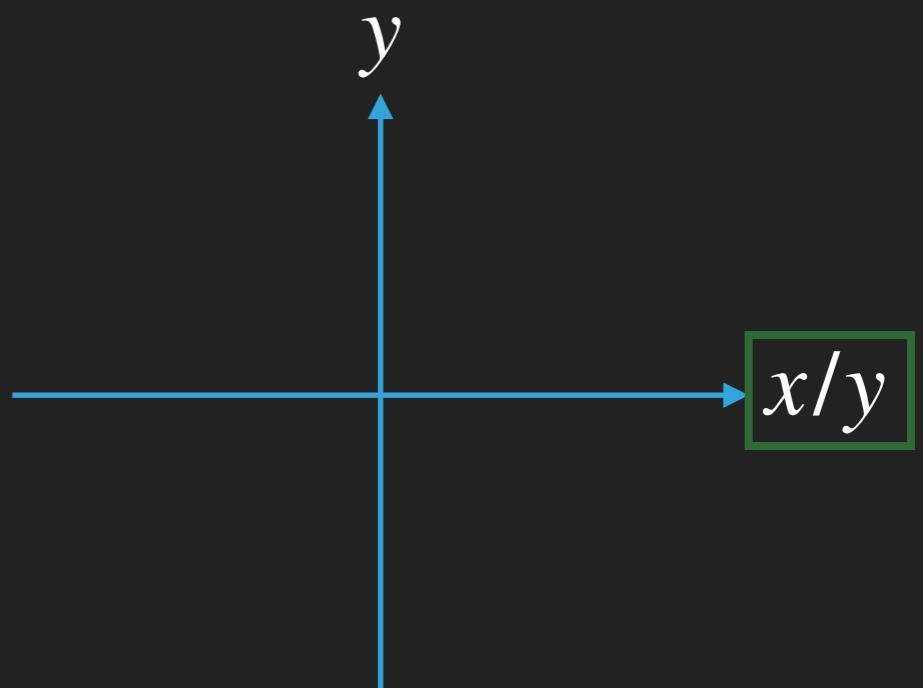
データ構造（再掲）

▶ class AffineOpen



affine_open_0

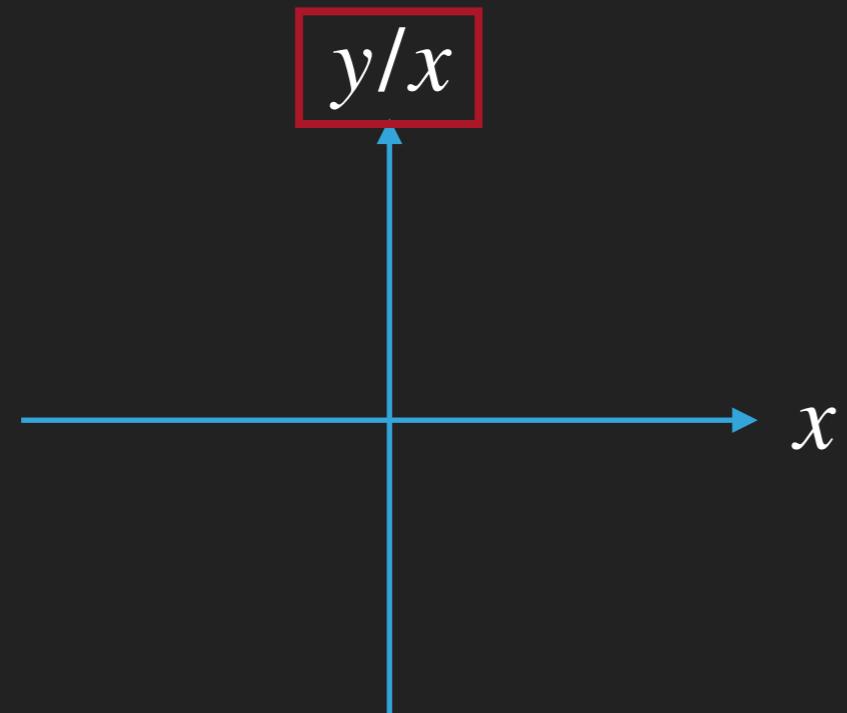
```
affine_open_0.glued = [1: {'affine': affine_open_1, 'axis': 0}]  
affine_open_1.glued = [0: {'affine': affine_open_0, 'axis': 1}]
```



affine_open_1

データ構造

▶ class AffineOpen



このAffineOpenの
情報は既に不要になっている。

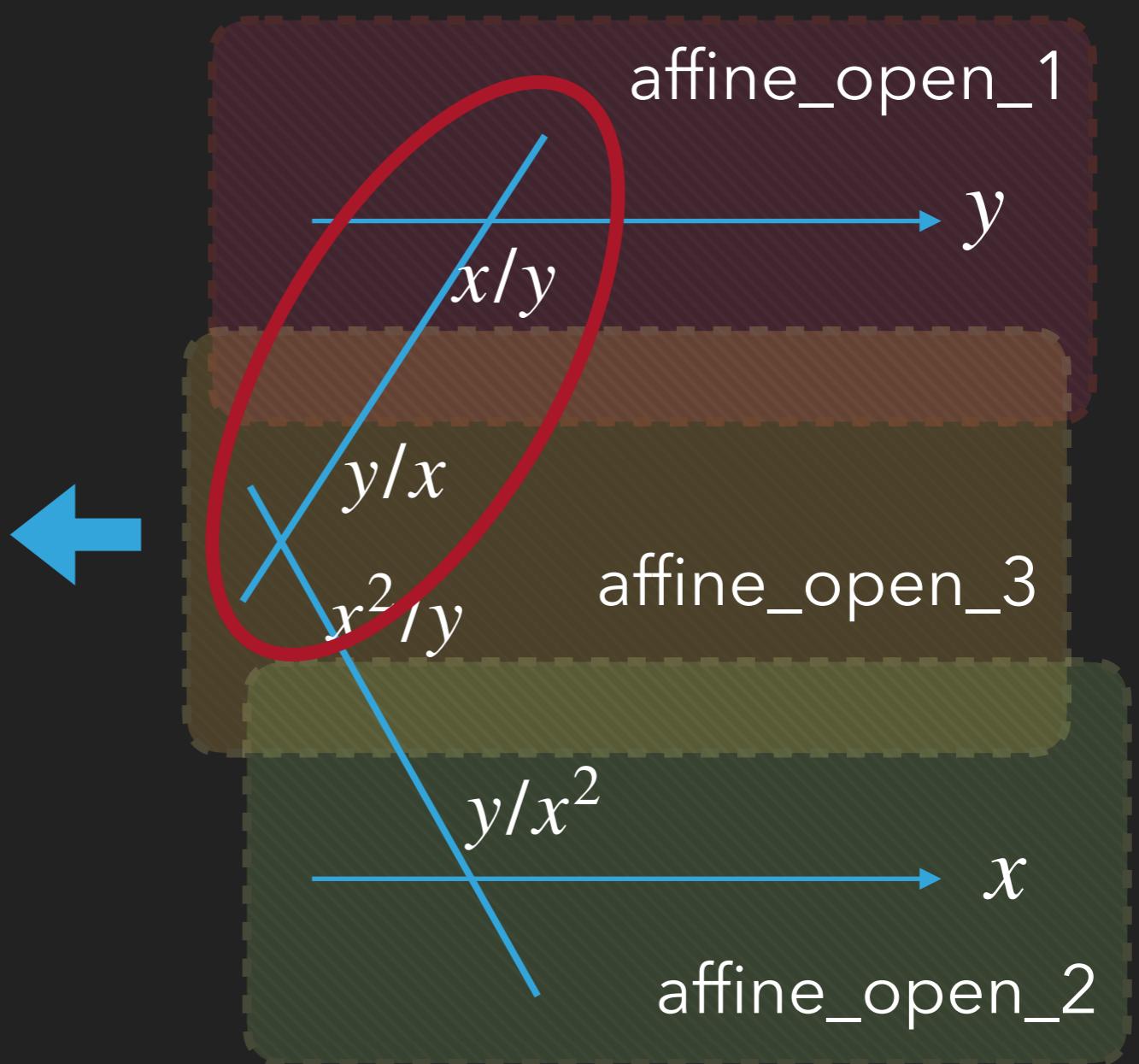
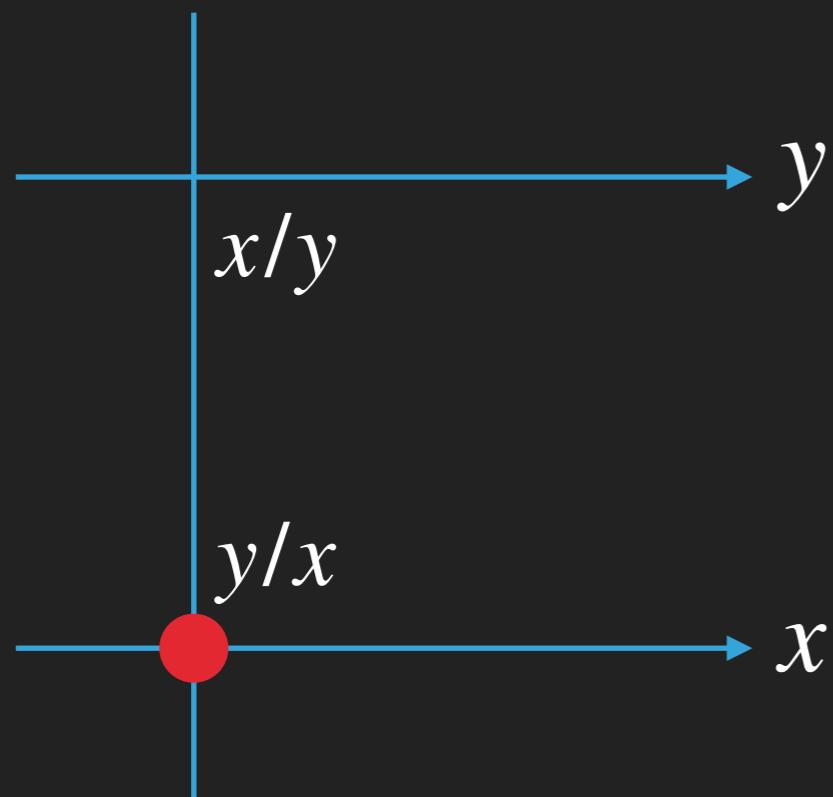
affine_open_0

```
affine_open_0.glued = [1: {'affine': affine_open_1, 'axis': 0}]  
affine_open_1.glued = [0: {'affine': affine_open_0, 'axis': 1}]
```

affine_open_1

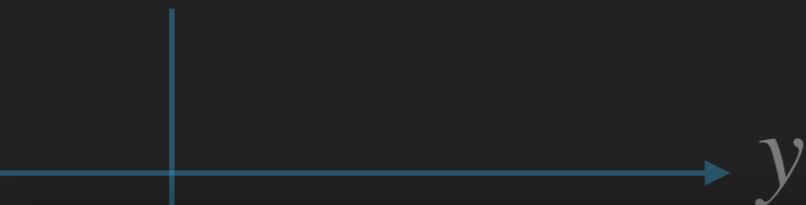
データ構造 (再掲)

▶ class AffineOpen

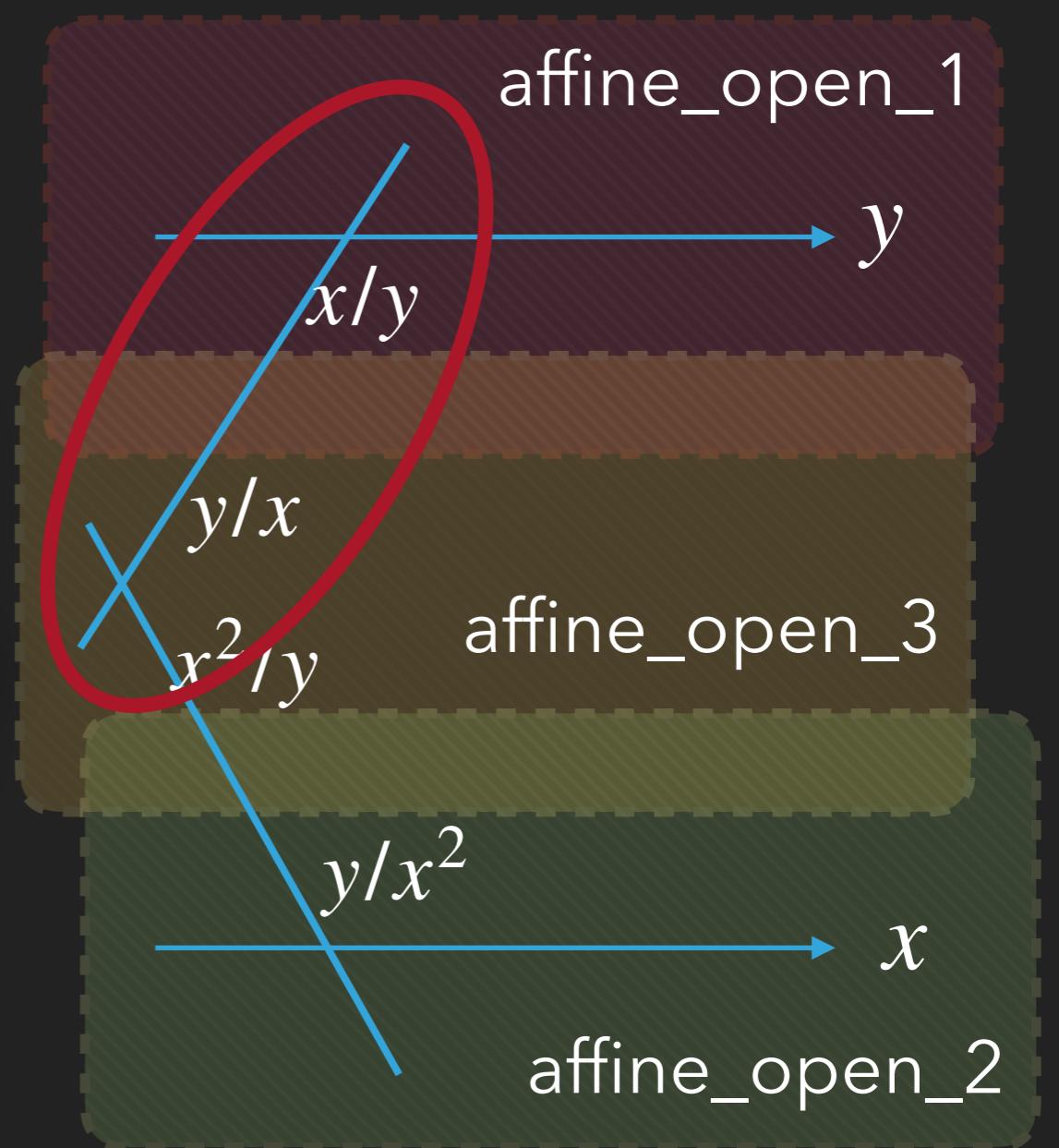
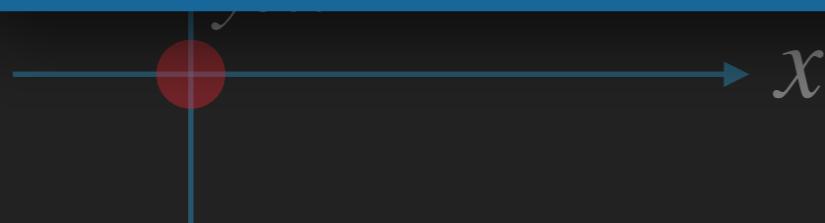


データ構造 (再掲)

▶ class AffineOpen

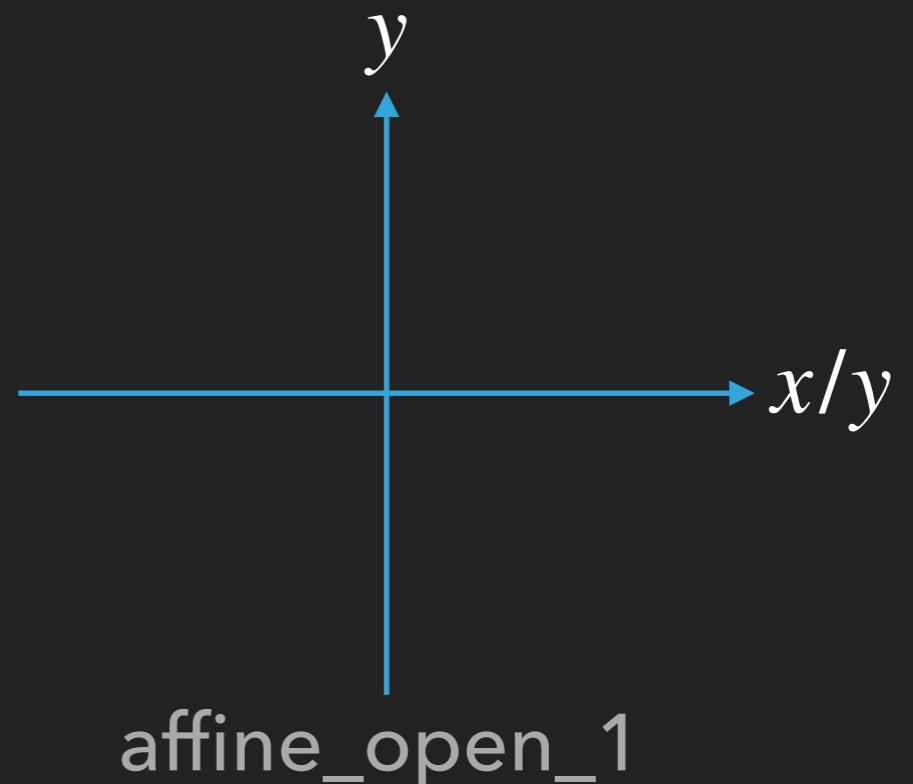
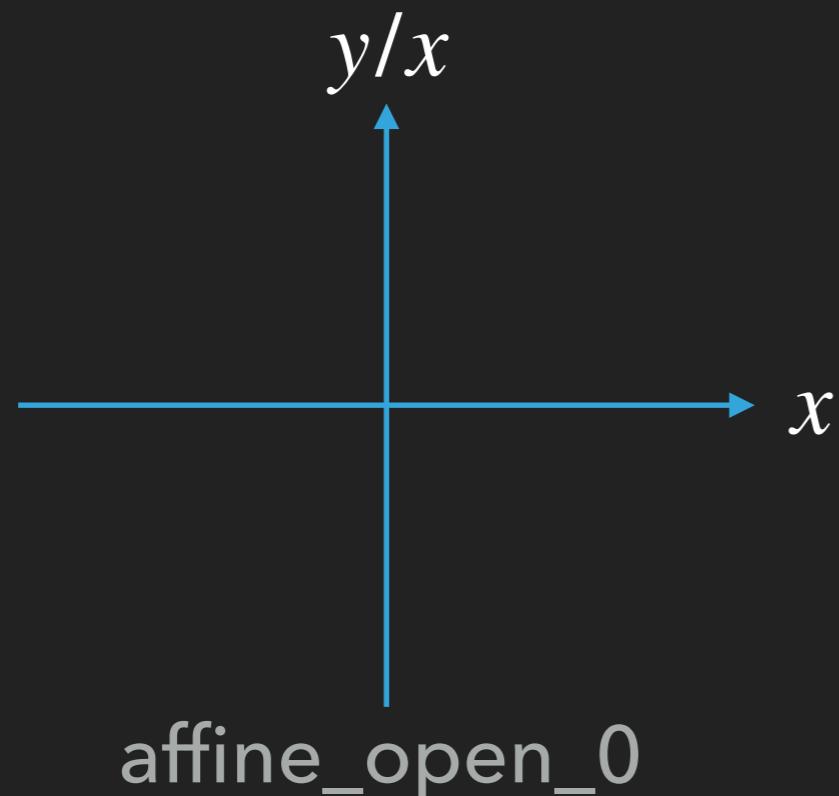


模式図にaffine_open_0が
出現していない。



データ構造

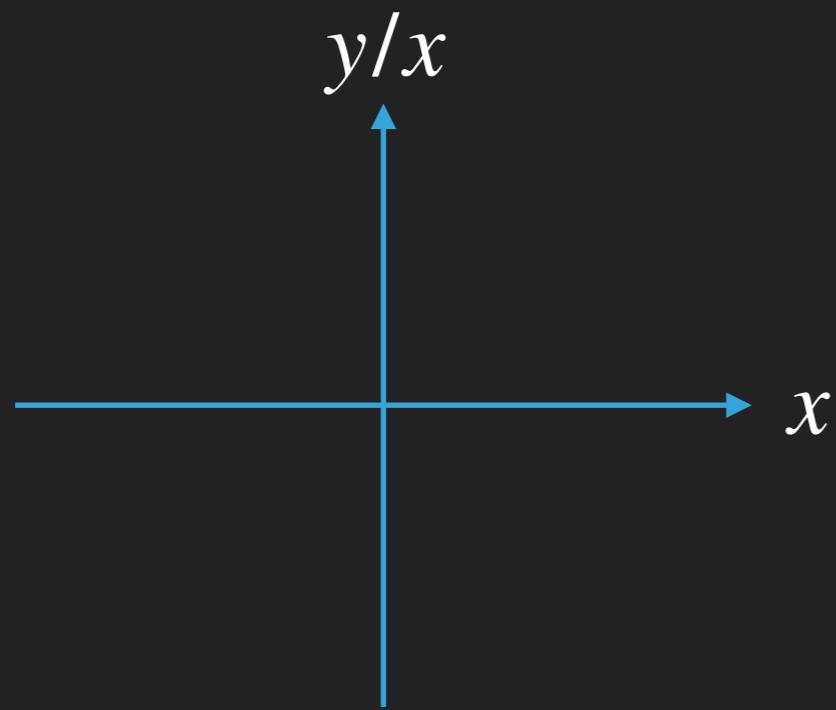
▶ class AffineOpen



```
>>> affine_open_0.detach(1)  
>>> affine_open_1.detach(0)
```

データ構造

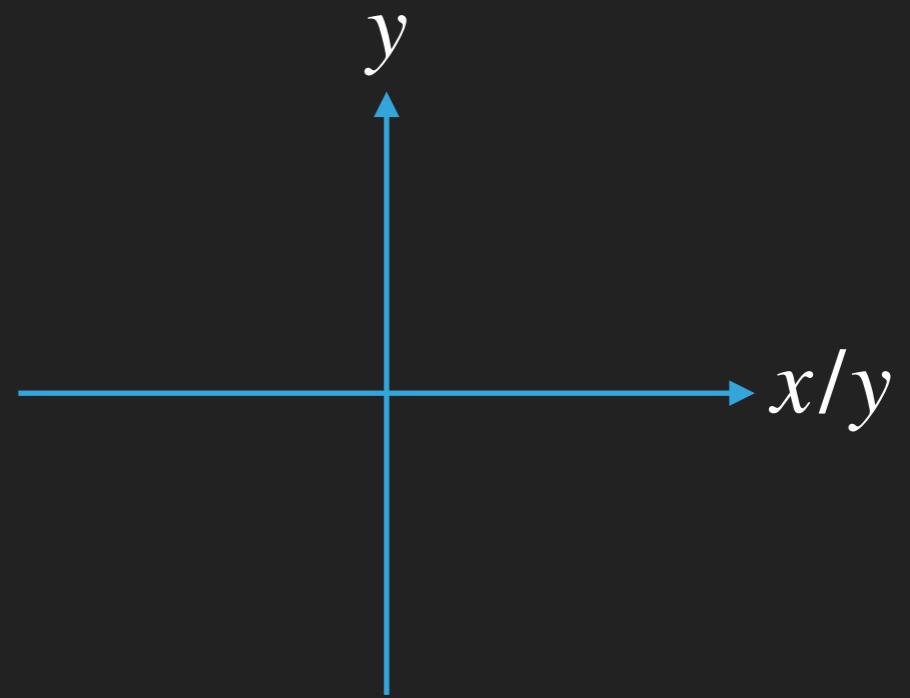
▶ class AffineOpen



affine_open_0

affine_open_0.glued = {}

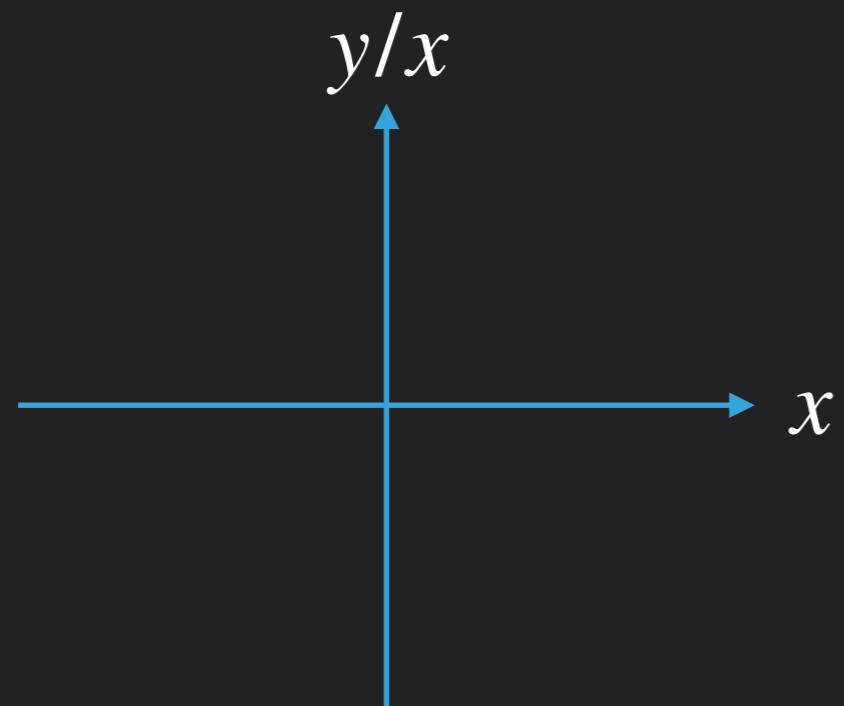
affine_open_1.glued = {}



affine_open_1

データ構造

▶ class AffineOpen



```
affine_open_0  
affine_open_0.glued = {}  
affine_open_1.glued = {}
```

模式図がコンピュータ上で
正しく構成される。

貼り合わせの情報が消失
= 貼り合わせが解除される。

データ構造

- ▶ class ExceptionalCurve

例外曲線を表すクラス

- ▶ divisors

例外曲線が存在するアフィン開集合と、そのアフィン開集合における例外曲線の方程式の組(辞書)を保持するリスト。

データ構造

▶ class ExceptionalCurve

例外曲線を表すクラス

▶ set()

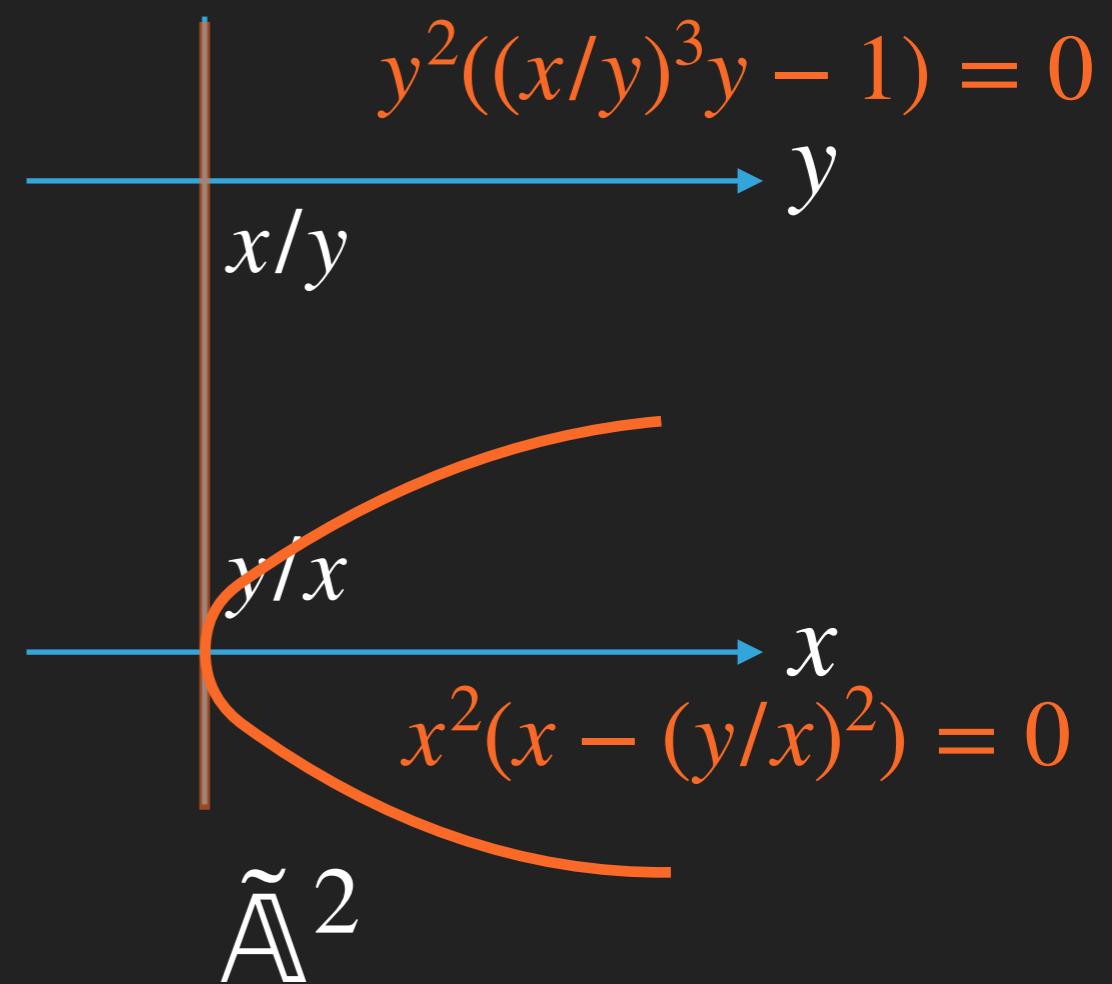
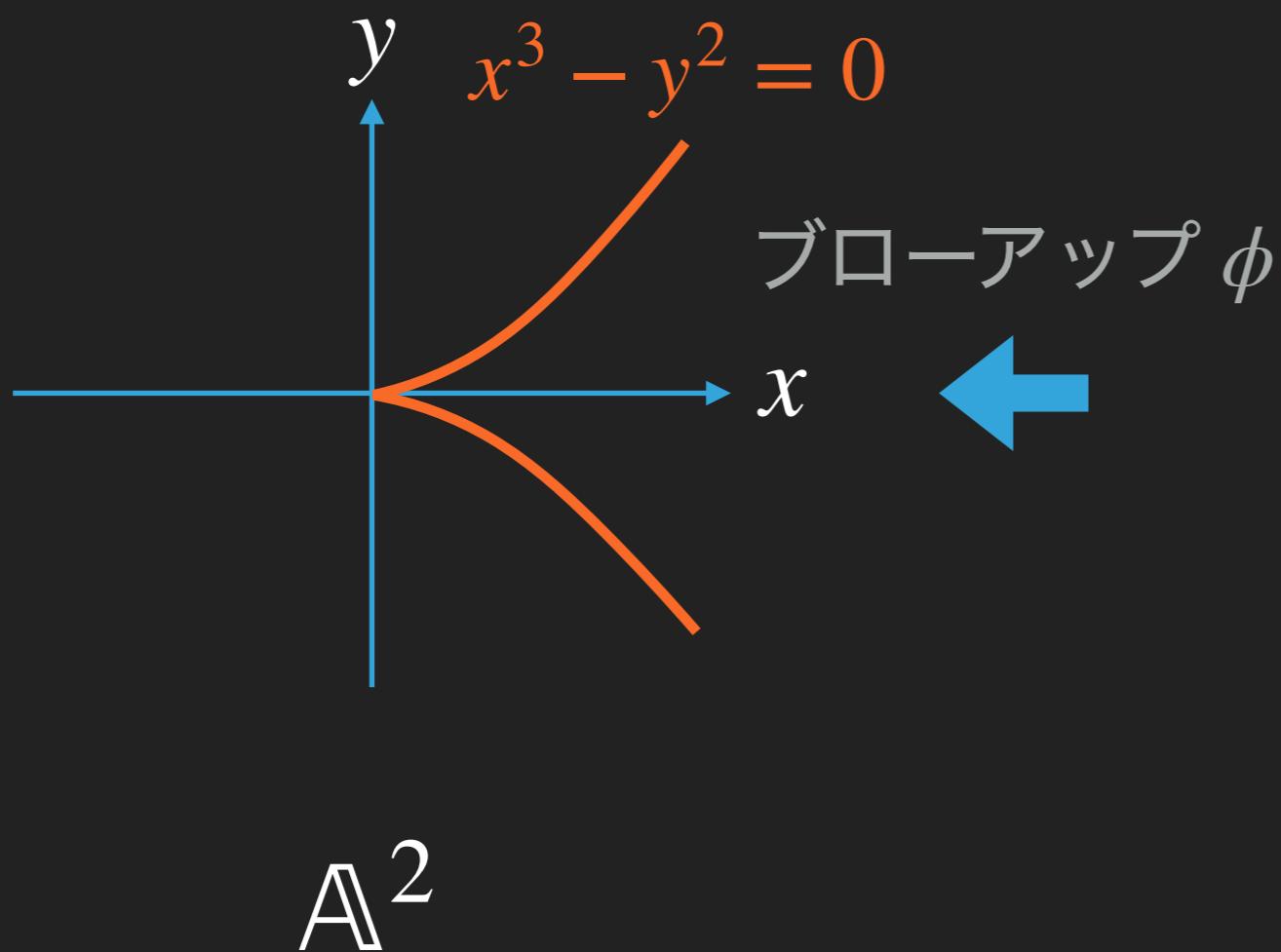
例外曲線の値（開集合と方程式）を定めるメソッド。

▶ intersection()

2つの例外曲線が交差する開集合が存在するかどうかを判定する
クラスメソッド。

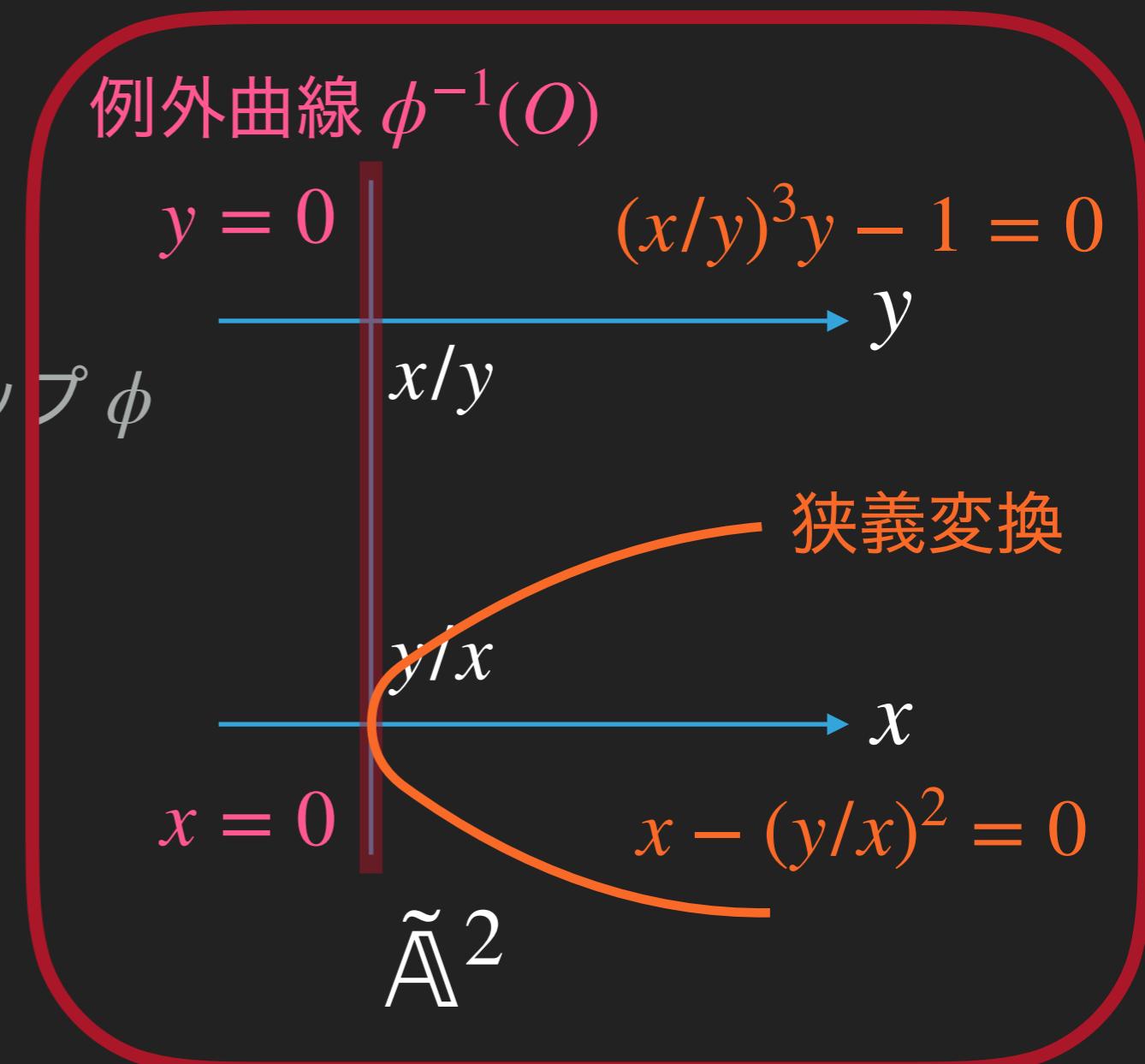
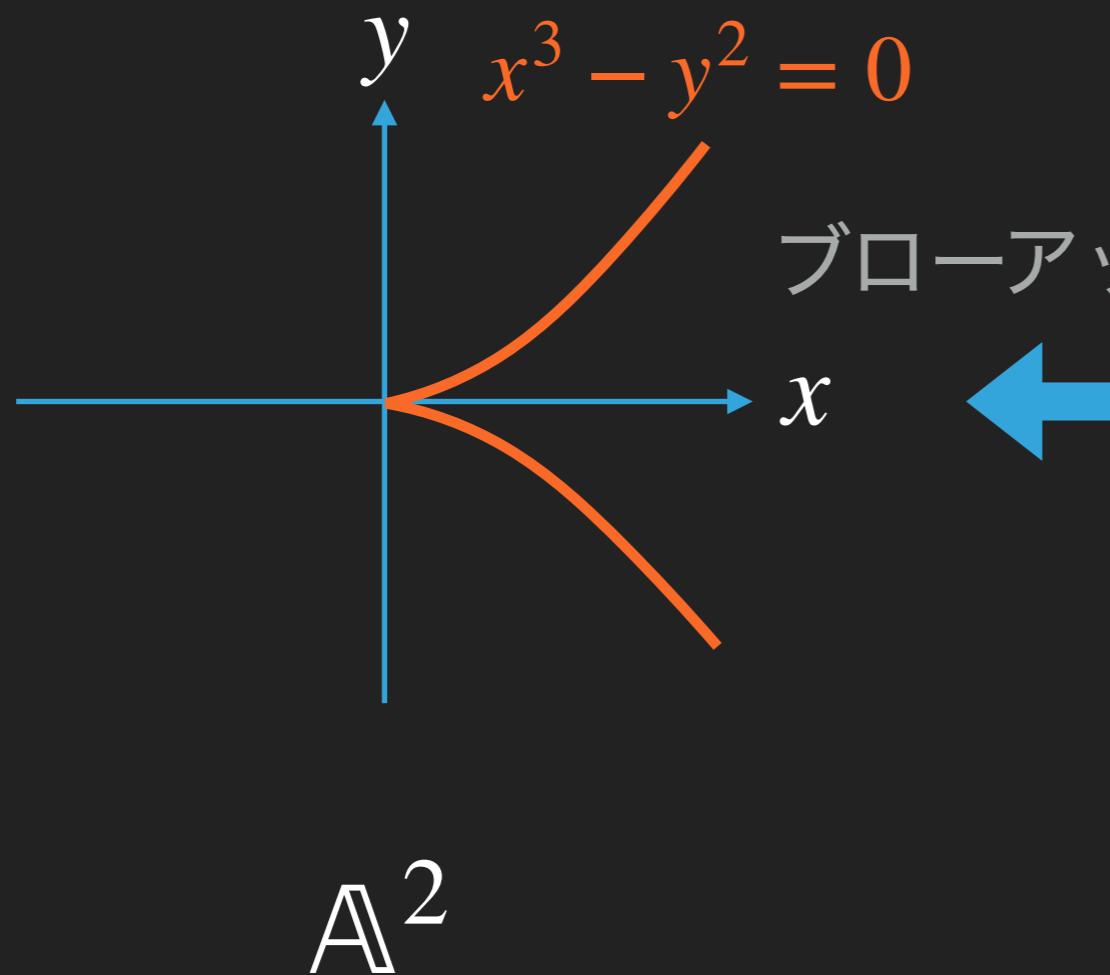
データ構造

▶ class ExceptionalCurve



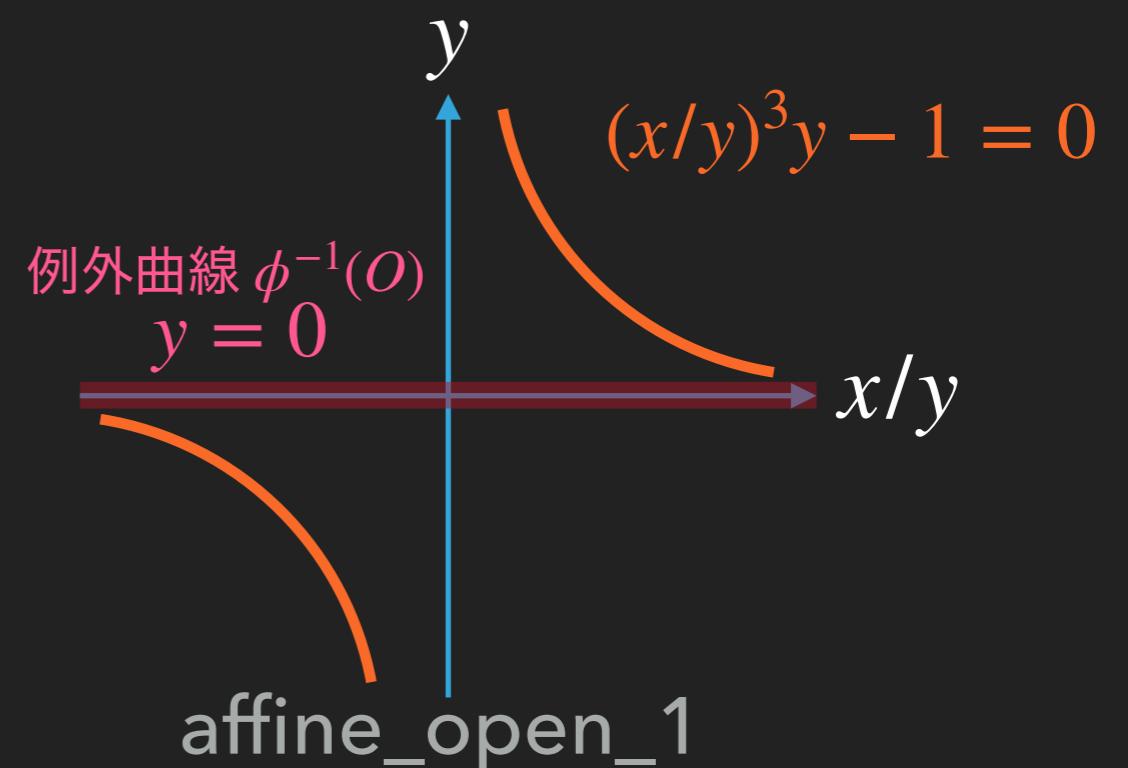
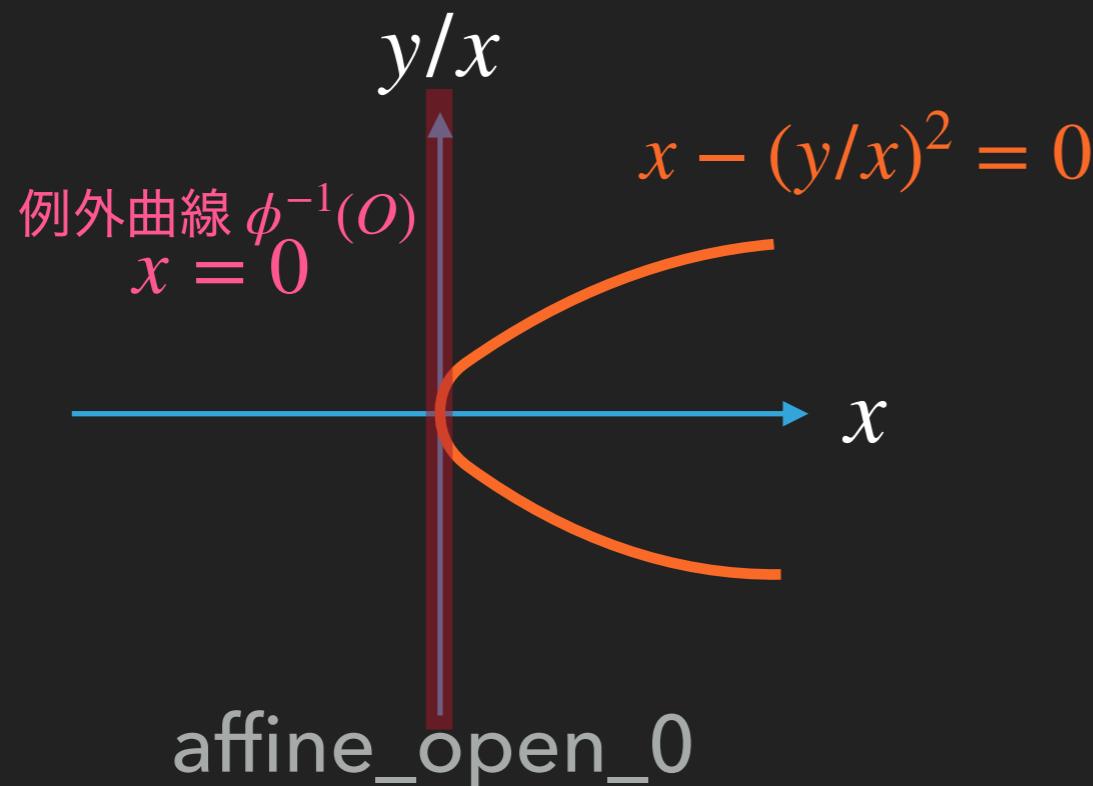
データ構造

▶ class ExceptionalCurve



データ構造

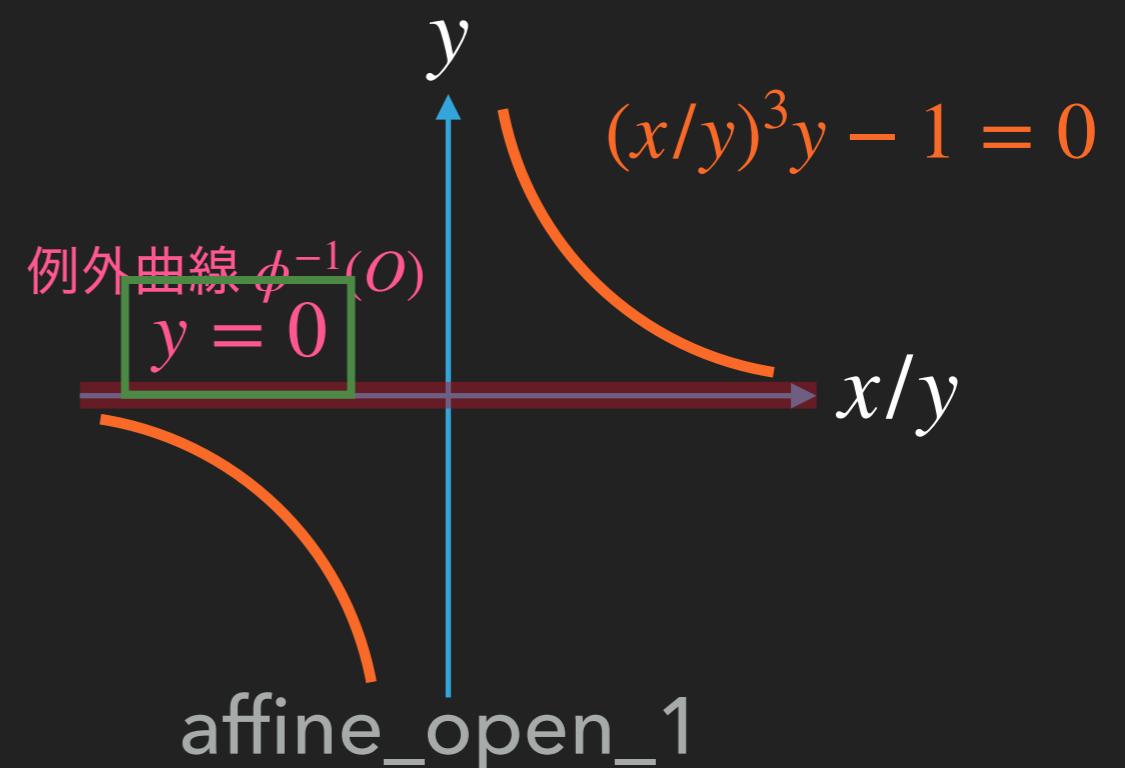
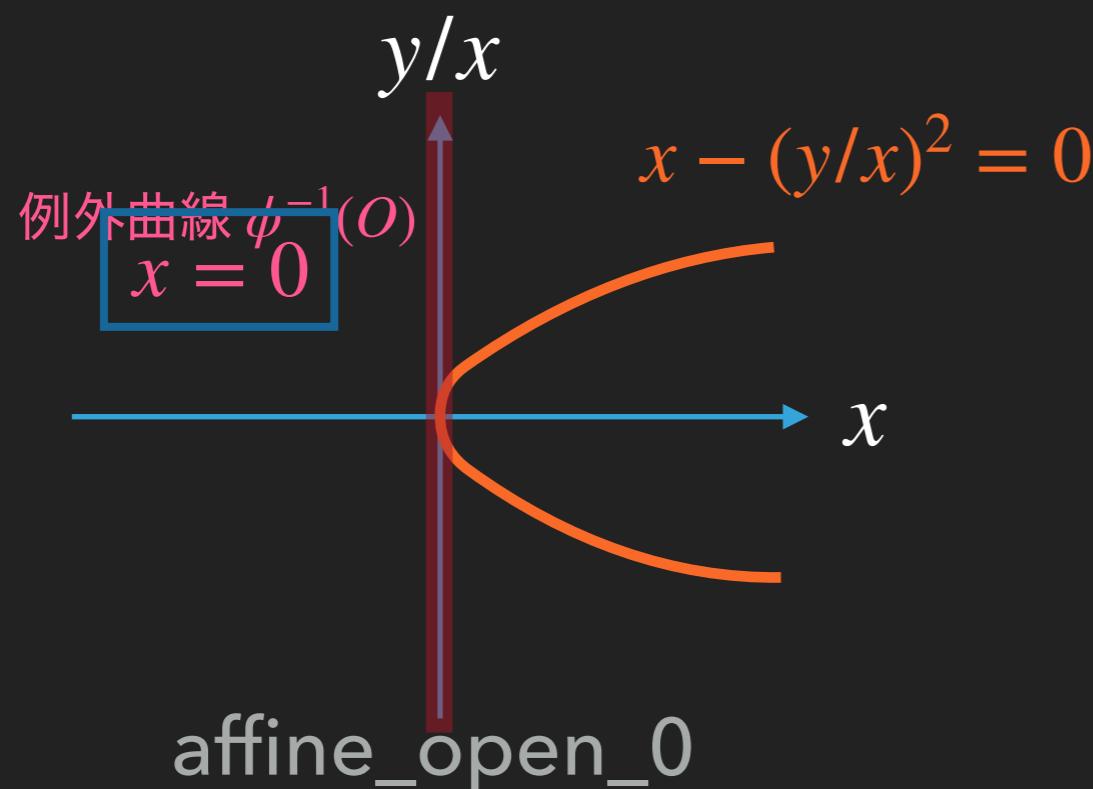
▶ class ExceptionalCurve



```
>>> exceptional_curve_1.set(affine_open_0, x)
>>> exceptional_curve_1.set(affine_open_1, y)
```

データ構造

▶ class ExceptionalCurve



```
exceptional_curve_1.divisors = [ {'open': affine_open[0], 'ideal': x},  
                                {'open': affine_open[1], 'ideal': y}]
```

データ構造

▶ class NonsingularStrictTransform

非特異狭義変換を表すクラス

(ExceptionalCurveクラスを継承している)

アルゴリズム

手計算

1. $\tilde{\mathbb{A}}^2$ のアフィン開集合を考える.
2. 各開集合上で曲線の狭義変換と例外曲線の方程式を求めて模式図を描く.
3. 特異性の判定
 1. 特異点があるとき
→ もう一度その点でブローアップ
 2. 特異点がないとき
 1. 例外曲線と狭義変換が横断的に交わるとき
→ 解消終了
 2. そうでないとき
→ 交点でもう一度ブローアップ
4. 模式図から双対グラフを求める.

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線の狭義変換と例外曲線の方程式を求める.
4. 特異性の判定（左と同じ）
5. `ExceptionalCurve`インスタンスを生成し、`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し、そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

アルゴリズム

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

```

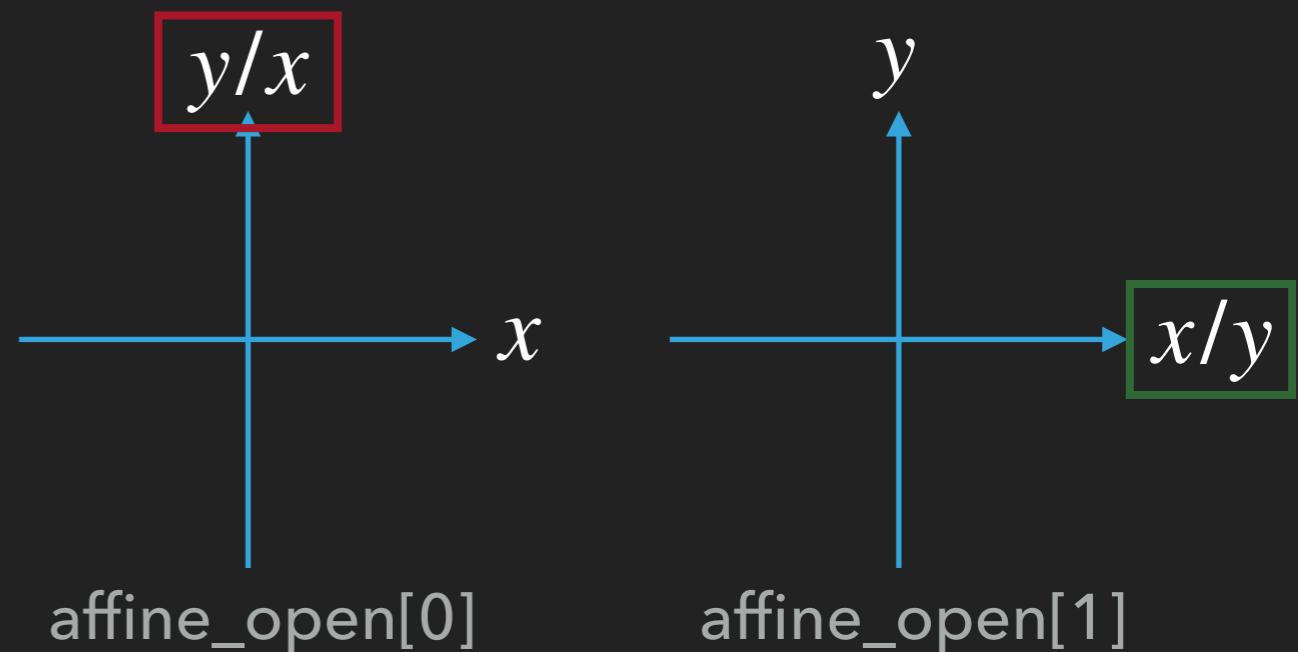
Sing V(f) = [{x: 0, y: 0}]
resolution of curve Singularities
019 by dafuyafu
on AA(x, y/x):
    f_x = x - y**2 : nonsingular
sympy Exc: V(x): not normal crossing
blow coordinate translation up as bu
    {x: 0, y: 0} --> {x: 0, y: 0}
name f_x = x - y**2 :
= symbols('x')
= symbols('y')
    f_x = -x*y**2 + 1 : nonsingular
    Exc: V(x): no crossing
f = x * y**2 / AA(x, y/x) # 19, 25-cusp
f = x * y**2 / AA(x**2/y, y/x) # 19, 25-cusp
f_y = x - y : nonsingular
ing = so Exc:[V(y):not, normal crossing]
print("f coordinate translation")
print("Sing V(f) = {y: 0, x: 0} --> {x: 0, y: 0}")
print("")
u.blowing_up(f):
    f_x = 1 - y : nonsingular
    Exc: V(x): normal crossing
    on AA(x**3/y**2, y/x):
        f_y = x - 1 : nonsingular
        Exc: V(y): normal crossing
    on AA(x/y, y):
        f_y = x**3*y - 1 : nonsingular
        Exc: V(y): no crossing
ExceptionalCurve({x, AA(x, y/x)}, {y, AA(x/y, y)}, {x, AA(x**3/y**2, y/x)})
ExceptionalCurve({x, AA(x, y/x**2)}, {y, AA(x**2/y, y/x)}, {y, AA(x**2/y, y/x)})
ExceptionalCurve({x, AA(x**2/y, y**2/x**3)}, {y, AA(x**3/y**2, y/x)})
NonsingularStrictTransform({1 - y, AA(x**2/y, y**2/x**3)}, {x - 1, AA(x**3/y**2, y/x)})

```

アルゴリズム 1-1

コンピュータアルゴリズム

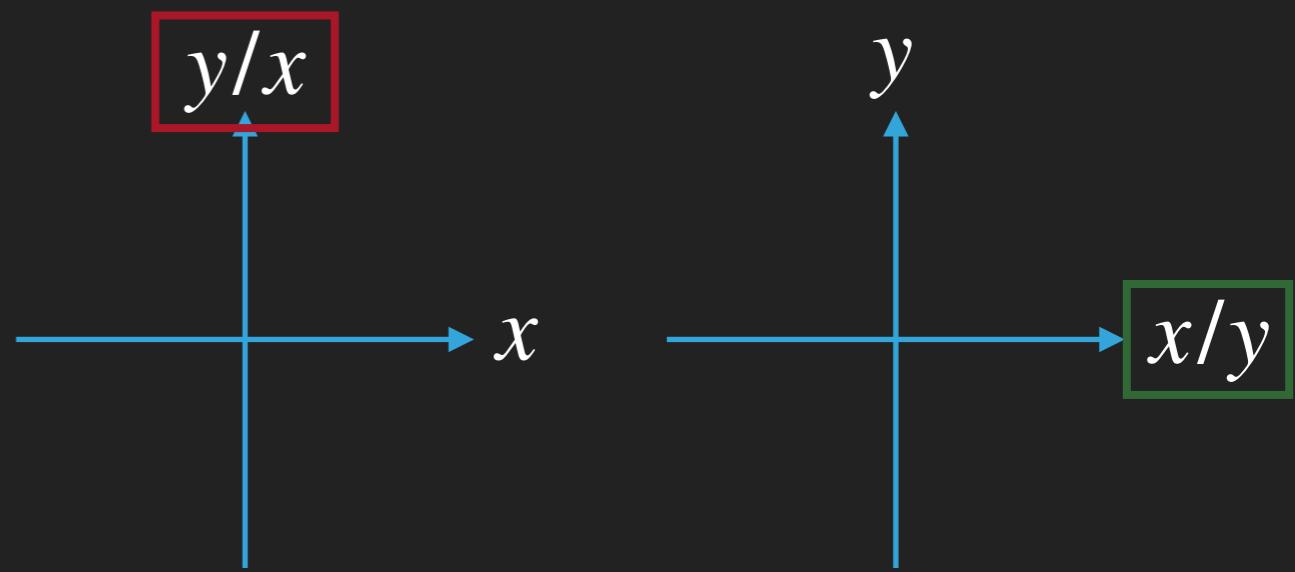
1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



アルゴリズム 1-2

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

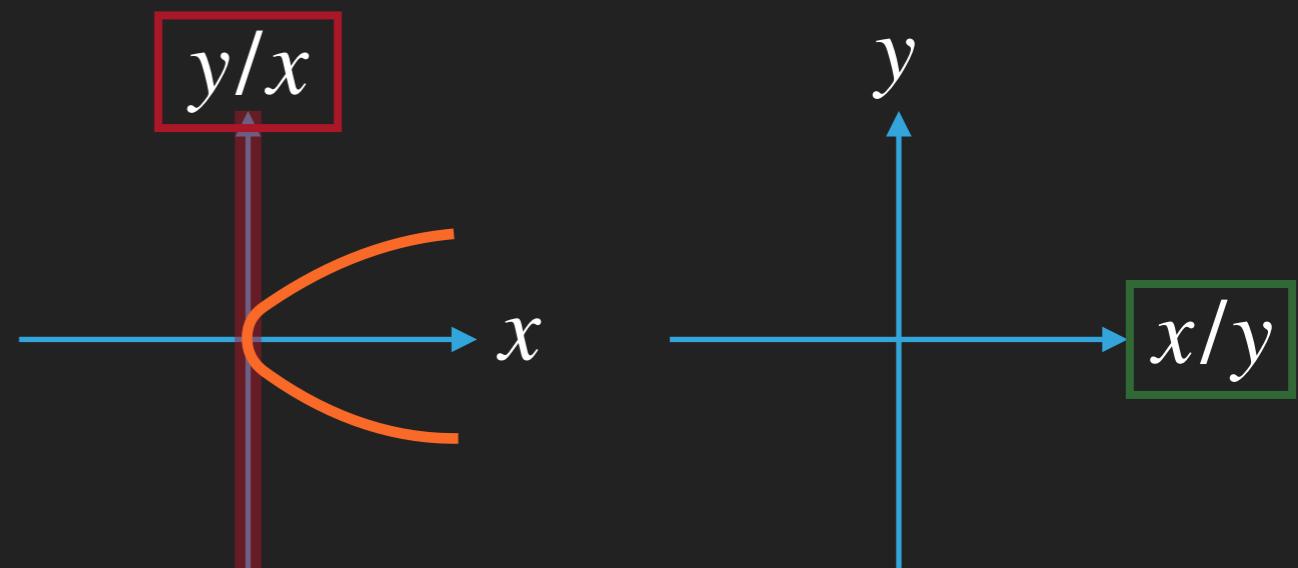


```
affine_open[0]
exceptional_list = []
exc1 = ExceptionalCurve()
```

アルゴリズム 1-3

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`

`exceptional_list = []`

`exc1`

$$x - (y/x)^2 = 0$$

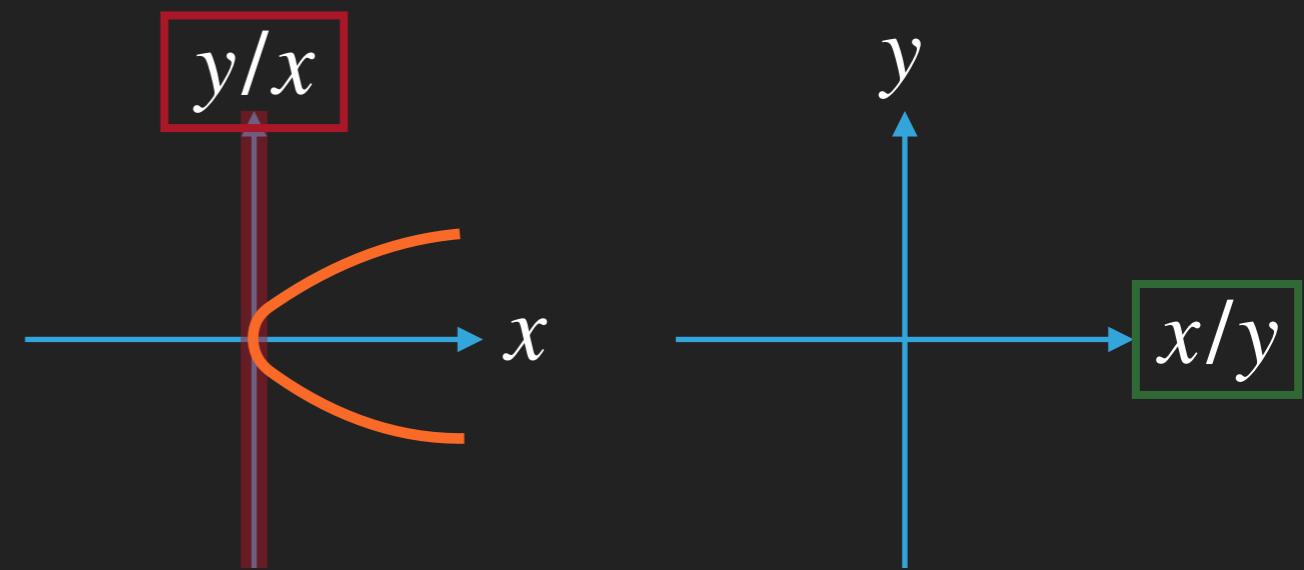
$$x = 0$$

`affine_open[1]`

アルゴリズム 1-4

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

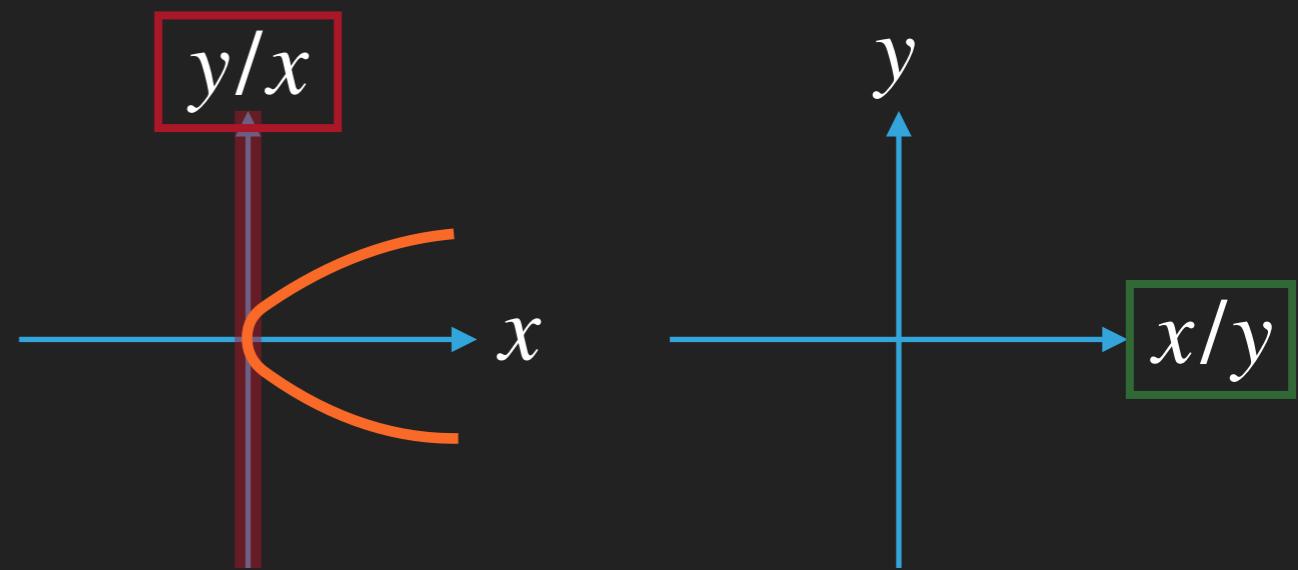
affine_open[0]           affine_open[1]
exceptional_list = []
exc1
x - (y/x)2 = 0 :nonsingular
x = 0 :not normal crossing

```

アルゴリズム 1-5

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

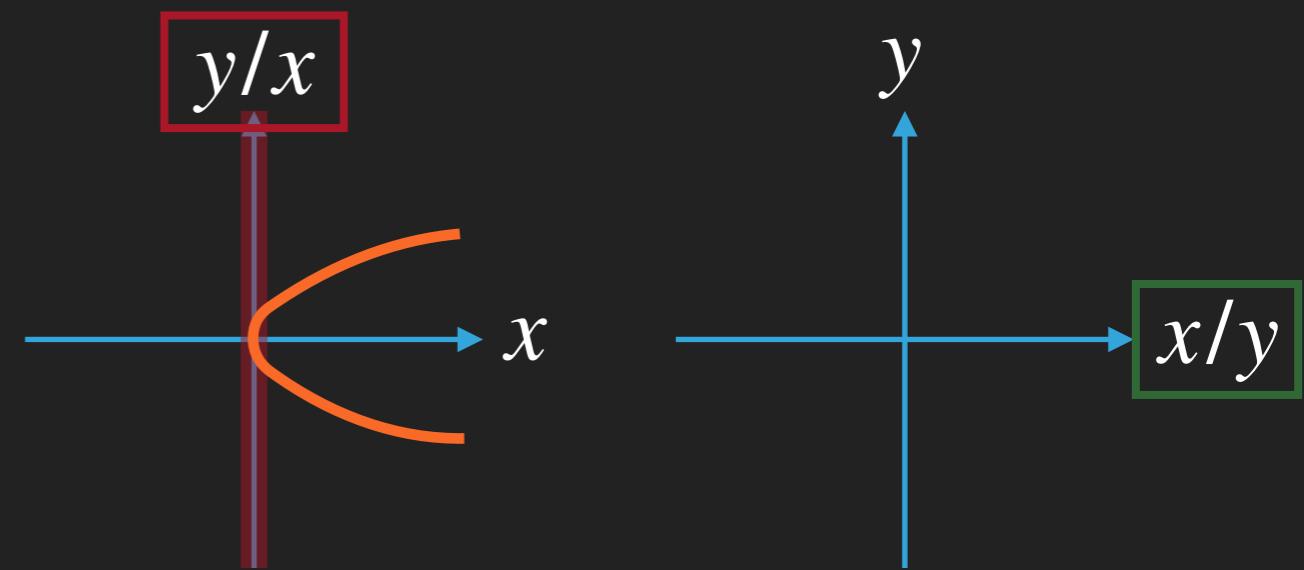
affine_open[0]           affine_open[1]
exceptional_list = []
exc1.set(affine_open[0], x)
x - (y/x)2 = 0 :nonsingular
x = 0 :not normal crossing

```

アルゴリズム 1-6

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]` `affine_open[1]`

`exceptional_list = []`

`exc1.divisors =`

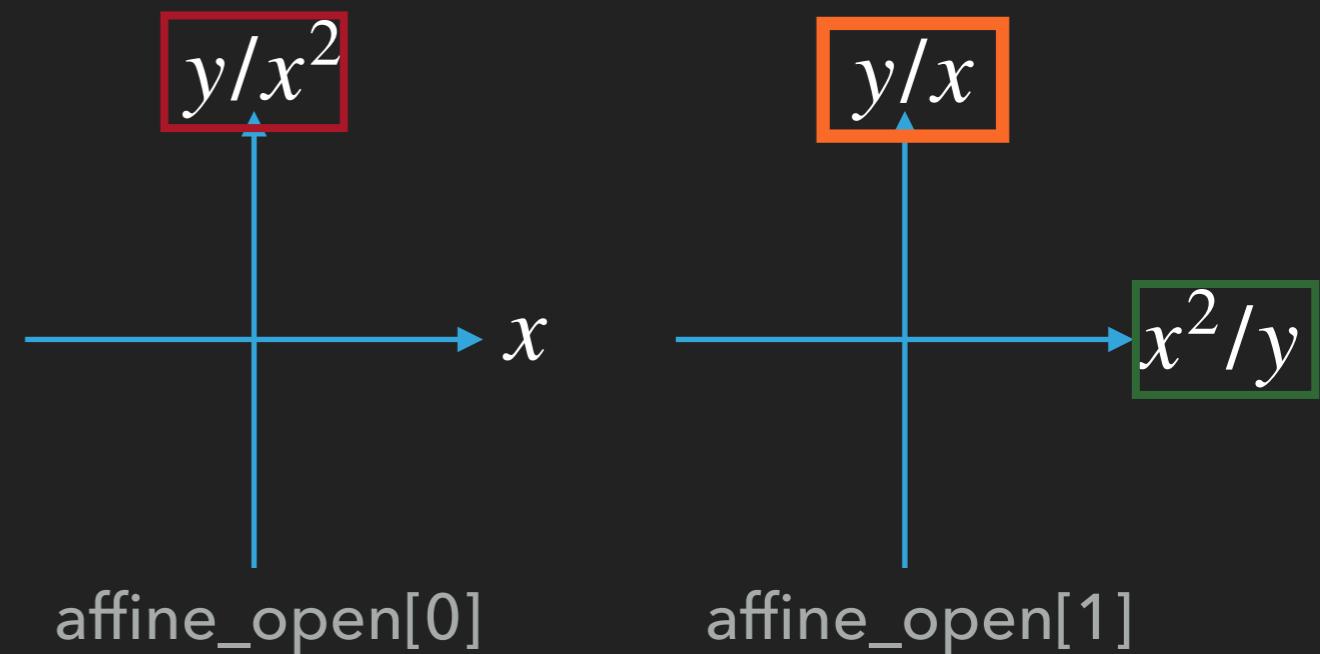
`[{'open': affine_open[0], 'ideal': x}]`

→再帰的にブローアップ

アルゴリズム 2-1

コンピュータアルゴリズム

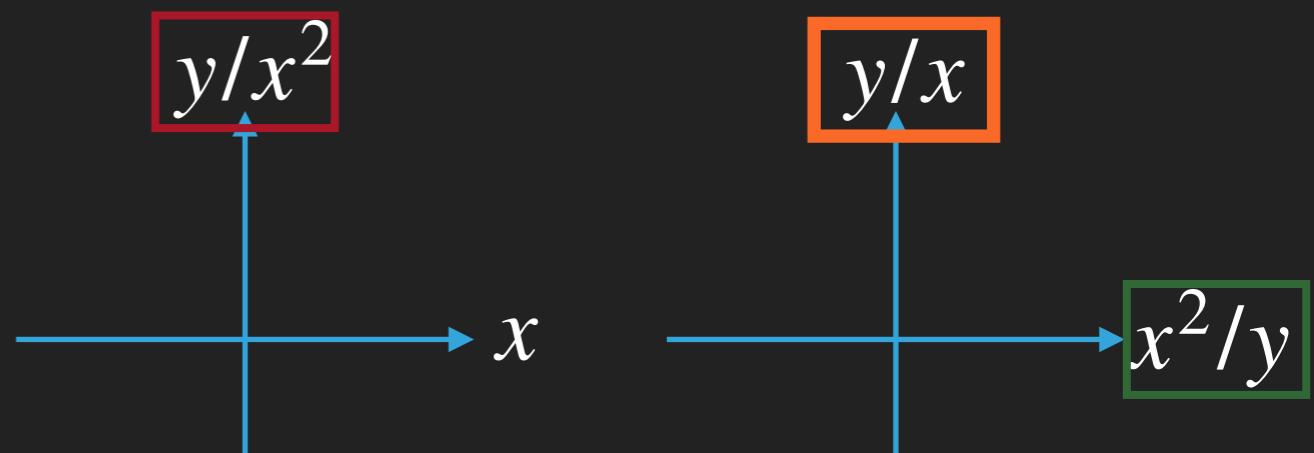
1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



アルゴリズム 2-2

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

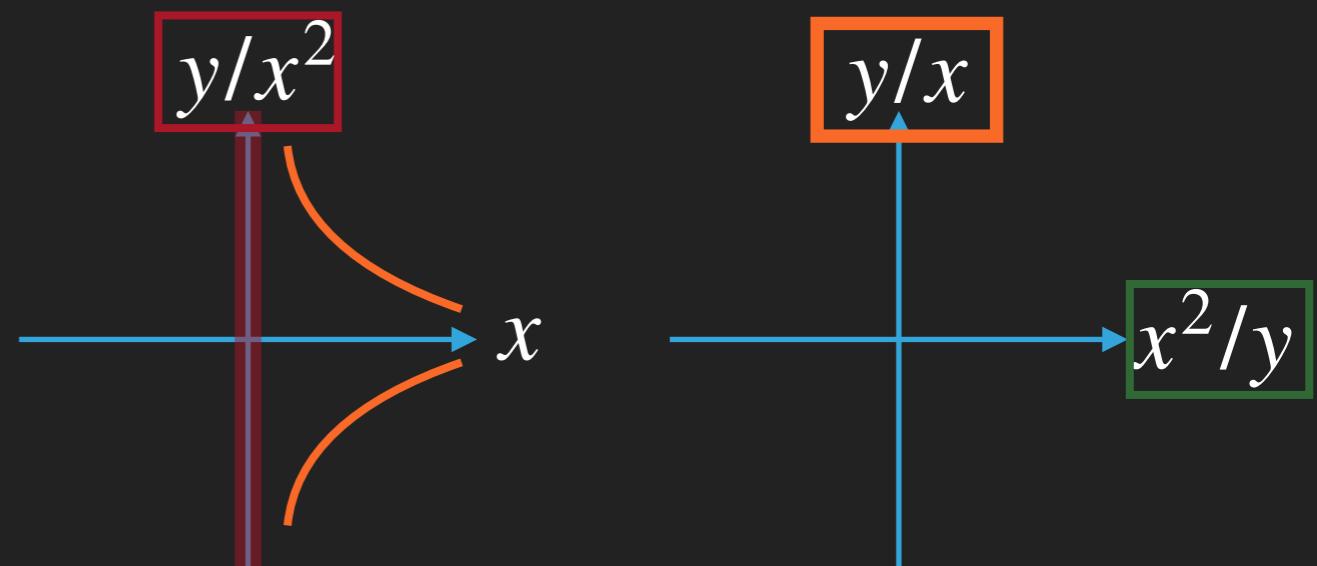


```
affine_open[0]           affine_open[1]
exceptional_list = []
exc2 = ExceptionalCurve()
```

アルゴリズム 2-3

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

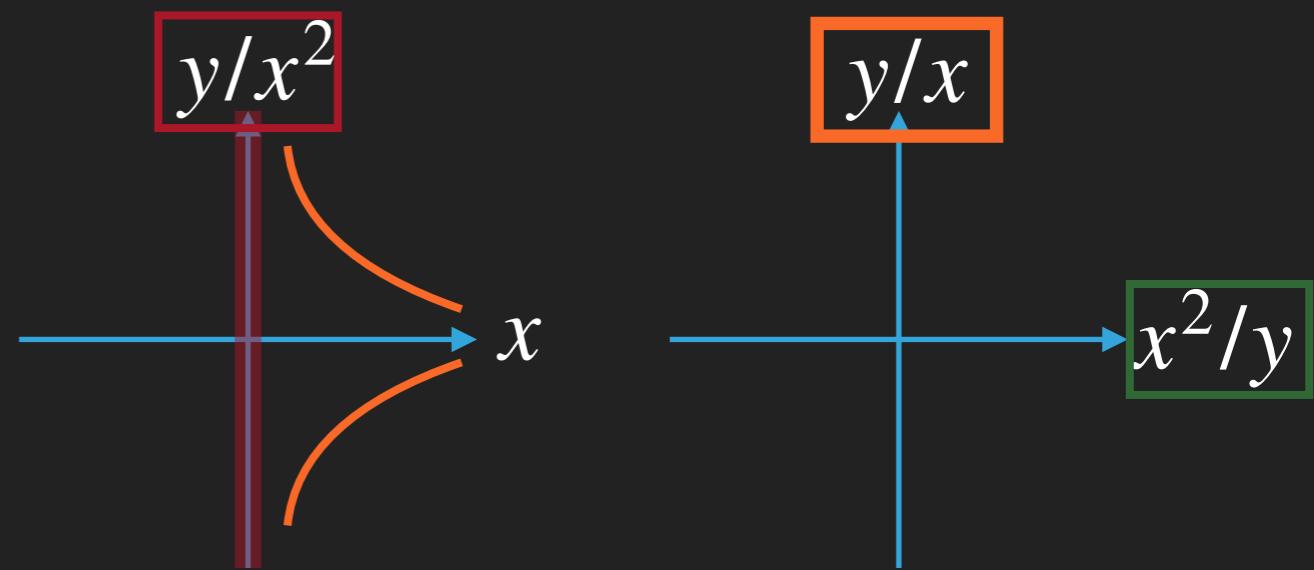


```
affine_open[0]
exceptional_list = []
exc2
-x(y/x^2)^2 + 1 = 0
x = 0
```

アルゴリズム 2-4

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

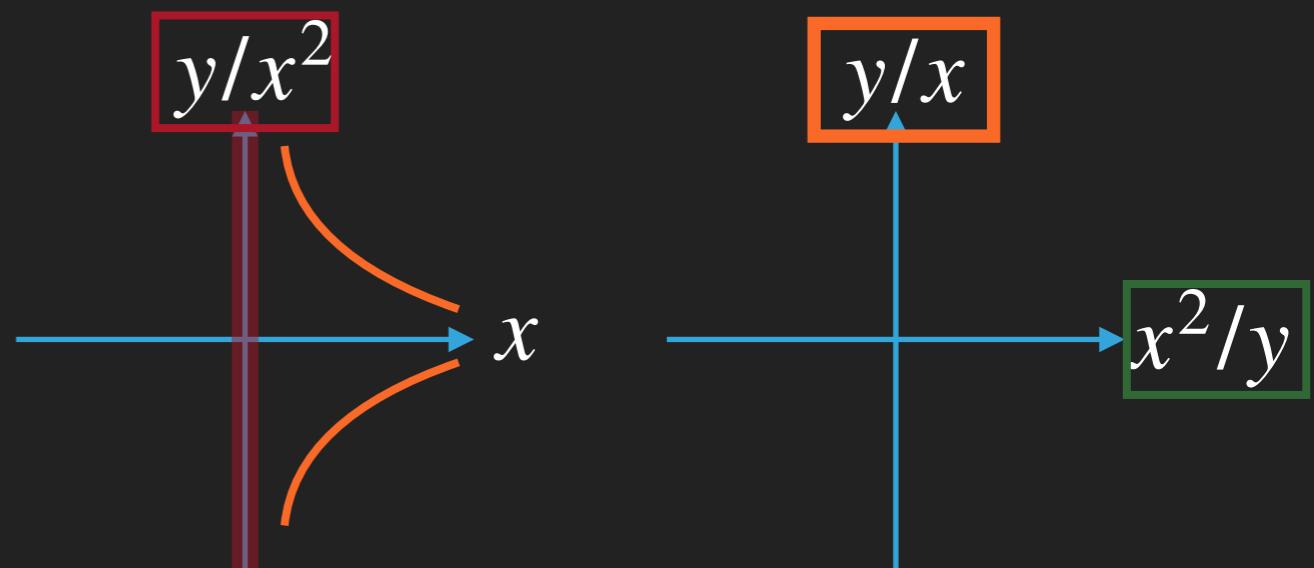


```
affine_open[0]
exceptional_list = []
exc2
-x(y/x^2)^2 + 1 = 0 :nonsingular
x = 0 :no crossing
```

アルゴリズム 2-5

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

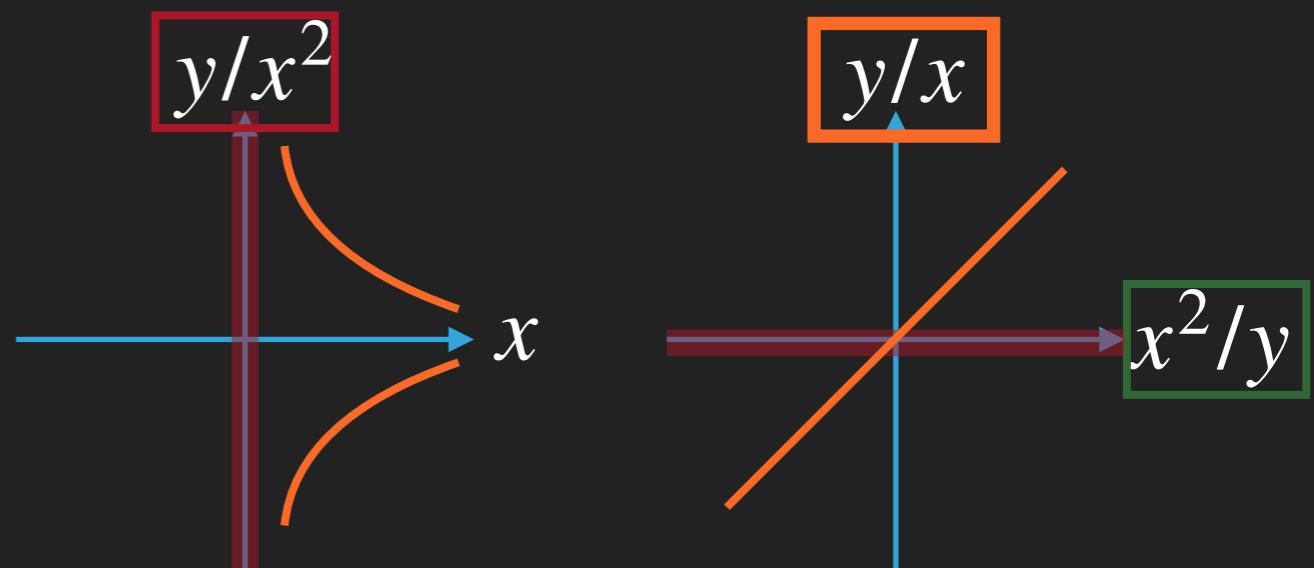
affine_open[0]           affine_open[1]
exceptional_list = []
exc2.set(affine_open[0], x)
-x(y/x^2)^2 + 1 = 0 :nonsingular
x = 0 :no crossing

```

アルゴリズム 2-6

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`

`exceptional_list = []`

`exc2`

$$(x^2/y) - (y/x) = 0$$

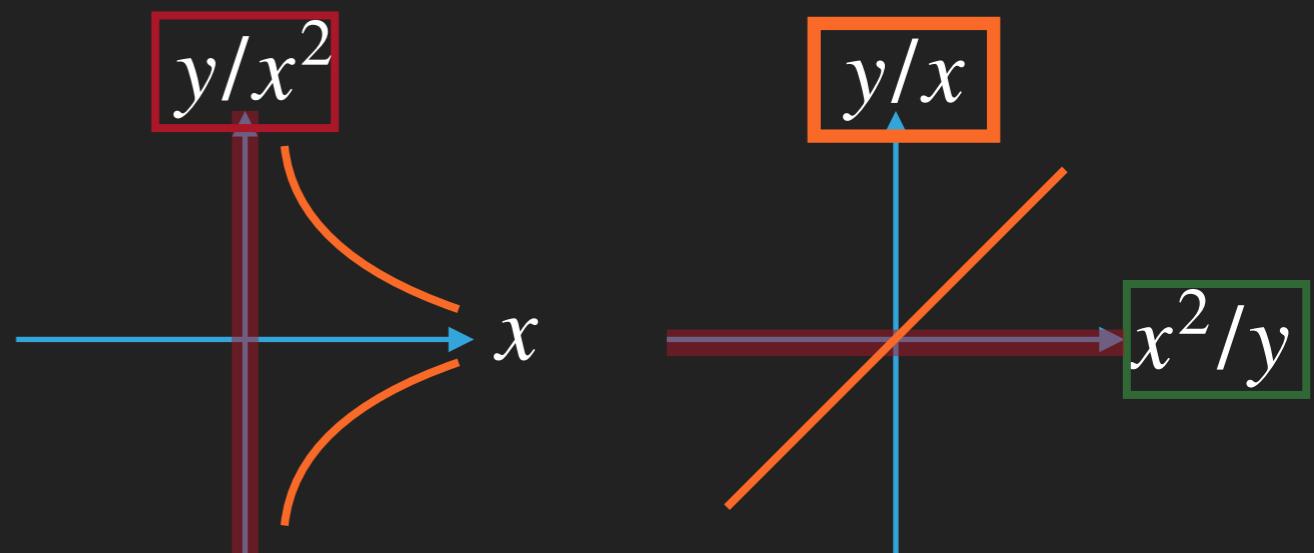
$$y = 0$$

`affine_open[1]`

アルゴリズム 2-7

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

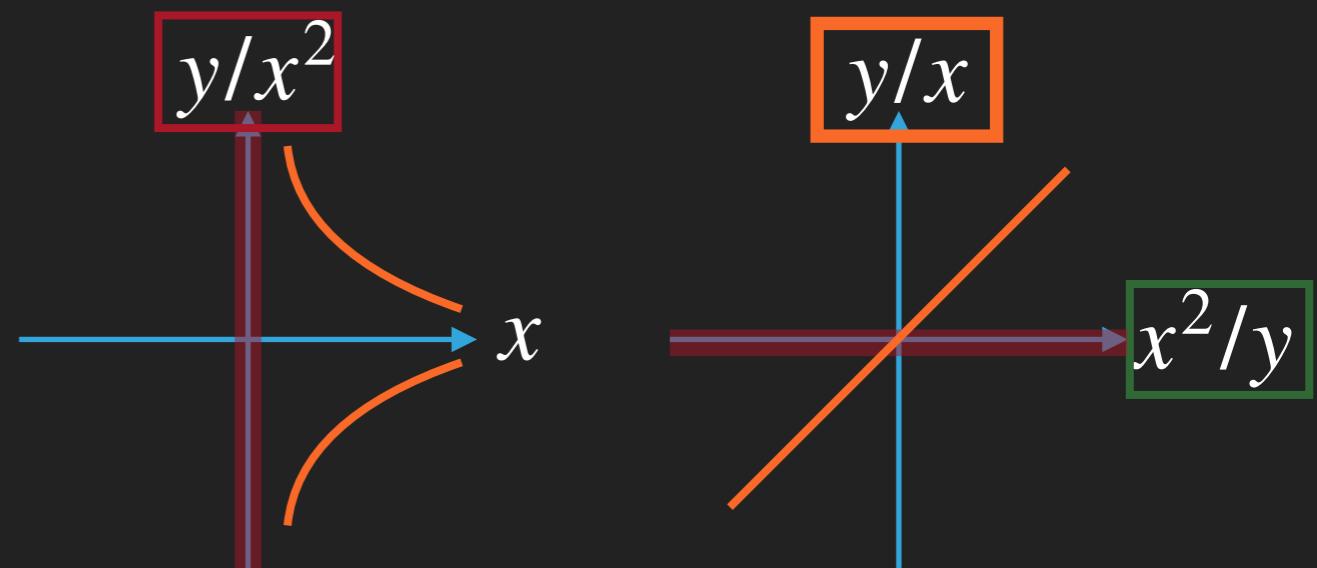


```
affine_open[0]
exceptional_list = []
exc2
(x2/y) - (y/x) = 0 :nonsingular
y = 0 :not normal crossing
```

アルゴリズム 2-8

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

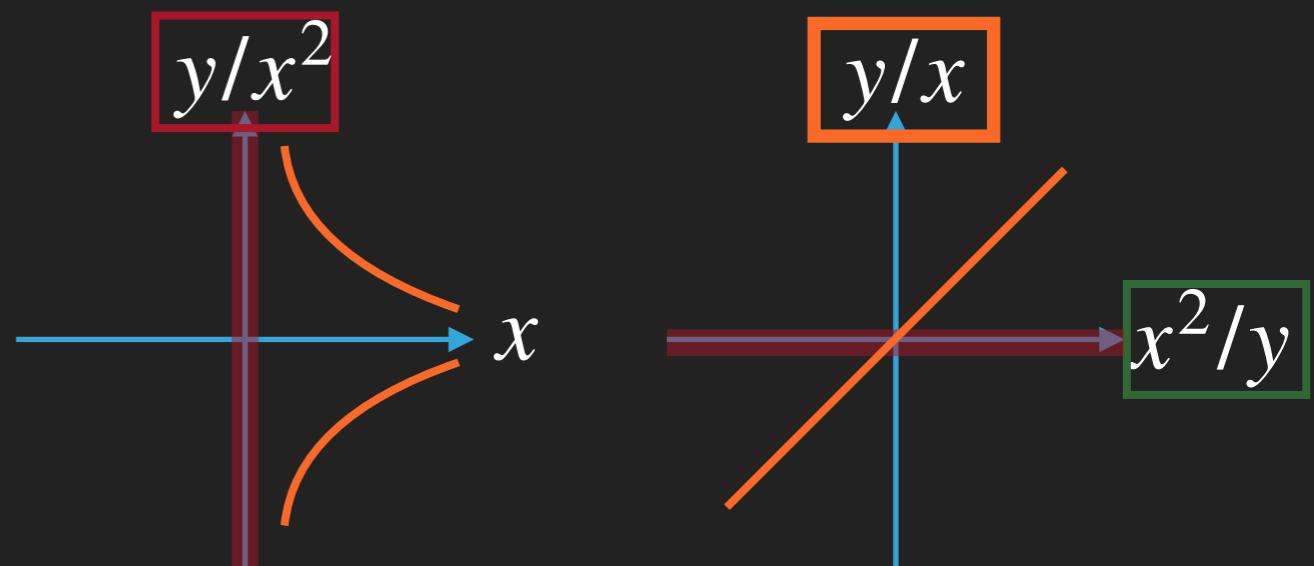


```
affine_open[0]
exceptional_list = []
exc2.set(affine_open[1], y)
(x2/y) - (y/x) = 0 :nonsingular
y = 0 :not normal crossing
```

アルゴリズム 2-9

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

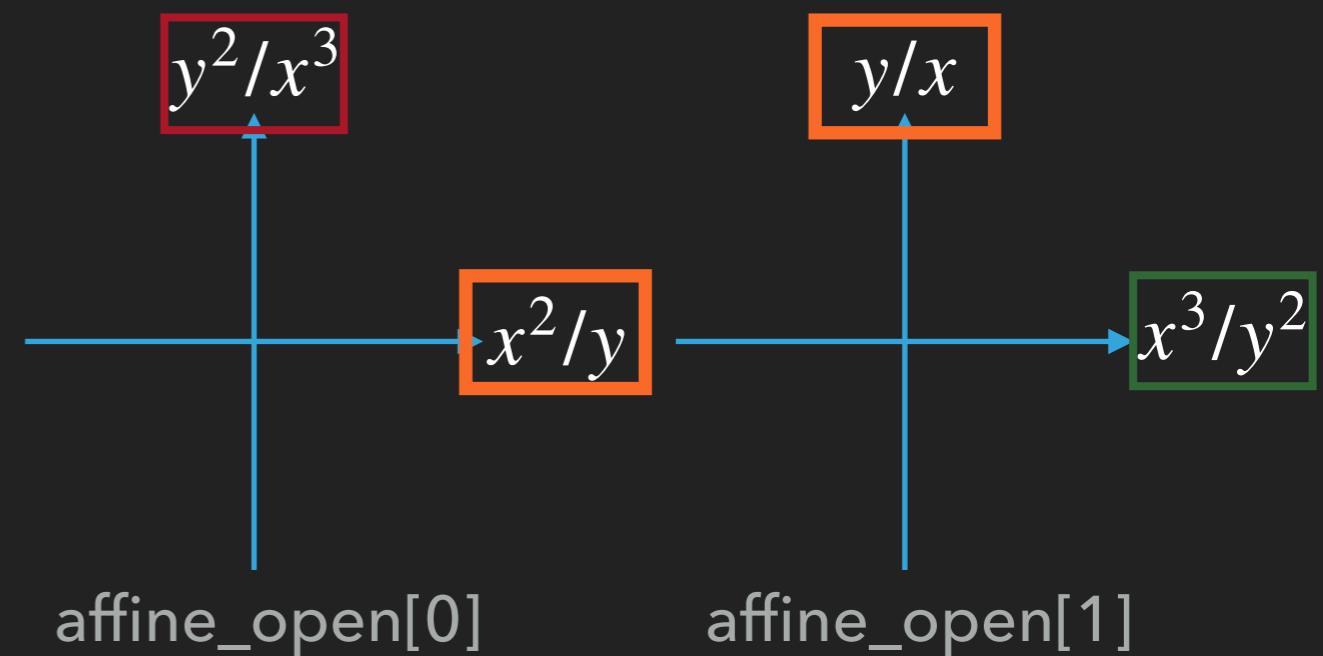


```
affine_open[0]
exceptional_list = []
exc2.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y}]
→再帰的にブローアップ
```

アルゴリズム 3-1

コンピュータアルゴリズム

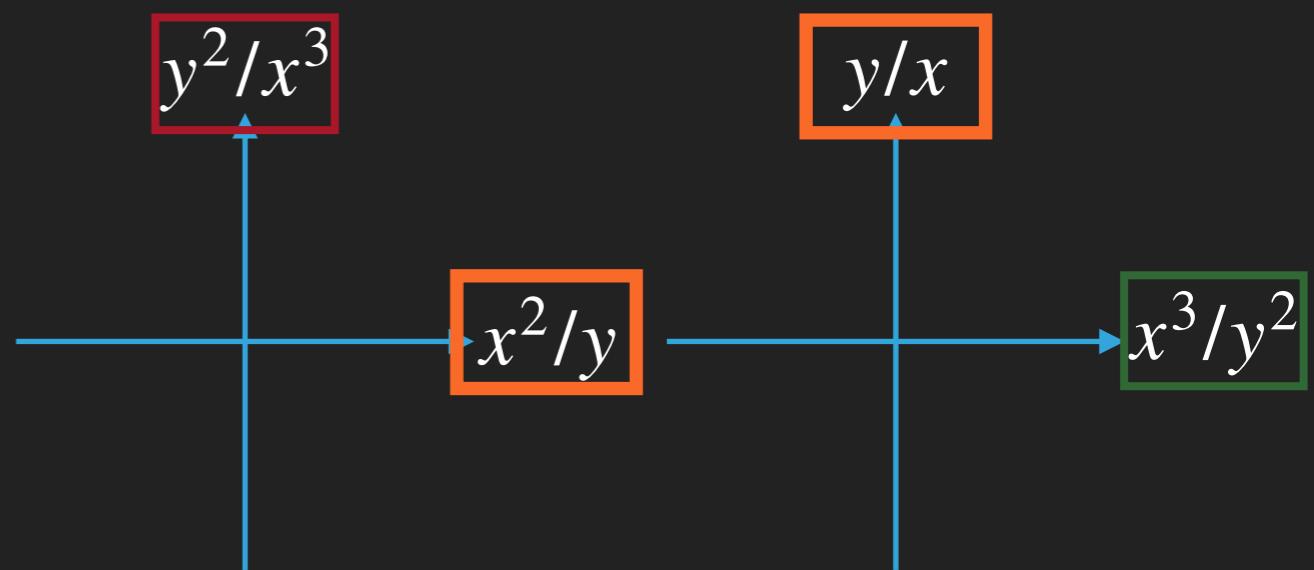
1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



アルゴリズム 3-2

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

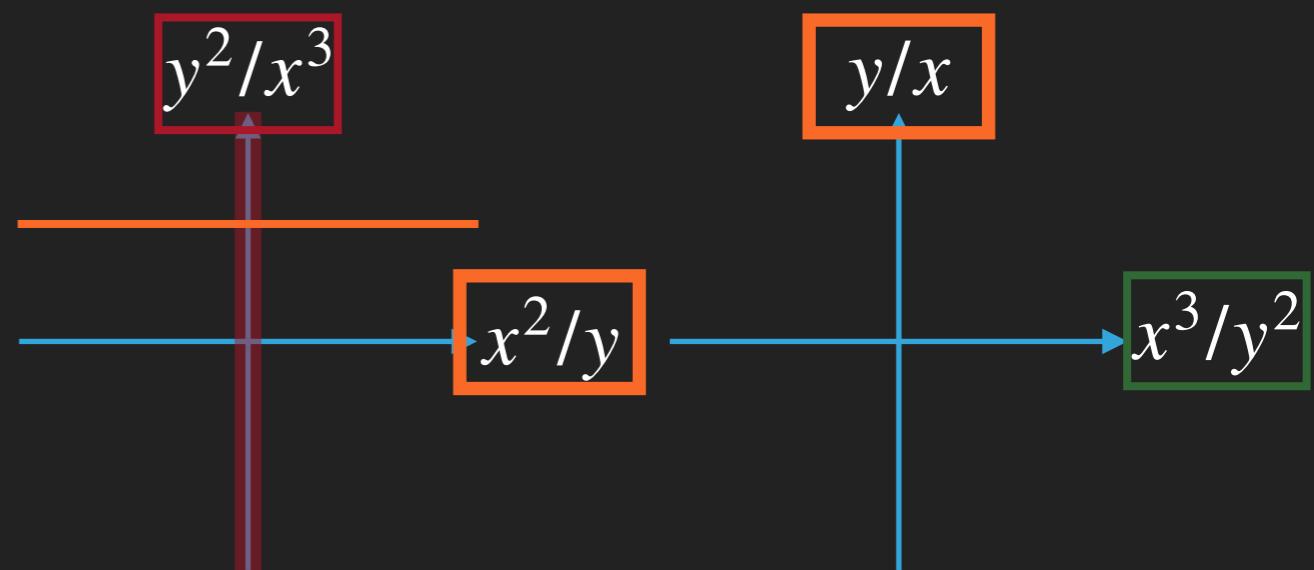


```
affine_open[0]           affine_open[1]
exceptional_list = []
exc3 = ExceptionalCurve()
```

アルゴリズム 3-3

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

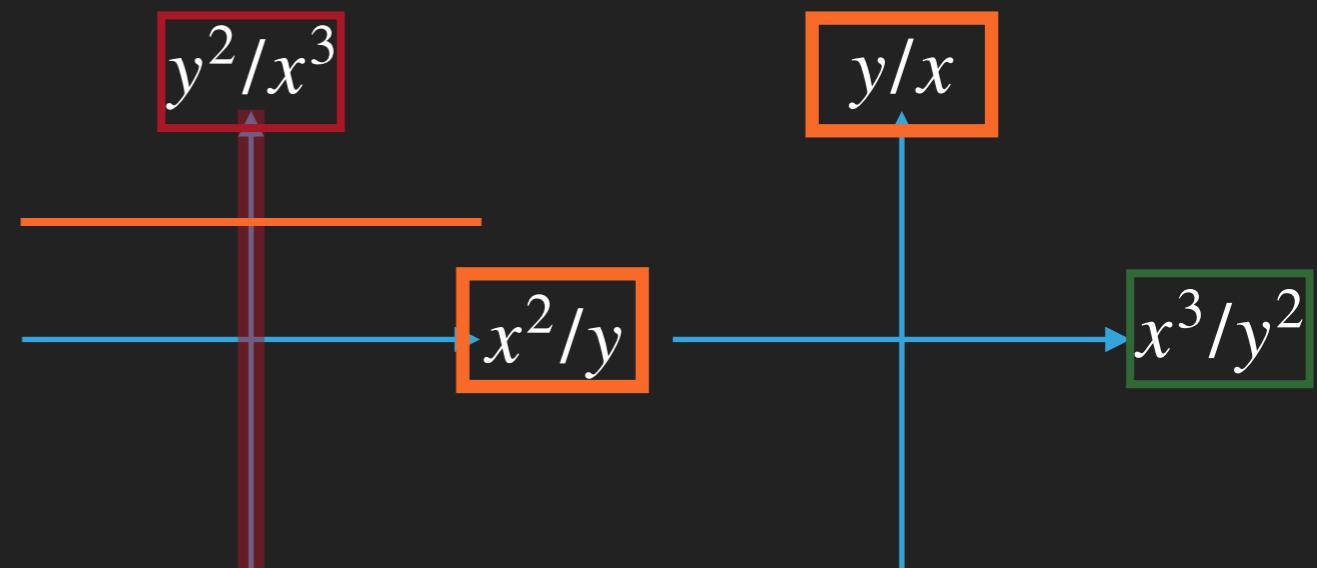


```
affine_open[0]
exceptional_list = []
exc3
1 - (y2/x3) = 0
x2/y = 0
```

アルゴリズム 3-4

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

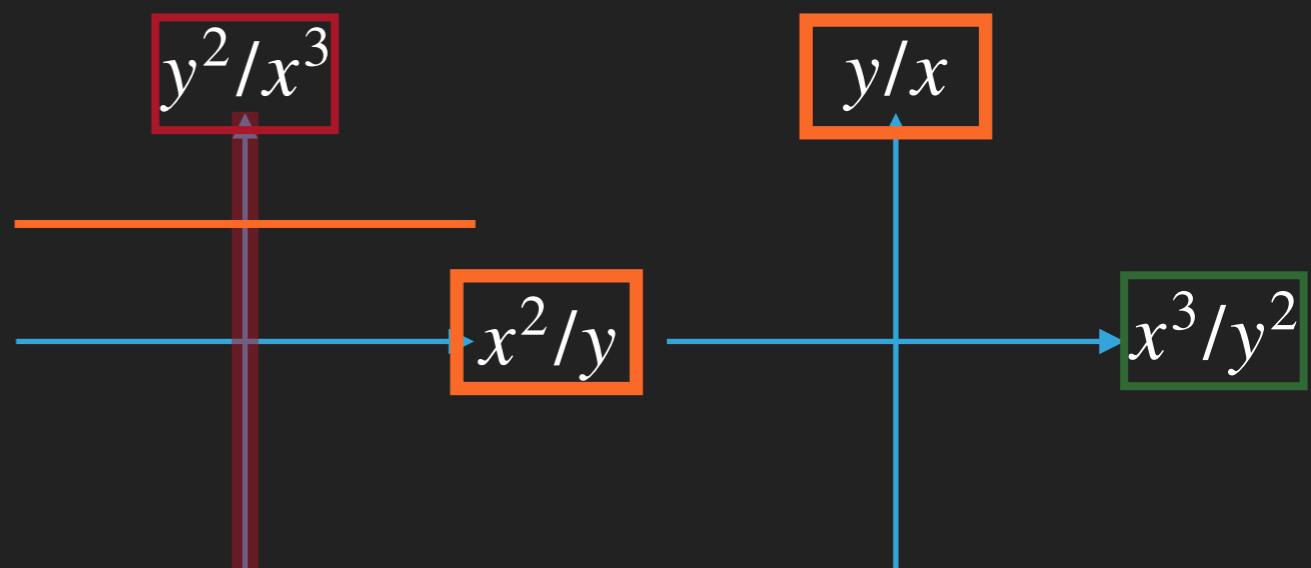
affine_open[0]           affine_open[1]
exceptional_list = []
exc3
1 - (y2/x3) = 0 :nonsingular
x2/y = 0 :normal crossing
nss = NonsingularStrictTransform()

```

アルゴリズム 3-5

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

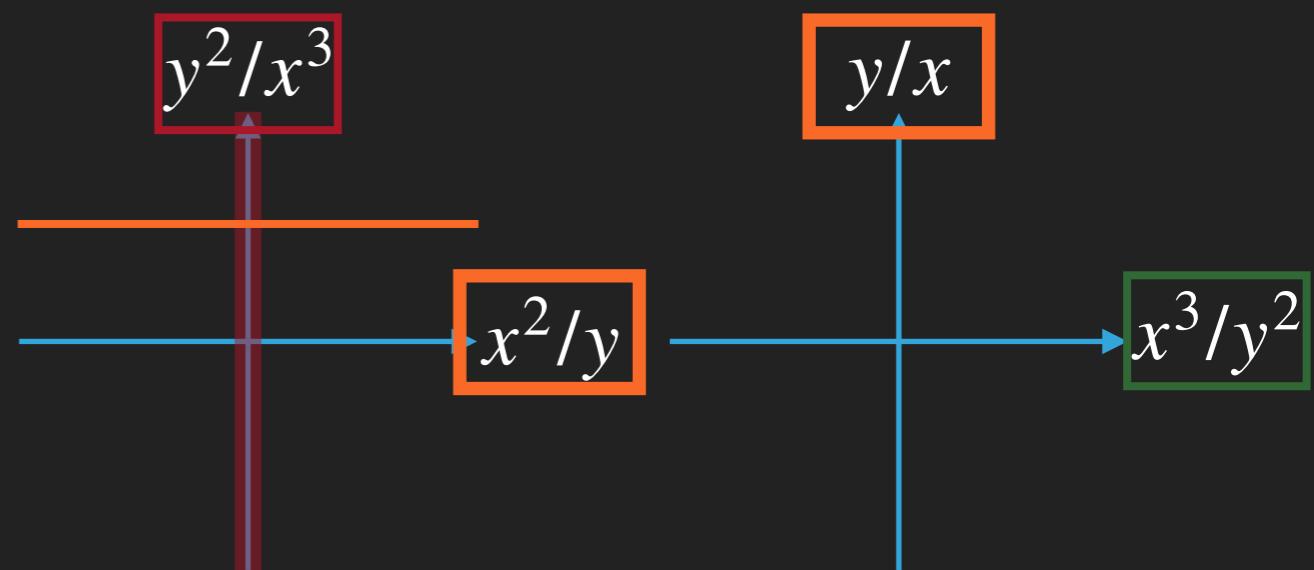
affine_open[0]
exceptional_list = []
exc3
1 - (y2/x3) = 0 :nonsingular
x2/y = 0 :normal crossing
nss.set(affine_open[0], 1 - y)

```

アルゴリズム 3-6

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

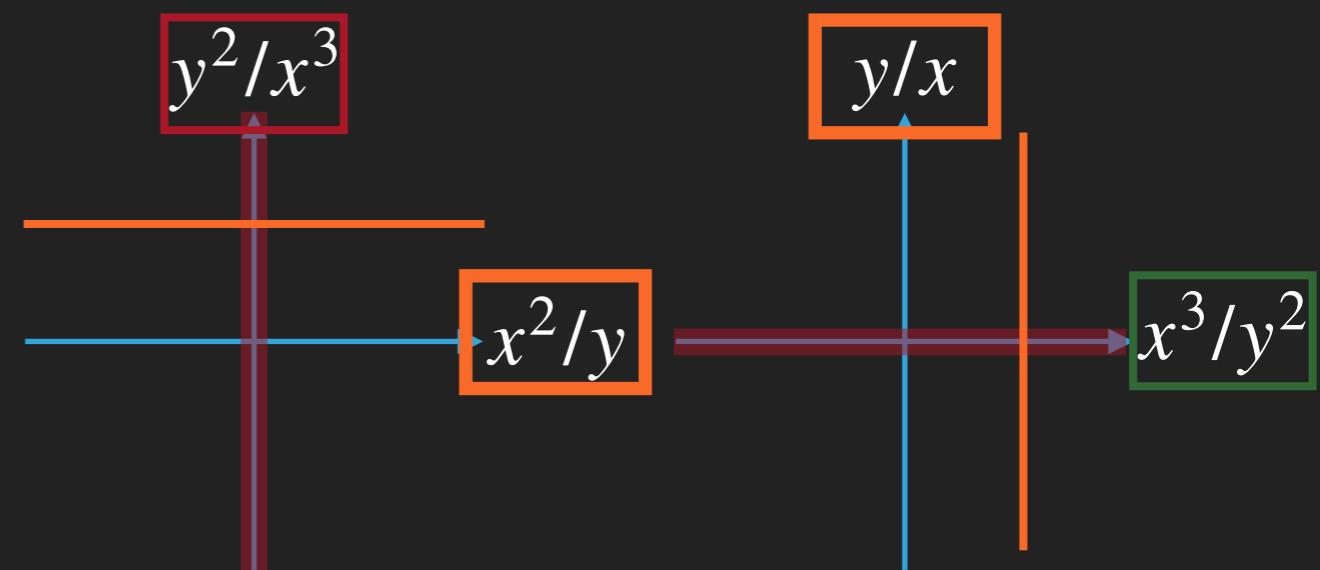
affine_open[0]           affine_open[1]
exceptional_list = []
exc3.set(affine_open[0], x)
1 - (y2/x3) = 0 :nonsingular
x2/y = 0 :normal crossing

```

アルゴリズム 3-7

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

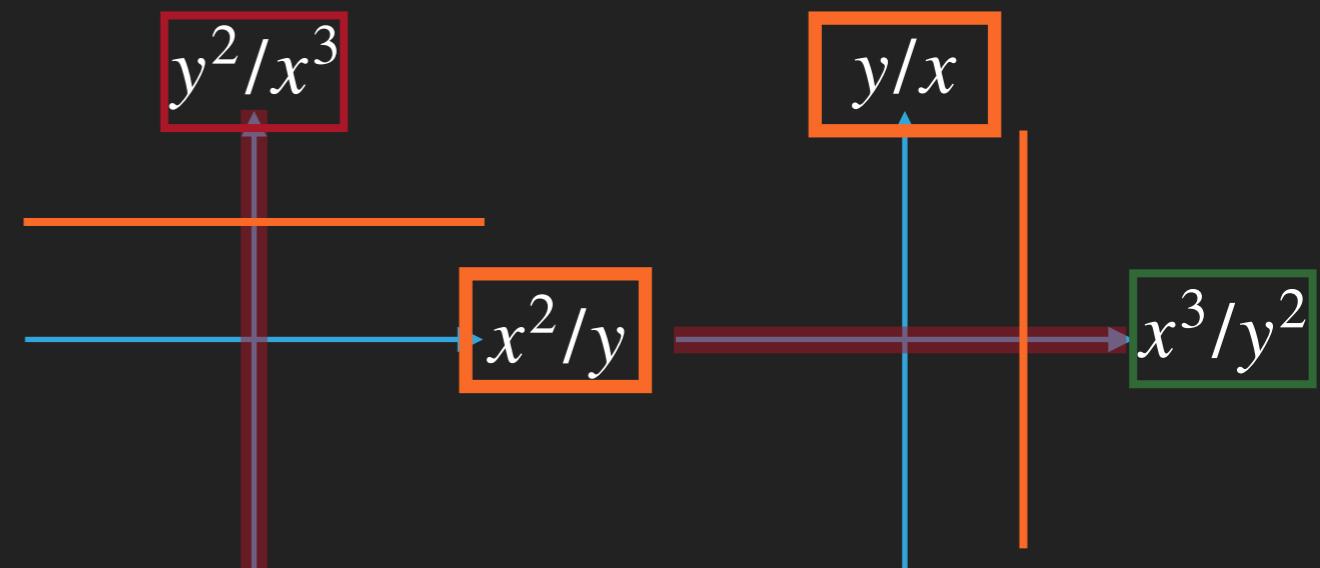


```
affine_open[0]
exceptional_list = []
exc3
 $(x^3/y^2) - 1 = 0$ 
 $y/x = 0$ 
```

アルゴリズム 3-8

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

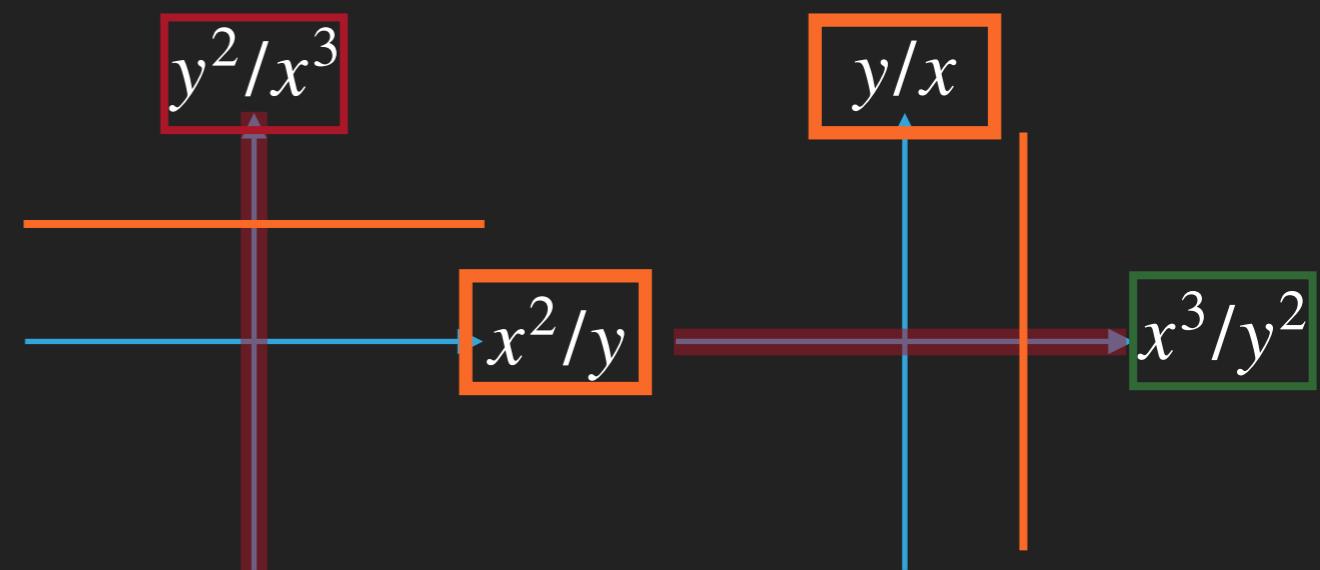
affine_open[0]
exceptional_list = []
exc3
(x3/y2) - 1 = 0 :nonsingular
y/x = 0 :normal crossing
nss.set(affine_open[1], x - 1)

```

アルゴリズム 3-9

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

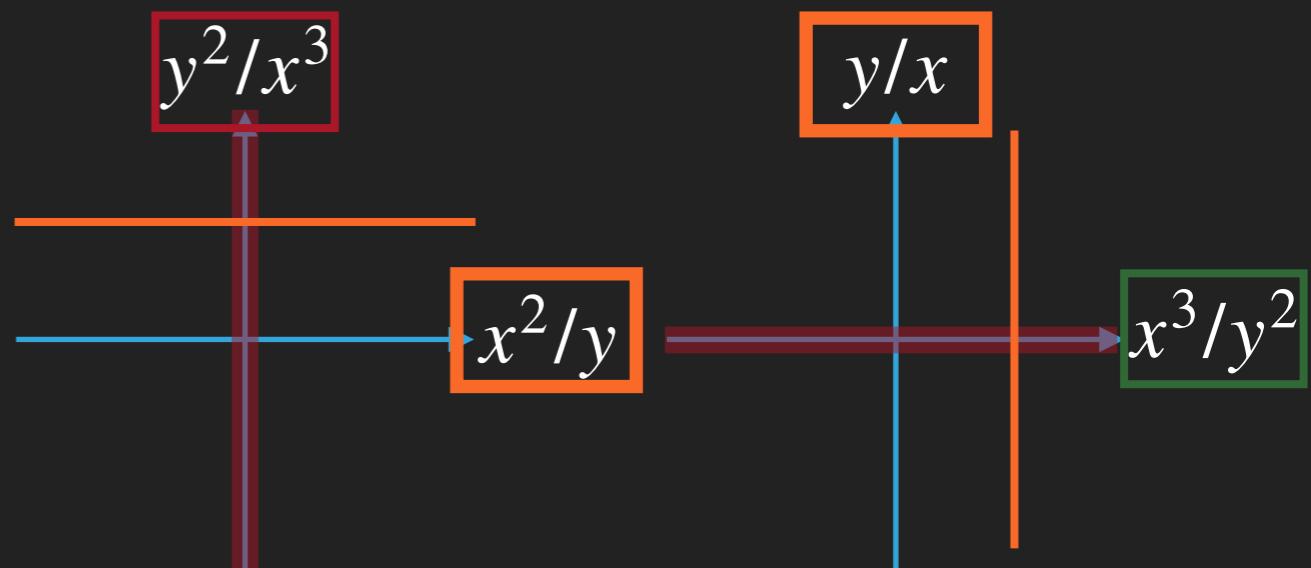


```
affine_open[0]           affine_open[1]
exceptional_list = []
exc3.set(affine_open[1], y)
(x3/y2) - 1 = 0 :nonsingular
y/x = 0 :normal crossing
```

アルゴリズム 3-10

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

affine_open[0]
exceptional_list = []
exc3.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y}]

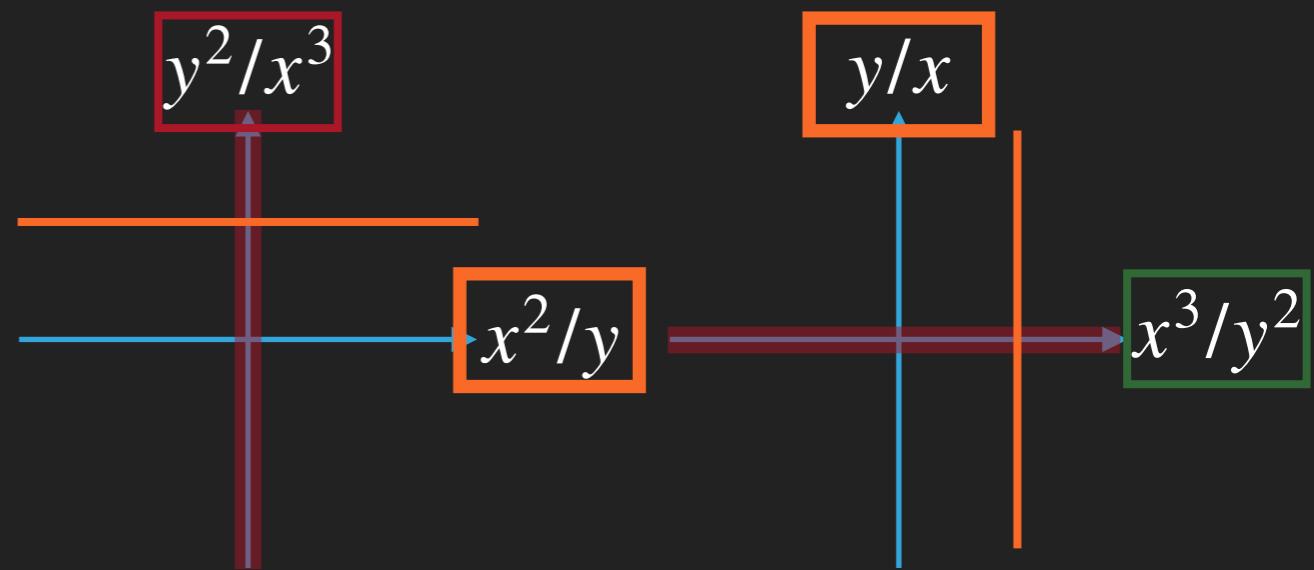
nss.divisors =
[{'open': affine_open[0], 'ideal': 1 - y},
 {'open': affine_open[1], 'ideal': x - 1}]

```

アルゴリズム 3-11

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`

`affine_open[1]`

`exceptional_list = [exc3, nss]`

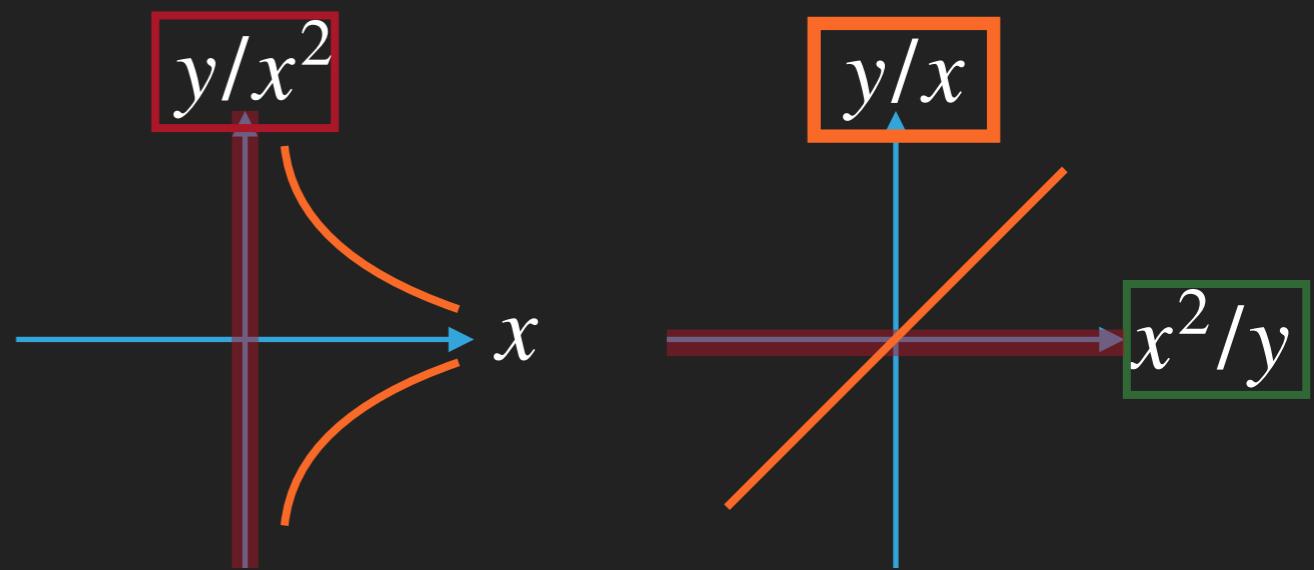
→一つ上の階層へ

`exceptional_list` を返す

アルゴリズム 2-9 (再掲)

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

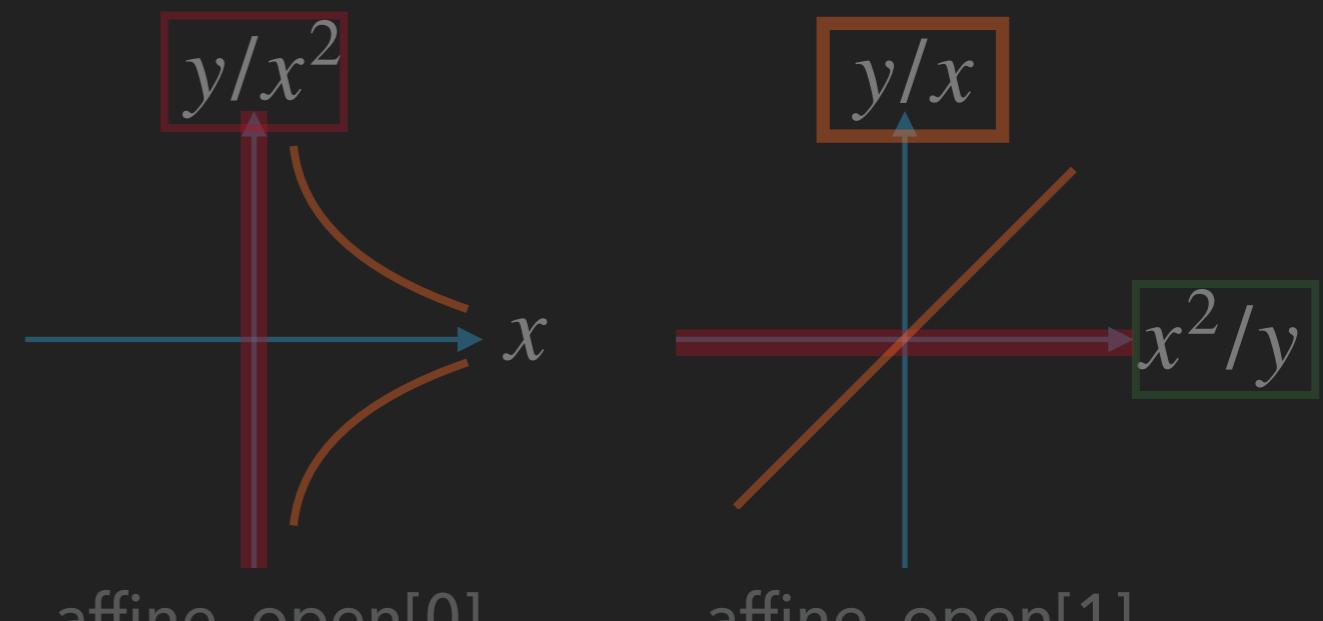


```
affine_open[0]           affine_open[1]
exceptional_list = [exc3, nss]
exceptional_curve.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y}]
→再帰的にブローアップ
```

アルゴリズム 2-9 (再掲)

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきた`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`

`affine_open[1]`

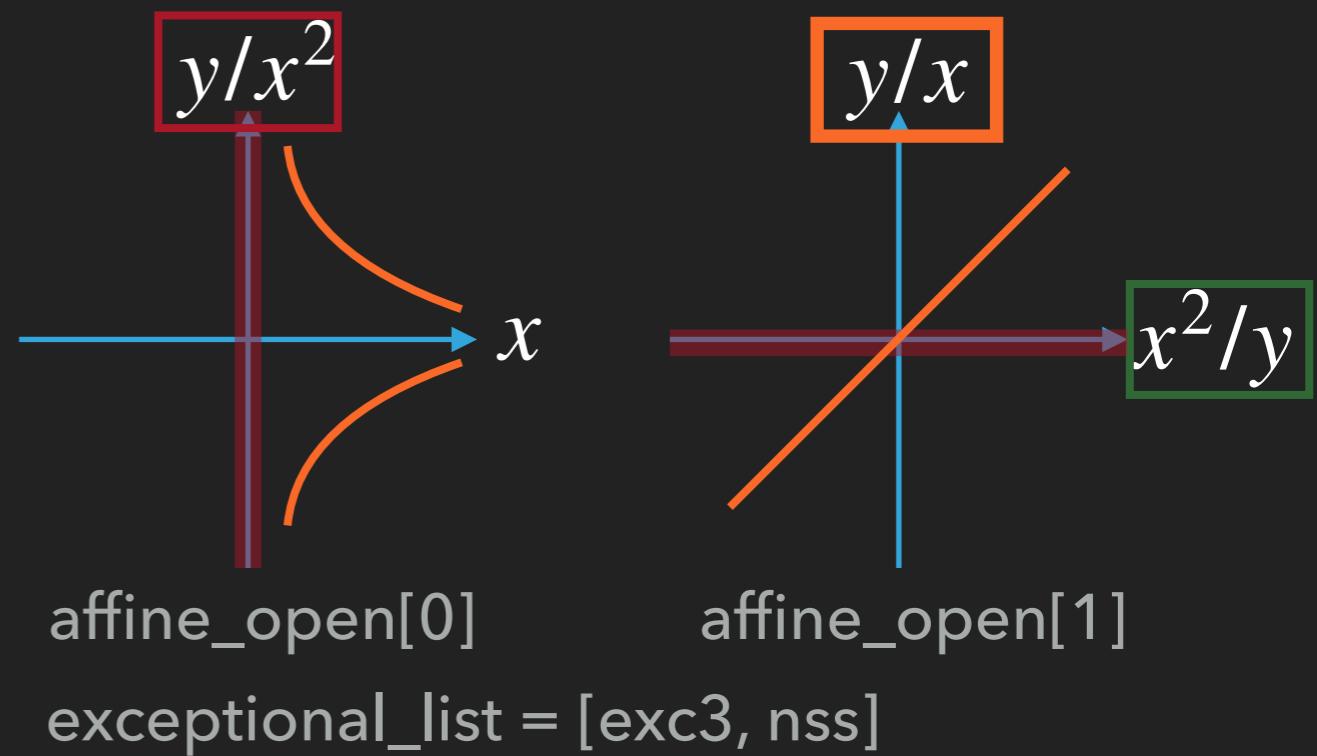
`exceptional_list = [exc3, nss]`

上の階層から返ってきた
リストを接続した.
→円錐的にフローアップ

アルゴリズム 2-10

コンピュータアルゴリズム

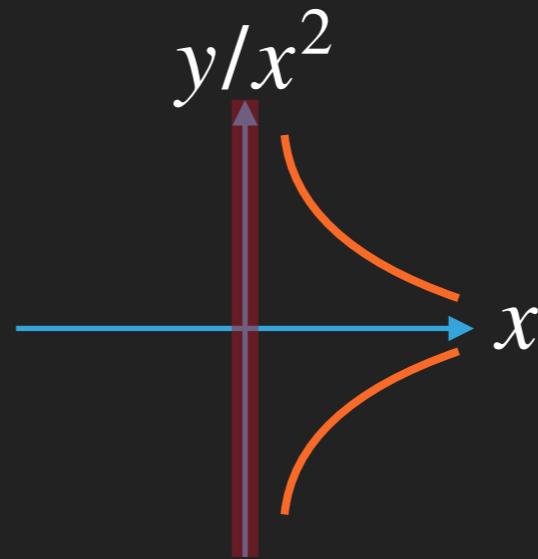
1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



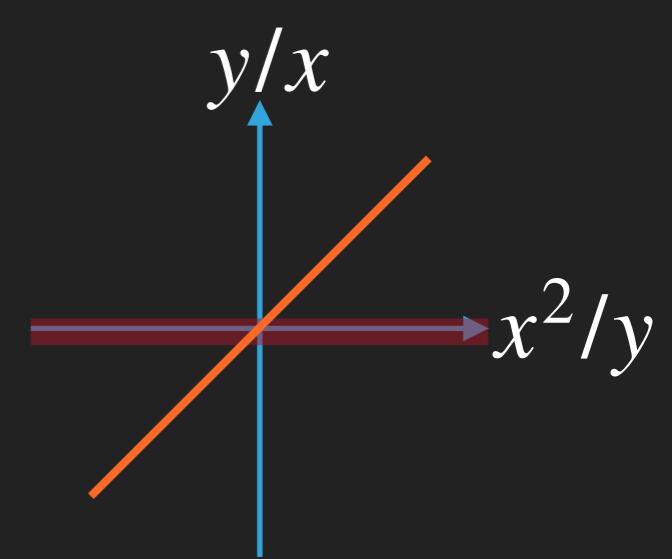
アルゴリズム 2-11

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



affine_open[0]



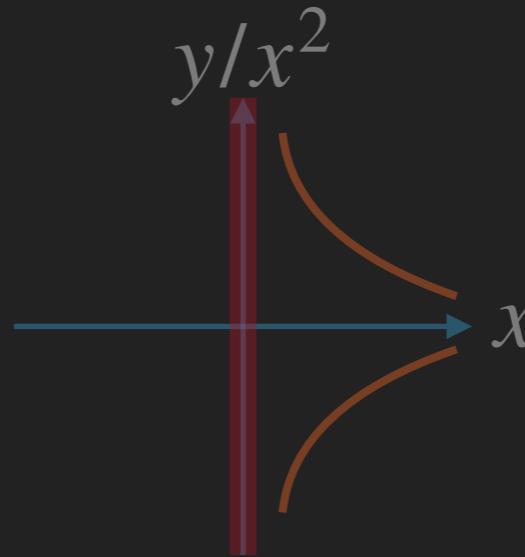
affine_open[1]

```
exceptional_list = [exc2, exc3, nss]
exc2.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y},
 {'open': exc3.divisors[0]['open'], 'ideal': y}]
```

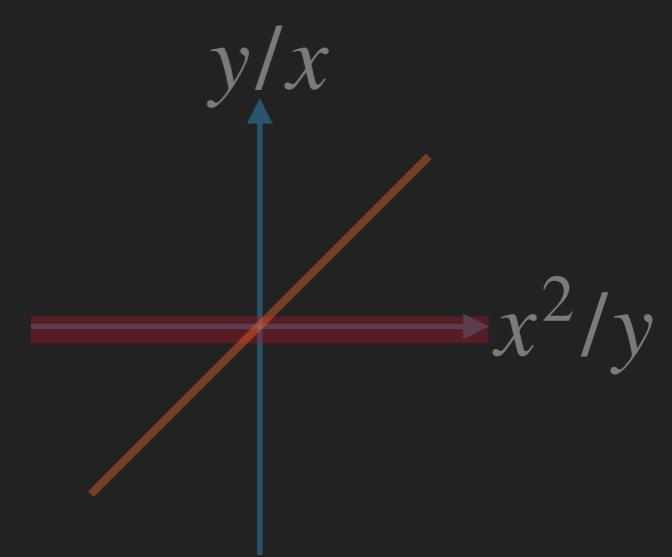
アルゴリズム 2-11

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`オブジェクトを求める.



`affine_open[0]`



`affine_open[1]`

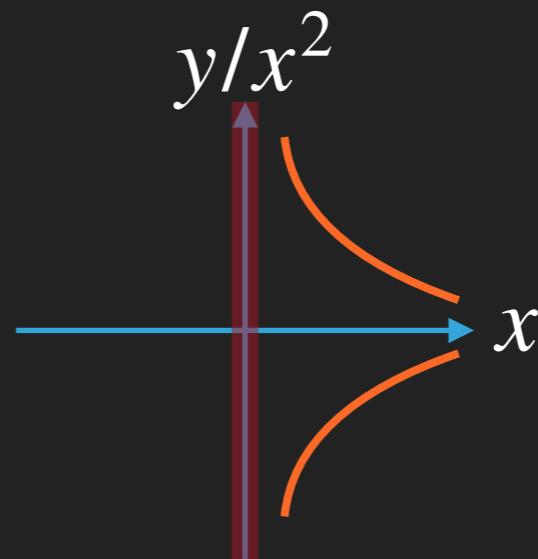
```
exceptional_list = [exc2, exc3, nss]
exc2.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y},
 {'open': exc3.divisors[0]['open'], 'ideal': y}]
```

上の階層から返ってきた
リストの中にある例外曲線から情報を受け取り
貼りあった先での値を追加.

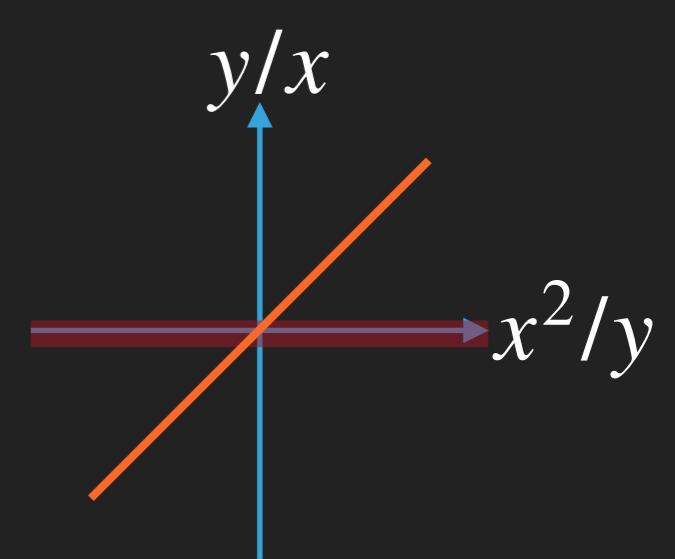
アルゴリズム 2-11

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



affine_open[0]



affine_open[1]

```

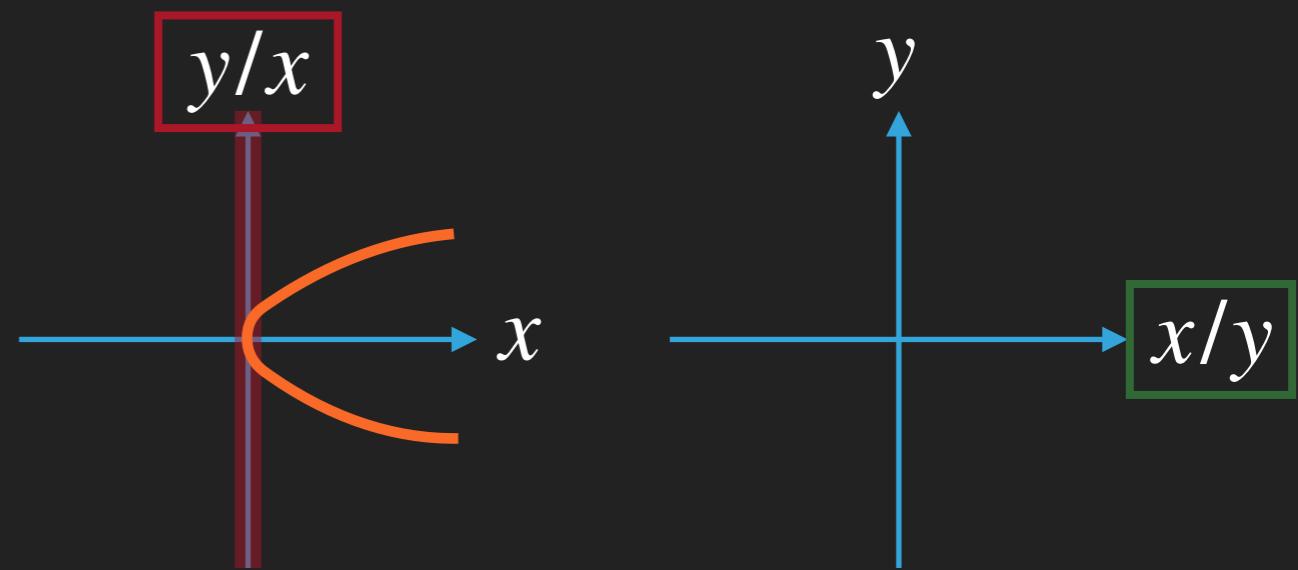
exceptional_list = [exc2, exc3, nss]
exc2.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y},
 {'open': exc3.divisors[0]['open'], 'ideal': y}]
→一つ上の階層へ
exceptional_list を返す

```

アルゴリズム 1-6 (再掲)

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]` `affine_open[1]`

`exceptional_list = [exc2, exc3, nss]`

`exc1.divisors =`

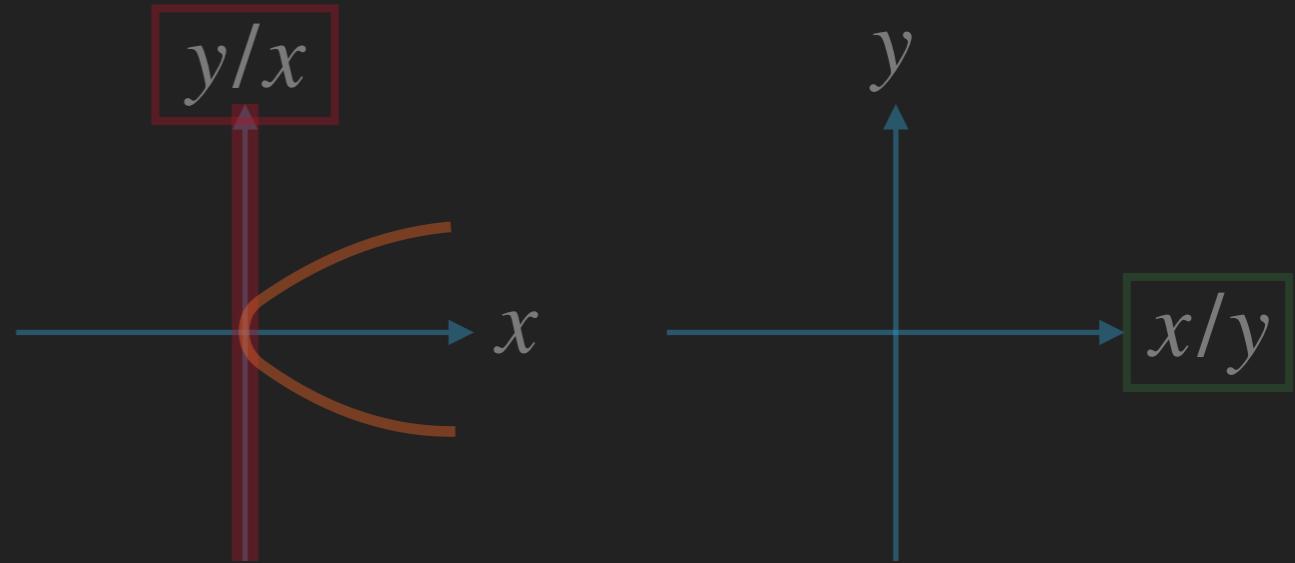
`[{'open': affine_open[0], 'ideal': x}]`

→再帰的にブローアップ

アルゴリズム 1-6 (再掲)

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきた`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`

`affine_open[1]`

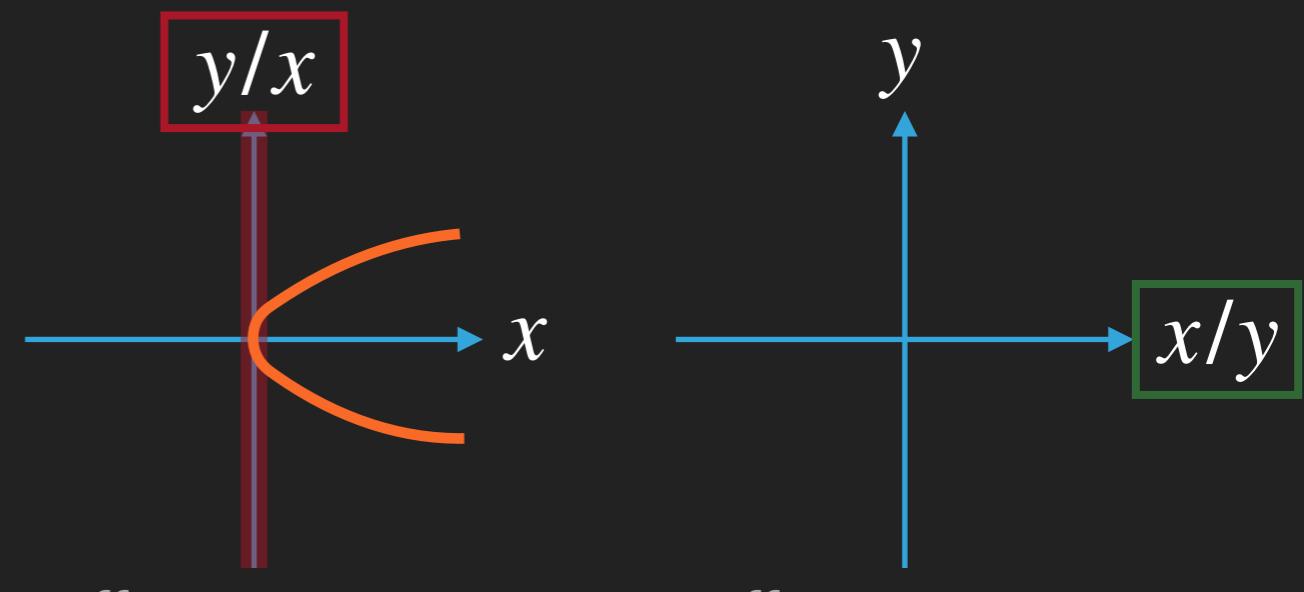
`exceptional_list = [exc2, exc3, nss]`

上の階層から返ってきた
リストを接続した.

アルゴリズム 1-7

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`

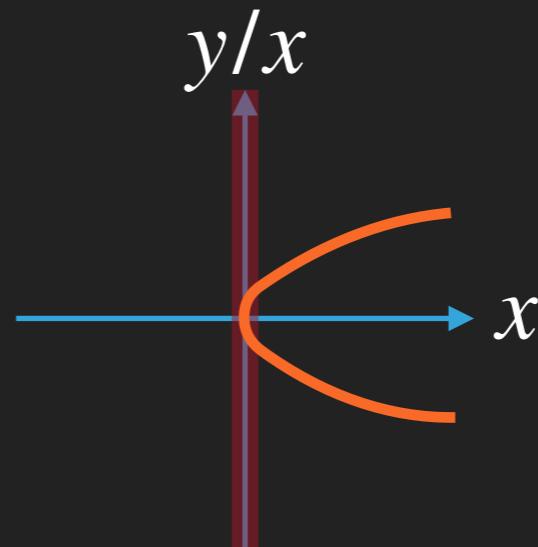
`affine_open[1]`

`exceptional_list = [exc2, exc3, nss]`

アルゴリズム 1-8

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



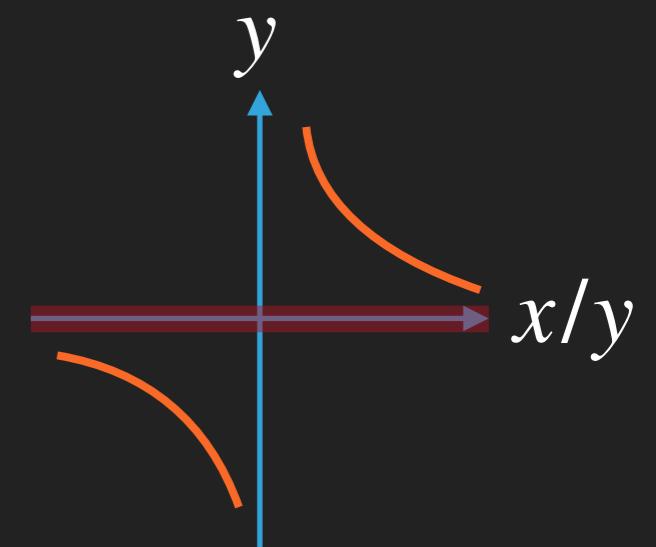
`affine_open[0]`

`exceptional_list = [exc2, exc3, nss]`

`exc1`

$$(x/y)^3 y - 1 = 0$$

$$y = 0$$

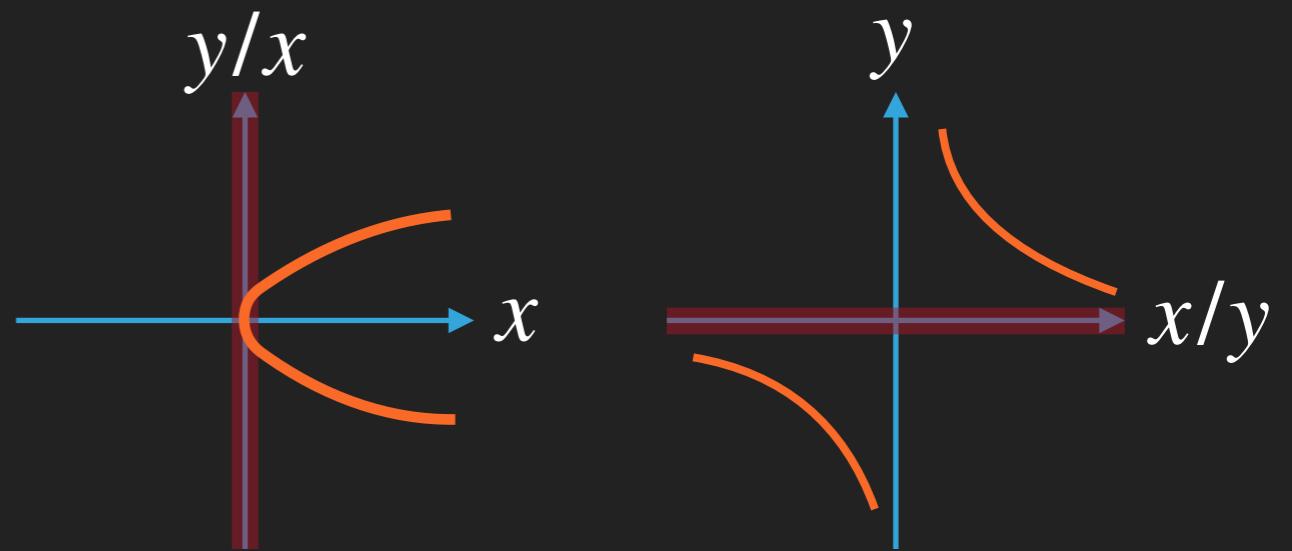


`affine_open[1]`

アルゴリズム 1-9

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`

`exceptional_list = [exc2, exc3, nss]`

`exc1`

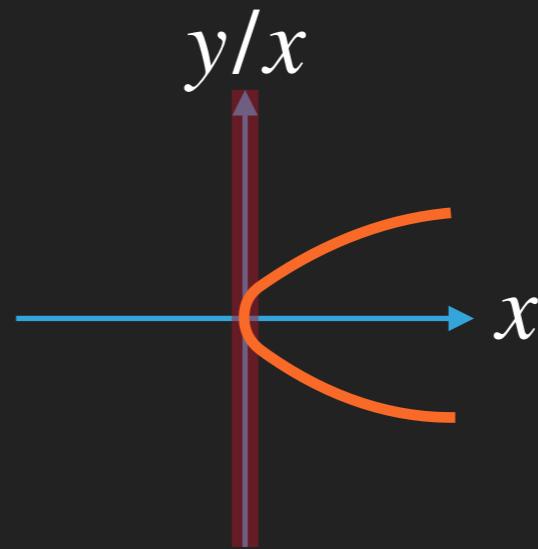
$(x/y)^3y - 1 = 0$:nonsingular

$y = 0$:no crossing

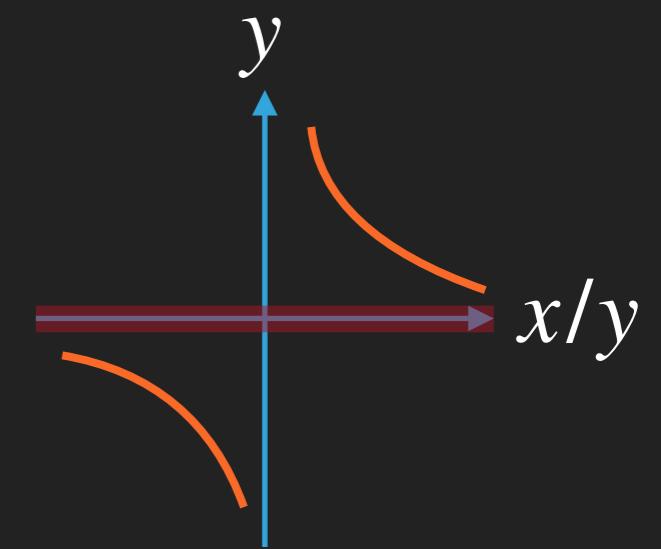
アルゴリズム 1-10

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



`affine_open[0]`



`affine_open[1]`

`exceptional_list = [exc2, exc3, nss]`

`exc1.set(affine_open[1], y)`

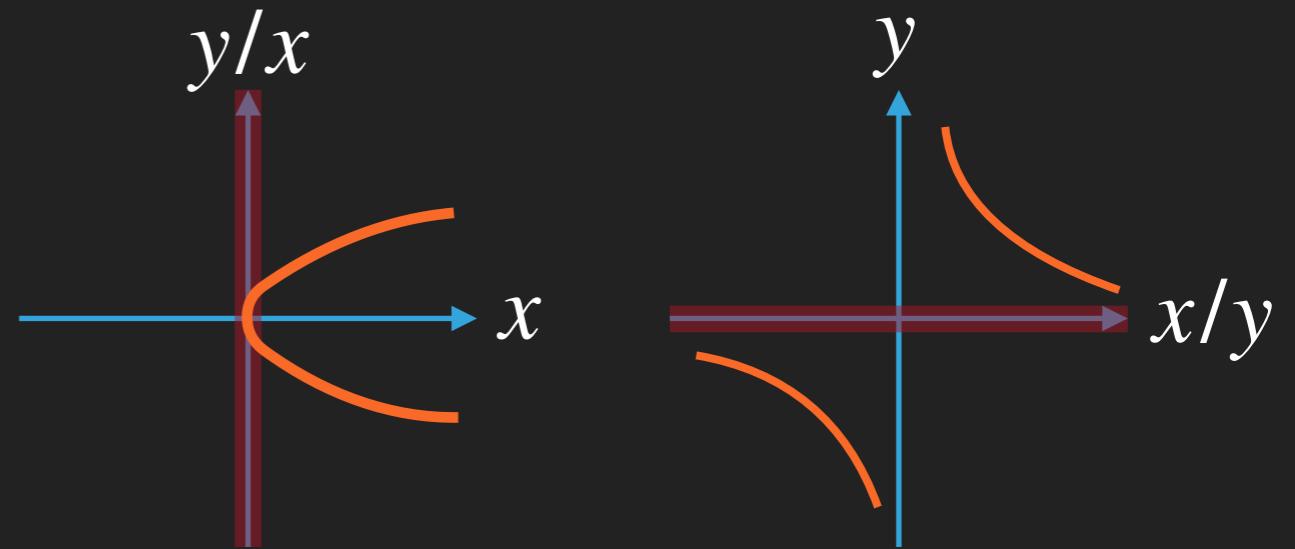
$(x/y)^3 y - 1 = 0$:nonsingular

$y = 0$:no crossing

アルゴリズム 1-11

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



```

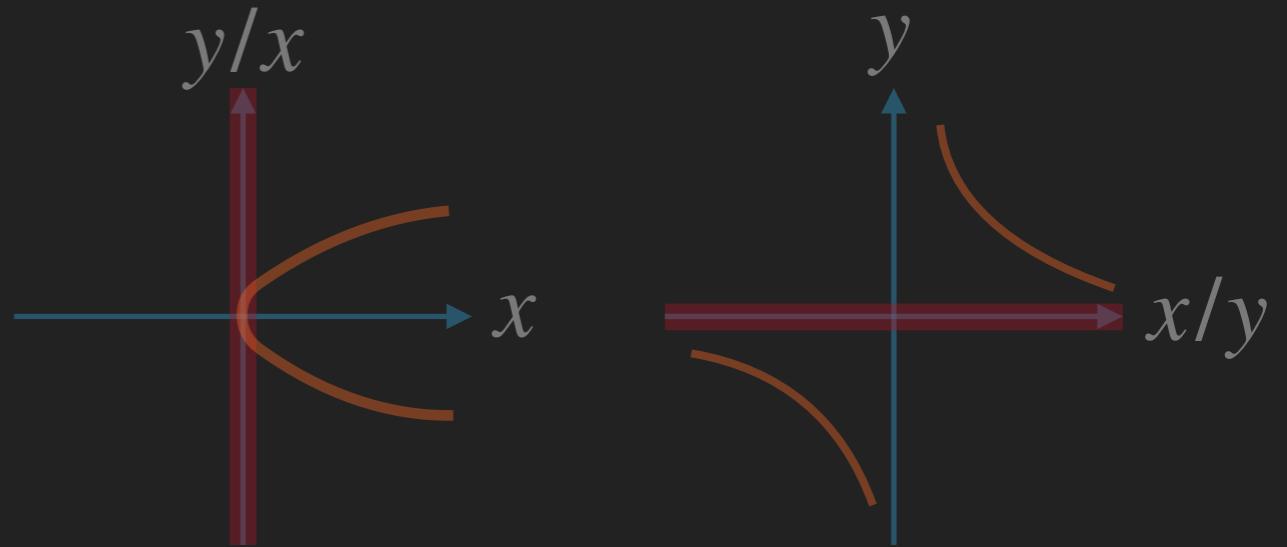
affine_open[0]           affine_open[1]
exceptional_list = [exc1, exc2, exc3, nss]
exc1.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y}]

```

アルゴリズム 1-11

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`オブジェクトを求める.



```

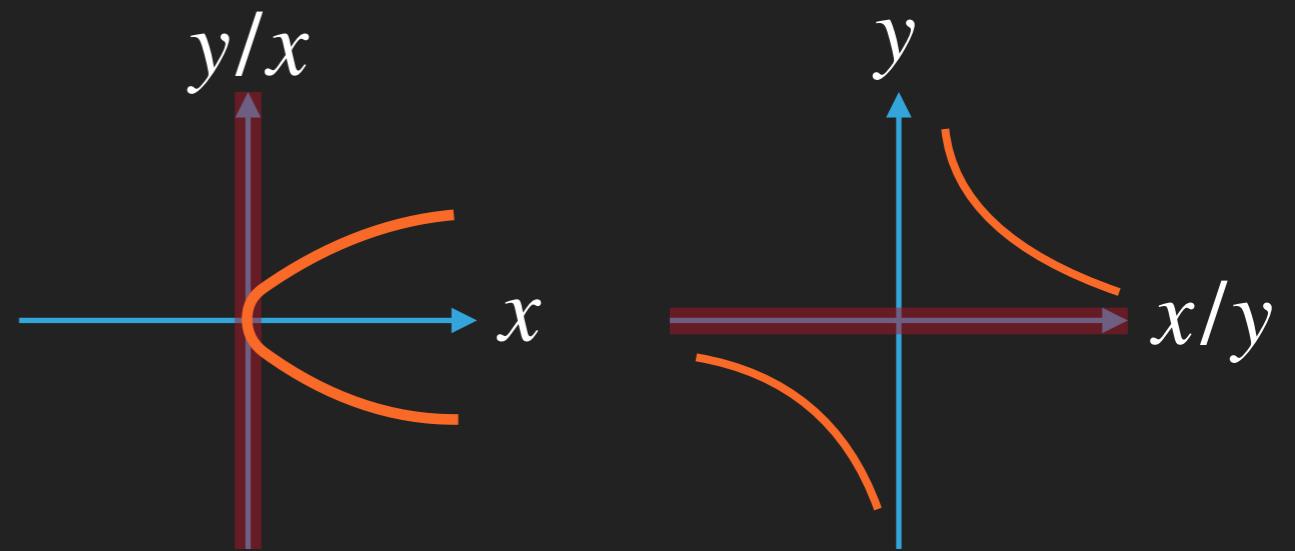
affine_open[0]           affine_open[1]
exceptional_list = [exc1, exc2, exc3, nss]
exc1.divisors =
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y}
 {'open': exc3.divisors[1]['open'], 'ideal': x}]
```

上の階層から返ってきた
リストの中にある例外曲線から情報を受け取り
貼りあった先での値を追加。

アルゴリズム 1-11

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる。
2. `ExceptionalCurve`インスタンスを入れるリストを宣言。
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める。
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく。
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す。
7. 再帰から返ってきたら`detach()`で切り離す。
8. 完成した`ExceptionalCurve`リストから双対グラフを求める。



`affine_open[0]`

`affine_open[1]`

`exceptional_list = [exc1, exc2, exc3, nss]`

`exc1.divisors =`

```
[{'open': affine_open[0], 'ideal': x},
 {'open': affine_open[1], 'ideal': y},
 {'open': exc3.divisors[1]['open'], 'ideal': x}]
```

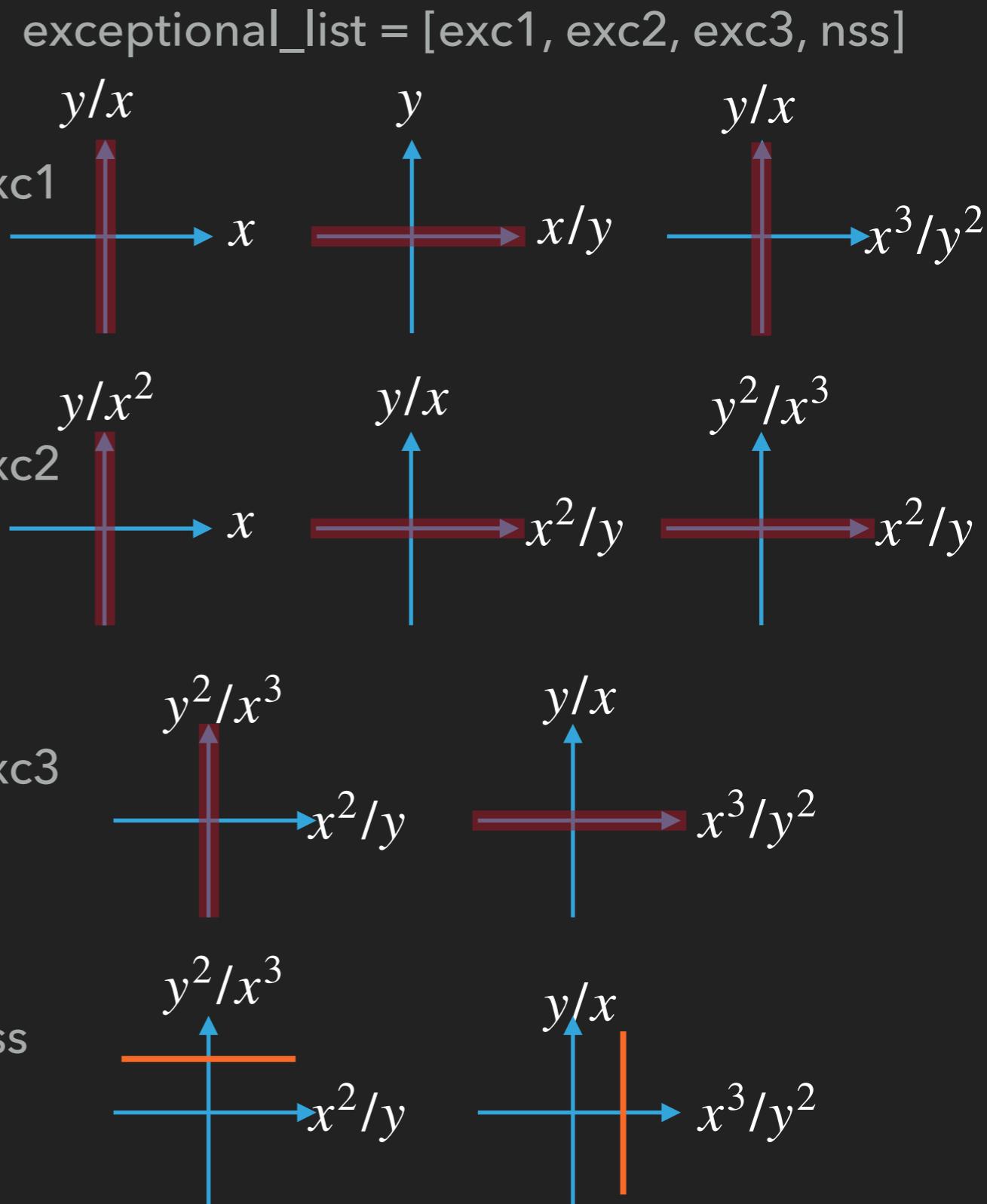
→呼び出し元へ

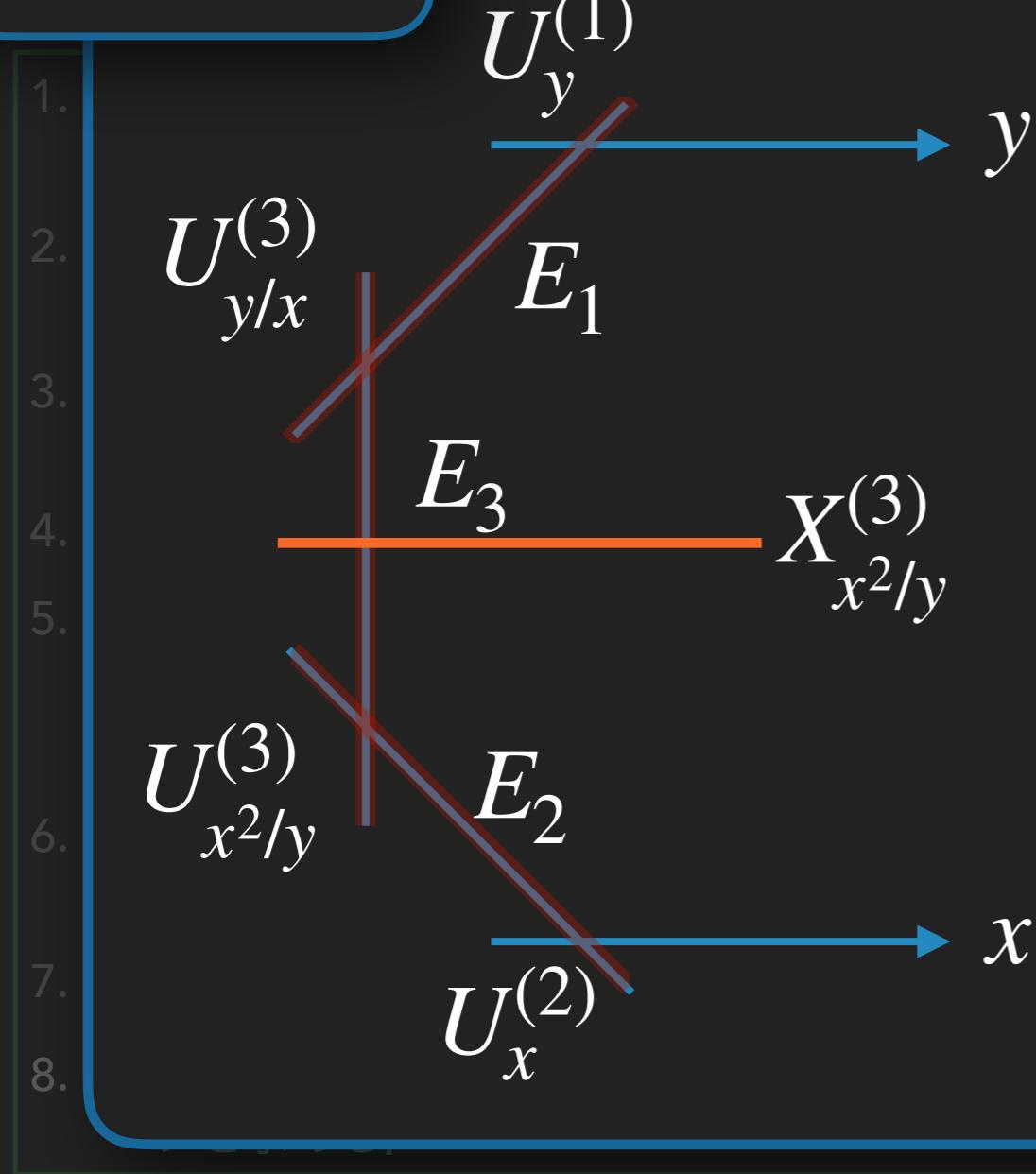
`exceptional_list` を返す

アルゴリズム 0-1

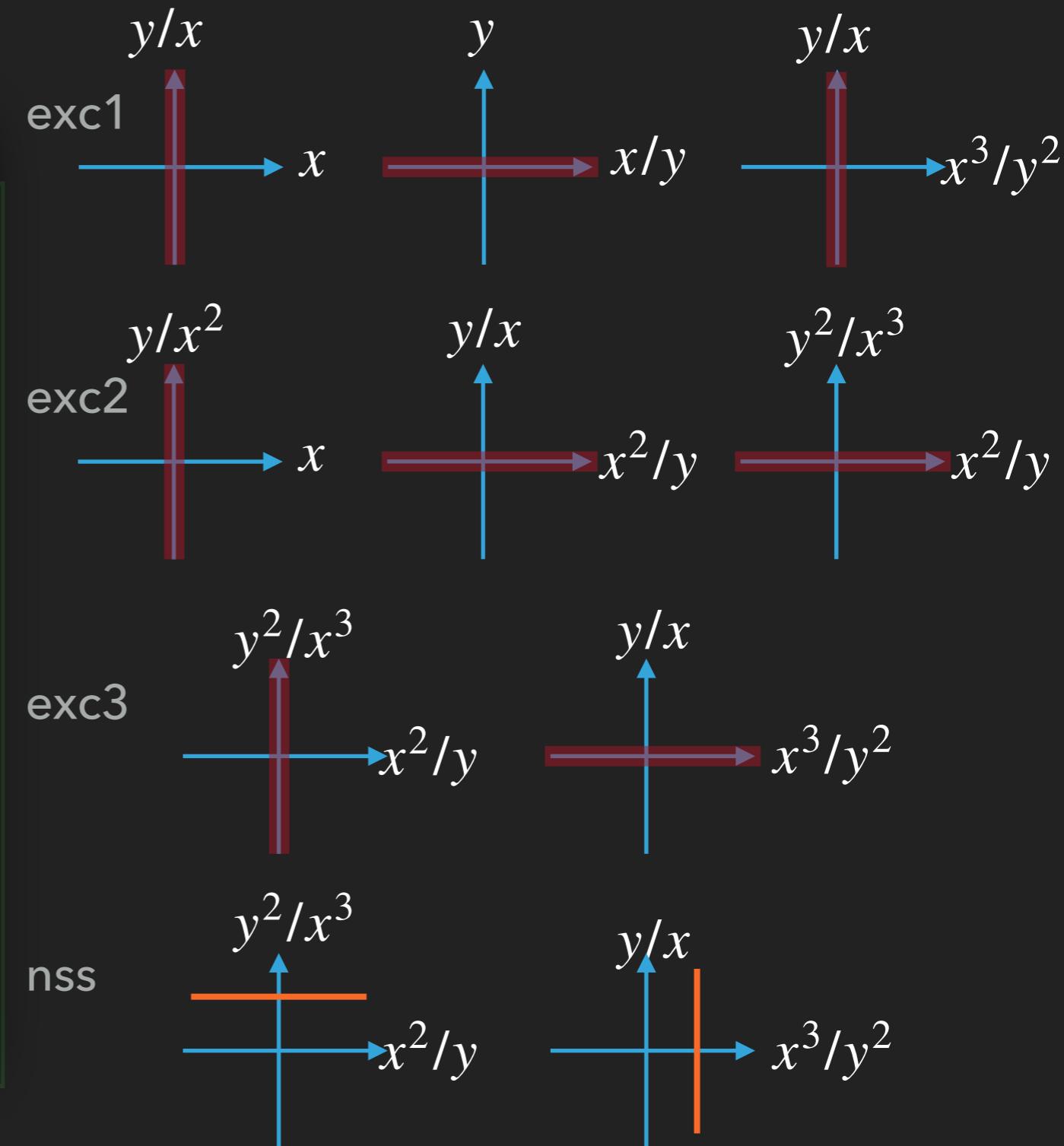
コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入しておく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



アルゴリズム 0-1
 模式図


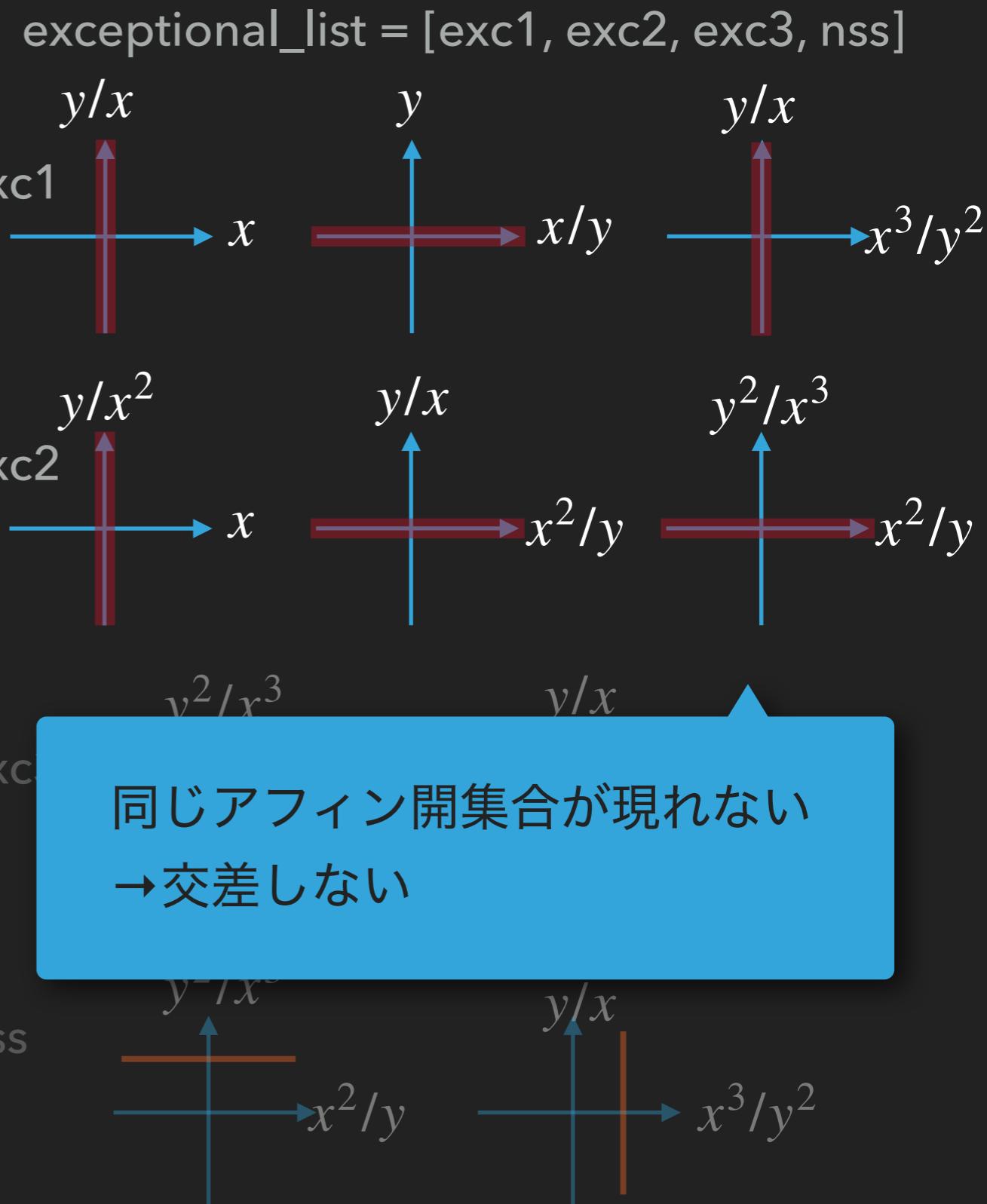
```
exceptional_list = [exc1, exc2, exc3, nss]
```



アルゴリズム 0-1

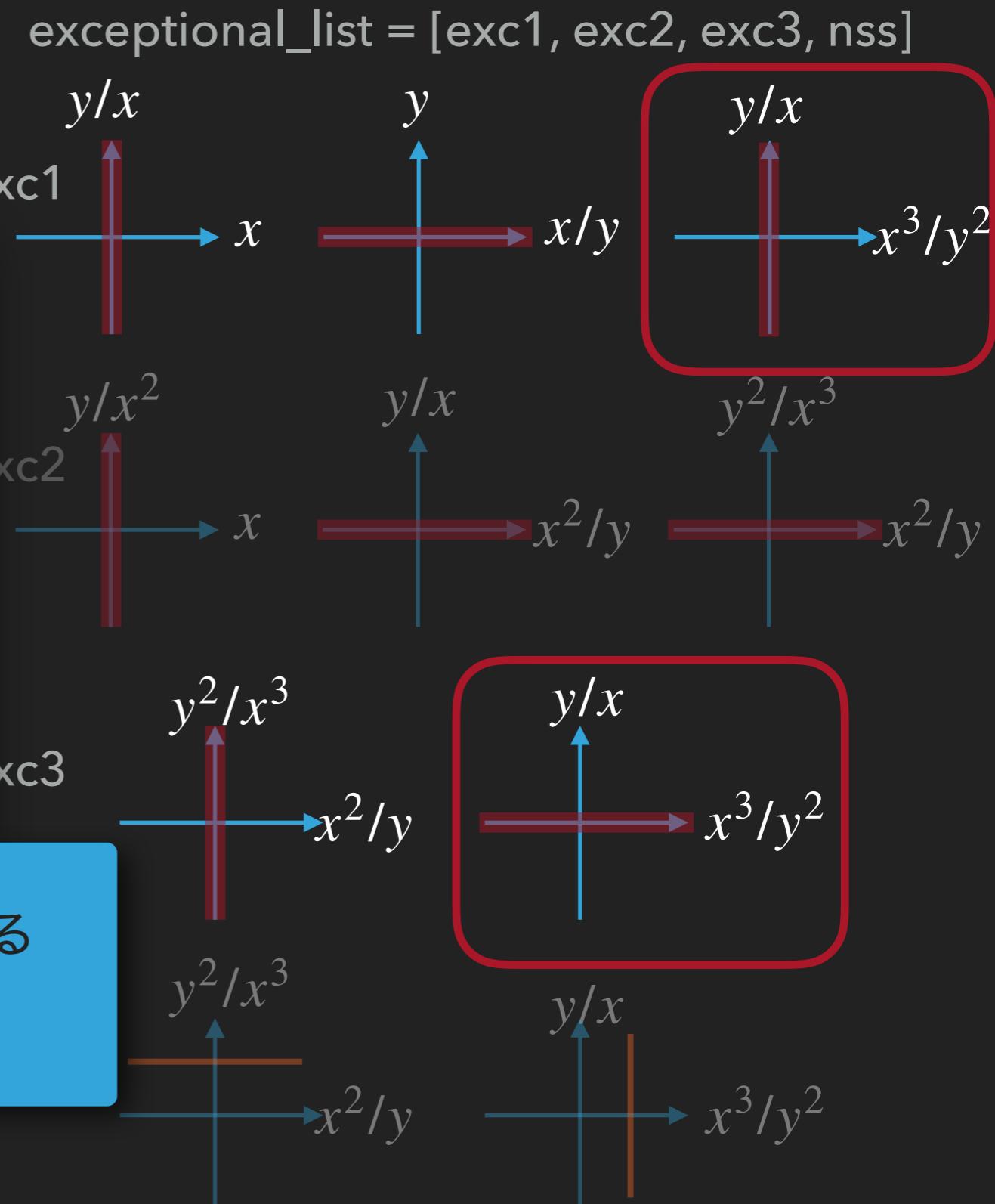
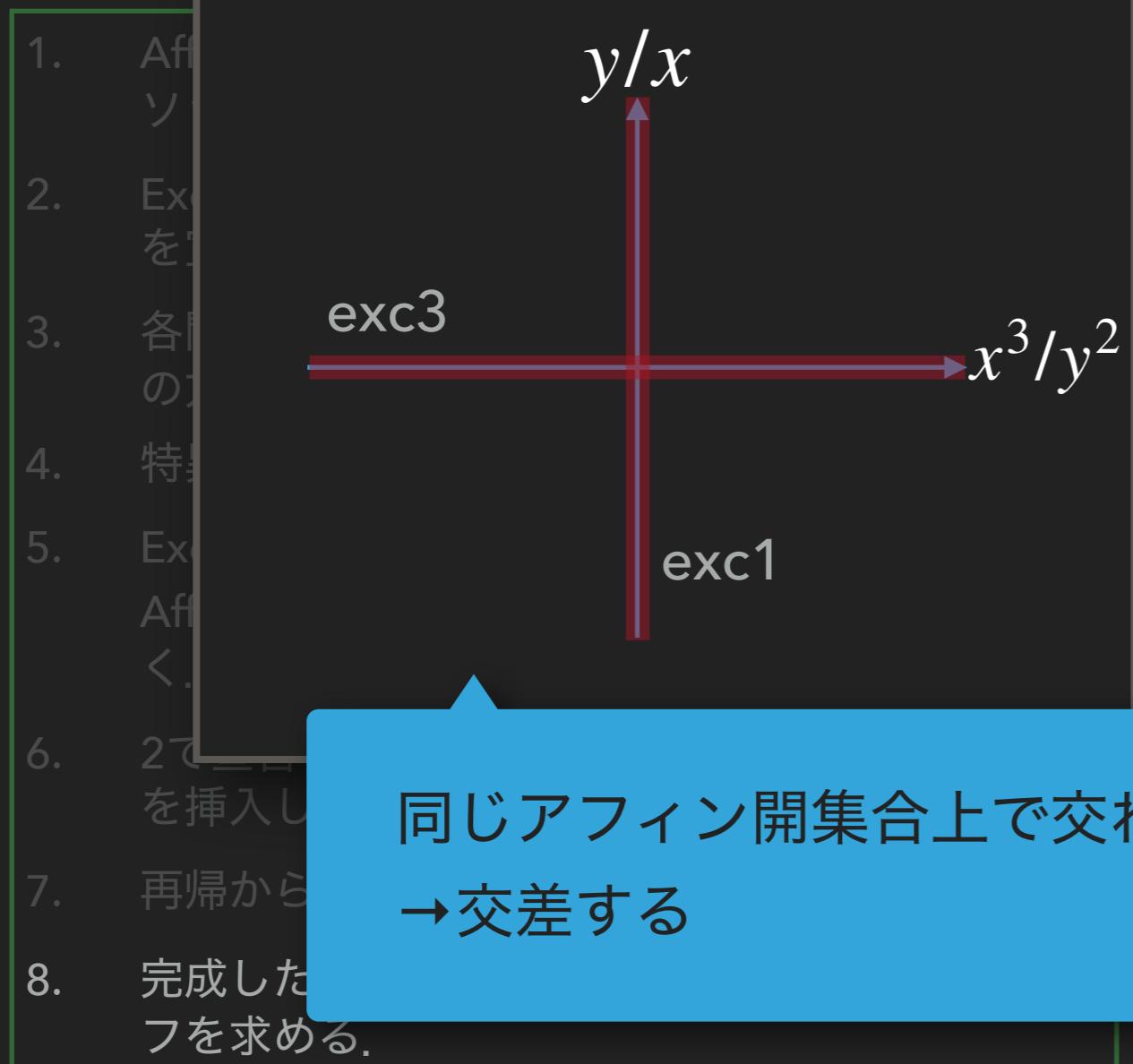
コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線のstrict transformと例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.



アルゴリズム 0-1

コンピュータアルゴリズム



アルゴリズム 0-1

コンピュータアルゴリズム

1. `AffineOpen`インスタンスを2つ生成し`glue()`メソッドで貼り合わせる.
2. `ExceptionalCurve`インスタンスを入れるリストを宣言.
3. 各開集合上で曲線の`strict transform`と例外曲線の方程式を求める.
4. 特異性の判定
5. `ExceptionalCurve`インスタンスを生成し,
`AffineOpen`インスタンスと方程式を代入していく.
6. 2で宣言したリストに5で生成したインスタンスを挿入し, そのリストを返す.
7. 再帰から返ってきたら`detach()`で切り離す.
8. 完成した`ExceptionalCurve`リストから双対グラフを求める.

`exceptional_list = [exc1, exc2, exc3, nss]`

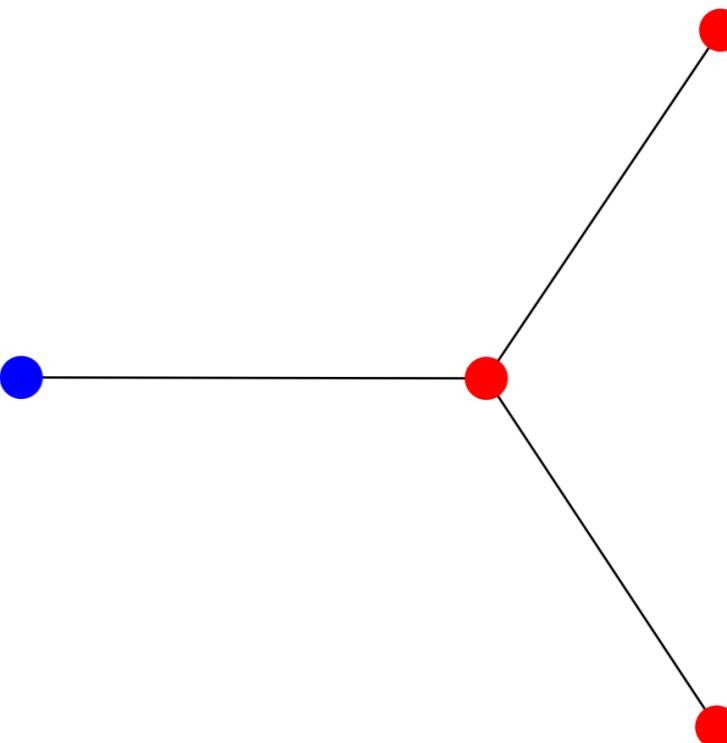


出力結果（再掲）

```

Sing V(f) = [{x: 0, y: 0}]
resolution of Curve Singularities
019 by dafuyafu
on AA(x, y/x):
    f_x = x - y**2 : nonsingular
    Exc: V(x): not normal crossing
    sympy.simplify(f_x)
    coordinate translation up as bu
    {x: 0, y: 0} --> {x: 0, y: 0}
    name f_x = x - y**2 :
    = symbols('x')
    = symbols('y')
    f_x = -x*y**2 + 1 : nonsingular
    = x ** -y ** 2
    Exc: V(x): no crossing
    f = x ** -y ** 2 # 3-cusp with co-tr
    f = on AA(x**2/y, y/x):# 19,25-cusp
    f = x ** f_y = x - y : nonsingular
    sing = soExc(V(y):f, not, normal crossing)
    print("f coordinate translation")
    print("Sing V(f) = {y: 0, x: 0} --> {x: 0, y: 0}")
    print("") f_y = x - y
    u.blowing_up(f)
    on AA(x**2/y, y**2/x**3):
        f_x = 1 - y : nonsingular
        Exc: V(x): normal crossing
    on AA(x**3/y**2, y/x):
        f_y = x - 1 : nonsingular
        Exc: V(y): normal crossing
    on AA(x/y, y):
        f_y = x**3*y - 1 : nonsingular
        Exc: V(y): no crossing
ExceptionalCurve({x, AA(x, y/x)}, {y, AA(x/y, y)}, {x, AA(x**3/y**2, y/x)})
ExceptionalCurve({x, AA(x, y/x**2)}, {y, AA(x**2/y, y/x)}, {y, AA(x**2/y, y**2/x**3)})
ExceptionalCurve({x, AA(x**2/y, y**2/x**3)}, {y, AA(x**3/y**2, y/x)})
NonsingularStrictTransform({1 - y, AA(x**2/y, y**2/x**3)}, {x - 1, AA(x**3/y**2, y/x)})

```

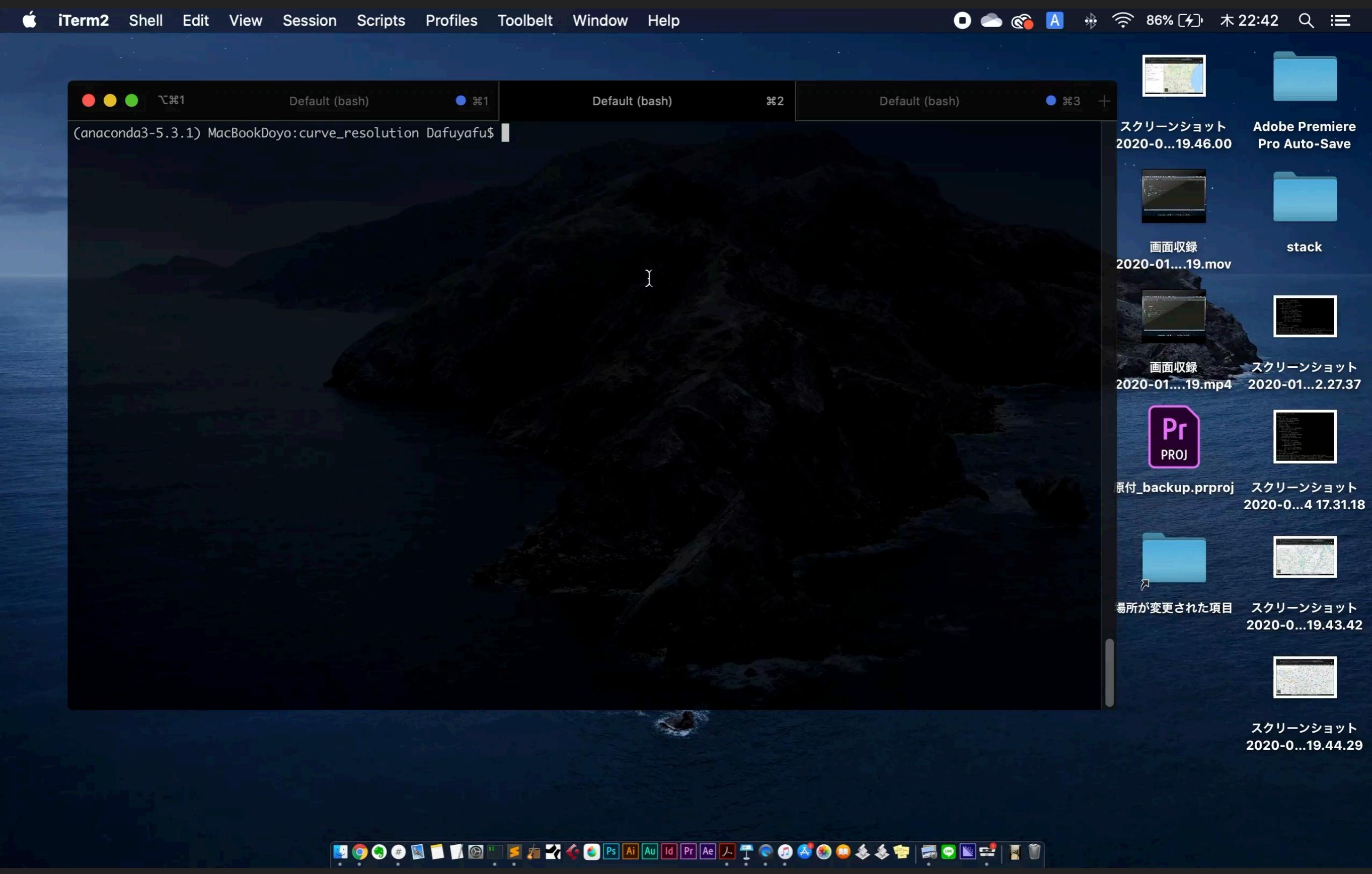


今後の展望

- ▶ 重み付き双対グラフの出力の実装
- ▶ より一般の曲線に対するブローアップのみによる特異点解消アルゴリズムを考える。
- ▶ 「特異点解消が求まるまでのブローアップの最小回数」は(特異点を持つ)代数多様体とどのような関係があるか？
- ▶ 参考文献リスト
 - [1] Hartshorne, R.: Algebraic Geometry. Springer-Verlag (1997)
 - [2] Eisenbud, D. Harris, J.: The geometry of schemes. Springer-Verlag (2000)
 - [3] Ishii, S.: Introduction to Singularities. Springer Japan (2014)
 - [4] 渡辺澄夫: 代数幾何と学習理論. 森北出版 (2006)
 - [5] 渡辺澄夫: ベイズ統計の理論と方法. コロナ社 (2012)

実行例

曲線 $C : x^{34} + y^{29} = 0$ の特異点解消



より一般の場合への対応

- ▶ 可約な場合

1回のブローアップで（既約な）例外曲線や狭義変換が複数発生する場合がある。

- ▶ 原点以外で特異点を持つ場合

適切な座標変換等の式の変換を行う必要がある

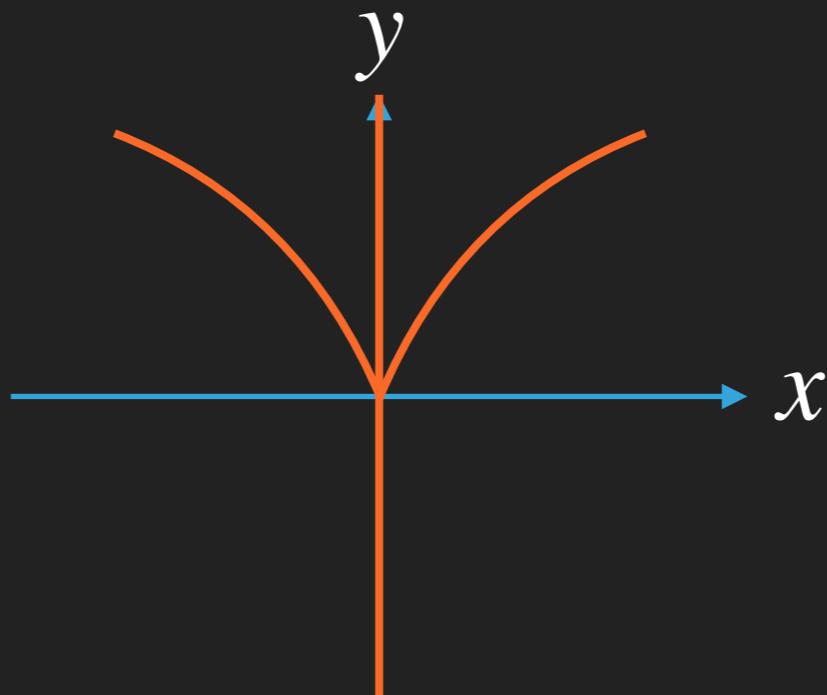
- ▶ 無理数などの取り扱い

(SymPyはある程度の無理数, 複素数を扱える)

可約な場合

▶ 例 : $x^3 + xy^3 = 0$

$x^3 + xy^3 = x(x^2 + y^3)$: 可約 (直線と曲線に分解される)



可約な場合

▶ 例 : $x^3 + xy^3 = 0$

$x^3 + xy^3 = x(x^2 + y^3)$: 可約 (直線と曲線に分解される)

$$\begin{aligned}x^3 + xy^3 &= y^5 \{(x/y^2)^3 y + (x/y^2)\} \\&= y^5(x/y^2) \{(x/y^2)^2 y + 1\} \\&= y[x_1 \{x_1^2 y_1 + 1\}] : \text{狭義変換が可約} \\&\quad (x_1 = 0, x_1^2 y_1 + 1 = 0)\end{aligned}$$

可約な場合

▶ 例 : $x^3 + xy^3 = 0$

$x^3 + xy^3 = x(x^2 + y^3)$: 可約 (直線と曲線に分解される)

$$x_1 = 0, x_1^2y_1 + 1 = 0$$

$y_1x_1 = 0$: normal crossing

$V(y_1, x_1^2y_1 + 1) = \emptyset$: no crossing

特異点解消が完了

$y_1 = 0, x_1^3y_1 + x_1 = 0$: not normal crossing

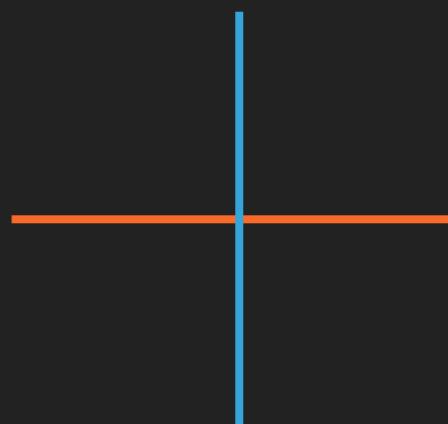
特異点解消が完了しない

可約な場合

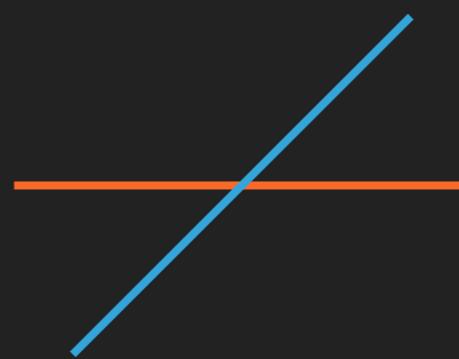
- ▶ 例 : $x^3 + xy^3 = 0$
 $x^3 + xy^3 = x(x^2 + y^3)$: 可約 (直線と曲線に分解される)
- ▶ 解決法:
 1. 例外曲線を既約分解してから
個別にExceptionalCurveインスタンスを生成する,
 2. ExceptionalCurveインスタンスに
自身を既約分解するメソッドを付け加える.
- ▶ 注) 貼りあった先の開集合での値を保持するにはどうすれば良いか?

正規交差性

- ▶ 正規交差 (normal crossing) ... 横断的に交差する状態



正規交差



正規交差でない

正規交差性

- ▶ 平均誤差関数 $K(w) \geq 0$ に特異点解消定理を適用すると

$$K(g(u)) = u^{2k}$$

とできる。また、対数尤度比関数 $f(x, g(u))$ が u の解析関数であるとする。このとき、ある解析関数 $a(x, u)$ が存在して

$$f(x, g(u)) = u^k a(x, u)$$

が成り立つ。([5] 4.2.2 補題20)

正規交差だから・・・