Computation of Bipath PH using Julia

2025/02/21

全体像

全体的な関数
(1)Bipathposets.jl (Moduleの定義)
(2)FSCvector.jl(行列の操作をまとめてる)

 A 型の分解
(3)AtypeMethod.jl (A 型の分解)
(4)FSC_Contraction_Persistence.jl (区間圧縮)

前回の話

Matrix Method
(5)BipathMatMethod.jl
(6)Print_functions.jl

可視化
(7)BipathPD.jl

例作成
(8)clique.jl

今回の話

＊体は$\mathbb{F}_2$.
IsFSC.jl(エラー発見用
もういらないので削除)

(1)Bipathposets.jl

♦Moduleの定義

```julia
module Bipathposets
include("IsFSC.jl")
include("FSCvector.jl")
include("FSC_Contraction_Persistence.jl")
include("AtypeMethod.jl")
include("Print_functions.jl")
include("BipathMatMethod.jl")
include("clique.jl")
include("BipathPD.jl")
end # module bipathposets
```

(2)FSCvector.jl

♦簡単な行列操作、単体複体のベクトル化などまとめている

(2.1)standardbasis(dim, k)

k^{dim} の標準基底を作る.

(2.2)vectorizationofSC(FSC, sumofcomplex)

右図参照

(2.3)boundary_operation(complex)

$\sigma_4 = [1,2] \mapsto [\sigma_1 = [1], \sigma_2 = [2]]$

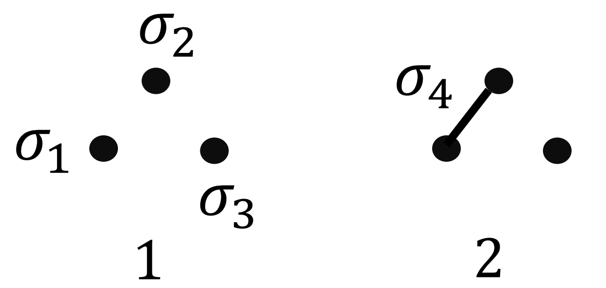
(2.4)find_lowest_nonzero(vect)

略

(2.5)reducematfromLtoR(matrix)

略

(2.2)の例.　長さ 2 のFSC

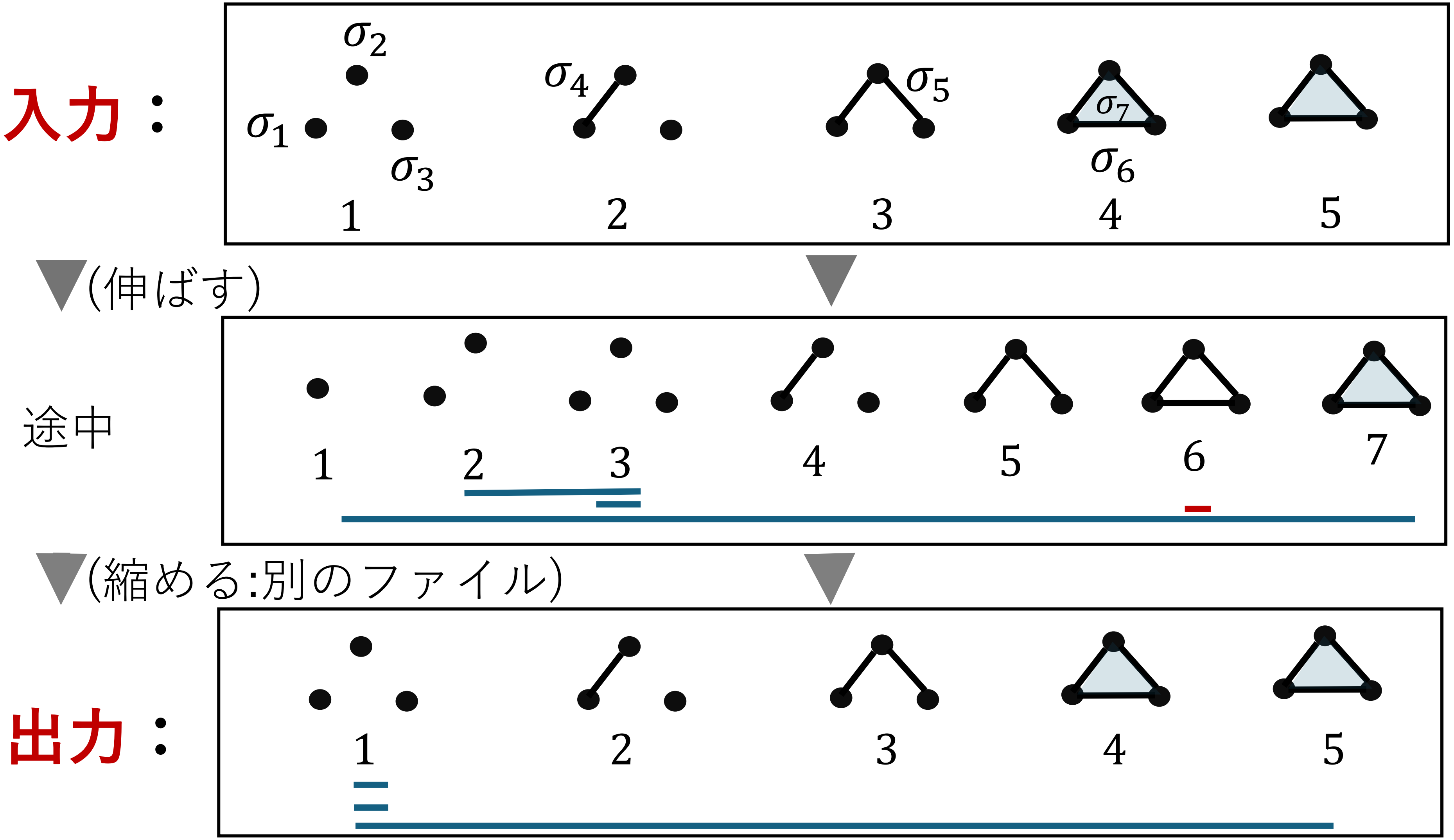


$\mathrm{FSC} = [\ [[\sigma_1, 1], [\sigma_2, 1], [\sigma_3, 1], [\sigma_2, 2]], 2]$

$$\sigma_1 \mapsto \begin{matrix}1\\0\\0\\0\end{matrix} \quad \sigma_2 \mapsto \begin{matrix}0\\1\\0\\0\end{matrix} \quad \sigma_3 \mapsto \begin{matrix}0\\0\\1\\0\end{matrix} \quad \sigma_4 \mapsto \begin{matrix}0\\0\\0\\1\end{matrix}$$

単体をベクトル化する.

(3)AtypeMethod.jl



(3)AtypeMethod.jl

参考図書：位相的データ解析から構造発見へ：
パーシステントホモロジーを中心に

◆A型のFSCの区間分解(with basis)を与える.

(3.1)getB(FSC)
(3.2)getBhat_and_basechange(BB)
(3.3)get_intervals_with_basis(FSC,Bhat,info_col_change)
(3.4)baseswithintervals(FS(FSC_Contraction_Persistence.jl

境界準同型を求める
「左から右」掃き出し法

$$\mathrm{FSC} \xrightarrow{(3.1)} B \xrightarrow{(3.2)} \hat{B} \xrightarrow{(3.3)} \text{Intervals} + \text{basis} + \text{boundary image at } \hat{1} \xrightarrow{(3.4)} \text{Intervals (contracted)} + \text{basis} + \text{boundary image at } \hat{1} + \text{boundary image at } \hat{0}$$

＋ 操作の記録

(Matrix problemの構成で必要)

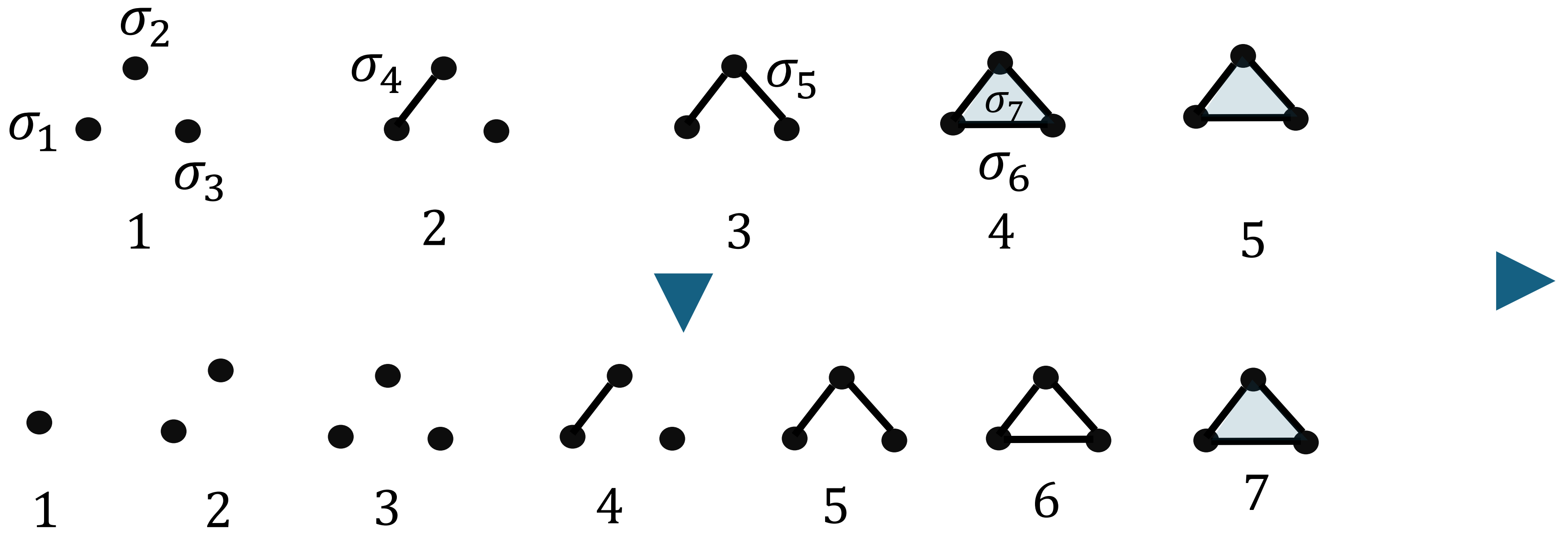
(3)AtypeMethod.jl

(3.1)getB(FSC):

♦ FSC=>境界準同型の行列表現(基底$\{\sigma_1, \dots, \sigma_N\}$による)を構成

*FSC= $[[[\sigma_1, b_1], \dots, [\sigma_N, b_N]], n]$　n: filtration の長さ

Ex. $[[[\sigma_1, 1], [\sigma_2, 1], [\sigma_3, 1], [\sigma_4, 2], [\sigma_5, 3], [\sigma_6, 4], [\sigma_7, 4]], 5]$



$$B = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(3)AtypeMethod.jl

(3.2)getBhat_and_basechange(B):

♦境界準同型の簡約化：境界準同型(B)=>[Bhat(行列), col_change(リスト)]

Bhat: Bを「左から右への掃き出し法」によって簡約されて得られる行列。

col_change: ふたつの整数のペア$[i, j](i < j)$ からなるリスト。
「左から右への掃き出し法」で行われた列の変換を記録

$$
B = \begin{array}{c} \begin{array}{ccccccc} & & & \sigma_4 & \sigma_5 & \sigma_6 & \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}
\quad \blacktriangleright \quad
\hat{B} = \begin{array}{c} \begin{array}{c} \qquad\qquad\qquad \sigma_4 + \sigma_5 + \sigma_6 \end{array} \\ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}
$$

col_change=[[5,6],[4,6]]

＊区間[a, ∞)に付随する基底
を求めるとき必要
(この情報はmatrix methodで必要)

(3)AtypeMethod.jl

(3.3)get_intervals_with_basis:
♦簡約化された行列から区間と付随する基底を得る
(FSC,Bhat,info_col) => [pairs(辞書),imagebasis(リスト)]

pairs:(区間に付随する)基底と区間の対応を与える辞書

imagebasis: 区間($\hat{1}$を含まない)とそれに付随する基底の対応を与えるリスト.

$$\hat{B} = \begin{array}{c c} & \begin{array}{ccccccc} & & & 4 & 5 & & 7 \end{array} \\ \begin{array}{c} \\ 2 \\ 3 \\ \\ \\ 6 \\ \\ \end{array} & \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

col_change=[[5,6],[4,6]]

pairs(辞書)

$[2,4) = [2,3] \leftrightarrow \sigma_1 + \sigma_2$
$[3,5) = [3,4] \leftrightarrow \sigma_3 + \sigma_2$
$[6,7) = [6,6] \leftrightarrow \sigma_4 + \sigma_5 + \sigma_6$

$[1,\infty) = [1,7] \leftrightarrow \sigma_1$

imagebasis(リスト)

$[\ [[\sigma_1 + \sigma_2], [2,3]],$
$[[\sigma_3 + \sigma_2],[3,4]],$
$[[\sigma_4 + \sigma_5 + \sigma_6],[6,6]\ ]$

($H(S)_{\hat{1}} = Z_{\hat{1}}/B_{\hat{1}}$
の$B_{\hat{1}}$のbasis elements)

(この情報はmatrix methodで必要)

(3)AtypeMethod.jl

◆FSCの(A型の)区間分解 with basis (赤い本参照).

(3.4)baseswithintervals(FSC)

◆(3,1)--(3,3)を用いて、FSCから以下を得る.

FSC->[pairs(辞書), imagebasisleft, imagebasis]

・pairs: 辞書、区間と生成元の対応(区間の長さが修正されている)

・imagebasisleft: boundary image at $\hat{0}$ 生成元のリスト. [](この例では空集合)

・imagebasis: boundary image at $\hat{1}$の生成元のリスト. $[[\sigma_1+\sigma_2],[\sigma_3+\sigma_2],[\sigma_4+\sigma_5+\sigma_6]]$

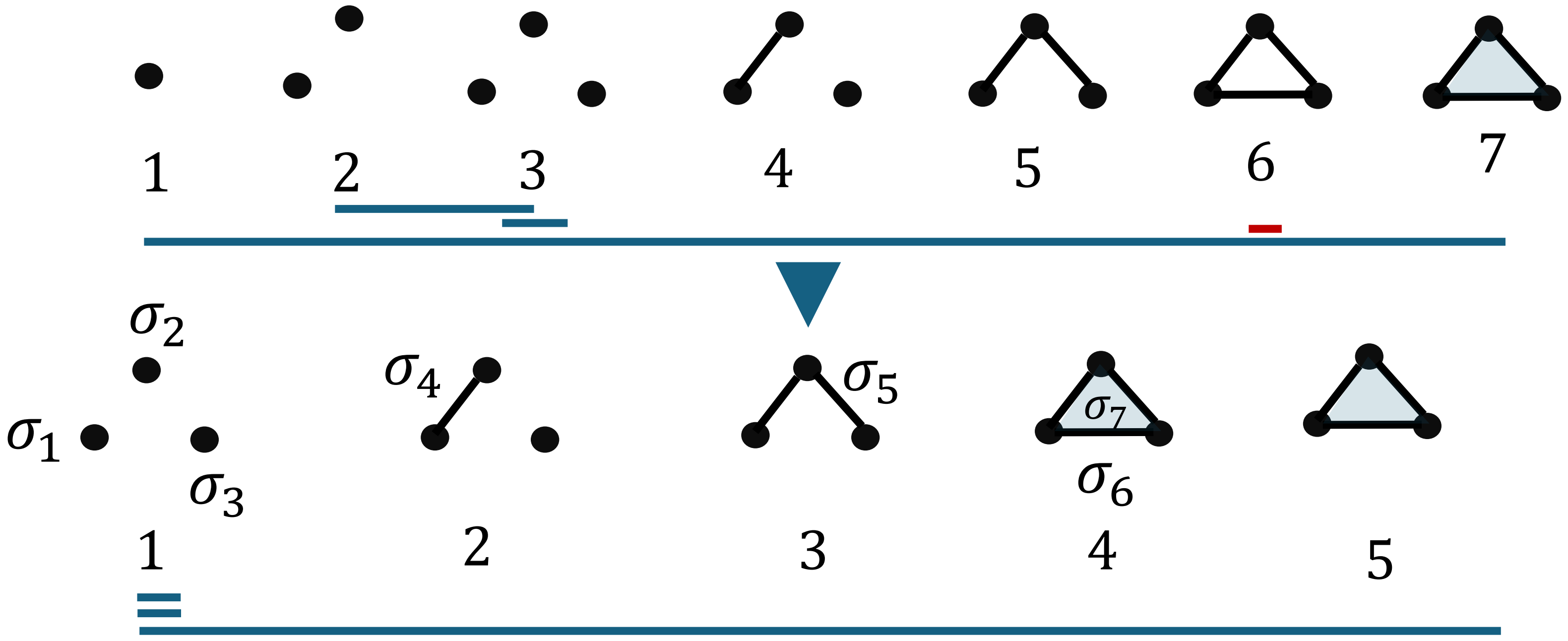
[2,3] ←→ $\sigma_1+\sigma_2$
[3,4] ←→ $\sigma_3+\sigma_2$
[6,6] ←→ $\sigma_4+\sigma_5+\sigma_6$

[1,7] ←→ σ_1

▼ FSC_Contraction_Persistence.jl

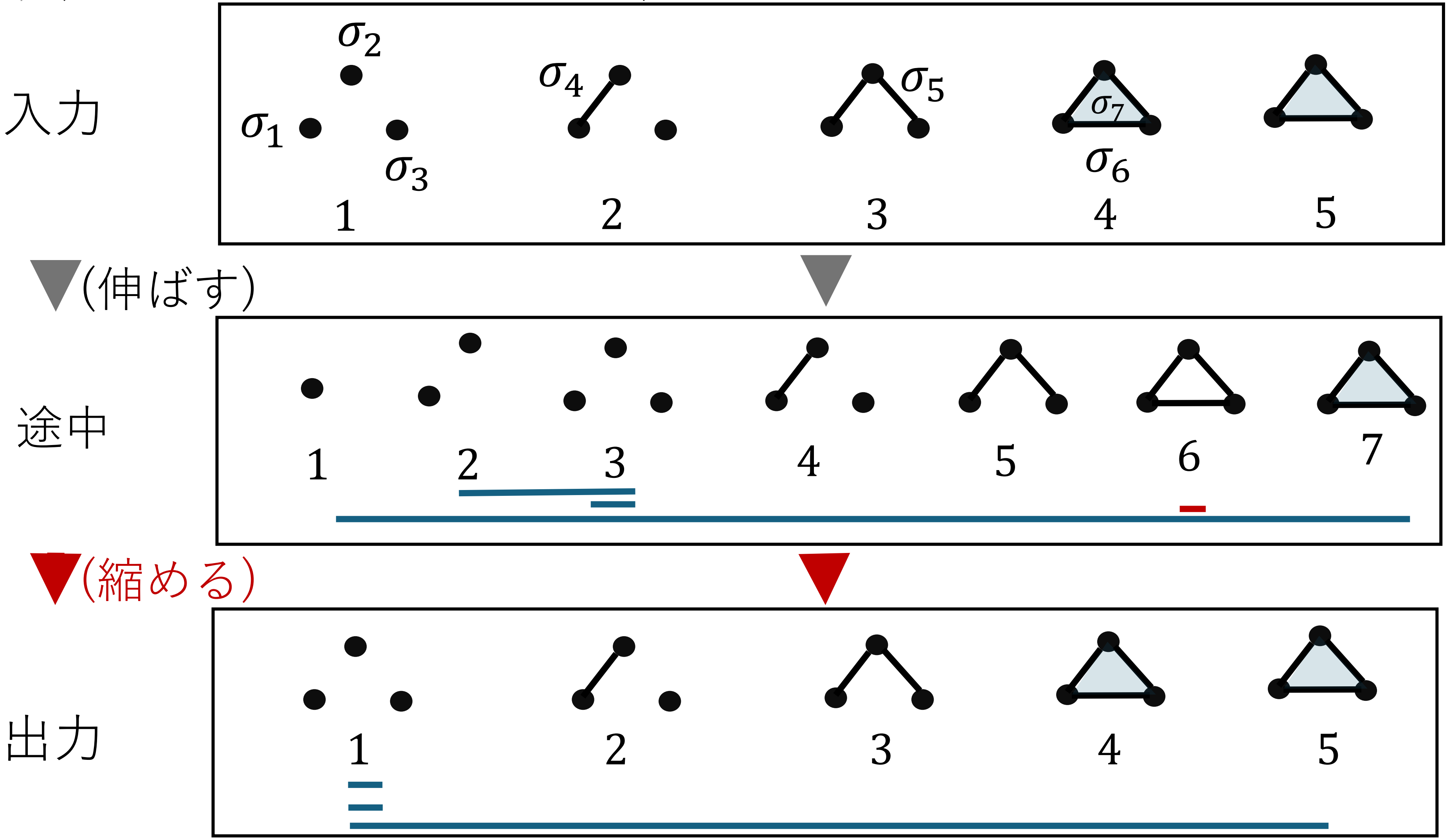
[1,1] ←→ $\sigma_1+\sigma_2$
[1,1] ←→ $\sigma_3+\sigma_2$

[1,5] ←→ σ_1



(4)FSC_Contraction_Persistence.jl

♦(区間分解アルゴリズムで)伸びた区間を縮め、本来の区間を得る.



(4)FSC_Contraction_Persistence.jl

♦(区間分解アルゴリズムで)伸びた区間を縮め、本来の区間を得る.

(4.1) contractbirth(birth, FSC)

(4.2) contractdeath(death, FSC)

(4.3) contractinterval(I=[birth,death], FSC)

(4.1)birth(整数)を縮める,
(4.2)death(整数)を縮める,
それらを合わせた
(4.3)区間$I = [b, d]$のbとdを(1)と(2)
を用いて縮め、新たな区間を得る.

(4)FSC_Contraction_Persistence.jl

(4.1)contractbirth(birth(整数), FSC)

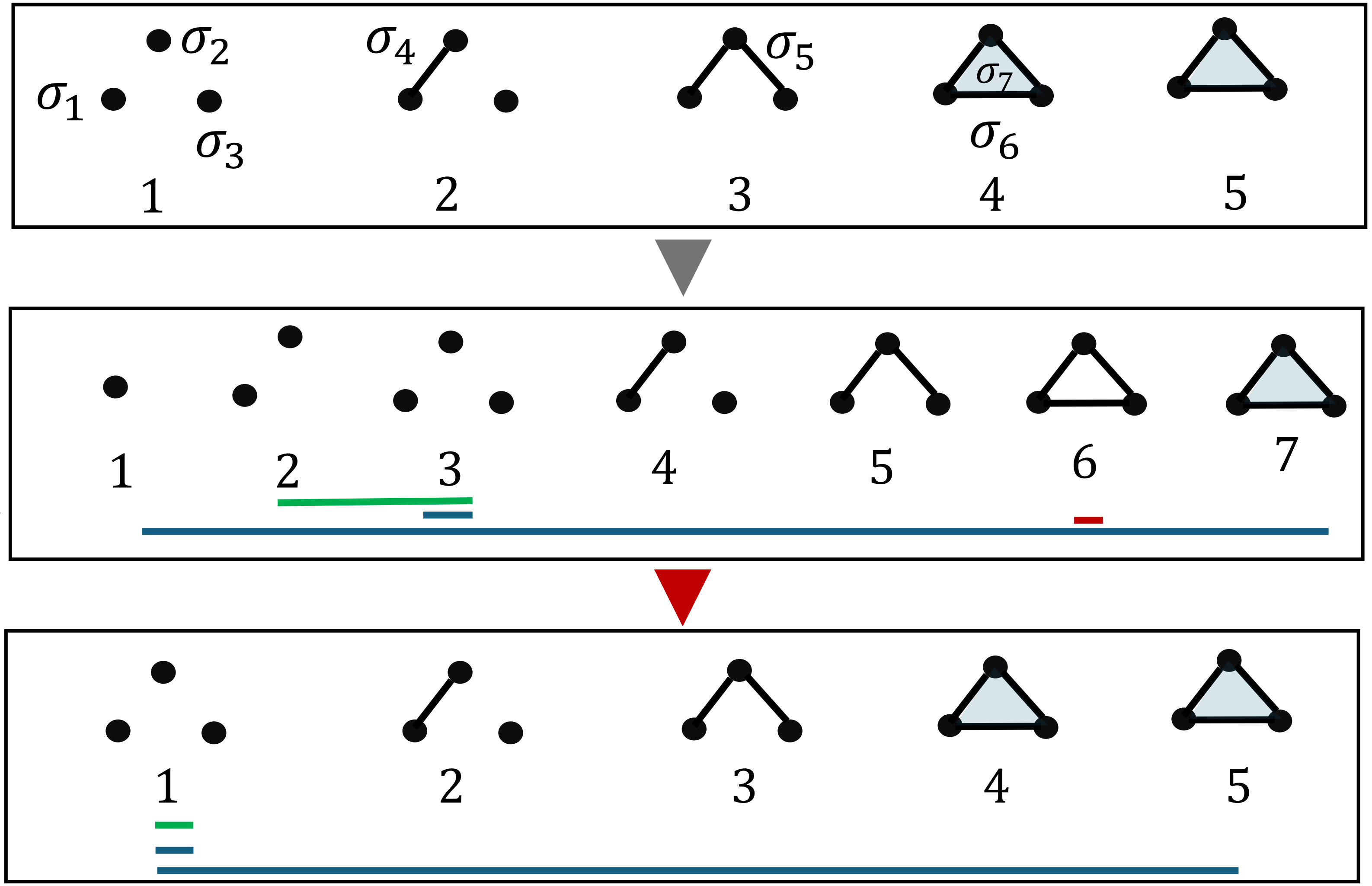
出力: FSC[1]のbirth番目の単体の誕生日(FSC[1][birth][2])（＊）

（＊）ある圧縮前の区間I=[birth, death]のbirthは、本来の区間に圧縮したとき、birth番目の単体の誕生日に移る。

Ex (1)右図において、 birth = 2 のとき、上の関数は 1 を出力する(σ_2の誕生日は1)。

Ex (2).右図において、birth = 6のとき、上の関数は4を出力する(σ_6の誕生日は4)。

$$FSC[1] = [[\sigma_1, 1], [\sigma_2, 1], [\sigma_3, 1], [\sigma_4, 2], [\sigma_5, 3], [\sigma_6, 4], [\sigma_7, 4]]$$



(4)FSC_Contraction_Persistence.jl

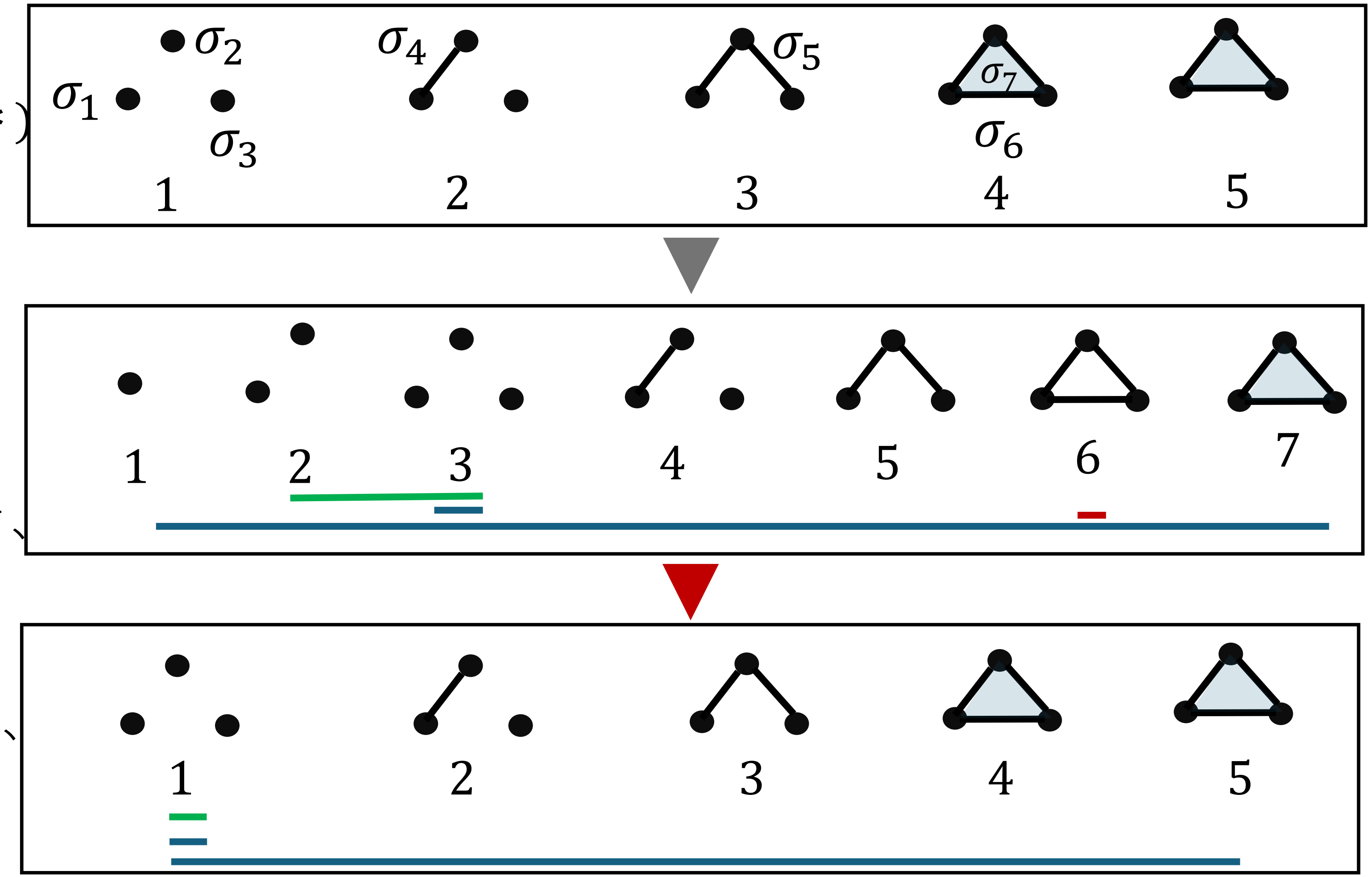
(4.2) contractdeath(**death**（整数）, **FSC**)
If **death** = length(**FSC[1]**):単体の個数
FSC[2]本来のfilt.の長さ)を出力.
else
FSC[1][death + 1][2] − 1 を出力(＊)

（＊）ある圧縮前の区間I=[birth, death]のdeathは、本来の区間に圧縮したとき、death+1番目の単体の誕生日-1に移る。

Ex (1)右図において、**death = 3** のとき、上の関数は１を出力する($\sigma_{\mathrm{death}+1}$の誕生日−1は1)。

Ex (2).右図において **death = 6** のとき、上の関数は3を出力する($\sigma_{\mathrm{death}+1}$の誕生日−1は3)。

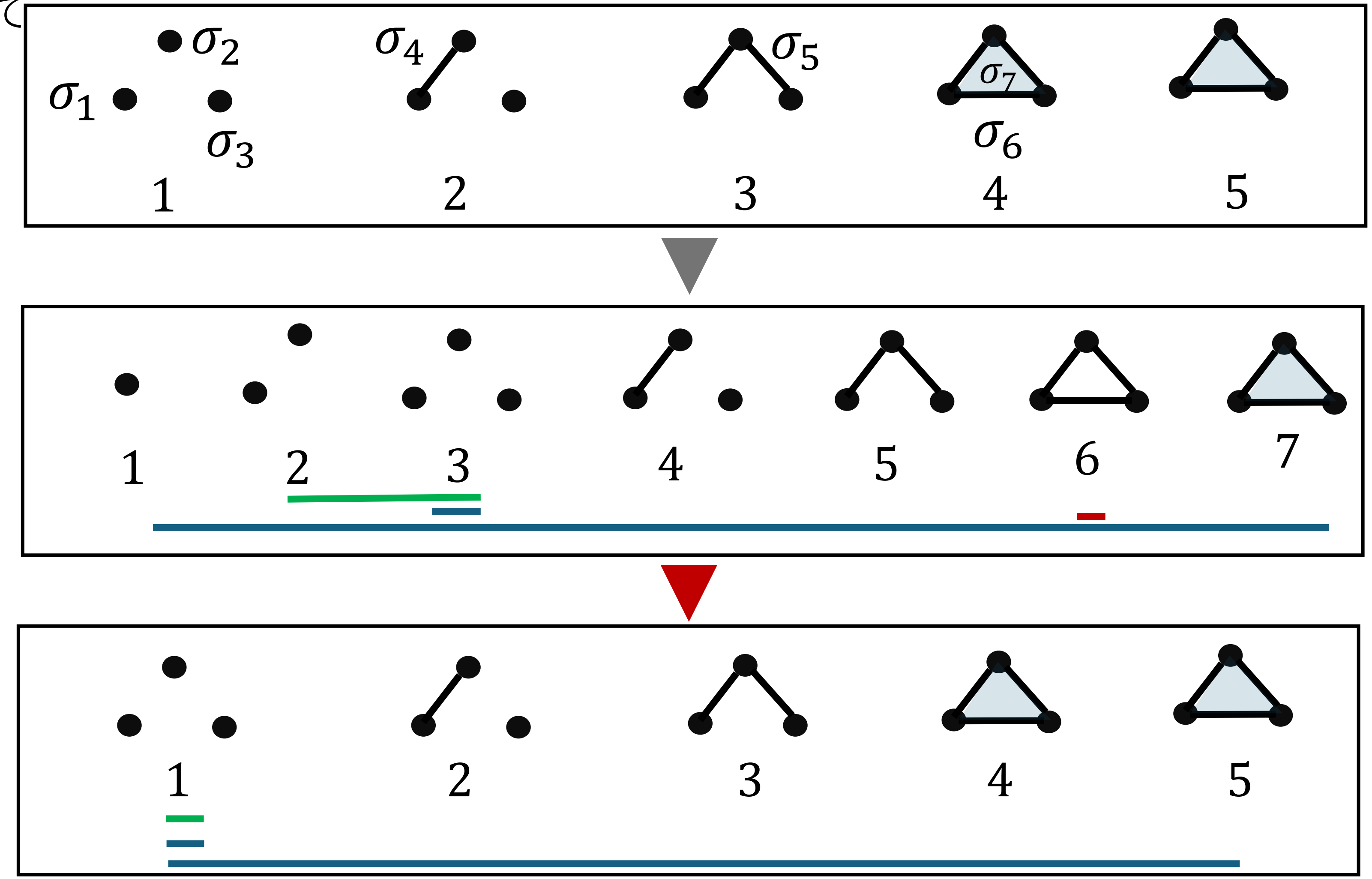
$$\mathrm{FSC}[1] = [[\sigma_1, 1], [\sigma_2, 1], [\sigma_3, 1], [\sigma_4, 2], [\sigma_5, 3], [\sigma_6, 4], [\sigma_7, 4]]$$



(4)FSC_Contraction_Persistence.jl

(4.3) contractinterval([birth, death], FSC)
b: = contractbirth(birth, FSC)
d: = contractdeath(death, FSC)に対して
$[b, d]$を出力

$\mathrm{FSC}[1] = [[\sigma_1, 1], [\sigma_2, 1], [\sigma_3, 1], [\sigma_4, 2], [\sigma_5, 3], [\sigma_6, 4], [\sigma_7, 4]]$



Ex (1). $I = [2,3]$は$[1,1]$に送られる

Ex (2). $I = [6,6]$は$[4,3]$に送られる.
(この関数で得られる以下のような組

$$[a, b](a > b)$$

は元々のフィルトレーションではその穴が無かったことを意味する。
このような組は区間分解した後に処理する)

(5)BipathMatMethod.jl

AtypeMatMethod.jl

Bipath filtration => Matrix Problem => 区間分解

(5.1), (5.2), (5.3), (5.4)

(5.5)

♦Matrix Problemの準備

(5.1)separateintervals(pairs, k)
(5.2)make_space(FSC_vect::Dict, int_basis)

♦Matrix Problemの構成・簡約化

(5.3)get_two_repmat(spaceUp, spaceDown)
(5.4)connect_updown(upintervalswithB, downintervalswithB, matrix)
(下からの上,左から右,掃き出し法)

♦区間の出力

(5.5)interval_decomposition

$$
\begin{array}{c}
\\
[1,1] \\
\vdots \\
[1,n-1] \\
[1,n] \\
[2,n] \\
\vdots \\
[n,n]
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
[1,1] & \ldots & [1,n-1] & [1,n] & [2,n] & \ldots & [n,n]
\end{array} \\
\left[\begin{array}{ccccccc}
* & \cdots & * & * & & & \\
\vdots & \ddots & \vdots & \vdots & & & \\
* & \cdots & * & * & & & \\
0 & \cdots & 0 & * & * & \cdots & * \\
 & & & 0 & * & \cdots & * \\
 & & & \vdots & \vdots & \ddots & \vdots \\
 & & & 0 & * & \cdots & *
\end{array}\right]
\end{array}
$$

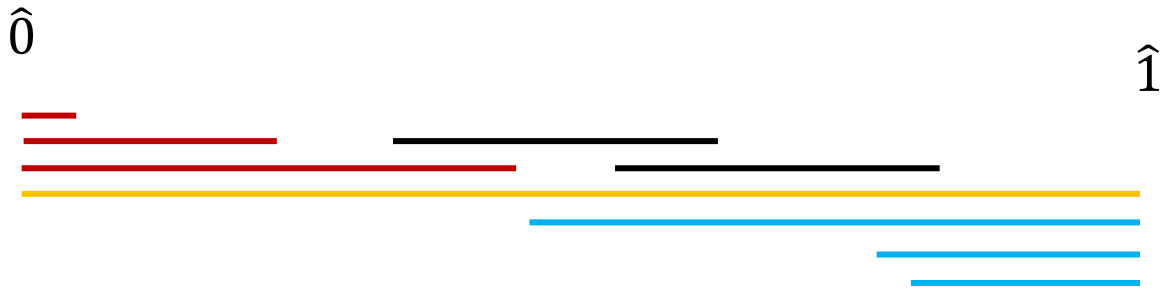
(5)BipathMatMethod.jl

◆Matrix Problemの準備

(5.1)separateintervals(pairs, k)

pairs(辞書):区間とその区間に付随する生成元の対応
k: filtrationの長さ

区間を4つのタイプに分ける

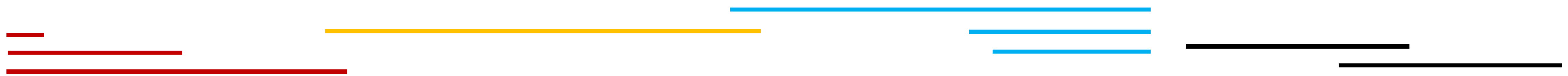


・最小を含み、最大を含まない(**L**)
・最少と最大を含む(**B**)
・最小を含まず最大を含む(**R**)
・それ以外(**U** and **D**).

それぞれのタイプの区間と(区間に付随する)生成元のペアからなるリストを出力

＊ AtypeMethod.jlの関数によってタイプ**L**と**R**の区間は“整列”されている。

[leftintervals,leftbasis], [center,centerbasis],[rightintervals,rightbasis], [others,othersbasis]



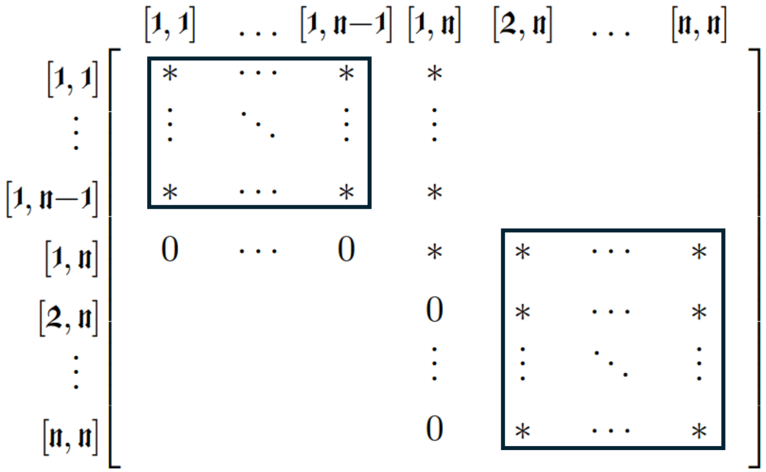
(5)BipathMatMethod.jl

◆Matrix Problemの準備

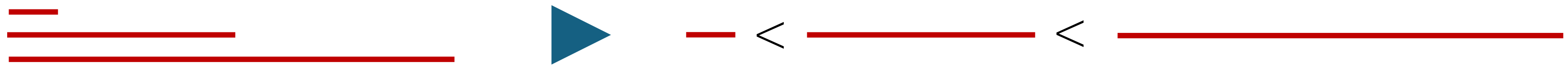
(5.2)make_space(FSC_vect::Dict, int_basis)

FSC_vect: FSCの単体をベクトル化したもの。
int_basis: 区間に付随するホモロジー類の生成元(鎖複体の元たち)。

int_basisをベクトル化し、それらをひとつの行列して出力する。



Ex. intbasis=[v_1 , v_2 , v_3]



$\hat{0}$

(既に整列されている)

▶ v_1 < v_2 < v_3

各v_iは単体の形式和(単体のリスト)

▶ 行列[$v_1 v_2\ v_3$] を出力(size= 単体数×区間の数)

ベクトル化

(5)BipathMatMethod.jl

◆Matrix Problemの構成

(5.3)get_repmat(spaceUp, spaceDown)　spaceUp, spaceDownは行列

連立方程式を解き、行列 $\pi_\ell \Lambda \iota_\ell$ と $\pi_\ell \Gamma \iota_\ell$ を求めるための関数。

LspaceUp　　RspaceUp

LspaceDown

RspaceDown

$$
\begin{array}{c}
 \\ [1,1] \\ \vdots \\ [1,n-1] \\ [1,n] \\ [2,n] \\ \vdots \\ [n,n]
\end{array}
\begin{array}{c}
\begin{array}{ccccccc} [1,1] & \dots & [1,n-1] & [1,n] & [2,n] & \dots & [n,n] \end{array} \\
\left[\begin{array}{ccccccc}
* & \cdots & * & * & & & \\
\vdots & \ddots & \vdots & \vdots & & & \\
* & \cdots & * & * & & & \\
0 & \cdots & 0 & * & * & \cdots & * \\
 & & & 0 & * & \cdots & * \\
 & & & \vdots & \vdots & \ddots & \vdots \\
 & & & 0 & * & \cdots & *
\end{array}\right]
\end{array}
$$

get_repmat(LspaceUp, LspaceDown)

$$
= \pi_\ell \Lambda \iota_\ell = \begin{bmatrix} * & \cdots & * \\ \vdots & \ddots & \vdots \\ * & \cdots & * \end{bmatrix}
$$

get_repmat(RspaceUp, RspaceDown)

$$
= \pi_\ell \Gamma \iota_\ell = \begin{bmatrix} * & \cdots & * \\ \vdots & \ddots & \vdots \\ * & \cdots & * \end{bmatrix}
$$

(5)BipathMatMethod.jl

◆Matrix Problemの簡約化(＋上下の区間の接続(with区間に付随する基底))

(5.4)connect_updown(upintervalswithB, downintervalswithB, matrix)　*正則行列$\pi_\ell \Lambda \iota_\ell$と $\pi_\ell \Gamma \iota_\ell$に適用

upintervalswithB: 区間のリストと,その区間たちに付随する生成元のリストの組.

$$[\,[I_1, I_2, I_3, I_4], [\sigma_1, \sigma_2, \sigma_3, \sigma_4]\,]$$

＊生成元の情報は「区間が何次のホモロジーの情報を持つか」を知るために必要！

downintervalswithB: 区間のリストと,その区間たちに付随する生成元のリストの組.

$$[\,[J_1, J_2, J_3, J_4\,], [\tau_1, \tau_2, \tau_3, \tau_4]\,]$$

Ex.

$$\begin{array}{cc} & \begin{array}{cccc} \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 \\ I_1 & I_2 & I_3 & I_4 \end{array} \\ \begin{array}{c} \tau_1\ J_1 \\ \tau_2\ J_2 \\ \tau_3\ J_3 \\ \tau_4\ J_4 \end{array} & \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \end{array} \quad \blacktriangleright \quad \begin{array}{cc} & \begin{array}{cccc} \sigma_1 & \sigma_2 & \sigma_3 & \sigma_4 \\ I_1 & I_2 & I_3 & I_4 \end{array} \\ \begin{array}{c} \tau_1\ J_1 \\ \tau_2\ J_2 \\ \tau_3\ J_3 \\ \tau_4\ J_4 \end{array} & \begin{bmatrix} 0 & 0 & 0 & \mathbf{1} \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 \\ \mathbf{1} & 0 & 0 & 0 \end{bmatrix} \end{array} \quad \blacktriangleright$$

[
$[[I_1, \sigma_1], [J_4, \tau_4]], [[I_2, \sigma_2\,], [J_2, \tau_2\,]],$
$[[I_3, \sigma_3], [J_3, \tau_3]], [[I_4, \sigma_4], [J_1, \tau_1]]$
]

入力　左から右 下から上の操作　置換行列　区間を繋げる　**出力**

(5)BipathMatMethod.jl

♦区間の出力(これまでの関数を用いて区間分解をし、出力)

(5.5)interval_decomposition(FSCa,FSCb)

入力: FSCa, FSCb (bipath filtration)
出力: 辞書 $i \mapsto \mathcal{B}(H_i(S))$ (ith bipath PHの区間分解), FSCaの長さ, FSCbの長さ

この関数はこれまでの関数を組み合わせたもの。

入力 –> A型区間分解×2 –> 行列を構成 –> 簡約化 –> 区間分解 –> 次数で区間を分類 –> 出力

FSCa

FSCb

入力

(5)BipathMatMethod.jl

◆区間の出力(これまでの関数を用いて区間分解をし、出力)

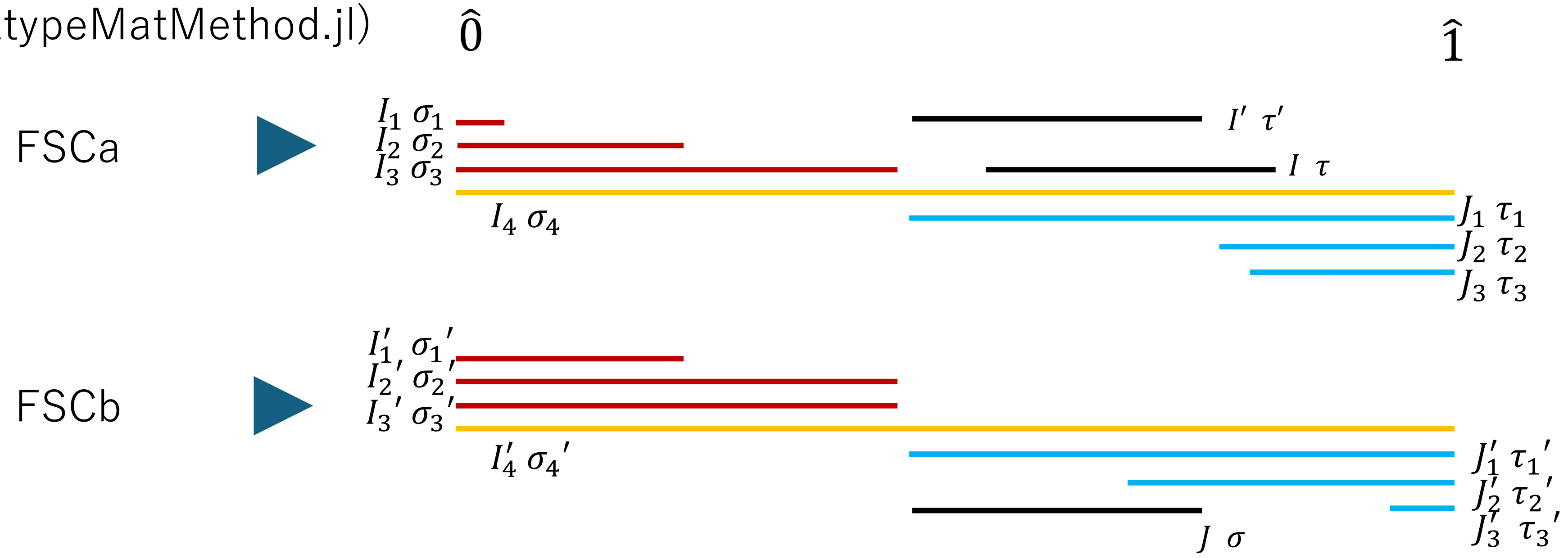
(5.5)interval_decomposition(FSCa,FSCb)

入力: FSCa, FSCb (bipath filtration)
出力: 辞書 $i \mapsto \mathcal{B}(H_i(S))$ (ith bipath PHの区間分解), FSCaの長さ, FSCbの長さ

この関数はこれまでの関数を組み合わせたもの。

入力 –> **A型区間分解×2** –> 行列を構成 –> 簡約化 –> 区間分解 –> 次数で区間を分類 –> 出力

(AtypeMatMethod.jl)



(5)BipathMatMethod.jl

◆区間の出力(これまでの関数を用いて区間分解をし、出力)

(5.5)interval_decomposition(FSCa,FSCb)

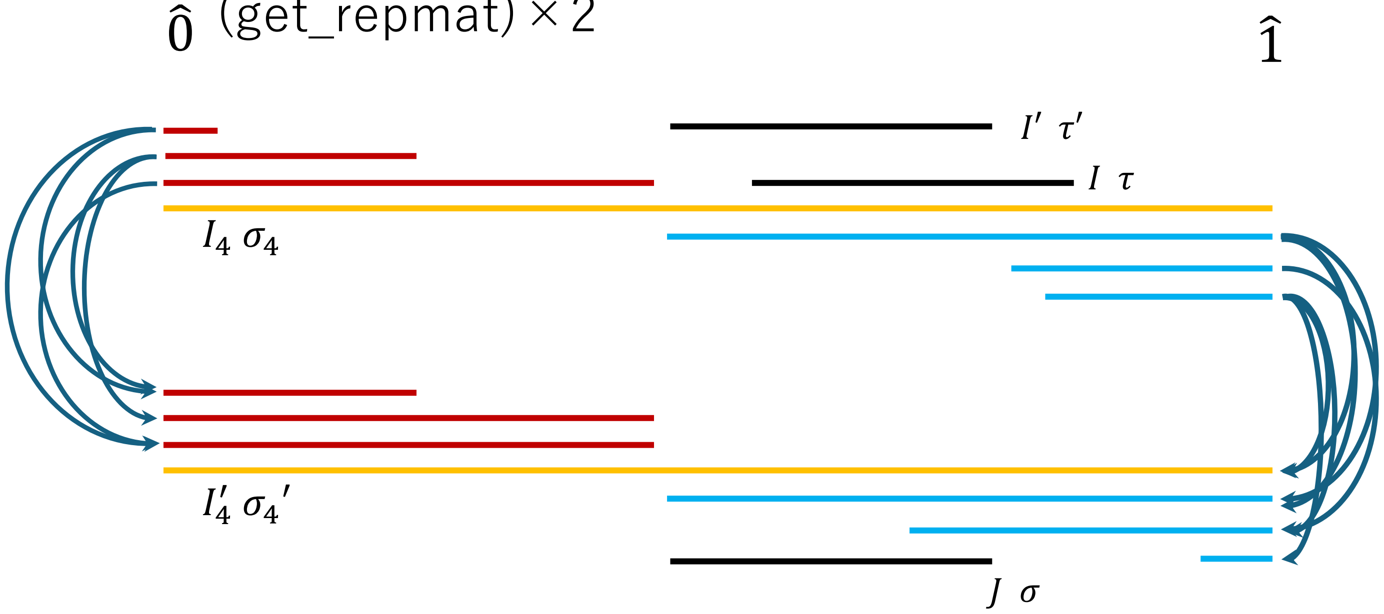
入力: FSCa, FSCb (bipath filtration)
出力: 辞書 $i \mapsto \mathcal{B}(H_i(S))$ (ith bipath PHの区間分解), FSCaの長さ, FSCbの長さ

この関数はこれまでの関数を組み合わせたもの。

入力 –> A型区間分解×2 –> **行列を構成** –> 簡約化 –> 区間分解 –> 次数で区間を分類 –> 出力

(get_repmat)×2

$$\begin{array}{ccc} & & \begin{array}{ccc}\sigma_1 & \sigma_2 & \sigma_3 \\ I_1 & I_2 & I_3\end{array} \\ \begin{array}{c}\sigma_1' \\ \sigma_2' \\ \sigma_3'\end{array} & \begin{array}{c} I_1' \\ I_2' \\ I_3'\end{array} & \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}\end{array}$$



$$\begin{array}{ccc} & & \begin{array}{ccc}\tau_1 & \tau_2 & \tau_3 \\ J_1 & J_2 & J_3\end{array} \\ \begin{array}{c}\tau_1' \\ \tau_2' \\ \tau_3'\end{array} & \begin{array}{c} J_1' \\ J_2' \\ J_3'\end{array} & \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix}\end{array}$$

(5)BipathMatMethod.jl

♦区間の出力(これまでの関数を用いて区間分解をし、出力)

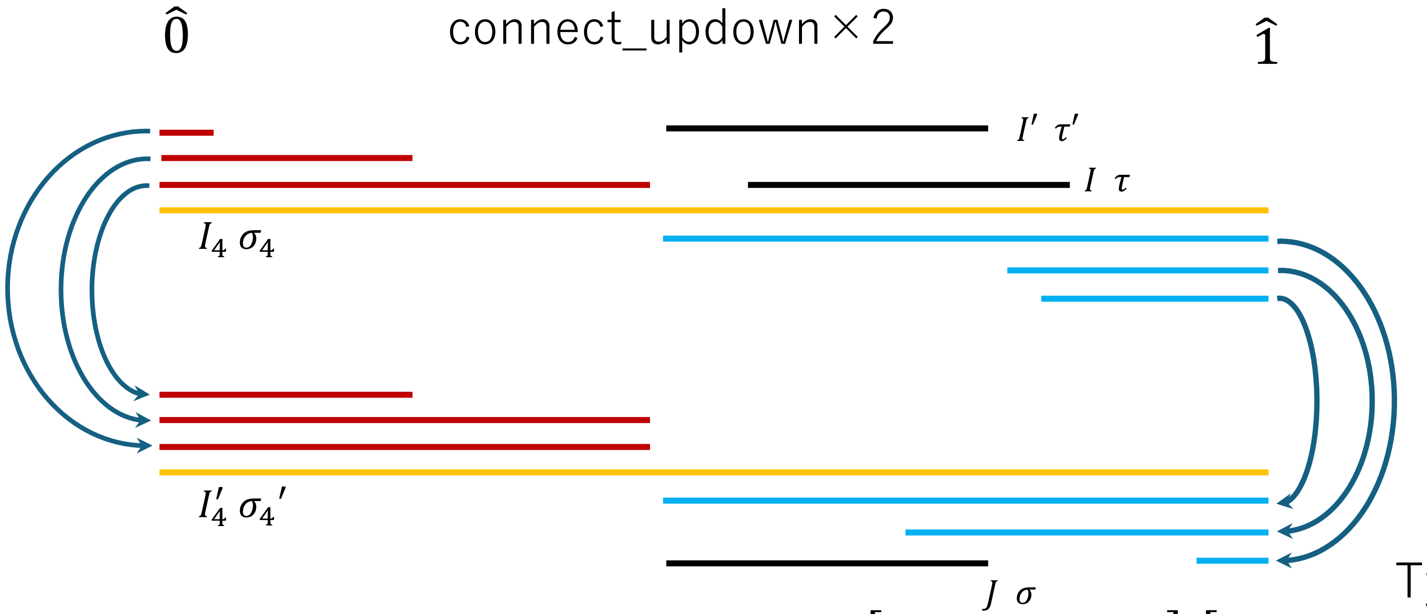
(5.5)interval_decomposition(FSCa,FSCb)

入力: FSCa, FSCb (bipath filtration)
出力: 辞書 $i \mapsto \mathcal{B}(H_i(S))$ (ith bipath PHの区間分解), FSCaの長さ, FSCbの長さ

この関数はこれまでの関数を組み合わせたもの。

入力 –> A型区間分解×2 –> 行列を構成 –> **簡約化** –> **区間分解** –> 次数で区間を分類 –> 出力

$$\begin{array}{cc|ccc} & & \sigma_1 & \sigma_2 & \sigma_3 \\ & & I_1 & I_2 & I_3 \\ \hline \sigma_1{}' & I_1' & 0 & 0 & 1 \\ \sigma_2{}' & I_2' & 0 & 1 & 0 \\ \sigma_3{}' & I_3' & 1 & 0 & 0 \end{array}$$



$$\begin{array}{cc|ccc} & & \tau_1 & \tau_2 & \tau_3 \\ & & J_1 & J_2 & J_3 \\ \hline \tau_1{}' & J_1' & 0 & 0 & 1 \\ \tau_2{}' & J_2' & 0 & 1 & 0 \\ \tau_3{}' & J_3' & 1 & 0 & 0 \end{array}$$

Type L

$[\,[[I_1,\sigma_1],[I_3',\sigma_3']],[[I_2,\sigma_2],[I_2,\sigma_2']],[[I_3,\sigma_3],[I_1',\sigma_1']]]$

Type R

$[[J_1,\tau_1],[J_3',\tau_3']],[[J_2,\tau_2],[J_2',\tau_2']],[[J_3,\tau_3],[J_1',\tau_1']]\,]$

(5)BipathMatMethod.jl

♦区間の出力(これまでの関数を用いて区間分解をし、出力)

(5.5)interval_decomposition(FSCa,FSCb)

入力: FSCa, FSCb (bipath filtration)
出力: 辞書 $i \mapsto \mathcal{B}(H_i(S))$ (ith bipath PHの区間分解), FSCaの長さ, FSCbの長さ

この関数はこれまでの関数を組み合わせたもの。

入力 –> A型区間分解×2 –> 行列を構成 –> **簡約化** –> **区間分解** –> 次数で区間を分類 –> 出力

Type: Left $[\,[[I_1,\sigma_1],[I_3',\sigma_3']],[[I_2,\sigma_2],[I_2,\sigma_2']],[[I_3,\sigma_3],[I_1',\sigma_1']]]$

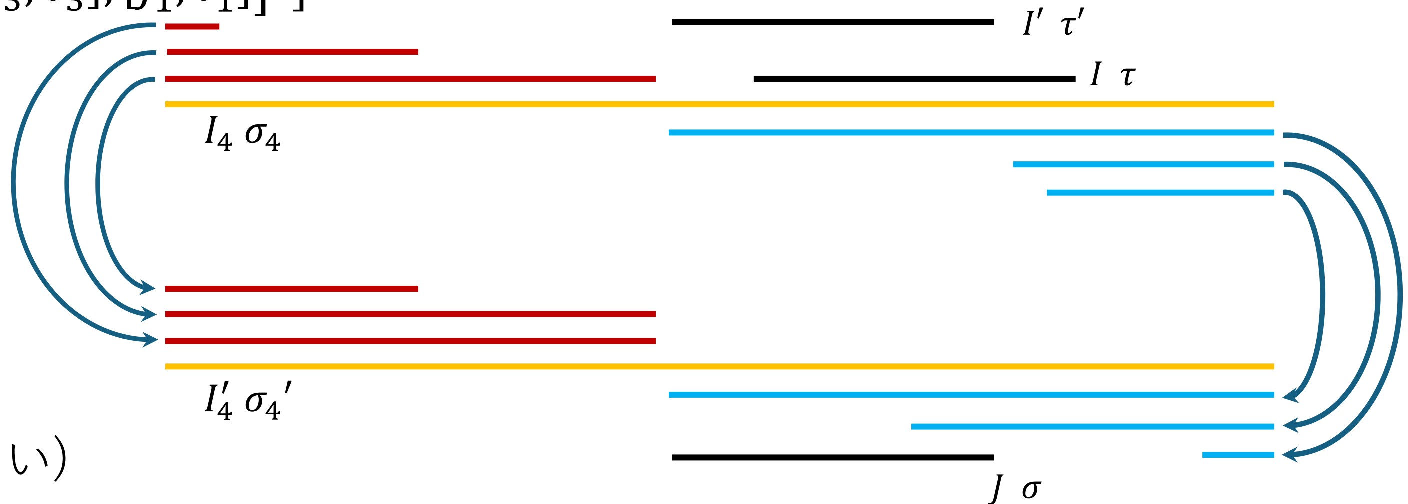
Type: Right $[[J_1,\tau_1],[J_3',\tau_3']],[[J_2,\tau_2],[J_2',\tau_2']],[[J_3,\tau_3],[J_1',\tau_1']]\ ]$

Type: Up $[[I,I'],[\tau,\tau']]$

Type: Down $[[J],[\sigma]]$

Type: Center* $[[I_4],[\sigma_4]]$

*上の区間だけとる($I_4'\ \sigma_4'$はいらない)



(5)BipathMatMethod.jl

◆区間の出力(これまでの関数を用いて区間分解をし、出力)

(5.5)interval_decomposition(FSCa,FSCb)

入力: FSCa, FSCb (bipath filtration)
出力: 辞書 $i \mapsto \mathcal{B}(H_i(S))$ (ith bipath PHの区間分解), FSCaの長さ, FSCbの長さ

この関数はこれまでの関数を組み合わせたもの。

入力 –> A型区間分解×2 –> 行列を構成 –> 簡約化 –> 区間分解 –> **次数で区間を分類** –> **出力**

Left = $[[[I_1,\sigma_1],[I_3',\sigma_3']],[[I_2,\sigma_2],[I_2,\sigma_2']],[[I_3,\sigma_3],[I_1',\sigma_1']]]$

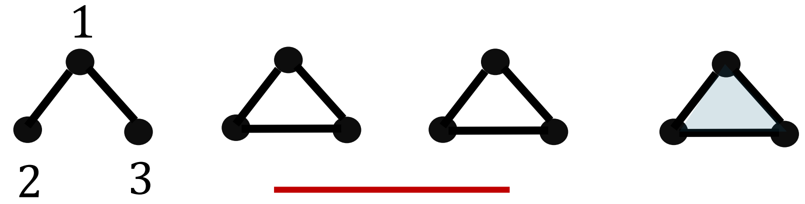
Right = $[[[J_1,\tau_1],[J_3',\tau_3']],[[J_2,\tau_2],[J_2',\tau_2']],[[J_3,\tau_3],[J_1',\tau_1']]]$

Up = $[[I,I'],[\tau,\tau']]$

Center= $[[I_4],[\sigma_4]]$

Down= $[[J],[\sigma]]$

単体を見れば区間が何次のホモロジーかわかる



I: $[[1,2],[2,3],[1,3]]$

各$i \in \{0,1,\dots\}$に対して、
$i \mapsto i$次の区間の集まり
を対応させることで辞書を作っている.

*Center: 上の区間だけとる(I_4' σ_4'はいらない)

(5)BipathMatMethod.jl

♦区間の出力(これまでの関数を用いて区間分解をし、出力)

(5.5)interval_decomposition(FSCa,FSCb)

入力: FSCa, FSCb (bipath filtration)
出力: 辞書 $i \mapsto \mathcal{B}(H_i(S))$ (ith bipath PHの区間分解), FSCaの長さ, FSCbの長さ

この関数はこれまでの関数を組み合わせたもの。

入力 –> A型区間分解×2 –> 行列を構成 –> 簡約化 –> 区間分解 –> 次数で区間を分類 –> **出力**

```
 ∃0_th homology, #[0,1] is 1
intervals with 0:
intervals with 1:
intervals up: <1,1> <3,5> <4,4>
intervals down: <1', 1'> <3', 5'> <4', 4'>
................
 ∃1_th homology, #[0,1] is 0
intervals with 0:
intervals with 1:
intervals up: <7,9> <8,8>
intervals down: <7', 9'> <8', 8'>
................
```

＊この関数を使うと、ターミナル上に左のような文字が表示される。区間の存在の有無が分かり便利。

次は表示するためのコードを見る。

(6)Print_functions.jl

♦bipath上の区間を以下のようにターミナルに表示する。

```
 ∃0_th homology, #[0,1] is 1
intervals with 0:
intervals with 1:
intervals up: <1,1> <3,5> <4,4>
intervals down: <1', 1'> <3', 5'> <4', 4'>
................
 ∃1_th homology, #[0,1] is 0
intervals with 0:
intervals with 1:
intervals up: <7,9> <8,8>
intervals down: <7', 9'> <8', 8'>
................
```

(6)Print_functions.jl

♦bipath上の区間を以下のようにターミナルに表示する。

(6.1)print_intL(interval)　(typy Lの区間 $\mapsto$ string ($\langle s, t\rangle$))
(6.2)print_intR(interval)　(typy Rの区間 $\mapsto$ string ($\langle s, t\rangle$))
(6.3)print_up(interval)　(typy Uの区間 $\mapsto$ string ($\langle s, t\rangle$))
(6.4)print_down(interval)　(typy Dの区間 $\mapsto$ string ($\langle s, t\rangle$))

だいたい同じ

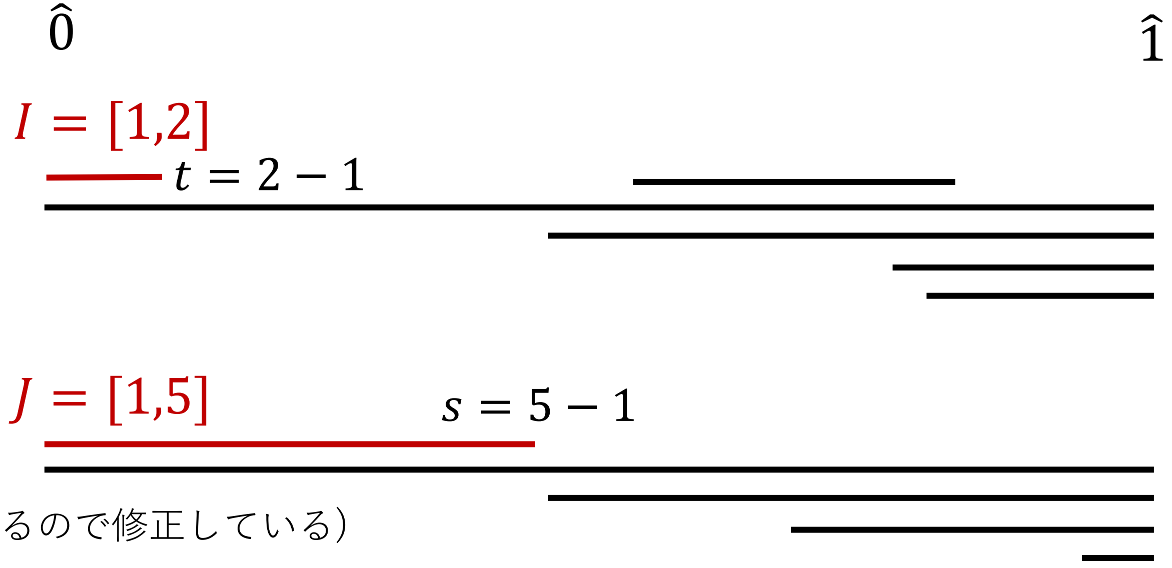
(6.5)print_intervals(header, intervals, printer) (上のコードを用いて区間をprintしてくれる.)

＊上4つの関数 print_intL, print_intR, print_up, print_downは殆ど一緒なのでprint_intLだけ説明
＊関数として区間$B_{n,m}$をprintするprint_centerは無い.（関数を作るまでもない）

(6)Print_functions.jl

♦Type Lの区間を$\langle s,t \rangle$の形で表す。

(6.1)print_intL(interval) (typy Lの区間 $\mapsto$ string ($\langle s,t \rangle$))



Ex. type Lの区間は上側の区間と下側の区間を合わせたものと考えている。いま、区間として

$$[I, J] = [[1,2],[1,5]]$$

を考える.

$$s = 5 - 1 = 4$$
$$t = 2 - 1 = 1$$

（A型と$B_{n,m}$の区間では数字が1ずれるので修正している）

より、出力は$\langle 4,1 \rangle$.

*(6.2)—(6.4)も同様に、bipath posetの区間 $\mapsto$ string ($\langle s,t \rangle$))の対応を与える

(6)Print_functions.jl

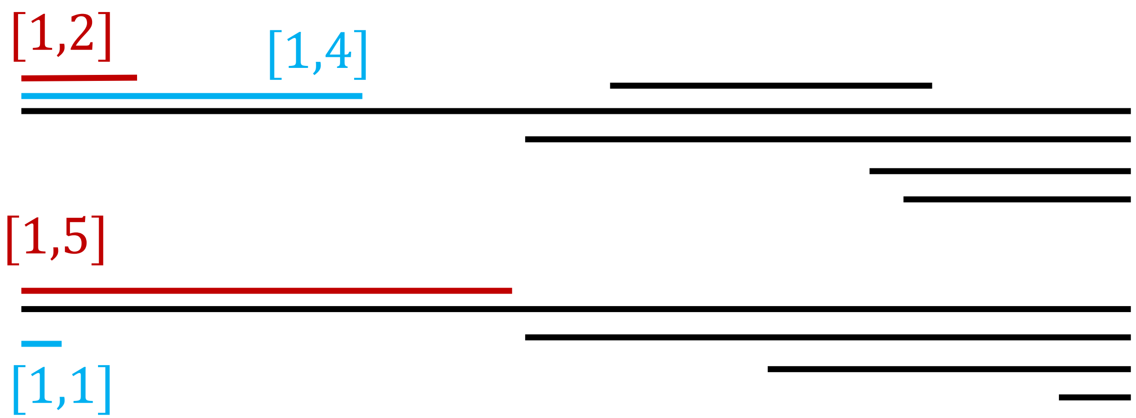
♦(6.1)—(6.4)を用いて区間をプリントする。

(6.5)print_intervals(header, intervals, printter) (区間をprintしてくれる.)

Ex.

print_intervals(“intervals with $\hat{0}$: ”, [[[1,2], [1,5]], [[1,4], [1,1]]] , print_intL)を計算。

intervals with $\hat{0}$: $\langle 4,1 \rangle, \langle 3, \hat{0} \rangle$



がターミナルに表示される(色は付かない).

(7)BipathPD.jl

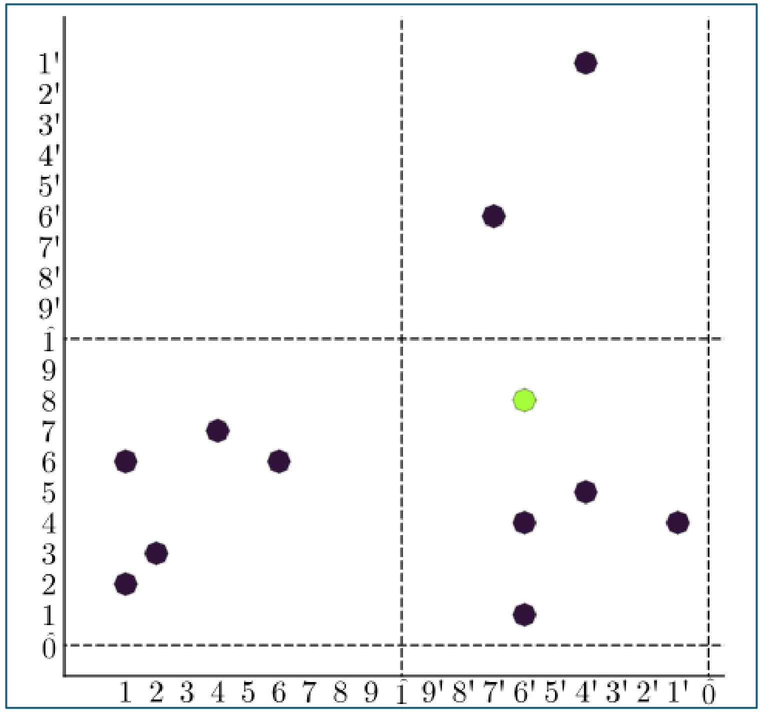
♦ BipathPD.jlではbipathの区間からBipathPDを出力する関数を書く。

入力：Bipathの区間+上下のpathの長さ $\xrightarrow{(7.1)}$ 平面上の点 $\xrightarrow{(7.2),(7.3)}$ **出力**： bipathPD

(7.4)

このファイルは４つの関数からなる。

(7.1) intervalstoplane(intL,intR,up,down,center,n,m)
(7.2) plotpoints(points,n,m)
(7.3) colorfunc(col, max::Int, mult::Int)
(7.4) plotintlist(X,i_th::Int)



i thのバイパスパーシステンス図

(7)BipathPD.jl

(7.1) intervalstoplane(intL, intR, up, down, center, n, m)

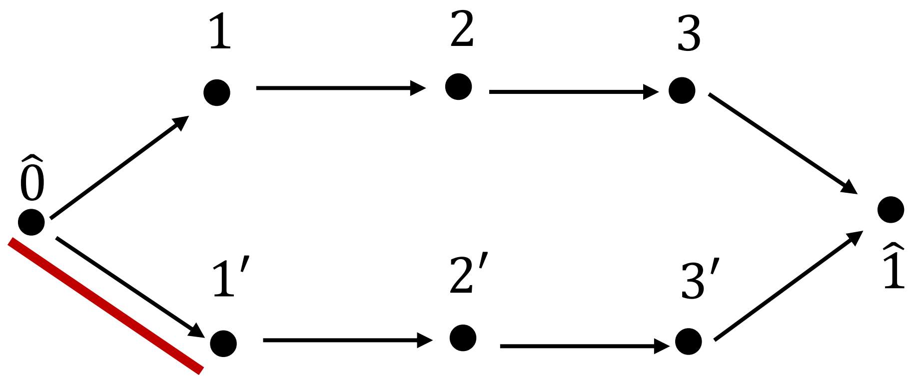
入力： Bipathの区間+上下のpathの長さ$\xrightarrow{(7.1)}$平面上の点→ 出力： bipathPD

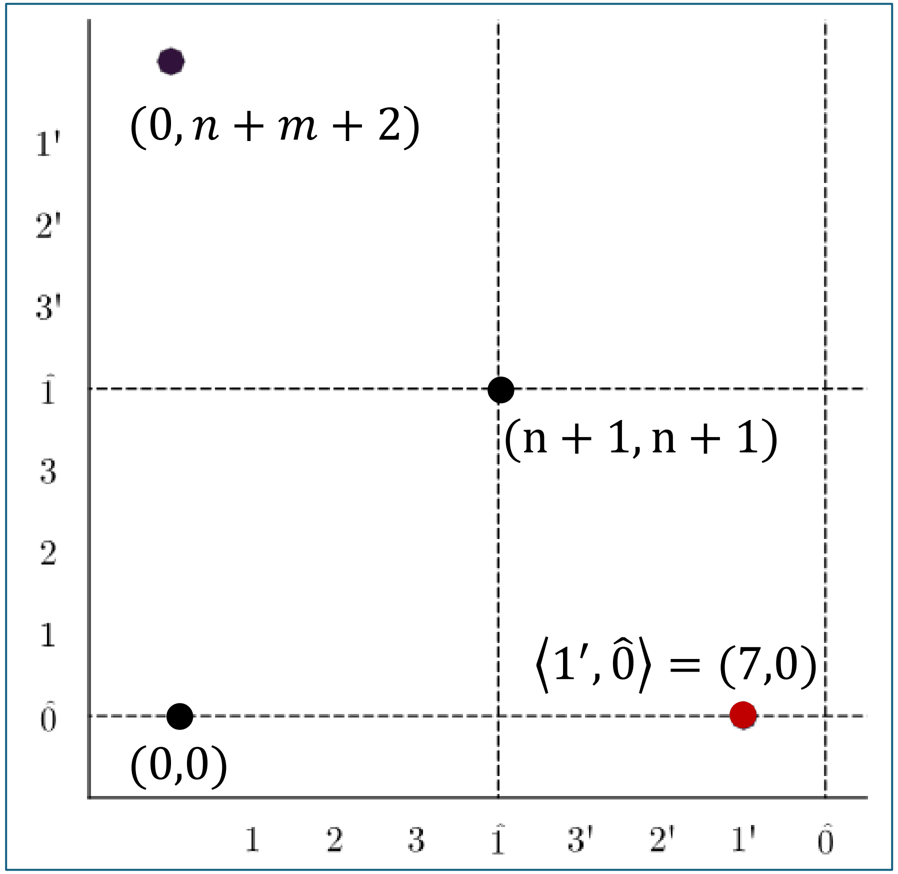
Bipath poset$B_{n,m}$のそれぞれのタイプの区間（ intL, intR, up, down, center ）を平面上の点の座標に変換する。

Ex. $n = 3, m = 3$

Type L: $[[1,1],[1,2]] \mapsto (7,0) = \langle 1', \hat{0} \rangle$

$B_{3,3} := [1, n+m+8] \mapsto (0, n+m+2)$



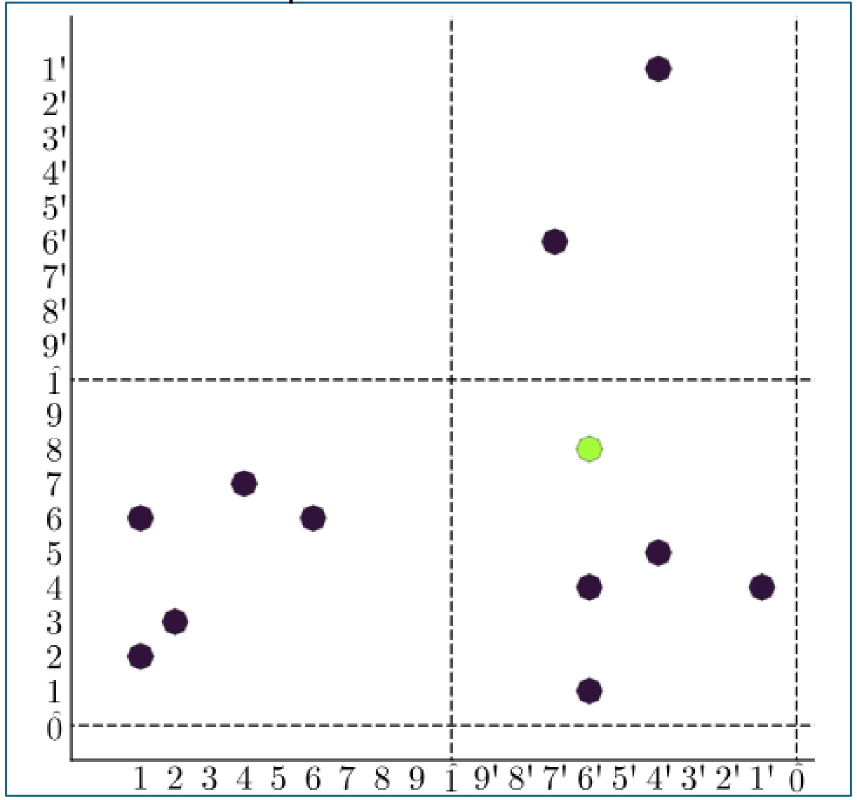


(7)BipathPD.jl

(7.2) **plotpoints(points,n,m)**　　#points=[[a,b],[c,d],[a,b],...]

(7.2)

入力： Bipathの区間+上下のpathの長さ→ 平面上の点→ **出力**： bipathPD



・与えられたpointsを平面上にプロットする。
・縦軸と横軸のメモリを書く。
・点線を書く。

＊区間を平面上の点(a,b)に対応させたものたちを入力とする

(7)BipathPD.jl

(7.3) colorfunc(color, max(全ての点の重複度の最大値), mult(プロットしたい点の重複度))

(7.3)

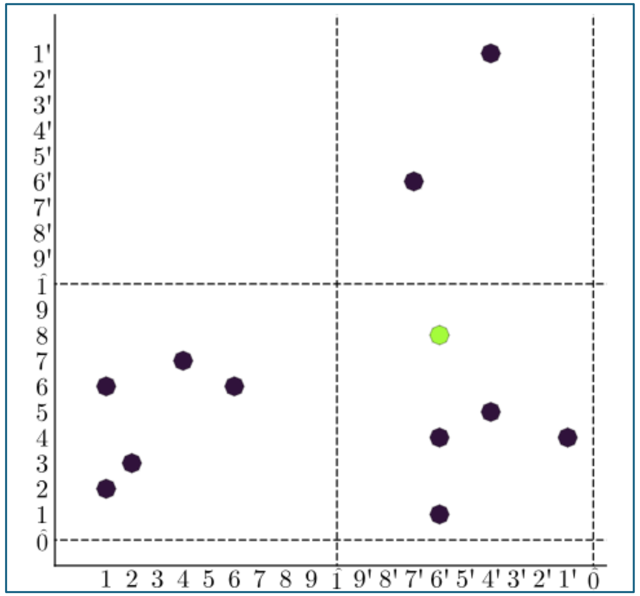
入力： Bipathの区間+上下のpathの長さ→ 平面上の点→ **出力**： bipathPD

♦ 重複度によってプロットする点の色を変えるための関数.

```
function colorfunc(col, max::Int, mult::Int)
    d = div(length(col), max)
    return (d <= 1 || max == 1) ? 1 : min(1 + d * (mult - 1), length(col))
end
```

max:全ての区間の重複度の最大値
mult:ある区間の重複度を入れる想定をしている

色(color)をmax等分して, $i\,/\mathrm{max} < \mathrm{mult} \leq (i+1)/\mathrm{max}$ にいるようなiを捜している.



`color =  cgrad(:turbo)`として採用　`Plots`

小　　重複度　　大

＊JuliaのPlotsを用いてプロットをしている。
＊ `cgrad(:turbo)`から色をとって来ている。
＊重複度が大きいほど赤色になるようにしている。

(7)BipathPD.jl

(7.4) plotintlist(X, i_th)

◆今までの関数を合わせて区間をプロットする。

(7.1)　　　　(7.2),(7.3)

入力：Bipathの区間+上下のpathの長さ → 平面上の点→ **出力**： bipathPD

(7.4)

X = interval_decomposition(FSCa,FSCb)

Bipath filtration
(FSCa, FSCb)

▶

区間分解

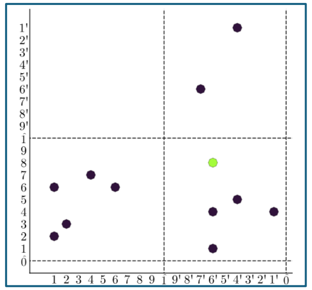
▶ $i$thの区間たち

(7.1) intervalstoplane(intL,intR,up,down,center,n,m)

▶ 平面上の点に変換

(7.3) plotpoints(points,n,m)

▶ 点をプロット、図を出力。



(8)clique.jl

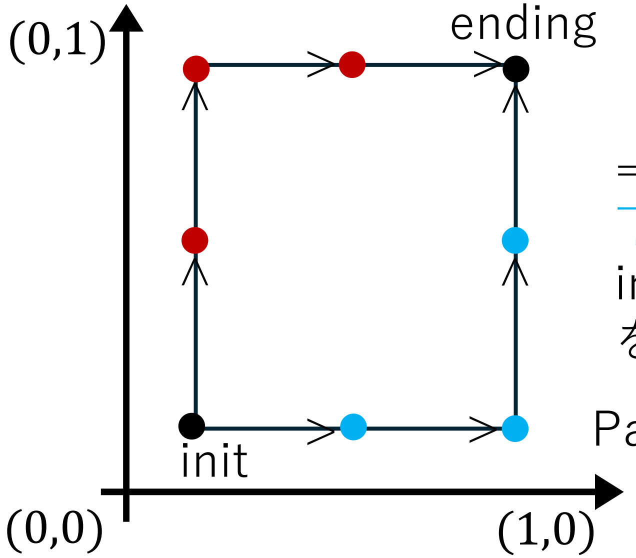
◆ clique.jlではランダムにクリーク複体のバイパスフィルトレーションを出力する関数を書く(例を作るため)。

このファイルは2つの関数からなる。

(8.1) get_rectangular_paths(init,ending,partition::Int)
(8.2) clique_random(G,faces,path1,path2)

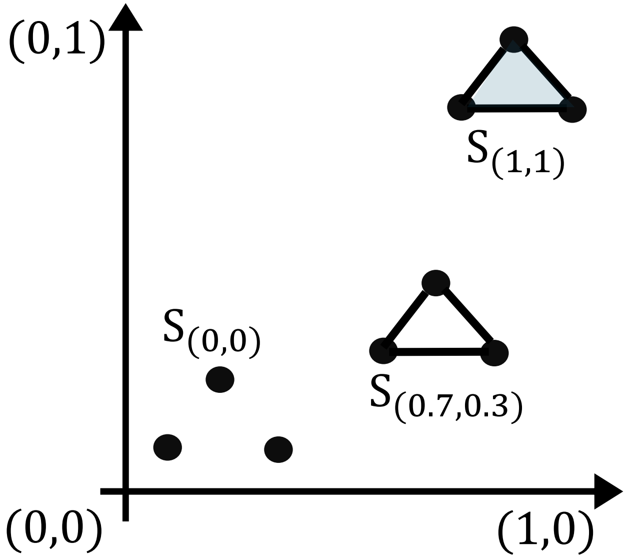
(8.2)Bi-filtration $S: [0,1]^2 \to$ Simpをランダムに作りバイパスに制限して得られるバイパスフィルトレーションを得る。($B_{n,,m} \to [0,1]^2 \to$ Simp)

(8.1)のpicture.



=> 上側のpathと下側のpath(共にinit,endingを含む)を取る

Partition = 2



(8)clique.jl

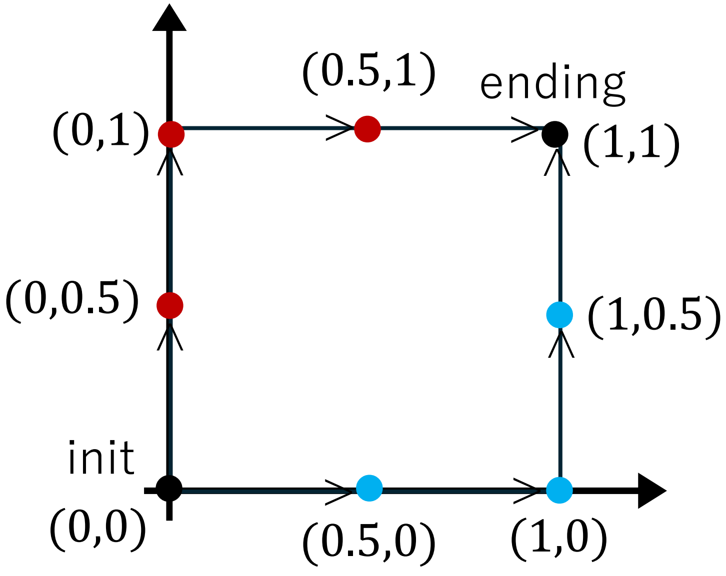
(8.1) get_rectangular_paths(init,ending,partition::Int)

入力:
init, ending $\in [0,1]^2$
partition $\in \mathbb{Z}_{>0}$.
出力:
uppath, underpath $\in ([0,1]^2)^{(2*\text{partition}+1)}$

例.
init = [0,0]
ending =[1,1]
partition=2
=>

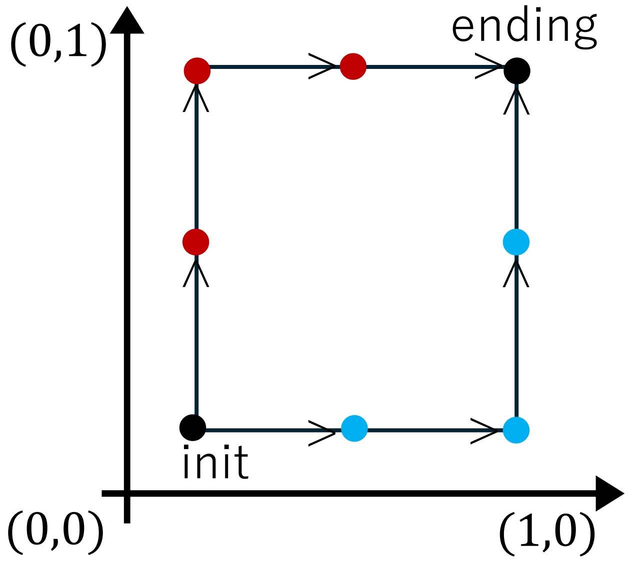
```
uppath =[ [0.0, 0.0],
          [0.0, 0.5],
          [0, 1],
          [0.5, 1.0],
          [1, 1]
        ]

underpath =[ [0.0, 0.0],
             [0.5, 0.0],
             [1, 0],
             [1.0, 0.5],
             [1, 1],
           ]
```

$B_{3,3}$-filtration



(8)clique.jl

```
function get_rectangular_paths(init,ending,partition::Int)
    underpath =[]
    uppath=[]
    lx = (ending[1] - init[1])/partition
    ly = (ending[2] - init[2])/partition
    for i in 1:partition
        push!(underpath,init+[lx*(i-1),0])
        push!(uppath,init+[0,ly*(i-1)])
    end
    push!(underpath,[ending[1],init[2]])
    push!(uppath,[init[1],ending[2]])
    for i in 1:partition-1
        push!(underpath,[ending[1],init[2]+ly*(i)])
        push!(uppath,[init[1]+lx*(i),ending[2]])
    end
    push!(underpath,ending)
    push!(uppath,ending)
    return uppath, underpath
end
```



(8)clique.jl

(8.2) clique_random(G,faces,path1,path2)

Bi-filtration S: $[0,1]^2 \to$ Simpをランダムに作り、
そこから(1)の関数を用いてバイパスフィルトレーショ
ンを得る($\mathrm{B}_{n,,m} \to [0,1]^2 \to$ Simp)

入力：
G ⋯ 完全グラフ　(using SimpleGraphs)
faces⋯ グラフGの頂点集合のべき集合。
path1,path2⋯ $[0,1]^2$の元のリストで$\mathbb{R}^2$ の順序で整列されているもの, かつ始点と終点が一致してるもの。((8.1)で得られるuppath, underpathを想定)

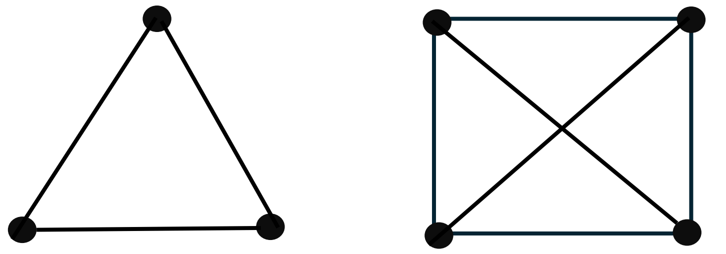
出力：バイパスフィルトレーション

１．(完全グラフGの頂点に$v = [i]$に重み[0,0]を乗せる)
グラフの辺$e = [i, j]$に重み$w_e \in [0,1]^2$を乗せる.
=>グラフのbi-filtration $S = \{S_{a,b}\}_{(a,b)\in[0,1]^2}$ ができる。

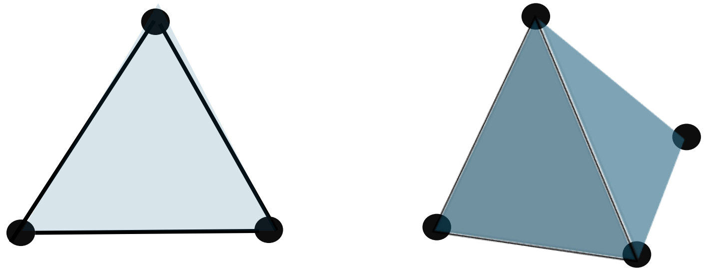
2.　各$(a, b) \in [0,1]^2$に対して$S_{a,b}$ から定まるクリーク複体$X(S_{a,b})$を取る。ことで、新たなX$(S) = \{X(S_{a,b})\}_{(a,b)\in[0,1]^2}$を得る.

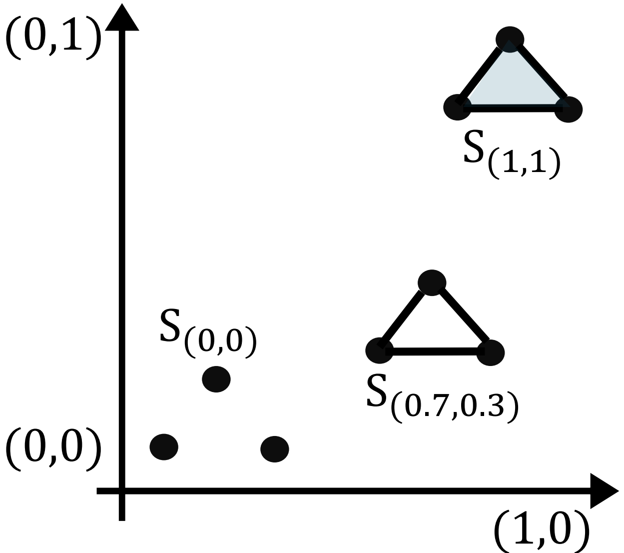
3.　Bipathに制限することでbipath filtrationを得る.

例. 完全グラフ(頂点数3, 4)



例. クリーク複体





(7)BipathPD.jl

(4) plotintlist(X,i_th::Int)

```
X = interval_decomposition(FSCa,FSCb)
```

```
function plotintlist(X,i_th::Int) #X = interval_decomposition(FSCa,FSCb)
    if (i_th in keys(X[1])) == false #check that we have points to plot.
        println("no ",i_th," homology.")
        return false
    end
    dict_int = X[1][i_th]
    n, m = X[2] - 2, X[3] - 2
    intL, intR, up, center, down = dict_int[1], dict_int[2], dict_int[3], dict_int[4], dict_int[5]
    intlist = intervalstoplane(intL,intR,up,down,center,n,m)
    plotpoints(intlist,n,m)
end
```

(5)BipathMatMethod.jl

◆Matrix Problemの準備

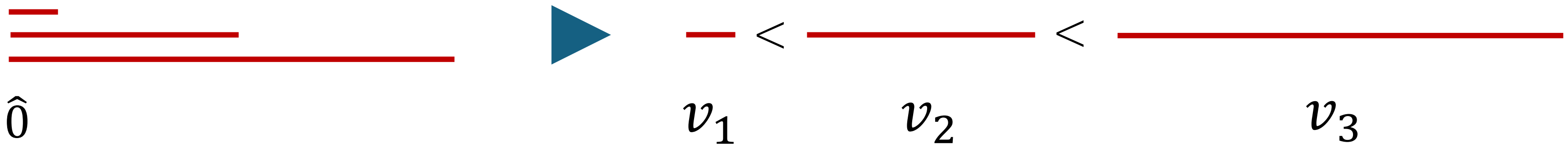
(5.2) get_basis_intervals_left(FSC,left_int_basis)

FSC: Filtered Simplicial Complex
left_int_basis: [leftintervals,leftbasis]

FSCに含まれる単体複体を整列し、単体たちによって生成されるベクトル空間の基底を固定する。この基底に対して、 type **L**の区間に対応する元をベクトル化し、それらをひとつの行列にする。

$$
\begin{array}{c|cccccccc}
 & [1,1] & \dots & [1,n-1] & [1,n] & [2,n] & \dots & [n,n] \\
\hline
[1,1] & * & \cdots & * & * & & & \\
\vdots & \vdots & \ddots & \vdots & \vdots & & & \\
[1,n-1] & * & \cdots & * & * & & & \\
[1,n] & 0 & \cdots & 0 & * & * & \cdots & * \\
[2,n] & & & & 0 & * & \cdots & * \\
\vdots & & & & \vdots & \vdots & \ddots & \vdots \\
[n,n] & & & & 0 & * & \cdots & *
\end{array}
$$

[leftintervals,leftbasis]



▶ 行列[$v_1 v_2\ v_3$] を出力(size= 単体数×区間の数)

(5)BipathMatMethod.jl

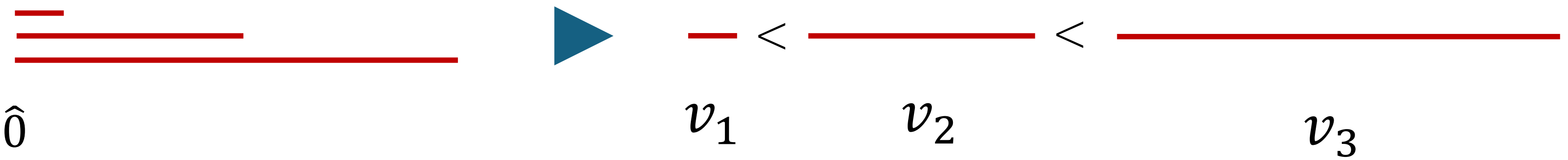
◆Matrix Problemの準備

(4) get_basis_intervals_left(FSC,left_int_basis, imagebasis_left)

$$
\begin{array}{c|ccccccc|}
 & [1,1] & \cdots & [1,n-1] & [1,n] & [2,n] & \cdots & [n,n] \\
\hline
[1,1] & * & \cdots & * & * & & & \\
\vdots & \vdots & \ddots & \vdots & \vdots & & & \\
[1,n-1] & * & \cdots & * & * & & & \\
[1,n] & 0 & \cdots & 0 & * & * & \cdots & * \\
[2,n] & & & & 0 & * & \cdots & * \\
\vdots & & & & \vdots & \vdots & \ddots & \vdots \\
[n,n] & & & & 0 & * & \cdots & *
\end{array}
$$

FSCに含まれる単体複体を整列し、単体たちによって生成されるベクトル空間の基底を固定する。この基底に対して、 type **L**の区間に対応する元をベクトル化し、それらをひとつの行列にまとめる

[leftintervals,leftbasis]



▶ 行列$[v_1 v_2\ v_3\ B_{\hat{0}}]$ を出力

$B_{\hat{0}}$ は境界準同型のimage at $\hat{0}$.

(5)BipathMatMethod.jl

◆Matrix Problemの準備

(3) get_basis_intervals_right(FSC,right_int_basis)

FSC: Filtered Simplicial Complex
right_int_basis: [rightintervals,rightbasis]

FSCに含まれる単体複体を整列し、単体たちによって生成されるベクトル空間の基底を固定する。この基底に対して、 type **R**の区間に対応する元をベクトル化し、それらをひとつの行列にする。

$$
\begin{array}{c|ccccccc|}
 & [1,1] & \dots & [1,n-1] & [1,n] & [2,n] & \dots & [n,n] \\
\hline
[1,1] & * & \cdots & * & * & & & \\
\vdots & \vdots & \ddots & \vdots & \vdots & & & \\
[1,n-1] & * & \cdots & * & * & & & \\
[1,n] & 0 & \cdots & 0 & * & * & \cdots & * \\
[2,n] & & & & 0 & * & \cdots & * \\
\vdots & & & & \vdots & \vdots & \ddots & \vdots \\
[n,n] & & & & 0 & * & \cdots & *
\end{array}
$$

▶ 行列$[w_1 w_2\ w_3\ ]$を出力(size= 単体数×区間の数)

(5)BipathMatMethod.jl

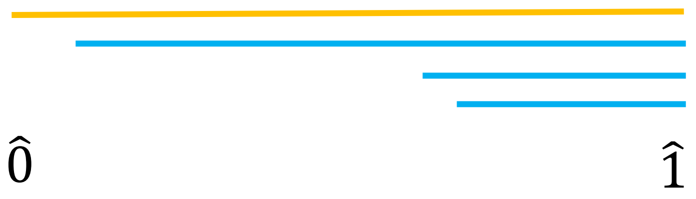
◆Matrix Problemの準備

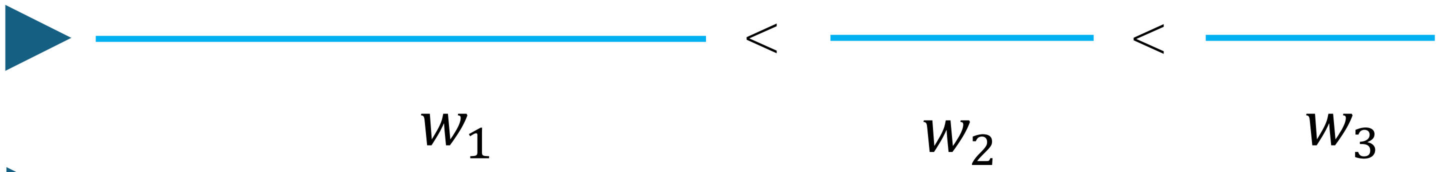
(5)get_basis_intervals_right(FSC,right_int_basis,imagebasis, centerwithB)

$$
\begin{array}{c|ccccccc|}
 & [1,1] & \dots & [1,n-1] & [1,n] & [2,n] & \dots & [n,n] \\
\hline
[1,1] & * & \cdots & * & * & & & \\
\vdots & \vdots & \ddots & \vdots & \vdots & & & \\
[1,n-1] & * & \cdots & * & * & & & \\
[1,n] & 0 & \cdots & 0 & * & * & \cdots & * \\
[2,n] & & & & 0 & * & \cdots & * \\
\vdots & & & & \vdots & \vdots & \ddots & \vdots \\
[n,n] & & & & 0 & * & \cdots & *
\end{array}
$$

FSCに含まれる単体複体を整列し、単体たちによって生成されるベクトル空間の基底を固定する。この基底に対して、 type **R**, **B**の区間に対応する元をベクトル化し、それらをひとつの行列にし、

$\hat{1}$での境界準同型のImageの基底をベクトル化したものをまとめる。





▶ 行列$[w_1 w_2\ w_3\ B_{\hat{1}}\ C]$ を出力

$B_{\hat{1}}$ は境界準同型のimage at $\hat{1}$.

Cは区間type **B**たちに付随する基底のベクトル化.

(5)BipathMatMethod.jl

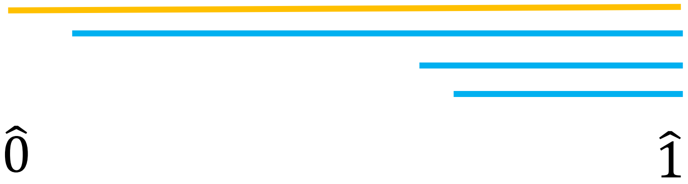
◆Matrix Problemの準備

(5.3)get_basis_intervals(FSC_vect, int_basis, image_basis = true, center basis = true)

FSC_vect: FSCの単体をベクトル化したもの。
int_basis: 区間に付随する鎖複体の元たち。
image_basis: 区間に付随する鎖複体の元たち。
center_basis: 区間に付随する鎖複体の元たち。

$$
\begin{array}{c|ccccccc|}
 & [1,1] & \dots & [1,n-1] & [1,n] & [2,n] & \dots & [n,n] \\
\hline
[1,1] & * & \cdots & * & * & & & \\
\vdots & \vdots & \ddots & \vdots & \vdots & & & \\
[1,n-1] & * & \cdots & * & * & & & \\
[1,n] & 0 & \cdots & 0 & * & * & \cdots & * \\
[2,n] & & & & 0 & * & \cdots & * \\
\vdots & & & & \vdots & \vdots & \ddots & \vdots \\
[n,n] & & & & 0 & * & \cdots & * \\
\end{array}
$$

FSCに含まれる単体複体を整列し、単体たちによって生成されるベクトル空間の基底を固定する。この基底に対して、 int_basis、image_basis, center_basisの中の区間に対応する鎖複体元をベクトル化し、それらをひとつの行列にする。



▶ $w_1 < w_2 < w_3$

▶ 行列$[w_1 w_2 \ w_3 \ B_{\hat{1}} \ C]$ を出力

$B_{\hat{1}}$ は境界準同型のimage at $\hat{1}$.

C

Cは区間type **B**たちに付随する基底のベクトル化.

(6)Print_functions.jl

(6.1)print_intL(interval) (typy Lの区間 ↦ string ($\langle s, t\rangle$))

♦Type Lの区間を$\langle s, t\rangle$の形で表す。

```
function print_intL(interval)
    s, t = interval[2][2] - 1, interval[1][2] - 1
    return "<" * (s == 0 ? "0̂" : string(s)) * "' ," * (t == 0 ? “0̂" : string(t)) * "> "
end
```

(6)Print_functions.jl

♦(6.1)—(6.4)を用いて区間をプリントする。

(6.5)print_intervals(header, intervals, printter) (区間をprintしてくれる.)

```
function print_intervals(header, intervals(同じtypeの区間たち), printter(print_intLなど先程の関数))
    print(header)
    for interval in intervals
        print(printter(interval))
    end
    println(" ")
end
```

Ex.

print_intervals(“intervals with $\hat{0}$: ”, [[[1,2], [1,5]], [[1,4], [1,1]]] , print_intL)

を計算。

intervals with $\hat{0}$: $\langle 4,1\rangle, \langle 3, \hat{0}\rangle$

がターミナルに表示される(色は付かない).