修士論文

絡み目の不変量としての Jones 多項式

岡山理科大学大学院
理学研究科

応用数学専攻・幾何学研究室
S97MM07・須藤 加奈子

目 次

1. 緒言

重要な不変量である Jones 多項式が 1 になる絡み目を探し, その中に非平凡な絡み目があるかどうか調べる事が最終的な目標である.

絡み目が平凡ならばその Jones 多項式は 1 だが, その逆が成り立つかどうかは未解決問題である. そこでできるだけ大量の絡み目の Jones 多項式を計算して Jones 多項式が 1 の非平凡な絡み目があるかどうかを調べたい.

その第 1 歩として, ネットワークを介した複数のコンピュータによる分散計算システムを作った.

謝辞

この論文を作成するにあたり, ご指導いただきました橋爪道彦先生に, 心から感謝いたします.

また, 数多くの有用なアドバイスをいただいた京都産業大学の山田 修司先生に, 心から感謝いたします.

2. 概要

まず絡み目の基本的な用語, 特に絡み目の不変量についてその計算アルゴリズムを中心に詳しく説明する.

次に組み紐の定義を述べ, 組み紐と絡み目の関係について解説する.

最後に代表的な不変量である Jones 多項式を計算するためのシステムの構成を説明する.

3. 結び目理論に関する準備

3.1. 絡み目の定義・正則表示

R^3 内の絡み目 L の多角形を結び目とよび, 互いに粗な結び目の和集合を絡み目とよぶ. μ 個の結び目 $K_1, K_2, \ldots, K_\mu$ から成る絡み目 L に対して $K_1, K_2, \ldots, K_\mu$ を成分とよぶ.

$R^3 = \{(x, y, z) | x, y, x \in R\}$ の $z = 0$ で定まる平面を R^2 と同一視する. R^3 中の絡み目 K の写像 $P : (x, y, z) \mapsto (x, y, 0)$ による K の像 $P(K)$ の各点 $v \in P(K)$ に対して

- $P^{-1}(v) \cap K$ の個数は高々2 個である.

- v が頂点の像ならば, $P^{-1}(v)$ の点の個数は 2 ではない.

をみたす時, K は正則の位置にあるという. この時像 $P(K)$ を正則射影とよぶ. さらに $P^{-1}(v) \cap K$ が 2 点の時, v を交差点とよぶ.

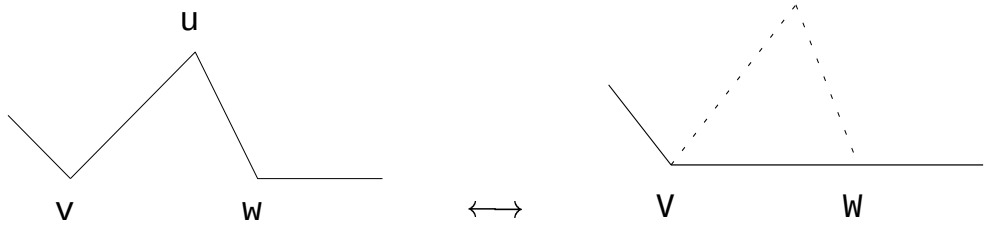
u を $P(K)$ の交差点とする. $K \cap P^{-1}(u)$ の 2 点を $q = (r, s, t)$ と $q' = (r, s, t')$ $(t < t')$ とする. この時, q' を通る辺の一部を上交差線とよび, q を通る辺の一部を下交差線とよぶ.

$P(K)$ の u を通る辺のうち下交差線に対応する交差点の近傍の一部を削除して得られる図形

を正則表示とよぶ.

3.2. 同値・平凡・不変量

R^3 内の絡み目 K のある頂点 u と, u をはさむ 2 辺上の 2 点 v, w で v と w を K と交わらない線分で結ぶことが出来るものに対して, 折線 vuw を線分 vw で置き換えると別の絡み目 K' を得る. K から K' を得る操作及びその逆の K' から K を得る操作を $\triangle$ 移動とよぶ.



2 つの絡み目 T と T' が $\triangle$ 移動の繰り返しで移りあう時, 同値であるといい $T \cong T'$ と書く.

交差しない円周の和集合を正則表示にもつ絡み目に同値な絡み目を, 平凡であるという.

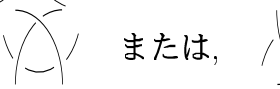
各々の絡み目 T に対してある数学的な量 $\rho(T)$ が決まって, $T \cong T'$ ならば $\rho(T) = \rho(T')$ となる時, 対応 ρ を絡み目の不変量という. 不変量 ρ が決まると, $\rho(T) \neq \rho(T')$ ならば $T \ncong T'$ である, と判る.

3.3. Reidemeister 移動

T の正則表示から, 以下に定めた移動と平面中の $\triangle$ 移動を繰り返し使って T' の正則表示に変形できた時, $T \cong T'$ である.

Reidemeister 移動 I : 

Reidemeister 移動 II :  ∼

Reidemeister 移動 III :  ∼  または,  ∼

逆に, $T \cong T'$ ならばそれらの正則表示は, Reidemeister 移動 I,II,III と平面中の $\triangle$ 移動の繰り返しで移り合う.

T と T' の正則表示が Reidemeister 移動 II, III と $\triangle$ 移動の繰り返しで移り合う時, T と T' は *regular isotopy*(正則同位) であるという.

T と T' の正則表示が Reidemeister 移動 I,II,III と $\triangle$ 移動の繰り返しで移り合う時, つまり絡み目として同値な時, T と T' は *ambient isotopy*(全同位) であるとも言う.

3.4. 有向絡み目

絡み目の各成分に向きが 1 つ指定された図形を有向絡み目とよぶ.

2 つの有向絡み目 T と T' が $\triangle$ 移動の繰り返しで向きを保ったまま移りあう時, 同型であるという.

有向絡み目 K と K の交差点 p に対して, p の符号 $\varepsilon = \varepsilon(p)$ を以下のように定める.

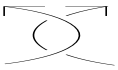
上交差線を下交差線と重なるまで反時計回りに回し辺が重なった時, 向きが一致すればその交差点の符号を $\varepsilon = 1$, 一致しなかったら $\varepsilon = -1$ とする.

K の各交差点の符号を全て合計したものを捻り数とよび $w(K)$ と書く. つまり

$$w(K) = \sum_{p} \varepsilon(p)$$

と定義する.

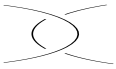
まず $w(K)$ が正則同位の不変量かどうかを調べる. Reidemeister 移動 II に関する不変性を調べる.

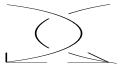
 を含む有向絡み目 K と, その部分に Reidemeister 移動 II をほどこした有向絡み目 K' に対して

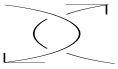
$$w(K) \quad = \quad w(\ \ ) + w(K')$$

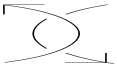
$$= \quad (1-1) + w(K')$$

$$= \quad w(K')$$

 に他の向きを与えた場合も

( の捻り数) $= 0$

( の捻り数)$= 0$

( の捻り数)$=0$

なので, 上と同じようにして Reidemeister 移動 II によって捻り数は不変であることが判る.

次に Reidemeister 移動 III に関する不変性を調べる.

 を含む有向絡み目 K と, その部分に Reidemeister 移動 III をほどこした有向絡み目 K' に対して

$$\begin{aligned} w(K) &= w(\text{  }) + w(K \text{ から} \text{  } \text{を除いた残りの図形} \beta) \\ &= (1+1-1) + w(\beta) \\ &= 1 + w(\beta) \end{aligned}$$

$$\begin{aligned} w(K') &= w(\text{  }) + w(K' \text{から} \text{  } \text{を除いた残りの図形} \gamma) \\ &= (1+1-1) + w(\gamma) \\ &= 1 + w(\gamma) \end{aligned}$$

明らかに $w(\beta) = w(\gamma)$ なので $w(K) = w(K')$ となる.

 に他の向きを与えた場合も

$$\begin{cases} (\text{  } \text{の捻り数}) = (-1-1-1) = -3 \\ (\text{  } \text{の捻り数}) = (-1-1-1) = -3 \end{cases}$$

$$\begin{cases} (\text{  }\text{の捻り数}) = (1-1+1) = 1 \\ (\text{  }\text{の捻り数}) = (1-1+1) = 1 \end{cases}$$

$$\begin{cases} (\text{  }\text{の捻り数}) = (-1+1+1) = 1 \\ (\text{  }\text{の捻り数}) = (-1+1+1) = 1 \end{cases}$$

$$\begin{cases} (\text{  }\text{の捻り数}) = (-1+1+1) = 1 \\ (\text{  }\text{の捻り数}) = (1+1-1) = 1 \end{cases}$$

$$\begin{cases} (\text{  }\text{の捻り数}) = (1-1+1) = 1 \\ (\text{  }\text{の捻り数}) = (1-1+1) = 1 \end{cases}$$

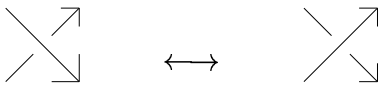
なので上と同じようにして ReidemeisterIII によって捻り数は不変である.

よって捻り数は正則同位に関して不変である.

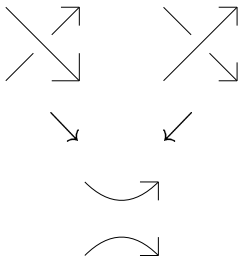
4. Conway 多項式

4.1. 交差交換・分離

有向絡み目の正則表示のある交差点に対して, 上交差線と下交差線を入れかえる操作を交差交換という.



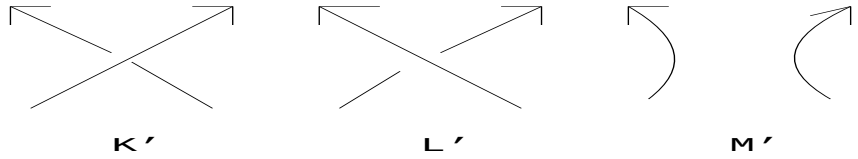
また向きに従って交差点を解消する操作を分離という.



4.2. Conway 多項式

有向絡み目 K に対して, 次の 3 つの公理を用いて変数 z の整係数多項式 $\nabla_K(z)$ を定義することができる.

- 有向絡み目 K と K' が同型ならば, $\nabla_K(z) = \nabla_{K'}(z)$
- K が平凡な時, $\nabla_K(z) = 1$
- 3 つの有向絡み目 K, L, M の正則表示を K', L', M' とする. 但し L' は K' のある交差点を交差交換したもので, M' は同じ交差点を分離したものとする.



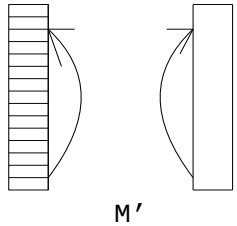
この時

$$\nabla_K(z) - \nabla_L(z) = z\nabla_M(z)$$

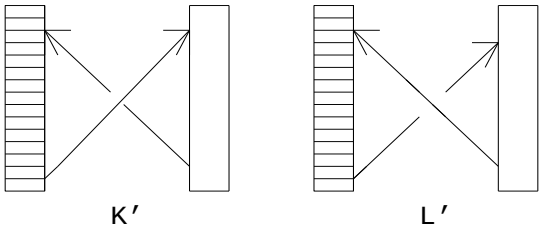
が成り立つ.

この 3 つの公理をみたす $\nabla_K(z)$ は実際に存在して, 有向絡み目の不変量になる. $\nabla_K(z)$ を有向絡み目 K の *Conway* 多項式とよぶ.

有向絡み目 M が分離可能ならば $\nabla_M(z) = 0$ である. 実際 M の正則表示を下の図のように分離した図形 M' とする.



有向絡み目 K, L の正則表示を K', L' とし, 下の図のように定める.



K', L' は明らかに同型な絡み目なので, 1 つ目の公理より

$$\nabla_K(z) = \nabla_L(z)$$

となり,2 つ目の公理より

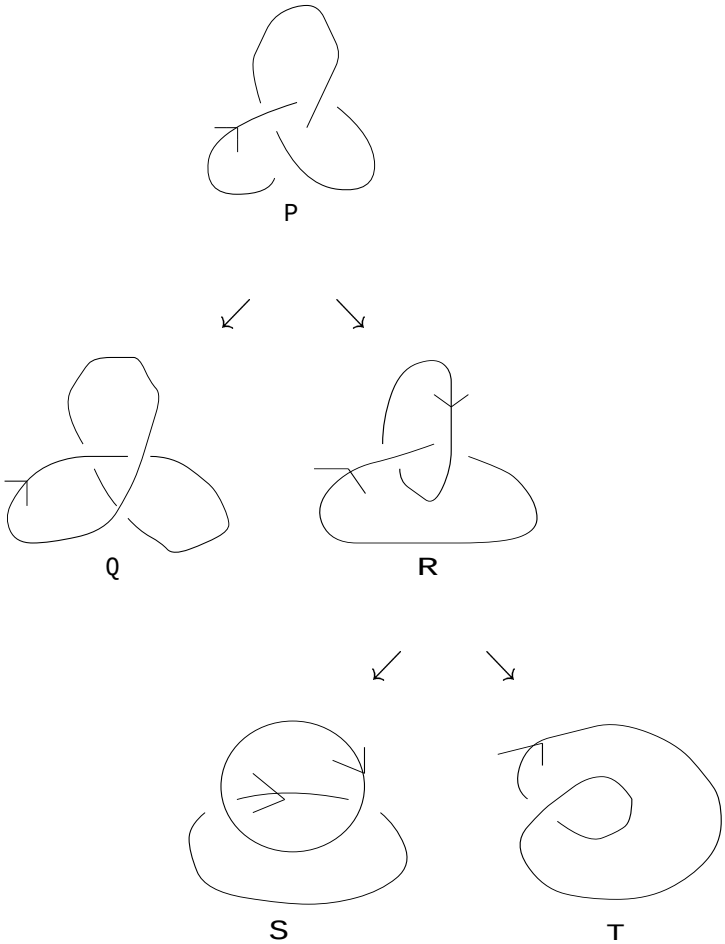
$$0 = \nabla_K(z) - \nabla_L(z) = z\nabla_M(z)$$

従って

$$\nabla_M(z) = 0$$

例えば 

の Conway 多項式は以下のとおり計算できる.



上の図の Q は P の一番下の交差点を交差交換した図形で, R は同じ交差点を分離した図形である. また, S, T も Q, R と同様である.

上の公理を使って Conway 多項式を計算すると

$$\nabla_R(z) - \nabla_S(z) = z\nabla_T(z)$$

S は分離可能なので, $\nabla_S(z) = 0$ となる.

$$\begin{aligned} \nabla_R(z) - 0 &= z \\ \nabla_R(z) &= z \end{aligned}$$

従って

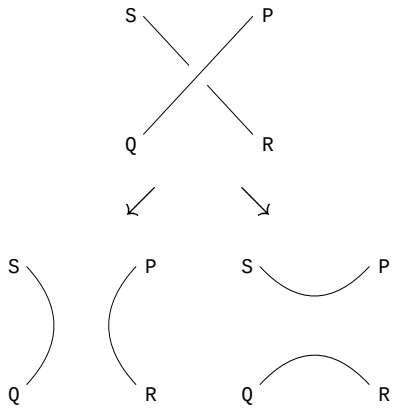
$$\begin{aligned} \nabla_P(z) - \nabla_Q(z) &= z\nabla_R(z) \text{ より} \\ \nabla_P(z) - 1 &= z^2 \\ \nabla_P(z) &= z^2 + 1 \end{aligned}$$

となる.

5. Bracket 多項式 · Jones 多項式

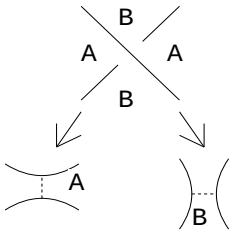
5.1. state

向きを考えない絡み目のある 1 つの交差点 N を考える. N で交差する各辺の 1 部をなす線分上に P, Q と R, S をとる. 但し N の周りに時計回りに P, R, Q, S が現れ, さらに線分 PQ が上交差線となるようにとる. この時 2 線分 PQ と RS を PR と SQ または PS と RQ に置きかえて交差点を解消する操作を分離とよぶ.



交差点 N において PQ を RS に重なるまで反時計回りに回転させた時, PQ が通る N を中心とした 2 つの扇形の領域を A 領域とよび, RS が通る領域を B 領域とよぶ.

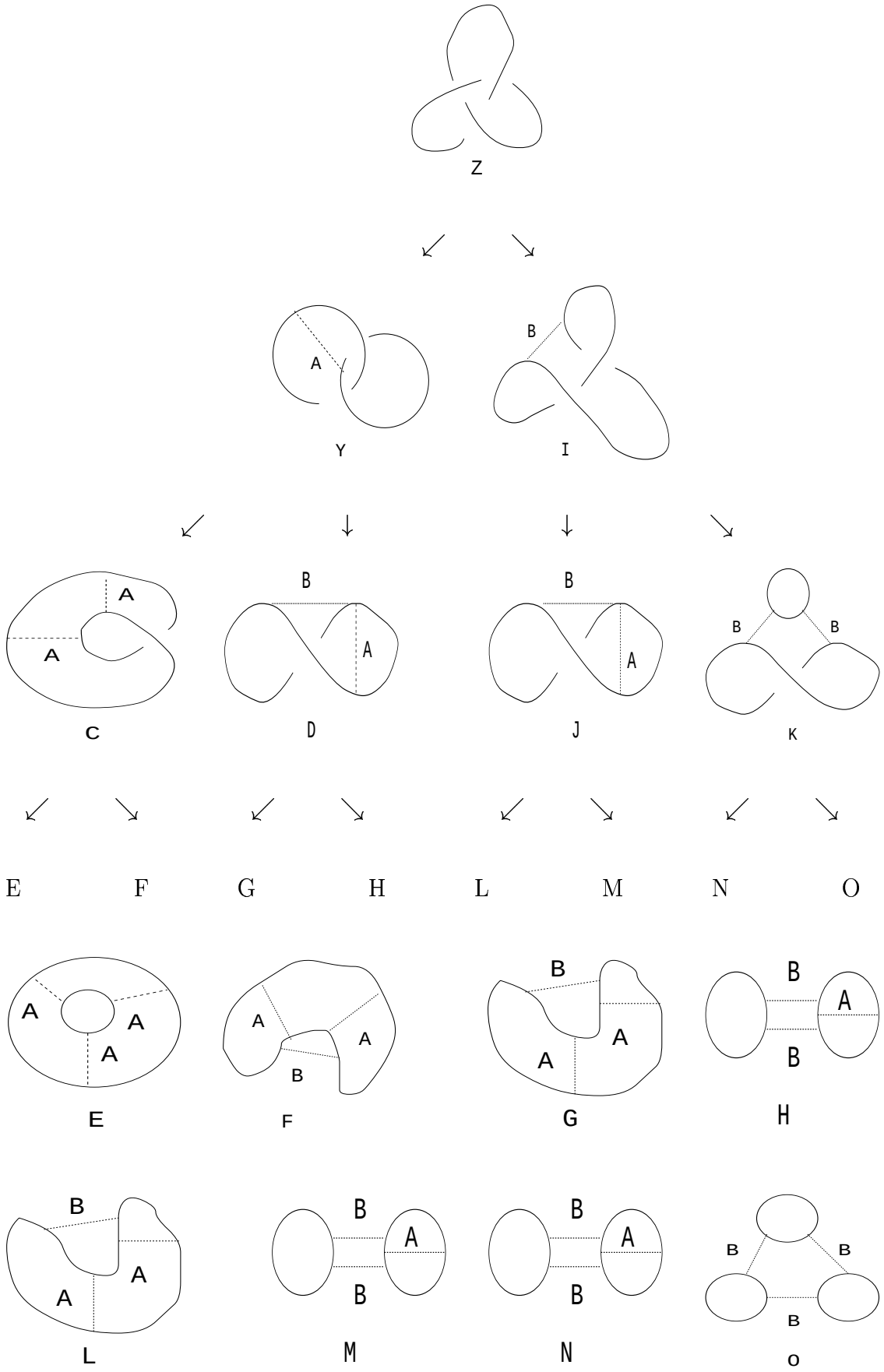
PQ と RS を PR と SQ におきかえるような分離を A 分離, PS と RQ におきかえるような分離を B 分離とよぶ. A 分離の場合は, PR 上の 1 点と SQ 上の 1 点の間を点線でつなぎ, その点線に A というラベルを付ける. B 分離の場合は, PS 上の 1 点と RQ 上の 1 点の間を点線でつなぎその点線に B というラベルを付ける.



全ての交差点において, A 分離または B 分離を適用し, 平凡な絡み目各々に A または B というラベルが付加された図形を $state$ とよぶ.

例えば 

の場合は以下の通りである.



となり, A の state は E, F, G, H, L, M, N, O となる.

5.2. Bracket 多項式

state 全体を使って, 以下のようにして正則同位の不変量が定義できる.

K の state σ に対して, A と B の多項式 $< K \mid \sigma >$ を

$$A^{(\sigma\text{の中のラベル }A\text{ の数})} \times B^{(\sigma\text{の中のラベル }B\text{ の数})}$$

と定める. また, 整数 $\| \sigma \|$ を (σ の絡み目としての成分数-1) と定義する.

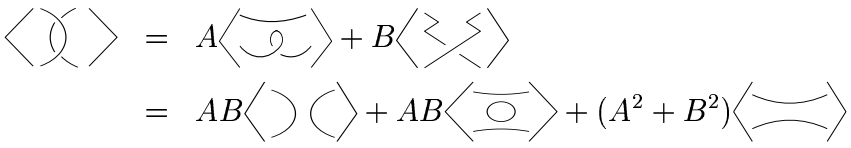
A と B と d の多項式

$$< K >= \sum_{\sigma} < K \mid \sigma > d^{\|\sigma\|}$$

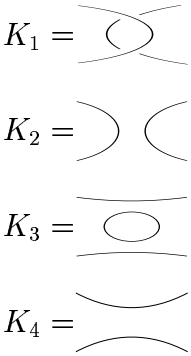
(但し和は state 全体にわたってとる.) を *Bracket* 多項式とよぶ.

この Bracket 多項式が不変量となるように, A, B, d 間にある関係を定めたい, そこで Reidemeister 移動の下での挙動を調べる.

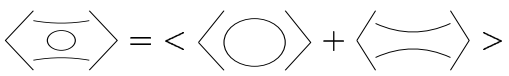
まず正則同位に関しての挙動を調べる. Reidemeister 移動 II については



となる.

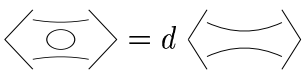


とする. 但し K_1, K_2, K_3, K_4 の残りの部分は同じであるとする.



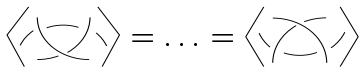
なので Bracket 多項式の定義より

となり



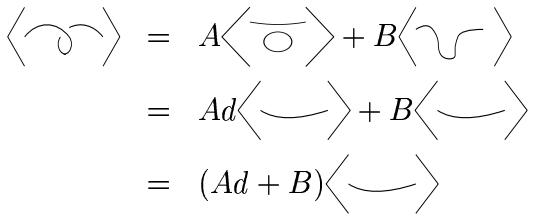
となる. 従って $AB = 1, ABd + A^2 + B^2 = 0$ とすると, Bracket 多項式は Reidemeister 移動 II の下で不変となる.

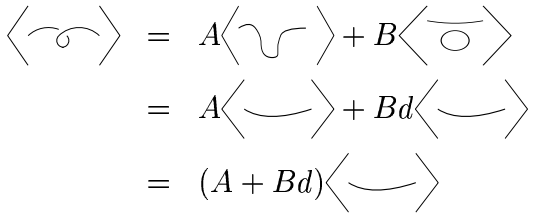
Reidemeister 移動 III について, $B = A^{-1}, d = -A^2 - A^{-2}$ とすると



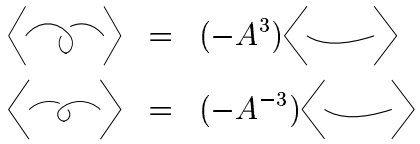
となる. よって $B = A^{-1}, d = -A^2 - A^{-2}$ とすれば, Bracket 多項式は正則同位に関して不変になる.

次に Reidemeister 移動 I についての関係を調べる.





ここで $B = A^{-1}, d = -A^2 - A^{-2}$ だから



となる.

例えば 5.1 の Z の Bracket 多項式は以下のとおり計算できる.

$$
\begin{aligned}
< E > &= Ad^{2-1} \\
&= Ad \\
< F > &= Bd^{1-1} \\
&= B
\end{aligned}
$$

となり

$$
\begin{aligned}
< C > &= A < E > +B < F > \\
&= Ad + B
\end{aligned}
$$

となることが判る.

他も同様に

$$
\begin{aligned}
< D > &= A < G > +B < H > \\
&= A + Bd \\
< J > &= A < L > +B < M > \\
&= A + Bd \\
< K > &= A < N > +B < O > \\
&= Ad + Bd^2 \\
< Y > &= A < C > +B < D > \\
&= A^2 d + 2AB + B^2 d \\
< I > &= A < J > +B < K > \\
&= A^2 + 2ABd + B^2 d^2
\end{aligned}
$$

と計算できるので

$$
< Z > = A < Y > +B < I >
$$

$$
\begin{aligned}
&= A^3d + 3A^2B + 3AB^2d + B^3d^2 \\
&= A^5 - A^{-3} + A^{-7}
\end{aligned}
$$

となる.

5.3. 正規化 Bracket 多項式

Braket 多項式 $< K >$ と $(-A^3)^{-w(K)}$ の積の多項式 $L_K(A) = (-A^3)^{-w(K)} < K > (A)$ を, 有向絡み目 K の正規化 *Braket* 多項式とよぶ. この多項式 L_K が全同位に関する不変量になることを以下に示す.

$w(K)$ も Bracket 多項式もともに正則同位に関して不変なので L_K も正則同位に関して不変である.

ReidemeisterI において正規化 Bracket 多項式の定義により,

$L$$(A)$ $= (-A^3)^{-w($  $)}\langle$$\rangle(A)$

$= (-A^3)^{-(1+w($  $))}(-A^3)\langle$  $\rangle(A)$

$= (-A^3)^{-w($  $)}\langle$$\rangle(A)$

$= L$$(A)$

よって L_K は全同位に関して不変である.

絡み目 K のすべての交差点で上交差線と下交差線を入れかえる絡み目を K の鏡像とよび, K^* と書く.

絡み目 K と K^* が同型ならば *achiral*(アキラル) とよび, 変形できなかった時 *chiral*(カイラル) とよぶ.

絡み目 K の A 領域と B 領域を交換したものが K^* なので Bracket 多項式に対して,

$$
< K > (A) =< K^* > (A^{-1})
$$

が成り立つ.

また, 上交差線と下交差線を入れかえる操作で交差点の符号はかわらないので K と K^* の捻り数は同じであり, 従って $L_{K^*}(A) = L_K(A^{-1})$ となることが判る.

よって $L_K(A) \neq L_K(A^{-1})$ ならば K は K^* とは同型でない. 従って正規化 Bracket 多項式は chiral かどうかの判定に使えることが判る.

例えば 4.2 の P の正規化 Bracket 多項式は以下のとおり計算できる.

$$w(P) = 3$$

P の向きを考えない絡み目は 5.1 の Z となるので,5.2 の計算結果である Z の Bracket 多項式より

$$< P > (A) = -A^5 - A^{-3} + A^{-7}$$

$$\begin{aligned} L_P(A) &= (-A^3)^{-w(P)} < P > (A) \\ &= (-A^3)^{-3}(-A^5 - A^{-3} + A^{-7}) \\ &= A^{-4} + A^{-12} - A^{-16} \end{aligned}$$

5.4. Jones 多項式

Jones 多項式 $V_K(t)$ とは $\sqrt{t}$ の *Laurent* 多項式で次の条件をみたすもののことである.

- 有向絡み目 K が K' と全同位ならば, $V_K(t) = V_{K'}(t)$
- K が平凡な結び目ならば, $V_K = 1$
- $t^{-1}V$  $-tV$  $= (\sqrt{t} - \frac{1}{\sqrt{t}})V$ 

正規化 Bracket 多項式 $L_K(A)$ に対して $A = t^{-1/4}$ を代入して得られる

$$L_K(t^{-1/4}) = (-t^{-3/4})^{-w(K)} < K > (t^{-1/4})$$

は上の条件をみたすことが判る. 実際

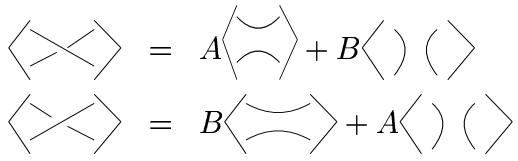
- $V_K(t) = L_K(t^{-1/4}) = L_{K'}(t^{-1/4}) = V_{K'}(t)$
- $K =$  とする.

  $w(K) = 0, \left\langle \right\rangle = 1$ なので

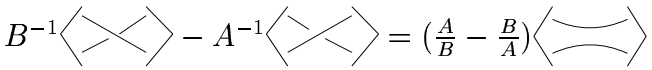
$$\begin{aligned} L_K(t^{-1/4}) &= (-t^{-3/4})^0 < K > \\ &= 1 \end{aligned}$$

  となる.

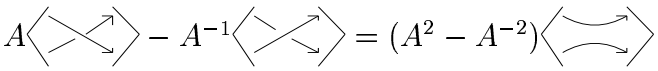
- $B = A^{-1}$ であることに注意すると,Bracket 多項式について次の公式が得られる.

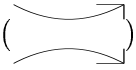


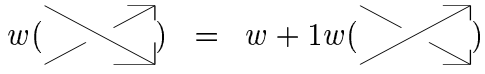
  従って



  よって

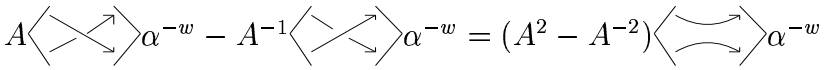


  ここで $w = w($  $)$ とおくと

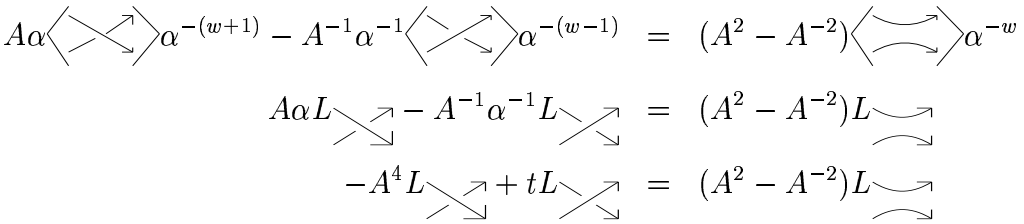


  となる.

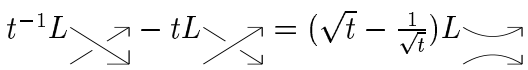
  $\alpha = -A^3$ とすると



従って



ここで $A = t^{-1/4}$ とおくと, 次の等式が導かれる.



つまり Jones 多項式は実際に存在して上のようにして正規化 Bracket 多項式から得られる. さらにこれ以外にないことも判る.

例えば 4.2 の P の Jones 多項式は以下のとおり計算できる.

5.3 より P の正規化 Bracket 多項式は

$$L_P(A) = A^{-4} + A^{-12} - A^{-16}$$

となり, $A = t^{-1/4}$ を代入すると

$$L_P(t^{-1/4}) = t + t^3 - t^4$$

となる.

6. 組み紐と絡み目

6.1. 組み紐の定義・正則表示

1 つの長方形の箱と, その 1 つの側面を決め, 以下固定する.

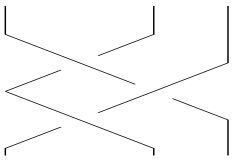
この箱の天井と床それぞれに, その側面に平行に n 個の点を等間隔に並べ, 順に $1, \ldots, n$ の番号を付ける.

組み紐の上で決めた側面への直交射影像が, 絡み目の正則射影と同様な条件をみたしている時, 正則射影とよぶ. また, 組み紐の正則表示も絡み目の正則表示と同様に定める.

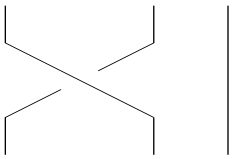
2 つの組み紐 W と W' が紐を切ったりつなげたりせず, さらに天井と床の点を移動したりもせずに 1 方から他方に変形できる時, W と W' は同型であるという.

組み紐 b_1 と b_2 に対し, b_1 の床の点と b_2 の天井の点を番号順につなげると新たな組み紐 b を得る. この b を b_1 と b_2 の積と定める.

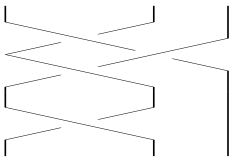
例えば組み紐 b_1 を



とし, 組み紐 b_2 を



すると, b_1 と b_2 の積 b は

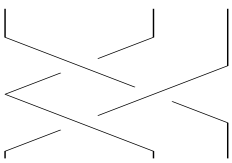


となる.

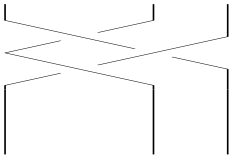
n–組み紐の同型類全体を B_n と表わすと, B_n はこの積に関して組み紐群とよばれる群になる. 実際

- 結合法則が成り立つのは明らかである.
- 交差点が 1 つもない, n 本の真っ直な紐からなる組み紐を 1 とする. 1 は明らかに B_n の単位元となる.

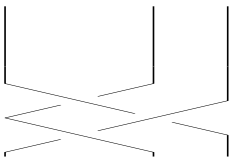
  例えば, 組み紐 b_1 を



とすると, $b_1 1$ は



となり,$1b_1$ は



となるので $b_1 1 = 1 b_1 = b_1$ となることが判る.

- ある組み紐に b 対して b の下に鏡をおき, 鏡にうつった組み紐を b' とする. 明らかに b と b' はお互いの逆元になっている.

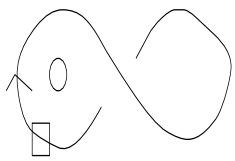
6.2. 組み紐と絡み目の関係

組み紐の天井の番号と同じ床の番号を組み紐の外側で絡まないように結ぶと絡み目になる. これを閉じたブレイドとよぶ.

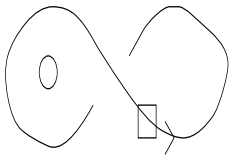
以下のようにして絡み目が全同位によって閉じたブレイドの形に変形できることが判る.

絡み目の紐と紐の間の余白の領域に, ある 1 点を定め, その 1 点を軸とする. 各成分上に 1 点を定め, そこを始点として時計回りに進む. もし反時計回りに回ったらその部分を軸の反対側に移動する. この作業を全ての成分について始点に戻るまで繰り返す. そうすると閉じたブレイドになる.

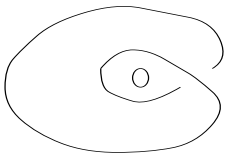
例えば



の中の ∘ を軸とし, □ の中の点を始点とする. 始点から時計回りに進む. そうすると下の図の □ で反時計回りに回ることになる.



そこで下の図のように軸の反対側に移動すると, 閉じたブレイドになる.



7. コンピュータプログラムによる Jones 多項式の計算

7.1. Perl プログラム における組み紐の表現

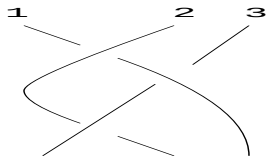
天井の番号 i, $i+1$ の点がそれぞれ床の $i+1$, i の点と結ばれ, 交差点は 1 個だけで, 天井の i と床の $i+1$ の点を結ぶ紐がその交差点における上交差線となっている正則表示をもつ組み紐を σ_i とする.

σ_i

組み紐群 B_n は $\sigma_1, \ldots, \sigma_{n-1}$ で生成される.

組み紐群の元を生成元の積 $\sigma_{i_1}^{\varepsilon_1}, \ldots, \sigma_{i_m}^{\varepsilon_m}$ と表わした時, 整数 $\varepsilon_1 i_1, \ldots, \varepsilon_m i_m$ の配列を組み紐の内部表現とする. 但し $\varepsilon_1, \ldots, \varepsilon_m$ は 1 か-1 である.

例えば



は積 $\sigma_1^{-1}\sigma_2\sigma_1^{-1}$ で表わされ-1,2,-1 の配列 [-1,2,-1] が組み紐の内部表現となる.

7.2. Perl プログラム における有向絡み目の表現

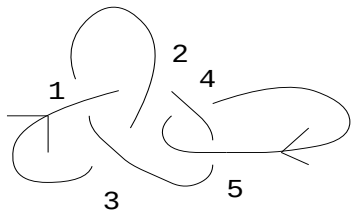
有向絡み目 K に対して, K の正則表示を 1 つ定めその交差点数を $p(\geq 1)$ とする. 各交差点に $1, \ldots, p$ までの数字でラベル付けする. 各成分を上で定めた交差点のラベルを絡み目の向き

に従って並べた配列で表わす. 但しその成分の辺が下交差線である場合はラベルにマイナス (−) の符号を付ける. 各成分の配列の配列を F とする.

各交差点の符号を上で付けたラベルの順に並べた配列を S とする.

F と S の 2 つの組を有向絡み目の内部表現とする.

例えば



は F =[[1,-2,4,-5,3,-1,2,-3],[-4,5]], S =[1,1,1,1,1] となる.

7.3. Perl プログラム における単項式・多項式の表現

単項式は係数, 文字とその巾の配列の配列で表わす.

例えば単項式 $-2xy^2z^3$ は [-2, [x, 1], [y, 2], [z, 3]] と表わされる.

多項式は単項式の配列の配列として表わす.

例えば多項式 $2 + a^{-2} - 2xt^2 + t - 4$ は [[2] , [1 , [a, -2]], [-2, [x, 1], [t, 2]], [1, [t, 1]], [-4]] と表わされる.

7.4. 分散計算システムの構成

ひとつのコンピュータで組み紐を次々に生成させる. 以下このコンピュータをサーバとよぶ.

他の (複数台の) コンピュータでサーバから組み紐ひとつを受け取り, 有向絡み目に変換し, Jones 多項式を計算する. この計算をするコンピュータをクライアントとよぶことにする.

その計算結果をサーバに報告し, サーバは組み紐とその計算結果である Jones 多項式を記録し続ける.

7.5. 組み紐の生成

正整数 k を 1 つ定める.

サーバで, あるひとつのプロセスを起動し紐数 $n < k$, 交差点数 $k - n$ の組み紐を全て生成する. こうして生成された組み紐は組み紐の内部表現を使い,

/紐数/s$\varepsilon_1 i_1$/s$\varepsilon_2 i_2$/, . . .,/s$\varepsilon_m i_m$/

の形で表わし,1 つのファイルに保存しておく.

7.6. クライアントからサーバへ組み紐の問い合わせ

サーバでは, クライアントから接続要求があるごとに 1 つプロセスが起動され, ファイルに保存された組み紐のうちまだ計算されていないものをひとつ読み出し, クライアントへそれを知らせる. クライアントはサーバとの接続を一旦切る.

この時サーバは組み紐を保存しているファイルの何バイト目 (ファイルポジション) までの組み紐をクライアントに知らせたかを別のファイルに記録しておく. こうすることで, 次にクライアントから要求がきた時にファイルからまだ計算されていない組み紐を検出し, クライアントに渡すことができる.

また複数台のクライアントから同時に問い合せがきても同じ組み紐を渡さないように以下のような排他制御をする. ファイルポジションを書き込んであるファイルはその内容を読み出して新しい内容に書きかえるまで排他ロックしておく.

7.7. 組み紐から Jones 多項式の変換

クライアントがサーバから組み紐を受け取り 6.2. で説明した関係を用いて有向絡み目に変換する.

変換された有向絡み目から 5.1 で説明した定義に従って全ての state を求めそれらの state から Bracket 多項式を計算する.

各交差点の符号から捻り数 $w(K)$ を計算し, Bracket 多項式の計算結果から 5.3. で説明した関係を用いて正規化 Bracket 多項式を計算する.

正規化 Bracket 多項式の計算結果から 5.4. で説明した関係を用いて Jones 多項式を計算する.

クライアントはサーバに接続し, サーバから受けとった組み紐とその変換結果である有向絡み目と計算した Jones 多項式を渡す. この時サーバ側では 1 つプロセスが起動され, 受け取ったものを所定のファイルに記録する.

8. 結果

8.1. Jones 多項式の計算の途中結果

- 上で説明したシステムを使って今までに計算した組み紐は, 紐数を i, 交差点数を t とした時に $i + t < 10$, $i \geq 3$, $t \geq 1$ の場合である.
- 結び目理論入門の本の 161 ページ〜の付録 C にある素な結び目・絡み目の Jones 多項式を計算した.

9. プログラム

最後に付録として, Jones 多項式を計算するために用いたプログラムと, 別の不変量として Conway 多項式を計算するために用いたプログラムを載せておく.

10. 目標

10.1. 今後の目標

今後は, 計算済みの絡み目が自明かどうかを調べるために, 他の不変量も計算できるようにし, 計算できる組み紐の数を増やすために, サーバ側の作業も分散できるよう考えていきたい.

11. 付録

11.1. Jones 多項式を計算するために用いたプログラム

```
#!/usr/bin/perl

#  組み紐を次々に生成し,
#  生成した組み紐をディレクトリとして作り,
#  またファイルに保存していくプログラム.

#コマンドラインで k の初期値と終了条件を設定する;
  ($early,$end) = splice(@ARGV, 0, 2);
  &link_braid($early,$end);

#組み紐を生成するために, このファイルを起動する事により;
#/var/links の下に/紐数/s と組み紐の内部表現で表わされたもの/
#というディレクトリを作る;
#/var/links/link のファイルに,
#/紐数/s と組み紐の内部表現で表わされたもの/を書き込む;

#k=4,n=3(1 から数え上げる) が初期値である, 但し k-n=1 となった時, 最小値である;
sub link_braid {
  local($early,$end) = @_;
  local($braidele,$origele,$origk,$orign);

  open(OUT, ">>/var/links/link");
  select((select(OUT),$|=1)[0]);

#/var/links/dustbraid が 0 バイト (空ファイル) でなかったら
```

```
#停電があり組み紐を途中でも生成させる;
  if (-s "/var/links/dustbraid") {
    open(OUT0, "/var/links/dustbraid");
    $braidele = <OUT0>;
    chop($braidele);
    close(OUT0);
    @origkn = split(/,/,$braidele);
    &begin($origkn[0],$origkn[1]);
  }else {
    &begin($early,3,$end);
  }
  close(OUT);
}

#初期値:k=4,n=3;
#k-n=>1 となるまで n を増やし, 紐数が増えると交差点数を減らしていく;
#k-n<1(交差点数が 1 未満) となったら k を増やし,
#n を初期値に戻し, 紐数が少ない時に交差点数を増やし計算していく;
#k の値の最後を決めてハードディスクの容量と相談する;
sub begin {
  local($k,$n,$end) = @_;

  do {
    if ($k > $end) {
      exit;
    }
    else {
      while($k-$n >= 1) {
```

```
        mkdir("/var/links/$n",0755);
        &number3($k,$n);
        ++$n;
      }
      $k=$k+1;
      $n=3;
    }
  } while (1) ;
}

sub begin_cross {
  local($k,$n,@string) = @_;
  local($u,$jones);

#停電対策として,$k と$n をに書き込む (上書き);
  open(OUT2, ">/var/links/dustbraid");
  print OUT2 "$k,$n\n";
  close(OUT2);

#ファイルに記入された紐数が (0 から数えて)n-1,
#交差点数が k-n の時の braid を 1 つずつ考えていく;
  $jones = "/var/links/$n/";
  print OUT "$n/";

#$r[$g] の各成分を 1 つずつ考えて, その成分が x で x が正ならば,s+x,
#負ならば s-x とするディレクトリを作る;
  for ($u=0; $u<@string; ++$u) {
    if ($string[$u] > 0) {
```

```
      mkdir($jones . "s+$string[$u]/",0755);
      $jones .= "s+$string[$u]/";
      print OUT "s+$string[$u]/";
    }
    else {
      mkdir($jones . "s$string[$u]",0755);
      $jones .= "s$string[$u]/";
      print OUT "s$string[$u]/";
    }
  }
  print OUT "\n";
}

#$n-1 = 成分数-1;
#$k-$n = 交差点の数;
sub number3 {
  local($k,$n) = splice(@_, 0, 2);

  if (@_ < $k-$n) {
    local($i);
    for ($i = 1 ; $i <= $n-1 ; ++$i) {
      &number3($k, $n, @_, $i);
      &number3($k, $n, @_, -$i);
    }
  }
  else {
    &begin_cross($k,$n,@_);
  }
```

```
}

1;

#!/usr/bin/perl

#  サーバ側に問い合せをしまだ計算されていない組み紐を受け取り,
#  有向絡み目に変換し,
#  Jones 多項式の計算結果を返すためにサーバ側に問い合せをする
#  プログラム.

require "/home/kanasama/jonespro/jones.pl";
use IO::Socket;
&non_ele;

sub non_ele {
  local($dust);

#/var/links/dust が 0 バイト (空ファイル) でなかったら停電があり,
#まだ計算されていない組み紐がある意.;
  if (-s "/var/links/dust") {
    open(DUST0, "/var/links/dust");
    $dust = <DUST0>;
    chop($dust);
    close(DUST0);

#組み紐$dust を計算していく;
    &make_jones($dust);
```

```
  }
  else {
    &loopjones;
  }
}

sub loopjones {
  local($i);

  for ($i=0; ;++$i) {
    &jones_socket;
  }
}

#&make_jones("4/s+1/");

sub jones_socket {
  local($socket,$origjones);

  do {
    $soket = IO::Socket::INET->new(PeerAddr => ´localhost´,
                                   PeerPort => ´jonesq´,
                                   Proto    => ´tcp´);
    $origjones = <$soket>;
    $socket->close;
    chop($origjones);
```

#$origjones が空だったら, 生成する側が生成を止めていることなのでもう終る;

```
    if ($origjones eq ´´) {
      sleep 86400;
    }
  } while ($origjones eq ´´) ;

#mee から受け取った組み紐を停電対策としてファイルに保管しておく;
  open(DUST1, ">/var/links/dust");
  print DUST1 "$origjones";
  close(DUST1);
  &make_jones($origjones);
}

sub make_jones {
  local($origjones) = @_;
  local(@origlink,@links,@jone,@link,$i);

  @origlink = split(/\//,$origjones);
#/,+,s は除く;
  push(@links,@origlink[1 .. $#origlink]);
  for($i=0; $i<@links; ++$i) {
    local(@link);
    @link = split(/s[\+]*/,@links[$i]);
    push(@jone,@link[1]);
  }
#紐数は 0 から数えるので-1 を加える;
  &make_link(@origlink[0]-1,[@jone],$origjones);
}
```

```
sub make_link {
  local($y,$r,$origjones) = @_;
  local($sign,$jonesa,$answer,$jonespoly);

  $sign = &braid_sign($r);
  $answer = &braid(0,0,$r,0,$y,$sign,[(0) x $y],[],[]);
  $jonespoly = &jones_answer($answer,$sign);
  $jonesa = IO::Socket::INET->new(PeerAddr => ´localhost´,
                                  PeerPort => ´9009´,
                                  Proto    => ´tcp´);
  unless (defined $jonesa) {
    print STDERR "error: $!\n";
    die;
  }
  print $jonesa "$origjones\n";
  &print_matrix_out([$sign],$jonesa);
  &print_matrix_out($answer,$jonesa);
  &print_out_out($jonespoly,$jonesa);
  $jonesa->close;

#停電対策としてファイルに保管しておいた組み紐を消す(ファイルを空にする);
  open(DUST2, ">/var/links/dust");
  close(DUST2);
  select(undef,undef,undef,0.2);
}

sub braid {
  local($i,$g,$link,$q,$t,$sign,$nth,$poly,$answer) = @_;
```

```
  local($new_i,$new_poly,$new_answer,$r,$u);


#まだ通ってない紐がある;
  if ($q == 0) {
    if ($$nth[$i] == 0) {
      $$nth[$i] = 1;
      ($new_i,$g,$link,$q,$t,$sign,$nth,$new_poly,$new_answer)
= &braid_loop($i,$g,$link,$q,$t,$sign,$nth,$poly,$answer);
      if ($q == $#$link) {
&braid($new_i,$g,$link,0,$t,$sign,$nth,$new_poly,$new_answer);
      }
      else{
&braid($new_i,$g,$link,$q+1,$t,$sign,$nth,$new_poly,$new_answer);
      }
    }
    else {
      if ($i == $g){
        push(@$answer,$poly);
        $poly = [];
      }
      $r = &nth_0($nth,$t);
      if ($r) {
&braid($r,$r,$link,0,$t,$sign,$nth,$poly,$answer);
      }
      else {
        $answer;
      }
    }
```

```
  }
  elsif ($q == $#$link) {
    ($new_i,$g,$link,$q,$t,$sign,$nth,$new_poly,$new_answer)
      = &braid_loop($i,$g,$link,$q,$t,$sign,$nth,$poly,$answer);
    &braid($new_i,$g,$link,0,$t,$sign,$nth,$new_poly,$new_answer);
  }
  else {
    ($new_i,$g,$link,$q,$t,$sign,$nth,$new_poly,$new_answer)
      = &braid_loop($i,$g,$link,$q,$t,$sign,$nth,$poly,$answer);
    &braid($new_i,$g,$link,$q+1,$t,$sign,$nth,$new_poly,$new_answer);
  }
}

sub braid_loop {
  local($i,$g,$link,$q,$t,$sign,$nth,$poly,$answer) = @_;
  local($x);

  if (&abs($$link[$q]) == $i+1) {
    if ($$sign[$q] > 0) {
      push(@$poly,-($q+1));
    }
    else {
      push(@$poly,$q+1);
    }
    $x = $i+1;
  }
  elsif (&abs($$link[$q]) == $i) {
    if ($$sign[$q] > 0) {
```

```
      push(@$poly,$q+1);
    }
    else {
      push(@$poly,-($q+1));
    }
    $x = $i-1;
  }
  else {
    $x = $i;
  }
  ($x,$g,$link,$q,$t,$sign,$nth,$poly,$answer);
}

#$$nth が 1 では無く,0 を探す;
sub nth_0 {
  local($nth,$t) = @_;
  local($r);

  for ($r=0; $r<$t+1; ++$r) {
    if ($$nth[$r] == 0) {
      return $r;
    }
  }
  return;
}

#符号を決める;
sub braid_sign {
```

```
  local($link) = @_;
  local(@flag,$i);

  $last = @$link;
  @flag = (1) x $last;
  for ($i=0; $i<@$link; ++$i) {
    if ($$link[$i] < 0) {
      @flag[$i] = -@flag[$i];
    }
  }
  [@flag];
}

#$link の絶対位置を返す;
sub abs {
  local($link) = @_;

  if ($link < 0) {
    -$link;
  }else {
    $link;
  }
}

sub print_matrix_out {
  local($poly,$jonesa) = @_;

  print $jonesa "[";
```

```
  foreach (@{$poly}) {
    print $jonesa "[";
    foreach (@$_) {
      print $jonesa "$_, ";
    }
    print $jonesa "],";
  }
  print $jonesa "]\n";
}



sub print_out_out {
  local($poly,$jonesa) = @_;
  local($sep);

  print $jonesa "[";
  foreach (@$poly) {
    print $jonesa $sep;
    &print_mono_out($_,$jonesa);
    $sep = ´,´;
  }
  print $jonesa "]\n";
}

sub print_mono_out {
  local($mono,$jonesa) = @_;
  local($c, @vars) = @$mono;
  local($") = ´, ´;
```

```
  print $jonesa "[$c";
  foreach (@vars) {
    print $jonesa ", [@$_]";
  }
  print $jonesa "]";
}

1;

#!/usr/bin/perl

#  他のクライアントから組み紐を受け取りの問い合せがきたら
#  このプロセスが起動し,
#  組み紐を重複して渡さないために lock する.

use Fcntl ´:flock´;
#import LOCK_* constants;
&search_dbm;

sub search_dbm {
  local($lines,$byte);

  open(JONES, "+</var/links/fpos");
  flock(JONES,LOCK_EX);
  open(LINKS, "/var/links/link");
  $lines = <JONES>;
  seek(LINKS,$lines,0);
```

```
  $_ = <LINKS>;
  $byte = tell LINKS;
  seek(JONES,0,0);
  truncate JONES,0;
  print JONES "$byte\n";
  close(LINKS);
  close(JONES);
  print "$_";
  select(undef,undef,undef,0.2);
}


#!/usr/bin/perl

#  有向絡み目から
#  Jones 多項式を計算するプログラム

require "/home/kanasama/lib/poly.pl";
require "/home/kanasama/jonespro/asmoothing.pl";
require "/home/kanasama/jonespro/bsmoothing.pl";

sub jones_answer {
  local($link,$sign) = @_;
  local($origsign,$poly1,$writhe,$number);
  local($jones,$jonespoly,$poly2,$y,$orignum);
  @{$origsign} = @{$sign};
  $poly1 = &say_all($link,$sign);

#Bracket 多項式の値 (<K>とする);
```

```
#  print "origblacket is\n";
#  &print_out($poly1);
  $poly2 = &d_substitution($poly1,d,[[-1,[a,2]],[-1,[a,-2]]]);

#<K>の値に d=-a^2-a^-2 を代入したもの;
#  print "blacket is\n";
#  &print_out($poly2);
  for($y=0; $y<@{$origsign}; $y++) {
    $writhe += ${$origsign}[$y];
  }
  $orignum=1;
  if ($writhe < 0) {
    $t = -$writhe;
  }else {
    $t = $writhe;
  }
  for ($n=1; $n<$t+1; ++$n) {
      $orignum = -1*$orignum;
  }
  $number = 3 * (-$writhe) + 0 ;
  $jones = &poly_multiple($poly2,[[$orignum,[a,$number]]]);

#正規化 Bracket 多項式の値;
#  print "normalized blacket is\n";
#  &print_out($jones);
#Jones 多項式の値;
   $jonespoly = &t_substitution($jones,a,[[1,[t,-1/4]]]);
   $jonespoly;
```

```
}


sub say_all {
  local($link,$sign) = @_;
  local($i,$p,$poly,$x,$b,@l);


 $x=0;
  foreach $b (@{$link}) {
    $x += @{$b};
  }
  if ($x != 0) {
    @l = @{$link};
    until (@{$l[0]}) {
      splice(@l,0,1);
      push(@l,[]);
    }
    $link = [@l];
    $i = ${$link}[0][0];
    if ($i<0) {
      $p = -$i;
    }else {
      $p = $i;
    }
  }
  $poly = &link_program($link,$i,$sign,$p);
  $poly;
}
```

```
sub link_program {
  local($link,$i,$sign,$p) = @_;

  if ($i) {
    local($poly1,$link1,$poly2,$new_sign1,$new_sign2,$link2);
    local($origsign,$blacket,$blacket1,$blacket2,$number,$link9);
    local($origsign0);
    local($t,$y);

    @{$origsign} = @{$sign};
    @{$origsign0} = @{$sign};
    $t = ${$sign}[$p-1];
    ($poly1,$new_sign1)
    = &aplus_bminus_smoothing($link,$i,$origsign0,$p);
    $link1 = &say_all($poly1,$new_sign1);

    $y = ${$origsign}[$p-1];
    ($poly2,$new_sign2)
    = &aminus_bplus_smoothing($link,$i,$origsign,$p);
    $link2 = &say_all($poly2,$new_sign2);

    if ($t == 1 || $y == 1) {
      $blacket1 = &poly_var_multiple($link1,[´a´,1]);
      $blacket2 = &poly_var_multiple($link2,[´a´,-1]);
    }else {
      $blacket1 = &poly_var_multiple($link1,[´a´,-1]);
      $blacket2 = &poly_var_multiple($link2,[´a´,1]);
    }
```

```
    $link9 = &poly_plus($blacket1,$blacket2);
    $link9;
  }else {
    $number = @{$link};
    [[1, [´d´,$number-1]]];
  }
}

sub print_out {
  local($poly) = @_;
  local($sep);

  print "[";
  foreach (@$poly) {
    print $sep;
    &print_mono($_);
    $sep = ´, ´;
  }
  print "]\n";
}

sub print_mono {
  local($mono) = @_;
  local($c, @vars) = @$mono;
  local($") = ´, ´;

  print "[$c";
  foreach (@vars) {
```

```
    print ", [@$_]";
  }
  print "]\n";
}

1;

#!/usr/bin/perl

#  A 分離の操作をするプログラム

sub aplus_bminus_smoothing {
  local($link,$i,$sign,$p) = @_;
  local($column,$e,$d,$q,$c,$t,$origsign);
  local(@columns,@vertex1,@vertex2);

  for ($q=0; $q<@{$link}; $q++){
    $column = @{${$link}[$q]};
    push (@columns,$column);
  }
  $c = 1;
  for ($e=0; $e<@{$link};){
    for ($d=$c; $d<$columns[$e]; $d++){
      push(@vertex1,${$link}[$e][$d]);
      if ($vertex1[$#vertex1] == -$i) {
        if ($e == 0) {
  splice(@vertex1,-1,1);
  push (@vertex2,[@vertex1]);
```

```
  @vertex1 = ();
}else{
  splice(@vertex1,-1,1);
  push(@{${\@vertex2}[0]},@{${$link}[$e]}[($d + 1) ..
  $columns[$e]-1],@vertex1);
          @vertex1 = ();
          last;
}
      }
    }
    if (@vertex1) {
      push (@vertex2,[@vertex1]);
    }elsif ($columns[$e] == 0) {
      push(@vertex2,[]);
    }elsif ($e == 0 && ${$link}[$e][$columns[$e]-1] == -$i) {
      push (@vertex2,[@vertex1]);
    }
    @vertex1 = ();
    $e++;
    $c = 0;
  }
  splice(@{$sign},$p-1,1,0);
  ([@vertex2],$sign);
}


1;


#!/usr/bin/perl
```

```
#  B 分離の操作をするプログラム


sub aminus_bplus_smoothing {
  local($link,$i,$sign,$p) = @_;
  local($column,$e,$d,$q,$c,$r,$t,$w,$h,$j,$f,$cut);
  local(@columns,@vertex1,@vertex2,@vertex4,@vertex3,@var,@var1);


  for ($q=0; $q<@{$link}; $q++){
    $column = @{${$link}[$q]};
    push (@columns,$column);
  }
  $c = 1;
  for ($e=0; $e<@{$link};){
    for ($d=$c; $d<$columns[$e]; $d++){
      push(@vertex1,${$link}[$e][$d]);
      if ($vertex1[$#vertex1] == -$i) {
        if ($e == 0) {
          splice(@vertex1,-1,1);
          @vertex3 =
          reverse (@{${$link}[0]}[($d + 1) .. $columns[0]-1]);
          push (@vertex1,@vertex3);
          push (@vertex2,[@vertex1]);
          if (@vertex3) {
            for ($r=0; $r<@vertex3; $r++) {
              if ($vertex3[$r] < 0) {
                $vertex3[$r] = $vertex3[$r]*(-1);
              }else{
```

```
        $vertex3[$r] = $vertex3[$r];
      }
        ${$sign}[$vertex3[$r]-1] =
        ${$sign}[$vertex3[$r]-1]*(-1);
      }
    }
    @vertex1 = ();
    last;
 }else{
   splice(@vertex1,-1,1);
   @var = reverse(@vertex1);
   @var1 = reverse (@{${$link}[$e]}
   [($d + 1) .. $columns[$e]-1]);
   push(@var,@var1);
   push(@{${\@vertex2}[0]},@var);
   if (@var) {
     for ($t=0; $t<@var; $t++) {
       if ($var[$t] < 0) {
         $var[$t] = $var[$t]*(-1);
       }else{
         $var[$t] = $var[$t];
       }
       ${$sign}[$var[$t]-1] = ${$sign}[$var[$t]-1]*(-1);
     }
   }
   @vertex1 = ();
   last;
}
```

```
      }
    }
    if (@vertex1) {
      push (@vertex2,[@vertex1]);
    }elsif ($columns[$e] == 0) {
      push(@vertex2,[]);
    }
    @vertex1 = ();
    $e++;
    $c = 0;
  }
  splice(@{$sign},$p-1,1,0);
  ([@vertex2],$sign);
}

1;
```

```
#!/usr/bin/perl

#  単項式や多項式を計算するプログラム

#多項式や単項式の中を簡単にする操作;

sub poly_normalize {
  local($poly) = @_;
  local($mono,$c,$var1,$var2,$row);
  local(@poly,@new_mono,@mono);
```

```
foreach $mono (@{$poly}){
  ($c,@mono) = @{$mono};
  @new_mono = ($c);
  if (@mono){
    ($var1,@mono) = sort alphavet @mono;
    while (@mono){
      $var2 = shift (@mono);
      if (${$var1}[0] eq ${$var2}[0]){
        $var1 = [${$var1}[0],${$var1}[1] + ${$var2}[1]];
      }else{
        push (@new_mono,$var1);
        $var1 = $var2;
      }
    }
    push(@new_mono, $var1);
  }
  push (@poly,[@new_mono]);
}
if (@poly > 0) {
  local(@new,@rest);
  local($mono1,$mono2);
  ($mono1, @rest) = sort {&cmp_sorted_mono($a,$b);} @poly;
  while (@rest) {
    $mono2 = shift(@rest);
    if (${$mono1}[0] == 0){
      $mono1 = $mono2;
    }elsif(!&cmp_sorted_mono($mono1, $mono2)) {
      $mono1 = [${$mono1}[0] + ${$mono2}[0],
```

```
        @{$mono1}[1 .. $#$mono1]];
      }
      else {
        push(@new, $mono1);
        $mono1 = $mono2;
      }
    }
    if (${$mono1}[0] == 0){
      $poly = [@new];
    }else{
      $poly = [@new, $mono1];
    }
  }
   else {
    $poly = [];
  }
  $poly;
}

#2 つの単項式を受け取る.
sub cmp_sorted_mono {
  local($mono1, $mono2) = @_;
  local(@m1) = @{$mono1}[1..$#{$mono1}];
  local(@m2) = @{$mono2}[1..$#{$mono2}];
  local($i, $result);

  for ($i=0; $i<@m1; $i++){
    return 1 unless ($i < @m2);
```

```
    $result = $m1[$i][0] cmp $m2[$i][0];
    return $result if ($result);
    $result = $m1[$i][1] <=> $m2[$i][1];
    return $result if ($result);
  }
  @m1 <=> @m2;
}

#配列を受け取り, 数値の大小によって配列をソートする;
sub number { ${$a}[1] <=> ${$b}[1]; }

#配列を受け取り, 文字列の大小によって配列をソートする;
sub alphavet { ${$a}[0] cmp ${$b}[0]; }

#1 つの多項式と 1 つの単項式を受け取り, 掛け合せる操作;
sub poly_var_multiple {
  local($poly) = shift;
  local(@poly, $mono, @mono);

  foreach $mono (@{$poly}) {
    @mono = @{$mono};
    foreach $var (@_) {
      push(@mono, [@{$var}]);
    }
    push(@poly,[@mono]);
  }
  &poly_normalize(\@poly);
}
```

```
#1 つの多項式と複数の (1 つでもよい) 単項式を受け取り, 全てを掛け合せる操作;
sub poly_mono_multiple {
  local($poly) = shift;
  local(@poly2,$i,$c,$a,$y,@number,$b,$mono,@result,@vars);

  foreach $mono (@_) {
    push(@number,${$mono}[0]);
    for ($i=1; $i<@{$mono}; $i++){
      push(@poly2,[@{${$mono}[$i]}]);
    }
  }
  for ($y=0; $y<@{$poly}; $y++) {
    ($c, @vars) = @{${$poly}[$y]};
    foreach $b (@number) {
      $c *= $b;
    }
    push(@result, [$c, @vars]);
  }
  &poly_var_multiple([@result],@poly2);
}

sub poly_multiple {
  local($poly,$poly3,@poly7) = @_;
  local($poly6,$z,$poly2,$b,@poly4,$poly5,@poly8,$poly9,$poly10);

  for ($x=0; $x<@{$poly3}; $x++){
    local(@mono2);
```

```
    push(@mono2,@{${$poly3}[$x]});
    $poly2 = &poly_mono_multiple($poly,[@mono2]);
    foreach $b (@{$poly2}) {
      push(@poly4,$b);
    }
  }
  if (@poly7) {
    if (@{$poly3}) {
      for ($i=1; $i<@poly7; $i++) {
        push(@poly8,$poly7[$i]);
      }
      $poly9 = &poly_multiple([@poly4],$poly7[0],@poly8);
      $poly5 = &poly_normalize($poly9);
    }else{
      $poly10 = &poly_multiple([@poly4],$poly7[0],[]);
      $poly5 = &poly_normalize($poly10);
    }
  }else {
    $poly5 = &poly_normalize([@poly4]);
  }
  $poly5;
}
```

#複数の多項式を受け取り, 全てを足す操作;

```
sub poly_plus {
  local(@poly1) = @_;
  local(@poly,$x,$y);
```

```
  for ($x=0; $x<@poly1; $x++) {
    push(@poly,@{$poly1[$x]});
  }
  &poly_normalize([@poly]);
}

#1 つの多項式と定数と 1 つの多項式を受け取り,
#1 つ目の多項式の中の定数に 3 つ目の多項式を代入する操作;
sub d_substitution {
  local($poly,$t,$mono1) = @_;
  local($e,@mono,$c,$x,@poly10,$poly9,$poly0);

  for ($e=0; $e<@{$poly}; $e++) {
    ($c,@mono) = @{${$poly}[$e]};
    @poly5 = ();
    for ($x=0; $x<@mono; $x++) {
      if ($mono[$x][0] eq $t) {
        push(@poly5,($mono1) x $mono[$x][1]);
        splice(@mono,$x,1);
      }
    }
    unless (@poly5) {
      push(@poly10,[[$c,@mono]]);
    }else {
      $poly9 = &poly_multiple([[$c,@mono]],@poly5);
      push(@poly10,$poly9);
    }
```

```
  }
  $poly0 = &poly_plus(@poly10);
  $poly0;
}

sub t_substitution {
  local($poly,$t,$mono1) = @_;
  local($e,@mono,$c,$x,@poly10,$poly9,$poly0,$link,$p,$b);

  for ($e=0; $e<@{$poly}; $e++) {
    ($c,@mono) = @{${$poly}[$e]};
    @poly5 = ();
    for ($x=0; $x<@mono; $x++) {
      if ($mono[$x][0] eq $t) {
        if ($mono[$x][1] < 0) {
          $p = -$mono[$x][1];
          $q = -1;
        }else {
          $p = $mono[$x][1];
          $q = 1;
        }
        push(@poly5,($mono1) x $p);
        splice(@mono,$x,1);
      }
    }
    unless (@poly5) {
      push(@poly10,[[$c,@mono]]);
    }else {
```

```
      $poly9 = &poly_multiple([[$c,@mono]],@poly5);
      ${$poly9}[0][1][1] = $q * ${$poly9}[0][1][1];
      push(@poly10,$poly9);
    }
  }
  $poly0 = &poly_plus(@poly10);
  $poly0;
}

1;

#!/usr/bin/perl

#  サーバ側で
#  クライアントから結果を返す問い合わせがきたら
#  このプロセスが起動し,
#  クライアントから計算結果を受けとる

open(STDERR, ">>/var/tmp/jonesa.error");
select((select(STDERR),$|=1)[0]);
local($jonesbraid,$jonessign,$joneslink,$jonesjones);
$jonesbraid = <STDIN>;
chop($jonesbraid);
$jonessign = <STDIN>;
chop($jonessign);
$joneslink = <STDIN>;
chop($joneslink);
$jonesjones = <STDIN>;
```

```
chop($jonesjones);
open(OUT1,">>/var/links/$jonesbraid/braid");
print OUT1 "$jonesbraid\n";
close(OUT1);
open(OUT2,">>/var/links/$jonesbraid/sign");
print OUT2 "$jonessign\n";
close(OUT2);
open(OUT3,">>/var/links/$jonesbraid/link");
print OUT3 "$joneslink\n";
close(OUT3);
open(OUT4,">>/var/links/$jonesbraid/jones");
print OUT4 "$jonesjones\n";
close(OUT4);
```

11.2. Conway 多項式を計算するために用いたプログラム

```
#!/usr/bin/perl

#  有向絡み目を受け取り
#  Conway 多項式の計算をしていくプログラム

#conway 多項式の計算;
require "/home/kanasama/conwaypro/crossing.pl";
require "/home/kanasama/conwaypro/smoothing.pl";
require "/home/kanasama/conwaypro/poly.pl";

#conway 多項式のコマンド受け取り;
#filename all ´[[-1,2,7,-8,-3,1,-2,3],[-4,5,-6,4,-5,6,8,-7]]´
```

```
#´[-1, -1, -1, -1, -1, -1, 1, 1]´;
#($command,$link,$sign) = splice(@ARGV, 0, 3);

#$link = eval $link;
#$sign = eval $sign;

#if ($command eq "all") {
#  $poly1 = &say_all($link,$sign);
#  &print_out($poly1);
#}elsif ($command eq "crossing"){
#  &say_crossing($link,$sign,@ARGV);
#}elsif ($command eq "smoothing"){
#  &say_smoothing($link,$sign,@ARGV);
#}

sub say_all {
  local($link,$sign) = @_;
  local($k,$c,$d,$f,$h,$u,$i,$column,$z,$j,$g,$a,$b);
  local(@columns,@vertex,@ver,@flag,@dust,@sign);

  for ($k=0; $k<@{$link}; $k++){
    $column = @{${$link}[$k]};
    push (@columns,$column);
  }

  for ($c=0; $c<@{$link}; $c++){
    for ($d=0; $d<$columns[$c]; $d++){
      $f = ${${$link}[$c]}[$d];
```

```
      push(@vertex,$f<0 ? -$f : $f);
    }
  }
  @ver = sort { $a <=> $b; } @vertex;

  @flag = (0)x ($ver[$#ver]);
  for ($h=0; $h<@{$link}; $h++){
    for ($u=0; $u<$columns[$h]; $u++){
      $i = ${${$link}[$h]}[$u];
      $k = $i<0 ? -$i : $i;
      push (@dust,$h,$u) if (!$flag[$k] && $i<0);
      $flag[$k] = 1;
    }
  }

  @sign = @{$sign};
  &crossing_loop($link,scalar(@dust),@dust,@sign);
}

sub crossing_loop {
  local($link,$j) = splice(@_, 0, 2);

  if ($j > 0 ){
    local($x,$y,$orig,$new_link,$poly1,$poly2,$i,$nomal,$poly7);
    local(@dust,@origsign,@new_sign,@sign);


    ($x,$y,@dust) = splice(@_, 0, $j);
```

```
    @sign = @_;

    $orig = &matrix_copy($link);
    @origsign = @sign;

#crossing;
    ($new_link, @new_sign) = &crossing_number($link,$x,$y,@sign);
    $poly1 = &crossing_loop($new_link,scalar(@dust),@dust,@new_sign);

#smoothing;
    ($link, @sign) = &smoothing_number($link,$x,$y,@sign);
    $poly2 = &say_all($link, [@sign]);

    $i = ${${$new_link}[$x]}[$y];

    if ($origsign[$i-1] == 1){
#conway;
     $poly7 = &poly_var_multiple($poly2,[´z´,1]);
     $poly1 = &poly_plus($poly1,$poly7);
    }elsif ($origsign[$i-1] == -1){
#conway;
      $poly8 = &poly_var_multiple($poly2,[´z´,1]);

      for ($t=0; $t<@{$poly8}; $t++){
$u = (-1) * ${$poly8}[$t][0];
        splice(@{${$poly8}[$t]},0,1,$u);
      }
      $poly1 = &poly_plus($poly1,$poly8);
```

```
    }
    $poly1;
  }else{
    $nomal = @{$link};
    if ($nomal <= 1){
      [[1]];
    }elsif ($nomal > 1){
      [[0]];
    }
  }
}

sub poly_plus {
  local($poly1,$poly8) = @_;
  local($poly);

  &poly_normalize([@{$poly1},@{$poly8}]);
}

sub say_crossing {
  local($link,$sign,@arg) = @_;
  local($foo,@sign2);

  print "all is crossing of $arg[0],$arg[1] is";
  ($foo,@sign2) = &crossing_number($link,$arg[0],$arg[1],@sign);
  &print_matrix($foo);
  &print_matrix([@sign2]);
}
```

```
sub say_smoothing {
  local($link,$sign,@arg) = @_;
  local($foo);

  print "all is smoothing of $arg[0],$arg[1] is";
  &smoothing_number($link,$arg[0],$arg[1],@sign);
  &print_matrix($foo);
}

sub print_matrix {
  local($link) = @_;

  print "[";
  foreach (@{$link}) {
    print "[";
    foreach (@$_) {
      print "$_, ";
    }
    print "],";
  }
  print "]\n";
}

sub matrix_copy {
  local($link)=@_;
  local(@baz);
```

```
  foreach (@{$link}) {
    push(@baz, [@$_]);
  }
  [@baz];
}

1;

#!/usr/bin/perl

#  交差交換の操作をする
#  プログラム

sub crossing_number {
  local($link,$chg1,$chg2,@sign) = @_;
  local($k,$column,@columns,$h,$j,$p,$q,$r,$c,@seter,$a);

  $link = &matrix_copy($link);
  for ($k=0; $k<@{$link}; $k++){
    $column = @{${$link}[$k]};
    push (@columns,$column);
  }

  ${${$link}[$chg1]}[$chg2] =~ s/^[\-]{1}(\d+)/$1/;

  $k = $1;
  $num = $k;
  for ($h=$chg1; $h<@{$link}; $h++){
```

```
    for ($j=0; $j<$columns[$h]; $j++){
      $p = ${${$link}[$h]}[$j] if ($j != $chg2 | $h != $chg1);
      $q = $p<0 ? -$p : $p;
      if ($k == $q){
        if ($p<0){
          $r = $p;
        }elsif ($p>0){
          $r = -$p;
          ${${$link}[$h]}[$j] = $r
        }
      }
    }
  }
#change:crossing sign;
  $sign[$num-1] = $sign[$num-1]*(-1);
  ($link, @sign);
}

1;

#!/usr/bin/perl

#  平滑化の操作をする
#  プログラム

sub smoothing_number {
  local($link,$chg1,$chg2,@sign) = @_;
  local(@baz,@columns,@set1,@var,@set2,@set3,@set4,@set5,@set6);
```

```
  local($foo4,$column,$columnl,$h,$i,$k,$p,$q,$r,$v,$x,$foo2);
  local($y,$o,$j,$a,$c,$w,$e,$g,$dec);

  $foo4 = &matrix_copy($link);

  $column = @{${$link}[$chg1]};

  for ($k=0; $k<@{$link}; $k++){
    $columnl = @{${$link}[$k]};
    push (@columns,$columnl);
  }

  $i = ${${$link}[$chg1]}[$chg2];
  $k = $i<0 ? -$i : $i;

OUTER:  for ($h=$chg2+1; $h<$column; $h++){

    $p = ${${$link}[$chg1]}[$h];
    $q = $p<0 ? -$p : $p;

    push(@set1,$p) if ($p != $k);
    if ($k == $q){
      push(@var,\@set1);
      last OUTER;
    }
  }
  if ($h<$column){
    for ($r = $h+1; $r<$column; $r++){
```

```
    $v = ${${$link}[$chg1]}[$r];
    push(@set2,$v);
  }
  for ($x = $0; $x<$chg2; $x++){
    push(@set3,${${$link}[$chg1]}[$x]);
  }
  push(@var,[@set2,@set3]);

  $foo2 = \@var;
  splice(@{$foo4},$chg1,1,@{$foo2});

}else{
  for ($y = 0; $y<$chg2; $y++){
    push(@set4,${${$link}[$chg1]}[$y]);
  }
  INNER: for ($o=$chg1+1; $o<@{$link}; $o++){
    for ($j=0; $j<$columns[$o]; $j++){
      $a = ${${$link}[$o]}[$j];
      $c = $a<0 ? -$a : $a;
      if ($k == $c){
last INNER;
      }
    }
  }
  for ($w=$j+1; $w<$columns[$o]; $w++){
    $e = ${${$link}[$o]}[$w];
    push(@set5,$e);
  }
```

```
    for($g=0; $g<$j; $g++){
      $dec = ${${$link}[$o]}[$g];
      push(@set6,$dec);
    }
    push(@var,[@set1,@set4,@set5,@set6]);
    $foo2 = \@var;
    splice(@{$foo4},$o,1,@{$foo2});
    splice(@{$foo4},$chg1,1);
  }
#change:smoothing sign;
  $num = $k;
  $pach = 0;
  splice(@sign, $num-1, 1, $pach);
  ($foo4, @sign);
}

1;
```

上の Jones 多項式を計算するために用いたプログラムの中の単項式や多項式を計算するプログラムも用いる.

参考文献

[1] 落合豊行・山田修司・富田英美子, コンピュータによる結び目理論入門 (1996), 星雲社

[2] Louis H.Kauffman, knots and physics(1993), World Scientific

[3] 鈴木晋一, 結び目理論入門 (1992), サイエンス社

[4] Larry Wall,Tom Christiansen,and Randal L.Schwartz (近藤 嘉雪 訳), プログラミング Perl 改訂版 (1997), オライリー・ジャパン