Maximaの計算事例

横田博史

平成18年9月9日(土)

目 次

まえがき

この文書は結び目理論への Maxima を適用する話と代数曲線や曲面を描画する話の二本建になっています.

先ず, 結び目理論への適用は,Maxima の持つ演算子の定義と利用者が定めた規則の適用という機能を説明する事を重視しています. その為, 数学的な側面に興味の無い方は, 数学的な側面は飛して収録したプログラムを読んで, 演算子の定義と規則の適用の参考にして下さい. 又, 本格的な利用を考えている方には, この文書で採用したアルゴリズムはあまり効率の良いものでは無い事を言っておきます.

次の代数曲線や曲面の描画では,Maxima の持つ属性の設定に注目してプログラムを作成しています. このプログラムを動かす為には surf が起動する環境でなければなりません.

最後に, この文書に収録したプログラムの古いバージョンが KNOPPIX/Math の例題に収録されています.KNOPPIX を使われている方は, これらのファイルをユーザーのホームディレクトリにコピーして利用されると良いでしょう. 特に,surfplot 函数はテンポラリファイルをカレントディレクトリに生成する方式を採用している為, カレントディレクトリが書き込み禁止になっている場合には動作しません.

平成 18 年 9 月 9 日 (土)　　横田博史

第1章　結び目のAlexander多項式

1.1　結び目の概要

ここでは結び目理論から結び目の Alexander 多項式の計算について簡単なお話をします.

ここで何故, 結び目なのかという理由ですが, 結び目理論はドイツ語では Knotten Theorie と呼びます. それにしても,Knotten!! 実に,KNOPPIX に似ていますねぇ…. そこで, 結び目愛好家の為に,Maxima で結び目の不変量を計算して, KNOPPIX/Math を Knotten Pics/Math と洒落込もうというのが目的です.

結び目理論の全般の話はクロウエル, フォックス [4], 本間 [6], 河内 [2], 村上 [9] 等を参照して下さい. 尚, 亀甲結びの結び方と言った別の方面の愛好家の方の要望には沿える内容ではありませんので, 悪からず.

結び目には蝶々結びとか, 色々な紐の結び方があります. ここで, 結び目のある二つの紐が与えられた時, その結び目が同じ結び方で出来ているのかどうかを判別するにはどうすれば良いでしょうか？現実問題としては, 紐を引いたりして同じ形に変形出来るかどうかで判別する方法がありますが, 数学では, どの様に表現すべきでしょうか？

先ず, 結び目理論で扱う結び目は, 紐の両端を繋いで輪にしています. 丁度, 図 1.1 に示す様に空間内部の円周になっています.

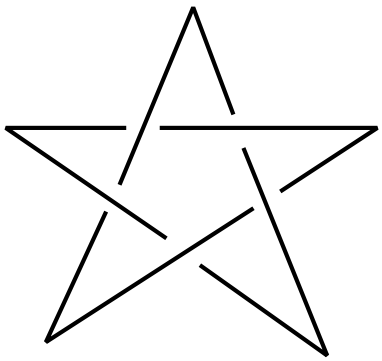


図 1.1: 星型結び目 (5_1)

こうするとどうでしょう, 結び目の両端が切れたままだと, 結び目の瘤の部

分を引いて行けば結び目が消えてしまいますが, 両端を繋げた為に引っ張っても結び目が解けて消えるとは限らなくなります.

この結び目は自分自身が交わる事の無い様に三次元空間 $\mathbb{R}^3$ に置かれています. この状態を円周 S^1 の三次元空間 $\mathbb{R}^3$ への埋め込みと呼びます. 更に, $\mathbb{R}^3$ から結び目を取り除いた空間 $\mathbb{R}^3 - K$ の事を結び目 K の補空間と呼び,$C(K)$ と記述します.

次に, 扱う結び目に現実的な制約を入れましょう. 先ず, ここで扱う結び目は, 有限個の折線で近似可能な結び目に限定しましょう. この有限個の折線で近似出来る結び目の事を順 (tame) な結び目と呼びます. 因に, 無限個の折線が必要な結び目を野性的 (wild) な結び目と呼びます. この様な野性的な結び目は, 力学系, 特にカオスの話に出て来る事があります.

次に, 結び目をゴム紐で出来たものの様に考えて, 結び目を $\mathbb{R}^3$ 内部で, 紐を切ったり, 紐を交点させずに変形して行く事で, 互いに移り合う事の出来る結び目を同値な結び目と呼びましょう. この事を数学では, 結び目 K_1 と K_2 に対し, amibient isotopy と呼ばれる図形が相互に連続的に変形して移り合える可能性を保証する写像が存在する事を意味します. ここでは結び目 K_1 と K_2 の間に ambient isotopy か存在する事を $K_1 \Leftrightarrow K_2$ と表記しましょう. この ambient isotopy という写像が結び目 K_1 と K_2 の間に存在する事は, 簡単に言ってしまえば, $t = 0$[分] の時が, 結び目 K_1 で, それから結び目の紐が自分自身と交点する事もなく, 結び目の形を徐々に変形させながら, $t = 1$[分] の時に結び目 K_2 に移って行く映画が作られる事を意味します.

そこで, 個々の結び目を特徴付けるものに何があるのでしょうか？次の節では, 結び目の射影図を考えますが, そこで現れる結び目の交点の総数もその一つです, 勿論, 射影図の取り方で交点の総数は色々変化します. 例えば, 平面 $\mathbb{R}^2$ に置かれた $x^2 + y^2 - 1 = 0$ で定義される円周の事を自明な結び目と呼びますが, この自明な結び目を Y 軸を中心に捻じったものを XY 平面に射影すれば, 交点は幾らでも出来ます. しかし, 結び目の射影図が持ち得る最小の交点数を考えると如何でしょうか. 自明な結び目の場合は 0 個になります. 自明な結び目以外の一般の結び目の場合, 交点数の下限は 0 個になるので, 交点数が必ず最小限を持つ事が選択公理によって保証されます. そこで, 最小の交点数で結び目を分類する方法が考えられますね.

実際どうなるでしょうか？先ず, 交点が無い結び目は自明な結び目の他には存在しません. この場合は一つだけです. 何か使えそうですね. 実際, この交点で分類したものが, 数学辞典 [7] の付録, 公式 7 の結び糸等の文献にある結び目の表です. 表を見て頂くと判りますが, 交点が 3 個と 4 個の場合は一つだけですが, 交点数が 5 個以上になると同じ交点数を持つ結び目が複数個存在します. その為, 結び目を交点数でグループ分けは出来ますが, 個々の結び目を特徴付けるには十分ではありません. 残念でした.

では, どうすればよいでしょうか. 天下り的になりますが, 結び目には結び

目群というものがあり, この結び目群で結び目は特徴付けられます. この結び目群は結び目 K の補空間 $C(K)$ の基本群と呼ばれる群の一種の事です. 既に三次元空間内部の結び目はその補空間で分類が可能な事が知られています.

ここで, 結び目群の計算方法は,Dehn による Dehn 表示と,Wirtinger による Wirtinger 表示の二つの計算方法が代表的ですが, どちらも結び目の射影図を描いて求める方法です. ここでは, より機械的に計算が出来る Wirtinger による結び目群の表示を利用します.

この結び目群は非常に情報を沢山含んでいますが, そのままでは非常に扱い難い代物です. 実際, 群も自由群に関係子を入れたものになるので, 二つの群が与えられた時に, それらの群が同じものかどうか判断するのはそんなに簡単ではない事が殆どです. 一応,Tietze 変換と呼ばれる操作で移り合える群は同じものである事が知られていますが, それでも結構大変です. そこで, もっと簡単に結び目を比較出来る方法があります. それが, 結び目群の表示で得られる関係子から Alexander 多項式を計算する方法で,Alexander 多項式を用いて結び目を比較します. 尚, 結び目群の表示から Alexander 多項式を求める時に,Fox の微分子という一風変った演算子を用います.

この章では,Maxima の規則について詳細を述べる為に, 結び目群から Alexander 方程式を生成する方法について簡単に述べます. その為, ここで紹介するプログラムは計算効率があまり良いものではない事 (特に Alexander 多項式を計算する個所) を予め断っておきます.

1.2 結び目の射影図

最初に結び目の射影図というものを描きます. これは三次元空間 $\mathbb{R}^3$ 内部の結び目を平面 $\mathbb{R}^2$ に射影したもので, 図 1.2 に示す要領で, 光源 O(無限遠点でも構いません) から出る灯で結び目を射影したものです.

ここで, 射影図上には結び目の紐自体が交点する個所が現われる事があります. この交点する個所を交点と呼びますが, この交点を十字で描くだけでは, どちらが上にあるか判らないので, 地図で高速道路のジャンクションを描く要領で, 上の道で下の道が切断される様に描きます. 即ち, 図 1.1 の様な絵を描きます. 尚, 交差点で上を通る道の事を上道, その下側の道を下道と呼びます.

ところが, 図 1.1 の様に綺麗な射影図が何時も描けるとは限りません. 図 1.3 の左側に示す様に,n(≥ 2) 重点や直交しない交点が出来る事もあります. ところが, 図 1.3 の様な交点は順な結び目では有限個しか出て来ないので, それらの性格の悪い交点を図 1.3 の右側に示す要領で, 紐を局所的に動かしてしまえば除去出来てしまいます. この処理を射影図に施すと, 最終的に射影図で現われる交点は二重点だけにする事が出来ます. この様にして, 有限個の交点で, それらが二重点のみとなった射影図の事を正則な射影図と呼びます.

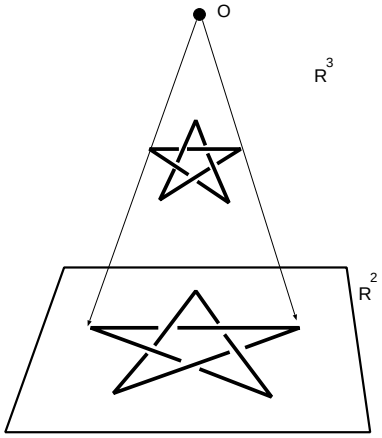


図 1.2: 結び目の射影図

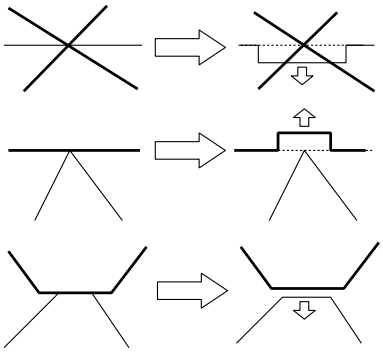


図 1.3: 変形可能な交点

ところで, 同じ結び目でも射影の方法で形が違うものに写ります. 射影図に対し, 図 1.4 に示す Reidemeister 移動と呼ばれる変形で結び目の射影図を変形する事も可能です.

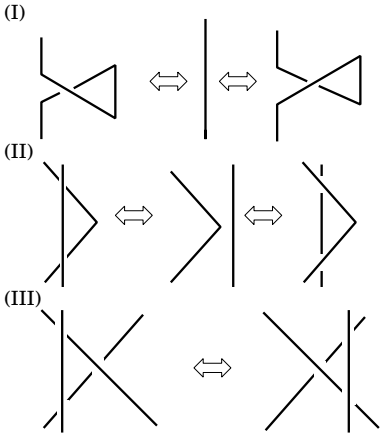


図 1.4: Reidemeister 変形

ここで, 同値な結び目は Reidemeister 移動を繰替えして行う事で, 相互に変形可能となる事が知られています.

1.3 結び目群の Wirtinger 表示

この節で, いよいよ結び目 K の結び目群 $G(K)$ の Wirtinger 表示を計算しましょう.

先ず, 綺麗な射影図が出来上った状態で結び目の射影図は交点で幾つかの上道で分割されていますね. では, 次に結び目に向きを入れて, それから各上道に適当な変数を割当てましょう.

図 1.5 では星型結び目 5_1 の射影図の各曲線に変数 v,w,x,y,z を割当てた様子を示しています.

先ず,Wirtinger 表示による結び目群では, 射影図上の曲線に割当てた変数が群の生成元となります. 次に, 結び目群の関係子は各交点で図 1.6 に示す方法で交点毎に決定されます.

因に, 交点の上の+1 と-1 は交点の符号と呼ばれるものです. 結び目の交点の符号の総和は符号和と呼ばれる不変量で, これも重要な結び目の不変量の一つです.

以上から,Wirtinger による結び目群の表示は, 次の形になります.

$\langle$ 上道$_1, \cdots,$ 上道$_n |$ 交点$_1, \cdots,$ 交点$_n \rangle$

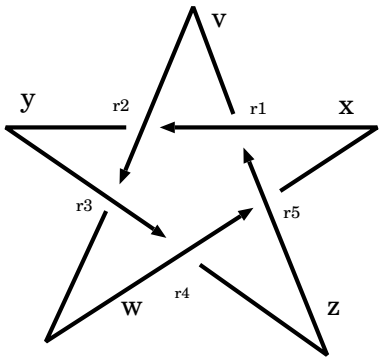


図 1.5: 星型結び目 5_1 と変数

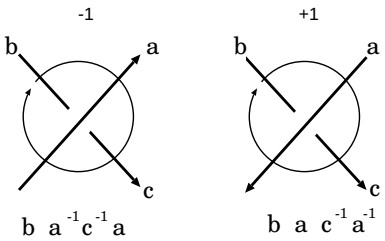


図 1.6: Wirtinger 表示

では何故,Wirtinger 表示で結び目群が表現出来たのでしょうか？この節も非常に天下り的になりますが, その事を簡単に説明しましょう.

先ず, 結び目の射影図の上道に付けた変数は図 1.7 に示す様に, $\mathbb{R}^3 - K$ 内部の点 P から出て, 結び目の上道の向きに対して右回りで上道を一回りし, 点 P に戻る閉じた道が対応します. この閉じた道は点 P に根本が固定されていますが, ゴム紐の様に伸縮自在で, 結び目を取り除いて出来た穴に邪魔されなければ, 点 P に潰れてしまう性質を持っています. ここで, 点 P を基点とする閉じた道 a_1 と a_2 が等しくなるのは, a_1 と a_2 が空間内部で連続的に互いに変形出来る場合とします.

閉じた道 a b による積 ab は点 P を出て a の道を辿って, 次に b の道を辿る閉じた道と定義します. この時, 点 P から動かない道は, この積の単位元となります. 更に, 生成元 a を閉じた道とすれば, a^{-1} を逆の同じ閉じた道で, 逆方向の左回りに上道を一周する閉じた道としましょう.

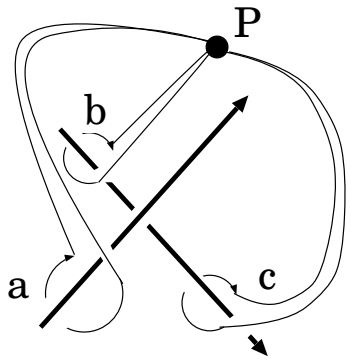


図 1.7: 結び目と閉じた道

ここで,a と a^{-1} の意味を説明するのが図 1.8 です.

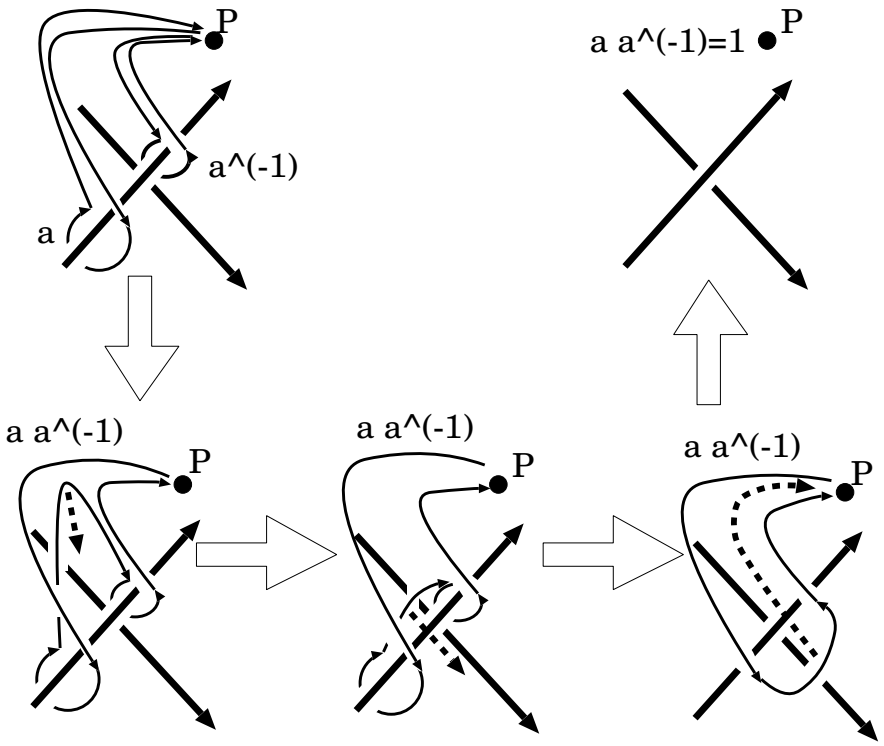


図 1.8: 閉じた道 a , a^{-1} と aa^{-1} の意味

この様に, 通常の閉じた道 a と a^{-1} は各々結び目の上道に遮られているので, この上道を越えて点 P へと潰れる事は出来ません.

ところが, 積 aa^{-1} は, 一度, 閉じた道 a を一回りしてから, 次は a^{-1} の道, つまり,a の道の逆方向に動く事を意味します. そこで, aa^{-1} は図 1.8 の中段左の様な道になります. この道を矢印に沿って動かすと, 結局, この閉道は結び目から外れてしまうので, 点 P に潰れる事になります. すると, 点 P から動かない点が単位元 1 となります. この事から, a^{-1} の意味が明確になるかと思います.

Wirtinger 表示の関係子の意味は, この図 1.8 でやっている事と似た処理を行う事で判ります. 先ず, 図 1.9 に示す様に, これらの閉じた道を関係子の順で繋ぐと, 曲線から外れた点 P から出て点 P に戻る向き付けられた円になります. この円を少しづつ動かして行くと, 結び目の射影図の曲線に引掛かからずに点 P に潰す事が出来ます.

この様に,Wirtinger 表示の関係子は, 関係子で表現される式が単位元 1 になる事を意味しているのです.

尚, 結び目群で 1 になるという事は, 空間上で邪魔される事無しに, 基準点

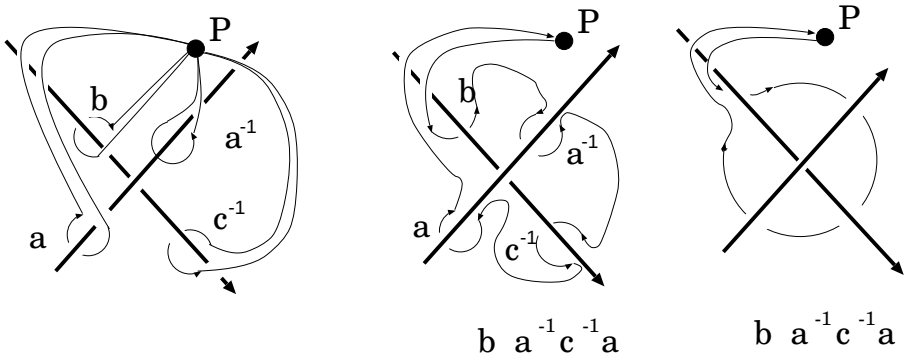


図 1.9: Wirtinger 表記の意味

P に潰れる事を意味します. ここで, $\mathbb{R}^3 - K$ 内部では, 閉じた道は, 埋め込まれた円盤, 即ち, 自分自身が交点した変な円盤ではなく, 歪んでいるだけの円盤の境界になっている事が知られています (Dehn の補題).

この事から, 自明な結び目の場合, 結び目群は $\langle x \rangle$, 即ち, $\mathbb{Z}$ に等しい事が判ります.

さて,Wirtinger 表示の話に戻ります. 図 1.5 の星型結び目 5_1 の場合, 関係子の計算は図 1.7 の関係式を参照すれば, 結び目群の Wirtinger 表現として以下が得られます

星型結び目 5_1 の Wirtinger 表現

$$\langle v, w, x, y, z | xz^{-1}x^{-1}v, vx^{-1}v^{-1}y, yv^{-1}y^{-1}w, wy^{-1}w^{-1}z, zw^{-1}z^{-1}x \rangle$$

尚, この群では関係子が交点の数だけ出来ますが, 実際はこの関係子の内で一つは不要です. そこで一つを取り除いたものでも構いません.

Wirtinger 表示による結び目群は図 1.4 に示す Reidemeister 移動の不変量となります. この事を説明する為には語の Tietze 変換を用います. 詳細は クロウエル, フォックス [4] 等の結び目理論の本を参照して下さい.

1.4 群環と Fox の微分作用素

次に, 結び目群 $G(K)$ の生成元を $x_1, \cdots, x_n$ とする時, これらで生成される自由群 $F = \langle x_1, \cdots, x_n$ とその群環 $\mathbb{Z}F_n$ を考えましょう. 群環 $\mathbb{Z}F_n$ は群の成分を可換演算子 $+$ を使って形式的な和を取ったものになります. この群環は, 積 $*$ に対しては群になり, 和$+$を結合律や分配律を満す様に入れるので環になります.

Fox の微分子 Δ は次の性質を持っています.

Fox の微分子 Δ

- $\Delta(x+y) = \Delta(x) + \Delta(y)$
- $\Delta(xy) = t(y)\Delta(x) + x\Delta(y)$

先ず,Fox の微分子 Δ は $\mathbb{Z}F_n \to \mathbb{Z}F_n$ の線形写像 ($\mathbb{Z}$ 準同形写像) となります. それから, 語の積に対しては, 何処となく普通の微分の積公式に似たものですが, 函数 $t : \mathbb{Z}F_n \to Z$ で, 群 F_n の各生成元に 1 を代入する線形函数 t か入っている事に注意して下さい.

この基本性質を利用すると,Fox の微分子が次の性質を持つ事が容易に確認出来ます.

Fox の微分子 Δ の性質

- $\Delta(n) = 0, n \in \mathbb{Z}$
- $\Delta(x^n) = \sum_{i=0}^{n-1} x^i \Delta(x), n \in \mathbb{N}$
- $\Delta(x^{-n}) = -\sum_{i=-n}^{-1} x^i \Delta(x), n \in \mathbb{N}$

では,Maxima で Fox の微分子を実際に表現してみましょう.

1.5 Maxima で遊ぶ Fox の微分子

Fox の微分子を Maxima で表現する為には,Fox の微分子で利用する各種の函数を最初に定義しなければなりません.

先ず,Fox 微分の積の性質で出て来た函数 t を定義しなければなりません. 但し,t は変数で使う事が多いので, ここでは函数名を $t1$ としましょう.

この函数は線形函数なので, 語に対してのみ函数を定義し, 後は declare 函数を使って線形性を持つ事を宣言してしまえば十分です. そこで, 以下の Maxima の函数として定義します.

函数 t1

```
t1(x):=block([vars:listofvars(x),n,i],
              n:length(vars),
              for i in vars do
                 (x:subst(1,i,x)),
                  return(x));
```

この函数では, 全ての変数に 1 を代入する為,listofvars 函数で語を構成する変数を全て取出し, subst 函数を使って変数に 1 を代入します. 実際は引数が語であれば 1 となる様に定義し, それから,declare 函数で函数 $t1$ が linear で

あると宣言しておけば十分ですが, ここでは subst 函数と listofvars 函数を使いたかったので, 敢て複雑にしています.

今度はこの函数 t1 の値域となる群環 $\mathbb{Z}F_n$ の Maxima での表現方法です. ここでは, 群 F_n の積は Maxima の非可換積である dot 積を用い, 冪はそれに対応する非可換積の冪^^を使います. 但し, 整数と語の積では, 可換積 $*$ を用います.

従って, 非可換積と可換積が混入した式を扱う事になりますが, この様な式の計算を Maxima は難無く実行します.

```
(%i76) 2*x.y+3*x.y;
(%o76)                              5 (x . y)
(%i77) 2*x.y.z+3*x.y;
(%o77)                    2 (x . y . z) + 3 (x . y)
```

これで群環の演算は一応大丈夫そうですね.

では, 与式が語であるかどうかを判別する述語函数を定義しましょう. 与えられた式を語として判断しても良いのは以下の 1 から 4 に限られます.

w が語となる条件

1. 与式がアトム w
2. 与式がアトムの積 $w_1.w_2 \cdots w_n$
3. 与式が整数とアトムの可換積を含む式 $n * w_1.w_2 \cdots w_n$
4. 与式がアトムや語の非可換冪 $x\hat{}\hat{}m$

これらの状況の判定は, 式に含まれる演算子で出来てしまいます. 先ず, 最初のアトム w の場合は演算子が含まれていませんが, 述語函数 atom で判別出来ます. それから $n * w_1.w_2 \cdots w_n$, $w_1.w_2 \cdots w_n$ と $x\hat{}\hat{}m$ の場合, 各々の内部表現の先頭は, 可換積 $*$, 非可換積 . と非可換積の冪^^の何れかになります.

以上から, 以下の判定条件で式を判別すれば良い事になります.

w が語となる判定条件

1. $atom(w)$ が true
2. $inpart(w, 0)$ が可換積 $*$
3. $inpart(w, 0)$ が非可換積 .
4. $inpart(w, 0)$ が非可換積の冪^^

定義すべき述語函数 wordp は, 最初に与式がアトムであるかどうかを調べ, アトムであれば true, そうでなければ, 与式の内部表現の第一成分を取出して,

それが非可換積積, 可換積, 冪の何れかであれば,true を返し, それ以外は false を返す函数になります.

この函数を Maxima で構築したのが以下の述語函数 wordp となります.

述語函数 wordp

```
wordp(w)   :=  if atom(w) then true
               else if member(inpart(w,0),[".","^^","*"]) then true
               else false;
```

この word 函数では,inpart 函数を用いて, $inpart(w,0)$ で与式の内部表現の第一成分を取出し,member 函数を使って, その式の第一成分の値が, 非可換積, 非可換積の冪^^か可換積 $*$ の何れかであれば true を返す仕様になっています.

では早速, この函数を試してみましょう.

```
(%i59) wordp(w);
(%o59)                                  true
(%i60) wordp(2*w.z);
(%o60)                                  true
(%i61) wordp(w.z);
(%o61)                                  true
(%i62) wordp(w.z+2*x.y);
(%o62)                                  false
```

これで, 与式が語であるかどうかが検証出来る述語函数が出来ましたね. 利用する前に declare 函数を使って, declare(wordp,linear) とすれば十分です. この述語函数の存在によって,Maxima にとって何が語となるかが判断出来る様になりました. この様な規則を定める対象を判別する述語函数が出来た段階で, 初めて規則が定義可能となりました.

では,Fox の微分子 D_fox : を Maxima で表現しましょう. ここでは最初に演算子の属性を宣言しておきます. この函数は, 引数の前に置くので,prefix 函数を用います. 定義するだけであれば, prefix("D_fox:") だけでも十分です. 尚,Maxima の演算子には, 演算子と被演算子を結び付ける力が定義可能です. この演算子と被演算子を結び付ける力の事を演算子の束縛力と呼びます. 例えば, 式 $1+2\cdot 3^4-4$ が与えられた時, 和, 積, 差と冪等の演算には順番があるので, この式を $1+(2\cdot(3^4))-4$ と解釈します.Maxima の表記では, `1+2*3^4-4` は `1+(2*(3^4))-4` となる訳です. この様に演算子の隣の被演算子を引き付けておく力を Maxima では束縛力と呼んで, その強さを数値で表現しています.Maxima では和は 100, 可換積の左束縛力が 120, 非可換積の左束縛力が 130 で右束縛力が 129, 非可換積の冪の左束縛力が 140 で右束縛力が 139 となり

ます.Maxima の函数を用いて演算子を定義した場合, 束縛力はデフォルトで 180 が設定されます.

その為,D_fox:x^^5 が与えられた時, 演算子 D_fox:は非可換冪の束縛力よりも大きい為に,(D_fox:x)^^5 が与えられたと Maxima は解釈します. そこで,x^^5 に D_fox:を作用させたければ,D_fox:(x^^5) の様に小括弧で括れば良いのです. 但し, 将来の安全性を考えれば,D_fox:の束縛力を^^よりも小さくした方が無難です. そこで, 右束縛力を 128 としましょう.

尚, ここでの宣言では, 具体的に函数を記述する必要は全くありません.Maxima は演算子に与えられた様々な性質や規則に基いて処理を行います. この宣言に続けて, 今度は線形性 $D(x+y)=D(x)+D(y)$ も追加しましょう. この線形性の宣言に declare 函数を用います. では以下の様に, 演算子 D_fox:の宣言を行いましょう.

```
(%i1) prefix("D_fox:",128);
(%o1)                                D_fox:
(%i2) declare("D_fox:",linear);
(%o2)                                 done
```

これで, D_fox の演算子としての属性と線形性が付加されました.

次に, 微分の性質を付加します. この性質は Maxima の規則として与えます. その為に, 並び変数の宣言を予め行い, 規則の宣言では, この宣言しておいた並びの変数を用います.

```
(%i3) matchdeclare([_x,_y],wordp);
(%o3)                                done
(%i4) defrule(Dfox_Prod,D_fox:(_x._y),
                        D_fox:_x*t1(_y)+_x.D_fox:_y);
(%o4) Dfox_Prod : D_fox: (_x . _y) ->
                        D_fox:_x t1(_y)+_x.D_fox:_y
```

この例では, 最初に matchdeclare 函数を使って, 規則を与える際に利用する並びの変数の定義を行っています.Maxima の規則は matchdeclare 函数で宣言した変数を用いて定義を行いますが, 一般の式への適用では, 後の述語函数を用いて, その式に適用する際に前提とした変数の条件を満すものかどうかを Maxima は検証します. この宣言を行わなければ,Maxima は defrule 函数で用いた変数に対してのみ規則の適用を行い, 一般の式に対して利用は出来なくなります. 尚,matchdeclare 函数では並びの変数を判定する述語函数を設定しますが, ここで true を設定すると, 全ての変数に対して true が返される事になるので, 条件を付ける必要が無い場合は true を述語函数として指定する事が可能です.

次に,defrule 函数を用いて, 規則名と, 適用する式の並びと規則の適用後の変換式を宣言します. この例では, 規則名を Dfox_Prod とし, その内容は D_fox : (_x . _y) を D_fox : _x * t1(_y) + _x.D_fox : _y で置換するものです. こうする事で, 変数 a と b が語としての属性を持っていれば, `D_fox:(a.b)` は規則 `Dfox_Prod` を適用する事で `D_fox:a * t1(b) + a .  D_fox:b` に置換される事になります.

以上で非可換積に対する微分の性質が付加されました.

ここで注意しなければならない事は, 非可換積に影響を与える大域変数 dotassoc をデフォルトの true のままにしていると,Maxima が $(x.y).z$ を自動的に $x.y.z$ に平坦化してしまう事です. これは内部的には自動的に $(.\ x\ y\ z)$ となる事を意味しますが, この規則の処理では二項演算を前提にしているので, $(x.y).z$ の様に平坦化される前の二項演算のままでなければ処理が上手く出来ません. その為, 結び目群を定義する前に, dotassoc:false; で大域変数 dotassoc を false に設定していなければなりません.

最後に, 与式が定数や非可換冪の場合の処理を組込みましょう. 先ず, 与式が定数の場合,Fox の微分子は 0 を返します. ここで環は整数環 $\mathbb{Z}$ なので, 定数は integer として仮定して構いません. 従って, 定数に対しては, 次の規則を入れれば十分です.

```
(%i5) matchdeclare(_a,integerp);
(%o5)                                done
(%i6) defrule(Dfox_const,D_fox:_a,0);
(%o6)                       Dfox_const : D_fox: _a -> 0
```

次に, 与式が非可換冪の場合を組込みましょう. この場合,matchdeclare で宣言する整数には正の場合と負の場合を各々宣言する必要があります. そこで, posintp と negintp の二つの述語函数を定義します.

posintp 函数と negintp 函数

```
posintp(x)   :=   if featurep(x,integer) then  if is(x>0) then true
negintp(x)   :=   if featurep(x,integer) then  if is(x<0) then true
```

posintp 函数と negintp 函数は基本的に同じ操作を行う函数です. 与えられた引数が整数であれば, 既に整数の属性を持っているので,featurep 函数は true を返し, 後は正負の判定を行えば良く, 一般の変数の場合, 先ず,declare 函数で整数の属性を持たせ, それから assume 函数で正負の性質を入れてあるものを扱います.

では, 函数の動作をここでも確認しておきましょう.

```
(%i11) declare(n1,integer);
```

```
(%o11)                                  done
(%i12) assume(n1>0);
(%o12)                                [n1 > 0]
(%i13) posintp(n1);
(%o13)                                  true
(%i14) declare(n2,integer);
(%o14)                                  done
(%i15) assume(n2<0);
(%o15)                                [n2 < 0]
(%i16) negintp(n2);
(%o16)                                  true
(%i17) negintp(n1);
(%o17)                                  false
```

これで道具か揃ったので, これらの述語函数を用いて正の冪の場合と負の冪の場合の展開規則を D_fox : に付与しましょう. 規則の与え方は, 微分としての展開規則 `Dfox_Prod` と同様です. 最初に規則を与える変数を matchdeclare 函数を用いて宣言し, その変数を用いて defrule 函数で規則を宣言します.

```
(%i9)matchdeclare(_ap,posintp)$
(%i10)matchdeclare(_an,negintp)$
(%i11)defrule(Dfox_PPower, D_fox:(_x^^(_ap)),
              sum(_x^^(i),i,0,_ap-1).D_fox:_x);
(%o11) Dfox_PPower : D_fox: (_x     )
              -> sum(_x   , i, 0, _ap - 1) . D_fox: _x
(%i12)defrule(Dfox_NPower, D_fox:(_x^^(_an)),
             -sum(_x^^(i),i,_an,-1).D_fox:_x);
(%o12) Dfox_NPower : D_fox: (_x     )
              -> - sum(_x   , i, _an, - 1) . D_fox: _x
```

これで,Fox の微分子を利用する為に必要な規則は全て揃いました. では試しに `r1=x.y.z^^(-1).y^^(-1)` に Fox の微分子を作用させた結果を計算してみましょう.

```
(%i9) dfr1:D_fox:(r1);
                                        <- 1>    <- 1>
(%o9)                D_fox: (x . (y . (z      . y     )))
```

入力しても別に展開されていません. 何故でしょう？Maxima では規則を定めるだけではなく, その定めた規則を式に対して適用する函数を必要とします. この様な函数には幾つかありますが, 今回は apply1 函数を用います.

必要な規則は, 語の非可換積の展開規則 Dfox_Prod と負の冪の規則 Dfox_NPower の二つです. この様に複数の規則を利用する場合, apply1 函数に適用する規則を順番に並べます. ここでは最初に展開規則を用い, それから負の冪の展開規則を用いるので, 以下の様に入力します.

```
(%i11) apply1(dfr1,Dfox_Prod,Dfox_NPower);
                     <- 1>               <- 1>     <- 1>
(%o11) x . (y . (- z      . D_fox: z - z      . (y      . D_fox: y))
                         + D_fox: y) + D_fox: x
```

確かに展開が出来ていますね.

以上の Fox の微分子の定義, 規則の宣言等を纏め, ファイル fox.mc に記入しておきましょう. このファイルの中に注釈を入れておくと後で読み易く, 管理を行うのも容易になります.Maxima では注釈は/*と*/の間に記述します. このファイル fox.mc の内容を以下に示しておきましょう.

```
/* MAXIMA */

/* 函数 t1

      x:結び目群の語.

Fox 微分で現われる函数. 線形性を持ち, 語を 1 に写す函数.
函数の定義では, 単純に語の変数に 1 を subst を用いて設定し,
線形性は declare を利用します.
*/

t1(x):=block([vars:listofvars(x),n,i],
              n:length(vars),
              for i in vars do
                 (x:subst(1,i,x)),
                  return(x));

declare(t1,linear);

/* Fox の微分子の定義.
```

```
演算子は prefix 函数を用いて, 前置式の演算子である事を宣言します.
尚, 被演算子の右束縛力は 128 とし, 非可換冪よりも小さな値とします.
更に, 線形性は declare 函数を用います.
*/

prefix("D_fox:",128);
declare("D_fox:",linear);

/* Fox の微分子の置換規則で用いる変数宣言と述語函数定義  */

/* _x と_y は語とします. */
matchdeclare([_x,_y],wordp);

/* 語の判定函数 */
wordp(w) :=
if atom(w) then true
else if member(inpart(w,0),[".","^^","*"]) then true
else false;

/* _a は整数とします. 判定函数は Maxima 組込の integerp を用います. */
matchdeclare(_a,integerp);

/* _ap は正整数,_an は負の整数とします. ここで判定函数は,
正整数の場合は posintp, 負の整数の場合は negintp を用います.*/
matchdeclare(_ap,posintp);
matchdeclare(_an,negintp);

/* 語の判定函数 */
posintp(x):= if featurep(x,integer) then
                if is(x>0) then true;
negintp(x):= if featurep(x,integer) then
                if is(x<0) then true;

/* 積規則 Dfox_Prod */

defrule(Dfox_Prod,D_fox:(_x._y),D_fox:_x*t1(_y)+_x.D_fox:_y);

/* 定数に対する規則 Dfox_const */
defrule(Dfox_const,D_fox:_a,0);
```

```
/* 正の冪に対する規則 Dfox_PPower */
defrule(Dfox_PPower, D_fox:(_x^^(_ap)),
                     sum(_x^^(i),i,0,_ap-1).D_fox:_x);

/* 負の冪に対する規則 Dfox_NPower */
defrule(Dfox_NPower, D_fox:(_x^^(_an)),
                    -sum(_x^^(i),i,_an,-1).D_fox:_x);

/* 非可換積の結合律の適用を阻止. 積規則を利用する為に必要 */
dotassoc:false;
```

このファイルの演算子, 函数や規則を用いたければ, load 函数を用いて `load("fox.mc");` と入力します. 更に,Maxima の起動時に直ちに使いたければ,Maxima を起動するディレクトリに maxima-init.mac という名前のファイルを作り, その中に, 予め, load(”fox.mc”); を記入しておきます. すると,maxima-init.mac があるディレクトリ上で maxima を起動させると, 自動的に maxima-init.mac 内部が解釈されて fox.mc が読込まれます.

これで結び目の Alexander 多項式を計算する為の道具が漸く揃いました.

1.6 Alexander 多項式

Alexander 多項式を計算する為には,Alexander 行列というものを計算しなければなりません. この Alexander 行列は結び目群 $G(K)$ の関係子に Fox の微分子を作用させる事で出来る行列です. 即ち, ベクトル $(r_1, \cdots, r_n)$ から Fox の微分子による Jacobian が Alexander 行列になります.

ここで, 先程の節で定義した Fox の微分子は, 通常の微分で言えば, D に相当します. 普通, 変数 x に対する微分 $\frac{d}{dx}$ が存在しても良さそうですが, Fox の微分子も同様です. この場合, 結び目群の Wirtinger 表現の生成元 x_i に対して,$\frac{\partial}{\partial x_i}$ が定義され, 他の生成元 x_j に対して, $\frac{\partial x_j}{\partial x_i} = \delta_{ij}$ を満します.

例えば. 星型結び目の場合, 生成元は x, y, z, v, w の 5 個あるので, Fox の微分子は具体的には, $\frac{\partial}{\partial x}, \cdots, \frac{\partial}{\partial w}$ と生成元の数だけ存在します.

ここで Maxima 上で定義した Fox の微分子 D_fox:には, 生成元の情報がありません. 実はこうした方が便利なのです. 例えば, 星型の場合,5 個の微分子を定義し, それらを使って行列を求めるたりする事や,2 変数の函数を定義して処理を行うよりも, ここで行った様に, 一つの演算子を決めて, $\frac{\partial}{\partial x_j} x_i = \delta_{i,j}$ の関係を後で代入する方が, より処理を機械的に行えるだけではなく, 余計な定義や処理が不要になるからです.

それから,Fox の微分子を作用させた後に, 結び目群 $G(K)$ の Wirtinger 表示による生成元を全て変数 t で置換し, 非可換積とその冪も全て可換積とその冪に置換します. この処理によって,D_fox:の Jacobian は $\mathbb{Z}[t^{-1}, t]$ の元を成分を持つ正方行列になります. この正方行列を Alexander 行列と呼びます.

結び目群 $G(K)$ の Alexander 多項式と呼ばれる多項式は, この Alexander 行列の余因子行列の各成分の最大公約因子の事です. 正確にはこの最大公約因子は一次の Alexander 多項式と呼ばれるものですが, 結び目理論ではこの多項式の事を Alexander 行列と呼びます.

尚,Alexander 多項式は Laurant 多項式環 $\mathbb{Z}[t^{-1}, t]$ のイデアルの生成元になります. その為,Alexander 行列を計算した時に, $t^n, n \in \mathbb{Z}$ の違いが生じる事がありますが,Alexander 多項式として定数項が零にならない様に各項の次数が正となるものを選択します.

この処理を纏めたプログラム calcAlexanderPoly を以下に示します.

```
/* MAXIMA */

/* calcAlexanderPoly

fox.mc と併用する事が前提です.

結び目 G はリストで表現します.
即ち,G=<x1,...,xn| r1,...,rm>を G:[[x1,...,xn],[r1,...,rm]]
で表現します.

calcAlexanderMatrix では,Wirtinger 表示の結び目群に対して
Alexander 行列の計算を行います. その為, 関係子 r1,..,rm の
個数 m は n か n-1 に等しくなければなりません. 従って, 関係子の
個数が n か n-1 に等しくなければエラーになります.
*/
calcAlexanderPoly(G):=
block(
      [vars:G[1],rels:G[2],amat,Alex:[],
       Ia,dfx,rdfx,rdfx,crdfx,mrow,n:length(G[1]),
       m:length(G[2]),AlexanderPoly,APolyData:false,
       tmp,lst:[]],
       /* 結び目群の Wirtinger 表示に限定 */
       if m=n or m=n-1 then
        (
         Ia:subst("[",matrix,ident(n)),
         for i from 1 thru n do append(lst,[i]),
```

```
dfx:map("D_fox:",vars),
/* 関係子リストに演算子を作用 */
rdfx:map(lambda([x],
         apply1(D_fox:x,Dfox_Prod,
                        Dfox_PPower,
                        Dfox_NPower)),
         rels),
/* 微分に 1 か 0 を設定 */
for i from 1 thru n do
  ( mrow:Ia[i],
    rdfx2:rdfx,
    for j from 1 thru n do
      (
        rdfx2:map(lambda([x],
                  subst(mrow[j],dfx[j],x)),rdfx2)
       ),
    /* 非可換積の冪を可換積の冪に変換 */
    crdfx:map(lambda([x],
              subst("^","^^",x)),rdfx2),
    /* 非可換積を可換積に変換し, 簡易化を実施 */
    amat[i]:ratsimp(map(lambda([x],
                        subst("*",".",x)),crdfx))
  ),
/* Alexander 行列を配列として最初に構築. */
for i from 1 thru n do
    Alex:append(Alex,[amat[i]]),
/* 配列データから行列データに変換 */
Alex:substpart(matrix,Alex,0),
/* 変数を t に変換します */
for i in vars do
    Alex:subst(t,i,Alex),
Alex:ratsimp(Alex),
/* Wirtinger 表現で関係子の総数が生成元の個数 n よりも
   一つ少ない場合の処理.Maxima の組込函数では余因子
   行列の生成は正方行列に限定されます.
*/
if m=n-1 then
  (
   /* nx1 の零行列を n 列目に追加. */
```

```
            tmp:addcol(Alex,zeromatrix(n,1)),
            /* 小行列式を計算. */
            tmp:map(lambda([x],
                    determinant(minor(tmp,x,n))),lst)
           )
          else
            tmp:expand(adjoint(Alex)),
          /* 余因子行列をリストに変換. */
          tmp:substpart("[",tmp,0),
          /* LGCD で Alexander 多項式を抽出 */
          AlexanderPoly:num(LGCD(map(LGCD,tmp))),
          /* Alexander 行列と多項式のリストを生成. */
          APolyData:[Alex,AlexanderPoly]
        )
      else
         /* エラー処理.Error!と表示するだけのシンプルなもの */
         print("Error!"),
      /* Alexander 行列を返します. */
      return(APolyData)
   )$
/* リストから最大公約因子を計算します.

Lp: 多項式, 或いは, 整数で構成されるリスト.

長さが 1 の場合, その成分をそのまま返します.
長さが 2 以上の場合, 先頭の二つの成分の最大公約因子を計算し,
その最大公約因子と頭の Lp の成分を二つ取り除いたリストを合せ
たリストを用いて, 再帰的に LGCD を呼出しています.
*/
LGCD(Lp):=
block(
      [a1,n,lgcd:false],
       /* Lp がリストであるかを確認. */
       if listp(Lp) then
         (
          /* Lp の長さを求めます. */
          n:length(Lp),
          /* Lp の長さが 1 の場合, 成分をそのまま返します. */
          if n=1 then
```

```
            lgcd:first(Lp)
          else
            (
             /* Lp の先頭二つで GCD を計算. */
             a1:gcd(Lp[1],Lp[2]),
             /* LGCD を再帰的に呼出しています.
                尚,rest(Lp,2) で先頭の二つの成分を削除した
                リストを生成し,append で二つのリストを結合します.
             */
             lgcd:LGCD(append([a1],rest(Lp,2)))
            )
          )
       else
         lgcd,
       /* lgcd を返して終ります. */
       return(lgcd)
)$
```

このファイルは calcAlexanderPoly 函数と LGCD 函数の二つで構成されています. calcAlexanderPoly 函数は計算した Alexander 行列と多項式をリストの形式で返します. 内部で処理している事は, 与えられた結び目群から生成元と関係子を取出し, 生成元に D_fox : を作用させたリストを生成します. このリストへの作用は map 函数を用います. 次に, 生成元の個数 n から n 次の単位行列 Ia を生成します. この単位行列の i 番目の行は i 列のみが 1 となるので, D_fox : $\mathrm{x_j} \Leftrightarrow \frac{\mathrm{d}}{\mathrm{dx_i}}\mathrm{x_j} = \delta_{\mathrm{ij}}$ への変換に用います. ここで,lambda 函数を用いて臨時の函数を構築し, その函数に対して subst 函数を用いて変換を行います.

それから, 非可換積を可換積, 非可換積の冪を可換積の冪へと変換します. この変換では substpart 函数を用います.Maxima の演算子は式の表現で 0 番目の位置に置かれる為,0 番目の成分を入れ換える様にしています. ここでも相手がリストの為,map 函数を効果的に適用する為に,lambda 函数を使って, 臨時の函数を定義しています. それからリストの形式から行列データに substpart 函数を用いて変換し, 各変数を全て t に置換えると Alexander 行列が出来ます.

次に Alexander 行列の余因子行列を計算しなければなりませんが, 余因子行列を計算する adjoint 函数や小行列を求める minor 函数も正方行列のみにしか使えません. Wirtinger 表示の場合, 変数よりも一つ関係子が少なくても問題が無い為, これでは余因子行列が計算出来ない事があります. そこで, 変数よりも関係子が一つ少ない場合には,Alexander 行列の n 列目に n 次の零ベクトルを addcol 函数を使って追加します. それから minor 函数で Alexander 行列 Alex の小行列 $Alex_{1\leq i\leq n,n}$ を minor 函数を用いて計算し,determinant

函数を map 函数を適用する事で作用させて必要な余因子行列を計算します. すると, 多項式成分の行列/リストが出来ますが, その成分の最大公約因子を求める為に, 新たに構築した LGCD 函数を使います.

この LGCD 函数はリストに含まれる各成分の最大公約因子を計算する函数で, リストの長さが 1 であれば, リストに含まれる式をそのまま返し, 長さが 2 以上であれば, 頭の二つに gcd 函数を使って最大公約因子を求め, 頭二つを削ったリストの先頭に結果を加えたリストを使って LGCD 函数を呼出すという再帰的な手法を用いています. この再帰的な手法は LISP で顕著な特徴の一つで,Maxima でもこの様に利用出来ます. 但し, 安易な再帰的な手法は速度的な面で問題が出易い傾向があります. 尚, ここでの処理では大きな行列 (例えば,10 × 10 の行列) を扱う事は考えていない為, この処理で特に大きな問題にはならないでしょう. 但し, 非力な計算機 (Pentium 100MHz 以下) では問題があるかもしれません. この辺は工夫されると面白いかと思います.

それから, 余因子行列をリストに変更し, この LGCD 函数を用いて最大公約因子を求めます. この時, 最初に map 函数で余因子行列リストの行成分の最大公約因子を計算すれば, 行の最大公約因子のリストが得られます. そのリストに今度も LGCD 函数を適用すれば, 最終的に Alexander 多項式が得られます.

尚, この Alexander 行列と多項式には幾何学的な表現があります. これは非常に視覚的に強烈ですが, 普遍被覆空間等と準備するものが多くなるので, ここでは述べません. 但し, この事に関しては,Rolfsen の Publish or Perish(超訳：鰄が嫌なら論文を書け) という凄い名前の出版社から出ている Rolfsen の名著 [14] に詳しく説明されています. 意味は判らなくても不思議な絵を図書館で眺めるだけの価値は十分あると思います.

1.7 Alexander 多項式で結び目を分類しよう

星型結び目 (5_1)

先程から出ている星型結び目の Alexander 多項式を計算しましょう. 先ず, 星型結び目の結び目群の Wirtinger 表示を以下に示します.

星型結び目の結び目群

$\langle v, w, x, y, z | xz^{-1}x^{-1}v, vx^{-1}v^{-1}y, yv^{-1}y^{-1}w, wy^{-1}w^{-1}z, zw^{-1}z^{-1}x \rangle$

では, 早速計算しましょう. ここの例題では, 予め, load("fox.mc") と load("AlexanderPoly.mc") が実行されているものとします.

```
(%i14) star:[[x,y,z,v,w],
[x.z^^(-1).x^^(-1).v,v.x^^(-1). v^^(-1). y,
y.v^^(-1).y^^(-1). w,
w.y^^(-1). w^^(-1).z,
z.w^^(-1).z^^(-1).x]]$
(%i15) K5_1:calcAlexanderPoly(star)$
(%i16) K5_1[2];
                              4    3    2
(%o16)                        t  - t  + t  - t + 1
(%i17)tex(K5_1[1]);
$$\pmatrix{{{t-1}\over{t}}&-1&0&0&{{1}\over{t}}\cr 0&{{1}\over{t}}&{{
 t-1}\over{t}}&-1&0\cr -1&0&0&{{1}\over{t}}&{{t-1}\over{t}}\cr {{1
 }\over{t}}&{{t-1}\over{t}}&-1&0&0\cr 0&0&{{1}\over{t}}&{{t-1}\over{t
 }}&-1\cr }$$
(%o17)                                  false
```

星型結び目の Alexander 多項式として $t^4 - t^3 + t^2 - t + 1$ が得られます, 従って, この結び目は自明な結び目と違う事が判りました.

ここで Alexander 行列は tex 函数で出力した結果を,TEX のソースファイルに入れると以下の様に表示されます.

$$\begin{pmatrix} \frac{t-1}{t} & -1 & 0 & 0 & \frac{1}{t} \\ 0 & \frac{1}{t} & \frac{t-1}{t} & -1 & 0 \\ -1 & 0 & 0 & \frac{1}{t} & \frac{t-1}{t} \\ \frac{1}{t} & \frac{t-1}{t} & -1 & 0 & 0 \\ 0 & 0 & \frac{1}{t} & \frac{t-1}{t} & -1 \end{pmatrix}$$

クローバー結び目 (3_1)

クローバー結び目は三葉結び目 (trefoil) とも呼ばれ, 図 1.10 に示す様に, 三枚の葉がある結び目です.

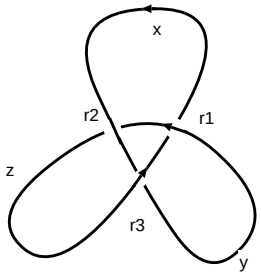


図 1.10: クローバー結び目 (3_1)

クローバー結び目の結び目群

$\langle x, y, z \mid xyz^{-1}y^{-1}, yzx^{-1}z^{-1}, zxy^{-1}x^{-1} \rangle$

```
(%i1) load("AlexanderPoly.mc");
(%o1)                          AlexanderPoly.mc
(%i2) Trefoil:[[x,y,z],
[x.y.z^^(-1).y^^(-1),y.z.x^^(-1).z^^(-1),z.x.y^^(-1).x^^(-1)]]$
(%i3) calcAlexanderPoly(Trefoil);
                       [   1     - t   t - 1 ]
                       [                    ]   2
(%o3)                 [[ t - 1    1     - t  ], t  - t + 1]
                       [                    ]
                       [  - t   t - 1    1   ]
```

この calcAlexanderPoly では Alexander 行列と Alexander 多項式のリストを返します. この結果から, クローバー結び目の Alexander 多項式は t^2-1+1 である事が判ります. この様に Alexander 多項式が 1 でない為, クローバー結び目は自明な結び目と等しくない事, 即ち, 本当に結ばれている事が判ります.

1.8 結び目の連結和と Alexander 多項式

二つの向き付けられた結び目が与えられた時, 連結和によって, 新しい結び目を生成する事が出来ます.

結び目の連結和

1. 先ず, 二つの結び目 K_1 と K_2 を用意し, 各結び目に向きを入れておきます.
2. 結び目 K_i 上の点 P_i を中心とする半径 r の球 $B_i(r), i \in \{1,2\}$ を取ります. これらの球は結び目の一部を含みますが, この際, 半径 r を十分小さく取れば, $B_i(r)$ がドーナッツの様な形状にする事が出来ます.
3. これらの球 $B_i(r)$ を結び目 $K_i, i \in \{1,2\}$ から結び目の曲線を含めて削除します.
4. 取り除いた個所で, 結び目の向きが一致する様に結び目 K_1 と K_2 を繋ぎ合せます.

この結び目を結び目 K_1 と K_2 の連結和と呼び, $K_1\#K_2$ と表記します.

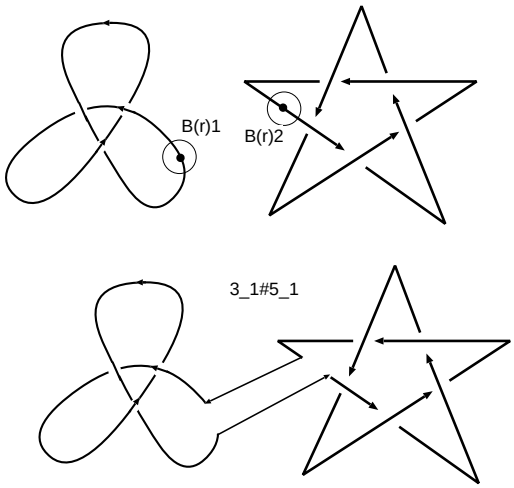


図 1.11: 結び目 3_1 と 5_1 との連結和

この時, 結び目の集合に, 連結和 $\#$ を入れる事で, 結び目の集合に半群の構造が入ります. 単位元は自明な結び目になります. 又, 与えられた結び目が連結和で生成されたものでなければ既約な結び目と呼びます.

ここで,3_1 と 5_1 の連結和の Alexander 多項式を計算してみましょう. この場合,$3_1\#5_1$ の Wirtinger 表示を直接計算しても構いませんが, 前節で計算した 3_1 と 5_1 の Wirtinger 表示があるので, それを利用しましょう.

$3_1\#5_1$ の場合, 図 1.12 の様に,K_1 側の結び目群はそのままにして, K_2 側の変数名を変更しておきます. そして, 繋げる個所の生成元に関係式を追加します. 例えば, この例では, x と y_1 の道を繋げるので $x = y_1$. 即ち, xy_1^{-1} を

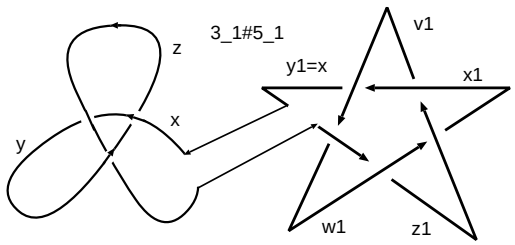


図 1.12: 結び目 3_1 と 5_1 との連結和の結び目群

追加します. ここで calcAlexanderPoly は簡易的なプログラムなので関係子を一つ抜いておく必要があります. その為, k_2 側の本来の関係子を一つ抜いて, その代わりに xy_1^{-1} を入れておきます.

$3_1\#5_1$ の結び目群

$$\left\langle\, x,y,z,x_1,y_1,z_1,v_1,w_1 \;\middle|\; \begin{array}{l} xyz^{-1}y^{-1}, yzx^{-1}z^{-1}, \\ zxy^{-1}x^{-1}, v_1x_1^{-1}v_1^{-1}y_1, y_1v_1^{-1}y_1^{-1}w_1, \\ w_1y_1^{-1}w_1^{-1}z_1, xy_1^{-1} \end{array} \,\right\rangle$$

```
(%i61)  star1:subst(x1,x,star)$
(%i62)  star1:subst(y1,y,star1)$
(%i63)  star1:subst(z1,z,star1)$
(%i64)  star1:subst(v1,v,star1)$
(%i65)  star1:subst(w1,w,star1)$
(%i66)  ts:[append(Trefoil[1],star1[1]),
append(append(Trefoil[2],rest(star1[2],1)),[x.y1^^(-1)])]$
(%i67) cs:calcAlexanderPoly(ts)$
(%i68) factor(cs[2]);
                        2            4    3    2
(%o68)                (t  - t + 1) (t  - t  + t  - t + 1)
```

この様に連結和の Alexander 多項式は二つの結び目の Alexander 多項式の積になります.

これは, 何故でしょうか？ 実際は非常に簡単な事で, 連結和の Alexander 行列を見ると判ります.

$3_1\#5_1$ の Alexander 行列

$$\begin{pmatrix}
1 & -t & t-1 & 0 & 0 & 0 & 0 & 1 \\
t-1 & 1 & -t & 0 & 0 & 0 & 0 & 0 \\
-t & t-1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & \frac{1}{t} & 0 \\
0 & 0 & 0 & \frac{1}{t} & \frac{t-1}{t} & -1 & 0 & -1 \\
0 & 0 & 0 & 0 & 0 & \frac{1}{t} & \frac{t-1}{t} & 0 \\
0 & 0 & 0 & \frac{t-1}{t} & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{t} & \frac{t-1}{t} & -1 & 0
\end{pmatrix}$$

如何でしょうか？ クローバ結び目と星型結び目の Alexander 行列の成分が対角線上に二つのブロックで出現していますね. この行列から余因子行列を計算する訳ですが, 当然, 二つのブロックの行列の行列式の積になります.

この事から, 与えられた結び目の Alexander 多項式が既約でなければ, この結び目は Alexander 多項式の各既約因子に対応する結び目連結和で構成されている可能性があります. 逆に言えば, 結び目の Alexander 多項式が既約であれば, 結び目は既約である可能性があります. 残念な事ですが, 既約であるとは言えません. 何故なら, Alexander 多項式が 1 となる非自明な結び目が存在する事が知られているからです.

1.9 結び目の鏡像と Alexander 多項式

ここでは結び目 K の鏡像 $\overline{K}$ について考えましょう. 結び目を図 1.13 に示す

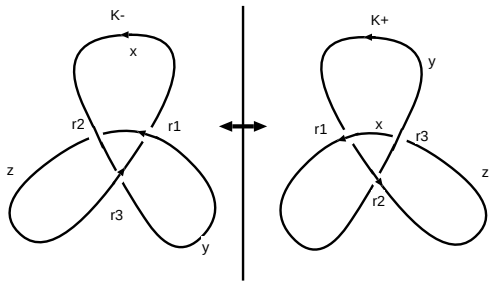


図 1.13: クローバー結び目の鏡像

ここで, K_+ とその鏡像 K_- の違いは何でしょうか？それは交点の符号が逆になる事です. K_+ の場合, 交点の符号は全て $+1$ になりますが, K_- では全て -1 になっています. その為, K_- と K_+ は本質的に違う結び目の筈です. そこで,Alexander 多項式を計算してみましょう.

K_-の結び目群

$\langle x, y, z \mid xzx^{-1}y^{-1}, yz^{-1}y^{-1}zy, yxy^{-1}z^{-1}\rangle$

```
(%i20) TrefoilM:[[x,y,z],
[ x.z^^(-1).x^^(-1).y,z.y^^(-1).
z^^(-1).x,y.x^^(-1).y^^(-1).z]]$
(%i21) tm:calcAlexanderPoly(TrefoilM);
                      [ t - 1    1          ]
                      [ -----    -     - 1  ]
                      [   t      t          ]
                      [                     ]
                      [   1           t - 1 ]   2
(%o21)               [[   -     - 1   ----- ], t  - t + 1]
                      [   t             t   ]
                      [                     ]
                      [        t - 1    1   ]
                      [  - 1   -----    -   ]
                      [          t      t   ]
```

ところが, k_- の Alexander 多項式を計算して見ると, クローバー結び目の場合は鏡像の区別がつきません.

この様に Alexander 多項式は便利ですが残念な事に万能ではありません.

1.10 おまけ：スケイン多項式

Alexander 多項式の計算には,Wirtinger 表示等を用いた方法の他に, 幾何学的な手法で Seifert 曲面を貼って計算する方法, Dehn 手術と普遍被覆空間の構成で視覚的 (?) に求める方法があります. これらの方法も面白い方法ですが, これらとは全く別の方法で, 結び目の交点の局所的な入れ替えによるスケイン関係と呼ばれる関係式から多項式を計算する方法があります.

このスケイン関係で Alexander 多項式の他に,Conway 多項式,Kauffman 多項式, そして,Jones 多項式等の結び目の不変量となる多項式が色々計算出来ます.

ここでは簡単にスケイン関係を用いた結び目多項式の計算を解説します. ここで結び目の多項式は Laurant 多項式環 $\mathbb{Z}[A, A^{-1}, Z, Z^{-1}]$ のイデアルの生成元とします.

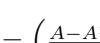
先ず, スケイン関係式とは局所的に結び目の上下関係を入れ換えたものに対して結び目の多項式の間で成立する関係式の事です.

スケイン関係式

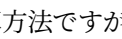
1. $\langle$  $\rangle = 1$

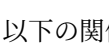
2. $A^{-1}\langle$  $\rangle$ - $A\langle$  $\rangle = Z\langle$  $\rangle$

ここで, 上記の 1,2 を使うと,n 個の自明な結び目が並んでいる n 成分の自明な絡み目  に対して, その絡み目の多項式として,

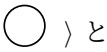
$\langle$  $\rangle = \left(\frac{A-A^{-1}}{Z}\right)^{n-1}$

が得られます.

この多項式の計算方法ですが,n 番目の結び目を捻じった  に上の 2 を適用すると以下の関係式が成立しますね.

$A\langle$  $\rangle - A^{-1}\langle$ $\rangle = Z\langle$ $\rangle$

ここで,  と は に同値なので,

$K_n = \langle$  $\rangle$ とすると, 以下の漸化式が得られるので, この漸化式を解く事で結果が得られます.

- $K_1 = 1$
- $(A - A^{-1})K_n = ZK_{n+1}$

この Skein 多項式の A と Z に変数を設定しなおす事で, 色々な結び目多項式が得られます.

スケイン多項式から得られる結び目多項式

多項式名	A の値	Z の値
Alexander 多項式	1	$t^{1/2} - t^{1/2}$
Conway 多項式	1	Z
Jones 多項式	t	$t^{1/2} - t^{1/2}$

尚,Alexander 多項式は適当に $\pm t^{n/2}$ 倍する必要があります.

では, ここで図 1.14 に示すクローバー結び目のスケイン多項式を計算してみましょう.

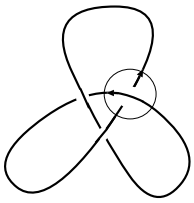


図 1.14: クローバー結び目

この図 1.14 で丸で囲った交点からスケイン関係を適用しましょう. この計算で本質的な事は, 図 1.15 に示すスケイン関係に関連する結び目や絡み目を描く事です.

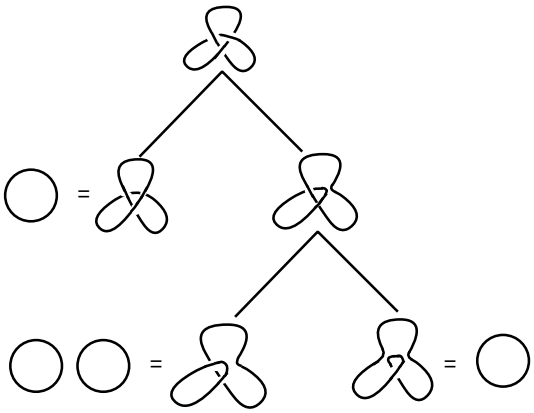


図 1.15: クローバー結び目のスケイン関係

このクローバー結び目の樹形図 1.15 の上 3 個から以下のスケイン関係式が得られます.

$A^{-1}\langle$  $\rangle$ - $A\langle$  $\rangle = Z\langle$  $\rangle$

ここで,  は自明な結び目と同値になる為, この結び目の多項式は 1 になります. 又,  は Hopf 絡み目と呼ばれる有名な絡み目です. そこで,Hopf 絡み目に対するスケイン関係式から多項式を計算します. 図 1.15 の下段がそれに対応します.
この下段から以下の関係式を得ます.
$A^{-1}\langle$  $\rangle - A\langle$  $\rangle = Z\langle$  $\rangle$

すると,  が自明な結び目と同値で  が二成分の自明な絡み目となるので, Hopf 絡み目  のスケイン多項式 $\langle$  $\rangle$ は $Z^{-1}A^{-2}\left(-AZ + A - A^{-1}\right)$ となり, この結果から,  のスケイン多項式 $\langle$  $\rangle$ として $A^{-4}\left(A^2Z^2 + 2A^2 - 1\right)$ が得られます.
Alexander 多項式の場合,$A = 1, Z = t^{1/2} - t^{-1/2}$ を代入する事で, $t + t^{-1} - 1 \sim t^2 - t + 1$ が得られます.

このスケイン多項式で鏡像を計算する場合は, 単純に A を A^{-1} に, Z を $-Z$ に置換えるだけで計算が出来ます.

すると, 先程のクローバ結び目の鏡像のスケイン多項式は,$A^2Z^2 - A^4 + 2A^2$ となります. これを Alexander 多項式に変換しても,$t^2 - t + 1$ となるので, 違いは判りません. では,Jones 多項式で比較するとどうなるでしょうか？

クローバー結び目の鏡像で Jones 多項式を比較

図 1.14 の Jones 多項式	$t^3 + t - 1$
図 1.14 の鏡像の Jones 多項式	$t^3 - t - 1$

この様に Jones 多項式では多項式自体が異なります. 即ち,Jones 多項式は Alexander 多項式や Conway 多項式よりも強力な結び目多項式なのです.

この様に, 基本群から Fox の微分子を使って計算した Alexander 多項式は, 中学生でも出来そうなお絵描きと多項式の計算になりました.

第2章　surfを使う話

2.1　代数曲面

標準的なMaximaで描く事の出来るグラフは $y = f(x)$ や $z = h(x, y)$ の書式や助変数を用いた函数のグラフに限定されます. その為, $x^2 + y^2 - 1 = 0$ を満す点のグラフを描きたい場合, $y = \sqrt{1 - x^2}$ と $y = -\sqrt{1 - x^2}$ の グラフを同時に描くか, $x(t) = \frac{2t}{t^2+1}$ と $y(t) = \frac{t^2-1}{t^2+1}$ を描く等の工夫が必要になります.

そこで,2変数や3変数の多項式の零点集合が簡単に描けるsurfの登場となります.

2.2　surfの概要

surfは変数 x,y,z の多項式 p の零点集合 $V(p)$ を描くアプリケーションです. 単純なGUIを持っており, 直接スクリプトを編集して描画する事も容易に出来ます. C風の簡単な処理言語を持っていますが複雑な処理は出来ません. 基本的に幾つかの基本的な設定を入力し, その設定に対して多項式の零点を描く事と多少の反復処理が出来る程度です.

surfの詳細はこの本では述べません.surfのサイト http://surf.sourceforge.net/ のオンラインマニュアル http://surf.sourceforge.net/doc.shtml や, 私の web-page (http://www.bekkoame.ne.jp/ ponpoko/Math/surf/SurfExamples.html) を参照して下さい.

尚, 残念な事ですが,UNIX環境でしか動作しないので,Windows上でどうしても使いたい方は,coLinuxかVMware Playerを使ってKNOPPIX/Mathで遊んで下さい.

2.3　Maximaからsurfを使う方法

Maximaからsurfを使う最も簡単な方法はsystem函数 を用いてsurfを呼出す事です. 勿論, 単純に呼出しても Maxima側の多項式がsurfに引渡される事はありません. そこで, 中間ファイルをMaxima側で生成し, それをsurfに引き渡す方法を採用します. この時,surfの起動時のオプションの-xを使って生成したファイルをsurfで処理させます. 即ち, `surf -x 〈ファイル〉`を実

行すると,surf は起動時に指定したファイルを読込み, そのファイルに記述されたスクリプトを実行します.

そうとなれば Maxima で surf 用のスクリプトを生成し, ファイルに書出す函数を構築すれば良いのです, 最も簡単な方法は, 多項式の変数を完全に x,y,z に限定し, 変数が 2 個あれば曲線,3 個ならば曲面を描く様に予め用意した surf の設定ファイルに曲線や曲面の方程式として多項式を引渡せば良いのです.

但し, この方法では x, y, z 以外の変数の多項式が使えません. 更に, 設定ファイルを別途準備するのもあまりエレガントではありませんね. その上, 設定ファイルの内容を必要に応じて Maxima から変更出来ないのも面白くありません. ここではもう少し Maxima に複雑な処理を行わせましょう.

2.4 surf の設定の取込み方法

surf では, 曲線や曲面を描く際に, 方程式が満す解を求める為に, 解法の設定, 解の精度や反復の回数, そして, 絵の大きさを各々, root_finder, epsilon,iterations ,width と height で行います. これらを surf に引渡す必要があります. 尚, これらの値は以下の値で十分でしょう.

求解に関するパラメータとウィンドウの大きさ

```
root_finder=d_chain_bisection;
epsilon=0.0000000001;
iterations=20000;
width=500; height=500;
set_size;
```

先ず,root_finder に指定可能な解法は色々ありますが, 自己交差を持つ曲面, 曲面と平面との断面が綺麗に描ける d_chain_bisection をここでは採用します. 絵の大きさはデフォルトの 200×200 では小さ過ぎるので見栄えの良い 500×500 で表示させる事にしましょう.

曲面の場合は上述の設定に加えて, 以下の設定を追加します.

曲面の場合の設定

```
do_background=yes;
background_red=255;
background_green=255;
background_blue=255;
rot_x=0.14; rot_y=-0.3;rot_z=1.0
scale_x=1.0; scale_y=1.0;scale_z=1.0;
```

先ず, 最初の 4 個のパラメータは背景の描画と背景色を指定するものです. 次に, rot_x, rot_y と rot_z は各々曲面を X 軸,Y 軸と Z 軸回りに回転させるパラメータで, 最後の scale_x, scale_y と scale_z は各々曲面を X 軸,Y 軸と Z 軸方向の倍率になります.

これらの値をファイルに書込むのが面倒であれば, 大域変数にしてしまえば便利です. しかし,Maxima の属性を用いるともっと扱い易くなります.

今回,Maxima の初期化ファイル maxima-init.mac に以下の設定を行います.

maxima-init.mac での属性設定

```
put(surfg, "d_chain_bisection",root_finder);
put(surfg, 20000,iterations);
put(surfg, 500,width);
put(surfg, 500,height);

put(surf, "yes",do_background);
put(surf, 255,background_red);
put(surf, 255,background_green);
put(surf, 255,background_blue);
put(surf, 0.14,rot_x);
put(surf, -0.3,rot_y);
put(surf, 0.0,rot_z);
put(surf, 1.0,scale_x);
put(surf, 1.0,scale_y);
put(surf, 1.0,scale_z);
```

ここでは Maxima の put 函数を用いて大域変数 surfg と大域変数 surf の属性として,surf の描画パラメータを指定します. 例えば, put(surf, -0.3,rot_y) で, 変数 surf の属性 rot_y に値-0.3 が設定されます.

初期設定の変更も同じ方法で値を変更すれば出来ます. これらの属性値を Maxima で取出す場合,get 函数を用います. 例えば,surf の background_red 属性に 255 を設定したければ, put(surf, 255,background_red); と入力します. 又,surf の background_red の属性値は get(surf,backgraound_red); で得られます.

これで, 二次元と三次元の共用の設定が属性と含まれる大域変数 surfg と曲面専用の設定が含まれる大域変数 surf の属性が設定されます.

次に,surf の描画ファイルに大域変数 surf と surfg に記述した属性を書出を行う部分の構成を考えましょう. 先ず,surf と surfg の属性は 10 個あります. この属性を一々函数に書込むのは流石に面倒です. そこで, properties 函数を使って,surf と surfg の属性に何があるか一覧のグラフを出させ, その属性名と

属性値の等式を作る方法を考えましょう.

要するに, 次の処理を自動的に行う様に do 文を組込めば良いのです.

```
(%i18) properties(surfg);
(%o18) [["user properties",height,width,iterations,root_finder]]
(%i19) properties(surf);
(%o19) [["user properties",scale_z, scale_y, scale_x, rot_z,rot_y,rot_x,
            background_blue,background_green,background_red,do_background]]
(%i20) %[1][6]=get(surf,%[1][6]);
(%o20) rot_y = -0.3
```

この例では,properties 函数で大域変数の属性リストを出させます. この属性リストは通常, 第一成分のリストに属性名が登録されています. 先頭は `"user properties"`なので, 第 2 番目以降の成分を演算子=の左側に置き, その成分の属性値を演算子=の右側に置くと surf 向けの等式が出来る訳です.

この処理で構築した等式 (上の例では,rot_y=-0.3) を Maxima の配列に入れて, stringout 函数でこの配列の内容をファイルに書出してしまえば良いのです. この様にしておけば, 別に変数名 (Maxima では属性) を憶えていなくても, 機械的に処理が行えるので, その属性を減らしたり増したりするのも容易になりますね.

2.5 多項式の処理

surf で用いる数値で $1.0e-3$ は使えますが,Maxima の様に bigfloat 等の数値はありません. そのまま Maxima の不動点少数を出力させると書式上の問題から,surf ではエラーになります. そこで, この様な余計な問題を避ける為, 多項式を expand 函数で一旦展開した後に,ratsimp 函数で簡約化します.ratsimp 函数を使うと多項式の係数は有理数に近似されます. その為,surf の処理で問題は一切生じなくなります.

次に重要な事ですが,surf で利用可能な多項式は基本的に変数 x,y,z の多項式に限定されます. 勿論,surf は x,y,z 以外の変数も扱えますが, その様な変数は内部での補助的な変数として扱われ, あくまでも描かれるグラフは XY 平面, 或いは XYZ 空間に限定されます.

その為, 多項式の変数を 2 か 3 個に限定する必要があり, x,y,z 以外の変数が使われていれば, それらを x,y,z で置換える操作もあった方が良いでしょう.

ここで, 多項式の変数を返す函数として Maxima には showratvars 函数があります.

例えば, `showratvars(x+y^2+a^3);` と入力すると, `[a,x,y]` が返されます. このリストは Maxima の変数順序 $>_m$ に対して小さな順で変数が左から

並びます. 因に,Maxima の変数順序 $>_m$ は一般的には辞書式順序と呼ばれる順序で, 数字よりもアルファベットが大きく, 同じ文字の場合は大文字の方が小文字よりも大きく, アルファベットを大のものから順に並べると逆アルファベット順になります. 順序の詳細に関しては, マニュアルを参照して下さい.

ところで, 式に `sqrt(2)` といった式が含まれていると, 実際は数値となる式も showratvars 函数は変数リストに含めて出力してしまいます. そこで, この様な事が生じない様に予め float 函数を使って式を評価しておきます.

こうして得られた変数リストで, 変数は Maxima の変数順序 $>_m$ に従って並べられますが,Maxima の変数リストの並べ方は基本的に, $>_m$ に対して小さい順に並べます. その為, 実際は通常のアルファベット順になります. そこで,showratvars 函数で得られた変数リスト vars の左側から順番に x,y,z を対応させれば良い事になります.

具体的には $vars[1]$ を x, $vars[2]$ を y, $vars[3]$ を z で置換えるので,subst 函数が使えそうです. ところが,subst 函数では, 置換リストを与えると左から右へと代入が行われる為,

subst([a=x,x=y,y=z],a+x+y) の結果は困った事に 3*z になってしまいます.

そこで, 置換を二段階で行います. 函数に局所的変数 `surf_tmp_x`, `surf_tmp_y` と `surf_tmp_z` に一旦置換え, これらを $[x, y, z]$ に最終的に置換える事で多項式を変数 x,y,z の多項式に変換します. 以上で変数変換を終えます.

実際に surf で曲線や曲面を描く為には, その方程式に対応する surf の描画命令をスクリプトの末尾に追加しなければなりません. この処理は曲線と曲面で異なるので, 曲線と曲面の場合に分けて surf の描画命令を追加します.

曲線の場合, 多項式を変数 curve に割当てて, 最後に draw_curve 命令にを追加します. 又, 曲面であれば, 多項式を変数 surface に割当て, 最後に描画命令である draw_curve 命令を追加します.

この描画命令を追加したスクリプトを,Maxima の system 函数から,surf に実行させる事が出来ます.

この考えで構築した Maxima の函数 surfplot を以下に示します.

```
/* Maxima */

/* 属性の設定.*/
/* surfg の属性設定

   surfg        平面曲線と空間曲面を描く際に用いる共通の設定.
   root_finder  零点を計算する際の, 解法の指定.
                d_chain_bisection を用いると自己交差も綺麗に
                描きます.
   iterations:  零点を計算する際の繰返しの上限を設定
```

```
   width:       画像の横幅
   height:      画像の高さ
*/
put(surfg, d_chain_bisection,root_finder);
put(surfg, 0.0000000001,epsilon);
put(surfg, 20000,iterations);
put(surfg, 500,width);
put(surfg, 500,height);

/* surf の属性設定

   surf には空間曲面を描く際に用いる設定を入れます.
   surf で色の指定は 0 から 255 までの整数で RGB を指定します.
   do_background:     背景色の設定を許可 (yes)
   background_red:    背景色の指定 (赤)
   background_green:  背景色の指定 (緑)
   background_blue:   背景色の指定 (青)
   rot_x:             X 軸回りの回転角度 (rad)
   rot_y:             X 軸回りの回転角度 (rad)
   rot_z:             Z 軸回りの回転角度 (rad)
   scale_x:           X 軸方向の倍率
   scale_y:           Y 軸方向の倍率
   scale_z:           Z 軸方向の倍率

*/
put(surf, yes,do_background);
put(surf, 0,background_red);
put(surf, 0,background_green);
put(surf, 0,background_blue);
put(surf, 0.14,rot_x);
put(surf, -0.3,rot_y);
put(surf, 0.0, rot_z);
put(surf, 1.0, scale_x);
put(surf, 1.0, scale_y);
put(surf, 1.0, scale_z);

/* surfplot

   引数は多項式. 多項式の変数は 2 個か 3 個でなければエラーに
```

```
   なります. 描画は surf を用いますが, この函数では, 臨時ファイル
   として surf.tmp に surf のスクリプトを書込み,system 函数で
   surf -x surf.tmp を実行させます.
*/

surfplot(f):=block(
[
 poly,poly0,vars,lls1:0,lls2:0,
 f:ratsimp(expand(f)),tmp,
 str,target,j,sl,obj,
 ls1:properties(surfg),
 ls2:properties(surf)
],
vars:showratvars(float(f)),
n:length(vars),
display2d:false,
if n=2 or n=3 then
  (lls1:length(ls1[1])-1,
   for i:1 thru lls1 do
      (str:ls1[1][i+1],
      (if str=epsilon then tmp:rat(get(surfg,str))
          else tmp:get(surfg,str)),
       surf_settings[i]:str=tmp
       ),
   if n=3 then
     (lls2:length(ls2[1])-1,
      for i:1 thru lls2 do
         (str:ls2[1][i+1],
          j:i+lls1,
          surf_settings[j]:str=get(surf,str)
          ),
       /* 変数の入換を行います.*/
       poly0:subst([vars[1]=surf_tmp_x,vars[2]=surf_tmp_y,
                    vars[3]=surf_tmp_z],f),
       poly:subst([surf_tmp_x=x,surf_tmp_y=y,surf_tmp_z=z],
                    poly0),
       /* 曲面を描く為の設定 */
       target:surface,
       obj:draw_surface)
```

```
   else
      (
       /* 変数の入換を行います.*/
      poly0:subst([vars[1]=surf_tmp_x,vars[2]=surf_tmp_y],f),
      poly:subst([surf_tmp_x=x,surf_tmp_y=y],poly0),
       /* 曲線を描く為の設定 */
       target:curve,
       obj:draw_curve),
  /* 配列 sl の定義. 描画設定と曲線/曲面の方程式+描画命令を割当 */
   j:lls1+lls2,
   array(sl,j+2),
  (for i:0 thru j-1 do
       sl[i]:surf_settings[i+1]),
   sl[j]:target=poly,
   sl[j+1]:obj,
   /* 只今, 描画中… */
   print("Surf is now drawing ", poly,". Please wait ...."),
   /* stringout 函数で, スクリプトの式を書込みます. */
   if n=2 then
     (stringout("surf.tmp",sl[0],sl[1],sl[2],sl[3],sl[4],
                           sl[5],sl[6]))
   else
     (stringout("surf.tmp",sl[0],sl[1],sl[2],sl[3],sl[4],
                           sl[5],sl[6],sl[7],sl[8],sl[9],
                           sl[10],sl[11],sl[12],sl[13],
                           sl[14],sl[15],sl[16])),
   /* system 函数で surf を起動します.XView 版の surf の場合は,
      -x オプションを外すと動く様です */
   system("surf -x surf.tmp")
   )
else
/* 多項式が,2 変数でも 3 変数でもなければエラーを表示します. */
print("Error!"))$
```

この函数を使うと 2 変数多項式や 3 変数多項式の零点のグラフが描けます. surfplot を実行すると, 直ちに surf が立ち上ります. 描画は計算機の能力と式の複雑さに依存する為, どの程度かかるかは一概に言えませんが, 最近の Pentium 3 の 1GHz 程度でもあれば, 余程複雑な曲面でもない限り,30 秒もあれば描画が完了します.

描いた絵は surf の側で保存も出来ます. この様な操作は描画ウィンドウ上で, マウスの右ボタンを押すと, 制御用のウィンドウが出るので, そこで色々操作を行います. 詳細は surf のマニュアルを参照して下さい.

surf の終了は描画ウィンドウ上でキーボードから q を入力すれば,surf を終了して Maxima に制御が戻ります.

では, 色々曲面や曲線を描いて遊んでみましょう.

2.6 簡単な例

最初に半径が 1 の球面を描いてみましょう. 球面の方程式は $x^2+y^2+z^2-1=0$ で与えられます. 従って, surfplot(x^ 2+y^ 2+z^ 2-1) と入力しましょう. すると surf が立ち上がって球面を描きます. これだけでは面白く無いので, もう少し捻りの効いた方程式を描いてみましょう. では, 次の 3 個の球面の方程式の積はどの様な曲面を描くでしょうか？

3 個の球面の方程式の積

$$\left(x^2+y^2+z^2-4\right)\cdot\left((x-2)^2+(y-2)^2+z^2-\sqrt{2}\right)\cdot \left((x+2)^2+(y-2)^2+z^2-\sqrt{2}\right)$$

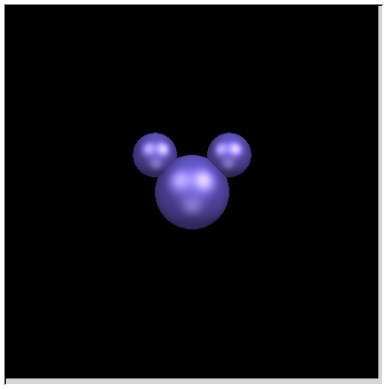


図 2.1: ？曲面

この 3 個の球面の方程式の積で表現される式は図 2.1 に示す 3 個の球面で構成された曲面になります. それにしても, この曲面は何処かで見た事のある様な気がしますね.

この曲面は原点を中心とする半径 2 の球面を中心に置いて, $(2,2,0)$ と $(-2,2,0)$ を中心とする半径 $\sqrt{2}$ の球面で出来ています. この様に複数の曲面の方程式の積で出来た方程式から得られる曲面は各曲面の和集合になります.

正確に言えば, 多項式 $f \in K[x_1, \cdots, x_n]$ に対し, $V(f)$ を多項式 f の零点集合とします. この時,$V(f_1 \cdot f_2 \cdots f_n)$ は $V(f_1) \cup V(f_2) \cup \cdots \cup V(f_n)$ となります.

従って, 多項式の零点集合を描く事は, 多項式の各因子の零点集合の和集合を描く事になります. この事から, 多項式の既約因子分解が出来ると, 後は各既約な多項式を調べてしまえば十分な事が判ります. 更に, 既約な元は多項式環 $K[x_1, \cdots, x_n]$ の素イデアルの生成元となるので, 結局, 曲線や曲面は素イデアルと密接に関連する事になります.

この事を怪しい絵を描く方法に適用するのであれば, 片っ端から描きたい多項式の積を surf で描かせれば, 怪しい絵が作られる事になります. 例えば, 以下の 3 個の円と楕円の積を描いたものは図 2.2 に示すアヒルの顔の様なものになりますね.

3 個の円の方程式と楕円の積

$(x^2 + y^2 - 9) \cdot ((x-2)^2 + (y-1)^2 - 1) \cdot ((x+2)^2 + (y-1)^2 - 1) \cdot$
$(x^2/9 + 2 * (y+1)^2 - 1)$

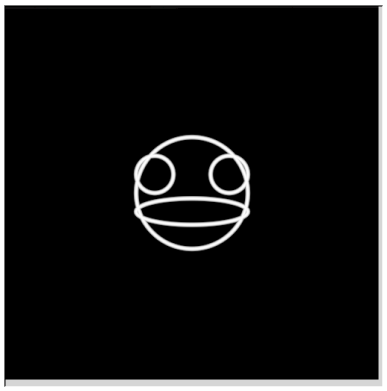


図 2.2: アヒル？

この様な巫山戯た使い方だけではなく, 座標を付けて曲線を描きたい場合にも使えます. この場合, 座標軸の方程式と曲線の方程式の積を描けば良いのです. 例えば, 原点を通る X 軸と Y 軸を描きたいのであれば,X 軸の方程式が $y = 0$,Y 軸の方程式が $x = 0$ となるので, 描きたい曲線に $x * y$ をかけたものを描けば良いのです. 例えば, `surfplot((x^ 2*(9-x^ 2)-4*y^ 2)*x*y);` と入力した結果を図 2.3 に示します.

この様にマジメな使い方も出来ます. が, 色々と巫山戯た使い方をしてみるのも楽しいかと思います.

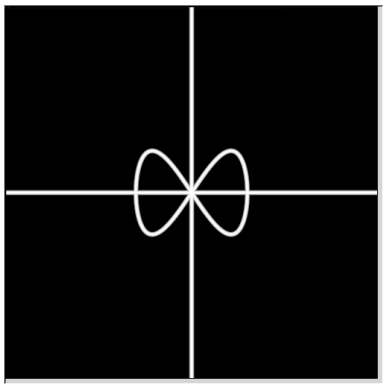


図 2.3: レムニスケート

2.7 Steiner のローマ曲面ギャラリー

今度は Steiner のローマ曲面を描いてみましょう. ローマ曲面は $x^2y^2 + x^2z^2 + y^2z^2 - 17xyz = 0$ を満す点の集合で, この曲面は写像 $f : (x, y, z) \to (xy, yz, zx)$ で原点中心の半径 1 の球面を写したものです.

この曲面は二次元の実射影空間の三次元空間への嵌め込みと呼ばれる平面の三次元空間での置かれ方の一つになります. 因に, 二次元の射影空間は Moebius の帯の境界となる円周に円盤を貼って出来る曲面に同相になりますが, この曲面は通常の三次元空間では実現出来ません. どうしても, 自分自身が交わる点が出来てしまいます. その辺が面白い曲面です.

最初に, 上記のファイル (surfplot.mc とします) を読込みましょう. 尚,Maxima の利用者定義の函数を含むファイルは load 函数で読込めます.surfplot.mc が Maxima を起動したディレクトリ上にある場合, `load("surfplot.mc");` で Maxima に読込みます. これでエラーも何も出なければ問題ありません. カレントディレクトリに surfplot.mc が無ければ, surfplot.mc を置いたディレクトリを直接指定します. この場合, 相対パス (カレントディレクトリを基準とするパス) でも絶対パス (ルートディレクトリを基準とするパス) でもどちらでも構いません.

では, 早速描いてみましょう.

`surfplot(x^ 2*y^ 2+x^ 2*z^ 2+y^ 2*z^ 2-17*x*y*z);` と入力して下さい.

すると, この函数はカレントディレクトリ上にデータファイル surf.tmp を生成し, surf に引渡し,surf が起動します. その結果, 図 2.4 に示す絵が得られます.

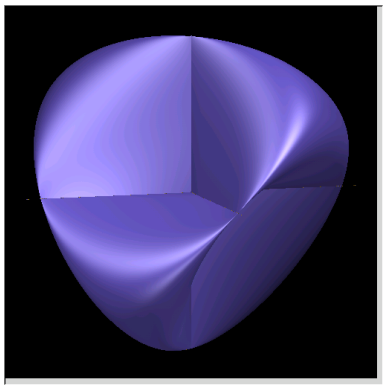


図 2.4: Steiner のローマ曲面

この絵で判る様に, ローマ曲面は X,Y,Z 軸で潰れた球面の様な形をしています. これだけでは何がどうなっているのか判りませんね. では, 曲面を平面で切った断面を見てみましょう. この平面による曲面の断面は Maxima+surf で簡単に見られます.

その為には, 断面の方程式を Maxima で計算しなければなりません. これは実は簡単な事です. 平面の方程式は (0,0,0) と異なる 3 個の実数 (a, b, c) と実数 d を用いて,$ax + by + cz - d = 0$ で表現されます. ここで,a, b, c のどれか一つは 0 でないので, ここでは c が零でないとしましょう. すると, 方程式を c で割る事で, $z + ax + by - c = 0$ の形式の方程式が得られます. そこで, z に $c - ax - by$ を代入した式を Maxima で計算し, その式を描けば良いのです.

具体的に XY 平面 $z = 0$ による断面はどうなるでしょうか？この場合, 断面の方程式として x^2y^2 が得られるので, 零点の集合として,X 軸と Y 軸が描かれてしまいます.

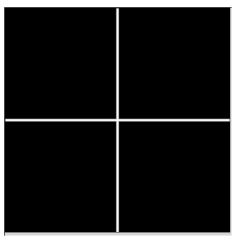
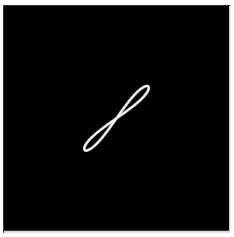


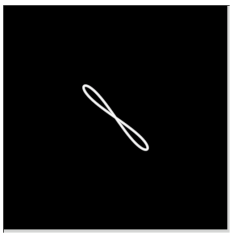
図 2.5: Steiner のローマ曲面,Z=0

以下に,z を [-8,-2,-0.1,0,0.1,2,8] で描いた結果を以降に示しておきます.

これらの断面図は単純に Z 軸を法線とする平面で描いたものですが, この曲面の面白さが出ています. $z = 0$ を除くと, 断面は 8 の字になっています. $z = 0$ に近付くにつれて,8 の字の上下が潰れて, 交差点付近が XY 軸に貼り付

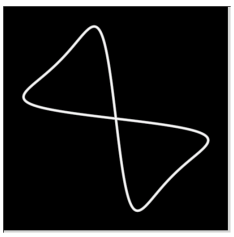


Steiner のローマ曲面,Z=8

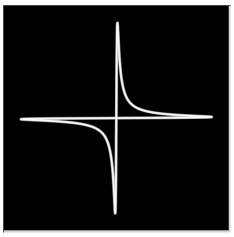


Steiner のローマ曲面,Z=-8

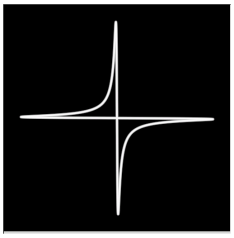
Steiner のローマ曲面,Z=2



Steiner のローマ曲面,Z=-2



Steiner のローマ曲面,Z=0.1



Steiner のローマ曲面,Z=-0.1

き, $z = 0$ を通り過ぎると, 再び 8 の字が出現しますが, 今度は潰れた 8 の字が 45 度回転した形で出現します. この曲線で自分自身が交わる点で, 接ベクトルの次元は 2 になります.

2.8 Maxima で終結式を使ってみよう

助変数表示された曲面や曲線の場合, 終結式を使って助変数を削除する事が可能です. Maxima には終結式を計算する resultant 函数があります. この resultant 函数の構文を以下に示します.

終結式を求める resultant 函数

$resultant(\langle$ 多項式$_1\rangle, \langle$ 多項式$_2\rangle, \langle$ 変数 $\rangle)$

resultant 函数は引数として, 二つの多項式と一つの変数を必要とします. ここで指定する変数は, 二つの多項式から消去したい変数になります. この終結式は, 二つの多項式を指定した変数の多項式として並び換えを行います.

Maxima では resultant 函数は bezout 函数と determinant 函数の合成で表現可能です.

ではデカルトの葉状曲線を用いて順番に作業を追ってみましょう.

```
(%i1) p1:(1+t^3)*x-3*t;
                                3
(%o1)                          (t  + 1) x - 3 t
(%i2) p2:(1+t^3)*y-3*t^2;
                                3             2
(%o2)                          (t  + 1) y - 3 t
(%i3) expand(p1);
                                3
(%o3)                           t  x + x - 3 t
(%i4) expand(p2);
                                3           2
(%o4)                           t  y + y - 3 t
(%i5) m1:matrix([x,0,-3,x,0,0],
                [0,x,0,-3,x,0],
                [0,0,x,0,-3,x],
                [y,-3,0,y,0,0],
                [0,y,-3,0,y,0],
                [0,0,y,-3,0,y]);
```

```
                        [ x   0   - 3   x    0   0 ]
                        [                          ]
                        [ 0   x    0   - 3   x   0 ]
                        [                          ]
                        [ 0   0    x    0   - 3  x ]
(%o5)                   [                          ]
                        [ y  - 3   0    y    0   0 ]
                        [                          ]
                        [ 0   y   - 3   0    y   0 ]
                        [                          ]
                        [ 0   0    y   - 3   0   y ]
(%i6) expand(determinant(m1));
                                    3                3
(%o6)                        - 27 y  + 81 x y - 27 x
```

この例では, 最初に二つの多項式多項式 $p1 = (1+t^3)x - 3t$ と $p2 = (1+t^3)y - 3t^2$ を定義しています. ここで, 多項式 $p1$ と多項式 $p2$ を展開すると, 各々, $p1 = xt^3 + 0t^2 - 3t + x$ と $p1 = yt^3 - 3t^2 + 0t + y$ とになります. 多項式 $p1$ と $p2$ の終結式は, この t の多項式と看做した場合の係数を並べたもので得られます. 両方の多項式の次数が 3 の為, 構築する行列 $m1$ は次の 6 次の正方行列になります.

$$m1 = \begin{bmatrix} x & 0 & -3 & x & 0 & 0 \\ 0 & x & 0 & -3 & x & 0 \\ 0 & 0 & x & 0 & -3 & x \\ y & -3 & 0 & y & 0 & 0 \\ 0 & y & -3 & 0 & y & 0 \\ 0 & 0 & y & -3 & 0 & y \end{bmatrix}$$

この行列の行列式を計算したものが終結式になります.

ここで, この終結式には面白い性質があります. それは終結式と二つの多項式の解の関係を示すものです.

一般的には, 変数 x を主変数とする多項式 f と g が以下で表現されるものとします.

$$f = \sum_{i=0}^{m} a_i x^i$$

$$g = \sum_{i=0}^{n} b_i x^i$$

この二つの多項式の終結式は次で表現されます.

$$resultant(f,g,x) = \det \begin{pmatrix} a_m & a_{m-1} & \cdots & \cdots & a_1 & a_0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & & \ddots & \ddots & \vdots \\ 0 & \cdots & a_m & a_{m-1} & \cdots & \cdots & a_1 & a_0 \\ b_n & b_{n-1} & \cdots & \cdots & b_1 & b_0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & & \ddots & \ddots & \vdots \\ 0 & \cdots & b_n & b_{n-1} & \cdots & \cdots & b_1 & b_0 \end{pmatrix}$$

ここで, 多項式 f と g の解を各々α_i,β_j とすると, 多項式 f と g の終結式は解 α_i,β_j を用いて以下の式に等しくなります.

$$resultant(f,g,x) = a_m^n b_n^m \prod_{1\le i\le m, 1\le j\le n} (\alpha_i - \beta_j)$$

この事は, 多項式 f と g が共通の解を持ては終結式が常に 0 になり, 同時に, 終結式が 0 になるのは多項式 f と g が共通の解を持つ時に限る事を意味します. ここでは, 零点集合を表現する多項式を計算するのが目的なので, 終結式の零点集合と, 多項式 $p1$ と $p2$ が同時に 0 になる零点集合は一致しなければなりません. この性質を利用して, 終結式を使って助変数を排除しているのです.

尚,Maxima ではこの行列を効率良い行列で書き出す bezout 函数もあります. この函数で, 上記の多項式 $p1$ と $p2$ の係数から構成される行列と, determinant 函数による結果を示しておきます.

```
(%i7) bezout(p1,p2,t);
                              [   0    3 x  - 3 y ]
                              [                   ]
(%o7)                         [ 3 y   - 9   3 x  ]
                              [                   ]
                              [ - 3 x  3 y    0   ]
(%i8) expand(determinant(%));
                                    3                3
(%o8)                        - 27 y  + 81 x y - 27 x
```

この計算を resultant 命令で一度に実行したものを以下に示します.

```
(%i5) p1:(1+t^3)*x-3*t;
                                    3
(%o5)                            (t  + 1) x - 3 t
(%i6) p2:(1+t^3)*y-3*t^2;
```

```
                                    3             2
(%o6)                         (t  + 1) y - 3 t
(%i7) i1:resultant(p1,p2,t);
                                      3           3
(%o7)                      - 27 (y  - 3 x y + x )
```

ここで定数倍は無視しても構いません. 実際, a を定数, $f(x_1,\cdots,x_n)$ を多項式とする時, $f(x_1,\cdots,x_n)$ の零点集合と, $af(x_1,\cdots,x_n)$ の零点集合は一致するからです.

一寸, 脇道に逸れますが, 多項式 f の零点集合 $V(f)$ と多項式 $g(x_1,\cdots,x_n)$ と多項式 $f(x_1,\cdots,x_n)$ の積の零点集合 $V(fg)$ の関係を思い出して下さい. $V(fg)$ は多項式 f と g の零点の両方を含み, $V(f)\cup V(g)$ になります. 更に, f で割切れる多項式 h の零点集合 $V(h)$ に対しては, $V(h)\supset V(f)$ が成立します.

ここで, 多項式 f で生成されるイデアル (f) を考えると, $V(f)$ がイデアル (f) に含まれる多項式の零点集合の中で最小の集合になります. このイデアルで考えてしまえば, 多項式が定数倍である事は大きな問題となりません. ですから, 多項式の零点集合を考える場合, そのイデアルを考えるのが最も手軽な手段になります.

次に, 応用で猿の腰掛と呼ばれる曲面を表示してみましょう. この曲面は助変数表示では以下の関係式を満すものです.

猿の腰掛の助変数表示

$$x-u=0$$
$$y-v=0$$
$$z-u^3+3uv^2=0$$

本当は, 最後の式の変数 u , v を x と y で各々置換えてしまえば済む話なのですが, ここでは, 終結式を使って猿の腰掛の変数 x , y,z の式に変換してみましょう. ここで, 注意する事は, 迂闊に,resultant(x-u,y-v,u) の様に片方の式にしか存在しない変数を使ってはいけません. 無意味な式が返って来るだけです. この例では, 変数 u と v が含まれているのは, $z-u^3+3uv^2=0$ のみなので,resultant はこの多項式を中心に, 例えば, $x-u$ から開始し, $y-v$ で終える手順になります.

```
(%i1) p1:x-u;
(%o1)                                  x - u
(%i2) p2:y-v;
(%o2)                                  y - v
(%i3) p3:z-u^3+3*u*v^2;
```

```
                                          2    3
(%o3)                          z + 3 u v  - u
(%i4) a1:resultant(p1,p3,u);
                                        3      2
(%o4)                          - z + x  - 3 v  x
(%i5) a2:resultant(a1,p2,v);
                                          2    3
(%o5)                          - z - 3 x y  + x
(%i6) surfplot(a2);
```

これで多項式として, $-z-3xy^2+x^3$ が得られました. このグラフを surf で描いたものが図 2.6 に示す, 尻尾の生えた猿に丁度良さそうな腰掛みたいな曲面が描かれます.

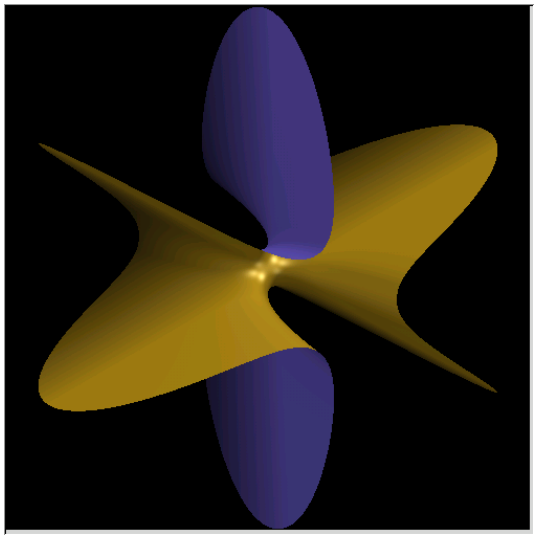


図 2.6: 猿の腰掛

尚,Maxima には eliminate 函数が存在するので, 実際は,
eliminate([x-u,y-v,z-u;3+3*u*v^ 2],[u,v]); で十分です.
試しに, 動作を確認してみましょう.

```
(%i13) eliminate([x-u,y-v,z-u^3+3*u*v^2],[u,v]);
                                          2    3
(%o13)                        [- z - 3 x y  + x ]
(%i14) %[1];
```

```
                                          2    3
(%o14)                          - z - 3 x y  + x
```

長々とした計算を行っていましたが,eliminate 函数を使えば, 簡単に出来る事でした. ご苦労さまでした.

関連図書

[1] J. リヒター-ゲバート/U.H. コルテンカンプ著, 阿原一志訳, シンデレラ幾何学のためのグラフィックス, シュプリンガー・フェアラーク東京,2001.

[2] 河内明夫編, 結び目理論, シュプリンガーフェラーク東京,1990.

[3] 下地貞夫, 数式処理, 基礎情報工学シリーズ, 森北出版,1991.

[4] クロウエル, フォックス, 結び目理論入門, 現代数学全書, 岩波書店,1989.

[5] ポール・グレアム,ANSI Common Lisp, ピアソン・エデュケーション,2002.

[6] 本間龍雄, 組合せ位相幾何学, 共立出版,1980.

[7] 寺坂英孝編, 現代数学小辞典, ブルーバックス, 講談社,2005.

[8] 丸山茂樹, クレブナー基底とその応用, 共立叢書 現代数学の潮流, 共立出版,2002.

[9] 村上順, 結び目と量子群, 数学の風景 3, 朝倉書店,2000.

[10] 日本数学会編, 数学辞典 第 3 版, 岩波書店,1987.

[11] H.Cohen, A Course in Computational Algebraic Number Theory,GTM 138, Springer-Verlag,New York-Berlin,2000.

[12] D.Cox,J.Little and D. O'Shea,Ideals, Varieties, and Algorithms, UTM,Springer-Verlag,New York-Berlin,1992.

[13] Gert-Martin Greuel, Gerhard Pfister, A Singular Introduction to Commutative Algebra, Springer-Verlag,New York-Heiderberg-Berlin,2000.

[14] D.Rolfsen, Knots and Links. Publish or Perish, Inc,1975.

[15] Hal Schenck, Computational Algebraic Geometry London Mathematical Society student texts;58,2003.

[16] Open AXIOM のサイト http://wiki.axiom-developer.org/FrontPage

[17] DERIVE のサイト http://www.derive.com

[18] GAP のサイト http://www-gap.dcs.st-and.ac.uk/

[19] Macaulay2 のサイト http://www.math.uiuc.edu/Macaulay2/

[20] Mathsoft Engineering & Education, Inc. http://www.mathcad.com/

[21] Wolfram Research Inc. http://www.wolfram.com/

[22] Waterloo Maple Inc. http://www.maplesoft.com/

[23] Maxima の SOURCEFORGE のサイト http://maxima.sourceforge.net/

[24] MuPAD のサイト http://www.mupad.de/

[25] PARI/GP のサイト http://pari.math.u-bordeaux.fr/

[26] REDUCE のサイト http://www.zib.de/Symbolik/reduce/

[27] Risa/Asir 神戸版 http://www.math.kobe-u.ac.jp/Asir/asir-ja.html

[28] OpenXM(Open message eXchange for Mathematics)
http://www.math.sci.kobe-u.ac.jp/OpenXM/index-ja.html

[29] SINGULAR のサイト http://www.singular.uni-kl.de/

[30] 株式会社シンプレックスのページ htto://www.simplex-soft.com/

[31] MathWorks,Inc. http://www.mathworks.com/

[32] Octave WebPage http://bevo.che.wisc.edu/octave/

[33] Scilab のサイト http://www.scilab.org/

[34] Yorick の公式サイト ftp://ftp-icf.llnl.gov/pub/Yorick/
Yorick の非公式サイト http://www.maumae.net/yorick/doc/index.php

[35] R のサイト http://www.r-project.org

[36] Insightful Corporation のページ http://www.insightful.com/

[37] Cinderella のサイト
本家：http://www.cinderella.de/
日本語版ホームページ http://cdyjapan.hp.infoseek.co.jp/

[38] dynagraph のサイト http://www.math.umbc.edu/ rouben/dynagraph

[39] KSEG のサイト http://www.mit.edu/ ibaran/kseg.html

[40] Geomview のサイト http://http.geomview.org/

[41] surf の sourceforge のサイト http://surf.sourceforge.net/

[42] XaoS のサイト http://wmi.math.u-szeged.hu/ kovzol/xaos

索 引