代数曲線のトポロジーとその計算ソフトウェア「モノミー君」について

メタデータ	言語: Japanese
	出版者: 明治大学科学技術研究所
	公開日: 2012-06-20
	キーワード (Ja):
	キーワード (En):
	作成者: 阿原, 一志
	メールアドレス:
	所属:
URL	http://hdl.handle.net/10291/13032

明治大学科学技術研究所紀要
34(3)：19－46. 1995

Mem. Inst. Sci. Tech. Meiji Univ.
34(3)：19－46. 1995

代数曲線のトポロジーとその計算ソフトウェア「モノミー君」について。

阿原一志

Topology of Algebraic curves and a computer software 'Mr. Monomie'.

Kazushi AHARA

School of Science and Technology, Meiji University
1-1-1, Higashimita, Tama-ku, Kawasaki, 214

Received June 14, 1995; Accepted October 6, 1995

Synopsis : There has been much interest in the study of degeneration of Riemann surfaces. If genus of general fibers is small, it is well known that classification of singular fibers is completed by Kodaira[K]. The author is making a computer software 'Mr. Monomie' to calculate homeo-types of singular fibers and their monodromies. The latest version of this software is 1.4i. In this paper we introduce how to use this software, some algorithms in the programm, and some applications for topological problems.

Key words : algebraic curve, degeneration of Riemann surface, C++ Language, Singularity.

目次

0 序論

代数曲線の退化の問題は最近興味を持たれている問題の一つである。有理曲線の場合および楕円曲線の場合には小平 [K] の研究によってすべて調べられた。ジーナスが高くなるにつれて、退化の分類の問題は難しさを増すが、松本-Montesinos[MM] は擬周期同相写像の分類問題に帰結することによって位相幾何的な解決を見た。一方で、具体的に代数曲面から $\mathbf{P}^1$ へのファイバリングが与えられた時に、その特異ファイバーと大域的モノドロミーを計算する試みもなされた。阿原 [A]、松本 [M] は5次のフェルマー型代数曲線からのファイバリングについてそれらを計算した。

「モノミー君」(Mr. Monomie) は Macintosh（米国アップルコンピュータ社）の上で動くコンピュータソフトウェアで、与えられた代数曲面から $\mathbf{P}^1$ へのファイバリングに対してその特異ファイバーと退化の分類と大域的モノドロミーを計算することを目的として開発が始められた。現在のバージョンは 1.4i であって、代数曲線の分類を位相幾何的に見ることが出来る機能を持っている。

0.1 モノミー君の機能

「モノミー君」の主な用途は、与えられた代数曲線や代数曲面に対して、その位相幾何的情報を計算することである。バージョン 1.4i の段階では次のことができる。

0.1.1 代数曲線のトポロジーを調べる

1. $\mathbf{P}^2(\mathrm{C})$ 上の代数曲線を、2変数多項式の形で入力できる。

2. 一方の変数への射影を考え、その branch locus の場所を計算し、その各々の branch point が滑らかであるか、crossing になっているか、その他か、を判定する。

3. $\mathrm{C}-\{\text{branch points}\}$ の（基点つき）基本群の生成元を自動的に与える。

4. branch covering の monodromy を置換群の元として与える。

5. 代数曲線の位相的情報を与える。

0.2 実例

具体例で見ると、次のようになる。例えば、今 $M=\{[x:y:z]\in\mathbf{P}^2(\mathrm{C})\mid x^3+y^3+z^3=0\}$ を考えよう。この曲線はトーラスと同相であることは良く知られている。これを $\{[x:y:1]\in\mathbf{P}^2(\mathrm{C})\}$ に制限して、$M'=\{(x,y)\in\mathrm{C}^2\mid x^3+y^3+1=0\}$ とし、$\pi:M'\to\mathrm{C}:(x,y)\mapsto y$ という写像を考える。π は M 上への拡張が可能であることはいうまでもない。この写像は branch covering になり、branch locus は $y^3+1=0$ の3つの根となる。それを $\{-1,a,a'\}$ であるとしよう。($a=(1+\sqrt{-3})/2$, $a'=(1-\sqrt{-3})/2$.) 原点を始点とする $\mathrm{C}-\{-1,a,a'\}$ の基本群の生成元として、それぞれを反時計回りに一回まわってくるループを考え、それらを γ_{-1}, γ_a,$\gamma_{a'}$ としよう。$y=0$ の時には、M' への逆像 $\pi^{-1}(0)$ は $(0,-1),(0,a),(0,a')$ の3箇所だから、これをそれぞれ x_0,x_1,x_2, と名前をつけよう。y を $\mathrm{C}-\{-1,a,a'\}$ の連続なループに沿って動かす時、$\pi^{-1}(y)$ も連続的に動くことに注意して、1周して戻ってきた時には $\pi^{-1}(y)$ は再び x_0,x_1,x_2 のどれかになっているから、ここに置換が生じていることになる。この写像 $\rho:\pi_1(\mathrm{C}-\{-1,a,a'\})\to S_3(x_0,x_1,x_2)$ を branch covering のモノドロミーと呼ぶ。この場合は、$\rho(\gamma_{-1})=\rho(\gamma_a)=\rho(\gamma_{a'})=(012)$ である。(ただし、(012)：$x_0\mapsto x_1,x_1\mapsto x_2,x_2\mapsto x_0$ である。）今の例の場合には、x_0,x_1,x_2 を頂点とし、γ_{-1},γ_a,$\gamma_{a'}$ の持ち上げのすべてを辺とするような M の胞体分割が得られ

ていることになる。従ってこれより、M の 1 次元ホモロジー群の元はすべて $\gamma_{-1},\gamma_a,\gamma_{a'}$ の持ち上げの組合せによって表現されることになり、実際に線形代数の計算によってそのシンプレクティックな生成元を決定することが出来る。（特に、M の基本群の生成元を決定することが出来る。）

代数曲線は $\mathbf{P}^2$ に埋め込まれていると見て無限遠点上のファイバーを決めていたが、それとは別に、単に何枚かの disk を張ることによって compactification する方法もある。オプションでその場合でのホモロジーも計算出来るようになっている。

0.3 続くバージョンへの展望

0.3.1 予定している minor-change

根の動きを、ブレード表示する。ここで注意すべきことは、根のいる複素平面上から 1 点を抜き去り、その上でのブレードを考える必要があるということである。たとえば、重根が起こっているような場合、長さ 2 の置換が起こっているが、その 2 つの根の動きを、C − { 重根 } でのブレードと思えば、その branch point がどのようなノットの cone になっているかを調べることが出来る。その計算と表示。

0.3.2 その後の予定

次に予想されるバージョン 2.0 においては、3 変数多項式の形で表現されている代数曲面を考え、その変数の一つをパラメータと見て代数曲線の族を考える。一般の場合には、general fiber はリーマン面 Σ となり、discrete に singular fiber が現れる。すべてのファイバーには、先の f が定義できるのであるから、パラメータを動かしながら f の branch points の動きを観察するプログラムを作成し、それから $\mathrm{Aut}\pi_1(\Sigma)$ の元を（厳密にいうと $\mathrm{Out}\pi_1(\Sigma)$ の元を）計算することが出来る。（それを Dehn twist の元の合成として表現することも出来る。）それにより、singular fiber の degeneration の具合を見ることが出来る。

0.3.3 付記

動作環境：
Macintosh, 漢字 Talk6 以上、空き RAM 容量 2MB。

請求先：
〒 214-71 川崎市多摩区東三田 明治大学理工学部数学教室 阿原一志
ahara@math.meiji.ac.jp

1 'Mr. Monomie' の使い方

1.1 $x^6+y^6+z^6$ を解く

モノミー君を用いて、次の問題を解いてみよう。

1.1.1 問

代数曲線 $M=\{[x:y:z]\in \mathbf{P}^2(\mathrm{C}):x^6+y^6+z^6=0\}$ が滑らかであることを確かめ、そのジーナスを計算し、基本群の標準的な生成元を求めよ。

1.1.2 答

（い）メニューバーから [Curves] を選び、プルダウンして [Input] を選択する。

（ろ）$M=\{[x:y:z]\in \mathbf{P}^2(\mathrm{C}):x^6+y^6+z^6=0\}$ で、$z=1$ の場合を考え、$\tilde{M}=\{(x,y)\in \mathbf{C}^2:x^6+y^6+1=0\}$ を考えることにする。そして、[Name of Curve] の脇の大きな枠に $\boxed{\texttt{x\^{}6+y\^{}6+1}}$ と記入する。

（は）[Main indeterminant] を $\boxed{\mathrm{x}}$ に、[Sub indeterminant] を $\boxed{\mathrm{y}}$ に変更する。

（に）[fixed] をクリックして、マークを消す。（こうしないと、多項式の変更を認めてもらえないため。）

（ほ）[Branch points] をクリックする。確認を求められるので、クリックしてあげる。しばらくして、カタツムリのダイアログが [click me] と表示するので、クリックする。ここでは

$$f:\tilde{M}\to \mathbf{C}:(x,y)\mapsto y$$

というマップの branch locus をすべて表示してくれる。必要に応じて [up]、[down] を用いてみると良い。この場合は、すべての branch locus に (smooth) のメッセージがついているので、M は多様体であることがわかる。

（へ）[Data] をクリックすると、オイラー数とジーナスが表示される。この場合は、ジーナス 10 のリーマン面であることがわかる。

（と）[Quit]、[Quit] で最初のメニューへと戻る。（ここで、[File][Save as] でデータを記録することもできる。）

（ち）メニューバーから [Curves][monodromy] を選択する。

（り）まず、$\pi_1(\mathrm{C}-\{\text{branch points}\})$ の基点と生成元とを自動的に生成してくれる。

（ぬ）monodromy を求めてくれる。[All of] をクリックする。しばらくして、カタツムリのダイアログに [Click me] と出てくるのでクリックする。

（る）各 branch points でのモノドロミーを一覧で見ることができる。

（を）さらに [Data] をクリックすると、ジーナスなどの情報の他に、normal crossing, higher cusp, higher crossing の個数が表示され、M が滑らかならば、基本群の標準的な基底（シンプレクティック基底）が表示される。ここでは、$l_1=BA\check{}$ と表示されているが、これは、$\pi_1(\mathrm{C}-\{\text{branch locus}\})$ の元 BA^{-1} を $f^{-1}(*)$ の 0 番を始点として lifting したループを意味する。ジーナス 10 であれば、基本群の基底は 10 組現れる。

（わ）[Quit] で戻り、[Loop] を選択すると何かループを入力するように促される。グラフを見て、（この場合は A から F）A から F までの文字からなるワードを書き込む。ただし、逆元も許し、その時には˘をつける。例えば、$AB^{-1}C$ という元について調べてみたければ、$\boxed{\mathrm{AB\check{}C}}$ と入力すればよろしい。大文字と小文字は区別されるので注意すること。（branch points が 32 個を越えれば、小文字も現れる。）

ループの基点 $*$ の逆像 $f^{-1}(*)$ には、自動的に番号が付けられており、今入力したループに沿って、y を動かした時の $f^{-1}(*)$ の入れ替わりを置換群の元で表示する。loop を入力しないで [Ok] をクリックすると警告がでる。

$f^{-1}(*)$ のどの元にどの番号がついているかは、メニューバーに戻って [Polynomial][Input] を選ぶと見ることが出来る。（た）の項を見ること。

（か） [Quit]、[Quit]、[Cancle] で最初のメニュー画面に戻る。

（よ）ここでデータをセーブした方が良い。（[File][Save as] でセーブできる。）今のバージョンでは、セーブしたファイルが「モノミー君」と同じフォルダにないと呼び出すことが出来ないので注意すること。

（た）メニューバーから [Polynomial][Input] を選ぶ。ここでは任意の代数方程式の根を求めることが出来るようになっている。特に、代数曲線の monodromy を求めた後であれば、$\pi_1(\mathrm{C}-\{\text{branch points}\})$ の基点における $f^{-1}(*)$ の番号のつき方を見ることが出来る。

1.2 $x^3=(t-1)^2(t+1)^2$ を解く

1.2.1 問

代数曲線 $M=\{[x:t:u]\in \mathbf{P}^2(\mathrm{C}) \mid x^3u=(t-u)^2(t+u)^2\}$ の位相的データを調べよ。

1.2.2 答

（い）メニューバーから [Curves][Input] を選択する。

（ろ） [Name of Curve] の脇の大きな枠に $\boxed{\texttt{x\^{}3-t\^{}4+2*t\^{}2-1}}$ と記入する。

（は） [Main indeterminant] を $\boxed{\mathrm{x}}$ に、[Sub indeterminant] を $\boxed{\mathrm{t}}$ に変更する。

（に） [fixed] をクリックして、マークを消す。

（ほ） [Monodromy] をクリックする。今度はモノドロミーを一本道で求めてくれる。branch points とそのモノドロミーが表示される。

さらに [Data] をクリックすると、位相的情報が現れる。求める M はトーラスと同相であることが確かめられる。

1.3 注意

1.3.1 メモリーについて

モノミー君は、メモリーを消費して動いているので、次数の高い計算をする時には、メモリー残量を注意しながらでないと、計算を終了させることができなくなる可能性がある。通常であれば、メモリーが不足すれば、警告が出て中断を促される。

中断すればデータは消えてしまうことに注意。続行しても、最悪の場合にはそのまま飛んでしまって帰ってこない場合もあり得る。その時には電源を切る必要がある。

また、メニューバーから [File][Memory Check] を選ぶと、残りメモリーを監視することができる。

1.3.2 エラーメッセージについて

プログラムが動作中に何かの都合で続行不可能になり、エラーメッセージを出す場合がある。

以下のメッセージがでた場合には、ユーザーの工夫によって回避することが出来る。（それ以外は、プログラムバグなので、作者まで連絡をいただければ幸いです。）

Blowing Up happens in Newton method:

方程式が複雑過ぎて、ニュートン法による根の追跡に失敗した。
No such element in pi1 of base space:
表示されているアルファベットの組合せを入力し直す。

2　操作マニュアル

この章では、モノミー君を操作するためのマニュアルを掲載する。
メニューバーは以下の五つよりなる。

File	グラフィックウィンドウの開閉、記録と呼び出し、メモリ管理。
Display	グラフィックウィンドウのズームイン、ズームアップ。
Polynomial	一変数多項式の入力と根。
Curve	二変数多項式の入力と、代数曲面のデータ計算。
Help	（日本語による）ヘルプ。

2.1　File

2.1.1　Show,Hide

branch points などの複素平面上の絵（以下、グラフィックウィンドーと呼ぶことにする）が表示された時に、そのウィンドーを消したり出したりする命令。

2.1.2　Open,Save as

計算したデータをファイルから呼び出したり、ファイルへ書き込んだりする命令。

2.1.3　Memory Check,Check Off

メモリの残量を示すウィンドーを表示したり消したりする命令。

2.2　Display

2.2.1　Zoom Up,Zoom Down

グラフィックウィンドーの表示縮尺を伸ばしたり縮めたりする命令。

2.2.2　Go Right,Go Left,Go Up,Go Down

グラフィックウィンドーの絵を平行移動させる命令。

2.2.3　Default

グラフィックウィンドーの表示サイズや位置を最初に戻す命令。

2.3　Polynomial

2.3.1　Input

多項式を入力し、その根を求める。[Curve] メニューでモノドロミーを求めた後であれば、基点の逆像の具体的な値を見ることが出来る。

2.4　Curve

2.4.1　Input

２変数多項式を代入し、それによって与えられる branch points や monodromy を求める。[Option] ボタンを押すと、無限遠点上での compactification の仕方を選ぶことが出来る。

2.4.2 branch points,pi1 loops,monodromy

[Curve][Input] コマンドのボタンと機能は同じ。

2.5 Help

2.5.1 Help

日本語によるヘルプ（簡単な解説）。

3 クラスと関数

モノミー君はC++言語を用いて記述されている。その中で使われているクラスと付随する関数について解説する。実際に、バージョン1.4iの段階では次のクラスがある。

`class complex;`	複素数を取り扱うクラス。
`class polynomial;`	一変数多項式を取り扱うクラス。
`class curves;`	二変数多項式を取り扱うクラス。
`class permutation;`	置換群の元を取り扱うクラス。
`class path;`	複素平面上の折れ線を扱うクラス。
`class matrix;`	（整係数）行列を取り扱うクラス。
`class graph;`	有向グラフを取り扱うクラス。
`class word;`	自由群の元を取り扱うクラス。

3.1 class complex

複素数のクラス class complex をもつ。一般には複素数を取り扱うライブラリはコンパイラのパッケージに含まれているのが普通であるが、等号成立判定の問題から、ここでは新しくライブラリを作り代える方が適当である。

3.1.1 演算子

```
+,-,*,/,+=,-=,*=,/=
// 加減乗除。

==,!=
// 一致、不一致。
```

二つの複素数 c,d があったときに、「モノミー君」がいつ c と d とを等しいとみなすかは非常に重要な問題である。「モノミー君」では、入力される式の係数が極端に大きくなったり小さくなったりしないことを前提に、$|c$ の実部 $-d$ の実部$| < \varepsilon$ かつ $|c$ の虚部 $-d$ の虚部$| < \varepsilon$ （ただし $\varepsilon = 10^{-7}$）をみたせば、「等しい」ということにしている。

3.1.2 関数

```
double real();
// 実部。

double imag();
// 虚部。
```

3.2 class polynomial

複素数係数の一変数の多項式を取り扱うクラス class polynomial をもつ。

3.2.1 演算子

```
+,-,*,/,+=,-=,*=,/=
// 多項式の加減乗除。

%,%=
```

// 多項式の剰余。

|
// 多項式が割り切れるかどうか。

==,!=
// 多項式の一致、不一致。

3.2.2 関数

char* polynomial::printp(char* z);
// 出力。（この出力に必要なバッファをzに確保しておくこともできる。そのために次の関数が用意されている。zを省略すれば、関数中で自動的にメモリを確保する。）

int polynomial::sizeof_printp();
// printp() に必要なバッファのバイト数を返す。

polynomial gcd(polynomial,polynomial);
// 最大公約因子。これはユークリッドの互除法そのままで基本的には良い。4.1を参照のこと。

polynomial lcm(polynomial,polynomial);
// 最小公倍因子を返す。

polynomial polynomial::derivative();
// 微分した式を返す。

int polynomial::Newton(complex *seed);
// *seedの値を初期値にしてニュートン近似をおこない、収束すればその値を再び*seedに収め、1を返す。失敗すれば0を返す。

complex polynomial::subst(complex);
// 代入。ここでは、Hornerの方法を用いて計算している。

polynomial atopoly(char *s,char c);
// 文字列sを文字cに関する複素数係数の多項式として解釈し、その値を返す。（従ってとくに、c='i'は許されない。）ただし、巾は'^'で表記されているとする。乗算は'*'が通常であるが、空白があれば乗算と解釈する。

polynomial doublesol(polynomial y);
// 方程式y= 0の重根が満たす方程式を返す。（重根がなければ定数項のみを返す。）具体的には、関数yが与えられた時に、$GCD(y,y')$を返す。二根が非常に近い場所にあるような方程式の場合、この関数で重根判定が出来るとは限らないので、十分な注意が必要である。

int number_of_solutions(polynomial y);
// 方程式の（重複を一つに数えた）根の個数を返す。これは単純に元の次数からdoublesol()の次数を引いたもの。従って、前項と同様の問題は存在する。

int multiplicity(polynomial y , complex c);

// 複素数c が方程式y= 0 の何重根であるかを返す。根でなければ 0 を返す。実際には、$y(c)$, $y'(c)$, $y''(c)$, $\cdots$ という列を考えて、どこまでが 0 （またはその近傍内）であるかを調べて出力する。従って、非常に近い二根がある場合に、それを判定できない場合はあり得る。

```
int reduction(polynomial& x, polynomial& y);
```
// 二つの多項式に対して、その最大公約数で割った式をx, y に改めて代入する。

注意： 重根に関わる関数については、代数方程式を連立法によって解く関数を経由して、より精密に現象を把握することができる。4.2 を参照のこと。

3.3 class curves

複素数係数の二変数の多項式を取り扱うクラス class curves をもつ。

3.3.1 演算子

```
+,-,*,/,+=,-=,*=,/=
```
// 多項式の加減乗除。

```
|
```
// 多項式が割り切れるかどうか。

```
==,!=
```
// 多項式の一致、不一致。

3.3.2 関数

```
char* curves::printp( char* z)
```
// 多項式の表示。z に表示するバッファを前もって確保しておくこともできるのは polynomial の場合と同じ。sizeof_printp() 関数も存在する。

```
polynomial reduction(curves*);
```
// 第二変数のみからなる因子で割り、その因子を返す。この関数には、polynomial の gcd() 関数を用いているので、精密な計算は望めない。

```
curves gcd(const curves&, const curves&);
```
// 基本的には、互除法を用いる。ただし、ここでは polynomial の gcd() 関数を用いているので、精密な計算は望めない。4.1 を参照のこと。

```
int reduction(curves* ,curves*);
```
// 共通因子を求め、それで両者を割る。

```
polynomial renritsu2(curves&,curves&);
```
// 2つの二変数多項式から第1変数を互除法によって消去し、（第2変数からなる）多項式として返す。

```
curves curves::derivative(int a);
```
// $a=0$: 第一変数による偏微分。$a=1$: 第二変数による偏微分。

```
polynomial curves::subst(complex&);
```

```
complex curves::subst(complex& t,complex& x);
```
// 代入。Horner 法を用いて計算している。

```
curves curves::atocurves(char* z,char indet0,char indet1);
```
// 文字列 z を文字 indet0,indet1 に関する多項式と解釈し、その式を返す。フォーマットは polynomial の場合と同じ。

3.4 class permutation

置換群の元を取り扱うクラス class permutation を持つ。

3.4.1 演算子

```
permutation a = "(0)(1 3 4)(2)";
```
という代入を許すとする。(ただし、置換の要素の数を特定するために、(0) や(2) は省略できないものとする。)

```
*,*=
```
// 置換の合成。(サイズが異なる場合は大きい方に合わせることにする。)

```
==,!=
```
// 置換の一致、不一致。(同上)

```
-
```
// 逆置換。

3.4.2 関数

```
char* permutation::cycle();
```
// "(0)(1 3 4)(2)" 形式の出力。

```
permutation restrict(int i);
```
// 置換の要素の個数を i に減らす。そのためには、i 以上の要素は置換を起こしていないことが必要である。

```
permutation expand(int i);
```
// 置換の要素の個数を i に増やす。

```
int permutation::goesto(int i);
```
// 置換において、i の行き先を返す。

```
int permutation::comesfrom(int);
```
// 置換において、i に写るような要素を返す。

3.5 class path

複素数の列を取り扱うクラス。class path をもち、`typedef path roots;` を宣言してある。

3.5.1 演算子

`*,*=`

// 複素数の列を複素平面上の折れ線と考えて、(以下、path と表現する。) もし一つの終点ともう一方の始点とが一致していた場合に、それらをつなげた折れ線を返す。

`+,+=`

// 複素数の列 (以下、roots と表現する。) の集合としての和を作る。

`-`

// 複素数の列を path と考えて、逆方向にたどった折れ線を返す。

3.5.2 関数

`complex path::startpoint(); complex path::endpoint();`

// path の始点と終点。

`int path::isclosed();`

// path が閉じていれば 1、さもなくば 0 を返す。

`int path::nullhomotopic(roots r);`

// 複素平面上から r の各点を取り除いた空間を考え、そこに path が含まれ、かつヌルホモトピックであれば 1 を返す。

`path roots::pi1_cycle(int i, complex org,int length);`

// 複素平面上から各点を取り除いた空間を考え、*org* を始点とし、長さ *length* で、i 番目の点を反時計回りに回ってくる *path* を返す。

`roots roots::sort(complex org);`

// *org* を視点として、角度の小さい順に、角度が同じならば遠い順に、ソーティングした roots を返す。

`complex get_origine(roots r);`

// r のどの点とも一致せず、各点の近くにある点を選びだし、返す。

`char* roots::printp(char* z);`

// 各点を羅列して z に出力する。

3.6 class matrix

(整係数) 行列を取り扱うクラス class matrix をもつ。

3.6.1 演算子

`+,+=,-,-=,*,*=`

// 行列の和、差、かけ算 (サイズがあっている場合にのみ計算する。)

`%,%=`

// 二つの行列の列数が等しい時に、それら二つを縦に並べた行列を返す。

`|,|=`

// 二つの行列の行数が等しい時に、それら二つを横に並べた行列を返す。

3.6.2 関数

`int matrix::Row_Scalar(int row,_COEF scalar);`
// *row* 行目を *scalar* 倍する。`int matrix::Col_Scalar(int col,_COEF scalar);` も同様。必ず 1 を返す。

`int matrix::Row_Exchange(int row1,int row2);`
// *row*1 行と *row*2 行を入れ換える。`int matrix::Col_Exchange(int col1,int col2);` も同様。必ず 1 を返す。

`int matrix::Row_Subst(int src,int tgt,int scalar);`
// *src* 行を *scalar* 倍して *tgt* 行に加える。`int matrix::Col_Subst(int src,int tgt,int scala` も同様。必ず 1 を返す。

`int matrix::Row_Col_Exchange();`
行列を転置する。必ず 1 を返す。

`matrix matrix::Trimming(int row,int col,int height,int width);`
// *row* 行 *col* 列を左上とする、$height \times width$ の行列を抜きだして返す。

`int matrix::Det(); int matrix::Trace();`
// 行列式とトレース。

`matrix matrix::Inverse();`
// 行列式が ±1 であった時の逆行列。

`char* printp(char* z);`
// 行列を出力表示して z に格納する。

3.7 class graph, class edge

頂点と辺からなる色づき有向グラフを取り扱うクラス class graph と、その構成する辺を取り扱うクラス class edge とを持つ。

3.7.1 class edge

class edge は以下の要素を持つクラスである。

- `int _start;`
  // 始点の番号
- `int _end;`
  // 終点の番号
- `int _number;`
  // 辺の通し番号
- `int _mode;`
  // 辺の色。
- `int _name;`
  // 辺の名前（通し番号とは別に番号をつけることが出来る）。

3.7.2 class graph

class graph は以下の要素を持つクラスである。

```
• int *vertex_rank;
```
// 0番のついた頂点と各頂点との最短距離が格納される。

```
• int max_vertex;
```
// 頂点の個数。

```
• edge *edges;
```
// 辺の列。

```
• int max_edges;
```
// 辺の本数。

3.8 class word

自由生成の群の元を表現するクラス class word を持つ。$a_1, a_2, a_3, \cdots$ で生成される自由群の元を次の例の要領でデータ化する。例えば、$a_1 a_3 a_5^{-1} a_2^{-3} a_{10}$ は

$$1, 3, -5, -2, -2, -2, 10$$

という数列で表現される。$a_i a_i^{-1}$ のようにキャンセルされる部分を持てば、自動的にキャンセルする。

3.8.1 演算子

```
*,*=
```
// 群のかけ算。

```
-
```
// 逆元。

3.8.2 関数

```
char* printp(char* a);
```
// 出力表示して a に格納する。

```
permutation eval(word w,permutation* p);
```
// permutation の列 $p_1, \cdots, p_\ell$ （長さ ℓ）に対して、$p_1, \cdots, p_\ell$ で生成される word w が与えられた時に、それを実際に計算して permutation を返す。

4 アルゴリズム

この章では、プログラム中で使われているいくつかの関数について、そのアルゴリズムを解説する。実際には、次の関数について論ずる。ただし、ここに掲載した関数そのものが「モノミー君」にあるわけではなく、デバッグやエラー検出などのプログラム上の都合による引数が加わったりしていることを付言しておく。

```
polynomial gcd(polynomial f, polynomial g);
```
一変数多項式の最大公約数を求める。

```
curves gcd(curves f, curves g);
```
二変数多項式の最大公約数を求める。

```
roots simultaneous_eq(polynomial f, roots r);
```
連立法により、代数方程式の根を求める。

```
roots solutions(polynomial f);
```
代数方程式の根を求める。

```
polynomial PolyMulti(polynomial f);
```
多項式の根の多重度を調べる。

```
permutation solution_trace(curves* f, path* p ,roots* r);
```
パラメータつきの方程式 f について、パラメータを path p に沿って動かした時の、根 r の動きを追跡して、permutation で返す。

```
roots renritsu(curves,curves);
```
2つの二変数多項式の共通根を求める。

`roots branchpoints(curves f);` 二変数多項式 f から決まる代数曲線に対して、projection のブランチポイントを計算する。

```
graph mono2graph(permutation *p,int length);
```
モノドロミーを permutation の列で表現し、それを入力した場合の胞体分割の 1-skelton を与える。

```
matrix mono2d1(permutation *p, int length);
```
モノドロミーを permutation の列で表現し、それから決まる胞体分割の 1 次元境界作用素 $\partial_1 : C_1 \to C_0$ を行列表示して出力する。

```
matrix mono2d2(permutation *p, int length);
```
モノドロミーを permutation の列で表現し、それから決まる胞体分割の 2 次元境界作用素 $\partial_2 : C_2 \to C_1$ を行列表示して出力する。

```
matrix Homology(matrix d1,matrix d2);
```
2つの行列 D_1, D_2（それぞれ d1,d2 で入力する）に対し、$D_1D_2 = O$ であったときに、kernel(D_1)/image(D_2) の（整係数）基底を計算し、それらを並べて行列の形で出力する。

```
matrix mono2intersection(matrix Homology, permutation *p,int length);
```
1次元ホモロジーの生成元とモノドロミーから、intersection matrix を作成し、出力する。

```
matrix Normalize_SP(matrix is);
```
intersection matrix を Symplectic 標準形に変形する座標変換を求める。

4.1 gcd

```
polynomial gcd(polynomial, polynomial);
```

多項式の最大公約因子を求める。一変数多項式 (class polynomial) の場合には、ユークリッドの互除法そのままで良い。すなわち以下のアルゴリズムで解決される。

(1) If f or g is a constant, return 1. Otherwise go to (2).

(2) If f divides g (or g divides f), return f (or g). Otherwise go to (3).

(3) If the degree of f is greater than that of g, replace f by $f\%g$ and go to (1). Otherwise replace g by $g\%f$ and go to (1).

```
curves gcd(curves, curves);
```

二変数多項式 (class curves) の場合にも、ほぼ同様のアルゴリズムで解決される。

(1) If f has a divisor of the second indeterminant, let it be p_f (, otherwise $p_f = 1$). Similarly, if g has one, let it be p_g.

(2) Replace f by f/p_f and replace g by g/p_g and let q be the g.c.d. of p_f and p_g.

(3) If f or g is a polynomial of the second indeterminant, return q. Otherwise go to (4).

(4) If f divides g (or g divides f), return fq (or gq). Otherwise go to (5).

(5) Let a_f be the coefficient of top-degree (w.r.t. the first indeterminant) of f, let a_g be similar, and let b be the g.c.d. of a_f and a_g. If the degree (w.r.t. the first indeterminant) of f is greater than that of g, then replace f by $(fa_g - ga_f)/b$. Otherwise replace g by $(ga_f - fa_g)/b$. Go to (3).

たとえ、f と g に共通の（第２変数だけの）因子がなかったとしても、$ga_f - fa_g$ が第２変数だけの因子を持つことはあり得るので、(1)、(2) のステップは必要である。

4.2 連立法による代数方程式の解

```
roots  simultaneous_eq(polynomial& p, roots* r);
```

多項式 p について、r を初期値とする連立法によって近似解を求め、その結果を返す。

f を n 次一変数代数方程式であるとする。その場合、複素平面上の異なる n 点 $z_1^{(0)}, \cdots, z_n^{(0)}$ に対して、

$$z_i^{(j+1)} = z_i^{(j)} - \frac{f(z_i^{(j)})}{c\Pi_{k\neq i}(z_k^{(j)} - z_i^{(j)})}$$

という漸化式で定義される点列の列を考える。ただし、c は f の n 次の係数であるとする。この時、$z_1^{(j)}, \cdots, z_n^{(j)}$ は（もし $f = 0$ が重根を持たなければ）異なる根にそれぞれ収束することが知られている。

この解法についての数学的事実をあげておこう。より詳しくは、伊理 [I] を見ると良い。

Theorem

(1) Let f be an algebraical polynomial of degree n. If f has no double solutions and $z_1^{(j)}, \cdots, z_n^{(j)}$ converge to points $\alpha_1, \cdots, \alpha_n$, then $\alpha_1, \cdots, \alpha_n$ are approximations of solutions of $f = 0$ and an order of the convergence is 2.

(2) [Theorem of Smith] For any $j = 1, 2, \cdots$, all solutions of $f = 0$ is containing in the union $\bigcup U_i$ of n-disks U_i given by

$$U_i = \{z; |z - z_i^{(j)}| \leq (n-1)|z_i^{(j)} - z_i^{(j-1)}|\}.$$

Precisely, for each connected components of $\bigcup U_i$, there are as many solutions as the number of disks consisting the component.

(3) Soppose $f=0$ has also a simple solution p_1 and an m-ple multi-solution p_2. If $z_1^{(j)}$ and $z_2^{(j)}$ converge on p_1, p_2 repectively, then

$$\log|z_1^{(j)}-z_1^{(j-1)}| \sim m\log|z_2^{(j)}-z_2^{(j-1)}|$$

for large j.

(4) Suppose $f=0$ has an m-ple multi-solution p and $z_1^{(j)},\cdots,z_m^{(j)}$ is enoughly near p then $(z_1^{(j)}+\cdots+z_m^{(j)})/m$ approaches to p with order 2.

実際に、この連立法を用いて計算を行なってみると、丸め誤差のために、$\log|z_i^{(j)}-z_i^{(j-1)}|$ は大きな j に対していくらでも小さくなるわけではない。ある j_0 があって、$j>j_0$ では $\log|z_i^{(j)}-z_i^{(j-1)}|$ はある一定の区間内を振動するようになる。それを定常状態であるというふうに呼ぶことにしよう。（その区間は一般に、$z_i^{(j)}$ が近付こうとしている根が何重根であるかに依存する。）その段階で、$z_i^{(j)}$ たちが、何重根に近寄ろうとしているかは Smith の定理よりほとんど分かるといって良い。（たとえば、１０次方程式で有効数字 16 桁の時に 3 重根までくらいであれば。）それを判定した上で、たとえば、$z_1^{(j)},\cdots,z_m^{(j)}$ が m 重根に近付くのであれば、

$$\log|(z_1^{(j)}+\cdots+z_m^{(j)})/m-(z_1^{(j-1)}+\cdots+z_m^{(j-1)})/m|$$

を観察して、それによって適当なところで計算を打ち切るのが賢明であろう。

```
roots solutions1(polynomial* f);
```

多項式 f に対して、$f=0$ の根を返す。（ただし、重根は一つに数える。）連立法のアルゴリズムができあがっているので、ここでは初期データさえ与えればよい。連立法は大域的に収束することが知られているが、次の初期データ（アバースの初期値と呼ばれる）がよいことが経験的に知られている。

$f(z)=z^n+a_{n-1}z^{n-1}+\cdots+a_1z+a_0$ に対して、中心 β を

$$\beta=-a_{n-1}/n$$

とし、半径 r を方程式

$$z^n-|a_{n-1}|z^{n-1}-\cdots-|a_1|z-|a_0|$$

の唯一の正の根によって決める。（これらの計算は大変ではない。）経験的によいとされる初期値は

$$z_i^{(0)}=\beta+r\exp\sqrt{-1}(\frac{2\pi}{n}+\frac{3}{2n})$$

である。この計算をした後に、連立法を呼び出して、根を求める。

（計算例）$f(z)=(z-1)(z+1)^2(z-i)^3(z+i)^4$

$1.000000000000000+0.000000000000000i$
$-0.999999999988788+0.000000000004970i$
$-0.000000014114107+1.000000001277793i$
$-0.000000262157283-1.000000239232556i$

```
polynomial PolyMulti(polynomial f);
```

方程式 $f=0$ の重根の起こり具合を多項式の形でデータ化し出力する。実際に、単根が a_1 個、2 重根が a_2 個、3 重根が a_3 個、……であった場合に、$a_1z+a_2z^2+a_3z^3+\cdots$ というようにデータ化して出力する。これは、代数方程式の根を求めるアルゴリズムに含まれているので、重ねて説明はしない。

4.3 $\pi_1(\mathbf{C} - \{points\})$ の生成元の作成

`path roots::pi1_cycle(int number,complex origin);`

roots に対して、origin を始点として、number 番の点だけを反時計回りに回ってくる閉折れ線を出力する。

まず、次の方法によって simple loop なるものを作成する。roots を $\tilde{z}_1, \tilde{z}_2, \cdots, \tilde{z}_n$ とし、簡単のため number$=1$ とする。$i = 1, 2, \cdots, n$ に対して $z_i = \tilde{z}_i - origin$ とする。実数 s と r を次で定義する。

$$s = \min\{\arg(z_1/z_i) \mid \arg(z_1/z_i) \neq 0\}, \qquad r = 1.1 \times |z_1|$$

そこで、中心 origin、半径 r、偏角の範囲が $[\arg(z_1) - s/2, \arg(z_1) + s/2]$ であるような扇型を考え、それを反時計回りに一周するようなループを simple loop と呼ぶことにする。

`pi1_cycle` は次のようにして決まる。すなわち、もし、z_1 と origin とを内分するような位置に roots の点がなければ、z_1 の simple loop を出力する。さもなくば、z_1 に最も近い内分点を調べ、それが z_i であったとときに、(z_1 の simple loop) $*$ ($-$(z_i の simple loop)) を出力する。

注意 1：もし roots の点を (1)origin から見た偏角の小さい順に (2) 偏角が等しければ origin から遠い順にソーティングしておけば、$\tilde{z}_i$ の`pi1_cycle`γ_i に対して、

$$\gamma_1 * \gamma_2 * \cdots * \gamma_n$$

(ループの合成) は、無限遠点の回りを時計回りに 1 周するループになっていることがわかる。このようにソーティングしておくことは、代数曲面の胞体分割を考える時に重要である。

4.4 根の追跡とモノドロミー

`permutation solution_trace(curves f, path p ,roots r);`

パラメータつきの方程式 $f(z, y) = 0$ について、パラメータ y を path p に沿って動かした時の、根 r の動きを追跡して、permutation で返す。

path p を $y_0, y_1, \cdots, y_R$ であるとする。このとき、$y_0 = y_R$ かつ roots r は $f(z, y_0) = 0$ の根であることが要求される。(すべてでなくとも良い。) path の各辺について、以下のようなアルゴリズムで根の追跡をする。

1. 辺 $y_i y_{i+1}$ を適当に等分に内分する。(Mr. Monomie では 20 等分しているが、これは好みの問題。) それを $y_i = y^0, y^1, y^2, \cdots, y^S = y_{i+1}$ としよう。

2. $f(z, y^j) = 0$ の根が分かっているとして、それを $z_1, \cdots, z_n$ としよう。この値を初期値として、方程式 $f(z, y^{j+1}) = 0$ の根を連立法によって求める。そうして求めたものを $\bar{z}_1, \cdots, \bar{z}_n$ としよう。(順番も込めて。)

3. パラメータ y の微細な動きに従って、z_i が $\bar{z}_i$ へと移動したことを確認するために、次の 2 点をチェックする。

   (a) $\bar{z}_i$ は $z_1, \cdots, z_n$ の中で z_i にもっとも近い。

   (b) $\bar{z}_i - z_i$ は $(y^{j+1} - y^j) f_y(z_i, y^j) / f_z(z_i, y^j)$ と「近い」偏角を持つ。

   もし満たされなければ、辺の内分をもっと細かくしてやり直す。

4. このアルゴリズムにより、path p に沿った $f(z, y) = 0$ の根の挙動を調べる。$y = y_R$ (終点) にたどり着いたら、roots r と比較して、それを置換群の元で表現して出力する。

注意1：　上の条件(a)は、パラメータをなめらかに動かしたときに、根が連続に動くことを利用してチェックしている。同様に、条件(b)は根がC^1級であることを利用してチェックしている。なお、パラメータyがpath pの辺上にあるときには$f(z,y)=0$は重根を起こさないことを要求されているので、条件(b)の分母は0にはならない。

注意2：　関数でroots rを与えることは必要である。基点上の根自体は、fとpとで求めることが出来るが、要はそれらにどのように番号がついているのかを指定しなければならず、それをrで与えると考えれば良い。

注意3：　上のアルゴリズムにより、path pの辺の長さは全く問われない。pがホモトピカルに求めたいループと等しくさえあれば、あとは、必要に応じてプログラム内で辺を細分しているからである。

4.5　2変数高次連立方程式を解く

`roots renritsu(curves f,curves g);`

2つの二変数多項式を解いて、解のうち第2変数の方だけをとり出して並べたものを出力する。

別のいい方をすれば、二変数の連立方程式から一方の変数を消去し、それについて解を求め、連立方程式の根となっているものだけを取り出して出力する、というアルゴリズムを行なっている。

要領としてはgcdを求める時と同じであるが、次の点に注意する。ただし、第1変数をz、第2変数をyとして話を進める。zを消去してyだけからなる式を求めようというわけである。

1. $f(z,y)$または$g(z,y)$が、yのみからなる因子を持っている場合には、その根を出力に加える。

2. $f(z,y)$が、zのみからなる因子を持っている場合には、それを解いて、$g(z,y)$に代入して、その根を出力に加える。$g(z,y)$が持つ場合にも同様。

3. $f(z,y)$と$g(z,y)$とが、zの混ざった共通因子を持つ場合には、これを無視する。（この場合は、根はすべてのyをとり得る。さらに、このような場合は、`gcd(f,g)`でチェックすることも出来る。

`roots branchpoints(curves f);`

2変数多項式fから決まる代数曲線に対して、projectionのブランチポイントを計算する。これは上の関数から容易に導き出せる。$g=f_z$として、`renritsu(f,g)`を求めれば良い。

4.6　モノドロミーと胞体分割

`graph mono2graph(permutation *p,int length);`

モノドロミーをpermutationの列で表現し、それを入力した場合の胞体分割の1-skeletonを与える。

代数曲線が多様体になっている時に、モノドロミーからホモロジーを求める第一段階となる。多様体から$\mathbf{P}^1$へのbranched coveringを考えよう。$\mathbf{P}^1$上のbranch pointsを$y_1,\cdots y_\ell$としよう。各branch pointsを1回まわるループを$\gamma_1,\cdots,\gamma_\ell$とする。ここで、を$\gamma_1*\cdots*\gamma_\ell$がヌルホモトピックであるようにループを選んでおく。γ_iに沿ったモノドロミーをp_iとする。従って、$p_1,\cdots,p_\ell$は関係式$p_\ell p_{\ell-1}\cdots p_1=id$を満たす。（関数に代入する時にはpは置換の列の先頭の

番地、length は列の長さを表す。）求めるべき 1-skeleton の頂点は、$\pi_1(\mathrm{C}-\{\text{branch points}\})$ の基点 $*$ 上のファイバーであり、その個数は置換の要素の個数で表されている。

すべての 1-cell は各置換の元から求めることが出来る。例えば (0)(123)(45) という置換があれば、頂点の個数は 6 個（$v_0,\cdots,v_5$ としよう。）であって、v_0 から自分へ戻る edge が一つ、v_1 から v_2 へ向かう edge がひとつ、同様に v_2 から v_3 へ、v_3 から v_1 へ、v_4 から v_5 へ、v_5 から v_4 へ向かう edge があることになる。以上のようにして 1-skeleton のデータが出来上がり、それを出力する。

4.7 モノドロミーと境界作用素

```
matrix mono2d1(permutation *p, int length);
```

モノドロミーを permutation の列で表現し、それから決まる胞体分割の 1 次元境界作用素 $\partial_1 : C_1 \to C_0$ を行列表示して出力する。

1-skeleton を求めたのと同じ方法で 1 次元境界作用素は求めることが出来る。ここで注意すべきことは、2 次元境界作用素との関係から、1-cell の生成元は規則にしたがって並んでいることが必要である。

```
matrix mono2d2(permutation *p, int length);
```

モノドロミーを permutation の列で表現し、それから決まる胞体分割の 2 次元境界作用素 $\partial_2 : C_2 \to C_1$ を行列表示して出力する。

上で作った 1-skeleton に 2-cell となる disk を張り付けていけば良い。その張り方は次のようにやる。

（1） branch points の周り。

例えば (0)(123)(45) という置換があれば、v_0 から自分へ戻る edge に 1 枚、v_1 から v_2 へ、v_2 から v_3 へ、v_3 から v_1 へ行く 3 つの edge に 1 枚、v_4 から v_5 へ、v_5 から v_4 へ行く 2 つの edge に 1 枚の計 3 枚の disk を張ることになる。

（2）外側。

多様体から $\mathbf{P}^1$ への branched covering を考えているのであるから、すべての branch point を囲むようなループの外側はやはり disk であって、この持ち上げが 2-cell になる。具体的には、$v_i(i=1,\cdots,n)$ から始めて $p_1,\cdots,p_\ell$ で次々に決まる 1-cell の列を作り、そこに 1 枚の disk を張る。

注意：`mono2d1` と `mono2d2` とでは、1-cell の並び方を共通にしておかなければならない。

4.8 ホモロジーと交差形式

```
matrix Homology(matrix d1,matrix d2);
```

2 つの行列 D_1, D_2（それぞれ d1,d2 で入力する）に対し、$D_1D_2=\mathbf{O}$ であったときに、$\mathrm{kernel}(D_1)/\mathrm{image}(D_2)$ の（整係数）基底を計算し、それらを並べて行列の形で出力する。

これは単純に線形代数の計算なので、特に問題はない。この場合には torsion は起こらないので torsion のあるようなホモロジー群の基底の計算には向かないが、いずれサポートするつもりである。

```
matrix mono2intersection(matrix Homology, permutation *p,int length);
```

1 次元ホモロジーの生成元とモノドロミーから、intersection matrix を作成し、出力する。

1 次元ホモロジーの元は、すべて 1-cell の和で書けているので、2 つの 1 次元ホモロジーの元に対して、その交差数を計算することは出来る。とくに、頂点上での「出入り」の位置関係を見て、それを符合付きに加えればよろしい。

```
matrix Normalize_SP(matrix is);
```

intersection matrix を Symplectic 標準形に変形する座標変換を求める。

intersection matrix は反対称行列であるので、Symplectic 標準形に変形する座標変換を求めるのは単なる線形代数の問題。この場合は整数係数の変換行列が答になることが前もって分かっているのと、基本変形をするサブルーチンは前もって作ってあるので、簡単にできる。

5 応用例

5.1 リーマン面の退化に関する計算例

滑らかな4次元多様体 M から2次元ディスク D への滑らかな写像 f があって、ある一点 $p \in D$ があって、任意の $z \in D - p$ についてはファイバー $f^{-1}(z)$ がリーマン面になっている場合を考えよう。その場合の $f^{-1}(p)$ を特異ファイバーといい、その位相型はモノドロミーと密接な関係にある。

今、実際に、次のような例で考えてみよう。

$$M' = \{(x,y,z) \in C; x^2 = (y^5 - y)(y - z)\}$$

$f : M' \to C : (x,y,z) \mapsto z$ という写像を考えると、$z = \pm1, \pm i, 0$ において特異ファイバーが起こっている。それ以外ではファイバーは 1-punctured な種数3のリーマン面である。各ファイバーについて穴があいているところを1点 compactification することにする。

この場合、各ファイバーからの写像 $\pi : F_z = f^{-1}(z) \to C : (x,y,z) \to y$ の branch points は、（手計算により）$(y^5 - y)(y - z) = 0$ を y について解いた解であることが分かる。

（1）そこで最初に、この $(y^5 - y)(y - z) = 0$ についてモノミー君で解析してみよう。主変数を y、副変数を z として入力してみる。この場合の branch points は ∞ も含めて6箇所。z–space のループの原点は $0.286i$ であることが分かる。（バージョン 1.4i 以降では、カーソルで指し示してクリックすると、その点の座標が表示されることから。）これを z_0 とおこう。[Monodromy] の [One of] モードを選び、A から F までの branch points について、「根の動き」を確認してみよう。当然ながら、$(y^5 - y)(y - z) = 0$ の根は、$0, \pm1, \pm i, z$ の6つなので、z が動けばそれにしたがって、根の一つが動くことになる。

（2）さて、次に、$f^{-1}(z_0)$ を考えてみよう。これは、式の上では、$\{(x,y) \in C; x^2 = (y^5 - y)(y - z_0)\}$ を考えることになる。今度はこの式をモノミー君で調べて見よう。主変数を x、副変数を y でやって見る。ただし、バージョン 1.4i 以上が必要で、[Input] を選び、[Option] ボタンを押して、[compactification mode] を [smooth] にする。これにより、モノミー君は与えられた式の穴があいているところを1点 compactification してすべてを考えてくれる。この時は、[Monodromy] から [All of] を選択して、すべてのモノドロミーを計算させた後、[Data] ボタンを押す。すると、$f^{-1}(z_0)$ はジーナス2のリーマン面で、そのシンプレクティックな基本群の生成元は

$$\begin{aligned} \ell_1 &= ba^{-1}, & m_1 &= cd^{-1}ea^{-1} \\ \ell_2 &= ad^{-1}ea^{-1}, & m_2 &= cd^{-1} \end{aligned}$$

であることが分かる。混乱を避けるため、y-space 上のループたちを $a, \cdots, f$ と書くことにする。（他のとり方もいくらでもあることは分かるが、今はこれで我慢する。）

そして、ここでのループの基点は y–plane 上で約 $y_0 := -0.286 + 0.029i$ であることが分かる。

（3）3番めに $f^{-1}(z_0)$ でのループの基点は $\pi^{-1}(y_0)$ の中の

$$(x,y,z) = (x_0, y_0, z_0) \sim (0.087 + 0.329i, -0.286 + 0.029i, 0.286i)$$

であることが、[Polynomial][Input] により分かる。（ループの基点は0と番号のついた点であることより。）そこで、y を y_0 に止めて、z を（1）でのループに沿って動かした時の、この点の行き先を調べる。そのためには、$x^2 - (y_0^5 - y_0)(y_0 - z)$ について調べれば良い。これはすぐに分かることだが、z についてのループが y_0 を囲んでいれば (x_0, y_0, z_0) は $(\bar{x_0}, y_0, z_0)$ へ、さもなくば自分自身へと戻ることが分かる。この場合は、A から E までのどのループも y_0 を含ん

でいないので、(x_0, y_0, z_0) は自分自身へ移ることが分かる。従って、$f^{-1}(z_0)$ でのループを絶えず y-space 上のループのように表現するが、それらはすべて (x_0, y_0, z_0) を基点とする持ち上げを考えていることとする。

（4）（1）で求めた、z-space 上のループに沿って z を動かした時、y-space 上のループたちがどう動くかを考える。

$z-plane$	
A	$a \mapsto a^{-1}e^{-1}aea, e \mapsto a^{-1}ea$
B	$b \mapsto bebe^{-1}b^{-1}, c \mapsto beb^{-1}e^{-1}cebe^{-1}b^{-1}, d \mapsto beb^{-1}e^{-1}debe^{-1}b^{-1}, e \mapsto beb^{-1}$
C	$c \mapsto cece^{-1}c^{-1}, d \mapsto cec^{-1}e^{-1}dece^{-1}c^{-1}, e \mapsto cec^{-1}$
D	$d \mapsto dede^{-1}d^{-1}, e \mapsto ded^{-1}$
E	$e \mapsto efef^{-1}e^{-1}, f \mapsto efe^{-1}$

（5）すると、（2）の情報と合わせると、例えば、z-plane 上の A というループに沿って動かすと、

$$\begin{aligned}
\ell_1 &= ba^{-1} \mapsto ba^{-1}e^{-1}a^{-1}ea = \ell_1(m_2^{-1}m_1)^2 \\
m_1 &= cd^{-1}ea^{-1} \mapsto cd^{-1}a^{-1}eaa^{-1}e^{-1}a^{-1}ea = m_1 \\
\ell_2 &= ad^{-1}ea^{-1} \mapsto a^{-1}e^{-1}aead^{-1}a^{-1}eaa^{-1}e^{-1}a^{-1}ea = (m_1^{-1}m_2)^2\ell_2 \\
m_2 &= cd^{-1} \mapsto cd^{-1} = m_2
\end{aligned}$$

となり、これによって $\mathrm{Inn}\pi_1(f^{-1}(z_0))$ を与えていることが分かる。

以上のように、退化したリーマン面のモノドロミーを求めることが出来る。この（4）、（5）の計算は面倒であるので、バージョン 2 以降ではこれも自動的に計算したいと考えている。

5.2　特異点の絡み目型に関する計算例

複素数係数の 2 変数多項式 $F(x, y)$ があって、次を満たすとする。

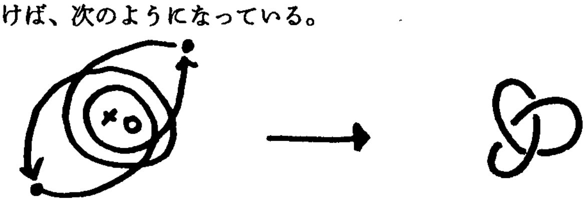
$$\begin{aligned}
F(0,0) = F_x(0,0) = F_y(0,0) = 0 \\
F_{xx}(0,0)F_{yy}(0,0) - (F_{xy}(0,0))^2 = 0
\end{aligned}$$

このような場合、$M = \{(x, y) \in \mathrm{C} \times \mathrm{C} \mid F(x, y) = 0\}$ は $(x, y) = (0, 0)$ で特異点を持つ。これを原点を内側に含むような球面で切ると、そこに絡み目が現れる。これをその特異点の絡み目型ということにする。

モノミー君バージョン 1.4i 以降では、この特異点の絡み目型を実際に計算することができる。ただし、1.4i の段階では、手作業が残るので、これを全自動化するのはこれからの課題である。

（例 1）$x^2 + y^3 = 0$

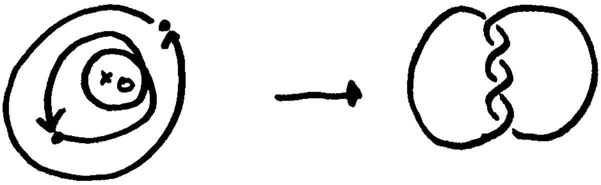
モノミー君に $x^2 + y^3$ の式を入れて、[Monodromy] から [One of] を選ぶ。少なくとも、0 は branch points の中に入っているから、これを選ぶ。今の場合は 1 箇所しかないので、ループとして A と入力する。すると、根の動きが見られるが、これを克明に（！！）記録する。例えば x を主変数にしておけば、次のようになっている。



根は原点を通らないので、これを $\mathbf{C}-\{0\}$ 上の braid と思って、その完備化した絡み目を作ってみると、trefoil であることが分かる。

（例2） $x^2+xy+y^3=0$

計算の仕方は同じ。この場合は2成分の絡み目になる。



参考文献

[A] K. Ahara, On the topology of Fermat type surface of degree 5 and the numerical analysis of algebraic curves, Tokyo J. Math. **16** (1993), 321-340.

[I] M. Iri, Numerical calculations, (in Japanese), (1981), Asakura-shoten.

[IYTO] M. Iri, H. Yamashita, T. Terano and H. Ono, An algebraic equation solver with global convergence property, RIMS Kokyuroku, **339** (1978), 43-69.

[K] K. Kodaira, On compact analytic surfaces, Ann. of Math., **79** (1963), 563-626.

[M] Y. Matsumoto, On the topological structure of the Fermat surface of degree 5, Kodai Math. J., **17** (1994), 560-570.

[MM] Y. Matsumoto and J. M. Montesinos-Amilibia, Pseudo-periodic homeomorphisms and degeneration of Riemann surfaces, Bull. Amer. Math. Soc. **30** (1994), 70-75.

[SM] M. Sugihara and K. Murota, Mathematics in numerical calculations, (in Japanese), (1994), Iwanami-shoten.