# 第1章　SurfとSINGULAR

## 1.1　surf

### 1.1.1　surf概要

surf は二次元平面内の代数曲線 (x と y の実係数多項式の零点), 三次元空間の代数曲面 (x,y,z の実数係数多項式の零点) の描画が行えるプログラムである. 固有の GUI を持ち, 簡単な条件分岐を持つ C 風の処理言語を有する。他の描画ソフト, 例えば,dynagraph の様に曲面をマウスで直接回転させたりする事は出来ないが, 対話的処理で曲面の回転や拡大・縮小等が行える. 尚,surf では与えられた多項式の零点集合 (Affine variety) を描けるが, それ自体で本格的な数式処理を行うものではない.

surf は GTK+(旧版は XView) を用いてその GUI を構築している為, 動作環境は GTK+ (旧版の場合は XView) がインストールされた環境に限定される.

ここでは surf を手軽に用いる為に必要な事項の解説を行う. より詳細に関しては, surf に附属のマニュアル等を参照されたい.

### 1.1.2　surfの操作ウィンドウ

surf 単独で起動する場合,KDE のメニューより surf を選ぶか,rxvt 等の端末で直接 `surf` と入力すれば, 図 1.1 に示す surf の制御ウィンドウが現れる. 尚, 曲線や曲面は別の描画用ウィンドウに表示されるが, この描画用ウィンドウは実際に描画を開始するまで現れない.

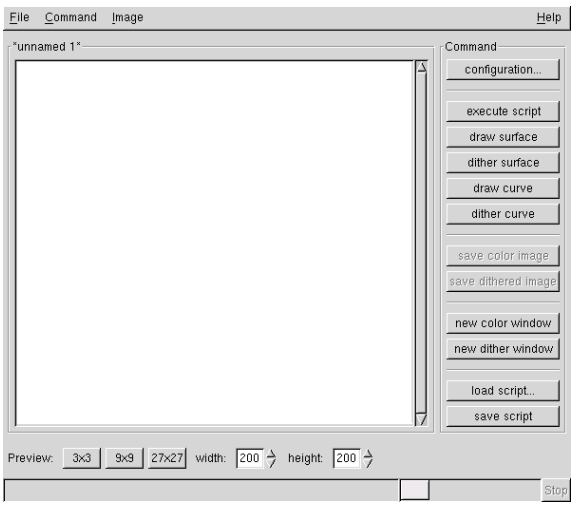


図 1.1: surf を立ち上げた状態

制御ウィンドウでは, スクリプトの記述から実際の描画, スクリプトの読込と保存, 描画する曲線や曲面の色彩, 透明度等の様々な設定や, 曲線や曲面を描く際に用いる解法の選択やそのパラメータの設定等が行える.

先ず,surf に描かせる式を含むスクリプトは図 1.1 の左側の大きな空欄に書込む. この空欄の下側に,`Preview:` という名目で, 3x3 ,9x9 ,27x27 と表記された 3 個のボタンが並んでいる. これらのボタンはトグルになっており,`nxn` のボタンを押せば, 曲面を表示する時点で,`nxn` 個の画素を一つのブロックとして曲面を標準よりも粗く表示する. これらのボタンを押さなければ, 一画素単位で描画を行い, 画像も最も細密に描かれる. ボタンを押す場合, 3x3 , 9x9 , 27x27 の順番で描かれる曲面が粗くなる. この機能は曲面が非常に複雑な場合、或いは計算機が非力な場合に, 手際良く曲面を表示したい時に用いると良い.

これらボタンの横に,`width` と `height` と表記され,200 と入力されている二つの入力欄がある. これは描く絵全体の大きさを指定するものである. デフォルトの絵全体の大きさは `200x200` である. 計算機が高速であれば, この値を大きなものにしても良い. 大体,Pentium 3 の 500MHz 以上であれば `400x400` で困る事は無い. 尚, この表示用ウィンドウの大きさはスクリプト内で変数 `width` と `height` に直接, 整数値を割当てて指定しても良い.

スクリプト記述欄の右側には `Command` との名目で纏められた 12 個のボタンが並んでいる. これらは操作内容に分類されてグループ化されている. 先ず, 一番上にある Configuration ボタンを押すと, 図 1.2 に示す surf の様々な制御を行うウィンドウを呼出し,surf の初期設定が行える. この中で特に注意を要するのが numeric で, ここでは解を検出する為の手法 (`rootfinder`) と繰返し計算の回数, 誤差の上限 (`epsilon`) の設定が行なえる. この個所は自己交差を持つ複雑な曲面や曲線, 平面による曲面の切断を描く際に特に重要となる.

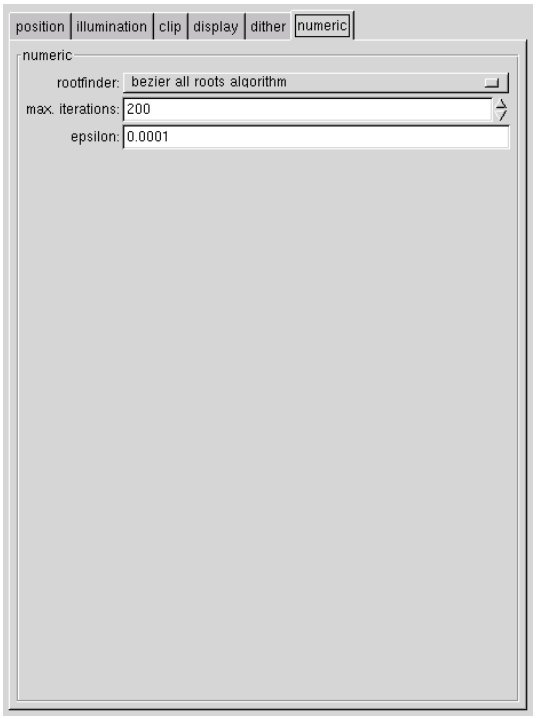


図 1.2: Configuration ボタンを押して出るウィンドウ

Configure ボタンの下には描画関連のボタンが 5 個纏めて配置されている．その中で一番上にあるのが execute script ボタンで, 左側空欄に記述したスクリプトを実行する. draw surface や draw curve ボタンもスクリプトで設定した曲面や曲線を描くが, スクリプト内部で `draw_surface` や `draw_curve` 命令が記述してあれば, 自動的に描画を開始するので, これらのボタンを使う必要は無い.

surf は基本的にカラーで曲線を描き, グレイスケール画像では表示を行わない．その為, モノクロ印刷向けにディザのかかった白黒画像を生成する事が可能である．これは dither surface や dither curve ボタンを用いる．このボタンを押すと, 描いた画像が表示されたウィンドウとは別のウィンドウが生成され, そこにディザをかけた画像が表示される. 以下の図 1.3 で, 左側にスクリプトで描いた画像, 右側にディザをかけた画像を示しておく.

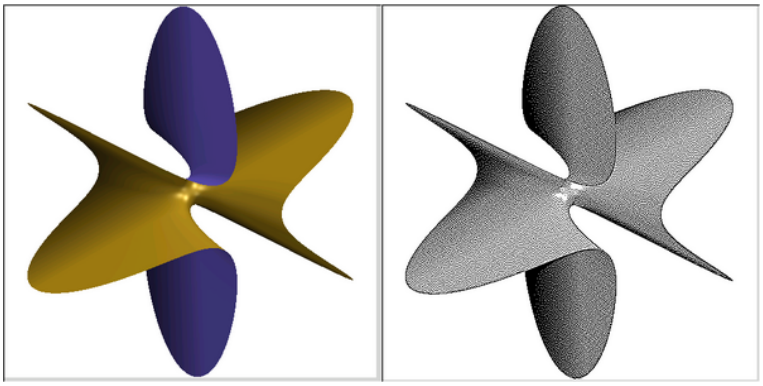

図 1.3: 元画像 (左) とディザをかけた画像 (右)

描画に関連するボタン群の下には, 生成した画像を保存する為のボタンが並んでいる. これらのボタンは生成した画像データを保存する為に用いる. ボタンを押すと, 図 1.4 に示すウィンドウが現われる. 尚, 保存可能なデーター形式には,ppm,.ras,.xwd,.jpeg が選らべる.

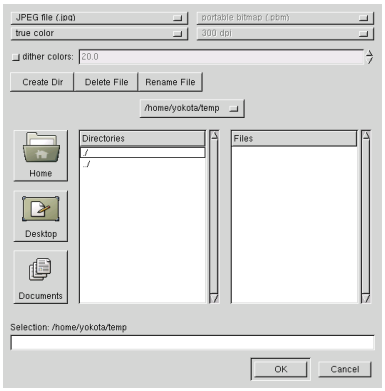


図 1.4: 画像保存の際に出るウィンドウ

画像保存のボタン群の下側には, カラー表示の新しいウィンドウを生成する new color window ボタンと白黒ディザ表示の新しいウィンドウを生成する new dither window ボタンがある. これらのボタンを押して生成されたウィンドウに画像が表示される.

その下にはファイル操作のボタン群がある. 先ず, load script... ボタンを押すと, ファイルに記述されたスクリプトの読込を実行する. その下にある save script ボタンで左側の入力欄に記述したスクリプトの保存を行う. 図 1.5 には load script... ボタンを押した際に現われるウィンドウを示している. save script を押した場合も同じウィンドウが現われる.

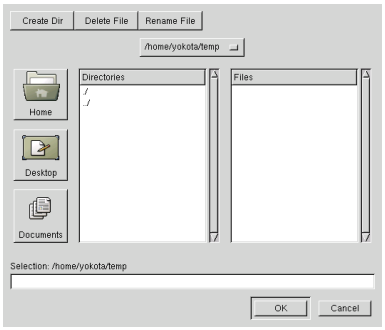


図 1.5: スクリプト保存の際に出るウィンドウ

一番下には Stop ボタンがある. このボタンは描画を中断する際に用いる.

以上で surf の GUI に関する説明を終える. 次の節では surf のスクリプトに関して簡単に述べる事とする.

### 1.1.3 スクリプト概要

surf には非常に原始的だが,C 風の処理言語を持っている. スクリプトの記述は図 1.1 に示した制御用ウィンドウ左側の大きな空欄に直接記述したり, Emacs 等のエディタで編集し, ファイルに保存したものを読込んだする事が可能である. 勿論,surf の起動時に`-x` オプションでファイルを指定する事で, バッチ処理を行う事も可能である.

surf には予約語 (命令や内部変数等) として登録されたものがあり, それらは変数名としては使えない. 又, 予約語に関しては宣言は不要である. 予約語に関しては, surf のマニュアルを参照されたい.

surf の入力は必ず; で終えなければならない. 又, 利用者が予約語以外の変数を利用する場合, 必ず変数を宣言しておく必要がある. ここで,surf で扱える変数のデータ型を表 1.1 に示しておく.

表 1.1: 変数の宣言

| 型 | 例 |
|---|---|
| 整数 | `int a=1;` 又は `int a; a=1;` |
| 倍精度 | `double b=1.28;` 又は `double b;b=1.28;` |
| 文字列 | `string s="test.abc";` 又は `string s; s="test.abc";` |
| 多項式 | `poly f=(x+y)^2-2*x*y+z^2;` 又は `poly f; f=(x+y)^2-2*x*y+z^2;` |

変数型に多項式があるが,surf では数式処理を行う訳ではない, 描く曲線や曲面を表現する為に用いるものである.

次に, 和, 積, 商, 冪等の算術演算子は C と全く同じものが利用可能である. それに対し、数学関数は表 1.2 に示すものに限定される.

表 1.2: 利用可能な数学関数

| 関数 | 意味 | 引数 | 返値 |
|---|---|---|---|
| sqrt | 平方根 | sqrt(int,double) | double |
| pow | 冪 | pow(int,int,double),pow(double,int,double) | double |
| sin | 正弦関数 | sin(int,double) | double |
| cos | 余弦関数 | cos(int,double) | double |
| arcsin | 逆正弦関数 | arcsin(int,double) | double |
| arccos | 逆余弦関数 | arccos(int,double) | double |
| tan | 正接関数 | tan(int,double) | double |
| arctan | 逆正接関数 | arctan(int,double) | double |

多項式に対しては表 1.3 に示す関数がある.

表 1.3: 多項式に作用する関数

| 関数 | 意味 | 引数 | 返値 |
|---|---|---|---|
| deg | 次数 | deg(poly) | int |
| len | 長さ | len(poly) | int |
| diff | 微分 | diff(poly,x,y,z) | poly |
| rotate | 回転 | rotate(poly,double,X-Axis,Y-Axis,Z-Axis) | poly |
| hesse | hesse 曲面 | hesse(poly) | poly |

微分, 回転,Hesse 曲面と, 入力された多項式に対して多項式を返す命令であるが, surf のスクリプトを実行しても, 直接その結果が何処かに出力される訳ではない. 内部的に用いられるだけである.

他に, 整数を単純に文字列に変換する `itostr`, 文字列の長さを指定して整数を文字列に変換する `itostrn` がある. これらの関数は自動処理で生成した複数の画像をファイルに落す際に決めた長さの文字列でファイル名を指定する際には重宝する.

ここ迄は比較的 C に近いものであったが, 制御文やループ処理に関しては大きく異なる. 実際,`for` 文や `while` 文を持たず,`if-goto` 文のみでループ処理を行う.

この `if-goto` 文の書式を以下に示す.

```
ラベル:
文 1;
.....
文 n;
if ( 整数式 ) goto ラベル;
```

この文では最初に文 1 から文 n までが実行され, 整数式が満されている場合には, ラベルに飛び, 文 1 から文 n を実行する. 整数式が満されなくなると, このループを抜け,`repeat-until` 文に似た動作を行う. 尚, ラベルは他の文とは異なり, :で終るのが特徴である. この点には十分に注意さたい.

以上で簡単に surf の概要の解説を終える. より詳細はマニュアルを参照されたい. 次の節では, 具体例を含めて重要な命令の解説を行う.

### 1.1.4 実行例

surf のスクリプトは単純である. 先ず, 最初に利用する変数の宣言を行ない, それから, 描く対象が曲線ならば,x と y の多項式を予約語 curve に割当て, 曲面ならば x,y,z の多項式を surface に割当てる. 更に, 曲線であれば draw_cuvrve, 曲面であれば draw_surface を式の割当て後に入れると, execute script ボタンを押した時点で,surf はスクリプトの処理を開始し, 与えた多項式の零点集合を描画する.

**曲線の表示例**　次の式で定義される多項式 $P_\delta$ を surf で描こう.

$$P_\delta\,(x,y) = x^3 + y^3 - 3xy - \delta$$

この多項式 $P_\delta$ の零点集合のグラフは $\delta = 0$ の時にデカルトの葉状曲線と呼ばれるものとなる.
ここでは if-goto 文を用いたループを用いて, 擬似的なアニメーションを行うスクリプトを示す.

```
width=400;
height=400;
double delta;
delta=0.5;
curve_width=2;
curve_red=255;
curve_green=255;
curve_blue=255;
clear_screen;
Descartes:
  curve=x^3+y^3-3*x*y-delta;
  clear_screen;
  draw_curve;
  delta=delta-0.01;
if (delta>-0.5) goto Descartes;
```

先ず、width=400; と height=400; で画像の大きさを幅 400, 縦 400 画素に設定している. 次に内部で助変数 $\delta$ を利用する為,double delta; で $\delta$ を倍精度の数値であると宣言している.curve_width で曲線の太さを定義している. デフォルトが 1.0 なので, 倍の太さを指定している事になる. これは太い方がアニメーションを実行する場合には見易いからである. 次の curve_red 等で曲線の色を指定している. この様に色の指定は 0 から 255 の整数値を RGB で指定する.clear_screen 命令は曲線の消去を行なう命令である.clear_screen 命令が無ければ曲線が重ね書きされる. Descartes: は if-goto で用いるラベルである. このラベルの末尾が:になっている事に注意されたい.
図 1.6, 図 1.7 と図 1.8 に $P_{-0.5}$,$P_0$ と $P_{0.5}$ のグラフを示しておく.

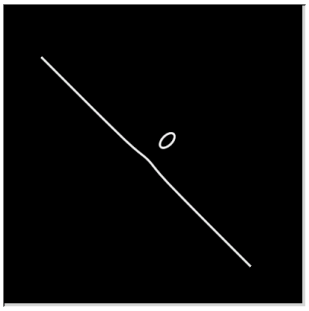

図 1.6: $x^3 + y^3 - 3xy + 0.5$

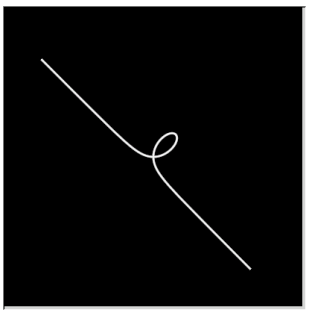

図 1.7: $x^3 + y^3 - 3xy$

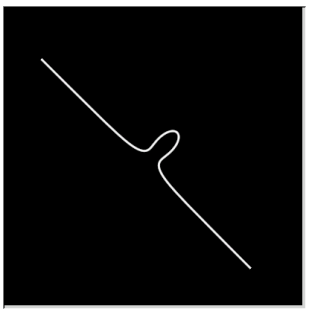

図 1.8: $x^3 + y^3 - 3xy - 0.5$

**曲面の表示**　先程の関数族 $P_\delta$ を曲面として描こう. 先ず,surf では多項式の変数を x,y,z にしなければならない. そこで,$\delta$ を z に置換え, 次の式の描画を行う.

$$x^3 + y^3 - 3 * x * y - z$$

以下に安易なスクリプトを示す.

```
width=400;
height=400;
clear_screen;
surface=x^3+y^3-3*x*y-z;
draw_surface;
```

最初に説明した様に,surf で曲面を描く場合, 曲面を定義する多項式を surface に代入し,`draw_surface` を実行すれば良い. このスクリプトを実行して表示された曲面を図 1.9 に示す.

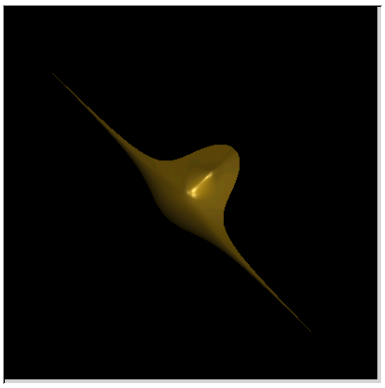

図 1.9: $x^3 + y^3 - 3xy + z$

実際に実行すると判るが, この図では金色の面しか見えていない.surf のデフォルトでは曲面の表を青, 裏側を金色とする為, この図では裏面しか見えていない. そこで,`rot_x`,`rot_y`,`rot_z` に適当な値を入れで回して見よう.

```
width=400;
height=400;
rot_x=0.1;
rot_y=1.8;
rot_z=0.6;
clear_screen;
surface=x^3+y^3-3*x*y-z;
draw_surface;
```

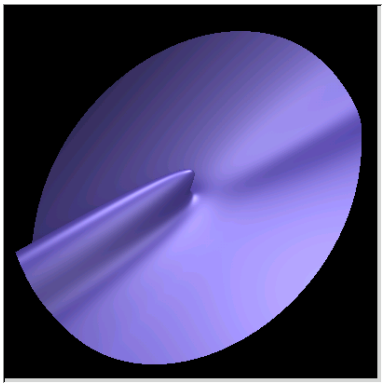

図 1.10: $x^3+y^3-3xy-z$ を回転したもの

このスクリプトを実行すると図 1.10 に示す絵が得られる. この図から $x^3+y^3-3xy-z$ の原点が鞍点となっている事が判る.

surf では平面による曲面の断面を表示する事が可能である. 平面を定義する多項式は予約語 `plane` に代入する. それから, 断面を描く為には `curve_with_plane` を入れると良い. 以下に平面 $Z=\delta$ による切断を描くスクリプトを示す.

```
width=400;
height=400;
double delta=0.6;
rot_x=0.1;
rot_y=1.8;
rot_z=0.6;
clear_screen;
surface=x^3+y^3-3*x*y-z;
draw_surface;
Descartes:
   plane=z-delta;
   cut_with_plane;
   delta=delta-0.2;
if (delta>-0.6) goto Descartes;
```

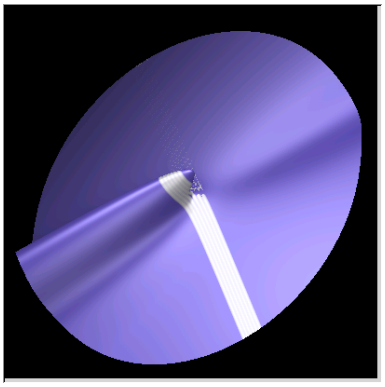

図 1.11: $x^3 + y^3 - 3xy - z$(その 3)

図1.11 を見ると, 断面を現す曲線が手前は綺麗に見えるものの、奥側が見えず, 交点も明瞭ではない. この様に, 曲線や曲面の自己交差や切断面が不明瞭な場合, 解法の変更やパラメータの調整を行う必要がある. この為には,surf の制御用ウィンドウ上の `configuration...` ボタンを押し, 新たに現われるウィンドウの `numeric` を選択すると,`rootfinder` の欄と解法パラメータの二つの欄が現れる. 今回は, デフォルトの解法の `bezier all roots algorithm` 以外の解法, 例えば,`D-chain and newton` に変更すると,surf は綺麗な断面を描く.

最後に, この断面を見易い様に, 鞍点の周辺を拡大し, 視点変更と曲線が見易い様に曲面を半透明にしたスクリプト例を以下に示し, そのスクリプトを実行して得られるグラフを図 1.12 に示しておく.

```
width=400;
height=400;
double delta=0.6;
scale_x=0.3;
scale_y=0.3;
scale_z=0.3;
rot_x=2.8;
rot_y=-0.2;
rot_z=0;
illumination=ambient_light+
diffuse_light+reflected_light
+transmitted_light;
transparence=50;
clear_screen;
surface=x^3+y^3-3*x*y-z;
curve_width=5;
draw_surface;
curve_red=255;
curve_green=0;
curve_blue=0;
Descartes:
   plane=z-delta;
   cut_with_plane;
   delta=delta-0.2;
if (delta>-0.6) goto Descartes;
```

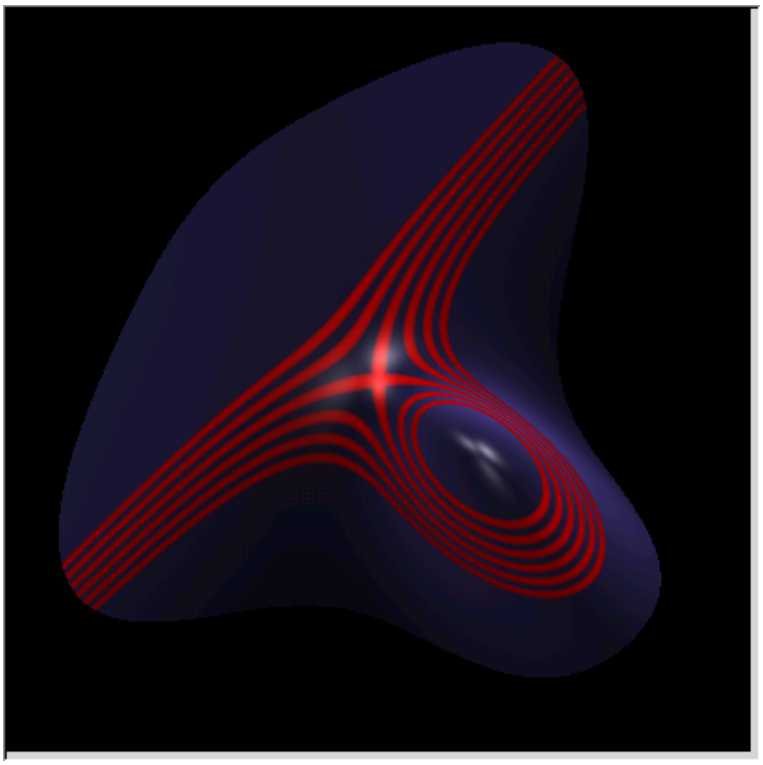

図 1.12: $x^3 + y^3 - 3xy - z$ と平面 $Z = \delta$ との断面

## 1.2 SINGULAR

### 1.2.1 SINGULAR 概要

SINGULAR は可換代数, 代数幾何と特異点理論に特化した計算機代数システムである. SINGULAR では, その処理は対話的に行われ, バッチ処理にも利用可能である. SINGULAR は C 風の処理言語を持っており, この処理言語でプログラムを組んだり, 更には, この言語で記述したプログラムでライブラリを構成して機能を拡張する事が可能である.SINGULAR 自体には GUI やグラフ表示機能を持っていないが、編集に Emacs, グラフ表示に Surf(後述), ヘルプに Mozilla といった外部プログラムで代用が出来る.

### 1.2.2 SINGULAR の起動とオンラインマニュアル

SINGULAR には, 通常の端末で利用可能な Singular,Emacs を起動して, その中で動作する ESingular, 端末を指定して, その端末上で起動する TSingular の三種類ある. Singular と TSingular は計算履歴をそのままファイルに保存する事が出来ない為, 一寸した計算は Singular や TSingular, 履歴をファイルに残したり, プログラムを記述する場合には ESingular を用いると良いだろう.

SINGULAR を起動すると以下の画面が表示される:

```
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations   /   version 3-0-0
                                                        0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann         \   May 2005
FB Mathematik der Universitaet, D-67653 Kaiserslautern    \
>
```

>が SINGULAR の入力行プロンプトである. 入力はこのプロンプトに続けて行う。SINGULAR の入力では, 末尾にセミコロン; を必ず付ける. セミコロンが無い場合,Singular は入力が継続していると判断して入力行の評価を行わずにプロンプトが">"から"."となる. 又, 入力した文字列に関しては, 大文字と小文字を判別する事に注意する.

SINGULAR は固有の GUI を持たない為, ヘルプを環境変数 BROWSER で指定されたブラウザ (デフォルトは mozilla) を用いる. 更に,ESingular では環境変数 EMACS で指定された Emacs(Emacs 或いは XEmacs 但し, デフォルトでは XEmacs) を用いる.

デフォルト以外のものを指定したい場合,SINGULAR を起動する際にオプションで Emacs の指定を行うと良い. ブラウザの場合,`--browser=(ブラウザ名)`, Emacs の場合,`--emacs=(emacs 名)` を与える. 例えば,ESingular でブラウザを info,emacs を emacs にする場合, 通常の端末、或いは KDE メニューからコマンドの実行を選択して現われるダイアログボックスの入力欄に以下を入力する:

```
ESingular --browser=info --emacs=emacs&
```

起動時に設定するオプションの幾つかは system 命令を用いて SINGULAR 内部で設定し直す事も可能である. 例えば, オンラインマニュアルの閲覧で用いるブラウザに info を利用する場合は以下の様に設定する.

```
>system("--browser","info");
```

他のオプションの一覧は,--help オプションを付けて SINGULAR を起動するか, SINGULAR の中で>system("--help","");を実行すると表示される. では実際に system のオンラインマニュアルを閲覧を行おう. オンラインマニュアルの閲覧は, 表 1.4 に示す様に?を調べたい項目の頭に付けて入力するか,help 命令を用いる.

表 1.4: オンランマニュアルの起動

| 命令 | 構文 | 入力例 |
|---|---|---|
| ? | ? 項目; | ?system; |
| help | help(項目); | help system; |

すると, デフォルトのままであれば mozilla が, 或いは--browser オプションで指定したブラウザが立ち上がる. 利用可能なブラウザは SINGULAR のバージョンによって異なるので注意が必要である. 尚,system("--browsers"); を実行すると, ブラウザとして利用可能なアプリケーションが表示されるので, 必要に応じて system 命令で変更して良い.

尚, 該当する項目が複数存在する場合には, ブラウザを起動せずに, その候補を全て表示する. その中から妥当なものを選んで再度ヘルプを起動させる事になる. 又, 項目無しに命令単体を入力した場合は,SINGULAR のマニュアルの表紙がブラウザに表示される.

SINGULAR の終了は quit; と入力すれば良い.quit; を入力すると, SINGULAR は Auf Wiedersehen. と出力して終了する.

### 1.2.3 SINGULAR で遊ぶ

再度,SINGULAR を立ち上げて, 今度は整数や文字列を入力しよう.

```
                     SINGULAR                                 /
 A Computer Algebra System for Polynomial Computations   /   version 3-0-0
                                                        0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann         \   May 2005
FB Mathematik der Universitaet, D-67653 Kaiserslautern    \
> 1;
1
> 1+2+3*4-5;
10
> "Hello world!"
. ;
Hello world!
> 4/2;
   ? no ring active
   ? error occurred in STDIN line 6: ‘4/2;‘
> 1.24;
   ? no ring active
   ? error occurred in STDIN line 7: ‘1.24;‘
> int a=1;
> string b="Hello world!";
> list c=1,2,3,4,5,"Hello world!";
> c. ;
[1]:
   1
[2]:
   2
[3]:
   3
[4]:
   4
[5]:
   5
[6]:
   Hello world!
>
```

この例で示す様に, 整数の和, 差, 積と文字列の処理は SINGULAR を立ち上げて直に出来る事が判る. 又, 行の末尾に; が無い場合, プロンプトが>から . になり, 継続行の入力待ちとなっている事が判る.

SINGULAR での文字列は"Hello world!"の様に引用符で囲ったものである. 最後の二つの入力では分数と小数を入力しているが, どちらもエラーになっている. これは分数や小数が扱える様な環 (ring) が SINGULAR 上でまだ定義されていない為である.

この例では整数型, 文字列型とリストの宣言を行っている.SINGULAR で扱う対象 (Object) には全て型 (Type) がある.SINGULAR で扱える型には多くのプログラム言語で見られる整数型 (int), 実数型 (real), 複素数型 (complex), 文字列型 (string), リスト (list) や行列 (matrix) に加え, 環 (ring), 多項式 (poly), イデアル (ideal), 商環 (qring) といった代数学で利用される対象もある.

基本的に,SINGULAR の対象は基礎環 (base ring) と呼ぶ環上で宣言し, その環上で処理を行う. 上記の例で宣言した整数型, 文字列型やリストは任意の環上で宣言可能であるが, 基礎環によっては宣言出来ない対象もある. 例えば, `ring r=0,x,dp;` で定義した環では実数が扱えない.

SINGULAR では環の宣言を以下の方法で行う.

```
ring 環の名前=係数体,(変数 1,... ,変数 n),順序;
```

係数体は実数 $R$, 複素数 $C$, 有理数 $Q$, 有限体 $Z_p$, 代数学拡大体等が設定可能で, 標数 p は 2147483629 以下の整数が指定可能である. ここで指定する順序は単項式順序で, 代表的なものに, 辞書式順序 (lp), 斉次辞書式順序 (Dp), 斉次逆辞書式順序 (dp) と重み付き逆辞書式順序 (wp) と重み付き辞書式順序 (Wp), 負辞書式順序 (ls), 負斉次逆辞書順序 (ds), 負斉次辞書順序 (Ds), 負重み付き逆辞書式順序 (ws) と負重み付き辞書式順序 (Ws) 等が選べる. 尚,SINGULAR で扱える順序の詳細についてはオンラインマニュアルを`?Monomial orderings;` で参照されたい。尚,`ring` 環の名前`;` で環を宣言する事も可能であるが, この場合は, 係数環が $Z/32003$, 変数が x,y,z で順序が dp の環となる.

環の定義例を次に示す.

```
> ring r=0,(x,y,z),lp;
> ring r1=(0,a),x,dp;
> minpoly=a^2+1;
> r1;
//   characteristic : 0
//   1 parameter    : a
//   minpoly        : (a2+1)
//   number of vars : 1
//        block   1 : ordering dp
//                  : names    x
//        block   2 : ordering C
>
```

先ず,1 変数の場合は `r1` の宣言の様に変数を括弧で括らなくても良い. 単純拡大体を係数体とする環を宣言する場合, 最初に環全体を宣言してから, 係数体の元の最小多項式を `minpoly` に割当てる. ここで minpoly の型は数 (number) であり, 多項式 (poly) ではない事に注意する. この様に最初に環を宣言してから, 付随する最小多項式を後で `minpoly` に設定するのが SINGULAR の流儀である.

ここで, 基礎環は特別に定義するものではなく, ポインタが置かれている環が基礎環となる.SINGULAR では環の宣言を行うとポインタが新規に宣言された環に移動する. 多項式, イデアル, 写像や商環等の対象はポインタがある環上で宣言される為, 必要に応じてポインタを移動させなければならない. ポインタを現在の環とは別の環に移す場合,setring 命令を用いる. 使い方は以下の様に環名前を直接指定すれば良い.

```
setring 環の名前;
```

更に, 基礎環の内容を知りたい時には, 基礎環の名前を入力すると基礎環の情報が返される. 又, 基礎環は内部変数 basering に割当てられる為,basering; と入力すると同様に環の内容を返す. 更に,nameof(basering); を実行すると, 基礎環の名前が返される.

この様子を以下の例に示しておく.

```
>  ring R=0,(x,y,z),lp;
> R;
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering lp
//                  : names    x y z
//        block   2 : ordering C
> basering;
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering lp
//                  : names    x y z
//        block   2 : ordering C
> ring Q=(3,a),(x,y),dp;
> basering;
//   characteristic : 3
//   1 parameter    : a
//   minpoly        : 0
//   number of vars : 2
//        block   1 : ordering dp
//                  : names    x y
//        block   2 : ordering C
> setring R;
> nameof(basering);
R
>
```

次に多項式, イデアルと商環の宣言方法を示す.

- poly 多項式名 = 多項式;
- ideal イデアル名 = 多項式$_1$, ..., 多項式$_n$;
- qring 商環名 = イデアル;

多項式の場合, 積記号*や冪記号^を省略して記述する事が可能である. 即ち,x*y^2 を xy2 と記述可能である. ここで,xy2 と xy^2 は別物である事に注意が必要である. 前者は x*y^2 と同値であるが, 後者は (xy)^2 となる. この様に, 積と冪記号を省略する場合は, 両方を省略するか, どちらも省略せずに記述するか明確にする必要がある.

イデアルの宣言で多項式を括弧 () で括っても構わない. 尚, イデアルを表示すると多項式のリストの様に出力される. イデアルから多項式を取り出すのは以下の方法で行うが, これはリストの場合と同様の方法である.

```
> ideal I=x+y^2-1,z^2+y*x;
> I;
I[1]=x+y2-1
I[2]=xy+z2
> poly p1=I[1];
> poly p2=I[2];
> p1;
x+y2-1
> p2;
xy+z2
>
```

この例ではイデアル I を宣言し, このイデアルを生成する 2 個の多項式を I[1] と I[2] で取り出している. この方式で取り出したものの型は多項式である.

商環 R/I の宣言で指定するイデアル I は単純に ideal で宣言したものも使えるが, その場合はイデアルが標準基底で無いと警告する. イデアルの標準基底は std(イデアル); で得られるので, そちらを使う方が警告が出なくて良い. std 命令の他に, 標準基底を計算する命令に groebner もある.

商環の宣言例を以下に示す.

```
> ring  r=0,(x,y,z),dp;
> qring ri=std(x^2+y^2-1);
> ri;
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering dp
//                  : names    x y z
//        block   2 : ordering C
// quotient ring from ideal
_[1]=x2+y2-1
> r;
//   characteristic : 0
//   number of vars : 3
//        block   1 : ordering dp
//                  : names    x y z
//        block   2 : ordering C
>
```

この例では基礎環 $Q[x,y,z]$ 上で, 商環 r1 の宣言で必要なイデアルを std(x^2+y^2-1); で与えている. この商環 ri は具体的には, $Q[x,y,z]/\left(x^2+y^2-1\right)$ である. この商環の情報は, 上述の様に対象名を SINGULAR に入力する事で表示される. この例では,r1 と r に関して表示されせている. 最初の ri; の実行で, その指標 (characteristic) は 0 で, 環の変数は x,y,z の 3 個, 順序は dp で, イデアルの商環 (quotient ring) である事が読み取れる.

SINGULAR で写像の宣言では map 函数を用いる. 例えば, 環 $A$ から $B$ への写像 $f : A \to B$ を宣言する場合,B を基礎環とする状態で写像 $f$ の宣言を行う. その為, 写像の宣言では, 定義域となる環と, その環の変数がどの様に基礎環の変数の多項式として写されるかを定義する.

写像の宣言例を次に示しておく.

```
> ring r1=0,(x,y,z),dp;
> ring r2=0,(a,b),dp;
> map f=r1,a,b,0;
> f;
f[1]=a
f[2]=b
f[3]=0
>
```

この例では, 環 r1 と環 r2 を各々$Q[x,y,z]$,$Q[a,b]$ とし, r1 から r2 への写像 f$(x,y,z) \to (a,b,0)$ の定義では,r1 の変数 x,y,z を基礎環の元,a,b,0 に各々対応させている. この様に, 写像の宣言では, 定義域となる環の各変数を, 写像の値域となる基礎環の元に一つ一つ定めて行けば良い. 尚,map 命

令には, 環の係数体に関して制約がある. 基本的に, 環の係数体の間に, 包含写像から誘導される写像が存在する場合は宣言可能だが, そうでない場合には宣言出来ない事がある. 詳細は?map を参照されたい.

次に,SINGULAR では写像によるイデアルの逆像も計算可能である. 例えば, 上の例で r2 のイデアル i2 の f による逆像は preimage(r1,f,i2); で求められる. 以下に示す例ではイデアル i2 を零イデアルとして f の核を計算している.

```
> ring r1=0,(x,y,z),dp;
> ring r2=0,(a,b),dp;
> map f=r1,a,b,0;
> ideal i2=0;
> setring r1;
> preimage(r2,f,i2);
_[1]=z
>
```

この例では零イデアルを環 r2 上にて ideal i2=0; で宣言している. 一般的に,ideal i2; の様に名前だけでイデアルを宣言すると, 零イデアル, 即ち,i2=0; が宣言される. ここで,i2 は環 r2 に含まれる対象であるが, 写像 f による i2 の逆像は環 r1 に含まれる対象である. その為に,preimae の前に setring r1; を実行してポインタを r2 から r1 に戻している.

これで最低限遊ぶ為の道具の説明を終えた. 次の節では SINGULAR で計算した曲線や曲面を定義する多項式函数を surf を用いて可視化する方法について解説する.

## 1.3 Singular から surf を使う

この節では SINGULAR から surf を用いて図形表示する方法について簡単に述べる。surf の詳細については surf に附属のマニュアルを参照されたい.

先ず,SINGULAR は前述の様に C 風の処理言語を持っており, その処理言語を用いてライブラリを構築し, 機能の拡張等が行える.SINGULAR には様々なライブラリが附属している. 附属のライブラリの詳細はマニュアルを参照されたいが, 標準で附属するライブラリに SINGULAR の計算処理で得られた多項式を前述の surf を用いて表示出来る `surf.lib` が含まれている.

SINGULAR から surf を使って曲線や曲面を描く場合, 予めライブラリ `surf.lib` を読込んでおかなければならない (SINGULAR には自動的に読込む機能は無い).SINGULAR 言語で記述されたライブラリの読込は `LIB` 命令で, `LIB "surf.lib";` の様に実行する。

SINGULAR から surf を使って描けるものは, 基本的に一つの多項式, 又は一つの単項生成イデアルに限定される. 描く対象が曲線か曲面であるかどうかは、基礎環の変数の個数と多項式の変数の個数から自動的に決定している. 尚, 曲面の場合は SINGULAR から対話的に視点を変更するといった機能も無いが, 引数として記述した surf の命令を直接引き渡す事が可能なので, 多少の融通なら効く. とはいえ, `surf.lib` は引き渡された命令を単純に surf のスクリプトの所定の位置に挿入する機能しか無く, 引き渡した命令が結果に反映されるとは限らない事に注意が必要である.

以下に SINGULAR を立ち上げてから曲線を描くまでの処理を示す.

```
                     SINGULAR                                /
 A Computer Algebra System for Polynomial Computations   /   version 2-0-5
                                                       0<
     by: G.-M. Greuel, G. Pfister, H. Schoenemann        \   March 2004
FB Mathematik der Universitaet, D-67653 Kaiserslautern    \
> ring r=0,(x,y),dp;
> LIB "surf.lib";
// ** loaded /opt/Singular/2-0-5/LIB/surf.lib (1.19.2.6,2002/07/17)
> plot(x^3+x^2-y^2);
```

先ず, ライブラリの読込を `LIB` で行った時点で, 読込んだライブラリファイルへのパスとバージョンが表示される事に注意されたい. `LIB` 命令は, 最初にカレントディレクトリ上に指定したファイルがあればそれを読込み, カレントディレクトリに無ければ, ライブラリファイルが格納された `LIB` ディレクトリから読込む. 因にライブラリファイルは SINGULAR の処理言語で記述されたアスキー形式のファイルである. このファイルの内容に関しては後で解説する.

`surf.lib` の `plot` 命令で図形を描く場合, 最低でも環が定義されている必要がある. ここでは `plot` 命令を実行する前に, 環 $Q[x,y]$ を定義している. 尚,`surf.lib` の読込自体は対象を描く前に読込まれていさえすれば良い.

`plot(x^3+x^2-y^2);` を実行すると,SINGULAR 内部で surf 向けのスクリプトを生成し, そのスクリプトを呼出した surf に引き渡して描画を開始する. 図 1.3 に,surf で描いた曲線を示す.

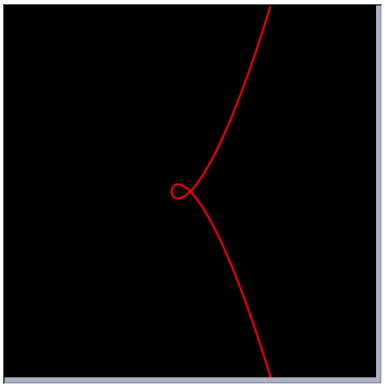

図 1.13: plot で描いた曲線

surf.lib を用いて描く絵の大きさは 500X500 に固定されている. 背景色は曲線の場合は黒, 曲面の場合は白がデフォルトで指定されいる.

以下に plot の構文を示す.

```
plot(多項式,"命令 1; 命令 2;... 命令 n;");
```

先ず,plot に与える第一引数は多項式, 或いは多項式で生成されるイデアルでなければならない. この多項式の変数は 2 変数か 3 変数のものに限定される. 助変数表示のものは扱えない. 第二引数はオプションであり, これは surf で実行させる命令をセミコロンで区切った一つの文字列である. 前述の様に,surf に引き渡されても効果の無い場合もある. これは plot 命令で引き渡した命令の挿入される場所が, その命令を実行する上で不適切な場合がある為である, オプションを反映される為に一番確実な方法は, 直接 surf 側でスクリプトを操作する事である.

そこで,(図 1.3 に示す) グラフが表示されているウィンドウを右クリックしよう. すると, 前節の図 1.1 に示した surf の制御画面が呼出される. 尚,SINGULAR で生成したスクリプトは左側の大きな入力欄に記述されている.

この入力欄に命令の追加や修正を行った後, 右側の execute script ボタンを押せば直ちに描写を開始する. ここで,verb+plot+命令で引き渡した surf の命令に誤りがある場合も, このウィンドウが呼出されて問題の個所にカーソルが置かれている.

surf.lib では,SINGULAR から対話的に操作が行えず,surf のスクリプトを調整してゆく必要がある. この surf.lib を改良したライブラリも幾つか存在しているが,surf を用いて図形を描くもので現在の SINGULAR に同梱されているライブラリは surf.lib のみである.

前述の preimage を用いてイデアルが定める曲面を描く例を以下に示す.

```
> ring r=0,(x,y,z),dp;
> poly sp4=x2+y2+z2-16;
> ideal i1=sp4;
> map f=r,xy,yz,zx;
> ideal steiner=preimage(r,f,i1);
> steiner;
steiner[1]=x2y2+x2z2+y2z2-16xyz
> plot(steiner,"background_red=0;background_green=0;
. background_blue=0;rot_x=2;rot_y=0.5;");
```

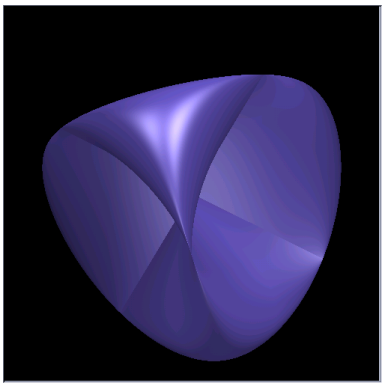

図 1.14: シュタイナーのローマ曲面

この例では、写像 $f : (x, y, z) \to (xy, yz, zx)$ による半径 4 の三次元球面の逆像を preimage で求め,plot 命令に Surf の背景色を設定する命令と一緒に引き渡している. 尚, この背景色の設定が長い為に途中で改行を入れているが, SINGULAR は行末のセミコロンが未入力の為に行が継続中であると判断して, ピリオド . を出している事に注意されたい.

曲線や曲面が, 助変数表示されている場合, そのまま plot に助変数が定義するイデアルを引き渡しても,surf.lib では図形が描けない. これは, surf.lib では対象が単項イデアルが一つの多項式に限定される為である. 逆に言えば, 助変数表示から一つの多項式を抽出すれば良く, これには eliminate 命令が使える. この eliminate 命令は不要なイデアルの変数を削除する命令で, 以下の様に二つの引数を取る.

```
eliminate(イデアル, 消去する変数の積);
```

今度は eliminate 命令を用いて図 1.3 に示すデカルトの葉状曲線を描く例を以下に示す.

```
> ring r1=0,(x,y,t),dp;
> ring r2=0,(x,y),dp;
> map f=r1,x,y,0;
> setring r1;
> ideal i1=(1+t^3)*x-3*t,(1+t^3)*y-3*t^2;
> ideal i2=eliminate(i1,t);
> poly c2=i2[1];
> c2;
x3+y3-3xy
> setring r2;
> plot(f(c2));
```

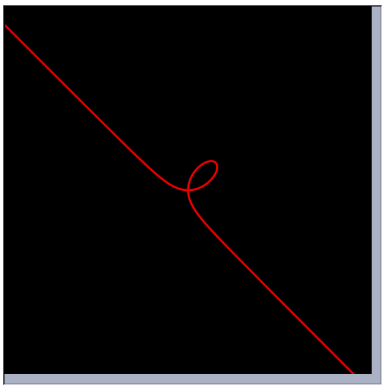

図 1.15: デカルトの葉状曲線

この例では, 助変数を含めた環 r1 とそれを除いた環 r2, 更に, r1 から r2 への射影 f を宣言し, 次に、r1 上でイデアル i1 を宣言している. デカルトの葉状曲線は助変数 t を用いて $x = \frac{3t}{1+t^3}, y = \frac{3t^2}{1+t^3}$ で表現される. 但し, ここでは, 基礎環 r1 が $Q[x, y, t]$ の為, イデアル i1 の宣言では分子をかけた形で宣言している. 次の eliminate 命令で, 助変数 t を削ったイデアル i1 の表現を求めている. この eliminate 函数では, 最初の引数と同じ型の対象, この例ではイデアルが返さる. それをイデアル i2 とする. それから基礎環を setring 命令でポインタを r2 に移し, 写像 f による c1 の像を plot

命令で描いた結果が図 1.3 である. ここで環 r1 ではなく環 r2 上で曲線を描いた理由は,surf.lib では環が 2 変数であり, 多項式が 3 変数でなければ曲線を描き, 環の変数, 或いは多項式の変数の何れかが 3 個であれば曲面を描く仕様となっている為である. この例の環 r1 上で描くと, 多項式は 2 変数であるものの, 環 r1 の変数が (x,y,t) の 3 変数になる為, 曲線ではなく曲面が描かれる. それを避ける為に,2 変数の環 r2 で描いている.

次に, 助変数表示された曲面の例として, 猿の腰掛 (図 1.3) の例を以下に示す.

```
> ring r3=0,(x,y,z,u,v),dp;
> ideal a1=x-u,y-v,z-u^3+3*u*v^2;
> ideal a2=eliminate(a1,uv);
> a2;
a2[1]=x3-3xy2-z
> plot(a2);
```

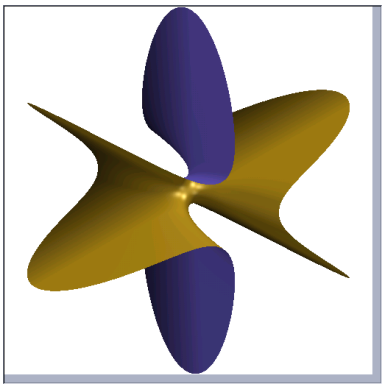

図 1.16: 猿の腰掛

この猿の腰掛では助変数に u,v の 2 変数を用いている. eliminate で削除する変数を複数指定する場合は単純に uv の様に並べれば良い. 又, この例で用いた環 r3 は x,y,z,u,v の 5 変数であるが, surf.lib では描く対象の多項式が 3 変数でありさえすれば曲面を描く. 又, この例では曲線も多項式が 2 変数であれば描かれてしまう.

以下に Singular で生成した猿の腰掛のスクリプトを以下に示す.

```
root_finder=d_chain_bisection;epsilon=0.0000000001;iterations=20000;
width=500; height=500; set_size; do_background=yes;
background_red=255; background_green=255; background_blue=255;
rot_x=0.14; rot_y=-0.3;
surface=
x^3-3*x*y^2-z
;
draw_surface;
```

このスクリプトの一行目には根を求める為のパラメータがある. ここでの値はデフォルトのものとは異っており, 曲面の自己交差や曲面の断面が上手く描ける様に厳し目の設定となっている. この個所は必要に応じて変更すべき個所でもある. 次の行で表示画面の大きさ (500 × 500 画素) が設定され,3 行目で背景色を定めている. 背景色や曲線の色等は RGB で 0 から 255 迄の整数で指定する.4 行目には X 軸と Y 軸の回転が記述されている. 尚, このスクリプトに含まれていないが,surf には Z 軸回りの回転の `rot_z` もある. それから, 変数 `surface` に引数の多項式が割当てられる. 最後の行の `draw_surface` で実際に図形が描かれる.

尚, 曲線の場合, 多項式は変数 `curve` に割当てられ,`draw_curve` で描かれる仕様となっている. 因に,`plot` 命令で引き渡された surf の命令群は描く対象が曲面であれば `surface` への割当の直前に, 曲線であれば `curve` への割当の直前に挿入される仕様である.

その為,surf で更に複雑な処理を行う事は,SINGULAR では出来ないものが多くある. 但し,SINGULAR で基本的な描画スクリプトを生成するので, このスクリプトを利用して更に高度な処理を行う事が容易になっている.

ここでは手始めに背景色を黒, 猿の腰掛を半透明にして, 平面による断面を surf を用いて表示してみよう. 背景色を黒にする方法は `background_red`,green, blue の値を全て 0 にすれば良い. 次に曲面の透明度は `illumination` と `transparence` の両方で指定する. 次に, 猿の腰掛を切断する平面の方程式は `plane` 変数に代入する. ここでは平面を $2*x+y-z-a$ とし, 変数 $a$ を-2.0 から 2.0 の範囲で動かし, 断面の色も $a$ の変化に従って変化する様にしたい. この場合は背景色と同様に曲線の色を指定する surf の変数 `curve_red`,green,blue の RGB$(0\cdots255)$ を変化させれば良い. 最後に `cut_with_plane` で `plane` 変数に入れた平面による曲面の断面が描かれる.

上記の事に留意して修正したスクリプトを以下に示す.

```
root_finder=d_chain_bisection;epsilon=0.0000000001;iterations=20000;
width=500; height=500; set_size; do_background=yes;
background_red=0; background_green=0; background_blue=0;
illumination=ambient_light+
diffuse_light+reflected_light
+transmitted_light;
transparence=50;
clear_screen;
rot_x=0.1; rot_y=-0.7;rot_z=0;
surface=
x^3-3*x*y^2-z
;
draw_surface;
double a=-2.0;
int i=0;
loop:
plane=2*x+y-z-a;
curve_red=255-5*i;
curve_green=10*i;
curve_blue=0;
cut_with_plane;
a=a+0.5;
i=i+5;
if (a<=2.0) goto loop;
```

このスクリプトで描かれた曲線を図 1.17 に示す.

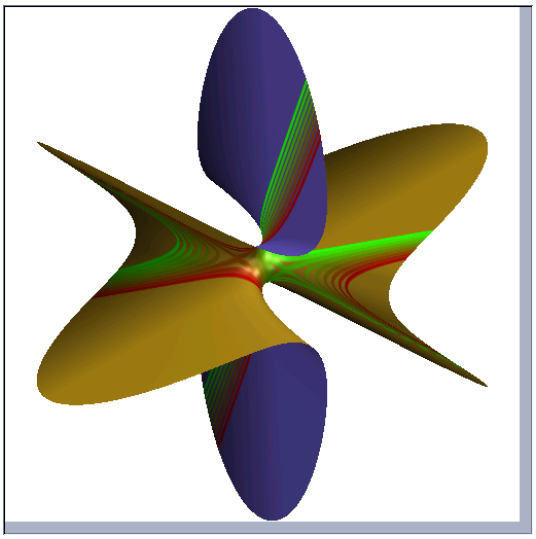

図 1.17: 猿の腰掛 (2)

## 1.4 SINGULAR で曲線の特異点を求める

### 1.4.1 臨界点と特異点

(導入を書く予定)
多項式 $f$ の臨界点と特異点は以下で定義される.

- 定義 1:多項式 $f$ の臨界点は, その偏微分が全て 0 になる点である.
- 定義 2:多項式 $f$ の特異点は, その偏微分が全て 0 になる $V((f))$ の点である.

### 1.4.2 計算に必要なもの

多項式の微分を行う SINGULAR の命令に `diff` がある. この `diff` 命令は次の書式を持つ.

```
diff(多項式, 環の変数)
```

根基イデアルは SINGULAR の `radical` 命令を使えば計算可能である. 因に,SINGULAR のオンラインマニュアルは便利なもので, ?radical; と入力すると,`radical` の書式だけでは無く, 探している命令が何処のライブラリに含まれているかも表示する. オンラインマニュアルから,`radical` 命令が `primdec.lib` に含まれている事が判る. 尚,`radical` 命令を使いたければ, LIB "primdec.lib" で予め読込んでおく事.

(radical を使った例を書く)

### 1.4.3 デカルトの葉状曲線の特異点

最初にデカルトの葉状曲線 $f = x^3 - y^3 - 3xy$ で遊ぼう. ここで, 曲線の特異点は, 曲線を表現する多項式 $f$ と, 環変数 x,y による偏微分 $\frac{\partial f}{\partial \mathrm{x}}$ と $\frac{\partial f}{\partial \mathrm{y}}$ との共通零点を求めれば良い.

これを SINGULAR で行う場合, 多項式 f とその偏微分 $f_x$ と $f_y$ で生成されるイデアル $I$ を宣言し, そのイデアル $I$ の根基イデアル $\sqrt{(I)}$ の零点集合が求める特異点の集合となる. こ, 最初にライブラリを読込んでおく. 尚,product 命令も使いたいので,general.lib も読込み, 最後にグラフ表示の為に surf.lib も読込んでおく.

では, デカルトの葉状曲線の特異点を SINGULAR で求めよう.

```
>ring R=0,(x,y),dp;
> poly f=x3-y3-3xy;
> ideal I=f,diff(f,x),diff(f,y);
>I;
I[1]=x3-y3-3xy
I[2]=3x2-3y
I[3]=-3y2-3x
> plot(product(I));
>ideal VI=radical(I);
> VI;
VI[1]=y
VI[2]=x
> ideal VS=f,VI;
> plot(product(VS));
>
```

この例では, 最初にライブラリを読込んでおき, それから環 r を定義する. この環上で $f = x^3 - y^3 - 3xy$ とその偏微分で生成されるイデアル I を定義している. ここでは poly f=x3-y3-3xy; で積と冪記号を省略して宣言している. イデアル I を生成する多項式の零点集合を surf で一度に表示する為, イデアルの生成元の積を計算する product を用いて一つの多項式を計算し, それを plot 命令で表示したものが図 1.18 に示すグラフである.

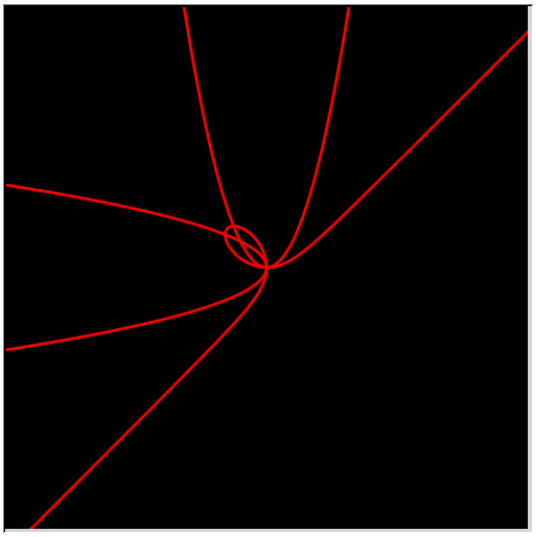

図 1.18: 特異点 1

このイデアル I の零点集合は, 図 1.18 に示す曲線が全て交差する点である. 但し, このままでは判り難い.

有理数係数の多項式環では, イデアルの零点集合はその根基イデアルの零点集合に等しい (Hilbert の零点定理) 為, 次に `radical(I)` でイデアル I の根基を求めている. ここで求めた根基イデアル `VI` は x,y で生成される. このイデアル `VI` と多項式 `f` を合せたイデアル `VS` を定義し,`product` で全ての生成元の積を取って `plot` で描いたものが図 1.19 である. この図からも原点 (0,0) が曲線 $f$ の特異点となっている事が読み取れる.

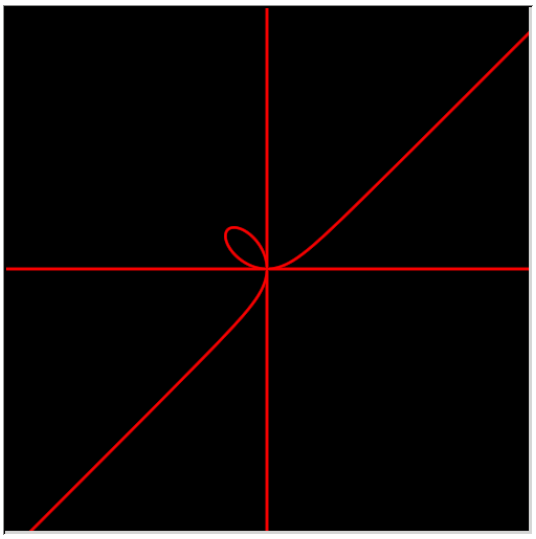

図 1.19: 特異点 2

(Macaulay2 で同じ事をさせてみよう! )

## 1.5 SINGULAR でライブラリを書く

### 1.5.1 言語仕様

SINGULAR は surf と比べて非常に大きなシステムである. その為, 命令全ての概要や処理言語の詳細について, 簡易に述べるのは難しい事である.

その為, ここでは前節で利用した `surf.lib` の内容について解説すると共に, プログラミング言語としての SINGULAR について簡単に紹介する.

尚,SINGULAR の構文は C と非常に似ている. その為, 多少なりとも C の知識があれば, ライブラリを読む事に苦労する事は, 内部で用いられている SINGULAR 独自の関数以外では殆ど無いだろう.

(後, 構文やライブラリの話を書く)

### 1.5.2 surf.lib

surf.lib は 2 個の手続で構成されたライブラリファイルである. 一つはイデアルを引数とし, 整数値ベクトルを返す num_of_vars で, もう一つはイデアルとリストを引数に取る plot である.

先ず最初に num_of_vars の内容を示す.

```
static proc num_of_vars(ideal I)
{
  intvec v;
  int i;
  poly p;
  for(i=size(I);i>0;i--)
  {
    p=I[i];
    while(p!=0)
    {
      v=v+leadexp(p);
      p=p-lead(p);
    }
  }
  return(v);
}
```

この例で判る様に,SINGULAR の手続は prco 関数名（型 引数,..) で開始する. SINGULAR の処理言語には for 文や while 文があり, 全体的に C と同様の構文である.

この手続では与えられたイデアル I に対し,size(I) でイデアルを生成する多項式の数を求め, 各生成多項式から, 次数のリストを取り出して, その総和 v を返している. 次数リストは leadexp 命令で求めており, この次数リストを次数に持つ単項式が lead(p) である.

次に, 実際の描画処理を行う plot 命令について解説する. 説明の為に, ここでは内容毎に分けて示す. 最初に plot の頭の初期設定を行う個所を以下に示す.

```
proc  plot(ideal I,list #)
"USAGE:   plot(I);  I ideal or poly
ASSUME: I defines a plane curve or a surface given by one equation
RETURN: nothing
NOTE: requires the external program 'surf' to be installed
EXAMPLE: example plot; shows an example
"
{
  string extra_surf_opts=" -x "; // remove this line for surf 0.9
  string l="/tmp/surf"+string(system("pid"));
  string err_mes; // string containing error messages
  def base=basering;
  intvec v=num_of_vars(I);
  int i,j,n;
  for(i=size(v);i>0;i--)
  {
    if (v[i]!=0) { n++; }
  }
  if (n==0 or n>3)
  {
    err_mes="Cannot plot equations with "+string(n)+" variables";
    ERROR(err_mes);
  }
```

ここでは最初に plot 命令の USAGE が記述されている. 本文で extra_surf_opts は surf にファイルを与えて起動する場合に用いるオプションである. 次の l には surf.lib で生成する surf のスクリプトファイル名がフルパスで指定されている. 又,base に basering を割当てる. 尚,def は型が不定の宣言である. これは基礎環が環 (ring) や商環 (qring) の可能性がある為である. 次の err_mes はエラーメッセージを格納する対象である.

v には先程説明した num_of_vars を用いて引数のイデアルを生成する多項式の次数ベクトルが割当てられる. それから for 文で次数ベクトル v で 1 以上の個所の総数を n に割当て, それが零や 3 より大であればエラーを返して終了する.

次の段では, 与えられたイデアル (多項式) を surf で描ける多項式に変換する為の工夫をしている個所を示す.

```
ring r=0,(x,y,z),dp;
short=0;
map phi=base,0;
j=1;
for(i=1;i<=size(v);i++)
{
  if (v[i]!=0)
  {
    phi[i]=var(j);
    j++;
    if(j==4) break;
  }
}
ideal I=simplify(phi(I),2);
```

最初に x,y,z の 3 変数の環 `r` を宣言し, それから写像 $phi : r \to base$ を宣言している. 但し, ここでは単に零写像を宣言しているだけで, 実際の写像の定義は `for` 文内部で行っている. この写像は, 与えられたイデアルが定義されている `base` の変数の順序に従い, 環 `r` の変数 x,y,z に対応付けるものである. こうする事で, 基礎環で定義された多項式は x,y,z の多項式に変換される.

この様な環と写像を宣言した理由は,surf で扱える多項式が x,y,z の多項式に限定される為である. この処理を実施している御陰で,`surf.lib` で描く多項式の変数は自由に設定可能となる所以である.

これで, 曲線や曲面を描く上で必要な環の定義を行った. 今度は与えられた多項式と基礎環の性質に従って, 曲線か曲面を描く為のスクリプトを生成する個所に移る.

先ず, 曲線を描くスクリプトを生成する個所がくる.

```
  if (leadcoef(I[1]) <0) { I[1]=-I[1]; }
  if (ncols(I)==1 and n<=2 and nvars(base)!=3) // curve
  {
    write(":w "+l,"clip=none;");
      write(l, "width=500; height=500; set_size; do_background=yes;
background_red=255; background_green=255; background_blue=255;");
    write(l,
    "root_finder=d_chain_bisection;epsilon=0.0000000001;iterations=20000;");
    write(l, "curve_green=0; curve_blue=0; curve_width=1.5;");
    if (size(#)>0)
    {
      write(l,#[1]);
    }
    write(l,"curve=",I[1],";");
    write(l,"draw_curve;");
  }
```

surf.lib の制約事項として, 与えられたイデアルが単項イデアルでなければならない. この事は ncols(I)==1 で生成元が一つである事を検証している.

それから, 曲線を描く為には, 単項イデアル I の生成多項式の変数の個数 n が 2 以下で, 単項イデアル I を定義した環の変数が 3 でなければ, 曲線を描く為のスクリプトを生成する.

最初の write 文で, ファイル名 l の先頭に:w を付けて, ファイルを書込み用に開き, それから clip=none; 書込む. 尚, 一度開いたファイルに対しては, ファイル名の頭に w:を追加する必要は無い. 次の write 文で, 描画ウィンドウの大きさの設定と背景色 (白) の設定を行う. それから, 根を求める為の設定を書込み, 最後に曲線の色 (赤) と太さを設定し, surf に引き渡すオプションがあればオプションリストの第一成分を書込み, 最後に write(l,"draw_curve;"); で曲線を描く命令を追加する.

次の曲面を描くスクリプトも曲線の場合と同様の構成となっている. 但し, 曲面を描くのは, イデアルが単項イデアルで, その生成多項式の変数が 3 個か, 基礎環の変数が 3 個の何れかでありさえすれば良い. その為, 基礎環の変数が 3 個の場合や, 生成多項式が 3 変数で, 基礎環が 3 個以上の変数を持っている場合でも, 曲面が描かれる.

```
  else
  {
    if (ncols(I)==1 and (n==3 or nvars(base)==3)) // surface
    {
      write(":w "+l,
      "root_finder=d_chain_bisection;epsilon=0.0000000001;iterations=20000;");
     write(l, "width=500; height=500; set_size; do_background=yes;
background_red=255; background_green=255; background_blue=255;");
      write(l,"rot_x=0.14; rot_y=-0.3;");
     if (size(#)>0)
       {
         write(l,#[1]);
       }
      write(l,"surface=",I[1],";");
      write(l,"draw_surface;");
    }
    else
    {
      err_mes="cannot plot "+string(ncols(I))+" equations in "
             +string(n)+" variables";
      ERROR(err_mes);
    }
  }
```

最後に, 生成したスクリプトを surf に引き渡して surf を起動する個所に移る.

```
  string surf_call;
  surf_call = "surf ";
  if (defined(extra_surf_opts))
  {
    surf_call = surf_call + extra_surf_opts + " ";
  }
  surf_call =surf_call+ l +" >/dev/null 2>&1";

  if("ppcMac-darwin"!=system("uname")){
     i=system("sh",surf_call);
  } else {
    surf_call= surf_call + " || "+ "singularsurf " +
      extra_surf_opts + " " +l +" >/dev/null 2>&1";
      i=system("sh",surf_call);
  }

  if (i!=0)
  {
    err_mes = "calling ‘surf‘ failed. (the shell return the error code "
          +string(i)+")."+newline+
    "probably the executable ‘surf‘ is not found.";
    ERROR(err_mes);
  }
  i=system("sh","/bin/rm "+l);
}
```

先ず、最初に文字列surf_callを宣言し, 文字列surf を割当てている. 次に,definedでextra_surf_optsが定義されていれば, 文字列surf_callにオプション ( -x) を追加する. 最後にファイル名とエラー処理の為の>/dev/null 2>&1 を追加している.

それからsystem命令を用いて, シェルにsurf_callを引き渡す事でsurfの立ち上げて描画を実行させている.

その後はエラー処理となる.

surf.libを利用して自分のライブラリを作ってみよう. 手順としては,surf.libを別名の複製を作る. SINGULARを起動し,LIB命令で読込む.

### 1.5.3 surf.libを改造してみる

(カレントディレクトリにsurf.libを別名でコピーし, 改造して遊ぶ話を書く)

SINGULAR のオプション
(不要か？)

```
 --emacs=EMACS          Singular を起動する為の Emacs プログラムとして EMACS を利用 (ESingular
のみ)
 --emacs-dir=DIR        emacs lisp ファイルを検索するディレクトリとして DIR を利用 (ESingular
のみ)
 --emacs-load=FILE      emacs を起動する際に、デフォルトの代りに読込む FILE(ESingular
のみ)
 --singular=PROG        emacs 内部で起動させる PROG(ESingular のみ)
 --no-call              プログラムを起動させない. どうの呼出を表示 (ESingular のみ)
 -b --batch                MP バッチモードで動作
 -c --execute=STRING       起動時に STRING を実行
 -d --sdb                  ソースコードデバッガを可能にする (実験的)
 -e --echo[=VAL]           変数'echo' に (整数値)VAL を設定する
 -h --help                 ヘルプメッセージを表示して終了する
 -q --quiet                起動時のバナーやライブラリ読込メッセージ表示を行わない
 -r --random=SEED          整数 (整数値)SEED による Seed random generator
 -t --no-tty               端末文字の再定義を行わない
 -u --user-option=STRING   Return STRING on `system("--user-option")'
 -v --version              バージョンと構成情報を表示する。
 --allow-net            ネットからヘルプページ (html) を取り込む事を許可する
 --browser=BROWSER      ヘルプを BROWSER([x,tk]info,netscape) で表示する
 --emacs                emacs 内部で動作させる場合のデフォルトを設定 (Singular のみ)
 --no-stdlib            'standard.lib' を起動時に読込まない
 --no-rc                起動時に`.singularrc' ファイルを実行しない
 --no-warn              警告を表示しない
 --no-out               全ての出力を抑制する
 --min-time=SECS        SECS(秒) よりも小さな時間の表示を行わない
 --MPport=PORT          MP 接続の為に PORT 番号を用いる
 --MPhost=HOST          MP 接続の為の HOST
 --MPrsh=RSH            MP 接続の為に RSH を利用
 --ticks-per-sec=TICKS 1 秒あたりの TICKS に対し、時間単位を設定
```