

Macaulay2の紹介

横田博史

平成22年1月27日(水)

まえがき

この文書は Macaulay2 の入門用に書いたものであり, Macaulay2 の非常に限定された範囲の話題のみを扱っている. Macaulay2 や Macaulay2 を使った実例は Schenk の本 [4] や Eisenbud 等の本 [2] を参照されたい.

ここで, Schenk の本は Macaulay2 を用いて代数学を学ぶ事を目的とした本である為, 数学のお勉強に重点が置かれているが, Eisenbud 等の本は Macaulay2 を使って代数や代数幾何を料理する事を目的とした本である為, Macaulay2 の言語や使い方に重点が置かれている. この傾向の違いを知った上で参照されると良いだろう.

この Macaulay2 では Singular のライブラリが利用されている. その為, Singular と同様に基礎環を構築して処理を行うといった似た面も持っている. この Singular に関しては, Greuel-Pfister の本 [3] を参照されると良い. こちらは Singular を使った本格的な可換環の教科書である.

平成 22 年 1 月 27 日 (水)

横田博史

目次

第 1 章	Macaulay2 の簡単な紹介	1
1.1	Macaulay2 の概要	1
1.2	Macaulay2 の起動, 入力と終了	2
1.3	オンラインマニュアル	3
1.4	その他の支援機能:	4
1.5	基本的な与件の型について	4
1.5.1	数	4
1.5.2	基礎的な算術演算子:	6
1.5.3	数値関数	8
1.5.4	真理値と論理式	9
1.5.5	表象と文字列	11
1.5.6	文字列操作の演算子	12
1.5.7	列, リスト, 配列と行列の定義	12
1.5.8	列, リストや配列の長さを求める演算子	14
1.5.9	行列の定義	16
1.5.10	行列の結合に関連する演算子	16
1.5.11	行列の成分取出に関連する演算子	17
1.5.12	リストと行列の算術演算	20
1.5.13	多項式環の定義	22
1.6	文	27
1.6.1	制御文	27
1.7	関数	29
1.8	M2 ファイル	30
1.9	ファイル出力	31
第 2 章	特異点遊戯	32
2.1	Macaulay2 による特異点の計算	32

第1章 Macaulay2の簡単な紹介

1.1 Macaulay2の概要

Macaulay2は可換環向けの数式処理システムである。Macaulay2はSingularの幾つかのライブラリを用いてる為にSingularと似た点も多くあるが、その一方で、Mathematica風の構文も見受けられ、Singularとは全く別の面白味のある数式処理システムとなっている。

オブジェクト指向の数式処理システム: Macaulay2はSingularと同様にオブジェクト指向の数式処理システムである。この事は、Macaulay2上の演算子や関数は対象に束縛されたメソッドであり、何かの処理を行う為にはその対象を最初に行わなければならない事を意味する。更に、同名の演算子や関数であっても対象が違えば処理内容や結果も異なる事がある。例えば、多項式やイデアルといった対象の計算を行う場合、その親にあたる環(基礎環:Base ring)を先ず生成しなければならず、対象が生成された場合でも、たとえば、積演算を考えると、その対象がスカラであるか行列であるかによって行列の積としての処理を行うか、成分単位の積となるかといった処理が異なる事を意味する。

フロントエンド: Macaulay2は固有のフロントエンドを持たない。個々の結果はその都度ファイルに残す事も可能だが、MathematicaやMapleの様に終了時にセッションをそのままファイルに簡単に残せないのが初心者にとっては難点であろう。その為、フロントエンドとして利用可能なEmacsやTeXmacsと併用した方が良い。特に、TeXmacsを利用すれば、レゾリューション等の図式や数式が美しくレンダリングされて表示されるので楽しい。

また、仮想端末でも0.9.8以降からGNU Emacs風の行編集と履歴の利用が可能となっており、オンラインマニュアルもhelp関数以外にTexinfoを用いるinfoHelp関数やMozilla等のHTMLブラウザを用いるviewHelp関数の二つの関数が用意され、使い勝手がより向上している。特に、viewHelp関数によって表示されるHTMLブラウザから、初心者向けの文書「gettimh started」やMacaulay2のマニュアルや例題といった文書を閲覧することが容易に行える。

動作環境: Macaulay2は基本的にUNIX環境で動作する。Macaulay2の実行ファイルが存在する環境は、FreeBSD, GNU Linux, MacOSXとMS-Windows

であるが、MS-Windows は Cygwin 環境で動作させることになる。その為、Cygwin 環境を整える必要があるが、個人的には、Cygwin で強引に UNIX 環境を構築するよりも、VMware Player や Sun の VirtualBox といった仮想計算機環境で KNOPPIX/Math 上の Macaulay2 を利用する事を、より簡便で充実した環境が得られる方法として提案しておく。

Macaulay2 に関する文献: Macaulay2 の日本語の文献は殆んどないのが現状で、この小冊子を除くと唯一、Macaulay2 の作者の一人である Dan Grayson へのインタビューが「数学のたのしみ No.11, 多項式環の視点: グレブナー基底」(日本評論社,1999 年)の「グレブナーエンジンのプログラマ達」に掲載されている程度である。

その一方で、英語の書籍としては Eisenbud, Grayson, Stillman と Sturmfels の本 [2] と Schenck の本 [4] の二つがある。

最初の Eisenbud 等の本には Macaulay2 の入門から応用迄が収録されているが、この本はどちらかと言えば、こちらは代数学や代数幾何に詳しい方が Macaulay2 の使い方を調べるのに適した本である。

次の Schenck の本の方は、Macaulay2 を用いた代数幾何学と代数的位相幾何学の入門書であり、これらの事項が手堅く纏められた程良い厚さの良い教科書である。この本は Macaulay2 で一寸した計算をしながら、代数学のお勉強をしたい方に向いている。

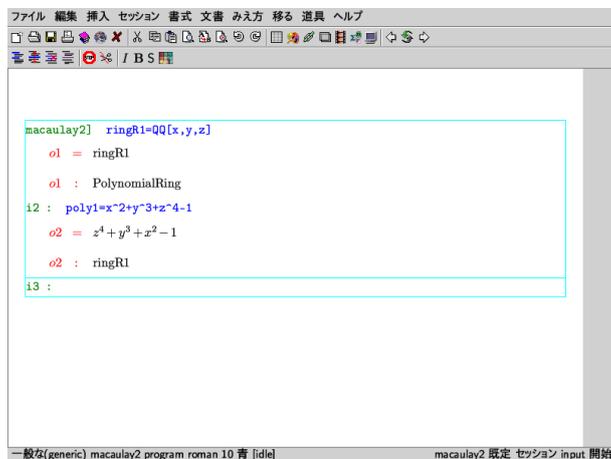
この冊子は上述の英語の書籍の様な本格的なものではなく、Macaulay2 の持つ独特の雰囲気を紹介する事を目的としている。従って、Macaulay2 や関連する数学の詳細は上述の本、或いは、Macaulay2 に付属のオンラインマニュアルを参照されたい。

1.2 Macaulay2 の起動, 入力と終了

起動方法: パスが通っていれば xterm 等の仮想端末に M2 と入力することで Macaulay2 が立ち上がる。KNOPPIX/Math を使っていれば、メニューバーの左から二番目にある \sqrt{x} メニューから Macaulay2 を選択すれば良い。猶、M2 は前述の様に GNU Emacs 風の編集機能を持つてはいるものの、履歴を文書で残す機能を持たない為に、GNU Emacs や TeXmacs 等と云った Macaulay2 の履歴をきちんとした文書で残すことの出来る外部プログラムと併用する事を勧める。

TeXmacs をフロントエンドとして使う場合: TeXmacs から Macaulay2 を利用する場合、最初に TeXmacs を立ち上げ、TeXmacs の「挿入」(insert) から「セッション」(session) を選ぶと TeXmacs で利用可能なアプリケーションの一覧が表示されるので、その中から Macaulay2 を選択すると良い。ここで、

図 1.1 に TeXmacs を Macaulay2 のフロントエンドとして利用した時の処理の様子を示している:



```
macaulay2] ringR1=QQ[x,y,z]
o1 = ringR1
o1 : PolynomialRing
i2 : poly1=x^2+y^3+z^4-1
o2 = z^4+y^3+x^2-1
o2 : ringR1
i3 :
```

図 1.1: TeXmacs 上で利用

プロンプトについて: Macaulay2 を起動すると入力を促すプロンプトが表示される. 入力プロンプトは “i1 : ” の様に入力行である事を示す文字 “i” と入力番号で構成された文字列, 出力のプロンプトも “o1 : ” の様に出力行である事を示す文字 “o” と入力番号で構成された文字列である.

猶, Macaulay2 では入力行末尾にセミコロン “;” を入れると入力に対する出力の表示が無効になる. それ以外では入力に対する値や与件型の表示を行う.

Macaulay2 の終了: 仮想端末上で Macaulay2 を起動した場合, `quit` と入力すれば Macaulay2 を終了する.

1.3 オンラインマニュアル

Macaulay2 は詳細なオンラインマニュアルを標準で持ち, 文書を読む為の関数として, `help` 関数, `infoHelp` 関数と `viewHelp` 関数がある. これらの関数の引数は全て同じで, 1 つの文字列を引数として取り, 表示される文書の形式が text 形式, texinfo 形式か HTML 形式と云った違いがある程度である. しかし, 使い勝手から言えば, `viewHelp` 関数を使う事が最も良いだろう.

help 関数: Text 形式の文書を標準出力に表示する関数. `help` とだけ入力すると, `help` 関数の概要が表示されるが, 一気に内容を表示する為に `xterm` 等

の仮想端末では、文書全体がバッファに収まり切らずに文書の頭が読めなくなる事がある。

infoHelp 函数: フロントエンドとして Texinfo を用いる函数である。文書は texinfo を用いてそのまま表示される為に、Macaulay2 による処理が中断してしまう短所がある。

viewHelp 函数: Firefox 等の HTML ブラウザを立ち上げて文書を表示する函数である。一旦、HTML ブラウザを立ち上げると、HTML ブラウザは Macaulay2 の制御を離れるので、Macaulay2 とは独立して利用する事が可能である。さて、`viewHelp` とすると HTML ブラウザが起動され、Macaulay2 とそのパッケージに関するオンラインマニュアルの見出しが表示される。次に `viewHelp "help"` と入力すると、今度は Macaulay2 の help 函数のマニュアルを表示する。これで help 函数の使い方が判るだろう。この様に `viewHelp "<事項>"` で調べたい<事項>に関連する文書が表示される。

1.4 その他の支援機能:

オンラインマニュアルの他に、Macaulay2 では文字列の一部から対応する函数名のリストを出力する `apropos` 函数、函数の例題を実行する `example` 函数等がある。

1.5 基本的な与件の型について

Macaulay2 は Singular と同様のオブジェクト指向の数式処理である為に、最初に環 (=基礎環) を定義し、それから、必要とする基礎環上の対象を構築して行く。つまり、最初に対象を生成しないことにはメソッドが使えないが、その対象を生成する為には対象を包含する環を定義しなければならない。但し、表象、列、文字列や数値 (整数、有理数、実数)、そして true や false で表現される真偽値と云った基本的な与件、及び、これらの与件で構成されたリスト、配列、行列については処理が行える仕様となっている。

1.5.1 数

Macaulay2 で利用者が基礎環を定義せずに扱える数として、整数、有理数と実数がある。ここで実数は Macaulay2 内部では浮動小数点数として表現されている。

Macaulay2 では整数環 \mathbb{Z} 、有理数環 \mathbb{Q} と実数環 \mathbb{R} は最初から Macaulay2 に組込まれた環であり、これらの環を利用者が定義する必要はない。従って、こ

これらの環に含まれる対象, 則ち, 数は, そのまま入力可能であり, 四則演算を含む算術的演算も可能である. 猶, Macaulay2 では整数環 \mathbb{Z} を ZZ, 有理数環 \mathbb{Q} を QQ, そして, 実数環 \mathbb{R} を RR と表記している.

ここで, 数の入力と計算例を示しておこう:

i1 : 1

o1 = 1

i2 : 1+2

o2 = 3

i3 : 2*3

o3 = 6

i4 : 3/5

o4 = $-\frac{3}{5}$

o4 : QQ

i5 : 2.9

o5 : 2.9

o5 : RR (of precision 53)

この例で示す様に整数の計算の場合は結果だけが表示されているが, 入力がある有理数と浮動小数点小数になれば, その計算で用いた与件型 (この場合は属する環) と「浮動小数点数の精度」を計算結果に続けて表示する. ここで浮動小数点数は符号部, 指数部と仮数部の三種類の固定長の 2 進数から構成され, 仮数部の 2 進数の bit 長に 1 を加えたものが「浮動小数点数の精度」と呼ばれる. Macaulay2 では既定値として, 53 (“of precision 53”) となっているが, このことから, 浮動小数点数の仮数部が 52 bit 長, 則ち, 倍精度であることが判る.

1.5.2 基礎的な算術演算子:

整数, 有理数や実数に対する基礎的な算術演算子として, 和, 差, 積, 商の四則演算子, 更に, 剰余, 冪乗や階乗がある. 四則演算子や冪の演算子は \mathbb{C} と同様の演算子を用いる事が可能であり, これらの演算子は利用者が生成した環に継承される:

基本的な算術演算子

演算子	例	概要
+	$a + b$	a と b の和
-	$a - b$	a と b の差
*	$a * b$	a と b の積
/	a / b	a の b による商
//	$a // b$	a の b による商
%	$a \% b$	剰余 $a \bmod b$
^	$a ^ b$	冪乗 a^b
!	$a !$	階乗 $a!$

i4 : $1+2*3-4^2/5$

o4 = $-\frac{19}{5}$

o4 : QQ

i5 : $1+0*1.9-2/5$

o5 = 0.6

o5 : RR

この結果は一見すると Maple の様な数式処理システムに似ているが, Macaulay2 はオブジェクト指向である為に, 基礎環が背後にある. 実際, $2/5$ は有理数環 QQ の元, 0.6 が実数環 RR の元となっている事が表示されている.

又, 対象が異なれば同名のメソッドも違う働きをする. この事を演算子/と演算子//の違いで把握しておこう. まず, 整数環 ZZ 上では, 演算子/が有理数を生成する演算子, 演算子//が整数の商を返却する演算子である:

i1 : $4/2$

o1 = 2

o1 : QQ

i2 : 5/2

$$o2 = -\frac{5}{2}$$

o2 : QQ

i3 : 5//2

o3 = 2

この様に、整数を被演算子とする場合には、これらの演算子の結果は異なっている事に注意されたい。しかし、有理数環上と実数環上の対象に対し、これらの演算子は同じ結果を返す演算子となる:

i3 : 5/2 // 2/5

$$o3 = -\frac{1}{4}$$

o3 : QQ

i4 : 5/2 / 2/5

$$o4 = -\frac{1}{4}$$

o4 : QQ

i5 : 5./2 // 2/5.

o5 = .25

o5 : RR (of precision 53)

i6 : 5./2 / 2/5.

o6 = .25

o6 : RR (of precision 53)

また、演算の順序は通常演算子の順序に依るが、ここで、括弧 () を用いることで、括弧内の演算を優先させる事が出来る点は C 等の言語と同様である。逆に括弧 () を用いなければ通常演算子の優先順位で処理される。

1.5.3 数値関数

Macaulay2 には \cos 等の三角関数, \exp や \log といった指数関数や対数関数が組込まれている。これらの関数は Maple 等の関数とは異なり、実数環 \mathbb{R} 上の関数である:

i20 : $\cos(\pi/2)$

o20 = 6.12303×10^{-17}

o20 : RR

i21 : $\exp(1)$

o21 = 2.71828

o21 : RR

i22 : $\log(1)$

o22 = 0.

o22 : RR

この例に示す様に、 $\cos(\pi/2)$ と入力しても直ちに評価されて浮動小数点数が結果として返却されている。更に、これらの数値関数は一般の環上に使える様に拡張されていない為、例えば、 x を変数として含む多項式環 $K[x]$ を定義しても、 $\sin(x)$ は変数 x に数値が割当てられていない為に誤りとなる。同様に、多項式の微分を行う diff 関数の引数として数値関数を引渡しても、 \sin 関数の引数が数値でない事に加え、式 $\sin(x)$ 自体が多項式でない為に誤りになる。

1.5.4 真理値と論理式

Macaulay2 には真理値と基本的な論理式の演算子が定義されており、利用者が構築した環に継承される。

真理値: Macaulay2 に於ける真偽値は true と false がある。

論理積, 論理和と否定の演算子: 真理値に対して基本的な演算を行う演算子として, 論理積, 論理和と否定がある:

論理演算子		
演算子	構文	概要
and	a and b	論理式 a と論理式 b の論理積
or	a or b	論理式 a と論理式 b の論理和
not	not(a)	論理式 a の否定
	not a	論理式 a の否定 (小括弧を外した表記)

二項演算子 and: 被演算子が全て true であれば true, その他は false を返却。

二項演算子 or: 被演算子が全て false であれば false, その他は true を返却。

単項演算子 not: 被演算子が true なら false, false なら true を返却。

ここで論理式 a and b に含まれる論理式 a が偽の場合, 論理式 b の評価を演算子 and は行わない。同様に演算子 or も a or b を構成する論理式 a が真の場合, 論理式 b の評価を行わない仕様となっている。その意味では, 演算子 and と演算子 or の定義は二項演算子を Curry 化したものになっており, この様子を関数型言語の Haskell([1] 参照) で定義すれば,

```
1 and' :: Bool -> Bool -> Bool
2 and' False x = False
3 and' True x = x
4 or' :: Bool -> Bool -> Bool
5 or' True x = True
6 or' False x = x
```

となるだろう。つまり, and' を例にすると, 第 1 引数が False なら第 2 引数とは無関係に False であり, 第 1 引数が True であれば, 第 2 引数の値となるという定義である。

論理式: Macaulay2 での論理式とは, 評価されることで真理値 {true,false} をその値として持つ式であり, 論理式を論理演算子 and, or, not で結合した対象も論理式になる. 論理式の例としては, 'not true' や 'true and false' といった式が挙げられる. ここで, 論理式を対象から生成する演算子として, 「比較の演算子」という二項演算子がある.

比較の演算子: 2つの被演算子の関係を真理値で表現する演算子であり, Macaulay2では, 被演算子の間に置く中置表現の演算子である. ここでMacaulay2の持つ比較の演算子を次に纏めておこう:

比較の演算子

演算子	構文	真となる条件
==	a==b	a と b が等しいければ真
>=	a>=b	a が b 以上であれば真
>	a>b	a が b より大であれば真
<=	a<=b	a が b 以下であれば真
<	a<b	a が b より小であれば真

i21 : 3==2

o21 = false

i22 : 3>=2

o22 = true

i23 : 3<4

o23 = true

比較の演算子を返す演算子: 演算子?は, `a ? b` とする事で a と b を比較した場合に真を返す比較の演算子を返却する:

i26 : 3 ? 3

o26 = ==

o26 : Keyword

i27 : 3 ? 5

o27 = <

o27 : Keyword

1.5.5 表象と文字列

数の他に Macaulay2 では「表象」(Symbol) と「文字列」(String) が扱える.

表象: アルファベットと数文字と一部の記号で構成され, 必ずアルファベットで開始する文字の羅列である. 但し, Macaulay2 の処理言語で意味を持つ for 等の文字の羅列は表象にならない. この表象は Macaulay2 で変数や関数名として利用出来る.

文字列: 二重引用符”で括られた文字の列である. 文字列の場合,”以外に用いてはならない文字はなく, 表象とは全くの別物である. 猶, 二重引用符”を文字列中に含める場合は \ を用いる. 例えば, “\” の様に表記する.

次に表象と文字列の例を示す.

i20 : x

o20 = x

o20 : Symbol

i21 : XYZ

o21 = XYZ

o21 : Symbol

i22 : w1 = "mike"

o22 = mike

i23 : w2 = "neko"

o23 = neko

この例で示す様に, 変数に値を割当てる場合, 演算子= を用いる.

1.5.6 文字列操作の演算子

文字列に対しては二つの演算子 `|` と `||` がある。共に文字列の結合を行って新しい文字列を生成する演算子であるが、改行コードが入るか入らないかの違いがある。これらの演算子 `|` と演算子 `||` の違いを次に示しておく：

```
i14 : "今日はとても"| "良い天気"
```

```
o14 = 今日はとても良い天気
```

```
i15 : "今日はとても"|| "良い天気"
```

```
o15 = 今日はとても  
      良い天気
```

```
i16 : "今日はとても"|"|" "良い天気"
```

```
o16 = 今日はとても
```

```
      良い天気
```

```
i17 : "\"三毛" | "猫\""
```

```
o17 = "三毛猫"
```

この例で示す様に、演算子 `|` の場合は単純に文字列を結合するだけだが、演算子 `||` の場合は二つの文字列を結合するだけでなく、文字列の間に改行文字が挿入される。これらの演算子は結果の表示等で利用すると良いだろう。

又、前述の様に文字列中に二重引用符を入れたければ、二重引用符を `"\"` で置換えると良い

1.5.7 列, リスト, 配列と行列の定義

基本与件型を持った対象から構成された対象に、列 (Sequence) , リスト (List), 配列 (Array) , 行列 (Matrix) がある。先ず、列, リストと配列の定義方法を以下に纏めておく。

列, リスト, 配列

対象	例	概要
列	(a_1, \dots, a_n)	対象をコンマ “,” で区切って小括弧で括った事で得られる対象
リスト	a_1, \dots, a_n	列を中括弧 { } で括った事で得られる対象
配列	$[a_1, \dots, a_n]$	列を大括弧 [] で括った事で得られる対象

次に列, リスト, 配列と行列の例を示す:

i27 : (1,2,3)

o27 = (1, 2, 3)

o27 : Sequence

i28 : 1..5

o28 = (1, 2, 3, 4, 5)

o28 : Sequence

i29 : (1,2,4,5/7)

$$o29 = \begin{pmatrix} 1 & 2.4 & - \\ & & 5 \\ & & 7 \end{pmatrix}$$

o29 : Sequence

i30 : {1,2,3,5,4}

o30 = {1, 2, 3, 5, 4}

o30 : List

i31 : [1,2,3,4,5]

o31 = [1, 2, 3, 4, 5]

o31 : Array

o50 : Sequence

i51 : a#9

o51 = 10

i52 : b=[10,11,12,13,14,15]

o52 = [10, 11, 12, 13, 14, 15]

o52 : Array

i53 : b#2

o53 = 12

i54 : c={1,2,3,5}

o54 = {1, 2, 3, 5}

o54 : List

i55 : c#2

o55 = 3

i56 : #a

o56 = 10

i57 : #b

o57 = 6

i58 : #c

o58 = 4

1.5.9 行列の定義

行列は関数 `matrix` を用いて定義する:

```
i32 : A=matrix {{1,2,3,4},{5,4,3,2}}
```

```
o32 = | 1 2 3 4 |
      | 5 4 3 2 |
```

```
                2      4
o32 : Matrix ZZ <---- ZZ
```

```
i33 : B=matrix( {{1,1},{-1,2}} )
```

```
o32 = | 1 1 |
      | -1 2 |
```

```
                2      2
o33 : Matrix ZZ <---- ZZ
```

ここでの例では A と B の二つの行列を `matrix` 関数を使って生成している。これらの例で示す様に、Macaulay2 の行列は線形写像として捉えられている。行列 A の生成では `A=matrix 1,2,3,4,5,4,3,2`, 行列 B の生成では `B=matrix(1,1,-1,2)` と、行列 A では小括弧 `()` を外している事に注意されたい。Macaulay2 では一変数関数の場合に限り関数の括弧を省略する事が許容されている。

1.5.10 行列の結合に関連する演算子

行列にも文字列の演算子と同名の、行列の結合に関連する演算子 `|` と演算子 `||` がある。ここで、演算子 `|` は与えられた二つの行列を水平方向に繋げた行列を生成し、演算子 `||` は与えられた二つの行列を上下に結合する。この様に文字列の場合と雰囲気的には似た操作を行う演算子となっている:

```
o66 = | 1 2 |
      | 3 2 |
```

```
                2      2
o66 : Matrix ZZ <---- ZZ
```

```
i67 : x2=matrix{{0,1},{1,0}}
```

$$o67 = \begin{vmatrix} 0 & 1 \\ 1 & 0 \end{vmatrix}$$

o67 : Matrix ZZ ² ₂ <--- ZZ

i68 : x1|x2

$$o68 = \begin{vmatrix} 1 & 2 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{vmatrix}$$

o68 : Matrix ZZ ² ₄ <--- ZZ

i69 : x1||x2

$$o69 = \begin{vmatrix} 1 & 2 \\ 3 & 2 \\ 0 & 1 \\ 1 & 0 \end{vmatrix}$$

o69 : Matrix ZZ ⁴ ₂ <--- ZZ

1.5.11 行列の成分取出に関連する演算子

演算子_と^によって, 行列成分や小行列の取出が行える:
演算子_と演算^の構文

演算子	構文
-	〈行列〉_{〈整数 ₁ 〉, …, 〈整数 _n 〉}
^	〈行列〉^{〈整数 ₁ 〉, …, 〈整数 _n 〉}

ここで, 行列の列や行を取出す演算子では, 行列を演算子の左側に置き, 演算子の右側に取出す成分を指示するリストを配置する. ここで, 行列の成分番号は 1 ではなく 0 から開始する事に注意されたい:

i125 : A1=matrix {{1,2,3},{0,1,3},{4,1,9}}

$$\begin{array}{l} \text{o125} = | 1 \ 2 \ 3 | \\ \quad | 0 \ 1 \ 3 | \\ \quad | 4 \ 1 \ 9 | \end{array}$$

o125 : Matrix $\mathbb{Z}^3 \leftarrow \mathbb{Z}^3$

i126 : A1_{0}

$$\begin{array}{l} \text{o126} = | 1 | \\ \quad | 0 | \\ \quad | 4 | \end{array}$$

o126 : Matrix $\mathbb{Z}^3 \leftarrow \mathbb{Z}^1$

i127 : A1_{0,2}

$$\begin{array}{l} \text{o127} = | 1 \ 3 | \\ \quad | 0 \ 3 | \\ \quad | 4 \ 9 | \end{array}$$

o127 : Matrix $\mathbb{Z}^3 \leftarrow \mathbb{Z}^2$

i128 : A1^{0}

$$\text{o128} = | 1 \ 2 \ 3 |$$

o128 : Matrix $\mathbb{Z}^1 \leftarrow \mathbb{Z}^3$

i129 : A1^{0,2}

$$\begin{array}{l} \text{o129} = | 1 \ 2 \ 3 | \\ \quad | 4 \ 1 \ 9 | \end{array}$$

o130 : Matrix ZZ \leftarrow ZZ

猶, 演算子[^]は数値に対しては冪演算を行うメソッドでもある為, 演算子[^]の左側が数値や変数の場合は冪演算子として処理される事に注意されたい. 一方で, 演算子₋に関してはリストの代りに数値を設定しても挙動に違いはない:

o137 = $\begin{vmatrix} 1 & 2 & 0 \\ 7 & 1 & 3 \\ 0 & 1 & 9 \end{vmatrix}$

o137 : Matrix ZZ \leftarrow ZZ

i138 : A1^{2}

o138 = $\begin{vmatrix} 0 & 1 & 9 \end{vmatrix}$

o138 : Matrix ZZ \leftarrow ZZ

i139 : A1²

o139 = $\begin{vmatrix} 15 & 4 & 6 \\ 14 & 18 & 30 \\ 7 & 10 & 84 \end{vmatrix}$

o139 : Matrix ZZ \leftarrow ZZ

i140 : A1₋{2}

o140 = $\begin{vmatrix} 0 \\ 3 \\ 9 \end{vmatrix}$

o140 : Matrix ZZ \leftarrow ZZ

i141 : A1.2

```
o141 = | 0 |
        | 3 |
        | 9 |

        3
o141 : ZZ
```

1.5.12 リストと行列の算術演算

リストに対しては MATLAB の様な算術演算も出来なくはない. ここでの算術演算は, 同じ長さで同じ書式のリスト同士のみで可能である:

```
i26 : a={{1,2,3,4},{5,6}}
```

```
o26 = {{1, 2, 3, 4}, {5, 6}}
```

```
o26 : List
```

```
i27 : b={{3,1,2,4},{10,9}}
```

```
o27 = {{3, 1, 2, 4}, {10, 9}}
```

```
o27 : List
```

```
i28 : 2*a+b/5
```

```

      13 21 32 44      69
o28 = {{--, --, --, --}, {12, --}}
      5  5  5  5      5
```

```
o28 : List
```

この例ではリスト a と b を定義した後に $2*a+b/5$ でリストの各成分に対して $2 \cdot a + b/5$ の処理を行っている. しかし, MATLAB や Octave と比べ, Macaulay2 はリストの処理に強いとは言えない. その一例として, 積の順番の問題を示しておく:

```
i30 : b*2
```

```
stdio :30:2: expected pair to have a method for '*'
```

```
i31 : 2*b
```

o31 = {{6, 2, 4, 8}, {20, 18}}

上の b^*2 と $2*b$ では前者は失敗するものの、後者では正常に終了している。結局、数値の積をリスト上の演算として継承させているが、この時、スカラーとして左側からの作用しか考慮していない為である。この様にリストに対する積や冪乗の演算子に関しては注意が必要である。

猶、リストに対しては他の数式処理と同様に、apply 函数を用いて Macaulay2 の函数を作用させる事が可能である：

i43 : apply ({1,2,3,4,5}, i->i^2)

o43 = {1, 4, 9, 16, 25}

o43 : List

i44 : apply ({1,2,3,4,5}, i->i^2*sin(i))

o44 = {0.841471, 3.63719, 1.27008, -12.1088, -23.9731}

o44 : List

ここで apply 函数でリストに作用させる函数を Maple の様に演算子-> で定義している事に注意されたい。この函数の定義方法に関しては §1.7 にて詳細を述べる。

次に、行列に対しては和、差、積として冪演算が可能である：

行列の演算子

演算子	例	概要
+	A+B	行列の和
-	A-B	行列の差
*	A-B	行列の積
^	A^n	行列の冪 ($n \geq 0$)

但し、冪^は右側の被演算子の型によって意味が異なる事に注意されたい。

1.5.13 多項式環の定義

多項式環を定義する構文を次に示しておく:

多項式環の定義

$\langle \text{環の名前} \rangle = \langle \text{係数環} \rangle [\text{変数}_1, \dots, \text{変数}_n]$

ここで, 予め定義されてる係数環には, 前述の様に整数環 \mathbb{Z} (\mathbb{Z}), 有理数環 \mathbb{Q} (\mathbb{Q}) と実数体 \mathbb{R} (\mathbb{R}) の三種類の環があり, 他に, 利用者が定義した多項式環も勿論, 係数環として利用可能である. 次に具体的な多項式環の定義例を示そう:

i1 : R=QQ[x,y,z]

o1 = R

o1 : PolynomialRing

i2 : R'=ZZ[x,y,z]

o2 = R'

o2 : PolynomialRing

i3 : poly1=x^2+2*y^2+3*z^2-1

o3 = x² + 2y² + 3z² - 1

o3 : R'

この例では環 R の係数環に有理数体 \mathbb{Q} (Macaulay2 では \mathbb{Q} と表記), 変数を x, y, z の 3 個を指定している. すると, 項の順序を逆辞書式順序とする多項式環が生成され, 新たに環を生成したり, 環の切替を行わない限り, ここで定義した環 R 上で処理が行われる. 勿論, 環の定義で変数順序の指定も可能である. また, 係数環として, $\mathbb{Z}/7$, 即ち, $\mathbb{Z}/7$ を持つ環も定義可能である.

猶, 係数環によっては使えない関数が存在するので注意が必要である. 例えば, 与えられたイデアルの Groebner 基底を求める際に gb 関数を用いるが, この gb は整数環 \mathbb{Z} では使えない. 何故なら, 整数環上では係数の割算が出来ない為である.

環を定義すると, 自動的にその生成した環にポインタが移動する. つまり, 新しく生成した環で多項式等の対象を生成したり, 操作する事になる.

複数の環がある環境で, 既存の別の環に基礎環を切り替える場合, use 関数を用いる. 因に, Singular では setring 関数を用いて基礎環の切替を行う:

```
i5 : R=ZZ[x,y,z]
```

```
o5 = R
```

```
o5 : PolynomialRing
```

```
i6 : R2=QQ[x,y]
```

```
o6 = R2
```

```
o6 : PolynomialRing
```

```
i7 : use R
```

```
o7 = R
```

```
o7 : PolynomialRing
```

Singular ではポインタが置かれた環上で様々な対象を生成するが, この事は Macaulay2 でも同様である. 但し, Singular と違い, Macaulay2 では環毎に同じ変数名を持つ対象を持たせる事は出来ない:

```
i1 : R1=QQ[x,y,z];
```

```
i2 : R2=QQ[x,y,a];
```

```
i3 : R3=QQ[x,a,b];
```

```
i4 : use R1;
```

```
i5 : f=x+y+z;
```

```
i6 : use R2;
```

```
i7 : f=x*y*a^2;
```

```
i8 : use R3;
```

```
i9 : f=x*a^2+b;
```

```
i10 : use R1;
```

```
i11 : f
```

```
      2
o11 = x*a + b
```

```
o11 : R3
```

この例では、環 R1, R2 と R3 を定義し、各々に多項式を変数 f に割当てている。結局、変数 f の内容は環毎に保存されない為に変数 f の書換が行われている。この様に Macaulay2 では一つの変数に複数の意味を持たす事が出来ないが、その一方で、基礎環が異っていても対象の内容をエコーバックで確認する事が可能となる。この様子は、Singular が垣根で小さく区切られた庭園のようになっていて、同じ名前の対象を各区画に一つ置く事が出来るが、Macaulay2 の場合は広々とした公園になっているので、同名の対象を置く事が許容されないとしても例えられるだろうか。

ある対象がどの環に属するものかを調べたければ、対象の名前を入力し、その返却値で判断する事も可能だが、それとは別に、ring 関数でも出来る。この関数は、指定した対象が定義された環の名前を返す関数である。

そして、ある環で構築した対象を別の環に送り込む関数が substitute である。この関数 substitute の引数は対象と複写先の環を指定する:

```
i1 : R0=ZZ[x,y,z,w]
```

```
o1 = R0
```

```
o1 : PolynomialRing
```

```
i2 : poly1=x^2+2*y^2+3*z^2-1
```

```
      2    2    2
o2 = x  + 2y  + 3z  - 1
```

```
o2 : R0
```

```
i3 : R=ZZ[x,y,z]
```

```
o3 = R
```

o3 : PolynomialRing

i4 : poly2=substitute(poly1,R)

$$o4 = x^2 + 2y^2 + 3z^2 - 1$$

o4 : R

i5 : poly1

$$o5 = x^2 + 2y^2 + 3z^2 - 1$$

o5 : R0

i6 : poly2

$$o6 = x^2 + 2y^2 + 3z^2 - 1$$

o6 : R

この例では最初に環 $R0 = \mathbb{Z}[x, y, z, w]$ を生成し、環 $R0$ 上の多項式 poly1 を定義している。それから環 $R = \mathbb{Z}[x, y, z]$ とその環 R 上の多項式 poly2 を substitute 関数を用いて環 $R0$ の多項式 poly1 で定義している。

定義した対象の詳細は describe 関数 で調べる事が可能である:

i1 : R=ZZ/5[x,y]

o1 = R

o1 : PolynomialRing

i2 : describe R

$$o2 = \frac{\mathbb{Z}\mathbb{Z}}{5} [x, y]$$

i3 : f=(x+2*y+3)^5

$$o3 = x^5 + 2y^5 - 2$$

o3 : R

i4 : g=(x,y)->x*y+2

o4 = g

o4 : Function

i5 : describe R

$$o5 = \frac{\mathbb{Z}\langle x, y \rangle}{5}$$

i6 : describe f

$$o6 = x^5 + 2y^5 - 2$$

この例では describe 関数を用いて環や多項式の情報を表示させている.

1.6 文

Macaulay2の式は, 対象と演算子を結合することで出来た対象で, Macaulay2の評価によって新たに Macaulay2 の対象を返すものであるが, Macaulay2 の文は, Macaulay2 の式に加え, a=1 の様に変数に値を割当てる代入文や, 条件分岐や反復処理の様に, Macaulay2 の対象に対して操作を行い, 新たな対象等を返却するものである.

Macaulay2 では文を小括弧 () を用いてグループ化する事も可能である. 文のグループ化は制御文や関数の定義で用いられる.

1.6.1 制御文

Macaulay2 の制御文には,if 文, for 文や while 文がある.

制御文

制御文	構文
if	if <条件式> then <処理 ₁ > if <条件式> then <処理 ₁ > else <処理 ₂ >
for	for <制御変数> = <初期値> to <終値> do <処理> for <制御変数> = <初期値> to <終値> list <処理>
while	while <条件式> do <処理> while <条件式> list <処理>

ここでの条件文は解釈される事で true か false が返却される Macaulay2 の式である. <処理> は Macaulay2 の複数の文を小括弧 () を使ってグループ化した文である.

先ず,Macaulay2 の if 文は C の if 文と同じ構文である. 次に簡単な例を示しておこう:

```
i14 : i=random(10);if even i then a1=1 else a1=2
```

```
o15 = 2
```

Macaulay2 の for 文は, do か list でその挙動が異なる. 先ず, do の場合は処理を実行させ, list の場合には処理した結果をリストにして返却する.

```
i61 : for i from 1 to 5 do print(i*5)
```

```
5  
10  
15  
20  
25
```

```
i62 : for i from 1 to 5 list i*5
```

```
o62 = {5, 10, 15, 20, 25}
```

```
o62 : List
```

この点は while も同様である:

```
i69 : i=1;while i<5 do (i=i+1;print i^5)
```

```
32
```

```
243
```

```
1024
```

```
3125
```

```
i71 : i=1;while i<5 list (i=i+1; i^5)
```

```
o72 = {32, 243, 1024, 3125}
```

```
o72 : List
```

while や for ループを抜ける関数として break 関数がある。この break 関数は単体、或いは一つの引数を持つ。単体の場合、break は何も返さないが、引数を指定した場合、break はその引数の値を返す。

```
i18 : for i from 2 to 100 do if not isPrime i then break i
```

```
o18 = 4
```

```
i19 : i=1;while i<=100 do (i=i+1;if not isPrime i then break)
```

```
i20 :
```

どちらも同じ処理を行っているが、while 文の場合は break の引数が無い為に引数の値 (この場合は i の値) が返されていない。

1.7 関数

Macaulay2 には関数 (Function) がある. Macaulay2 の関数定義では演算子 `->` を用いる:

関数定義の構文

$$\langle \text{関数名} \rangle = (\langle \text{変数}_1 \rangle, \dots, \langle \text{変数}_n \rangle) \rightarrow (\langle \text{文}_1 \rangle; \dots \langle \text{文}_n \rangle;)$$

ここで文は `a=1` の様な変数の割当や制御文等であり, 複数の文はセミコロンの ; で区切ったものになる. 全体的な雰囲気は Maple の関数定義に似ている. 具体的な例を次に示しておこう:

```
i32 : f=(x,y,z) -> x^2+y/z
```

```
o32 = f
```

```
o32 : Function
```

```
i33 : f(1,2,3)
```

```
o33 = -
      5
```

```
o33 : QQ
```

この例では単純に $(x, y, z) \rightarrow x^2 + y/z$ の変換を行う. Macaulay2 には演算子 `:=` があり, この記号を使って関数内部で変数を割当てると, その変数は関数内部の局所変数として解釈される:

```
i4 : f=(x,y,z)->a:=x+y^3;
```

```
i5 : f(1,2,3)
```

```
o5 = 9
```

```
i6 : a
```

```
o6 = a
```

```
o6 : Symbol
```

```
i7 : g=(x,y,z)->a=x+y^3;
```

i8 : g(1,2,3)

o8 = 9

i9 : a

o9 = 9

最初の関数 f では演算子:=を用いている為, a は関数内部の値が設定されずに表象のままである. 次の g の例では演算子=を用いている為, a に値が設定されている事に注意されたい.

1.8 M2 ファイル

Macaulay2 で利用可能な関数は m2 ディレクトリに末尾が m2 のテキストファイルに記述されている. Macaulay2 ではこれらの m2 ファイルの読込に load 関数と needs 関数を用いる.

m2 ファイルは Macaulay2 の複数の関数や変数を定義する事が可能である. 記述は基本的に Macaulay2 に入力する書式で構わない. ここで, m2 ファイル内部での注釈行は演算子-0 えんさんし@演算子!-を先頭に置いた文字の羅列である. この演算子-は, Macaulay2 に直接入力する場合, 演算子の前に Macaulay2 の通常の入力行を必要とするが, m2 ファイルで用いる場合, その様な制約はない. この m2 ファイルの読込に用いる load 関数と needs 関数の構文を次に示しておく:

load 関数と needs 関数の構文

関数	構文
load	load "<ファイル名>"
needs	needs "<ファイル名>"

load 関数と needs 関数は共に指定した m2 ファイルを Macaulay2 に読込む関数であるが, load 関数が単純に指定したファイルを Macaulay2 に読込む関数であるのに対し, needs 関数は指定した m2 ファイルが既に読込まれている場合には m2 ファイルの読込を行わない関数である.

これらの関数は大域変数 path に設定されたリストに含まれるディレクトリから指定されたファイルを検索し, このリストに含まれていればファイルの読込を行う. 従って, 大域変数 path に含まれるディレクトリ上にファイルが存在する場合, これらの関数の引数としてはファイル名のみで構わない. しかし, ファイルが大域変数 path に含まれない場合, フルパスでファイルを指定しなければならない.

code 関数は Macaulay2 の関数がどの m2 ファイルに収録されたもので、具体的な関数の定義内容を表示する関数である。この code 関数は m2 ファイルの中身を表示する為、m2 ファイルを記述せずに定義した関数に対しては効力が無い。

1.9 ファイル出力

Macaulay2 からファイルへの書込みでは、演算子<<を用いる。

この演算子<<によって標準出力とファイルの出力の指定が行える。左被演算子を持たない場合、右被演算子を Maculay2 で解釈した結果を標準出力に出力する。左側に文字列を置いた場合、出力先は文字列で指定されたファイルになり。演算子<<の右被演算子に close が指定される迄、同じファイルのままとなる。猶、演算子<<は原始的な出力関数の為、単体で用いられた場合はファイルの先頭から書き込みを行う。又、演算子<<による出力の影響は普通の関数には及ばない。例えば、print 関数で表示した結果は<<によってファイルに切り替えられる訳ではない。その為、綺麗な書式のファイルを生成する事は出来ない。一時的に結果を残す程度の事しか出来ない。

第2章 特異点遊戯

2.1 Macaulay2による特異点の計算

曲線や曲面上のある点が、その点での偏微分が全て0になる場合、この点を特異点と呼ぶ。この章では Macaulay2 を使って特異点を探して遊ぶ。

最初に平面曲線 $x^3 - y^2 = 0$ を考えよう。この曲線の X, Y 方向の偏微分は各々、 $3x^2$ と $-2y$ である。従って、この平面曲線の特異点は原点 $(0, 0)$ になる。

図 2.1 に surf を使って描いたこの曲線のグラフを示す:

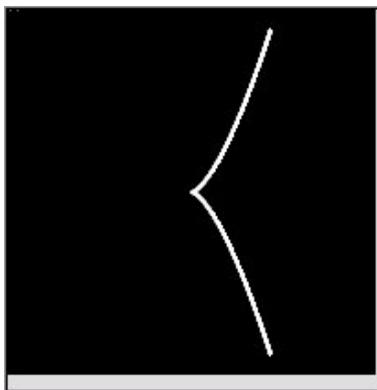


図 2.1: $x^3 - y^2 = 0$ のグラフ

このグラフから原点付近が縊れている事が判る。

ここで、この曲線の特異点の計算を Macaulay2 で行う。まず、曲線を多項式 f の零点集合として与え、多項式とその偏微分から生成されるイデアル I を定義する。すると、この多項式とその偏微分から生成されるイデアルの零点集合 $V(I)$ が多項式 f で与えられる曲線の特異点集合となる。

ここで多項式環の係数体が代数的に閉じている場合、 $V(I)$ を零点集合とするイデアルは I の根基イデアル (radical) \sqrt{I} に一致する事 (Hilbert の零点定理) が知られている。従って、特異点を調べたければ、先ず根基イデアル \sqrt{I} を計算し、その性質を調べれば良いことになる。

ここで、環 R のイデアル I の根基 \sqrt{I} は、環 R のイデアルで、 $g \in \sqrt{I}$ であれば、ある自然数 $n (\geq 1)$ に対して、 $g^n \in I$ となる性質を持つ。例えば、イデアル I が多項式 $x^2 + 2x + 1$ で生成されていれば、 I の根基 \sqrt{I} は $x + 1$

で生成されるイデアルになる. 更に, イデアル I の根基 \sqrt{I} は $I \subset \sqrt{I}$ を満たす. ここで, 根基の別の表現は, イデアル I を包含する全ての素イデアルの共通部分集合でもある.

さて, 実際に $x^3 - y^2 = 0$ を Macaulay2 で料理してみよう:

```
i1 : R=QQ[x,y];

i2 : R

o2 = R

o2 : PolynomialRing

o3 : describe(R)

o3 = QQ[x..y, Degrees => {2:1}, Heft => {1}, MonomialOrder =>
```

```
-----

{MonomialSize => 32}, DegreeRank => 1]
{GRevLex => {2:1} }
{Position => Up }
```

```
i4 : f=x^3-y^2;

i5 : neko=ideal {f, diff(x,f), diff(y,f)}
          3 2 2
o5 = ideal (x - y , 3x , -2y)

o5 : Ideal of R

i6 : radical neko

o6 = ideal (y, x)
```

さて, ここまでの処理を内容に沿って解説することにしよう:

環の定義: 最初に多項式 $f = x^3 - y^2$ が生息する環 R を, 有理数体 \mathbb{Q} を係数環とする環として定義している. Macaulay2 はオブジェクト指向の考えを取り入れている為, 考察すべき対象が定義可能な環を予め定義しなければなら

ない. この例では, 環 R を定義したことによって, 漸く, 多項式やイデアルを扱う事が可能となる. 猶, 定義した環は, 環の名前を入力しても真面に表示されないが, describe 函数を用いる事で, より詳細な情報を得る事が可能となる.

多項式の定義: 環 R を定義した後に, 漸く多項式 f とその偏微分から生成されるイデアル $neko$ が定義出来る様になる. ここで, 多項式は多くの数式処理と同様の表記で入力すれば良い.

イデアルの定義: 函数 ideal を用いてイデアルが定義出来る. この ideal 函数の構文を次に纏めておこう:

ideal 函数の構文

\langle イデアル名 $\rangle = \text{ideal} (\langle$ 多項式 $_1 \rangle, \dots, \langle$ 多項式 $_n \rangle)$
 \langle イデアル名 $\rangle = \text{ideal} (\langle$ 多項式リスト $\rangle)$
 \langle イデアル名 $\rangle = \text{ideal} (\langle$ 多項式行列 $\rangle)$

猶, 函数 ideal の小括弧 () は最初の構文の様に複数の多項式をコンマで区切る場合を除いて省略可能である.

多項式の微分: ここで考察するイデアルは, 多項式 f と f の変数 x や変数 y による微分も必要である. 多項式の微分では diff 函数を用いる. この diff 函数の構文も次に纏めておく:

diff 函数の構文

$\text{diff} (\langle$ 変数 \rangle, \langle 多項式 $\rangle)$
 $\text{diff} (\langle$ 変数 \rangle, \langle 行列 $\rangle)$

函数 diff の構文は, Maple, *Mathematica* や Maxima とは違い, 第 1 引数に変数で, 第 2 引数に微分すべき多項式が続く事に注意されたい. 更に, diff 函数で微分可能な対象は多項式と多項式を成分とする行列に限定される. 即ち, cos 函数の様な Macaulay2 組込の数値函数の微分が出来ない事を意味する.

イデアルの根基: ここで, イデアル $neko$ が生成出来たので, 今度はその根基を計算させる. 根基イデアルは radical 函数で計算出来る. この radical 函数の構文も次に纏めておこう:

radical 函数の構文

radical(\langle イデアル $\rangle)$

radical 函数による計算結果から, $x^3 - y^2$, $3x^2$ と $-2y$ で生成されるイデアル $neko$ の根基が (y, x) で生成されるイデアルである事が判る. 従って, この根基の零点集合は $x = 0$, $y = 0$ のみである. 則ち, $(0, 0)$ がこの曲線の特異点である.

今度は任意の有理数 t に対して, $x^3 - y^2 + t$ を考えよう. ここでも最初と同様の方法で根基イデアルを計算する:

```

i6 : R=QQ[x,y,t]

o6 = R

o6 : PolynomialRing

i7 : f=x^3-y^2+t;

i8 : neko = ideal { f, diff(x,f), diff(y,f) }

          3    2        2
o8 = ideal (x  - y  + t, 3x , -2y)

o8 : Ideal of R

i9 : radical neko

o9 = ideal (t, y, x)

o9 : Ideal of R

```

この計算結果から、根基イデアルが t, y, x で生成されたイデアルとなるので、 t が零でなければ $x^3 - y^2 + t$ には特異点が存在しない事が分る。実際、図 2.2 は surf を用いて $t = 1$ の場合を可視化したものであるが、全体的に滑らかな曲線となっている。

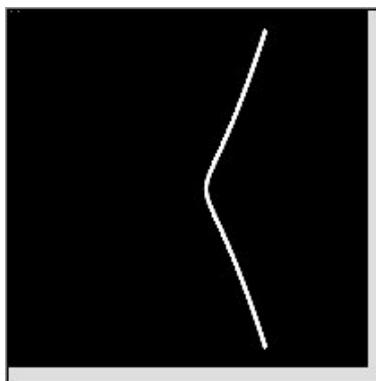


図 2.2: $t = 1$ の場合

関連図書

- [1] Kees Doets and Jan van Eijck, The Haskell Road to Logic, Maths and Programming, College Publications, 2004.
- [2] Eisenbud et al.(Eds), Computations in Algebraic Geometry with Macaulay 2, Springer-Verlag,New York-Heidelberg-Berlin,2002.
- [3] Gert-Martin Greuel, Gerhard Pfister, A Singular Introduction to Commutative Algebra, Springer-Verlag,New York-Heidelberg-Berlin,2000.
- [4] Hal Schenck, Computational Algebraic Geometry London Mathematical Society student texts;58,2003.
- [5] Macaulay2 のサイト <http://www.math.uiuc.edu/Macaulay2/>
- [6] SINGULAR のサイト <http://www.singular.uni-kl.de/>

索引

演算子

$*$, 7
 $+$, 7
 $-$, 7
 \dots , 14
 $/$, 7
 $=$, 11
 $\#$, 14
 \wedge , 7
 \rightarrow , 21
 $:=$, 29
 \ll , 31
and, 10
not, 10
or, 10
!, 8

関数

I
ideal, 33

記号

QQ, 6, 22
RR, 6, 22
ZZ, 6, 22

制御文

for, 27
if, 27
while, 27

大域変数

P
path, 30

データ型

false, 9
true, 9

関数, 29

行列 (Matrix), 13

シンボル, 11

配列 (Array), 13

表象, 11

文字列, 11

リスト (List), 13

列 (Sequence), 13

関数

A

apply, 21

C

code, 31

D

describe, 25

L

load, 30

M

matrix, 16

N

needs, 30

R

radical, 34

ring, 24

S

substitute, 24

U

use, 23

き

局所変数, 29

し

実数, 6

せ

整数, 6

ほ

ポインタ, 22

ゆ

有理数, 6

り

リストの処理, 20