Macaulay2の紹介

横田博史

平成18年10月25日(水)

まえがき

この文書は Macaulay2 の入門用に書いたものである．より詳細は Schenk の本 [3] や Eisenbud 等の本 [1] を参照されたい．

尚,Schenk の本は Macaulay2 を用いて代数学を学ぶ為の本の為, 数学のお勉強に重点が置かれているのに対し,Eisenbud 等の本は Macaulay2 を使って代数や代数幾何を料理する本の為,Macaulay2 の言語や使い方に重点が置かれている．

又,Macaulay2 では Singular で用いられているライブラリが利用されている．この Singular に関しては,Greuel-Pfister の本 [2] を参照されると良い. こちらは Singular を使った代数学の教科書である．

平成 18 年 10 月 25 日 (水)　　横田博史

目 次

x

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. その一方で,Macaulay2の構文にはMathematica風なものも見受けられ,Singularとは別の面白味のある数式処理システムである.

Macaulay2の大きな特徴として挙げられる事は,Macaulay2はオブジェクト指向の数式処理システムである事だろう. その為, 函数はオブジェクトに束縛されたメソッドとなる, 従って, 基本的に何かの処理を行う為にはオブジェクト生成を最初に行わなければならない. 又, 函数も対象が違えば結果も当然異なる. 具体的に述べると, 多項式やイデアルといったものの計算を行う為, その親にあたる環を先ず生成しなければならない. 又, 演算子^は左側の被演算子が行列で右側の被演算子が数値の場合は行列の冪となる. ところが, 右側の被演算子が整数リストの場合, 行列の列の取り出しの演算子になる.

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であるが,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.

Macaulay2は詳細なオンラインマニュアルを有する. 尚,0.9.8より前のMacaulay2では, `help` 函数でしか参照出来ず,xterm等の仮想端末を利用する場合でも, Emacs風の行の編集と履歴の利用が出来ない. そして,`help` 函数はテキストの内容を一気に表示するものである. その為,xtermの様な仮想端末では表示内容がバッファに入り切らない事もあるので,EmacsやTeXmacs等のアプリケーションをフロントエンドとして使わなければ尚更不便になる.

これが0.9.8になるとEmacs風の行編集と履歴の利用が可能で, オンラインマニュアルも `help` 函数以外にTexintoを用いる `infoHelp` 函数やMozilla等のブラウザを用いる `viewHelp` 函数の二つの函数が用意されており, 何かと使い勝手が向上している.

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”グレブナエンジンのプログラマ達”に掲載されている.

英語の書籍としては Eisenbud 等の本 [1] と Schenck の本 [3] の二つが出ている.

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, 代数幾何に詳しい方で,Macaulay2 を使って何かをしたい為に,Macaulay2 の使い方を調べるのに適した本だろう.

次の Schenck の本の方は,Macaulay2 を用いた代数幾何, 代数的位相幾何学の入門書で, これらの事項が手堅く纏められた良いテキストである. 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.

この冊子は手早く Macaulay2 の使い方を解説する事を目的としている.Macaulay2 の詳細に関しては, 上述の本や Macaulay2 のオンラインマニュアルを参照されたい.

1.2 起動方法と入力

Macaulay2 の起動方法はパスが通っていれば,xterm 等の仮想端末に M2 と入力すると Macaulay2 が立ち上がる.KNOPPIX/Math を使っていれば, メニュ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.

すると図 1.1 の様に Macaulay2 が立ち上がる.

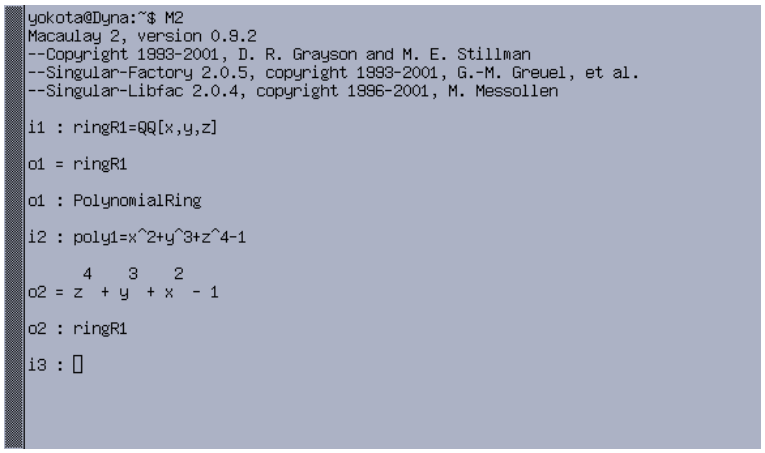

図 1.1: kterm 上で利用

但し,M2 は前述の様に編集機能を持ったフロントエンドではないので,GNU Emacs や TeXmacs 等と云った外部のプログラムと併用する事を勧める.

特に,TeXmacs をフロントエンドとして使う場合,Macaulay2 で処理した結果が美しくレンダリングされて表示されるので目の保養にも良い.TeXmacs を用いる場合には, 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 を選択すれば良い.

図 1.2 に TeXmacs をフロントエンドにした様子を示している. 図 1.1 の kterm 上の Macaulay2 と同じ操作を行っているが, TeXmacs 上の方がきれいに表示されている事が分かる.

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 `i` と入力番号の組合せで, 出力のプロンプトも同様に出力 `o1 :` の様に出力行である事を示す `o` と入力番号の組合せとなっている.

尚,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エコーバックが無効になる. それ以外では, 入力に対して値やデータ型のエコーバックを行う.

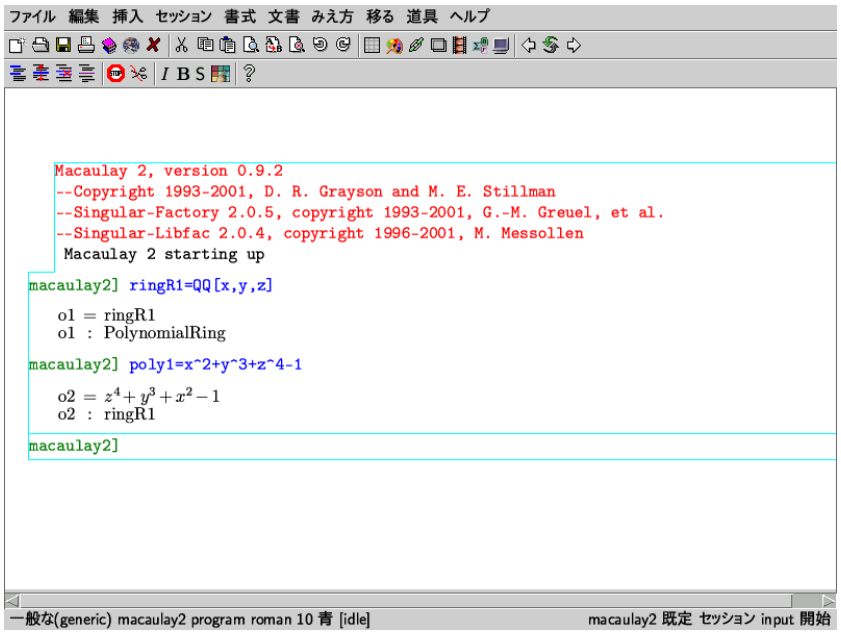

図 1.2: TeXmacs 上で利用

Macaulay2 は詳細なオンラインマニュアルを持っている. 最初にオンラインマニュアルを読む函数として `help` 函数がある.`help` 函数は help とだけ入力すれば,help 函数の概要が表示される. 尚,help 函数は一気に内容を表示する為, xterm 等の仮想端末で Macaulay2 を起動した場合, ファイルの内容がバッファに入り切らずに頭が読めなくなる事がある. その為,`help` 函数しか使えない古い Macaulay2 や,xterm 等の仮想端末でどうしても使いたい場合,Emacs 上で利用する事を勧める.

Macaulay2-0.9.8 では, 普通のテキスト形式のマニュアルを表示する `help` 函数の他に,Texinfo を用いる `infoHelp` 函数,HTML ブラウザを利用する `viewHelp` 函数が使える.

ここでは Firefox 等の HTML ブラウザを用いる `viewHelp` について簡単に述べよう. 尚, 引数は help,infoHelp,viewHelp も全て同じである. 単純にオンラインマニュアルが通常の Text,texinfo か HTML かの違いが生じる程度であるが, 今風に `viewHelp` を使うのが便利が良いだろう.

先ず, `viewHelp` とすると 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. `viewHelp "help"` で 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. 基本的に viewHelp "⟨ 事項 ⟩" で調べたい ⟨ 事項 ⟩ のマニュアルが表示される.

最後に Macaulay2 を終了する場合,`quit` と入力すれば良い.

1.3 基本的なデータ型について

Macaulay2 では Singular と同様に最初に環を定義し, その環上で様々な対象を構築して行く.Macaulay2 はオブジェクト指向の処理言語の為, オブジェクトを生成しなければ, メソッドが使えない. これを具体的に言うと, 最初に環を定義しなければ, 多項式や写像は定義出来ないし, これらを処理する函数も使えない事になる.

尚,Symbol,Sequence, 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データ型, これらのデータで構成されたリスト, 配列, 行列の入力と処理は可能である. 更に, それらを処理する函数も定義可能である.

1.3.1 数

Macaulay2 で扱える数値は, 整数, 有理数, 実数である. Macaulay2 では様々な対象を定義する為に, 予めその対象が含まれる環を定義する必要があるが, 整数環 $\mathbb{Z}$, 有理数環 $\mathbb{Q}$ と実数環 $\mathbb{R}$ は最初から Macaulay2 に組込まれた環となっている. これらの環は Macaulay2 では各々`ZZ`,`QQ`, `RR` で表記される.

従って, これらの環に含まれる数値に関しては, そのまま入力が可能であり, それらの数値に対しては四則演算も出来る.

数値の入力例を以下に示しておく:

```
i1 : 1

o1 = 1

i2 : 1+2

o2 = 3

i3 : 2*3

o3 = 6

i4 : 3/5

     3
o4 = -
     5
```

```
o4 : QQ

i5 : 2.9

o5 : 2.9

o5 : RR
```

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, 入力が有理数と浮動点小数になれば, その計算で用いたデータ型 (この場合は属する環) を計算結果に続けて表示する.

Macaulay2 では四則演算は C や Maple 等の数式処理と同様の演算子が利用可能である.

四則演算

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	冪乗 a^b

尚,FORTRAN で用いられる冪乗の演算子**は Macaulay2 では使えない.

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

     19
o4 = --
      5

o4 : QQ

i5 : 1+0*1.9-2/5

o5 = 0.6

o5 : RR
```

この例の様に数値のみであれば電卓風に利用可能である.

その他の数値演算子として階乗!がある. ここで, 演算子!を複数続ける事が可能である. 例えば, 4!!は (4!)!と解釈される.

1.3.2 数値函数

Macaulay2 には cos 等の三角函数や exp や log といった指数函数や対数函数が組込まれている. これらの函数は Maple 等の函数とは異なり, 実数環 $\mathbb{R}$ 上の数値函数である.

```
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
```

これらの数値函数は一般の環上に使える様に拡張されていない為,x を変数として含む多項式環を定義しても,texttttsin(x) は x に数値が割当てられていない為にエラーを返す. 更に, 多項式の微分を行う diff 函数の引数として, 数値函数を引渡しても,sin の変数が数値でない事に加え,sin(x) 自体が多項式でない為にエラーになる.

1.3.3 論理式と論理演算

Macaulay2 で真偽値は `true` と `false` で表現される.

ここで Macaulay2 の論理式は, 評価を行った結果として true か false を返す式の事である.

代表的な論理式としては,`a==1` の様に比較の演算子を用いたものが挙げらでる. 以下に Macaulay2 の持つ比較の演算子を示す.

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
```

尚,Macaulay2 には面白い演算子?がある. この演算子は, `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す函数である.

```
i26 : 3 ? 3

o26 = ==

o26 : Keyword

i27 : 3 ? 5

o27 = <
```

```
o27 : Keyword
```

Macaulay2 の論理演算子を以下に示しておく.

論理演算子

演算子	構文	概要
and	a and b	論理式 a と b の論理積
or	a or b	論理式 a と 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)

以下に具体例を示す.

```
i20 : 3>1 and 3<2

o20 = false

i21 : 3>1 or 3<2

o21 = true

i22 : not(3<1)

o22 = true
```

尚,`not` に関しては,`not 3<1` の様に小括弧 `()` を外しても構わない.

ここで, 論理演算子 `and` に関しては, 論理式 `a or b` に含まれる論理式 `a` が偽の場合, 論理式 `b` の評価を Macaulay2 は行わない. `and` と同様に論理演算子 `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 Macaulay2 は行わない仕様となっている.

1.3.4 表象と文字列

数値の他に,Macaulay2 では表象 (Symbol) と文字列 (String) が扱える.

先ず, 表象はアルファベットで開始するアルファベットと数値と一部の記号で構成される文字の羅列である. 但し,Macaulay2 の処理言語に含まれる for 等の文字の列は表象にならない. 表象は Macaulay2 で変数として利用可能である.

次に, 文字列は二重引用符"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. 尚,"を文字列中に含める場合,\ を用いる. 例えば, \" の様に表記すると良い.

以下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.

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.

以下に,!と||の演算子の違いを示しておく.

```
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,|の場合は単純に文字列を結合するだけであるが, ||の場合は二つの文字列を結合するだけではなく, 文字列の間に改行が入る. 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

1.3.5 列, リスト, 配列と行列の定義

Macaulay2 には基本データ型で構成された列 (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

列	$(a_1, \cdots, a_n)$	対象をコンマ, で区切って小括弧で括ったもの
リスト	$a_1, \cdots, a_n$	列の中身を中括弧{}で括ったもの
配列	$[a_1, \cdots, a_n]$	列の中身を大括弧 [] で括ったもの

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)

                5
o29 = (1, 2.4, -)
                7

o29 : Sequence

i30 : {1,2,3,54}

o30 = {1, 2, 3, 54}

o30 : List

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

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

o31 : Array
```

Macaulay2 の面白い機能に,1 づつ増加する数列を簡単に生成する事が容易に行える事がある, この場合, 演算子.. を用いる. 演算子.. は ⟨ 開始値 ⟩..⟨ 終値 ⟩ で, ⟨ 開始値 ⟩ から開始して 1 刻みで ⟨ 終値 ⟩ までの数値列を生成する. 例えば,(1..5) で (1,2,3,4,5) が生成される. この演算子.. は下付きの添字を表現する演算子_と組合せる事で,x_1,x_2,..,x_10 の様に添字付けられた表象列を x_1..x_10 と入力して容易に生成する事が可能になる.

以下に, 演算子.. の例を示す.

```
i1 : 1..4

o1 = (1, 2, 3, 4)

o1 : Sequence

i2 : 12..19

o2 = (12, 13, 14, 15, 16, 17, 18, 19)

o2 : Sequence

i3 : x_1..x_5

o3 = (x , x , x , x , x )
       1   2   3   4   5

o3 : Sequence
```

列, リスト, 及び配列の成分や長さを求める演算子に# がある.#の構文を以下に示しておく.

演算子#

構文	概要
⟨a⟩ # ⟨ 整数 ⟩	⟨a⟩ の第 ⟨ 整数 ⟩ + 1 番目の成分
# ⟨a⟩	⟨a⟩ の長さ

列, リスト, 配列の成分は C と同様に 0 から開始するので, 配列 a の左から第 5 番目の成分を取出したい場合,`a#4`と入力する事に注意されたい.

以下に演算子#の実行例を示しておく.

```
i50 : a=(1..10)

o50 = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

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
```

列, リストと配列と異なり, 行列は函数 `matrix` を用いて定義する.
以下に matrix 函数の例を示しておく.

```
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 matrix 函数を使って `A` と `B` の二つの行列を生成している. 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.
この際に, 行列 A の生成では `A=matrix 1,2,3,4,5,4,3,2` で行列 B の生成では `B=matrix( 1,1,-1,2 )` としているのに対し, 行列 A では小括弧 `()` を外している事に注意されたい. この様に Macaulay2 の特徴として, 一変数函数の場合に限って, 函数の括弧を省略する事が許容されている.
行列の演算子として`|`と`||`の二つがある. これらの演算子は文字列の演算子としても現われているが, 行列の演算子としても少し似た動作となるが別の働きをする.
先ず, 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.

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

```
      | 3 2 |

               2        2
o66 : Matrix ZZ  <--- ZZ

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

o67 = | 0 1 |
      | 1 0 |

               2        2
o67 : Matrix ZZ  <--- ZZ

i68 : x1|x2

o68 = | 1 2 0 1 |
      | 3 2 1 0 |

               2        4
o68 : Matrix ZZ  <--- ZZ

i69 : x1||x2

o69 = | 1 2 |
      | 3 2 |
      | 0 1 |
      | 1 0 |

               4        2
o69 : Matrix ZZ  <--- ZZ
```

次に, 行列から列や行を取出す場合, 演算子_と^を用いる.

演算子_と^

⟨ 行列 ⟩_{⟨ 整数$_1$⟩, $\cdots$, ⟨ 整数$_n$⟩}
⟨ 行列 ⟩ ^ {⟨ 整数$_1$⟩, $\cdots$, ⟨ 整数$_n$⟩}

行列の列や行を取出す演算子は演算子の左側に行列を置き, 右側に取出す成分のリストを配置する. ここで, 行列に関しても成分番号は 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}}

o125 = | 1 2 3 |
       | 0 1 3 |
       | 4 1 9 |

                3        3
o125 : Matrix ZZ  <--- ZZ

i126 : A1_{0}

o126 = | 1 |
       | 0 |
       | 4 |

                3        1
o126 : Matrix ZZ  <--- ZZ

i127 : A1_{0,2}

o127 = | 1 3 |
       | 0 3 |
       | 4 9 |

                3        2
o127 : Matrix ZZ  <--- ZZ

i128 : A1^{0}

o128 = | 1 2 3 |

                1        3
o128 : Matrix ZZ  <--- ZZ

i129 : A1^{0,2}

o129 = | 1 2 3 |
       | 4 1 9 |
```

```
                2        3
o130 : Matrix ZZ  <--- ZZ
```

尚, 演算子^は冪演算子としての特性を持つ為, 演算子^の左側がリストでなく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.

尚, 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 = | 1 2 0 |
       | 7 1 3 |
       | 0 1 9 |

                3        3
o137 : Matrix ZZ  <--- ZZ

i138 : A1^{2}

o138 = | 0 1 9 |

                1        3
o138 : Matrix ZZ  <--- ZZ

i139 : A1^2

o139 = | 15 4  6  |
       | 14 18 30 |
       | 7  10 84 |

                3        3
o139 : Matrix ZZ  <--- ZZ

i140 : A1_{2}

o140 = | 0 |
       | 3 |
       | 9 |

                3        1
o140 : Matrix ZZ  <--- ZZ

i141 : A1_2
```

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

         3
o141 : ZZ
```

1.3.6 リストと行列の演算

Macaulay2 の面白い機能で, リストに対しては MATLAB の様な四則演算も出来なくはない点がある. 演算は同じ長さで同じ書式のリスト同士で可能である.

```
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 $2 \cdot a + \frac{b}{5}$ の処理を行っている.

但し,MATLAB や Octave と比べると,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. この様にリストに対する積や冪乗に関しては注意が必要である.

この様に,MATLAB の様な処理が行えるとは限らない. 尤も, その様な処理を Macaulay2 で行う事は考え難いが….

尚, 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 1.5 小節にて詳細を述べる.

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

+	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^に関しては右側の被演算子の挙動によって意味が異なる事に注意されたい.

1.3.7 環

Macaulay2 では環の定義は以下の構文で環を定義する.

環の定義

⟨ 環の名前 ⟩ = ⟨ 係数環 ⟩[変数$_1$, $\cdots$, 変数$_n$]

ここで係数環としては, デフォルトで Macaulay2 に定義されている 3 個の環, ZZ(整数環 $\mathbb{Z}$),QQ(有理数体 $\mathbb{Q}$),RR(実数体 $\mathbb{R}$), 更に, 利用者が定義した環が利用可能である.

以下に具体的な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

       2     2     2
o3 = x  + 2y  + 3z  - 1

o3 : R'
```

この例では環 R の係数環に有理数体 $\mathbb{Q}$(Macaulay2 では `QQ` と表記) を指定し, 変数を x, y, z の 3 個を指定している. この指定を行うと, 逆辞書式順序で環が生成され, 以下の処理では新たに加群や環を定義するまでは, ここで定義した環 R 上で処理を行う. 環の定義では変数の順序の指定等も行える. 勿論, $\mathbb{Z}\mathbb{Z}_7$, 即ち,`ZZ/7` の様な環も定義可能である.

尚, 係数環によっては使えない函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` 函数を用いるが, この gb は整数環 $\mathbb{Z}$ では使えない (係数の割算が出来ない為!).

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. 即ち, 新しく生成した環で多項式等の対象を生成し, 操作する事となる.

ここで, 複数の環がある環境で, 既存の別の環に環を切り替える場合, use 函数を用いる. 因に Singular では setring 命令を用いる.

```
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
```

但し,Singular ではポインタが置かれた環 (基礎環と呼ぶ) 上で様々な対象を生成するが,Macaulay2 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, 変数 f の内容の書換が行われている事が判る. 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様になっており, 同じ名前の物は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なので, 同名の物を置く事が許容されないとで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て, そのエコーバックで判断する事も可能であるが, それとは別に,`ring` 函数を用いる事でも出来る. この函数はその対象が定義された環の名前を返す函数である.

ある環で構築した対象を別の環に代入する函数が `substitute` である. この函数 `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)

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

o4 : R

i5 : poly1

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

o5 : R0

i6 : poly2

      2     2     2
o6 = x  + 2y  + 3z  - 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 函数を用いて環 R0 の多項式 poly1 で定義している.

尚, 定義した対象の詳細は `describe` 函数 で調べる事が可能である.

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

o1 = R

o1 : PolynomialRing

i2 : describe R
```

```
     ZZ
o2 = -- [x, y]
      5

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

      5     5
o3 = x  + 2y  - 2

o3 : R

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

o4 = g

o4 : Function

i5 : describe R

     ZZ
o5 = -- [x, y]
      5

i6 : describe f


      5     5
o6 = x  + 2y  - 2
```

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

1.4 文

Macaulay2 の文は,`a=1` の様に値を割当てる文, 単純に `1+2*3` や `3<1` の様に Macaulay2 を使って解釈を行うものであ!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函数の定義で用いられる.

1.4.1 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 ⟨ 条件文 ⟩ then ⟨ 処理$_1$⟩
	if ⟨ 条件文 ⟩ 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 ⟨ 条件文 ⟩ do ⟨ 処理 ⟩
	while ⟨ 条件文 ⟩ 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 の文である. ⟨ 処理 ⟩ は Macaulay2 の文, 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. 以下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
```

これは 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
```

white や for ループを抜ける函数として `break` がある. break 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 (この場合は i の値) が返されていない.

1.5 函数

Macaulay2 には函数 (Function) がある.Macaulay2 の函数定義では以下に示す様に演算子->を用いる.

函数定義の構文

⟨ 函数名 ⟩ = (⟨ 変数$_1$⟩, $\cdots$, ⟨ 変数$_n$⟩) -> (⟨ 文$_1$⟩; $\cdots$ ⟨ 文$_n$⟩;)

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; で区切ったものになる. 全体的な雰囲気は Maple の函数定義に似ている.

以下に具体的な例を示しておこう.

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

o32 = f

o32 : Function

i33 : f(1,2,3)

      5
o33 = -
      3

o33 : QQ
```

この例では単純に $(x, y, z) \to x^2 + y/z$ の変換を行う.Macaulay2 には演算子:=があり, この記号を使って函数内部で変数を割当てると, その変数は函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
```

最初の函数 f では演算子:=を用いている為,a は函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.

1.6 M2 ファイル

Macaulay2 で利用可能な函数は 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 函数と needs 函数を用いる.

m2 ファイルは Macaulay2 の複数の函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.

m2 ファイルの読込に用いる load 函数と needs 函数の構文を以下に示しておく.

load 函数と needs 函数

```
load "⟨ファイル名⟩"
needs "⟨ファイル名⟩"
```

load 函数と needs 函数は共に指定した m2 ファイルを Macaulay2 に読込む函数であるが,load 函数が単純に指定したファイルを Macaulay2 に読込む函数であるのに対し,needs 函数は指定した 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函数である.

これらの函数は大域変数 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, これらの函数の引数としては, 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 に含まれない場合には, フルパスでファイルを指定しなければならない.

code 函数は Macaulay2 の函数がどの m2 ファイルに収録されたもので, 具体的な函数の定義内容を表示する函数である. この code 函数は 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函数に対しては効力が無い.

1.7 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<<は通常出力とファイルの出力の指定が行える. 左側被演算子を持たない場合は, 右側の被演算子を 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となり. <<の右側被演算子に close が指定される迄, 同じファイルのままとなる. 尚,<<は原始的な出力函数の為, 単体で用いられた場合, ファイルの先頭から書き込みを行う.

更に, この演算子<<による出力の影響は普通の函数には及ばない. 例えば, print 函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　はじめに

この章では Macaulay2 を使った具体例として, 曲線や曲面の特異点を計算させてみよう.

2.2　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事にする.

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.

次の図 2.1 に surf で用いて描いたこの曲線のグラフを示す.

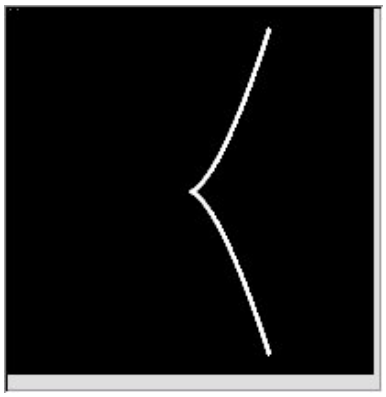

図 2.1: $x^3 - y^2 = 0$ のグラフ

上のグラフから原点付近が縊れている事が判る.

この特異点の計算を Macaulay2 で行う. 先ず, 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) $\sqrt{I}$ に一致する事が知られている

(Hilbelt の零点定理). 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事になる.

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, $g^n \in I$ となる性質を持つものである. 例えば, イデアル I が多項式 x^2+2x+1 で生成されている場合, I の根基 $\sqrt{I}$ は $x+1$ で生成されるイデアルとなる. イデアル I の根基 $\sqrt{I}$ は $I \subset \sqrt{I}$ を満すイデアルでもある. この根基の別の表現は, 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として表現される.

では, 実際に $x^3-y^2=0$ を Macaulay2 を使って遊んでみよう.

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

i2 : f=x^3-y^2;

i3 : neko=ideal {f, diff(x,f), diff(y,f)}
              3    2    2
o3 = ideal (x  - y , 3x , -2y)

o3 : Ideal of R

i4 : radical neko

o4 = ideal (y, x)
```

この例では, 最初に有理数体 $\mathbb{Q}$ を係数環に持つ環 R とその環上の多項式 f を定義している. この様に Macaulay2 では多項式の処理を行う前に必ず親の環を定義しなければならない. その為, 環を定義して初めて多項式や多項式で生成するイデアル等が扱える様になる.

ここでは多項式 f とその偏微分から生成されるイデアル *neko* を定義する. 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. 但し, 変数への代入は大域変数として利用するのであれば演算子=を用い,Macaulay2 のプログラムで局所的に利用するのであれば, 演算子:=を用いる.

イデアルの定義では函数 `ideal` を用いる. この `ideal` 函数の構文を以下に示しておく.

ideal 函数の構文

`ideal` (⟨ 多項式$_1$⟩, ⟨ 多項式$_n$⟩)
`ideal`(⟨ 多項式リスト ⟩)
`ideal`(⟨ 多項式行列 ⟩)

基礎環上の多項式, 多項式の列にリスト, 多項式行列等からイデアルは定義可能である. ここで, 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.

そして, 多項式の微分では `diff` 函数を用いる. この `diff` 函数の構文も以下に示しておこう.

diff 函数の構文

`diff`(⟨ 変数 ⟩, ⟨ 多項式 ⟩)
`diff`(⟨ 変数 ⟩, ⟨ 行列 ⟩)

この様に, 微分の函数 `diff` は Maple,Mathematica,Maxima とは違い, 最初の引数が変数, その次に多項式となる事に注意されたい. 更に,`diff` 函数で微分可能なもの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` の様な Macaulay2 組込の数値函数の微分は出来ない.

イデアルの根基は `radical` 函数で計算出来る. この `radical` 函数は以下に示す様に引数にイデアルを取り,. 根基を返すが, この根基はイデアルである.

radical

radical(⟨ イデアル ⟩)

この例では,x^3-y^2,$3x^2$,$-2y$ 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となる事が分る. この根基の零点集合は $x=0, y=0$ のみである. 以上から, $(0,0)$ がこの曲線の特異点となる事が分る.

今度は任意の有理数 t に対して,x^3-y^2+t を考えよう. ここでも上と同様の方法で根基イデアルを計算しよう.

```
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
```

上記の結果から, t が零でなければ $x^3 - y^2 + t$ には特異点が無い事が分る.

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, この様に滑らかな曲線となっている事が判る.

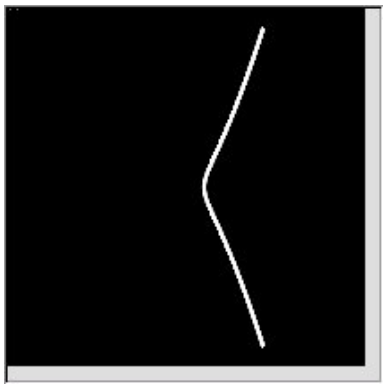

図 2.2: $t = 1$ の場合

関連図書

[1] Eisenbud et al.(Eds), Computations in Algebraic Geometry with Macaulay 2, Springer-Verlag,New York-Heiderberg-Berlin,2002.

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

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

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

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

索 引