MuPAD: Multi Processing Algebra Data Tool
MuPAD 1.4 上級デモンストレーションツアー

Klas Drescher, Frank Postel, Thorsten Schulze[1]

1997 年 1 月 15 日

[1] 幸谷智紀・角谷悟 共訳, Ver.0.2:2000 年 3 月 7 日 (火)

概 要

この文書は「デモンストレーションツアー MuPAD1.4」[1]の補足説明であり，MuPAD 文書のうちの一つです。特定の数学分野における幾分高度な計算を取り扱っています。

[1](訳注)“Demonstration Tour with MuPAD 1.4”のこと。ちなみに日本語訳は我々の訳したものが http://member.nifty.ne.jp/tkouya/ で公開されている。

目 次

この文書のうち，1・3・5・7 章は Frank Postel が，2 章は Klans Dresher が，4 章は Thorsten Schulze がそれぞれ執筆しました。

Paul Zimmermann はこの文書を丹念に読み，重要なヒントを与えてくれました。ここに感謝いたします。

第1章　MuPAD 1.2.2からの変更点についての覚え書き

MuPAD 1.2.2 以降，沢山の新しい関数が導入・拡張され，幾つかのライブラリパッケージに追加されてきました。

この章では，MuPAD 1.2.2 との主要な違いについてだけ述べておきたいと思います。MuPAD 1.4 における新規事項や変更点についての完全な情報はマニュアルを参照して下さい。

1.1　ライブラリパッケージを明示的に読み込んでおく必要がなくなった

利用できる MuPAD のライブラリパッケージは，全てセッションの最初で読み込んでいましたが，`loadlib` コマンドを使うことで，明示的に読み込んでおく必要が無くなりました。

1.2　domains や linalg 等のパッケージを使用する際の変更点

`domains` ライブラリの関数は，次の 3 種類に分類されました。

- **axiom** は `Ax` ドメインのメソッドになります。
- **category** コンストラクタは `Cat` ドメインのメソッドになります。
- **domain** コンストラクタは `Dom` ドメインのメソッドになります。

従って，この 3 種類の関数の使い方は MuPAD 1.2.2 とは異なっています。それをこれから例示していきたいと思います。

セッションをリセットして，パッケージが読み込まれていないことを確認して下さい。

```
>> reset():
```

`loadlib("domains")` という文が無くとも，Z_4 上の正方行列の環を生成できるようになりました。

```
>> M := Dom::SquareMatrix(3, Dom::IntegerMod(4));

                  Dom::SquareMatrix(3, Dom::IntegerMod(4))

\end{{verbatim}
```

反面，ドメインコンストラクタである{\tt Matrix}や{\tt IntegerMod}といった呼出し方の代わりに，{\tt Dom::SquareMatrix}とか{\tt Dom::IntegerMod}というように呼び出さなければなりません。

乱数行列を生成してみましょう。

```
\begin{verbatim}
>> A := M::random();
```

```
                    +-                          -+
                    |  1 mod 4, 2 mod 4, 1 mod 4  |
                    |                             |
                    |  3 mod 4, 0 mod 4, 2 mod 4  |
                    |                             |
                    |  1 mod 4, 0 mod 4, 1 mod 4  |
                    +-                          -+
```

この行列 M が単なる環なのか，可換環なのかをチェックすることも可能です。

```
>> M::hasProp(Cat::Ring), M::hasProp(Cat::CommutativeRing);
```

```
                          TRUE, FALSE
```

Ring や CommutativeRing といったもの (カテゴリ) は，Cat ライブラリの関数として呼び出さなければなりません。

更に，明示的に linalg パッケージを呼び出さずに，そのパッケージの関数 det を利用することができるようになりました。

```
>> linalg::det(A);
```

```
                          2 mod 4
```

ユーザは export コマンドを使って，これらのドメインのメソッドを「エクスポート」する事もできます。従って，例としては，次の文

```
>> export(Ax): export(Cat): export(Dom):
```

を実行した後，以下のように書けばよい，ということがご理解いただけるでしょう。

```
>> M := Matrix(DistributedPolynomial([x], Integer));
```

```
      Dom::Matrix(Dom::DistributedPolynomial([x], Dom::Integer))
```

1.3 識別子への書き込み防止ができる

MuPAD 1.2.9 からは，値やライブラリ関数の書き込みを防止する機能 (ライトプロテクト) が導入されています。

```
>> expr := exp(x^2) * ln(x - 1);
Error: Illegal assignment. Identifier is write protected [expr]
```

更に，ある識別子に値を代入すると同名の関数オプションが使えなくなってしまうような場合，ユーザに警告を発するようになります。例えば，RatExpr 識別子を見てみましょう。この名前は indets 関数オプションにもあります。

```
>> RatExpr := (x^2 + 1) / (x - 1);


Warning: protected variable RatExpr overwritten
```

$$\frac{x^2 + 1}{x - 1}$$

逆に，識別子を保護しないようにもできます (None オプションを使います)。識別子の値が変更されたときに警告を出すよう，保護しておくこともできます (Warning オプションを使います)。

protected 関数はそのような変更を行う働きをします。オプションの一つを使って，ライトプロテクトの種類を定義してみましょう。

```
>> protected(hold(RatExpr), None): RatExpr := (x^2 + 1) / (x - 1);
```

$$\frac{x^2 + 1}{x - 1}$$

ライブラリ関数を使う際には問題になることもありますので，識別子のライトプロテクトの状態変更は用心深くやって下さい。

第2章　上級解析学

積分や式の単純化，方程式の求解といった問題への MuPAD の関数の使い方を幾つか示してみたいと思います。

もっと情報が欲しい時には「MuPAD 1.4 デモンストレーションツアー」も参照してみて下さい。

2.1　式の操作

セッションを初期化してください。

```
>> reset():
```

simplify 関数は，任意の代数表現を簡略化することができます。例えば，$\sqrt{127} - \sqrt[12]{127^6} = 0$ であることを示すには次のようにします。

```
>> simplify( sqrt(127) - ((127)^6)^(1/12) );
```

```
                                  0
```

あるいは，$\sqrt{14+3\sqrt{3+2\sqrt{5-12\sqrt{3-2\sqrt{2}}}}}$ といった表現を簡単に略にすることができます。

```
>> simplify(
sqrt( 14 + 3 * sqrt(3 + 2 * sqrt(5 - 12 * sqrt(3 - 2 * sqrt(2)))) )
, sqrt);
```

```
                                 1/2
                                2    + 3
```

また，この関数は指数関数を含んだ表現については特別な簡略化を行います。例えば $\frac{e^x+1}{e^{2x}-1}$ は次のようになります。

```
>> simplify( (exp(x) + 1) / (exp(2 * x) - 1), exp);
```

```
                                   1
                              ----------
                              exp(x) - 1
```

代数表現を展開するには，expand 関数を使って下さい。ここで，$\frac{\cos(5x)}{\sin(2x)\cos(x)^2}$ に適用してみます。

```
>> expand( cos(5 * x) / (sin(2 * x) * cos(x)^2) );
```

$$\frac{\cos(x)^2}{2\sin(x)} - 5\sin(x) + \frac{5\sin(x)^3}{2\cos(x)^2}$$

ここでは，$\sin(n\ x)$ と $\cos(m\ x)$ $(m, n$ は共に整数$)$ という関数が，和の形に展開されました。

有理式の標準形を求めるときには，`normal` 関数が使えるでしょう。$\frac{\sin(x)^2-(e^x)^2}{\sin(x)^2+2\sin(x)e^x+(e^x)^2}$ を例に考えて下さい。

```
>> normal( (sin(x)^2 - exp(x)^2) / (sin(x)^2 + 2 * sin(x) * exp(x) + exp(x)^2) );
```

$$\frac{\sin(x) - \exp(x)}{\sin(x) + \exp(x)}$$

$\frac{\sin(2x)-5\sin(x)\cos(x)}{\sin(x)(1+\tan(x)^2)}$ という表現を，sin と cos の項で表わしたいときには，`combine` 関数を使って下さい。

```
>> e := (sin(2 * x) - 5 * sin(x) * cos(x)) / (sin(x) * (1 + tan(x)^2));
```

$$\frac{\sin(2\ x) - 5\cos(x)\sin(x)}{\sin(x)\,(\tan(x)^2 + 1)}$$

```
>> combine( normal(expand(e)), sincos);
```

$$-\frac{9\cos(x)}{4} - \frac{3\cos(3\ x)}{4}$$

2.2 自動的に式を簡略化する際の留意点

MuPAD は ∞ が普通に計算できる数として認識しています。

```
>> 2 * infinity + PI;
```

$$\mathrm{infinity}$$

```
>> bool( 2 > -infinity);
```

$$\mathrm{TRUE}$$

但し，以下の計算は定義されていません[1]。

```
>> infinity / infinity;
```

```
undefined
```

```
>> infinity - infinity;
```

```
undefined
```

しかし，ある種の式の簡略化は自動的に行われません。例えば，以下のようなものです。

```
>> (-1)^(1/3);
```

```
   1/3
(-1)
```

この場合は，rectform 関数を使って，複素解の主値 $\frac{1}{2} + \frac{\sqrt{3}}{2}i$ を得ることができます。

```
>> rectform( % );
```

```
      /  1/2 \
      | 3    |
1/2 + | ---- | I
      \  2   /
```

式の簡略化は，複素数体上では一般に有効になっておらず，働きません。例えば，次のような例があります[2]。

```
>> sqrt( x^2 );
```

```
  2 1/2
(x )
```

```
>> ln( exp(x) );
```

```
ln(exp(x))
```

ある特定の性質 (「実数である」,「正の数である」...) を持つ未知数を含んだ式に対する簡略化は，simplify 関数と共に assume 関数を使って手助けできます。最後の例では，x が実数値を持っていると表現することで，式の簡略化を行っています。

```
>> assume( x, Type::RealNum ): simplify( ln( exp(x) ) );
```

```
x
```

[1] (訳注) 思いっきり意訳。原文は"and"だけ。

[2] (訳注) 特に指定しない識別子は複素数という扱いになると，私は理解しました。従って，この場合は自動的に式の簡略化を行ってくれない，ということなんでしょう。

2.3 多項式の因数分解

セッションを初期化して下さい。

```
>> reset();
```

以下のステートメントは Q 上の多変数多項式を因数分解しています。

```
>> factor( 7 * x^3 + 7 * x^2 * y + 3 * x * y + 3 * y^2 );

                                          2
                    [1, x + y, 1, 3 y + 7 x , 1]
```

この結果は，$7x^3 + 7x^2y + 3xy + 3y^2$ が体 Q 上で $1 \cdot (x+y)^1 \cdot (3y+7x^2)^1$ と分解されることを意味しています。

Factor 関数は多項式の因数分解をより自然な形で行ってくれます。

```
>> Factor( x^100 - 1 );

                 2            2    3    4        5    10    15    20
(x - 1) (x + 1) (x  + 1) (x + x  + x  + x  + 1) (x  + x   + x   + x   + 1)

    2        3    4        4    2    6    8
  (x  - x - x  + x  + 1) (x  - x  - x  + x  + 1)

    10    5    15    20        20    10    30    40
  (x   - x  - x   + x   + 1) (x   - x   - x   + x   + 1)
```

以下のステートメントでは，有限体 Z_5 上で多項式を因数分解をしています。

```
>> factor( poly( x^4 - 3 * x^2 + 1, [x], IntMod(5) ) );

   [1, poly(x + 2, [x], IntMod(5)), 2, poly(x - 2, [x], IntMod(5)), 2]
```

$x^4 - 3x^2 + 1$ は Z_5 上で $1 \cdot (x-2)^2 \cdot (x+2)^2$ と因数分解されることが分かります。

poly 関数はユーザに変数と係数のドメインを限定させることができます。そうすることで，多項式としてのデータ型が生成され，効率的な多項式計算が可能になります。

多項式と因数分解の詳細及び例題は第 3 章をご覧下さい。

2.4 微分と積分

セッションを初期化して下さい。

```
>> reset();
```

数式処理システムで重要なのは，微分・積分を記号操作だけで行えるという便利さにあります。

```
>> int( exp(x^2 / 2), x);
```

$$-\frac{1}{2}\,I\,(2\,PI)^{1/2}\,\mathrm{erf}\left(\frac{1}{2}\,I\,x\,2^{1/2}\right)$$

```
>> simplify( diff( %, x ) );
```

$$\exp\left(\frac{x^2}{2}\right)$$

以下は，Trager-Rothstein アルゴリズムを使って計算されたものです。このアルゴリズムは不定積分を代数方程式の根を引数に持つ対数関数の和の形で表わすものです。

```
>> int( 1 / (x^5 - x - 1), x );
```

$$\mathrm{sum}\left(X73\,\ln\left(x - \frac{309\,X73}{625} - \frac{21716\,X73^2}{625} - \frac{45904\,X73^3}{625} - \frac{183616\,X73^4}{625} - 256/625\right),\ X73 = \mathrm{RootOf}(15\,X74^5 - 80\,X74^3 + 160\,X74^2 + 2869\,X74 - 1)\right)$$

和の中の `X15` という識別子は根の集まりを表わしており，このうち一つは実数になります。これを検算すると，`diff` 関数が和も扱えるということが分かります。

```
>> diff( %, x );
```

$$\frac{1}{x^5 - x - 1}$$

定積分は次のようにして実行できます。

```
>> int( sin(p * x / l)^2, x = 1/2..1 );
```

$$\frac{l\,\sin\left(\frac{p}{l}\right)}{4\,p} - \frac{l\,\sin\left(\frac{2\,p}{l}\right)}{4\,p} + 1/4$$

```
>> int( sin(x)/x, x = 0..infinity );
```

$$\frac{\pi}{2}$$

あるいは

```
>> int( x^2 / (1 + x^3)^(3/2), x = 0..2 );
```

$$\frac{4}{9}$$

とします。数値積分についてはヘルプの `numeric::fint` 関数の項を参照して下さい。

2.5 有限和と無限和

セッションを初期化して下さい。

```
>> reset();
```

前の節の例で見てきたように，MuPAD は特殊関数の和を計算することができます。$\sum_{k=1}^{\infty}(1/k^2 + 1/k^3)$ の閉形式は次のように与えられます。

```
>> sum( 1/k^2 + 1/k^3, k = 1..infinity );
```

$$\zeta(3) + \frac{\pi^2}{6}$$

以下の例では，Gosper のアルゴリズムで超幾何関数の和が計算に使われています。

```
>> setuserinfo( Sum, 2 ): sum( binomial(n + k, k), k);
entering indefinite summation
enter Gosper
f(k)*f(k + (-1))^(-1) = k^(-1)*(k + n)
input is hypergeometric
case 2a of Gosper's algorithm
d=, 0
number of equations is, 1
solution is, {F[0] = (n + 1)^(-1)}
```

$$\frac{k\ \mathrm{binomial}(k + n, k)}{n + 1}$$

この例でも分かるように，setuserinfo 関数もまた第一引数にドメインを取ります。setuserinfo 関数を呼び出すと，ドメインで定義されている全ての関数にだけ影響を与えます。

最後ｎ sum 関数の例を以下に示します。

```
>> sum( 1 / (n^2 + 21 * n), n = 1..infinity );

                                18858053/108636528
```

ここでは Abramov のアルゴリズムが使われています。

2.6 級数展開と極限計算

セッションを初期化して下さい。

```
>> reset();
```

より高等な解析学の重要な部分は，級数展開と極限計算から成り立っています。MuPAD では，Taylor 展開, Laurent 展開, Puiseux 展開，その他の一般級数展開[3]が行えます。

```
>> s := series( sin(x), x = 0);

                                    3    5
                                   x    x        6
                              x - -- + --- + O(x )
                                   6    120

>> t := series( cos(x), x = 0);

                                     2    4
                                    x    x       6
                               1 - -- + -- + O(x )
                                    2    24

>> s^2 + t^2;

                                          6
                                     1 + O(x )

>> s / t = series( tan(x), x = 0);

                      3      5                3      5
                     x    2 x       6        x    2 x       6
                x + -- + ---- + O(x ) = x + -- + ---- + O(x )
                     3     15               3     15
```

[3](訳注)"general series expansion"を取りあえずこう訳してあります。

Puiseux 展開が存在しない場合の表現は次のようになります。

```
>> series( sin(1 / x + exp(-x)) - sin(1 / x), x = infinity);
```

$$\exp(-x) - \frac{\exp(-x)}{2} + \frac{\exp(-x)}{4} + O\left(\frac{\exp(-x)}{6}\right)$$
$$2 \qquad 24$$
$$x \qquad x \qquad \qquad x$$

このような場合，series 関数は一般級数展開を試みます。この表現は gseries というデータ型になります。

```
>> domtype( % );
```

gseries

一般級数展開は，asympt 関数を使って直接実行することができます。

```
>> asympt( exp(sin(1 / x + exp(-exp(x)))) - exp(sin(1 / x)), x, 2);
```

$$\exp(-\exp(x)) + \frac{\exp(-\exp(x))}{x} + O\left(\frac{\exp(-\exp(x))}{2}\right)$$
$$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad x$$

この関数は，D.Gruntz が 1995 年に開発したアルゴリズムに基づいており，MuPAD では limit 関数で作られています。

```
>> limit( sin(x) / x, x = 0 );
```

1

これは記号的に極限計算をする最新最良のアルゴリズムとして知られており，サブ表現の最速変化の概念[4]に基づいています。以下がその好例です。

```
>> limit(
x * ln(x) * (ln(x * exp(x) - x^2))^2 / ln(ln(x^2 + 2 * exp(exp(3 * x^3 * ln(x))))),
x = infinity );
```

1/3

2.7 方程式と連立方程式を解く

セッションを初期化して下さい。

```
>> reset();
```

[4](訳注)most-rapidly-varyingってどう訳すんでしょう？

以下は連立一次方程式の例です。

```
>> solve( { x + y + z = 2, 2 * x - y = 3 - z, x + z = -y + 2}, {x, y, z} );

                    { {           z            2 z } }
                    { { y = 1/3 - -, x = 5/3 - --- } }
                    { {           3             3  } }
```

solve 関数は解集合を返します。

```
>> solve( {x^2 + y^2 = 1, a * x + b * y = 0}, {x, y} );

{ --                   4    2  2 1/2 --
{ |         b y      (a  + a  b )     |
{ |  x = - ---, y = ---------------  |,
{ |          a            2    2     |
{ --                     a  + b      --

   --                     4    2  2 1/2 -- }
   |         b y         (a  + a  b )     | }
   |  x = - ---, y = - ---------------  | }
   |          a              2    2     | }
   --                       a  + b      -- }
```

以下のように，RootOf 関数は，5 次以下の既約多項式の解を表わす目的で使用されます。

```
>> solve( x^6 - 3 * x^2 + 5 * x, x );

                                          5
                       {0, RootOf(- 3 x + x  + 5 )}
```

対数関数や三角関数，その他の関数を含んだ方程式でも，solve 関数で解くことができます。

```
>> solve( sqrt(ln(x))  = ln(sqrt(x)), x );

                                {1, exp(4)}

>> s := solve( cos(x) / sin(x) = 0, x );

                          { / PI  3 PI    PI      \ }
                          { | --, ----, - --, ... | }
                          { \ 2    2      2       / }
```

この出力結果は解の離散集合を表わしています。

```
>> domtype( op(s, 1) );

                            Dom::DiscreteSet
```

以下のように，不等式も解くことができます。

```
>> solve( abs(x - 1) < 2, x );
```

```
                              {[-1 < x, x < 3]}
```

2.8 常微分方程式を解く

セッションを初期化して下さい。

```
>> reset();
```

MuPAD には ode という関数があり，ユーザは常微分方程式を定義できます。常微分方程式を解く前に，まずこの関数を使って常微分方程式を定義します。
$y'(x) + y(x) = \sin(x)$ という方程式を考えてみましょう。

```
>> eq1 := ode( y'(x) + y(x) = sin(x), y(x) );
```

```
                  ode(y(x) + diff(y(x), x) = sin(x), y(x))
```

この後，既出の solve 関数を使って方程式を解くことができます。

```
>> solve( % );
```

```
                      { sin(x)   cos(x)              }
                      { ------ - ------ + C1 exp(-x) }
                      {   2        2                 }
```

MuPAD は一階の非斉次線型常微分方程式も認識してくれます。対応する斉次方程式の解は，複素定数を表わす識別子を使って表現されています。

```
>> solve( ode( y''''(x) - 5 * y''(x) + 4 * y(x) = 2 * cos(x), y(x) ) );
```

```
    { cos(x)                                                      }
    { ------ + C2 exp(x) + C3 exp(-x) + C4 exp(2 x) + C5 exp(-2 x) }
    {   5                                                         }
```

連立の常微分方程式も解くことができます。例として次のものを考えます。

$$\begin{aligned} f'(x) - f(x) + g'(x) + 2g(x) &= 1 + e^x \\ g'(x) + 2g(x) + h'(x) + h(x) &= 2 + e^x \\ f'(x) - f(x) + h'(x) + h(x) &= 3 + e^x \end{aligned}$$

この解は次のようになります。

```
>> solve( ode(
{f'(x) - f(x) + g'(x) + 2 * g(x) = 1 + exp(x),
 g'(x) + 2 * g(x) + h'(x) + h(x) = 2 + exp(x),
 f'(x) - f(x) + h'(x) + h(x) = 3 + exp(x)}, {f(x), h(x), g(x)}) );

{ {        exp(x)                                      exp(x) ln(exp(x))
{ { g(x) = ------ + C8 exp(-2 x), f(x) = C7 exp(x) + ----------------- - 1,
{ {          6                                              2

          exp(x)                   } }
   h(x) = ------ + C6 exp(-x) + 2 } }
            4                      } }
```

2.9 その他の例

複素数を扱う関数もあります。例えば，表現の実数部分と虚数部分を求めるには次のようにします。

```
>> Re( sin(2 + 3 * I) + exp(2 - I) + (2 + 3 * I)^(1/3) );

                                          1/6    / atan(3/2) \
         cos(1) exp(2) + sin(2) cosh(3) + 13    cos| --------- |
                                                   \     3     /
```

rectform 関数は表現 f に対して，$\mathcal{R}(f) + i\mathcal{I}(f)$ を計算します。f の未知数は複素数として解釈されます。

```
>> lm( rectform( cos(z) + I * sin(z) ) );

            sin(Re(z)) cosh(Im(z)) - sin(Re(z)) sinh(Im(z))
```

更に，rectform 関数は実数部も求めることができます。

```
>> Re( rectform( cos(z) + I * sin(z), {z} ) );

                                cos(z)
```

contfrac 関数は連分数形式を求めます。

```
>> cPI := contfrac(PI, 5);

                          1
                ----------------------- + 3
                          1
                ------------------- + 7
                        1
```

```
---------------- + 15
     1
----------- + 1
  1
----- + 291
"..."
```

$\sqrt{2}$ の連分数形式の最初の部分を計算します。

```
>> cSqrt2 := contfrac( sqrt(2), 2 );
```

```
                1
----------------- + 1
      1
------------- + 2
    1
--------- + 2
  1
----- + 2
"..."
```

また，普通に連分数同士の計算ができることも見ておくといいでしょう。例えば $\pi/\sqrt{2}$ の連分数形式を求めるには，既に計算済みの連分数形式のまま除算をしてやればいいのです。

```
>> cPI / cSqrt2;
```

```
    1
--------- + 2
  1
----- + 4
"..."
```

第3章　多項式

セッションを初期化して下さい。

```
>> reset():
```

3.1　多項式のデータ型

MuPAD は，多項式演算の基礎となる独自のデータ型を提供しています。これによって，効率的に演算ができ，多項式をより自然に扱うことができます。

多項式表現を使う関数は内部でこのデータ型を使用しています。このデータ型の多項式は poly 関数で生成されます。

```
>> poly( x^2 + 2 * x + 1 );
```

```
                              2
                poly(x  + 2 x + 1, [x])
```

多変数多項式と他の環上の多項式は同じように構成されています。x と y の多項式は次のようになります。

```
>> poly( x^2 * y^2 + x * y + 1 );
```

```
                            2  2
                poly(x  y  + x y + 1, [x, y])
```

次の多項式の表現は自動的に展開されています。

```
>> poly( x * (2 * x + y + 2)^2 + 2 );
```

```
                3      2        2      2
       poly(4 x  + 4 x  y + 8 x  + x y  + 4 x y + 4 x + 2, [x, y])
```

poly 関数では，変数を明示的に与えることができますから，変数の順番を変えることも可能です。

```
>> poly( x^2 * y^2 + x * y + 1, [y, x] );
```

```
                            2  2
                poly(y  x  + y x + 1, [y, x])
```

剰余類 Z/Z_7 の係数を持つ x, y の多項式は次のようになります。

```
>> poly( x^2 * y^2 + 6 * x * y - 2 * x - 5, IntMod(7) );

                2  2
         poly(x  y  + (-1) x y + (-2) x + 2, [x, y], IntMod(7))
```

ユーザが定義したドメインも係数環として使用することが可能です。
演算子+, -, *, ^は多項式に対しても普通に使用できます。

```
>> a := poly(2 * x^2 - 4 * x * y - 2 * x + 4 * y, [x, y], IntMod(17));
   b := poly(x^2 * y - 2 * x * y^2, [x, y], IntMod(17) );

                 2
         poly(2 x  + (-4) x y + (-2) x + 4 y, [x, y], IntMod(17))

                    2             2
                poly(x  y + (-2) x y , [x, y], IntMod(17))

>> a + b; a * b;

      2        2           2
poly(x  y + 2 x  + (-2) x y  + (-4) x y + (-2) x + 4 y, [x, y], IntMod(17))

        4           3  2         3        2  3      2  2           3
poly(2 x  y + (-8) x  y  + (-2) x  y + 8 x  y  + 8 x  y  + (-8) x y ,

   [x, y], IntMod(17))
```

Z_{17} 上の 2 つの多項式の最大公約数を求めてみます。

```
>> gcd(a, b);

                  poly(x + (-2) y, [x, y], IntMod(17))
```

最小公倍数は次のようになります。

```
>> lcm(a, b);

            3           2  2         2          2
    poly(2 x  y + (-4) x  y  + (-2) x  y + 4 x y , [x, y], IntMod(17))
```

多項式を再帰的に表現することも可能です。以下では，Poly 関数を使って作られた多項式ドメインを係数ドメインとして使用されています。

```
>> a := poly( 5 * x^2 * y^2 + 2 * y + 3, [x], Poly([y]) );

                    2   2
             poly((5 y ) x  + (2 y + 3), [x], Poly([y], Expr))
```

以下のように，係数が y の多項式になっています。

```
>> coeff( a );
```

```
                              2
                  poly(5 y , [y]), poly(2 y + 3, [y])
```

終わりに, 多項式分解の例をお見せしましょう。与えられた多項式p に対して, $p(x) = q_1(q_2(...(q_n(x))...))$ となる多項式列 $q_1, ..., q_n$ を以下で示します。

```
>> decompose( x^6 + 6 * x^4 + x^3 + 9 * x^2 + 3 * x - 5 );
```

```
                                2             3
                           x + x  - 5, 3 x + x
```

```
>> decompose( poly(x^4 - 3 * x^3 - x + 5, [x], IntMod(5)) );
```

```
           2                                    2
     poly(x  + (-1) x, [x], IntMod(5)), poly(x  + x, [x], IntMod(5))
```

3.2 多項式の因数分解

以下は，有理数体上の多変数多項式を因数分解しています。

```
>> factor( 6 * x^2 + 18 * x ^ 24 );
```

```
                                    22
                        [6, x, 2, 3 x   + 1, 1]
```

```
>> factor( 4 * x^3 + 4 * x^2 * y + 3 * x * y + 3 * y^2 );
```

```
                                          2
                     [1, x + y, 1, 3 y + 4 x , 1]
```

Z 上で因数分解します。

```
>> factor( x^3 + 1 );
```

```
                                  2
                     [1, x + 1, 1, x  - x + 1, 1]
```

```
>> P := x^8 + 8 * x^7 + 28 * x^6 + 56 * x^5 + 70 * x^4 + 56 * x^3 + 28 * x^2
 + 8 * x^2 + 8 * x + 1;
```

```
                2       3       4       5       6      7    8
```

```
        8 x + 36 x  + 56 x  + 70 x  + 56 x  + 28 x  + 8 x  + x  + 1

>> factor(P);

                  2       3       4       5       6      7    8
    [1, 8 x + 36 x  + 56 x  + 70 x  + 56 x  + 28 x  + 8 x  + x  + 1, 1]
```

従って，この多項式 p は Q 上で因数分解できません。

第4章　線型代数

この章では,「デモンストレーションツアー」にある行列や線型代数に関する機能の紹介を更に広く行います。

4.1　行列の演算

セッションを初期化して下さい。

```
>> reset():
```

次のステートメントでは，多項式環上で 3 次の正方行列環を生成しています。

```
>> export(linalg):
>> MP := Dom::SquareMatrix( 3, Dom::Polynomial(Dom::Integer) );

            Dom::SquareMatrix(3, Dom::Polynomial(Dom::Integer))

>> MP::hasProp( Cat::Ring ), MP::hasProp( Cat::CommutativeRing );

                                  TRUE, FALSE
```

カテゴリにおける Cat::Ring や Cat::CommutativeRing と同様，ドメインコンストラクタ Dom::SquareMatrix, Dom::DistributedPolynomial，Dom::Integer は domains パッケージの一部であり，それぞれ順に Ax，Cat，Dom ライブラリの中に見つけることができます。

では，行列を生成して計算ができることを見ていきます。

```
>> A := MP( [[0, y, 1], [0, x^2, 0], [y^3, 0, y^5]] );

                                +-            -+
                                |   0,  y,  1   |
                                |               |
                                |       2       |
                                |   0, x ,  0   |
                                |               |
                                |   3       5   |
                                |  y ,  0, y    |
                                +-            -+

>> B := MP( [x, 1, y], Diagonal );
```

```
+-         -+
|  x, 0, 0  |
|           |
|  0, 1, 0  |
|           |
|  0, 0, y  |
+-         -+
```

行列演算は MuPAD の標準演算子を使って行うことができます。

```
>> A * B - B * A;
```

```
+-                                  -+
|        0,      y - x y, - x + y    |
|                                    |
|        0,         0,        0     |
|                                    |
|     4      3                       |
|  - y  + x y   ,    0,        0     |
+-                                  -+
```

```
>> ( A + B )^2;
```

```
+-                                                  -+
|       2    3               2                  5    |
|      x  + y ,    y + x y + x  y,      x + y + y    |
|                                                    |
|                     2    4                         |
|         0,        2 x  + x  + 1,          0       |
|                                                    |
|   4    8      3         4         2    3      6    10  |
|  y  + y  + x y ,       y ,       y  + y  + 2 y  + y    |
+-                                                  -+
```

この行列 A は以下で分かるように正則ではありません。

```
>> 1 / A;
```

```
FAIL
```

が，行列 A が有理数体で定義されていれば

```
>> Af := Dom::Matrix(Dom::Fraction(A::coeffRing))( A );
```

```
+-           -+
```

```
|   0,  y,  1  |
|              |
|       2      |
|   0, x ,  0  |
|              |
|   3       5  |
|  y ,  0, y   |
+-            -+
```

逆行列を計算することができます。

```
>> Af^(-1);
```

```
+-                  -+
|           3        |
|     2   y    1     |
|  - y ,  --,  --    |
|          2    3    |
|         x    y     |
|                    |
|         1          |
|    0,   --,   0    |
|          2         |
|         x          |
|                    |
|          y         |
|    1,  - --,  0    |
|           2        |
|          x         |
+-                  -+
```

結果を確認してみましょう。

```
>> Af * %, % * Af;
```

```
+-          -+  +-          -+
|  1, 0, 0   |  |  1, 0, 0   |
|            |  |            |
|  0, 1, 0   |, |  0, 1, 0   |
|            |  |            |
|  0, 0, 1   |  |  0, 0, 1   |
+-          -+  +-          -+
```

今度は，任意の演算表現[1]上で行列を定義してみましょう。これは行列の標準的な係数ドメインでもあります。

[1] (訳注)arbitrary arithmetical expressions のこと。

```
>> M := Dom::Matrix();
```

```
Dom::Matrix(Dom::ExpressionField(id, iszero))
```

行列の exponential を計算したいのであれば

```
>> A := M( [[-13, -10], [21, 16]] );
```

```
+-          -+
|  -13, -10  |
|            |
|   21,  16  |
+-          -+
```

とし，次のステートメントを打ち込んで下さい。

```
>> exp( A, t );
```

```
+-                                                          -+
|    15 exp(t) - 14 exp(2 t),    10 exp(t) - 10 exp(2 t)     |
|                                                            |
|  - 21 exp(t) + 21 exp(2 t) , - 14 exp(t) + 15 exp(2 t)     |
+-                                                          -+
```

同様に，行列 A のノルムも簡単に計算できます。例として無限大 (∞) ノルムと Frobenius ノルムを計算してみます。

```
>> norm( A ), norm( A, Frobenius );
```

```
                                  1/2
                 37, 966
```

4.2 linalg ライブラリパッケージ

セッションを初期化して下さい。

```
>> reset();
```

linalg ライブラリは，MuPAD で線型代数を行うための沢山の関数から構成されています。その全関数を export してみましょう[2]

```
>> export(linalg):
```

有理数体 Q 上で行列を定義してみましょう。

[2](訳注) 原文では trace 関数について Warning が出ていますが，1.4.2 では出ていません。原文でも特にこの Warning の説明はないし，なんなんでしょ？

```
>> A := Dom::Matrix(Dom::Rational)( [[1, -3, 3], [3, -5, 3], [6, -6, 4]] );
```

```
                              +-          -+
                              |  1, -3, 3  |
                              |            |
                              |  3, -5, 3  |
                              |            |
                              |  6, -6, 4  |
                              +-          -+
```

この行列の固有値・固有ベクトル[3]は以下のように与えられ,

```
>> eigenVectors( A );
```

```
  -- --         -- +-   -+  +-    -+ -- --  --        -- +-      -+ -- -- --
  |  |          |  |  1  |  |  -1  |  |  |  |        |  |  1/2  |  |  |  |  |
  |  |          |  |     |  |      |  |  |  |        |  |       |  |  |  |  |
  |  |  -2, 2,  |  |  1  |, |   0  |  |  |, |  4, 1, |  |  1/2  |  |  |  |  |
  |  |          |  |     |  |      |  |  |  |        |  |       |  |  |  |  |
  |  |          |  |  0  |  |   1  |  |  |  |        |  |   1   |  |  |  |  |
  -- --         -- +-   -+  +-    -+ -- --  --        -- +-      -+ -- -- --
```

A を対角化すると次のような行列になります。

```
>> jordanForm(A);
```

```
                              +-           -+
                              |  -2,  0, 0  |
                              |             |
                              |   0, -2, 0  |
                              |             |
                              |   0,  0, 4  |
                              +-           -+
```

これは，既出の A の固有値・固有ベクトルで決定されるものです。

可換環上の行列の行列式を計算するちょっとした例を示します。

```
>> A := randomMatrix(
2, 2, Dom::DistributedPolynomial([x], Dom::IntegerMod(6))
);
```

```
            +-                                              -+
            |           3      4      5       2    3         |
            |  - 2 x + x  + 2 x  + 2 x  ,    x  + x  - 2     |
            |                                                 |
```

[3](訳注)eigensystem の和名ってなんでしょ？「固有システム」だと違和感があるなあ … 。

```
+-                                                    -+
|        2    3    5                    3             |
|     - x  + x  - x  + 1 ,    - 2 x - 2 x  - 2         |
+-                                                    -+
```

この出力結果はあなたのものとは違っているはずです。この行列はランダムに生成していますからね。

```
>> det( A );
```

$$-2x^8+x^7-x^6-x^5+2x^4-x^3+3x^2+3x+2$$

二つの多項式の終結式[4]を求める簡単な (勿論，効率は良くありませんが) 方法があります。

```
>> f := poly( 1 + 2 * x + 10 * x^2, IntMod(7) );
   g := poly( x^2 + 23 * x - 5, IntMod(7) );
```

$$\mathrm{poly}(3x^2+2x+1, [x], \mathrm{IntMod}(7))$$

$$\mathrm{poly}(x^2+2x+2, [x], \mathrm{IntMod}(7))$$

上の f, g に対応する Sylvester 行列の行列式を以下で計算しています。

```
>> det( sylvester(f, g, x));
```

$$3 \bmod 7$$

与えられた体上の一次独立なベクトル列から，linalg::scalarProduct 関数で定義されるスカラー積の意味で直交なベクトルの集合を計算することが可能です。

次の例題を考えます。

> $[0,1]$ 区間での連続関数の線型空間での内積が以下のように定義されている。
>
> $$(f,g)=\int_0^1 f(t)g(t)dt$$
>
> 多項式 $1, t, t^2, t^3, \ldots$ の直交系を計算せよ。

linalg::scalarProduct 関数を次のように再定義します。

```
>> sysassign( linalg::scalarProduct, proc(f, g) local F, R, t;
   begin
       R := f::coeffRing;
       F := int( expr(f[1]) * expr(g[1]), t = 0..1 );

       R(F)
end_proc):
```

[4] (訳注)って何だ？

識別子 linalg::scalarProduct は通常ライトプロテクトがかけられていますから，このような割り当ては強制的に行わなければならないことを覚えておいて下さい。

そうしておいて，最初の 5 つの多項式を要素に持つ行列を生成します。

```
>> A := Dom::Matrix()( [[1, t, t^2, t^3, t^4]] );
```

```
                          +-       2   3   4-+
                          | 1, t, t , t , t  |
                          +-                -+
```

では，linalg::ogSystem 関数を使って，A の列方向に直交化を行いましょう。

```
>> S := ogSystem( col(A, 1..4) );
```

```
--                                          +-                          -+ --
|                                           |           2                |  |
|  +- -+  +-       -+  +-       2       -+ |  3 t   3 t     3           |  |
|  | 1 |, | t - 1/2 |, | - t + t  + 1/6  |, |  --- - ---- + t  - 1/20    |  |
|  +- -+  +-       -+  +-               -+ |   5     2                  |  |
--                                          +-                          -+ --
```

S のベクトルを次のようにして正規化します。

```
>> map( S, normalize );
```

```
--
|
|  +- -+  +-  1/2          -+  +-   1/2         2        -+
|  | 1 |, | 12    (t - 1/2) |, | 180    (- t + t  + 1/6 ) |,
|  +- -+  +-                -+  +-                        -+
--

   +-                                    -+ --
   |          /          2             \  |  |
   |      1/2 | 3 t   3 t     3         |  |  |
   |  2800    | --- - ---- + t  - 1/20 |  |  |
   |          \  5     2               /  |  |
   +-                                    -+ --
```

最後に，行列の Cholesky 分解を計算する例を示します。加えて行列が正定値であることもわかりますね。[5]。

```
>> S := Dom::Matrix(Dom::Rational)(
[[4, -2, 4, 2], [-2, 10, -2, -7], [4, -2, 8, 4], [2, -7, 4, 7]]
);
```

[5](訳注) 下三角行列の対角成分が全て正だからですね。

```
+-                  -+
|   4, -2,  4,  2   |
|                   |
|  -2, 10, -2, -7   |
|                   |
|   4, -2,  8,  4   |
|                   |
|   2, -7,  4,  7   |
+-                  -+
```

```
>> linalg::cholesky( S );
```

```
 +-               -+
 |   2,  0, 0, 0   |
 |                 |
 |  -1,  3, 0, 0   |
 |                 |
 |   2,  0, 2, 0   |
 |                 |
 |   1, -2, 1, 1   |
 +-               -+
```

第5章　グラフィックス

セッションを初期化して下さい。

```
>> reset():
```

5.1　二次元グラフィックス

曲線を描く自作プロシージャを使うには以下のようにします (図 5.1)。

```
>> my_curve_x := proc(u)
   begin
       sin(u) * sin(2 * u) * sin(3 * u):
   end_proc:

   my_curve_y := proc(u)
   begin
       sin(4 * u) * sin(5 * u) * sin(6 * u):
   end_proc:

   plot2d(Axes = Box, Ticks = 0,
       Scaling = UnConstrained,
       PointStyle = Circles, PointWidth = 6,
       [Mode = Curve,
           [hold(my_curve_x(u)),
            hold(my_curve_y(u))],
           u = [-PI, PI],
           Grid = [200],
           Style = [LinesPoints],
           Color = [Height, [1, 1, 0],
                            [1, 0, 1]]
       ]):
```

次の例では，曲線をいろいろなスタイルで描画できることが分かります。`[0, 2*PI]` 区間における `cos(f)` 曲線を描いてみます (図 5.2)。

```
>> plot2d(Labels = ["", ""], Labeling = TRUE,
          Title = "Different Curve Styles",
          Axes = Origin, AxesOrigin = [0, 0],
```

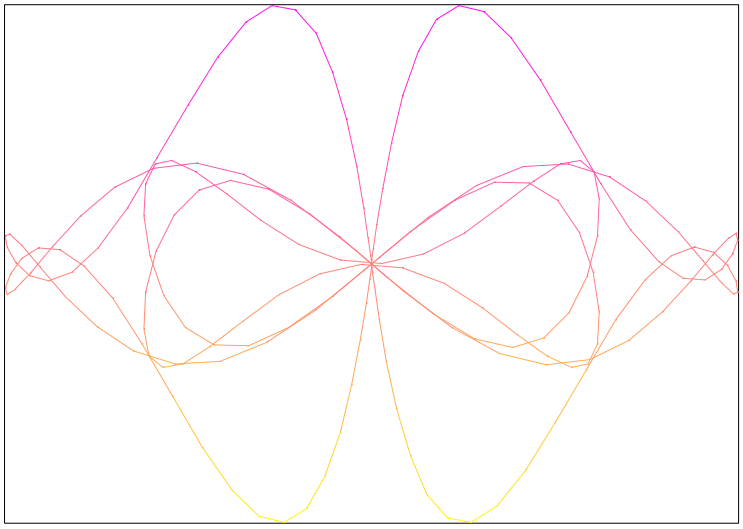

図 5.1: 自作プロシージャによる描画

```
[Mode = Curve,
     [u, 2 * cos(u)], u = [0, PI/2],
     Grid = [15], Style = [Points],
     Color = [Height, [1, 1, 0],
                      [1, 0, 0]],
     Title = "Points",
     TitlePosition = [2, 2.5]
],
[Mode = Curve,
     [u, 2 * cos(u)], u = [PI/2, PI],
     Grid = [15], Style = [Lines],
     Color = [Height, [0, 1, 1],
                      [0, 0, 1]],
     Title = "Lines",
     TitlePosition = [3.5, 5.5]
],
[Mode = Curve,
     [u, 2 * cos(u)], u = [PI, 3*PI/2],
     Grid = [15], Style = [LinesPoints],
     Color = [Height, [0, 1, 1],
                      [0, 0, 1]],
     Title = "LinesPoints",
     TitlePosition = [6, 7.5]
],
[Mode = Curve,
```

```
        [u, 2 * cos(u)], u = [3*PI/2, 2*PI],
        Grid = [15], Style = [Impulses],
        Color = [Height, [0, 0, 1],
                         [1, 0, 1]],
        Title = "Impulses",
        TitlePosition = [8, 4]
]);
```

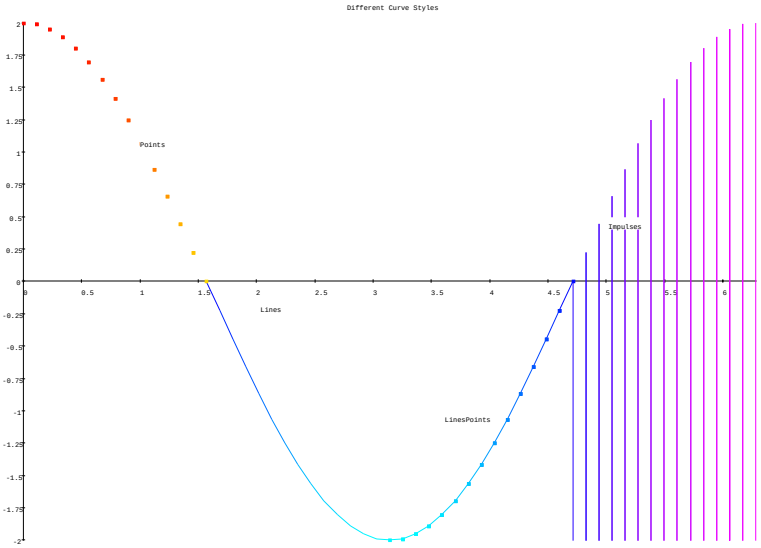

図 5.2: スタイルを変えて描画する

どちらも MuPAD のグラフィックツールである「VCam」を使えば，対話的に描画したり制御したりすることが出来るようになります。

5.2 三次元グラフィックス

この節では三次元グラフィックスの例を幾つか提示したいと思います。現在，MuPAD は空間曲線や曲面を描くことが出来るようになっています。

以下は，描画コマンドを使っていろいろな作業を行いつつ，その中で前景色と背景色を指定する方法の例です (図 5.3)。

```
>> plot3d(Axes = None, Ticks = 0,
          Scaling = UnConstrained,
          CameraPoint = [114, 0, 155],
          BackGround = [1, 1, 1],
          ForeGround = [0, 0, 1],
          [Mode = Surface,
                [4*v*cos(u)-v*cos(4*u),
```

```
 4*v*sin(u)+v*sin(4*u), -6*cos(v)],
u = [0, 2*PI], v = [0.1, 6],
Grid = [41, 20],
Style = [ColorPatches, AndMesh],
Color = [Height, [0.4, 0.4, 0.4],
                 [0, 1, 1]]]);
```

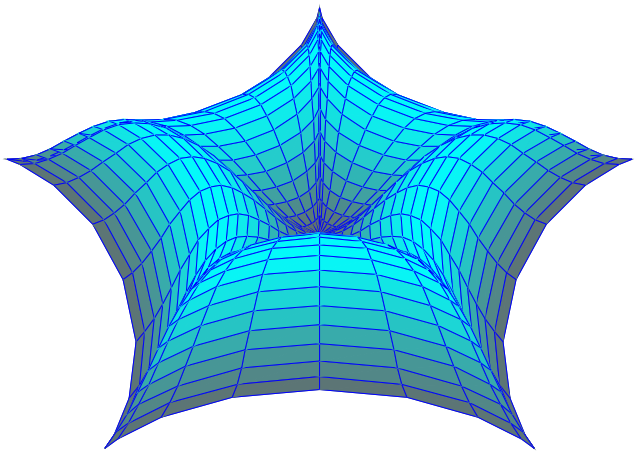

図 5.3: 三次元画像の例

次は，俗に Benjamin-小野方程式[1] と呼ばれているものの解を表わしている例です。この方程式は非線型波動現象の場に現れるものです。

benjamin-ono_2() 関数の三番目，四番目のパラメータ c1, c2 は, 相互に作用するソリトンの漸近速度を表わすものです (図 5.4)。

(注意) 計算に二分ほどかかります。

```
>> benjamin_ono_2 := proc(x, t, c1, c2)

   local e1, e2, f, fx;

   begin
       e1 := c1 * (x - c1 * t):
       e2 := c2 * (x - c2 * t):

       f  := 1 - e1 * e2 + 4 * c1 * c2 / (c1 - c2)^2 + I * (e1 + e2):
       fx := -c1 * c2 * (2 * x - c2 * t - c1 * t) + I * (c1 + c2):

       30 * abs(fx / f):
```

[1] (訳注) という方程式があるんですか？

```
   end_proc:

>> plot3d(Axes = None, Ticks = 0,
          Scaling = UnConstrained,
          Title = "Twosoliton of the B.O.",
          [Mode = Surface,
                [u, v,
                 hold(benjamin_ono_2(u, v, 1/4, 3/5))],
                 u = [-40, 40], v = [-30, 30],
                 Grid = [50, 50],
                 Smoothness = [1, 0],
                 Style = [ColorPatches, AndMesh],
                 Color = [Height, [0, 0, 1],
                                  [1, 1, 1]]
          ]):
```

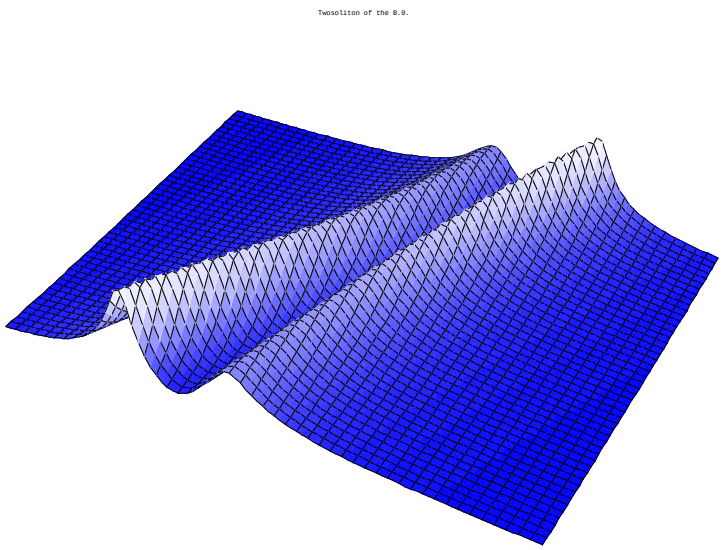

図 5.4: Benjamin-小野方程式の解

どちらの例も VCam を使って対話的に描画したり操作することが出来るということを覚えておいて下さい。

5.3 グラフィックの基本命令

グラフィックの基本命令を見ていくことにします。現行バージョンの MuPAD では，`point` 関数と `polygon` 関数という二つの異なる基本命令が使えるようになりました。これらの命令は図形を描くのに用いられます。

以下は，四面体の例です (図 5.5)。

```
>> tetra_hedron := proc(p1, p2, p3, p4)
   begin
       polygon(p1, p2, p4, Closed=TRUE, Filled=TRUE),
       polygon(p1, p3, p4, Closed=TRUE, Filled=TRUE),
       polygon(p2, p3, p4, Closed=TRUE, Filled=TRUE)
   end_proc:

   mid_point := proc(p1, p2)
       local x, y, z;
   begin
       x := op(p1,1) + (op(p2,1) - op(p1,1)) / 2.0:
       y := op(p1,2) + (op(p2,2) - op(p1,2)) / 2.0:
       z := op(p1,3) + (op(p2,3) - op(p1,3)) / 2.0:
       point(x, y, z):
   end_proc:

   tetra_rec := proc(p1, p2, p3, p4, n)
       local np1, np2, np3, np4, np5, np6;
   begin
       if n = 0 then
           tetra_hedron(p1, p2, p3, p4):
       else
           np1 := mid_point(p1, p2):
           np2 := mid_point(p2, p3):
           np3 := mid_point(p3, p1):
           np4 := mid_point(p1, p4):
           np5 := mid_point(p2, p4):
           np6 := mid_point(p3, p4):
           tetra_rec(p1, np1, np3, np4, n-1),
           tetra_rec(np1, p2, np2, np5, n-1),
           tetra_rec(np2, p3, np3, np6, n-1),
           tetra_rec(np4, np5, np6, p4, n-1):
       end_if:
   end_proc:

   a := point( 1.0, 0.0          ,      0.0):
   b := point(-0.7, 0.5*sqrt(3.0),      0.0):
   c := point(-0.7,-0.5*sqrt(3.0),      0.0):
   d := point( 0.1, 0.0          , sqrt(3.0)):

   plot3d(Axes = None,
          CameraPoint = [0.15, -0.1, 10.0],
          [Mode = List, [tetra_rec(a, b, c, d, 3)],
```

```
            Color = [Height]
        ]);
```

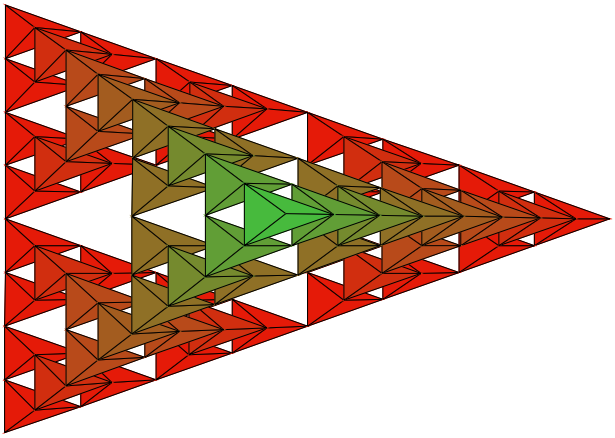

図 5.5: グラフィック基本命令の使用例

5.4 カラー関数を定義する

この節では，描画された物体に装飾を施す際の詳細，即ち，カラー関数の定義について解説します。カラー関数は座標ごとの物体の色分けや，関数のモジュラスによって記述される曲面への複素関数の Fatou 集合の射影等の色分けを行います。

次の例は，カラー関数を定義した後，位相 `Phi` の複素平面のモジュラスを色分けしています。これは，全ての複素数 `z` が `z = abs(z) exp(I*Phi)` と表わせることを利用しています。

(注意) 計算するのに大体 1.5 分かかります。

```
>> complex_surf := proc(re, im)
   begin
       1 / ((re + I * im)^3 + 1):
   end_proc:

   phase := proc(x_coord, y_coord, z_coord, u_val, v_val)
       local erg, real, imag, Phi;
   begin
       erg := complex_surf(x_coord, y_coord):
       real := op(erg, 1):
       imag := op(erg, 2):
       if abs(real) > EPS then
```

```
        Phi := atan(imag / abs(real)):
    else
        Phi := sign(imag) * PI / 2:
    end_if:
    if float(Phi) < 0 then
        value := (Phi - MIN_PHI) / (0 - MIN_PHI):
        [1-value, value, 0]:
    else
        value := (Phi) / MAX_PHI:
        [0, 1-value, value]:
    end_if:
end_proc:

MIN_PHI := float(-PI/2):
MAX_PHI := float( PI/2):
EPS     := 10^(-DIGITS):

plot3d(Axes = None, Scaling = UnConstrained,
       [Mode = Surface,
             [u, v, min(abs(complex_surf(u, v)), 2.0)],
             u = [-2, 2], v = [-2, 2],
             Style = [ColorPatches, AndMesh],
             Color = [Function, phase],
             Grid = [50, 50]
       ]):
```

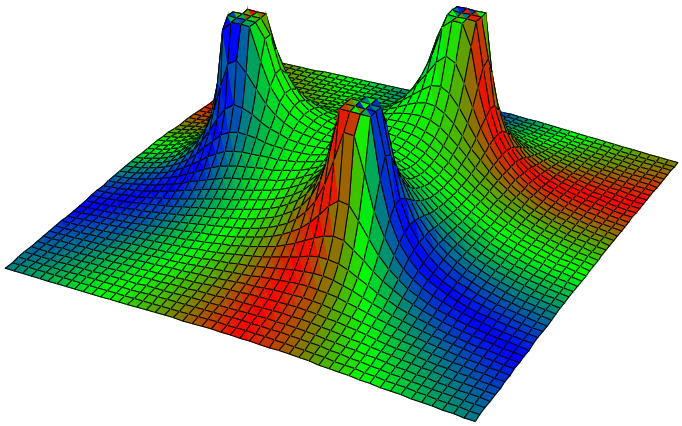

図 5.6: 複素表面のモジュラスに位相ごとの色づけを行う

第6章　多項式イデアルのGröbner基底

groebner パッケージは，多項式イデアルの Göbner 基底を計算するための関数を持つものです。この多項式の係数は，有理数か体となるドメインでなければなりません。

セッションを初期化し，groebner パッケージを export して下さい。

```
>> reset(): export(groebner):
```

6.1　groebner ライブラリパッケージ

info コマンドを使うと，パッケージ情報を表示できます。

```
>> info(groebner);
Library 'groebner': Calculation of Gr"obner-bases for polynomial ideals
Interface:
dimension, gbasis, normalf, spoly
```

spoly 関数は多項式 p_1 と p_2 の S 多項式[1]を計算します。

```
>> p1 := poly(x^2 * y + 5 * x^2 + y^2, [y, x]);

                          2      2      2
                    poly(y  + y x  + 5 x , [y, x])

>> p2 := poly(7 * x * y^2 - 2 * y^3 + 1, [y, x]);

                             3      2
                  poly((-2) y  + 7 y  x + 1, [y, x])

>> spoly(p1, p2);

                  4         2  3          2  2         2
       poly((-2) y  + (-7) y  x  + (-10) y  x  + (-1) x , [y, x])
```

この出力結果は，項の次数の合計が大きい順に計算されています。変数の辞書順に計算させることも可能です。

```
>> spoly(p1, p2, LexOrder);
```

[1] (訳注) 自信なし。

```
                    2  2         2              2
poly((-2) y  x  + (-7) y  x + (-10) y x  - 1, [y, x])
```

normalf 関数は，多項式イデアル P の多項式 q モジュロの正規形 (全体を約分して) を返します[2]。

例では，イデアル $P = (p_1, p_2)$ と約分される多項式 q を考えます。

```
>> q := poly(3 * x^3 * y + 2 * x^2 * y^2 - 3 * x * y + 5 * x, [y, x]);
```

```
                    2  2        3
             poly(2 y  x  + 3 y x  + (-3) y x + 5 x, [y, x])
```

すると，次数の合計順に並べられた q の正規形が，次のようにして与えられます。

```
>> normalf(q, [p1, p2]);
```

```
            2         2                      3       2
 poly((-10) y  x + 10 y  + (-3) y x + (-15) x  + 50 x  + 5 x - 1, [y, x])
```

約分された多項式イデアルの Gröbner 基底は，関数 gbasis で計算出来ます。イデアルは多項式のリストで与えられていなければなりません。

```
>> gbasis([p1, p2]);
```

```
        3         2                        2      2      2
[poly(2 y  + (-7) y  x - 1, [y, x]), poly(y  + y x  + 5 x , [y, x]),

           2            3        2
   poly(50 y  + y + 175 x  + 251 x  - 5, [y, x])]
```

Gröbner 基底は，変数の辞書順で計算することも可能です。また，未知数を見つけやすいよう，gbasis 関数に未知数の順を変更させることもできます。

```
>> gbasis([p1, p2], LexOrder, Reorder);
```

```
                6       5           4         3
[poly(175 x + 4 y  + 49 y  + (-345) y  + (-4) y  + 50 y + 1, [x, y]),

          7       6         4          3
   poly(4 y  + 69 y  + (-4) y  + (-20) y  + y + 5, [x, y])]
```

6.2 連立代数方程式の解

次の多項式を考えます。

[2] (訳注) 代数は苦手で … 。

```
>> p1 := poly(x + y * z - 2, [x, y, z]);

                        poly(x + y z - 2, [x, y, z])

>> p2 := poly(x * z + y - 3, [x, y, z]);

                        poly(x z + y - 3, [x, y, z])

>> p3 := poly(x * y + z - 5, [x, y, z]);

                        poly(x y + z - 5, [x, y, z])
```

連立代数方程式 $\{p_1 = 0, p_2 = 0, p_3 = 0\}$ が与えられたとき，一体，解は存在するのでしょうか？

解は幾つあるのでしょうか？　そんな疑問は，イデアル $P =< p_1, p_2, p_3 >$ の Gröbner 基底を与えることで解決できます。

基底が定数多項式 (並び順に関係なく) を含んでいなければ，解が存在する，ということはよく知られています。

それ故，まず最初に約分された基底を次数順に求めてみましょう。

```
>> B := gbasis([p1, p2, p3]):
   map(B, expr);

  3                      2                     2    2
[z  - 3 x - 2 y - z - 5 z  + 11, 5 z - 2 x + x  - z , z + x y - 5,

                2    2
   5 z - 3 y + y  - z , y + x z - 3, x + y z - 2]
```

更に，それぞれの不定式 X に対して，ある n に対し X^n という形になる Gröbner 基底の主要項が存在すれば，有限個の解しかないと結論付けられます。

基底 B の主要項はこの仮定を満足しています。

```
>> map(B, lterm, DegreeOrder):
   map(%, expr);

                          3   2        2
                        [z , x , x y, y , x z, y z]
```

$\{p_1 = 0, p_2 = 0, p_3 = 0\}$ という型の連立代数方程式の他の例について考えてみます。

```
>> p1 := poly(x^2 * y + 4 * y^2 - 17, [x, y]);

                            2        2
                        poly(x  y + 4 y  - 17, [x, y])

>> p2 := poly(2 * x * y - 3 * y^3 + 8, [x, y]);
```

```
                                    3
                    poly(2 x y + (-3) y  + 8, [x, y])
```

```
>> p3 := poly(x * y^2 - 5 * x * y + 1, [x, y]);
```

```
                                2
                    poly(x y  + (-5) x y + 1, [x, y])
```

項の並びに関係なく，この方程式は解を持ちません。

```
>> gbasis([p1, p2, p3]):
   map(%, expr);
```

```
                                  [1]
```

```
>> gbasis([p1, p2, p3], LexOrder):
   map(%, expr);
```

```
                                  [1]
```

第7章　ドメイン—ユーザ定義のデータ型

MuPADで提供されている基本データ型とは別に，ユーザは新たなデータ型を生成することができます。

7.1　Z/Z_7体

まず，セッションを初期化して下さい。

```
>> reset():
```

以後，体Z_7を定義し，新たにユーザ定義ドメインとします。そのようなデータ型は，関数domainを使って生成します。

```
>> Z7 := domain("Z7"):
```

この関数の引数はドメインのkeyと呼ばれます。keyはドメインごとに固有のものでなければなりません。
new関数を使ってZ7ドメインの要素を定義することができます。

```
>> new( Z7, 2 );
```

```
                                new(Z7, 2)
```

これではあまりユーザに親切とは言えません。というのは，要素を生成するより前にnew関数に与えられた引数をチェックできないからです (ここでは，引数が本当に整数であるかどうかをチェックしたいところです)。では他にも要素を生成することのできる方法はないのでしょうか？　行列を考えて下さい。これはリストのリストとして与えることができますから，行列の要素を計算する関数を用意しても良いし，配列を用意しても良いわけです。
このような問題を解決するには，Z7ドメインにnewメソッド (ドメインの関数を「メソッド」と呼びます) を作ります。このメソッドは，Z7ドメインが関数のように呼ばれた時，即ち，Z7(2)のように呼び出される時に使われます。ここでnewメソッドは2を引数として呼び出されています (このメソッドが存在しないときには，エラーメッセージが出力されます)。

```
>> Z7::new := proc( n )
   begin
        if args(0) <> 1 then
            error("expecting one argument")
        elif domtype(n) <> DOM_INT then
            error("expecting an integer as argument")
        else
```

```
            new( Z7, n mod 7 )
        end_if
   end_proc:

>> a := Z7( 100 ); Z7( x );

                                    new(Z7, 2)
Error: expecting an integer as argument [Z7::new]
```

new 関数はオーバーロードできないことを覚えておいて下さい。

次は，Z7 の出力形式を改良します。そのために，Z7 用の print 関数を作り，オーバーロードしなければなりません。

目的のドメインのメソッド f を作ることで，関数 f をオーバーロードするのが普通のやり方です。当然，関数 f がオーバーロード可能であるときに限られます。関数がオーバーロード可能であるかどうか，その引数のうちどれがオーバーロード可能であるのかは，その関数のヘルプの項に書いてあります。

```
>> Z7::print := proc(x) begin expr2text(extop(x, 1))."(7)" end_proc:
a;

                                    2(7)
```

以下のメソッドは，要素の加算と乗算を実行するものです。

```
>> Z7::_plus := proc()
        local s, e, l;
   begin
        l := split( [args()], fun(domtype(args(1)) = Z7) );
        if nops(l[1]) = 0 then
            return( _plus( op(l[2]), op(l[3]) ) )
        end_if;

        s := 0;
        for e in l[1] do
            s := (s + extop(e, 1)) mod 7;
        end_for;

        e := _plus(op(l[2]), op(l[3]));
        if e = 0 then new( Z7, s )
        else hold(_plus) ( new( Z7, s ), e )
        end_if
   end_proc:

>> Z7::intmult := proc(x, n)
   begin
```

```
        if n < 0 then Z7::negate( _plus(x $ -n) )
        else _plus( x $ n )
        end_if
    end_proc:

>> Z7::_mult := proc()
         local m, e, l;
     begin
         l := split( [args()], fun(domtype(args(1)) = Z7) );
         m := 1;
         for e in l[1] do
             m := (m * extop(e, 1)) mod 7;
         end_for;

         m := new( Z7, m );
         e := _mult(op(l[2]), op(l[3]));
         if domtype(e) = DOM_INT then Z7::intmult(m, e)
         else hold(_mult) ( m, e )
         end_if
    end_proc:
```

幾つか計算させてみて，正しく動作するかどうか調べてみましょう。

```
>> b := Z7(1); c := Z7(11);

                                    1(7)

                                    4(7)

>> a + a + Z7(3) * b, a * b * b;

                                 0(7), 2(7)
```

次のメソッドは，テストと変換に必要になるものです。

```
>> Z7::testtype := proc(x, y) begin bool(extop(x, 0) = y); end_proc:
    Z7::domtype := proc(x) begin Z7 end_proc:
    Z7::convert := proc(x)
    begin
         case domtype(x)
         of Z7 do
             return( x )
         of DOM_INT do
             return( Z7( x ) )
         otherwise
```

```
            return( FAIL )
        end_case
   end_proc:
```

以上のように定義すると，以下のような結果を得ます。

```
>> domtype(a), testtype(a, DOM_STRING), Z7::convert( 100 );

                                Z7, FALSE, 2(7)
```

Z_7 は体ですから，零元と単位元を持ちます。

```
>> Z7::zero := Z7::new(0):
   Z7::one := Z7::new(1):
```

体に必要な演算はまだあります。

```
>> Z7::_divide := proc(x, y) begin new(Z7, (extop(x, 1) / extop(y, 1)) mod 7); end_proc:
   Z7::_negate := proc(x) begin new(Z7, -extop(x, 1) mod 7) end_proc:
   Z7::_subtract := proc(x, y) begin Z7::_plus(x, Z7::_negate(y)) end_proc:
   Z7::_invert := proc(x) begin new(Z7, (1/extop(x, 1)) mod 7) end_proc:
   Z7::_power := proc(x, n)
        local d, i;
   begin
        if n = 0 then return (Z7::one) end_if;
        if n > 0 then
            d := x;
        else
            d := Z7::invert(x); x := d;
        end_if;
        for i from 2 to abs(n) do
            d := Z7::_mult(d, x);
        end_for;

        d
    end_proc:
```

期待通り，新しいオブジェクト上でより沢山の演算ができるようになっています。

```
>> a^5 - b / c; 1 / Z7(100);

                                   2(7)

                                   4(7)
```

しかし，重要なのは，新たな多項式の係数ドメインとしてこのドメインが使えることなのです。
例えば次のようなことが可能です。

```
>> p := poly( 10 * x + 7 * x^2 - 1, [x], Z7 );
```

```
poly(3(7) x + 6(7), [x], Z7)
```

従って，新たなドメインによって，体上の多項式を扱うアルゴリズムを拡張したことになるのです (本当は嘘です。MuPAD は既に体 Z_7 を知っているからです)。

しかしながら，Z7 が体であること我々は知っていても，MuPAD はそれを知りません。アルゴリズムに有用なその情報をどうやってシステムに渡すことができるのでしょうか？

次の節でこの問題の解決策を見ていくことにします。

7.2 domains ライブラリ

セッションを初期化して下さい。

```
>> reset():
```

Dom パッケージは，整数環や有理数体といったデータ型を予め定義してます。

更に，データ型を生成する関数 (所謂「ドメインコンストラクタ」) も発見するでしょう。例えば，Z_n 環は一つの引数 n に依存し，この正の整数は剰余クラスを表わしています。

Z_{12} を作るのは Z_7 を作るのと酷似しており，全ての 7 のモジュロ演算を 12 のモジュロ演算に置き換えるだけでいいのです。

というわけで，MuPAD に Z_7 を生成するために，ドメインコンストラクタ Dom::INtegerMod を使用します。

```
>> Z7 := Dom::IntegerMod(7);
```

```
Dom::IntegerMod(7)
```

同様にして，7 のモジュロ演算を実行できます。

```
>> a := Z7(-100); b := Z7(57); c := Z7(2);
```

```
5 mod 7
```

```
1 mod 7
```

```
2 mod 7
```

```
>> a - b^2; a * b / c + a^3;
```

```
4 mod 7
```

```
5 mod 7
```

しかしながら，ドメインを使う際には別の重要な相違点が存在します。MuPAD には mod 関数を使った基本演算がないのです。

Z7 ドメインがその要素，+演算子，*演算子とで体を形成することは「知って」います。

```
>> Z7::hasProp( Cat::Field );
```

```
TRUE
```

ところが，Z_{12} は体ではありません (環ではあります)。

```
>> Z12 := Dom::IntegerMod(12): Z12::hasProp( Cat::Field ), Z12::hasProp( Cat::Ring );
```

```
FALSE, TRUE
```

MuPAD において，環や体といった代数構造は「カテゴリ」と呼ばれており，Cat ライブラリには沢山の予め定義されたカテゴリが見つけられます[1]

```
>> info( Dom );
Library 'Dom': Basic domain constructors
Interface:
Dom::AlgebraicExtension,
Dom::ArithmeticalExpression,
Dom::BaseDomain,
Dom::Complex,
Dom::DerivativeProduct,
Dom::DifferentialExpression,
Dom::DifferentialPolynomial,
Dom::DifferentialProduct,
Dom::DifferentialVariable,
Dom::DihedralGroup,
Dom::DiscreteSet,
Dom::DistributedPolynomial,
Dom::Expression,
Dom::ExpressionField,
Dom::ExteriorAlgebra,
Dom::ExteriorProduct,
Dom::Float,
Dom::Fraction,
Dom::FreeModule,
Dom::GaloisField,
Dom::Ideal,
Dom::IndependentDifferentialExpression,
Dom::Integer,
Dom::IntegerMod,
```

[1] (訳注) 原文では途中出力をカットしているが，ここでは全部載せてある (Ver. 1.4.1)。

```
Dom::Interval,
Dom::IntervalSet,
Dom::LinearDifferentialFunction,
Dom::LinearDifferentialOperator,
Dom::LinearFunction,
Dom::LinearOrdinaryDifferentialOperator,
Dom::Matrix,
Dom::MatrixGroup,
Dom::MonoidAlgebra,
Dom::MonoidOperatorAlgebra,
Dom::Multiset,
Dom::MultivariatePolynomial,
Dom::Numerical,
Dom::Pade,
Dom::PermutationGroup,
Dom::Polynomial,
Dom::PolynomialExplicit,
Dom::PowerProduct,
Dom::Product,
Dom::Quaternion,
Dom::Rational,
Dom::Real,
Dom::SparseMatrix,
Dom::SquareMatrix,
Dom::TruncatedPowerSeries,
Dom::UnivariatePolynomial,
Dom::UnivariateRationalFunctions,
Dom::UnivariateSkewPolynomial


>> info( Cat );
Library 'Cat': Basic category constructors
Interface:
Cat::AbelianGroup,
Cat::AbelianMonoid,
Cat::AbelianSemiGroup,
Cat::Algebra,
Cat::CancellationAbelianMonoid,
Cat::CommutativeRing,
Cat::DifferentialFunctionCat,
Cat::DifferentialRing,
Cat::DifferentialVariableCat,
Cat::EntireRing,
```

```
Cat::EuclideanDomain,
Cat::FactorialDomain,
Cat::Field,
Cat::FiniteCollectionCat,
Cat::FiniteMonoidRing,
Cat::GcdDomain,
Cat::Group,
Cat::HomogeneousFiniteCollectionCat,
Cat::HomogeneousFiniteProductCat,
Cat::IntegralDomain,
Cat::LeftModule,
Cat::MatrixCat,
Cat::Module,
Cat::Monoid,
Cat::MonoidRing,
Cat::OrderedAbelianMonoid,
Cat::OrderedMonoid,
Cat::OrderedSet,
Cat::PartialDifferentialRing,
Cat::PolynomialCat,
Cat::PrincipalIdealDomain,
Cat::QuotientField,
Cat::RestrictedDifferentialFunctionCat,
Cat::RightModule,
Cat::Ring,
Cat::Rng,
Cat::SemiGroup,
Cat::SetCat,
Cat::SkewField,
Cat::SquareMatrixCat,
Cat::UnivariatePolynomialCat,
Cat::UnivariateSkewPolynomialCat,
Cat::VectorSpace
```

domains ライブラリを用いれば，MuPAD は，前の節で Z7 を作ったようにドメインを作るより簡便な方法が使えます。更にこのライブラリは，ユーザがドメインに対して独自の代数構造が使えるように (そして生成させるように) することができます。

が，それは別のデモンストレーションツアー[2]にて … 。

[2](訳注) 今のところ (2000 年 3 月上旬)，これは存在していません。

第8章　訳者より

本文書は”Advanced Demonstration with MuPAD 1.4”の翻訳です。1999 年 6 月から訳出を開始し，翌 2000 年 2 月 2 日に最初の公開バージョンを完成させました。長くかかったのは，途中で訳者がサボっていたからです。

何分，学生時代の不勉強がたたって，訳がチクハグになっている可能性大です。皆様のご指摘お待ちしております。

平成 12 年 3 月 9 日

幸谷智紀 <tkouya@na-net.ornl.gov>
角谷　悟 <ICE11104@chiba.ice.or.jp>