Mapleの基本操作と実践 第5版

西谷滋人
京都大学工学研究科材料工学科
(version 5 for Maple9)

Department of Informatics,
Kwansei Gakuin University, Sanda, Japan
e-mail : nishitani@ksc.kwansei.ac.jp

平成16年9月30日

本書は PowerBookG4 上で Maple 9 と L^AT_EX を使用して編集，整形を行ないました．
Maple と Maple 9 は Waterloo Maple Inc. の登録商標です．
Macintosh, Power Macintosh は Apple Computer, Inc. の登録商標です．
UNIX は AT&T ベル研究所の登録商標です．
X Window System はマサチュセッツ工科大学の登録商標です．
Microsoft Windoes は Microsoft Corporation の登録商標です．
その他，本書中の製品名・会社名は，一般にそれぞれ各社の商標・登録商標です．

まえがき

”Any sufficiently advanced technology is indistinguishable from magic.“

Arthure C. Clarke の言葉です．Mathematica という数式処理ソフトを初めて使ったときにもこれと同じ感覚を持ちました．数式処理には大型計算機センターの Reduce を使うしかなかった当時に，私の持っていた MacSE で複雑な数式を一瞬にして簡単化し，数式らしく表示してくれたのを目のあたりにしたとき，また一歩科学が魔法に近づいた気がしました．

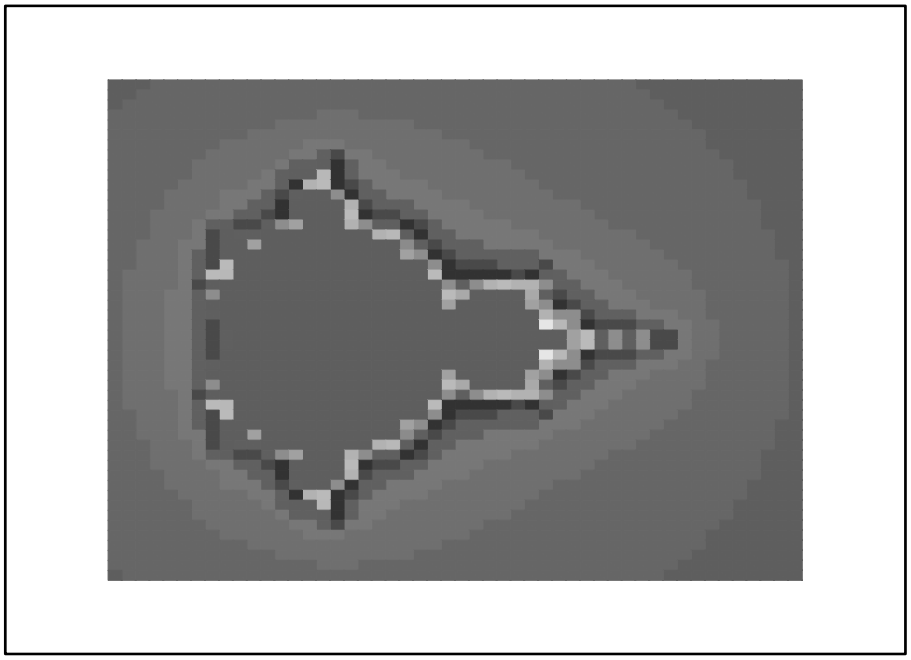
本書で使用法を解説する Maple は 1985 年に，Mathematica に先駆けて市場に現れました．カナダの Waterloo 大学と ETH Zürich での研究成果をもとにした計算ライブラリの内部処理は公開されており，その数式処理に対する信頼性の高さで定評があります．日常，私が Maple をどのように使っているかというと，紙と鉛筆，電卓，グラフソフト，programming 言語の代わりとして,（1）論文を読むときの数式の導出　（2）ちょっとしたルーチン計算,（3）結果のグラフ化，そして（4）プログラムを作るときのプロトタイプの作成，等です．

Mandelbrot 集合 を描くという実例を取り上げてみましょう．専門書をひもとくと (「C 言語によるアルゴリズム事典」，奥村晴彦著，技術評論社 1991）

> Mandelbrot(マンデルブロート）集合　Mandelbrot set
> (中略)
> 計算機では，ある領域の点 (x,y) について,
> z <-- x + i y; count <-- M;
> while (|z| <= 4) and (count>0) do begin
> z <-- z2 - (x + i Y); count <-- count -1
> end;
> 点 (x,y) に count で決まる色 (count=0 なら黒) を付ける;
>
> とする．黒い部分が Mandelbrot 集合で，それ以外の色は，その点と Mandelbrot 集合との ”近さ ” を表す光背である．

となっています．この後，プログラムコードが続き，どこかにある Graphics 表示用の謎のルーチンを呼ぶように指示してあります．これを Maple で実現しようと思うと，

```
> mandel:=proc(x,y)
local z0,z,count;
z0:=x+y*I;
z:=0;
count:=20;
while (evalf(abs(z))<4.0 and count>0) do
z:=z^2-z0;
count:=count-1;
od;
count;
end:
> with(plots):
densityplot(mandel,-1..2.5,-1.5..1.5,axes=none,colorstyle=HUE,
style=patchnogrid,grid=[50,50]);
```



となります．どうです?!　数学的な記述そのままで，しかもわずか数分で目的とするMandelbrot集合を実際に描くという作業が終了してしまいます．

数式処理や関数のグラフ化等を扱う優秀な市販ソフトウェアとしてはMathematicaとMapleが有名です．どちらかの使い方になじんでいればもう一方の，あるいは今後現れるであろうより優秀なソフトを修得するのにそれほど時間はいらないと思います．私自身もMathematicaからMapleへ乗り換えるのに1ヶ月程ですみました．本書があれば1週間で乗り換えられたと確信しています．

本書ではソフトの詳しい構造や数学の厳密な適用などには余り注意を払っていません．とにかく問題を理解し，解いていくために Maple をいかに使用するか，つまり道具として数学 (Maple) をいかに使うかに力点を置いて解説しています．ちゃんとした文法やコマンド引数などは help を参照し，試行錯誤をお願いします．

Maple で作った書類は L^AT_EX や HTML 等にも簡単に変換してくれます．本書は実際に L^AT_EX で出力したファイルから加工しています．また，本書の全てのスクリプトを HTML で

`http://www.mtl.kyoto-u.ac.jp/Maple/Maple.index`

から引けるようになっています．

このテキストは 1996 年以来 10 年近くの間改定を重ねてきました．最初は，京都大学材料工学科の演習用として，主に材料工学関連の題材を扱っていました．今回の改定では，関西学院大学理工学部情報科学科の 3 年生に対して，著者が開講しました数値計算，ならびに数値計算演習，および数式処理演習でつかった題材を用いています．

目 次

Maple を初めとする数式処理ソフトを習得するときに心がけるべき鉄則をまとめておきます．うまく出力されないときに試してください．

鉄則 0 restart をかける：続けて入力すると前の入力が生きています．違う問題へ移るときやもう一度入力をし直すときには restart を入力してください．

鉄則 1 出力してみる：紙面の関係で出力を抑止していますが，できるだけ多く出力するようにしてください．最後の：を；に変えるとか，printf を使ってください．

鉄則 2 関数に値を代入してみる：数値が返ってくるべき時に変数があればどこかで入力をミスっています．plot で Plotting error,empty plot が出た場合にチェック下さい．

鉄則 3 内側から順に入力する：長い入力は内側の関数から順に何をしているか確認しながら打ち込んで下さい．

第1章　基本操作

1.1　最初の一歩

ここでは Maple の基礎的な操作法と注意事項を述べます．

1.1.1　Maple の起動法

PC ではほかのソフトウェアと同様の立ち上げ方で起動します．unix では xmaple とすると GUI 版の maple が立ち上がります．maple だけですと普通の vt100 上での text 版が立ち上がります．これを unix の redirection 機能を使って filter として機能させることも可能です．起動できない場合は PATH と executability を確認して下さい．

1.1.2　簡単な演算

それでは簡単な計算を実行させてみましょう．

```
> 1+1;
```

$$2$$

enter と shift+enter は違った意味を持ちます．enter は入力，shift+enter は改行です．複数行にまたがる入力では改行には shift+enter を入力し，最後に enter をいれればその領域すべてを一度に入力したことになります．

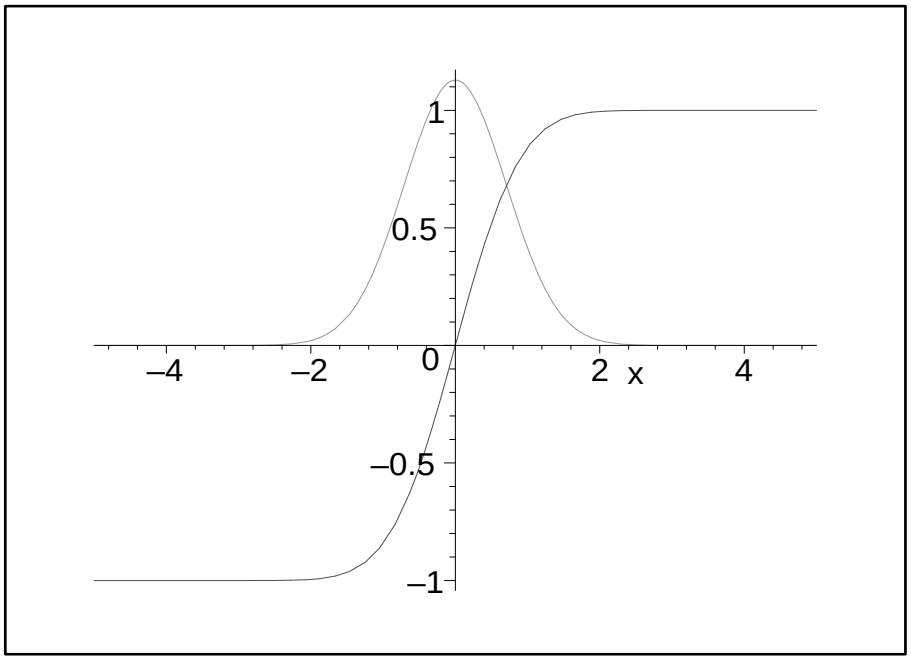
```
> set1:={1,2,3};(shift+enter)
> set2:={3,4};(enter)
```

$$set1 := \{1, 2, 3\}$$
$$set2 := \{4, 3\}$$

これ以降の記述では最後の (enter) や (shift+enter) を省きます．最後の；(セミコロン) を忘れがちです．出力させたくないときには最後の；を：(コロン) にすれば，なにも出力しません．ただし，内部での代入は実行されています．入力の順番はエンターをいれた順番であり，画面の上下とは関係ありません．

また入力領域のどの位置にカーソルがあってもエンターでその領域全部を入力したことになります．次に関数のプロットをしてみましょう．

```
> plot({erf(x),diff(erf(x),x)},x=-5..5);
```



となります．意味は直観的で，“erf とその微分 (diff) を-5から 5まで表示する” です．

1.1.3 間違い修正

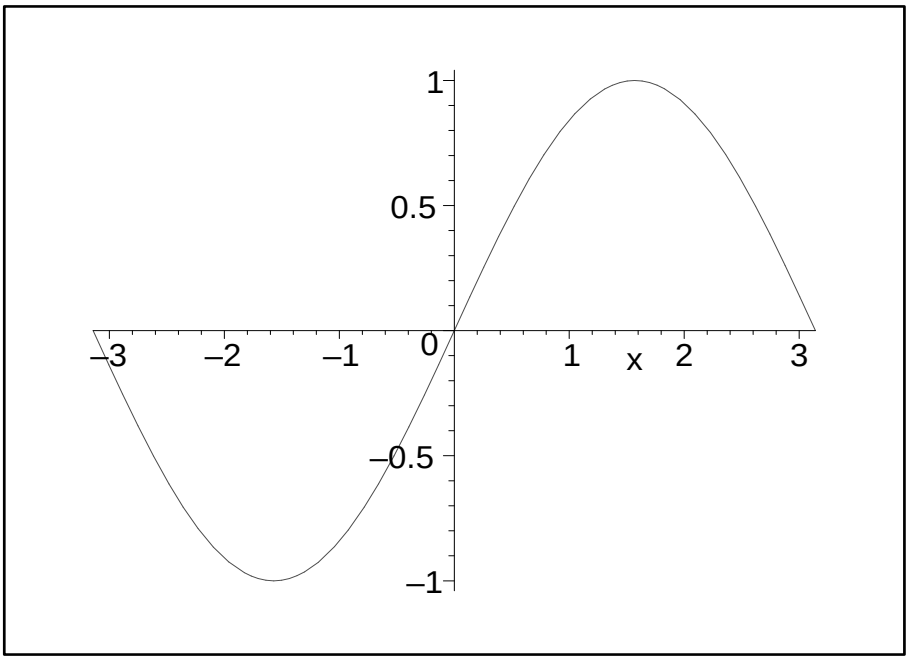
打ち間違いなどの訂正はアローキー，あるいはマウスのクリックによってできます．後は訂正して enter を入れれば入力されます．ある領域を選択して，削除・カットおよびペーストなどの作業もマウスを使ってできます．例えばサイン関数を－πから＋πまでプロットしようとした時

```
> plot({sin(x)},x=-pi..pi);
```

```
Error, (in plot/transform) cannot evaluate boolean: -pi < -4
```

という警告が出ました．これは Maple では大文字と小文字を区別しているためにおこったことです．そこで，pi→Pi と修正すると無事表示されます．

```
> plot({sin(x)},x=-Pi..Pi);
```



1.1.4 実行の中断

enter を押した後，入力ミスに気づいて計算をやめさせたいときには中断ができます．PC では tool bar の stop マークで，そして unix では interrupt button で中断します．

1.1.5 ヘルプファイル

```
> ?help;
```

でキーワードに関するヘルプが表示されます．その他，??や？index あるいは？（キーワードの最初の一部）を使って，類推によってキーワードの情報を取り出すことができます．その他の help に関する操作はメニューバーの ”Help ” にいくつか用意されています．以下は help の出力です．記述は Windows 版を除いて英語です．英語が分からなくても Examples を参考にすればだいたい予測できます．

```
> ?plot;
```

plot - *create a two-dimensional plot of functions* **（関数の**

説明)

Calling Sequence:（呼び出し）

plot(f, h, v)

plot(f, h, v,...)

Parameters:（引数の説明）

f - function(s) to be plotted

h - horizontal range

v - vertical range (optional)

Description:（詳しい解説）

- A typical call to the plot function is plot(f(x),x=a..b), where f is a real function in x and a..b specifies the horizontal real range on whi ch f is plotted.

⋮
中略
⋮

Examples:（使用例）

```
> plot(cos(x) + sin(x), x=0..Pi);
> plot(tan(x), x=-Pi..Pi);
> plot([sin(t), cos(t), t=-Pi..Pi]);
> plot(sin(t),t);
```

Since no domain is specified in the last example, we use t=-10..10.

⋮
中略
⋮

See Also: （関連する項目）

plot[spec] where spec is one of ***infinity***, ***polar***, ***parametric***, ***multiple***, ***ranges***, ***function***, ***options***, ***structure***, ***setup***, ***device***, ***replot***, ***style***, ***color***. Use ***plot3d*** for plotting surfaces. See ***discont*** and ***fdiscont*** regarding discontinuities.

1.2 簡単な数式処理とプロット

簡単な数値の代入と関数，関数の定義と式の変形，グラフのプロットを扱います．式の変形は難しく，なかなか思うように単純化してくれません．人間だとすぐに分かることも，Maple に教えてやらなければなりません．しかし，Maple はややこしい式の変形でも係数や次数を見落とすことはありません．

1.2.1 変数，式の定義とそのキャンセル

Maple の基本的な代入は

```
> mass:=10;
```

$$mass := 10$$

によって行われます．あらかじめ Maple で定義されていてよく使う定数は Pi, I, infinity などです．(?ininames 参照) 式の定義も同様に行えます．

```
> force:=-mass*accel;
```

$$force := -10\, accel$$

直前の結果を参照するには%を使います．

```
> exp1:=%;
```

$$\mathrm{exp1} := -10\, accel$$

これら一度数値を入れた変数を元へ戻すには

```
> restart;
```

によって行います．これで起動初期のなにも入力されていない状態に戻ります．ひとつの定義だけを初期状態に戻したいときには ’（シングルクォート）をもちいて

```
> mass:='mass';
```

$$mass := mass$$

によって行います．これによって，

```
> force:=-mass*accel;
```

$$force := -mass\, accel$$

となります．一時的な代入は subs で行います．

```
> subs(mass=10,accel=14,force);
```

$$-140$$

こうすると，

```
> force;
```

$$-mass\ accel$$

とそれぞれの変数が数値を取るのではなく，変数のままで扱われます．逆に一時的に値ではなく変数そのものを使用するにもシングルクォートを使います．

```
> x:=2;y:=3;
```

$$x := 2$$

$$y := 3$$

```
> f:='x+y'; g:=x+y;
```

$$f := x + y$$

$$g := 5$$

（注：続けて入力される方はここら辺で restart をかけてください．以下では数値を代入した変数を，free の変数として使っています．そのまま入力を続けると叱られます）

1.2.2 関数

よく使う関数はそのままの形で使えます．三角関数 (trigonometric functions) はラジアンで入れてください．log （もちろん ln も）は自然対数です．底をあらわにするときは

```
> log[2](5);
```

$$\frac{\ln(5)}{\ln(2)}$$

としてください．数値として取り出したいときには

```
> evalf(%);(evaluating float の略)
```

2.321928094

Maple が提供する膨大な数の関数から，目的とするものを探しだすには help を使って下さい．以下に関数等の index を表示する keywords をまとめておきます．

? inifcns - 起動時から認識されている関数

? index[package] - 関連する関数を集めたパッケージです．微分方程式 (DETools), 線形代数 (linalg), プロット関係の関数 (plots) 等があります．

? index[function] - Maple の標準関数．

これらの関数は起動時から読み込まれている関数と，ユーザーが呼び出さなければならない関数とがあります．呼び出しが必要な時には

```
> with( package);
```
でおこなってください.

単純なユーザー関数の定義は次の 2 種類を使います.

i) 矢印による定義.

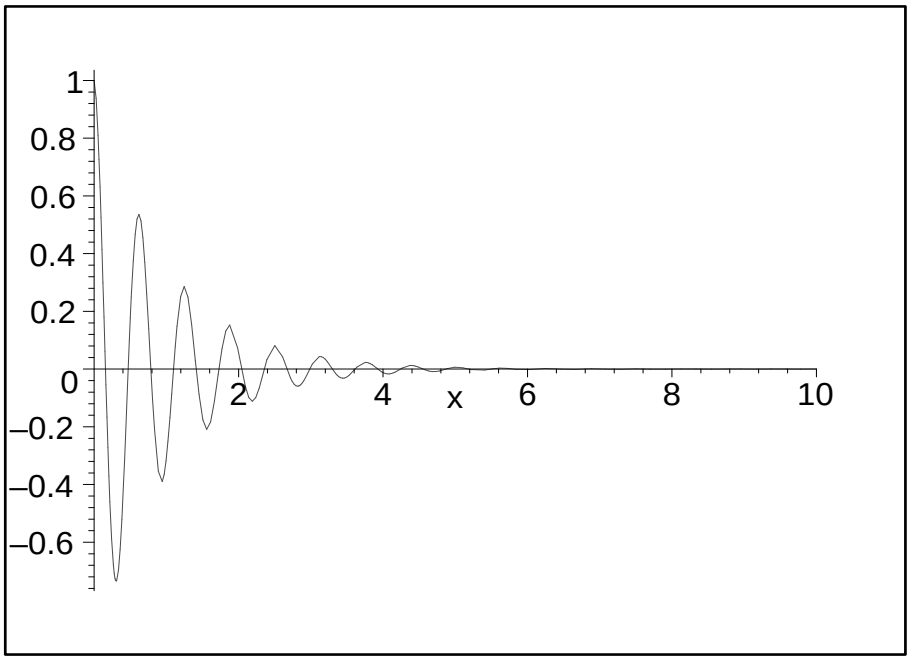
ii) unapply による定義.

矢印による定義は普通の入力のようにして,
```
> restart:
eq1:=x->exp(-x)*cos(10*x);
```
$$eq1 := x \to e^{(-x)} \cos(10\,x)$$
unapply は一度求めた式を関数として定義するときに使います. たとえば以下では eq1 という式に入っている定義にしたがって, x を変数とする f1 という関数と定義しています.
```
> restart:
eq1:=exp(-x)*cos(10*x);
f1:=unapply(eq1,x);
```
$$eq1 := e^{(-x)} \cos(10\,x)$$
$$f1 := x \to e^{(-x)} \cos(10\,x)$$
まちがって数式に対して矢印での定義をすると
```
> f1:=x->eq1;
```
$$f1 := x \to eq1$$
```
> f1(1);
```
$$e^{(-x)} \cos(10\,x)$$
となり変数を変数とみなしてくれません.

ややこしい操作が必要な関数を定義するには第 1.5 節で解説する proc を使います.

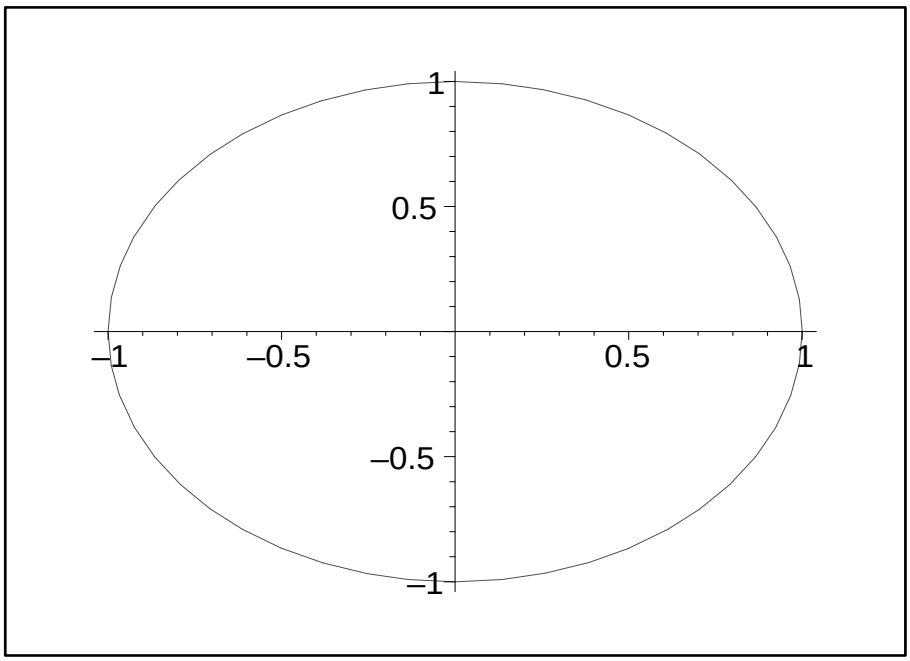
1.2.3 プロット

複雑な関数も Maple だとすぐに視覚化してくれます. 先程のユーザー定義関数を使って, 関数が実際にどのような形をしているか plot させてみましょう.
```
> plot(eq1(x),x=0..10);
```



となります．パラメーターによるプロットも可能です．例えば

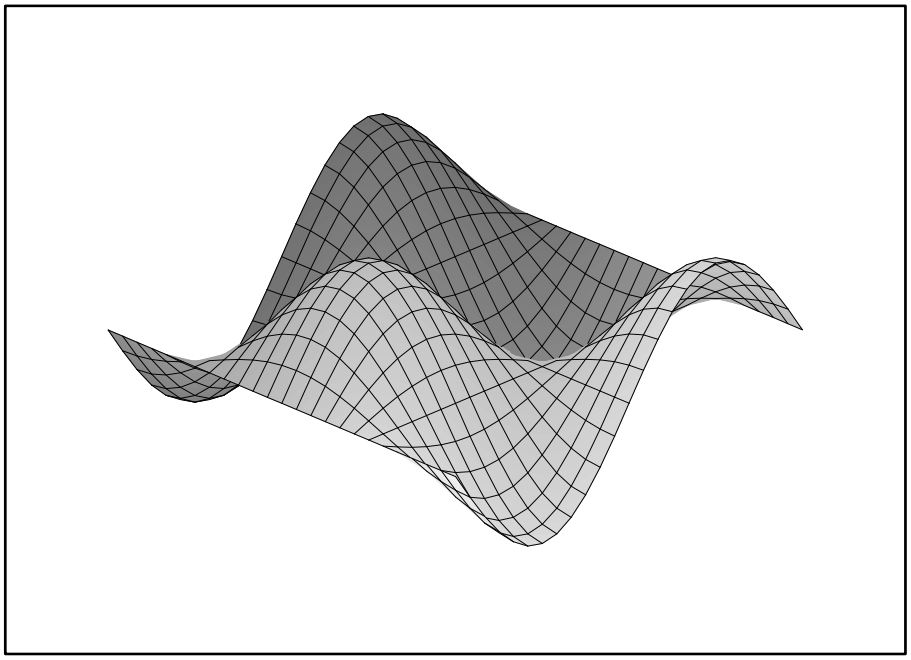
```
> plot([sin(t),cos(t),t=-Pi..Pi]);
```



によって円が描けます．真円にしたいときには出力図形をマウスで選んだ後，メニューバー下段の (1:1) ボタンを押してください．2 変数の場合には plot3d を使います．変数名が明らかな時には省略可能です．

```
> f1:=(x,y)->sin(x)*cos(y);
> plot3d(f1,-Pi..Pi,-Pi..Pi);
```

$$f1 := (x,\, y) \to \sin(x)\cos(y)$$



オプションや特殊な plotting 法があります．plot に対する一部の簡単な操作（視点の変更，表面の加工，軸の挿入) はメニューバーからできます．さらに

```
> with(plots):
```

で呼び出される plots package には多くの便利な表示関数が用意されています．その一部の機能については後ほど実例をお見せします．詳しい内容はそれぞれの help を参照ください．

1.2.4 方程式の解

solve で方程式の解が求まります．

```
> eqset:={x+y=1,y=1+x^2};
```

$$eqset := \{x + y = 1,\, y = 1 + x^2\}$$

```
> solve(eqset,{x,y});
```

$$\{x = 0, y = 1\}, \{x = -1, y = 2\}$$

これだけでは

```
> x;y;
```

$$x$$

$$y$$

のように変数 x,y へ代入されていません．これを代入するには

```
> solset:=solve(eqset,{x,y});
```

$$solset := \{x = 0, y = 1\}, \{x = -1, y = 2\}$$

```
> solset[1];
```

$$\{x = 0, y = 1\}$$

```
> assign(solset[1]);
> x;y;
```

$$0$$

$$1$$

のように assign を使います． 解を整数の範囲で求める isolve というのもあります．また，代数的に解けない非線形の方程式などに対しても数値的に解く関数 (fsolve) が用意されています．

1.2.5 式の変形

式の展開や簡単化のために使われるコマンドを以下にまとめて記します．詳しい意味や使用例はヘルプを参照してください．”MapleV ラーニングガイド”の 2.6 数式の変形 に分かりやすい実例が載っています．

- expand - Expand an Expression （展開）
- simplify - Apply Simplification Rules to an Expression （簡単化，これですべてうまく行くといいのですが...）
- collect - Collect Coefficients of Like Powers （項の次数で式をまとめる）
- combine - combine terms into a single term （規則にしたがって式をまとめる）
- coeff - extract a coefficient of a polynomial （多項式の係数の取り出し）
- sort - sort a list of values or a polynomial （式や値のソート）

- factor - Factor a Multivariate Polynomial （一つ以上の変数を持つ多項式の因数分解）
- normal - normalize a rational expression （約分，通分）
- convert - convert an expression to a different form （関数を違う関数へ変換）

- gcd - greatest common divisor of polynomials （多項式の最大公約数）
- lcm - least common multiple of polynomials （多項式の最小公倍数）
- numer - numerator of an expression （分母）
- denom - denominator of an expression （分子）
- lhs, rhs - left hand side or right hand side of expression （式の左辺あるいは右辺）
- radsimp - simplification of an expression containing radicals （分母の有理化）

- assume - The Assume Facility （変数の範囲や関係を規定）

1.3 微積分とその応用

前章の簡単な数式処理の続きで微分・積分とその周辺の題材です．級数は物理や化学の論文でよく使われています．基本的にはうまく解析的に解が出てこないところで，近似として使われます．したがって，どの程度の近似かを常に意識する必要があります．

1.3.1 総和の計算

総和は

```
> sum(i,i=1..10);
```

$$55$$

のようにして求まります．以下は高校でお目にかかった公式です．

```
> sum(i^2,i=1..n);
```

$$\frac{1}{3}(n+1)^3-\frac{1}{2}(n+1)^2+\frac{1}{6}n+\frac{1}{6}$$

```
> sum(a^i,i=1..infinity);
```

$$-\frac{a}{a-1}$$

よく似た関数に，product というのがあります．これは数列の和ではなく積を求めてくれます．また，add，mul というのがありますが，こちらは数値の列の和や積を高速に求めてくれます．

1.3.2 極限

極限の値は limit によってもとまります．

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

$$1$$

```
> limit(1/x,x=0,complex);
```

$$\infty-\infty I$$

高校で習った知識がそのまま使えます．

```
> sum(1^i/i!,i=0..infinity);
```

$$e$$

例えば，以下のような式に単純な代入を行うと

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

$$r := \frac{x^2-1}{(x+1)\,(x^2+2)}$$

```
> subs(x=-1,r);
```

```
Error, division by zero
```

とエラーを返してきます．これの極限をとると

```
> limit(r,x=-1);
```

$$\frac{-2}{3}$$

つぎに右側極限と左側極限とが違う値に収束している場合は

```
> limit(tan(x),x=Pi/2);
```

$$undefined$$

です．しかし，右と左を指定すると

```
> limit(tan(x),x=Pi/2,right);
> limit(tan(x),x=Pi/2,left);
```

$$-\infty$$

$$\infty$$

とちゃんと求めてくれます．

1.3.3 微分

微分は diff によって行います．

```
> diff(x^2,x);
```

$$2\,x$$

```
> diff(y^2*x^2,x,x);
```

$$2\,y^2$$

関数の一般形のままでも形式的な微分を表示してくれます．

```
> c:=(x,t)->X(x)*T(t);
```

$$c := (x,\,t) \to \mathrm{X}(x)\,\mathrm{T}(t)$$

```
> diff(c(x,t),t);
> diff(c(x,t),x,x);
```

$$\mathrm{X}(x)\,(\tfrac{\partial}{\partial t}\,\mathrm{T}(t))$$

$$(\tfrac{\partial^2}{\partial x^2}\,\mathrm{X}(x))\,\mathrm{T}(t)$$

1.3.4 微分方程式

微分方程式は dsolve によって解きます．例えば，

```
> eq:=diff(y(x),x)+y(x)=0;
```

$$eq := (\frac{\partial}{\partial x}\,\mathrm{y}(x)) + \mathrm{y}(x) = 0$$

```
> dsolve(eq,y(x));
```

$$\mathrm{y}(x) = \_C1\, e^{(-x)}$$

と求められます．さらに初期条件を付け加えるときには

```
> dsolve({eq,y(0)=a},y(x));
```

$$\mathrm{y}(x) = a\, e^{(-x)}$$

として代入します．この他に連立微分方程式，べき級数解 (option=series)，数値解 (option=numeric) 等も求めることができます．詳しい解説，例が ”MapleV による数式処理入門 ”阿部寛著（講談社，1997) にあります．

1.3.5 積分

不定積分，定積分はそれぞれ

```
> int(ln(x),x);
```

$$x\ln(x) - x$$

```
> int(sin(x),x=-Pi..0);
```

$$-2$$

などで求めます．int を integrate としても同じ結果を得ます．積分公式がないと求められないような関数も

```
> eq:=x^2/sqrt(1-x^2);
> int(eq,x);
```

$$eq := \frac{x^2}{\sqrt{1-x^2}}$$

$$-\frac{1}{2}\,x\sqrt{1-x^2} + \frac{1}{2}\arcsin(x)$$

```
> eq2:=exp(-x^2);
> int(eq2,x=-z..z);
```

$$eq2 := e^{(-x^2)}$$

$$\sqrt{\pi}\,\mathrm{erf}(z)$$

という具合です．積分を実行するのではなく，積分記号だけを表示させたいときには Int とします．部分積分を施すには，with(students) に intparts という関数が

あります.

1.3.6 級数

Maple で式の Tayler 級数展開を見てみましょう．位数は最後につけます．例えばある関数を原点の周りで 4 次まで展開してみましょう.

```
> series(f(x),x=0,4);
```

$$\mathrm{f}(0) + \mathrm{D}(f)(0)\,x + \frac{1}{2}\,(\mathrm{D}^{(2)})(f)(0)\,x^2 + \frac{1}{6}\,(\mathrm{D}^{(3)})(f)(0)\,x^3 + \mathrm{O}(x^4)$$

この関数を取り出すためには convert を使います.

```
> convert(%,polynom);
```

$$\mathrm{f}(0) + \mathrm{D}(f)(0)\,x + \frac{1}{2}\,(\mathrm{D}^{(2)})(f)(0)\,x^2 + \frac{1}{6}\,(\mathrm{D}^{(3)})(f)(0)\,x^3$$

これと unapply を組み合わせれば，多様な関数の定義ができます.

1.3.7 複素関数

多くの関数は複素数を引数としてとることができます.

```
> series(exp(x),x=0.5*I,3);
```

$$.8775825619 + .4794255386\,I + (.8775825619 + .4794255386\,I)\,(x - .5\,I) + (.4387912810 + .2397127693\,I)\,(x - .5\,I)^2 + \mathrm{O}((x - .5\,I)^3)$$

```
> int(exp(x/2),x=1+1*I..0+1*I);
> evalc(%);
```

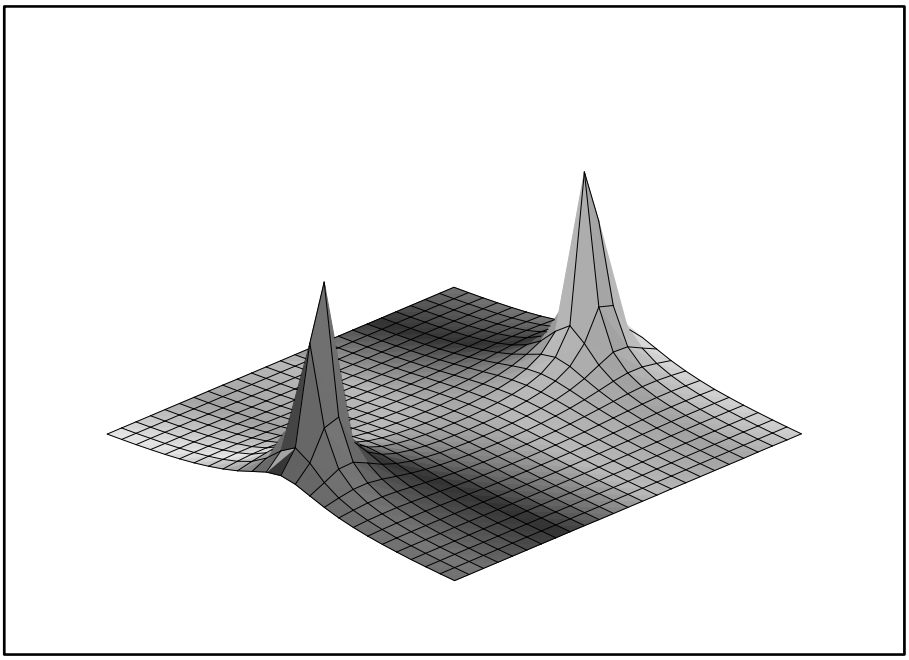
$$-2\,e^{(1/2+1/2\,I)} + 2\,e^{(1/2\,I)}$$

$$-2\,e^{(1/2)}\cos(\frac{1}{2}) + 2\cos(\frac{1}{2}) + I\,(-2\,e^{(1/2)}\sin(\frac{1}{2}) + 2\sin(\frac{1}{2}))$$

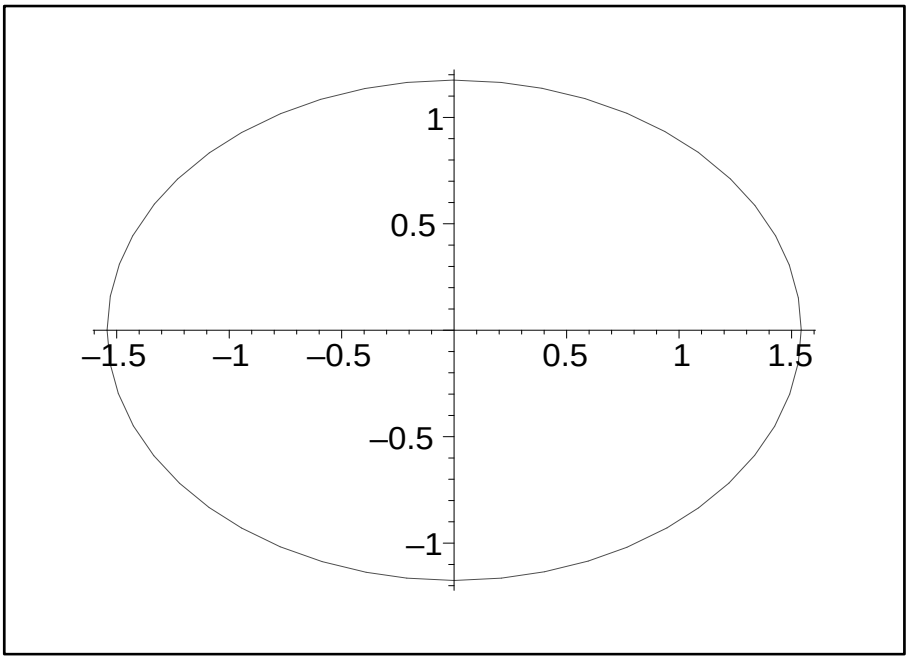
evalc は複素数の実数部と虚数部をあらわな形に分けて表示してくれます．実数部だけを取りだすときには Re()，虚数部だけは Im()，複素共役を求めるのには conjugate() です．さらに複素平面での関数のプロットもお任せでやってくれます.

```
> with(plots):
> complexplot3d( sec(z) , z = -2 - 2*I ..  2 + 2*I );
```

```
Warning, the name changecoords has been redefined
```



```
> complexplot(sin(x+I),x=-Pi..Pi);
```



1.4 データ構造と線形代数

Maple は種々のデータ構造が使えます．これらデータ構造の中で混乱しやすい集合，リスト，表，配列についてまとめて説明します．さらに行列とベクトルを扱う線形代数のパッケージについて述べます．

1.4.1 集合，リスト，表，配列

集合 set {a,b,c}

集合は普通に使われる集合と同じ意味を持ちます．集合に対する組み込みの関数には以下のものがあります．

- FUNCTION:union - set union operator （和）
- FUNCTION:intersect - set intersection operator （積）
- FUNCTION:minus - set difference operator （差）

```
> set1:={1,2,3};
set2:={3,4};
```

$$set1 := \{1, 2, 3\}$$

$$set2 := \{3, 4\}$$

```
> set3:=set1 union set2;
```

$$set3 := \{1, 2, 3, 4\}$$

```
> set4:=set1 intersect set2;
set5:=set1 minus set2;
```

$$set4 := \{3\}$$

$$set5 := \{1, 2\}$$

集合では値が重複する時には一つだけを残して他は削除されます．値には基本的に順番がありません．

リスト list [a,b,c]

リストには順番があります．また，重複する場合にもそのまま残されます．C 言語や FORTRAN で用意されている配列に対応します．ただし，C 言語の配列とちがい，一番最初の要素をさす添え字は 1 です．これは FORTRAN と同じです．

```
> list1:=[1,2,3];
> list2:=[3,4];
```

$$list1 := [1, 2, 3]$$
$$list2 := [3, 4]$$

この結合をつくるときには op(list) を使います．この関数は list の要素からなる部分列を生成します．これを使ってリストの内容を取りだし，さらに結合を作ることができます．

```
> list3:=[op(list1),op(list2)];
```

$$list3 := [1, 2, 3, 3, 4]$$

集合と違って，要素は全て保存されます．リストに要素を追加する (append, prepend) ときにも同様の手を使います．リストに含まれている要素の個数を知りたいときには nops を使います．

```
> n:=nops(list3);
```

$$n := 5$$

要素の内容は list[index] で取り出せます．

```
> list3[5];
```

$$4$$

index は 1 から始まります．全ての要素を表示したいときには print(list) を使います．データの入れ換えは単純に，

```
> list3[2]:=x;
```

$$list3_2 := x$$

です．全部を表示してみると

```
> list3;
```

$$[1, x, 3, 3, 4]$$

表 table

Mapleでは表形式のデータ構造を取ることができます．その場合index(subscript)はアルファベットの名前なども使えます．表を作るときには table 関数を用いて以下のようにします．

```
> atomicweight:=table([Fe=56,C=12]);
```

$$atomicweight := \mathrm{table}([C = 12, Fe = 56])$$

```
> atomicweight[Fe];
```

$$56$$

表の変更は

```
> atomicweight[Fe]:=55.847;
```

$$atomicweight_{Fe} := 55.847$$

で行います．また追加は

```
> atomicweight[Al]:=27;
```

$$atomicweight_{Al} := 27$$

です．内容の表示は print で行います．

```
> print(atomicweight);
```

$$\text{table}([Al = 27,\ C = 12,\ Fe = 55.847])$$

削除は

```
> atomicweight[C]:='atomicweight[C]';
```

$$atomicweight_{C} := atomicweight_{C}$$

```
> print(atomicweight);
```

$$\text{table}([Al = 27,\ Fe = 55.847])$$

です．(”ちょん”はここでは普通のシングルクォート”’”を使います．バッククォート”‘”ではありません)

配列 array

配列は表の subscript （添字) を整数に限定したものです．これは行列やベクトルとして使えます．表の場合の table と同じように配列では array を用います．ただし，subscript の範囲を明示する必要があります．ベクトルは array(1..n) で配列は array(1..m,1..n) として指定します．特別の関係を持った行列は以下のようにして定義することができます．

```
> array(identity,1..3,1..3);
> array(sparse,1..3,1..3);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> array(1..2,1..2,[[1,2],[3,4]]);
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

```
> A:=array(antisymmetric,1..3,1..3);
> A[1,2]:=1;A[1,3]:=2;A[2,3]:=3;
> print(A);
```

$$A := \mathrm{array}(antisymmetric,\ 1..3,\ 1..3,\ [])$$

$$A_{1,2} := 1$$

$$A_{1,3} := 2$$

$$A_{2,3} := 3$$

$$\begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 3 \\ -2 & -3 & 0 \end{bmatrix}$$

構造の確認，構造変換と関数の適用

type 前に作ったデータがどういう構造を持っているのか分からなくなってしまうときがあります．例えば，リストなのか，配列なのかという場合です．このようなときには

```
> type(A,array);
```

$$true$$

などで確認することができます．返答は true か false です．

convert 集合，リスト，表，配列の相互のデータ構造間の構造変換も行ってくれます．以下では配列をリストのリスト (listlist) に変換しています．

```
> convert(A,listlist);
```

$$[[0,\ 1,\ 2],\ [-1,\ 0,\ 3],\ [-2,\ -3,\ 0]]$$

map 全ての要素に関数を適用したい場合には

```
> map(sin,A);
```

$$\begin{bmatrix} 0 & \sin(1) & \sin(2) \\ -\sin(1) & 0 & \sin(3) \\ -\sin(2) & -\sin(3) & 0 \end{bmatrix}$$

のように map によっておこないます．さらに map によってデータの任意の列から成るあらたな listlist をつくる事も可能です．

```
> map(u->[u[1],u[3]],convert(A,listlist));
```

$$[[0,\ 2],\ [-1,\ 3],\ [-2,\ 0]]$$

1.4.2 線形代数

Maple の標準関数を使って線形代数 (linear algebra) を適用することは可能ですが面倒です．従来は Maple ライブラリーに用意されている linalg package を使っていました．バージョン 6 で新たに LinearAlgebra(LA) が追加され，その速度と操作性の良さから，今後これが標準となるようです．複雑な線形代数の代数的操作が必要な時には linalg をお使いください．

Matrix や Vector の生成には以下のようにたくさんの方法があります．

```
> with(LinearAlgebra):
> ma0:=Matrix(A);
> ma1:=<<1,2,3>|<4,5,6>|<7,8,9>>;
```

$$ma0 := \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 3 \\ -2 & -3 & 0 \end{bmatrix}$$

$$ma1 := \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

```
> vec1:=Vector([x,y,z]);
> vec2:=<1,2,3>;
> vec3:=<1|2|3>;
```

$$vec1 := \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$vec2 := \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

$$vec3 := [1,\ 2,\ 3]$$

等です．ここで Vector は縦（列）ベクトル (column vector) を生成することに注意ください．行列の転置を行う演算 Transpose(vec) を行うと横（行）ベクトル (row vector) となります．(英語で座席は row で，新聞の囲み記事は column です)．

多くの高機能な演算が LinearAlgebra package に用意されています．使い方は以下の通りです．matrix 計算の時には次元が整合していなければなりません．

```
> ma0.ma0;
> MatrixAdd(ma0,ma1,X,Y);
```

$$\begin{bmatrix} -5 & -6 & 3 \\ -6 & -10 & -2 \\ 3 & -2 & -13 \end{bmatrix}$$

$$\begin{bmatrix} Y & X+4\,Y & 2\,X+7\,Y \\ -X+2\,Y & 5\,Y & 3\,X+8\,Y \\ -2\,X+3\,Y & -3\,X+6\,Y & 9\,Y \end{bmatrix}$$

```
> ma0.vec1;
```

$$\begin{bmatrix} y+2\,z \\ -x+3\,z \\ -2\,x-3\,y \end{bmatrix}$$

```
> Transpose(vec1).vec1;
> Multiply(vec1,Transpose(vec1));
```

$$x^2+y^2+z^2$$

$$\begin{bmatrix} x^2 & x\,y & x\,z \\ x\,y & y^2 & y\,z \\ x\,z & y\,z & z^2 \end{bmatrix}$$

全ての演算が with(LinearAlgebra) を実行したときに表示されますので，関連している項目のヘルプを参照してください．以下に主に使う演算を示します．

- MatrixAdd - matrix or vector addition（和）(MatrixAdd(A,B,c1,c2):c1*A+c2*B)
- DotProduct - vector dot (scalar)product （内積）
- CrossProduct - compute the cross product of two Vectors （外積）
- MatrixInverse - compute the inverse of a matrix （逆行列）
- Determinant - determinant of a matrix （行列式）
- Trace - the trace of a matrix （対角和）
- Adjoint - compute the adjoint of a matrix （随伴）
- Transpose - compute the transpose of a matrix （転置行列）
- Eigenvalues - compute the eigenvalues of a matrix （固有値）
- Eigenvectors - find the eigenvectors of a matrix （固有ベクトル）

- VectorAngle - compute the angle between vectors （角度）
- Dimension - determine the dimension of a Matrix or a Vector （要素数）

linalg にもこれらに対応する関数が用意されています．関数名は微妙に違います．linalg にはこの他にも curl,diverge,grad,jacobian などの微分演算子や，行列やベクトルの行や列を操作するオペレーションが用意されています．

1.5 MapleV でのプログラミング

1.5.1 プログラミングの流れ

ある決まった操作をするときや関数が複雑になってきた場合にプログラミングが必要になってきます．Maple では FORTRAN や C などの一般的なプログラム言語より圧倒的に楽にプログラミングができます．このプログラミングの便利さは，ちょっとしたルーチン計算のために使えるだけでなく，本格的なプログラムを書く前にそのひな型を検討する際にも多いに役立ちます．感じとしては UNIX の shell でちょこちょこと定型処理を造るようです．

では，Maple ではプログラムはどのようにして書くのでしょうか？その前にプログラミングの一般的な注意を述べておきます．

1. 本当にプログラミングが必要か？
   いくらプログラミングが楽だといってもやはりプログラミングには時間がかかります．従って，先ず本当にその処理や関数をわざわざ書く必要があるのかを考えてください．『あるものは使え！』が原則です．Maple で提供されている多くの関数の中に使える関数があるかを先ず探してみてください．

2. プログラムの作成
   プログラムが必要であるという結論に達したときにはそれを実際に作っていくしかありません．この作業の手順は普通のプログラムと全く同じです．本格的に問題を解くときにはアルゴリズムやデータ構造を考える必要がありますが，我々が解く必要がある問題はほとんどの場合単純です．従って，素直に手でやる計算を自動化するだけで出来上がります．下手にテクニックを考えるよりも何をしているのかが見るだけで理解できるプログラムを書くように心掛けてください．

3. 汎用性？
   さらに汎用性や，多くの人が何度も使う可能性がある場合には ERROR,type check, ヘルプファイル，ユーザライブラリの作成のような作業を行う必要があります．

1.5.2 Maple の制御構造

一般的なプログラム言語が提供する次の 3 種の制御構造が Maple でも使えます．

if 場合分け

for 決まった分だけぐるぐる

while あるところまでぐるぐる

この構文を以下に示します.

if *condition1* **then**
statement sequence 1
elif *condition2* **then**
statement sequence 2
else
default statement sequence
end if;

for *i* **from** *start* **by** *change* **to** *finish*
do
statement sequence
end do;

while *condition*
do
statement sequence
end do;

next,break

for や while- loop の途中で，処理を省きたいときや loop から脱出したいときには next,break をそれぞれ使います．構文は

- **if** *condition* **next**; od まで跳んで次の loop へ

```
> for i from 1 to 10 do
> if (modp(i,2)=0) then next fi;
> printf("%d, ",i);
> end do;
1, 3, 5, 7, 9,
```

- **if** *condition* **break**; 一番近い od の外へ出て loop を脱出

```
> for i from 1 to 10 do
> if (i>5) then break fi;
> printf("%d, ",i);
> end do;
1, 2, 3, 4, 5,
```

ひな型

procedure （手続関数）のひな型です.

name := **proc**(*input sequences*)
local *variable sequence;*
global *variable sequence;*
options *option sequence;*
description *descripiton sequence;*
statement 1;
⋮
statement n;
end proc;

procedure からの戻り値は最後の statement からの出力です.

1.5.3 プログラミングの実践

それでは学生の成績データの平均と分散を求める procedure を題材に実際のプログラミングの流れを見てみましょう. 先ずデータを用意します.

```
> list1:=[48,56,68,72];
```

$$list1 := [48, 56, 68, 72]$$

次にひな型の作成とデータがうまく読み込まれているかの確認をしておきます.
改行は shift+enter でおこないます.

```
> stat:=proc(data1)
> print(data1);
> end;
```

$$stat := \mathbf{proc}(data1)\ \mathrm{print}(data1)\ \mathbf{end\ proc}$$

```
> stat(list1);
```

$$[48, 56, 68, 72]$$

次に実際の処理がうまく動くか確かめておきます. [1]

```
> n:=nops(list1);
> ave:=add(list1[i],i=1..n)/n;
> disp:=0;
> for i from 1 to n do
> disp:=disp+(list1[i]-ave)^2;
> od:
> disp:=disp/n;
```

[1]for を使った総和は教育的なものです. 実際の数値を足しあわせるときには例で示してあるような, add 関数を使ったほうが, はるかに計算速度は速いです.

$$n := 4$$
$$ave := 61$$
$$disp := 0$$
$$disp := 91$$

出力が多すぎる時は od; を od:とセミコロンをコロンに変えることで抑制することができます．これらを先ほど作ったひな型に加え local 変数を宣言しておきます．local variables は proc 内でのみ使われる変数です．proc 内で global を指定して外部の変数を参照することも可能です．同じ名前が使われているときには内部変数が優先されます．

```
> stat:=proc(data1::list)
> local n,ave,disp,i;
> if nargs=0 then ERROR("no argument") fi;
> n:=nops(list1);
> ave:=add(list1[i],i=1..n)/n;
> disp:=0;
> for i from 1 to n do
> disp:=disp+(list1[i]-ave)^2;
> od:
> disp:=disp/n;
> printf("Average       =%10.5f\n",ave);
> printf("Variance      =%10.5f\n",disp);
> printf("Standard disp.=%10.5f\n",evalf(sqrt(disp)));
> end proc;
> stat(list1);
```

stat := **proc**(*data1*::*list*)
local *n*, *ave*, *disp*, *i*;
if nargs = 0 **then** ERROR(“no argument”) **end if**;
n := nops(*list1*);
ave := add(*list1*$_i$, *i* = 1..*n*)/*n*;
disp := 0;
for *i* **to** *n* **do** *disp* := *disp* + (*list1*$_i$ − *ave*)^2 **end do**;
disp := *disp*/*n*;
printf(“Average =%10.5f \n”, *ave*);
printf(“Variance =%10.5f \n”, *disp*);
printf(“Standard disp.=%10.5f \n”, evalf(sqrt(*disp*)))
end proc

```
Average       =  61.00000
```

```
Variance      =  91.00000

Standard disp.=   9.53939
```

ここで，出力の見栄えをよくするために printf 文を使っています．書式は C 言語でおなじみのものですが，慣れない人は help を参照してください．文字列はダブルクォートで囲まれています．またエラー処理を追加しています．さらに最初の引数を data1::list とすることによって，data1 の type が list であるかどうかを Maple が判断してくれます．

上のプログラム中に nargs という変数が出てきています．proc 内には特別の意味を持った nargs と args というのがあります．これまた UNIX 上の C 言語ではおなじみです．nargs は proc がいくつの input を受け取ったかを表しています．それぞれの input は args[i] によって取りだすことが可能です．もうひとつ proc 内で重要な特殊名 procname があり，これはプロシージャの名前が割り当てられています．

もうひとつの重要なコマンドは RETURN で，それ以降の計算が必要ないときに戻り値を持って procedure を抜ける関数です．これを使った再帰的な関数を定義してみましょう．近似で有用な Chebyshev 多項式 です．これを漸化式で表すと

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \quad \text{for n} >= 2 \tag{1.1}$$

で，$T_0(x) = 1$ と $T_1(x) = x$ です．これを Maple で表すと，

```
> T:=proc(n::nonnegint,x::name)
> #Chebyshev polynomials
> option remember;
> if n=0 then RETURN(1);
> elif n=1 then RETURN(x);
> fi;
> 2*x*T(n-1,x)-T(n-2,x);
> end proc;
```

$T := \mathbf{proc}(n::nonnegint,\ x::name)$
$\mathbf{option}\ remember;$
$\quad \mathbf{if}\ n = 0\ \mathbf{then}\ \mathrm{RETURN}(1)\ \mathbf{elif}\ n = 1\ \mathbf{then}\ \mathrm{RETURN}(x)\ \mathbf{end\ if};$
$\quad 2 * x * \mathrm{T}(n-1,\ x) - \mathrm{T}(n-2,\ x)$
$\mathbf{end\ proc}$

となります．option remember は remember table に結果を保存させて計算速度を上げるためにつけてあります．# はコメントに使っています．この結果を plot させてみます．まず，5 次の Chebyshev 多項式までを関数として定義します．“||” は連結作用素で二つの要素を連結してくれます．

```
> for i from 0 to 5 do
> f||i:=unapply(expand(T(i,x)),x);
> od;
```

$$f0 := 1$$

$$f1 := x \to x$$

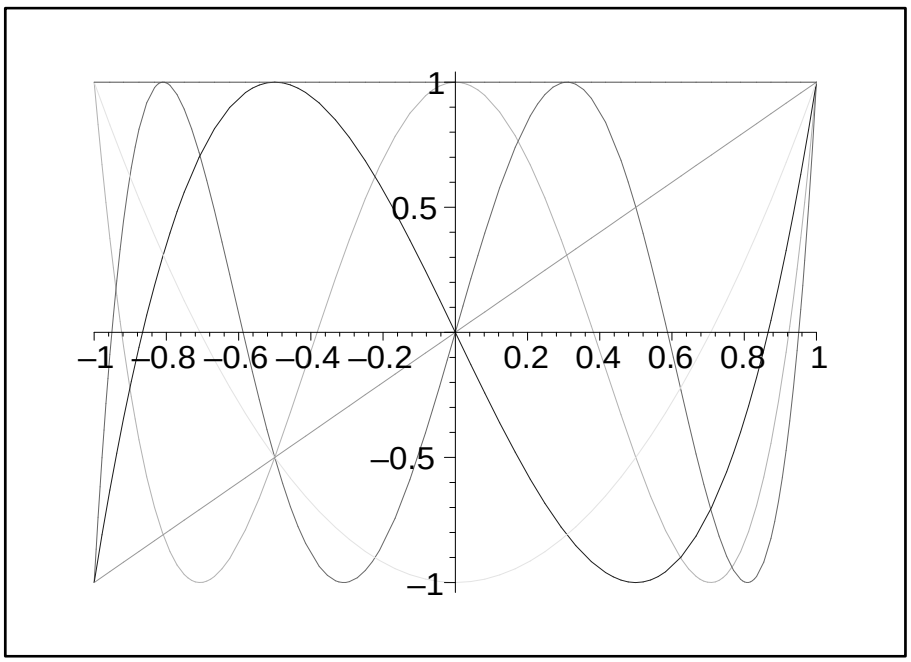
$$f2 := x \to 2\,x^2 - 1$$

$$f3 := x \to 4\,x^3 - 3\,x$$

$$f4 := x \to 8\,x^4 - 8\,x^2 + 1$$

$$f5 := x \to 16\,x^5 - 20\,x^3 + 5\,x$$

```
> plot({f0,f1,f2,f3,f4,f5},-1..1);
```



“Maple V プログラミングガイド” には proc などの厳密な規則と有益な実例がのっています．ぜひ，一読されることをお奨めします．

1.5.4 その他のテクニック

debug に関する注意

プログラムにはバグが付き物です．バグ取りには printlevel,trace, mint,profile 等を使います．この中で一番簡単なのは計算途中の出力の量を変えてくれる，print-

level の設定です．これは printlevel:=11 などとして，設定します．元に戻すには printlevel:=1 としてください．trace は特定のプロシージャの結果だけを調べます．mint は maple とは別のプログラムとして用意され syntax のチェックなどをしてくれ，本格的なプログラムを作るときなどに便利です．(付録??で使用例を紹介しています)．さらに実行速度が問題になるときには profile を使います．その他，help file,user library の作り方も含めて help あるいは “Maple V プログラミングガイド” を参照してください．

定義関数の plot に関する注意

例えば

$$\text{filter}(x) \quad = \quad \begin{array}{l} 1 - x/10 \ \text{ for x} < 10 \\ 0 \ \text{ for x} >= 10 \end{array} \tag{1.2}$$

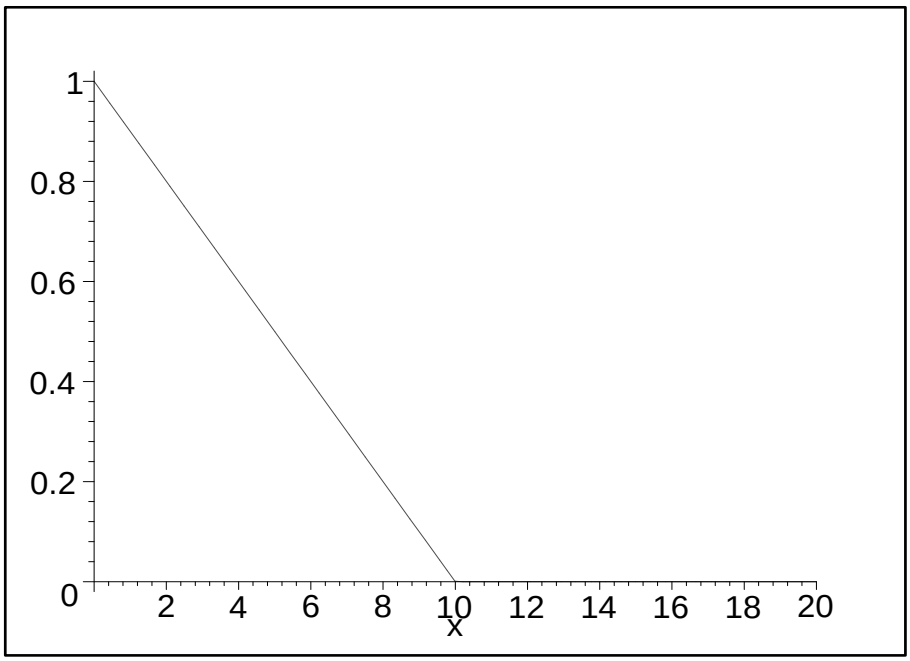
という関数を考えてみましょう．これは

```
> filter:=x-> if x<10 then 1-x/10 else 0 fi;
```

$filter := \mathbf{proc}(x)\,\mathbf{option}\ operator,\ arrow;\ \mathbf{if}\, x < 10\, \mathbf{then}\ 1 - 1/10 * x\, \mathbf{else}\, 0\, \mathbf{end\ if end\ proc}$

となります．うまく定義できているか確認しておきましょう．

```
> plot('filter(x)',x=0..20);
```



ここで，自分で定義した関数の plot を行うときにはシングルクォートで囲む必要があることに注意ください．plot では初めに引数がチェックされます．したがって，ユーザー定義関数では引数が名前であるために if 文などでうまく処理されず error が返ってきます．これを回避するにはシングルクォートを使って引数の評価を行わずに plot コマンドへ渡すことで解決できます．

参考図書

Maple に関する解説書が最近盛んに出版されています．一部を紹介します．

”MapleV による数式処理入門 ” 阿部寛著（講談社，1997)
間違いなく現在日本語で読める絶好の入門書．著者が長年 (たぶん?)，専門で使い込いこまれた経験が凝縮されています．特に微分方程式の具体的な解き方は充実しています．リリース 5 でもほとんど書き換える必要なく使えます．

”はじめての MapleV リリース 4 ” K.M. ヒール，M.L. ハンセン，K.M. リカード著，笠島友美訳（シュープリンガー・フェアラーク東京，1997)
リリース 4 の”Maple V Learning Guide”の訳．リリース 3 やリリース 5 の日本語版がいかにもマニュアルの翻訳という雰囲気で読みにくいのに対して，リリース 4 版は数学を理解している人がコマンドを確認しながら翻訳されたらしく，とても理解しやすくなっています．リリース 5 でも使えます．

”MapleV リリース 5 ラーニングガイド ” K.M. ヒール，M.L. ハンセン，K.M. リカード著，示野信一他訳（シュープリンガー・フェアラーク東京，1998)
新しいリリース 5 の”Maple V Learning Guide”の訳．

“MapleV リリース 5 プログラミングガイド ” M.B. モナガン他著（シュープリンガー・フェアラーク東京，1999)
プログラミングや Maple の内部構造などをより詳細に知りたいときには購入する必要があります．内容は一般的な help では得られないような Maple の概念を統一的に説明しています．ただ，記述が抽象的（数学的？）なため初心者には何を言っているのか全く理解できません．Maple V を系統的に理解し，効率良く使うには是非とも必要です．

”Maple V と利用の実際-数式処理と CG- ” 小国　力著（サイエンス社 1997)
多くの解説書 (MATLAB,Mathematice) を出している著者による Maple V 解説書．広い分野を網羅しており，経験のある研究者が具体的な問題を解くときの足掛かりとなる．

”Maple V で見る数学ワールド” 示野信一著（シュプリンガーフェアラーク社 1999)
Maple を使って広い分野の数学を視覚化しようという意欲的な著書．内容は大学初学年制でも理解できるが，使われている技巧は高度．