数式処理ソフトMapleで理解する数値計算の基礎

西谷滋人
関西学院大学理工学部情報科学科

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

平成17年5月23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．

数式処理ソフト Maple で理解する数値計算の基礎

目 次

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本書の記法を述べます.

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．user interfaceはJava baseで，ほとんどのOS環境で同じ面構えです．linux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terminal上でのcharacter版が立ち上がります.

1.1.2　コマンド入力

それでは簡単な計算を実行させてみましょう．プロンプト (">") に続けていくつかのコマンドを打ち込んでみてください．記号 [*enter*] はenterキーあるいはreturnキー，記号 [*shift+enter*] はshiftキーを押しながらのenterです.

```
> 1+1;[ enter]
```

$$2$$

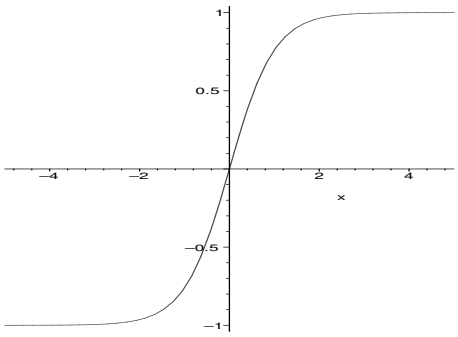
```
> factor(x^2-3*x+2);[ shift+enter]
> 3/2+5/3;[ shift+enter]
> 100!;[ enter]
```

$$(x-1)\,(x-2)$$

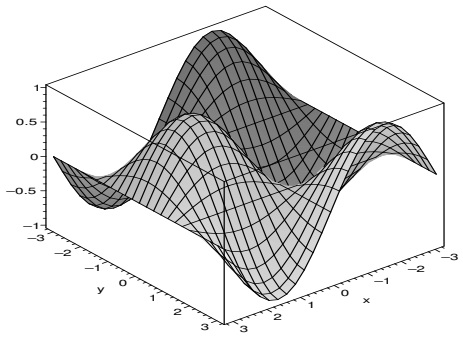
$$\frac{19}{6}$$

933262154439441526816992388562667004907159682643816214685929638952\
175999932299156089414639761565182862536979208272237582511852109168\
640000000000000000000000000

```
> plot(tanh(x),x=-5..5);[ enter]
```




```
> eq1:=sin(x)*cos(y):[ shift+enter]
> plot3d(eq1,x=-Pi..Pi,y=-Pi..Pi);[ enter]
```



- 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 shift+enter で改行を挿入します．
- 入力領域 (デフォルトでは赤く表示されています) のどこかにカーソルを持っていきクリックすると，たて棒が明滅する入力待ち状態になります．そこで enter キーをたたけば，その領域すべてを一度に入力したことになります．
- 入力の順番は enter をいれた順番であり，画面の上下とは関係ありません．
- 最後の；(セミコロン) を忘れがちです．セミコロンはコマンドの区切りを表します．
- 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．
- これ以降の記述では記号 [*enter*] や [*shift+enter*] を省きます．

- これ以降の記述で行頭に続けて複数のプロンプト (">") が表示されているのは，[*shift+enter*] で改行が入力されている「ひとかたまり」の入力領域を意味します．

1.1.3 間違い修正

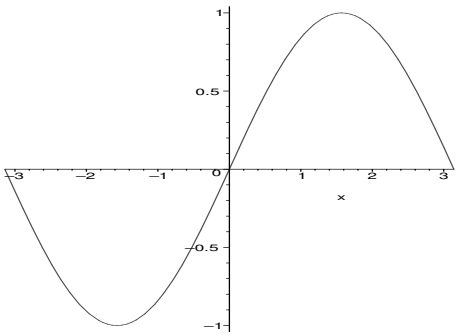
打ち間違いなどの訂正は普通のエディターソフトと同様に，アローキーあるいはマウスのクリックによってできます．訂正して enter を入れれば入力されます．ある領域を選択して，削除・カットおよびペーストなどの作業もマウスを使ってできます．例えばサイン関数を $-\pi$ から $+\pi$ までプロットしようとした時

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

```
Error, (in plot) range values must be real constants
(エラー, (plot 関数内では) 範囲の値は実定数でなければならない)
```

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 ヘルプファイル

Maple が用意しているたくさんのコマンドを全て憶える必要ありません．ヘルプで関連する項目を捜します．英語での検索になりますので初学者には少し酷かもしれません．[1]

ヘルプは？に続けて関連する関数を入れます．

[1]不十分ですが，http://ist.ksc.kwansei.ac.jp/~nishitani/Maple/JtoE.html に和英の対応表を用意しています．これから充実させていきます．

```
> ?plot;
```

で plot に関するヘルプが表示されます．?? 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．ヘルプでは，関数の簡単な説明に続いて，

- Calling Sequence:（呼び出し）
- Parameters:（引数の説明）
- Description:（詳しい解説）
- Examples:（使用例）
- See Also:（関連する項目）

があります．Windows 版では記述は一部日本語に訳されています．英語が分からなくても Examples（使用例）を参考にすればだいたい予測できます．と言うより日本語訳を読んでも初めはチンプンカンプンです．Maple のコマンドのコンセプトに慣れるまでは使用例をまねるのが一番の早道です．

1.2 簡単なコマンド

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．

1.2.1 変数の代入とキャンセル

変数への数値の代入は

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

$$mass := 10$$

によって行われます．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に置き替わるのではなく，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，未定義の free 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`（`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`(evaluate float の略) を使います．

```
> evalf(%);
```

$$2.321928094$$

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(plots):

Warning, the name changecoords has been redefined
```

などでおこなってください.

1.2.3 ユーザー関数の定義

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代入のようにします.

```
> f1:=x->-x*ln(exp(-1/x)/(1-exp(-1/x)));
```

$$f1 := x \to -x \ln\left(\frac{e^{(-\frac{1}{x})}}{1-e^{(-\frac{1}{x})}}\right)$$

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 eq2 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 f2 という関数と定義しています.

```
> eq2:=(1+exp(-1/T))/(1-exp(-1/T)):
> f2:=unapply(eq2,T);
```

$$f2 := T \to \frac{1+e^{(-\frac{1}{T})}}{1-e^{(-\frac{1}{T})}}$$

まちがって数式に対して矢印での定義をすると意図通りに変数に置き変わってくれません．変数 T に値 3 を入れようとして

```
> f2(3);
```

$$\frac{1+e^{(-1/3)}}{1-e^{(-1/3)}}$$

と期待した結果に対して,

```
> f3:=T->eq2;
> f3(3);
```

$$f3 := T \to eq2$$

$$\frac{1+e^{(-\frac{1}{T})}}{1-e^{(-\frac{1}{T})}}$$

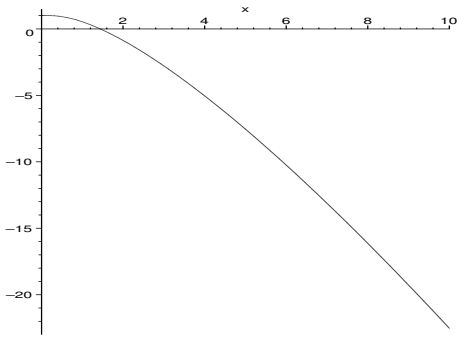
となり変数を変数と認識していないことがわかります.

1.2.4 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した関数を使って，関数が実際にどのような形をしているか plot させてみま

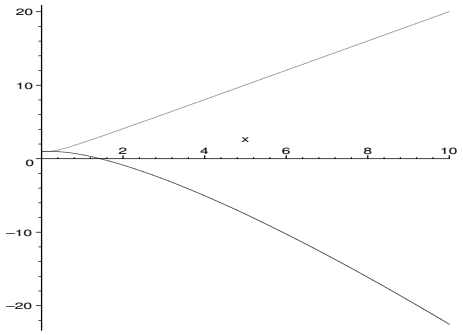
しょう.

```
> plot(f1(x),x=0..10);
```



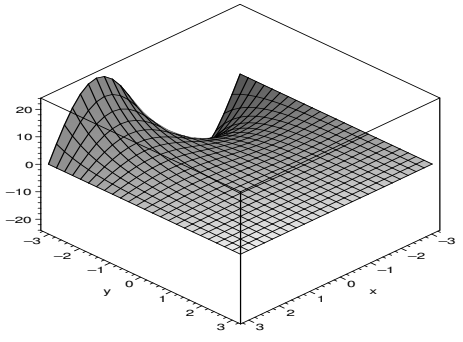
となります. ２つの関数を一つの領域にプロットするには

```
> plot({f1(x),f2(x)},x=0..10);
```



2変数の場合には plot3d を使います.

```
> plot3d(sin(x)*exp(-y),x=-Pi..Pi,y=-Pi..Pi);
```



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．上の出力では外枠をつけるようにメニューバーで指定しています．さらに

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

で呼び出される plots package や plottools package には多くの便利な表示関数が用意されています．詳しい解説は 1.6 節でおこないます．

1.2.5 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};
> solve(eqset,{x,y});
```

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

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

これだけでは

```
> x;y;
```

$$x$$

$$y$$

のように変数 x,y は変数のままです．x,y に解の数値を代入するには assign を使い

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

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

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

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

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

$$0$$

$$1$$

のようにします．代数的に解けない方程式に対しても数値的に解く関数 fsolve があります．

```
> restart;
> f1:=x->-x*ln(exp(-1/x)/(1-exp(-1/x)));
> fsolve(f1(x)=0,x);
```

$$f1 := x \to -x \ln\left(\frac{e^{(-\frac{1}{x})}}{1-e^{(-\frac{1}{x})}}\right)$$

1.442695041

1.2.6　微分

微分は diff によって行います.

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

$$2\,x$$

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.

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

$$2\,y^2$$

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

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

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

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

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

$$(\tfrac{d}{dx}\,\mathrm{X}(x))\,(\tfrac{d}{dt}\,\mathrm{T}(t))$$

1.2.7　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);
```

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

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

```
> eq2:=exp(-x^2);
> int(eq2,x=0..zz);
```

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

$$\frac{1}{2}\sqrt{\pi}\,\mathrm{erf}(zz)$$

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．

演習問題

1. 二つの関数 $x, \cos(x)$ を $x = -5..5$ でプロットせよ．
2. 新しい関数 $f(x) = x - \cos(x)$ を定義して，$x = -5..5$ で表示せよ．
3. `fsolve` を用いて，$x - \cos(x) = 0$ を解け．
4. アインシュタイン結晶の内部エネルギーは

$$E(T) = \frac{1+\exp(-1/T)}{1-\exp(-1/T)} \tag{1.1}$$

で表される (T は規格化した温度)．その温度依存性をプロットせよ．また，比熱 ($C(T) = \mathrm{d}E/\mathrm{d}T$) を求め，その温度依存性をプロットせよ．

1.3　Mapleでのプログラミング

Maple は普通のプログラミング言語と同じくプログラムが組めます．ヘビーな計算には向きませんが，ちょっとしたプログラムなら Maple で十分です．すでに組み込まれている Maple の関数を利用すれば，驚くほど早く実際に動くプログラムが作られます．また，C 言語と似ていますので，書き換えることも容易です．

1.3.1　値の代入，出力

Maple は C 言語などのコンピュータ言語と違って変数の初期設定で型宣言をする必要がありません．

```
> i:=1;
```

$$i := 1$$

```
> x:=3;
> y:=2.0;
> z:=x+y;
```

$$x := 3$$
$$y := 2.0$$
$$z := 5.0$$

出力を明示的におこなうには print を使います．さらに，printf を使えば C 言語と同じ形式で出力整形をすることが可能です．次の例では\t はタブを，\n は改行を表します．また，%d，%e，%f はそれぞれ整数，指数表示，浮動小数点数で x,y,z を表示せよと意図しています．%10.5f などは全部で 10 桁，小数点以下 5 桁を意味します． 複素数の指定も Z で出来ます．

```
> printf("%03d, %5.3e\t%10.5f\n",x,y,z);

003, 2.000e+00    5.00000

> printf("%10.5Zf\n",z+y*I);

   5.00000  +2.00000I
```

1.3.2　算術演算

算術演算子+,-,*,/が通常の和，差，積，商を表します．ただ，商は，

```
> 3/4;
> 3/4+2/5;
```

$$\frac{3}{4}$$

$$\frac{23}{20}$$

のように分数が出てきます.

整数の割り算には irem(余り) と iquo(商) があります.

```
> irem(10,3);
> iquo(10,3);
```

$$1$$

$$3$$

浮動小数点数に直すには evalf を用います. 第2引数が出力桁数を指定します.

```
> evalf(10/3);
```

$$3.333333333$$

```
> evalf(Pi,30);
```

$$3.14159265358979323846264338328$$

浮動小数点演算の有効数字はデフォルトでは 10 桁ですが, Digits に値を代入することによって有効数字を一括して指定することが可能です.

```
> Digits:=20;
> evalf(exp(1));
```

$$Digits := 20$$

$$2.7182818284590452354$$

浮動小数点数から整数に直すにはいくつかの関数があります.

trunc 数値から数直線で 0 に向って最も近い整数

round 数値の四捨五入

floor 数値より小さな最も大きな整数

ceil 数値より大きな最も小さな整数

負の値の時に floor と trunc は違った値を返します. 小数点以下を取りだすには frac が用意されています.

%は C 言語とは全く違った働きが与えられています. 直前の出力結果を参照します. %%は二つ前, ... となっています.

```
> %;
```

$$2.7182818284590452354$$

べき乗は^です.

```
> 1.2^3.4;
```

$$1.8587296919794811670$$

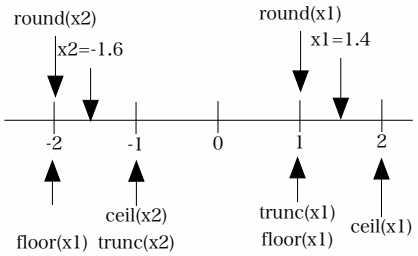


図 1.1: 数直線上での trunc，round，floor，ceil の関係.

1.3.3　プログラムの流れの制御

if-else

条件分岐の基本となる if-else です．構文は

```
if <分岐条件> then
  <動作の記述>
| elif <分岐条件> then <動作の記述> |
| else <動作の記述> |
end if
(| |中は省略してもよい)
```

です．ある値の絶対値を返すには

```
> x:=-4;
> if (x>0) then
>   y:=x;
> else
>   y:=-x;
> end if
```

$$x := -4$$
$$y := 4$$

となります．条件分岐には関係演算子<，<=，>，>=，=，<>や論理演算子 and，or，xor，implies，not が使えます．その他にもブール値を返す関数として evalb, type などいくつかあり，条件分岐に使えます．

do-loop

何回も同じ動作を繰り返す for-loop です．構文は

```
for <変数> from <初期値> | by <増減値> |  to <終値> do
  <動作の記述>
end do;
(| |中はなくともよい)
```

です．具体的には，

```
> total:=0;
> for i from 1 to 10 do
>   total:=total+i;
> end:
> total;
```

$$total := 0$$
$$55$$

```
> for i from 2 by -2 to -4 do
>   i;
> end do;
```

$$2$$
$$0$$
$$-2$$
$$-4$$

です．loop 回数が少ない間は，loop の中身も出力されます．これを止めるには `end do;` の最後のセミコロンをコロンに換えます．while-loop も同じように使えます．

```
while <分岐条件> do
  <動作の記述>
end do;
```

next, break

do-loop の途中で違った操作を加えるための命令です．`next` に来たら do-loop を一回スキップします．`break` に来たらそこで do-loop を一つ抜けます．具体例で見てください．

```
> for i from 1 to 5 do
>   if (i=3) then next; end if;
>   i;
> end do;
```

1
2
4
5

```
> for i from 1 to 5 do
>   if (i=3) then break; end if;
>   i;
> end do;
```

1
2

1.3.4 配列

リスト

[ ] で囲まれた数字などの並びはリストと呼ばれ，添字1から始まる1次元の配列を表します．

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

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

リストの要素を参照するのは以下の通りです．

```
> list1[3];
> list1[-1];
> list1[2..4];
```

3
4
[2, 3, 4]

添字で-1,-2は最後から一つ目，二つ目です．C言語と違い添字0はありません．

```
> list1[0];

Error, invalid subscript selector
```

op コマンドでリストの中身だけを取りだすことが出来ます．これを利用して，リストの頭やおシリに要素を付け加えることが可能です．

```
> op(list1);
```

$$1,\ 2,\ 3,\ 4$$

```
> list1:=[op(list1),5];
```

$$list1 := [1,\ 2,\ 3,\ 4,\ 5]$$

リストの一要素を置換するには以下のようにします．

```
> list1[4]:=x;
> list1;
```

$$list1_4 := x$$

$$[1,\ 2,\ 3,\ x,\ 5]$$

リストの一要素を削除するには以下のようにします．

```
> list1:=subsop(4=NULL,list1);
> list1;
```

$$list1 := [1,\ 2,\ 3,\ 5]$$

$$[1,\ 2,\ 3,\ 5]$$

nops で要素の数を返します．

```
> nops(list1);
```

$$4$$

リストに対してよく使う手を二つ．(”＃”より後ろはコメント文で，Maple は読み飛ばします)

```
> aa:=[];#空で初期化
> for i from 1 to 3 do
>   aa:=[op(aa),i];#付け足していく
> end do:
> print(aa);
```

$$aa := []$$

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

```
> n:=nops(aa); #要素数の取りだし
> total:=0;
> for i from 1 to n do #for-loopの上限に使います
>   total:=total+aa[i];
> end do:
> print(total);
```

$$n := 3$$
$$total := 0$$
$$6$$

[ ] を二重化することで 2 次元の配列を作ることも可能で，リストのリスト (listlist) と呼ばれます.

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

$$aa := [[1,\ 2],\ [3,\ 4]]$$
$$2$$

多重化すれば多次元配列が同様に使えます.

array

array というのは listlist とほとんど同じですが，添字をどこから始めても OK になっています．あらかじめサイズの分かっている配列や，1 からでない添字を使いたいときに便利です.

```
> A:=array(1..3,1..3,diagonal);
> print(A);
```

$$A := \mathrm{array}(diagonal,\ 1..3,\ 1..3,\ [])$$
$$\begin{bmatrix} A_{1,1} & 0 & 0 \\ 0 & A_{2,2} & 0 \\ 0 & 0 & A_{3,3} \end{bmatrix}$$

```
> B:=array(0..1,-1..2,[[1,2,3,4],[5,6,7,8]]):
> B[1,-1];
```

$$5$$

convert で array と listlist などの相互変換が可能.

```
> A2:=convert(aa,array);
> print(A2);
```

$$A2 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$i \times j$ の array の要素数をとりだすには，以下のようにします.

```
> n:=1:
> i:=op(2,op(2,eval(B))[n])-op(1,op(2,eval(B))[n])+1;
> n:=2:
> j:=op(2,op(2,eval(B))[n])-op(1,op(2,eval(B))[n])+1;
```

$$i := 2$$

$$j := 4$$

set

{} によって囲まれた並びは set と呼ばれ，集合とおなじ取り扱いができます．リストと異なり，要素には重複がなく，基本的に順番がありません．したがって，以下の三種の set は同じ集合を意味します．

```
> {x,y,z},{y,z,x},{x,x,y,z,z,x};
```

$$\{x,\ z,\ y\},\ \{x,\ z,\ y\},\ \{x,\ z,\ y\}$$

集合の要素の操作には以下のような，和 (union)，積 (intersect)，差 (minus) が用意されています．

```
> {x,y,z} union {u,v,z};
```

$$\{v,\ x,\ z,\ y,\ u\}$$

```
> {x,y,z} intersect {u,v,z};
```

$$\{z\}$$

```
> {x,y,z} minus {z};
```

$$\{x,\ y\}$$

要素の取出はリストと同様に [ ] を使いますが，順序が Maple の内部事情で変わるので注意が必要です．

```
> {x,y,z}[1];
```

$$x$$

1.3.5 proc 化

いままでに紹介した制御の流れ，データ構造を使って複雑なプログラムを組むことが可能となります．ある程度長くなった，あるいは何度も呼びだすルーチンは関数として登録しておくと便利です．これには proc を使います．ひな形は以下の通りです．

```
<変数名>:=proc(<引数>);
  local <変数>;
```

```
  global <変数>;
  <動作の記述>
end proc;
```

global,local は C 言語と同じ意味で，関数の内部だけで使われるのが local, 外部を参照するのが global です．global,local を省略しても Maple が適当に判断してくれます．したがって最も単純には，あらかじめ作っておいた動作の記述に，変数名:=proc() を頭につけて，おしりを end proc; とすれば即席関数ができ上がります．

リストを受け取ってその和を返す関数を示します．proc の次の ( ) に受け取る引数を書き，戻り値は最後の実行文です．ここでは，a というリストを受け取り，要素数を見て，和をとり，最後にその和を返しています．

```
> total:=proc(a)
> local S,n,i;
> n:=nops(a);
> S:=0;
> for i from 1 to n do
>   S:=S+a[i];
> end do;
> eval(S);
> end proc:#ここまでが関数定義

> aa:=[3,5,7];
> total(aa);
```

$$aa := [3,\ 5,\ 7]$$
$$15$$

演習問題

1. **素数判定プログラムの Maple script**
   C 言語で作った以下の素数判定プログラムを Maple script に書き換えよ．ただし，scanf は値の代入に置き換え，整数の余りは irem 関数を使え．

```
#include <stdio.h>
int main(void){
  int i,n;
```

```
  scanf("%d",&n);
  for (i=n-1;i>1;i--){
    if (n%i==0){
      break;
    }
  }
  if (i==1){
    printf("%d is a prime number.\n",n);
  } else {
    printf("%d is not a prime number.\n",n);
  }
  return 0;
}
```

2. **Google のリクルート**
   グーグル、謎の人材募集広告の問題｛自然対数の底 e の値中の，最初の連続する 10 桁の素数｝を解け.

   使用する Maple 関数：

   `evalf` ：指定した精度で実数表記.

   `floor` ：小数点以下を切り取った整数で表示

   `isprime` ：入力が素数なら true, 違えば false

   まず，`evalf` で 200 桁の `exp(1)` を求め，それぞれの桁の数値を `floor` と 10 倍計算を組み合わせて取りだす (下記参照)．その後，連続する整数を 10 桁の整数に直し，`isprime` で素数かどうかを検証するループを回す.

   それぞれの桁の数値を取りだす部分は，例えば

```
> EE:=2.718;
> i:=1;
> AA[i]:=floor(EE);
> EE:=10*(EE-AA[i]);
```

$$EE := 2.718$$
$$i := 1$$
$$AA_1 := 2$$
$$EE := 7.180$$

   となる.

3. **エラストテネスのふるい**
   100 までの素数をすべて求めて出力するプログラムを作れ.

   前に作った素数判定プログラムに整数を順に代入して調べるのはロスが大きい. **エラストテネスのふるい**が比較的簡単. 初めに 0 から 100 までの配列を用意し 0 で初期化する. これを番兵と見なして, まず 2 の倍数 (2 自身は除く) に全部印をつける. 次に, 3 の倍数 (3 自身は除く) に全部印をつける. こうして, 100 まで繰り返し, 印がつかないで残っている数は「どの数の倍数でもない数」であるから素数である.

   表 1.1: 番兵 (配列) の変化の様子.

要素	1	2	3	4	5	6	7	8	9	10	···	98	99	100
初期値	0	0	0	0	0	0	0	0	0	0	···	0	0	0
2 が終了	0	[0]	0	1	0	1	0	1	0	1	···	1	0	1
3 が終了	0	0	[0]	1	0	1	0	1	1	1	···	1	1	1

4. **双子の素数**
   p が素数で $p+2$ も素数のとき, これらは双子の素数と呼ばれる. 10 以上, 1000 以下の双子の素数を全部見つけて出力せよ.

5. **ゴールドバッハの予想**
   「6 以上の偶数は二つの素数の和として表わされる」という予想を 100 以下の偶数について検証せよ.

6. **Newton-Raphson 法**
   Newton-Raphson 法をもちいて, $x - \cos(x) = 0$ の解を求めよ. ただし, $f'(x) = 1 + \sin x$.

   Newton-Raphson 法は適当な値 x_0 から出発して,

$$x1 = x0 - \frac{f(x0)}{f'(x0)} \tag{1.2}$$

   という操作を順次繰り返し, 解に近づけていく. これは $y = f(x)$ のグラフに接線をひいて x 軸との交点を次の近似値とする事に相当する.

   for-loop で 3 回ほど回して $x0, x1, f(x)$ を出力してみよ. 最後は「ある小さな値 (eps=1.0e-10 など) を仮定して, 前回の近似値との差がこの値以下になれば終了する」というのが常とう手段 (収束判定条件).

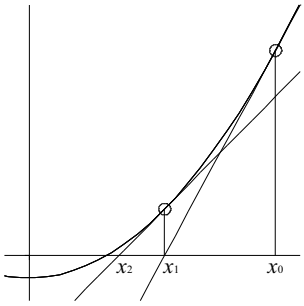


図 1.2: Newton-Raphson 法の模式図.

7. **ルートの距離**
   二つの位置座標

   ```
   0.0 0.0
   1.0 1.0
   ```

   から距離を求める関数を作れ.

   次に，4 つの位置座標

   ```
   x[0]=(0.0, 0.0)
   x[1]=(1.0, 1.0)
   x[2]=(1.0, 0.0)
   x[3]=(0.0, 1.0)
   ```

   を読み込んで，座標順に `[0,1,2,3,0]` と巡る距離を求めよ.

1.4　Mapleの線形代数

線形代数に関連するコマンドです．ベクトル，行列などは array や list を工夫しても使えますが，LinearAlgebra パッケージが用意してくれているデータ構造や関数を使うと，高速・簡単に線形代数計算が出来ます．厄介な逆行列や固有値も行列を生成さえすればすぐに求まります．

1.4.1　ベクトル，行列の生成

先ず

```
> with(LinearAlgebra):
```

が必要です．次に matrix, vector の生成方法ですが，以下のようにいくつもの方法が用意されています．

先ずは標準的な Vector コマンドを使ったベクトルの生成．

```
> v1:=Vector([x,y,z]);
```

$$v1 := \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

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．横 (行) ベクトル (row vector) の生成は以下の通りです．(英語で座席は row, 新聞の囲み記事は column です)

```
> v2:=Vector[row]([x,y,z]);
```

$$v2 := [x,\ y,\ z]$$

[ ] のかわりに，< >と | を使っても縦・横ベクトルが生成できます．

```
> v1:=<x,y,z>;
```

$$v1 := \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

```
> v2:=<x|y|z>;
```

$$v2 := [x,\ y,\ z]$$

次は標準的な Matrix コマンドを使った行列の生成．2 行 3 列の行列を listlist から生成しています．

```
> A0:=Matrix(2,3,[[1,2,3],[4,5,6]]);
```

$$A0 := \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

ベクトルと同様に< >と|を使っての生成です.

```
> A1:=<<1,2,3>|<4,5,6>|<7,8,9>>;
```

$$A1 := \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

特殊な形状を指定する事が出来ます. ここでは単位行列 (shape=identity) を生成しています.

```
> E:=Matrix(3,3,shape=identity);
```

$$E := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

listlist から convert を使った変換です.

```
> A3:=[[1,2],[3,4]];
> A4:=convert(A3,Matrix);
```

$$A3 := [[1, 2], [3, 4]]$$

$$A4 := \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

1.4.2 ベクトル, 行列の演算

大規模なベクトル, 行列の煩雑な演算も, Maple は忠実に実行してくれます. 先ずは和とスカラー積です. 通常の数値, 変数の算術演算と同様の記法です.

```
> A5:=Matrix(2,2,[[3,-1],[1,2]]);
> a*A4+b*A5;
```

$$A5 := \begin{bmatrix} 3 & -1 \\ 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} a+3b & 2a-b \\ 3a+b & 4a+2b \end{bmatrix}$$

"."(ピリオド) が行列の積を表わします.

```
> A4.A4;
```

$$\begin{bmatrix} 7 & 10 \\ 15 & 22 \end{bmatrix}$$

"."は行列とベクトル, あるいはベクトル同士の内積にも使われます.

```
> A1.v1;
```

$$\begin{bmatrix} x+4\,y+7\,z \\ 2\,x+5\,y+8\,z \\ 3\,x+6\,y+9\,z \end{bmatrix}$$

```
> v2.v1;
```

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

行列の積で次元があわないときには Error が返ってきます.

```
> v1.A1;
```

```
Error, (in LinearAlgebra:-VectorMatrixMultiply) invalid input:
LinearAlgebra:-VectorMatrixMultiply expects its 1st argument, v, to be
of type Vector[row] but received Vector[column](3, [...], datatype =
anything, storage = rectangular, order = Fortran_order, shape = [])
```

ベクトルの外積 (outer product) は OuterProductMatrix です.

```
> OuterProductMatrix(v1,v2);
```

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

1.4.3　逆行列，行列式，転置

逆行列は MatrixInverse で一発で求まります.

```
> A3:=Matrix(3,3,[[1,2,1],[4,5,6],[7,8,9]]):
> MatrixInverse(A3);
```

$$\begin{bmatrix} \frac{-1}{2} & \frac{-5}{3} & \frac{7}{6} \\ 1 & \frac{1}{3} & \frac{-1}{3} \\ \frac{-1}{2} & 1 & \frac{-1}{2} \end{bmatrix}$$

行列式は，Determinant です.

```
> Determinant(A3);
```

$$6$$

A1 の MatrixInverse を試みると

```
> MatrixInverse(A1);
```

```
Error, (in LinearAlgebra:-LA_Main:-MatrixInverse) singular matrix
```

と叱られます．これは行列式が 0 だからです．

```
> Determinant(A1);
```

$$0$$

転置 Transpose の使用例です．

```
> Transpose(A0);
```

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
> Transpose(v1);
```

$$[x,\, y,\, z]$$

```
> Transpose(v1).v1;
```

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

1.4.4 固有値

固有値，固有ベクトルは Eigenvectors で一発で求まります．

```
> Eigenvectors(A1);
```

$$\begin{bmatrix} \frac{15}{2} + \frac{3\sqrt{33}}{2} \\ \frac{15}{2} - \frac{3\sqrt{33}}{2} \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{4(\frac{99}{2} + \frac{21\sqrt{33}}{2})}{3(\frac{11}{2} + \frac{3\sqrt{33}}{2})(\frac{13}{2} + \frac{3\sqrt{33}}{2})} & \frac{4(\frac{99}{2} - \frac{21\sqrt{33}}{2})}{3(\frac{11}{2} - \frac{3\sqrt{33}}{2})(\frac{13}{2} - \frac{3\sqrt{33}}{2})} & 1 \\ \frac{\frac{165}{2} + \frac{21\sqrt{33}}{2}}{12(\frac{11}{2} + \frac{3\sqrt{33}}{2})} & \frac{\frac{165}{2} - \frac{21\sqrt{33}}{2}}{12(\frac{11}{2} - \frac{3\sqrt{33}}{2})} & -2 \\ 1 & 1 & 1 \end{bmatrix}$$

浮動小数点に直します．l(lambda) に固有値を，V に固有ベクトルを格納します．

```
> l,V:=evalf(Eigenvectors(A1));
```

$$l,\, V := \begin{bmatrix} 0. \\ 16.11684397 \\ -1.116843970 \end{bmatrix}, \begin{bmatrix} 1. & 0.6861406616 & -2.186140661 \\ -2. & 0.8430703308 & -0.5930703307 \\ 1. & 1. & 1. \end{bmatrix}$$

V の列ベクトルが対応する固有ベクトルです．Column で行列の行を要素とするベクトルが作れます．これを用いて固有値方程式

$$A1.V_2 = \lambda_2 V_2 \tag{1.3}$$

を確かめてみます.

```
> l[2].Column(V,2);
```

$$\begin{bmatrix} 11.0584219844797715 \\ 13.5876329772398865 \\ 16.1168439700000015 \end{bmatrix}$$

```
> A1.Column(V,2);
```

$$\begin{bmatrix} 11.0584219847999989 \\ 13.5876329772000002 \\ 16.1168439695999979 \end{bmatrix}$$

ついでに行ベクトルの取り出しは `Row` です.

演習問題

1. **逆行列**
   次の連立方程式の解を求める.

```
2x + 3y -z = -3
2x +  y -z =  1
 x + 3y +z = -6
```

   (a) 左辺の係数で行列 A をつくる.

   (b) その逆行列 A^{-1} を求める.

   (c) $A^{-1}A = E$ を確認せよ.

   (d) 右辺の値で作ったベクトル b と逆行列を掛け,

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ Ex &= A^{-1}b \end{aligned} \tag{1.4}$$

   より解を求める.

2. **固有値**
   次の対称行列

```
H:=Matrix(2,2,[[1,1],[1,3]]);
```

の固有値を Eigenvalues を使って求めよ．また，

```
H2:= H - x * Matrix(2,2,shape=identity);
```

で作った行列 H2 の行列式から 2 次方程式をつくり，その解を solve を使って求めた結果と比較せよ．

1.5　Mapleの数式処理

数式の変形に関するコマンドをまとめています．手で直すほうが圧倒的に早くきれいになる場合が多いです．しかし，テイラー展開や，複雑な積分公式，三角関数と exp 関数の変換などの手間がかかるところを，Maple は間違いなく変形してくれます．ここで示したコマンドを全て覚える必要は全くありません．というか忘れるものです．ここでは，できるだけコンパクトにまとめて，悩んだときに参照できるようにしたつもりです．初めての人は，ざっと 1 節を眺めた後，2 節の鉄則あたりからじっくりフォローしてください．

1.5.1　コマンド解説

コマンド表

まず数式処理でよく使うコマンドをいくつかの範疇に分類してまとめておきます．このほかにも前節までに示した，`solve`(解), `diff`(微分), `int`(積分) 等は頻繁に数式の導出・変形に登場します．

表 1.2: 式の操作に関係するコマンド

式の変形	式の分割抽出	代入，置換，仮定
simplify:簡単化	lhs, rhs:左辺，右辺	subs:一時的代入
expand:展開	numer, denom:分子，分母	assume:仮定
factor:因数分解	coeff:係数	assuming:一時的仮定
normal:約分・通分	nops, op	assign:値の確定
combine:公式でまとめる		about:仮定の中身
collect:次数でまとめる		anames('user'):使用変数名
sort:昇べき，降べき		restart,a:='a':初期化
convert:形式の変換		

コマンド使用例

式の変形

expand: 展開 `expand(`*exp1*`)`

factor: 因数分解 `factor(`*exp1*`)`

normal: 約分・通分 `normal(`*exp1*`)`

combine: 公式でまとめる `combine(`*exp1*`)`

collect: 次数でまとめる `collect(`*exp1*`,x)`

convert: 形式の変換 `convert(`*exp1,opt*`)`

```
> restart;
> convert(sin(x),exp);
```

$$\frac{-1}{2}\,I\,(e^{(x\,I)} - \frac{1}{e^{(x\,I)}})$$

```
> convert(sinh(x),exp);
```

$$\frac{1}{2}\,e^{x} - \frac{1}{2}\,\frac{1}{e^{x}}$$

```
> convert(exp(I*x),trig);
```

$$\cos(x) + \sin(x)\,I$$

```
> convert(1/(x-1)/(x+3),parfrac);
```

$$-\frac{1}{4\,(x+3)} + \frac{1}{4\,(x-1)}$$

表 1.3: convert による形式の変換

opt	意味
polynom	級数を多項式に変換
trig	三角関数に変換
sincos	tan を含んだ式を sin, cos に
exp	指数関数形式に変換
parfrac	部分分数に変換
rational	浮動小数点数を有利形式に変換

simplify:簡単化 `simplify(`*exp1*`)`, `simplify(`*exp1,* 副関係式`)`

```
> exp1:=3*sin(x)^3-sin(x)*cos(x)^2;
```

$$\mathrm{exp1} := 3\sin(x)^3 - \sin(x)\cos(x)^2$$

```
> simplify(exp1);
```

$$-(4\cos(x)^2 - 3)\sin(x)$$

```
> simplify(exp1,{cos(x)^2=1-sin(x)^2});
```

$$4\sin(x)^3 - \sin(x)$$

sort:昇べき，降べきソート `sort(`*exp1*`)`, `sort(`*exp1*`,[x,y])`,
`sort(`*exp1*`, [x],opts);`*opts=tdeg,plex,ascending,or descending*(それぞれ総次数順序，辞書式順序，昇順，降順)[2]

```
> exp1:=x^3+4*x-3*x^2+1:
> sort(exp1);
```

$$x^3 - 3\,x^2 + 4\,x + 1$$

```
> sort(exp1,[x],ascending);
```

$$1 + 4\,x - 3\,x^2 + x^3$$

```
> exp2:=x^3-3*x*y+4*x^2+y^2:
> sort(exp2);
```

$$x^3 + 4\,x^2 - 3\,x\,y + y^2$$

```
> sort(exp2,[x]);
```

$$x^3 + 4\,x^2 - 3\,y\,x + y^2$$

```
> sort(exp2,[y],descending);
```

$$y^2 - 3\,x\,y + x^3 + 4\,x^2$$

式の分割抽出

lhs, rhs: 左辺，右辺 `lhs(`*exp1=exp2*`)`

numer, denom: 分子，分母 `numer(`*exp1/exp2*`)`

coeff: 係数 `coeff(exp1,x^2)`

op,nops: 要素の取り出し, 要素数 `op(`*exp1*`)`, `nops(`*exp1*`)`

代入，置換，仮定

subs: 一時的代入 `subs(`関係式`,`*exp1*`)`

```
> exp1:=x^2-4*x+4;
> subs(x=a+2,exp1);
```

$$\mathrm{exp1} := x^2 - 4\,x + 4$$

$$(a+2)^2 - 4\,a - 4$$

assume: 仮定 assume(関係式)

assuming: 一時的仮定 exp1 assuming 関係式

```
> sqrt(exp1);
```

$$\sqrt{(-2+x)^2}$$

[2]ascending, descending は Maple9.5 以降

```
> sqrt(exp1) assuming x>2;
```

$$-2 + x$$

additionally: assume に加えての仮定

assign: solve で解いた値の確定.

about: assume で仮定した内容の確認

restart,a='a': 値の初期化

anames('user'): 使っているユーザー定義変数の確認

```
> anames(`user`);
```

exp1, exp2

級数展開

series: 級数展開, `series(exp1,x,4)`

```
> series(exp(x),x);
```

$$1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \mathrm{O}(x^6)$$

```
> series(sin(x),x=Pi/3,2);
```

$$\frac{\sqrt{3}}{2} + \frac{1}{2}(x - \frac{\pi}{3}) + \mathrm{O}((x - \frac{\pi}{3})^2)$$

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

$$\frac{\sqrt{3}}{2} + \frac{x}{2} - \frac{\pi}{6}$$

省略操作

`||`: 連結作用素, 前後の変数をくっつけて新たな変数とする.

```
> a||1;
> a||b;
```

$$a1$$

$$ab$$

seq: for-loop の単純表記

```
> seq(i,i=0..3);
```

0, 1, 2, 3

map: 関数の割り当て

これらを組み合わせて実行すると, 効率的に式を扱うことが出来る.

```
> map(sin,[seq(a||i,i=0..3)]);
```

$$[\sin(a0), \sin(a1), \sin(a2), \sin(a3)]$$

その他

add, mul: 単純な和，積

sum, product: 公式にも対応した和，積.

```
> add(x^i,i=1..3);
```

$$x + x^2 + x^3$$

```
> add(x^i,i=1..n);
```

```
Error, unable to execute add
```

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

$$\frac{x^{(n+1)}}{x-1} - \frac{x}{x-1}$$

```
> mul(x^i,i=1..3);
```

$$x^6$$

```
> mul(x^i,i=1..n);
```

```
Error, unable to execute mul
```

```
> product(x^i,i=1..n);
```

$$\prod_{i=1}^{n} x^i$$

limit: 極限

```
> limit(exp(-x),x=infinity);
```

$$0$$

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

$$\infty$$

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

$$-\infty + \infty\, I$$

1.5.2　鉄則とその具体例

Maple をはじめとする数式処理ソフトの習得にあたって初心者がおちいる共通の過ちを回避する鉄則があります．それは

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る．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;` を入力して初期状態からはじめる．

鉄則１出力してみる： 多くのテキストではページ数の関係で出力を抑止しているが，初心者が問題を解いていく段階ではデータやグラフをできるだけ多く出力する．

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る．`plot` で `Plotting error,empty plot` が出た場合にチェック．

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む．

です．

例えば，複雑な積分として以下のような問題があったとします．

$$\int_{-\infty}^{\infty} x e^{-\beta c x^2} \left(1+\beta g x^3\right) dx \tag{1.5}$$

最新の Maple では改良が施されていて，このような複雑な積分も一発で

```
> f1:=unapply(x*exp(-beta*c*x^2)*(1+beta*g*x^3),x);
```

$$f1 := x \to x\, e^{(-\beta\, c\, x^2)} \left(1+\beta\, g\, x^3\right)$$

```
> int(f1(x),x=-infinity..infinity);
```

$$\begin{cases} \dfrac{3\, g\, \sqrt{\pi}}{4\, \beta\, c^2\, \sqrt{\beta\, c}} & \mathrm{csgn}(\beta\, c) = 1 \\ \infty & otherwise \end{cases}$$

と求まるようになっています．ここでは βc が正の場合 (csgn(βc)=1) とそれ以外の場合 (otherwise) に分けて答えを返しています．しかしこのような意図したきれいな結果を Maple が返してくれるとは限りません．これだけですと，なにかうまくいかないときにお手上げになってしまいます．このようなきれいで簡単な結果に行き着く前の，裏でおこなういくつかの予備計算を省略せずに示します．

先ず鉄則 0 にしたがって `restart` をかけ，関数を定義します．

```
> restart;
> f1:=unapply(x*exp(-beta*c*x^2)*(1+beta*g*x^3),x);
```

$$f1 := x \to x\, e^{(-\beta\, c\, x^2)} \left(1+\beta\, g\, x^3\right)$$

次には鉄則 1 にしたがって積分する前にどのような関数かプロットしてみます．
そのまま `plot` へ投げると

```
> plot(f1(x),x=-10..10);

Warning, unable to evaluate the function to numeric values in the
region; see the plotting command's help page to ensure the calling
sequence is correct

Plotting error, empty plot
```

と怒られます．これは鉄則２にあるとおり，数値を代入すれば

```
> f1(10);
```

$$10\,e^{(-100\,\beta\,c)}\,(1+1000\,\beta\,g)$$

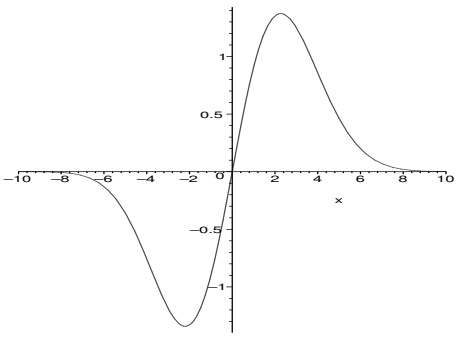
で，beta,c,gなどのパラメータの値が入っていないためとわかります．適当に

```
> c:=1; g:=0.01; beta:=0.1;
```

$$c := 1$$
$$g := 0.01$$
$$\beta := 0.1$$

として，変数に値を代入し，再度プロットを試みると

```
> plot(f1(x),x=-10..10);
```



とグラフを書いてくれます．これから $-\infty..\infty$ の積分がなんらかの値を取ることが期待できます．

さらに鉄則３にしたがって，式を頭から打ち込むのではなく内側からみていきます．これは問題を解いていく時に思考が必ずたどるであろう順番に相当します．先ず変数に入れた数値をクリアします．

```
> c:='c'; g:='g'; beta:='beta';
```

$$c := c$$
$$g := g$$
$$\beta := \beta$$

不定積分でこの関数が積分できることを確認し，

```
> int(f1(x),x);
```

$$-\frac{1}{2}\frac{1}{e^{(\beta c x^2)}\beta c}+\beta g\left(-\frac{1}{2}\frac{x^3 e^{(-\beta c x^2)}}{\beta c}+\frac{3}{2}\frac{-\frac{1}{2}\frac{x e^{(-\beta c x^2)}}{\beta c}+\frac{1}{4}\frac{\sqrt{\pi}\,\mathrm{erf}(\sqrt{\beta c}\,x)}{\beta c\sqrt{\beta c}}}{\beta c}\right)$$

次に x=-alpha..alpha の定積分を実行してみます.

```
> int(f1(x),x=-alpha..alpha);
```

$$-\frac{1}{4}\frac{g\,(4\,\alpha^3 e^{(-\beta c\alpha^2)}\beta c\sqrt{\beta c}+6\,\alpha e^{(-\beta c\alpha^2)}\sqrt{\beta c}-3\sqrt{\pi}\,\mathrm{erf}(\sqrt{\beta c}\,\alpha))}{\beta c^2\sqrt{\beta c}}$$

さらに alpha$\to\infty$ としてみます

```
> limit(int(f1(x),x=-alpha..alpha),alpha=infinity);
```

$$\lim_{\alpha\to\infty}-\frac{1}{4}\frac{g\,(4\,\alpha^3 e^{(-\beta c\alpha^2)}\beta c\sqrt{\beta c}+6\,\alpha e^{(-\beta c\alpha^2)}\sqrt{\beta c}-3\sqrt{\pi}\,\mathrm{erf}(\sqrt{\beta c}\,\alpha))}{\beta c^2\sqrt{\beta c}}$$

ところがこれでは答えを返してくれません. 積分した後のそれぞれの項を見ると beta*c>0 を仮定すれば簡単になることが分かります. このような変数の仮定 (assume) は

```
> assume(beta*c>0);
```

でおこないます. 結果として最初に出した解答

```
> limit(int(f1(x),x=-alpha..alpha),alpha=infinity);
```

$$\frac{3\sqrt{\pi}\,g}{4\,\beta\,c^2\sqrt{\beta\,c}}$$

が導かれるのです. 数式処理ソフトでの数式処理とは数式処理ソフトが『自動的にやって』くれるのではなく, 実際に紙と鉛筆で解いていく手順を数式処理ソフトに『やらせる』のだということを肝に銘じてください.

1.5.3 実戦例

どうしても解かなければならない課題を前にコマンドリファレンスのあちこちを参照しながら解いていくのが数式処理を修得する最速法です. しかし, なかなか共通する適当な課題がありません. 以下では大学物理の初歩として比較的多くの人が出会う「熱膨張係数の導出 (キッテル固体物理)」と「トンネル効果 (シッフ量子力学)」を取り上げます. 著者がどのようないじくり方をしているかを眺めてみてください. その前に「式のフォローのデフォルト」です.

式のフォローのデフォルト

Maple で実際に数式をいじる状況というのは，ほとんどの場合が既知の数式変形のフォローです．例えば，論文で「(1) 式から (2) 式への変形は自明である」とかいう文章で済ましている変形が本当にあっているのかを確かめたい時です．一番単純なやり方は自明と言われた前後の式が一致していることを確かめるだけで十分です．最も単純には

```
> ex1:=(x-3)^4;
```

$$ex1 := (x-3)^4$$

```
> ex2:=x^4-12*x^3+54*x^2-108*x+81;
```

$$ex2 := x^4 - 12\,x^3 + 54\,x^2 - 108\,x + 81$$

```
> expand(ex1-ex2);
```

$$0$$

というように，変形の前後の式を手入力してその差を expand した結果が 0 か否かでします．0 ならば式の変形は保証されていますから，その導出が間違いでなく誤植などもないことだけは信じられます．ただ，これだけでは変形の哲学や技法が身に付くわけではありませんので，あくまでも苦し紛れのデフォルトであることは心に留めておいてください．リバースエンジニアリングがいかに楽であり，かつ危険かが分かります．

熱膨張係数の導出

（参考 キッテル著 固体物理学入門 宇 野良清他訳，丸善 1978 ）熱膨張 (thermal expansion) は原子間ポテンシャルの 3 次以上の項によって現れます．平衡点からの原子の変位 x のポテンシャルエネルギーを

$$U(x) = cx^2 - gx^3 \tag{1.6}$$

と取ることができます．2 次までの項では古典的な調和振動子を表し，熱膨張は現れません．x^3 の項は原子間相互作用の非対称性を表し，この項が熱膨張係数と直接かかわってきます．

有限温度での平均の変位は，ボルツマン分布関数を計算することで求まります．平均の位置 x は，熱力学的な確率で重みづけられ，

$$\langle x \rangle = \frac{\int_{-\infty}^{\infty} x \exp\left(-\beta U(x)\right) dx}{\int_{-\infty}^{\infty} \exp\left(-\beta U(x)\right) dx} \tag{1.7}$$

で計算できます．ここで $\beta \equiv 1/(k_{\mathrm{B}}T)$ です．この積分を実行すると

$$\langle x \rangle = \frac{3g}{4\beta c^2} \tag{1.8}$$

となります．

最後の導出が問題です．ここでは先ず近似によって式を簡単化します．これはお決まりの Taylor 展開です．味噌はポテンシャルエネルギーの項で 2 次の項はそのまま残し，3 次以上を展開することです．実際に Maple で展開してみると，

```
> restart;
> U:=c*x^2-g*x^3;
> eU:=expand(exp(-beta*U));
```

$$U := c\,x^2 - g\,x^3$$

$$eU := \frac{e^{(\beta\,g\,x^3)}}{e^{(\beta\,c\,x^2)}}$$

```
> ex:=convert(series(numer(eU),x,4),polynom);
```

$$ex := 1 + \beta\,g\,x^3$$

```
> f1:=ex/denom(eU);
```

$$f1 := \frac{1 + \beta\,g\,x^3}{e^{(\beta\,c\,x^2)}}$$

という式が導かれます．一見簡単に見えるかもしれませんが，一つ一つを内側から入力してコマンド表を参照し，出力を見ながらどのように変形が進んでいくかを確認してください．

次にこの式の積分です．分子と分母を別々に積分してみます．鉄則の具体例で例示した積分です．

```
> den:=int(f1,x=-infinity..infinity);
```

$$den := \begin{cases} \dfrac{\sqrt{\pi}}{\sqrt{\beta\,c}} & \mathrm{csgn}(\beta\,c) = 1 \\ \infty & otherwise \end{cases}$$

```
> num:=int(x*f1,x=-infinity..infinity);
```

$$num := \begin{cases} \dfrac{3\,g\,\sqrt{\pi}}{4\,\beta\,c^2\,\sqrt{\beta\,c}} & \mathrm{csgn}(\beta\,c) = 1 \\ \infty & otherwise \end{cases}$$

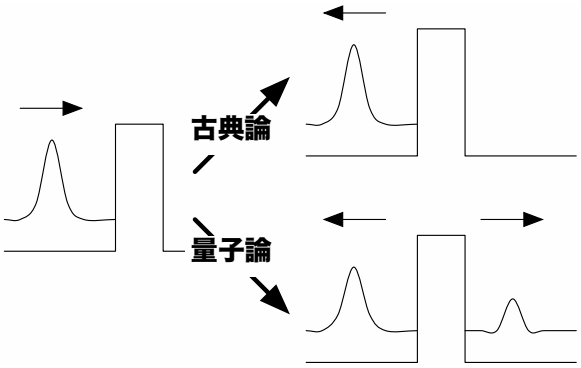
`beta c >0` の場合とそれ以外の結果を別々に返しています．題意から明らかなように `beta c >0` ですので，それを仮定 (`assume`) して，分子÷分母を実行します．

```
> assume(beta*c>0):
```

```
> num/den;
```

$$\frac{3\,g}{4\,\beta\,c^2}$$

トンネル効果



量子効果の基礎的な例である 1 次元のトンネル効果についての式を導いてみましょう[3]．上図のようなポテンシャルを考えます．詳しい解説は成書を参考にしていただくとして，以下で扱う数式の簡単な説明だけをしておきます．まず，ポテンシャルエネルギー $V(x) = 0$ の領域での波動方程式は

$$-\frac{\hbar^2}{2m}\frac{d^2\varphi(x)}{dx^2} = \varepsilon\varphi(x) \tag{1.9}$$

ですから，波動関数は

$$\begin{aligned} &x \le 0 \quad \text{では} \quad \varphi(x) = A\exp(ikx) + B\exp(-ikx) \\ &x \ge a \quad \text{では} \quad \varphi(x) = C\exp(ikx). \end{aligned} \tag{1.10}$$

ここで $k = \sqrt{2m\varepsilon/\hbar^2}$ は波数ベクトルの大きさです．

ポテンシャルの壁の内側では $\varepsilon \lesseqgtr V_0$ によって事情が変わってきます．$\varepsilon \ge V_0$ では $\kappa = \sqrt{2m(\varepsilon - V_0)/\hbar^2}$ と定義すると，波動関数は

$$0 \le x \le a \text{ では} \quad \varphi(x) = F\exp(i\kappa x) + G\exp(-i\kappa x) \tag{1.11}$$

[3]シッフ著 量子力学（井上 健訳），吉岡書店 1970．小出昭一郎著 基礎物理学選書 5A ー量子力学，裳華房 1969．

となります．波動関数は粒子の座標に関する滑らかな連続関数でなければならないという条件を $x=0$ と $x=a$ に適用すると，条件はそれぞれ

$$\begin{aligned}
&x=0 \quad \text{で } \varphi(x) \text{ が連続：} \quad A+B=F+G \\
&x=0 \quad \text{で } \varphi'(x) \text{ が連続：} \quad k(A-B)=\kappa(F-G) \\
&x=a \quad \text{で } \varphi(x) \text{ が連続：} \quad F\exp(i\kappa a)+G\exp(-i\kappa a)=C\exp(ika) \\
&x=a \quad \text{で } \varphi'(x) \text{ が連続：} \quad \kappa F\exp(i\kappa a)-\kappa G\exp(-i\kappa a)=kC\exp(ika)
\end{aligned} \tag{1.12}$$

で与えられます．係数が 5 個で，方程式が 4 個ですから，それぞれの係数の比だけが求まります．これらから F,G を消去して，B/A：入射波と反射波の複素振幅の比，および C/A：入射波と透過波の複素振幅の比が求まります．これらの二乗が反射係数と透過係数にほかなりません．結果は

$$\begin{aligned}
\left|\frac{B}{A}\right|^2 &= \left[1+\frac{4k^2\kappa^2}{(k^2-\kappa^2)^2\sin^2\kappa a}\right]^{-1} = \left[1+\frac{4\varepsilon(\varepsilon-V_0)}{V_0^2\sin^2\kappa a}\right]^{-1} \\
\left|\frac{C}{A}\right|^2 &= \left[1+\frac{(k^2-\kappa^2)^2\sin^2\kappa a}{4k^2\kappa^2}\right]^{-1} = \left[1+\frac{V_0^2\sin^2\kappa a}{4\varepsilon(\varepsilon-V_0)}\right]^{-1}
\end{aligned} \tag{1.13}$$

となります．

$0<\varepsilon<V_0$ ならば，$\alpha=\sqrt{2m(V_0-\varepsilon)/\hbar^2}$ として波動関数は

$$0\le x\le a \text{ では} \varphi(x)=F\exp(\alpha x)+G\exp(-\alpha x) \tag{1.14}$$

です．同様な計算によって

$$\left|\frac{C}{A}\right|^2=\left[1+\frac{V_0^2\sinh^2\alpha a}{4\varepsilon(\varepsilon-V_0)}\right]^{-1} \tag{1.15}$$

となります．$mV_0a^2/\hbar^2=8$ の場合の透過率を求めると下図のようになります．エネルギーがポテンシャル障壁よりも小さい条件 $|E/V_0|<1$ でも透過波の比強度 $|C/A|^2$ が有限の値を取る現象がトンネル効果です．

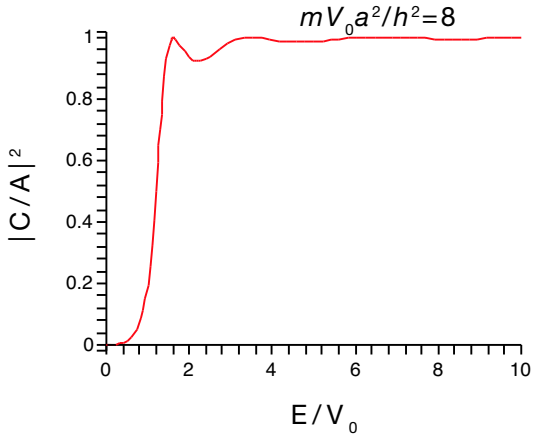
実際に C/A を導出して，その振る舞いをプロットさせてみましょう．先ずは波動関数の定義です．

```
> restart;
> psi1:=x->A*exp(I*k*x)+B*exp(-I*k*x);
```

$$\psi 1 := x \to A\,e^{(k\,x\,I)} + B\,e^{(-I\,k\,x)}$$

```
> psi2:=x->E*exp(I*kappa*x)+F*exp(-I*kappa*x);
```

$$\psi 2 := x \to E\,e^{(\kappa\,x\,I)} + F\,e^{(-I\,\kappa\,x)}$$




```
> psi3:=x->C*exp(I*k*x);
```

$$\psi 3 := x \to C\, e^{(k\, x\, I)}$$

次に $x = 0, x = a$ での 0 次，1 次微分の連続条件を入れます．

```
> eq1:=psi1(0)=psi2(0);
```

$$eq1 := A + B = E + F$$

```
> eq2:=simplify(subs(x=0,diff(psi1(x),x))=subs(x=0,diff(psi2(x),x)));
```

$$eq2 := A\, k\, I - B\, k\, I = E\, \kappa\, I - F\, \kappa\, I$$

```
> eq3:=psi2(a)=psi3(a);
```

$$eq3 := E\, e^{(\kappa\, a\, I)} + F\, e^{(-I\, \kappa\, a)} = C\, e^{(k\, a\, I)}$$

```
> eq4:=simplify(subs(x=a,diff(psi2(x),x))=subs(x=a,diff(psi3(x),x)));
```

$$eq4 := \kappa\, (E\, e^{(\kappa\, a\, I)} - F\, e^{(-I\, \kappa\, a)})\, I = C\, k\, e^{(k\, a\, I)}\, I$$

この 4 つの方程式を A,B,C,E,F について解きます．

```
> solve({eq1,eq2,eq3,eq4},{A,B,C,E,F});
```

$$\left\{A = -\frac{1}{4}\,\frac{e^{(k\,a\,I)}\,(-k^2\,e^{(-I\,\kappa\,a)} + k^2\,e^{(\kappa\,a\,I)} - \kappa^2\,e^{(-I\,\kappa\,a)} + \kappa^2\,e^{(\kappa\,a\,I)} - 2\,k\,\kappa\,e^{(-I\,\kappa\,a)} - 2\,k\,\kappa\,e^{(\kappa\,a\,I)})\,C}{k\,\kappa\,e^{(\kappa\,a\,I)}\,e^{(-I\,\kappa\,a)}}\right.$$

$$, C = C, B = -\frac{1}{4}\,\frac{(k-\kappa)\,e^{(k\,a\,I)}\,(k+\kappa)\,(-e^{(-I\,\kappa\,a)} + e^{(\kappa\,a\,I)})\,C}{k\,\kappa\,e^{(\kappa\,a\,I)}\,e^{(-I\,\kappa\,a)}}, F = -\frac{1}{2}\,\frac{e^{(k\,a\,I)}\,(k-\kappa)\,C}{\kappa\,e^{(-I\,\kappa\,a)}},$$

$$\left. E = \frac{1}{2}\,\frac{e^{(k\,a\,I)}\,(k+\kappa)\,C}{\kappa\,e^{(\kappa\,a\,I)}}\right\}$$

```
> assign(%);
```

assign で確定しておきます．次に実際に A/C とその複素共役 (conjugate) との

積から比強度を出すのですが，うまく複素共役を取れるように係数 kappa，k,a が実数を仮定していることを Maple に教えてやります．

```
> assume(kappa,real);assume(k,real),assume(a,real);
```

conjugate を取ってから三角関数 (trig) へ変換すると式がややこしくなりましたので，先に三角関数へ convert してから複素共役を取ります．

```
> CC:=convert(A/C,trig):
> CC1:=combine(conjugate(CC)*CC);
```

$$CC1 := \frac{1}{8}\,\frac{2\,\kappa^2\,k^2\cos(2\,\kappa\,a) + 6\,\kappa^2\,k^2 - k^4\cos(2\,\kappa\,a) - \kappa^4\cos(2\,\kappa\,a) + k^4 + \kappa^4}{\kappa^2\,k^2}$$

後はその単純化です．

```
> C_num:=simplify(expand(numer(CC1)),
> {cos(kappa*a)^2=1-sin(kappa*a)^2,
> cos(k*a)^2=1-sin(k*a)^2});
```

$$C\_num := 8\,\kappa^2\,k^2 + (-4\,\kappa^2\,k^2 + 2\,k^4 + 2\,\kappa^4)\sin(\kappa\,a)^2$$

```
> C_den:=denom(CC1);
```

$$C\_den := 8\,\kappa^2\,k^2$$

```
> saa:=sin(kappa*a);
```

$$saa := \sin(\kappa\,a)$$

```
> CC2:=collect(C_num/C_den,saa);
```

$$CC2 := 1 + \frac{1}{8}\,\frac{(-4\,\kappa^2\,k^2 + 2\,k^4 + 2\,\kappa^4)\sin(\kappa\,a)^2}{\kappa^2\,k^2}$$

これで透過率が求まりました．次に，いくつかの条件式から k,kappa,a を求めます．

```
> NN:=8;
> a2:='a2';
> a2:=solve(m*V0*a2/h2=NN,a2);
> kappa2:=2*m*(epsilon-V0)/h2;
> k2:=2*m*epsilon/h2;
```

$$NN := 8$$

$$a2 := a2$$

$$a2 := \frac{8\,h2}{m\,V0}$$

$$\kappa 2 := \frac{2\,m\,(\varepsilon - V0)}{h2}$$

$$k2 := \frac{2\,m\,\varepsilon}{h2}$$

```
> CC3:=simplify(
> subs({k=sqrt(k2),kappa=sqrt(kappa2)},coeff(CC2,saa^2)));
```

$$CC3 := \frac{V0^2}{4\,(\varepsilon - V0)\,\varepsilon}$$

```
> CC4:=simplify(subs(epsilon=x*V0,CC3));
```

$$CC4 := \frac{1}{4\,(x-1)\,x}$$

```
> a2a2:=simplify(subs(epsilon=x*V0,sqrt(kappa2*a2)));
```

$$a2a2 := 4\sqrt{x-1}$$

```
> CC5:=1+CC4*sin(a2a2)^2;
```

$$CC5 := 1 + \frac{1}{4}\,\frac{\sin(4\sqrt{x-1})^2}{(x-1)\,x}$$

この式を関数と定義してその振る舞いをプロットします.

```
> f1:=unapply(CC5,x);
```

$$f1 := x \to 1 + \frac{1}{4}\,\frac{\sin(4\sqrt{x-1})^2}{(x-1)\,x}$$

```
> plot(1/f1(x),x=0..10);
```

この結果が冒頭に示した透過率の図です.

課題

1. **必須課題**
   鉄則の具体例と実戦例 (熱膨張係数の導出とトンネル効果) を (i) コマンドの内側から入力する，(ii) それぞれのコマンドの意味をテキストで参照する，に注意してフォローしなさい.

2. **自由課題**
   実戦例のトンネル効果で，$0 < \varepsilon < V_0$ の場合に得られる `sinh` を含んだ式を導出せよ．例で示した `sin` を含んだ式の導出を少し換えれば求まる．ただし，

$$\cosh^2(\alpha * a) = 1 + \sinh^2(\alpha * a) \tag{1.16}$$

   であることに注意.

1.6 Mapleによるグラフ作成

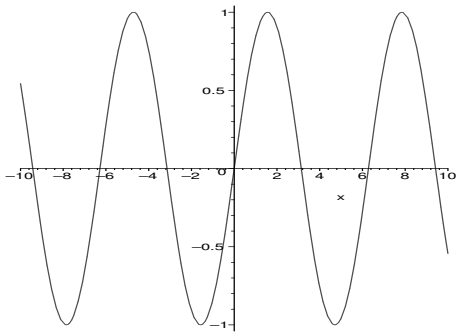
関数の視覚化などの便利な道具が多数用意されています．基本となるグラフ作成に絞っていくつかの手法を紹介します．

1.6.1 簡単なグラフ化

plot

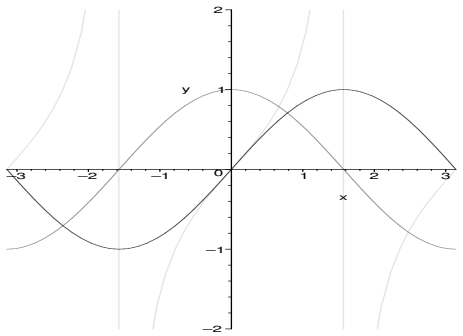
もっとも簡単に関数を表示するには，表示したい関数とその変数を指定するだけで，

```
> plot(sin(x),x);
```



と表示してくれます．さらにいくつもの関数を同時に表示したいときには，集合({})あるいはリスト([])でくくってplot関数に入れます．

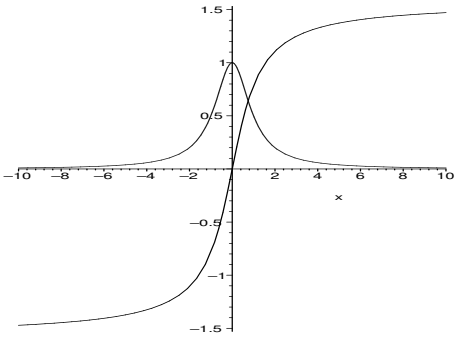
```
> plot({sin(x),cos(x),tan(x)},x=-Pi..Pi,y=-2..2);
```



ここではさらに定義域と表示域を指定しています．その他のグラフの細かいオプション指定は?plot[options];で説明されています．対話的におこなう，plots[interactive]();もありますが，遅いです．

さらに plots パッケージの display を使って複数のグラフを表示することも可能です.

```
> with(plots):
> p1:=plot(arctan(x),x,color=black):
> p2:=plot(diff(arctan(x),x),x,color=blue):
> display(p1,p2);
```



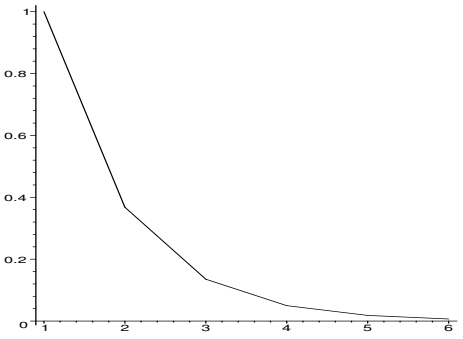
ここでは，p1 と p2 にそれぞれの関数のプロットを代入して，display 関数で一度に表示しています.

listplot

離散的なデータは listplot で表示してくれます.

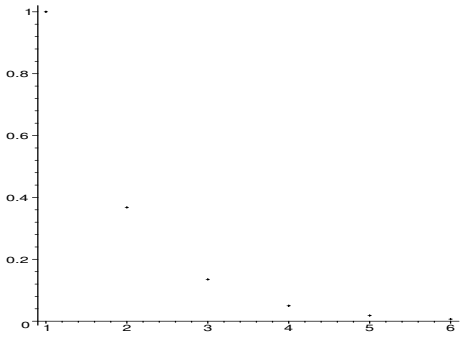
```
> with(plots):
> T:=[seq(exp(-i),i=0..5)];
> listplot(T);
```

$$T := [1,\ e^{(-1)},\ e^{(-2)},\ e^{(-3)},\ e^{(-4)},\ e^{(-5)}]$$



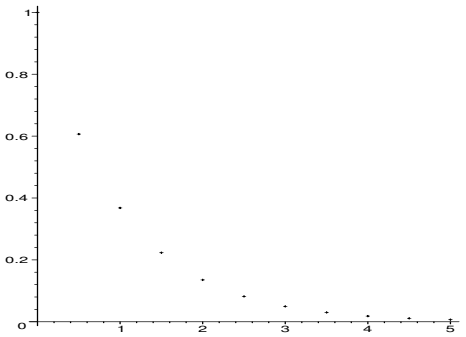
listplot は受け取った list の要素を y_i に，1 から始まる添字を x_i にして，デフォルトでは線でグラフを書きます.

```
> listplot(T,style=point);
```



とすると point で書いてくれます．それぞれの値の横軸 x_i が $1, 2, 3, \cdots$ では不都合なときには，

```
> T:=[seq([i/2,exp(-i/2)],i=0..10)]:
> pointplot(T);
```



として，2 次元の listlist 構造を用意し，$[x_i, y_i]$ を入れて pointplot 関数で表示します．listplot のように線でつなぎたい時には，

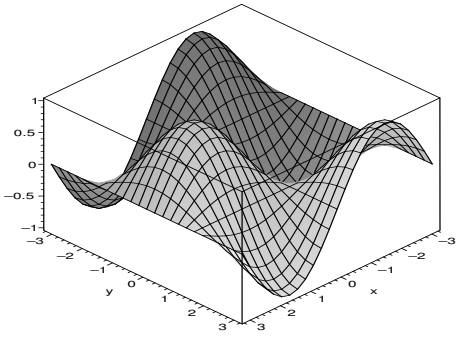
```
> pointplot(T,connect=true);
```

とします．

plot3d

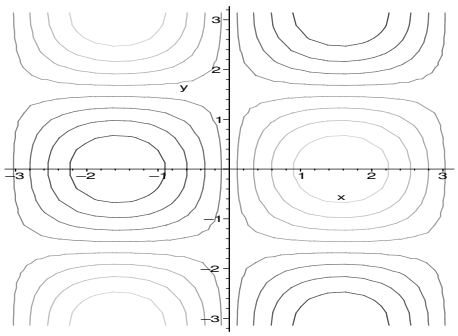
2 次元の定義域を持つスカラー関数，つまり 3 次元の関数は plot3d 関数で表示されます．

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



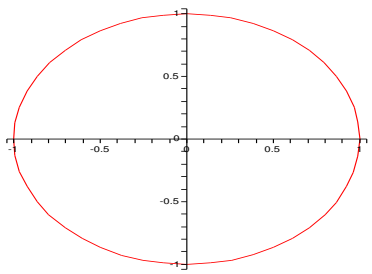
等高線図なども書けます.

```
> contourplot(sin(x)*cos(y),x=-Pi..Pi,y=-Pi..Pi);
```



これらの関数を使うときには plots パッケージを呼びだす必要があります. `with(plots);` とすれば用意されている全ての関数が表示されます. 少し特殊な場合で, 円を描くには parameteric plot によって以下のようにして書きます.

```
> plot([sin(t),cos(t),t=0..2*Pi]);
```



1.6.2 Mapleの描画関数の覚書

Mapleにはいくつかの描画レベルに合わせた関数が用意されています．それぞれの関数がどのような意図で作られ，どのような時にどの関数を使うのが正しいのか迷ってしまうことがあります．基本的には以下のようにパッケージごとに大まかな目的があり，それぞれの機能を担っています．

描画の下位関数 plot[structure]にあるPLOT,PLOT3Dデータ構造が一番下でCURVES，POINTS，POLYGONS，TEXTデータを元に絵を描く．

plotsパッケージ 簡単にグラフを書くための道具．たとえばpointplotは，pointを使って関数を表示する事を当初の目的としている．その他，animate，listplot，logplot，polarplot,contourplotなどもここに入っている．

plottoolsパッケージ PLOTよりもう少し上位で，グラフィックスの基本形状を生成してくれる関数群．arc，arrow，circle，curve，line，point，sphereなどの関数があり，PLOT構造を吐く．表示にはplots[display]を使う．

1.6.3 アニメーション

動画(アニメーション)の作成法についてです．plotsパッケージにあるanimate関数を使います．構文は以下の通りで，[]内に動画にしたい関数を定義し，tで時間を変えていきます．

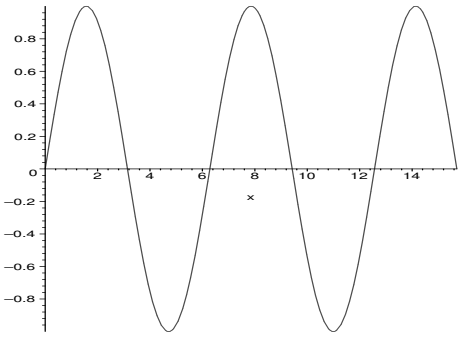
```
> with(plottools):with(plots):
> animate(plot, [sin(x-t),x=0..5*Pi], t=0..10):
```

おなじ動作を，display関数でinsequence=trueとしても可能です．この場合は第一引数で入ったリスト([])に一連の画像を用意し，コマ送りで表示させます．

```
> tmp:=[];
> n:=10;
> for i from 0 to n do
> t:=i;
> tmp:=[op(tmp), plot(sin(x-t),x=0..5*Pi)];
> end do:
> display(tmp,insequence=true);
```

$$tmp := []$$

$$n := 10$$


0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
2
4
6
8
10
12
14
x

1.7 データの入出力

作った animation を gif ファイルとして保存したり，測定データなどを読み込んで表示する手軽な方法があると便利です．そのためにはファイルとのやりとりをする必要があります．いくつかの手法をまとめておきます．

1.7.1 ファイル名の取得

ファイル名の取得は，Java の標準関数を使った Maplet パッケージの GetFile 関数を使うと手軽にできます．

```
> with(Maplets[Examples]):
> file1:=GetFile();
```

```
Warning, the protected name LinearAlgebra has been redefined and
unprotected

Initializing Java runtime environment.
```

$$file1 := \text{“/Users/bob/Desktop/data1.txt”}$$

GetFile 関数を呼びだして開いたファイル選択ウィンドウでファイルを指定するとファイルのパスが file1 に入ります． Windows では

```
> with(StringTools):
> file2:=SubstituteAll(file1,"\\","/");
```

で”\”を”/”に変換する必要があります．日本語のファイル名は文字化けして使えません．ファイル名の変更は手でやるか，あるいは Substitute 関数を使います．

```
> with(StringTools):
> file2:=Substitute(file1,"data1","data2");
```

```
Warning, the assigned name Group now has a global binding
```

$$file2 := \text{“/Users/bob/Desktop/data2.txt”}$$

1.7.2 簡単なデータのやりとり

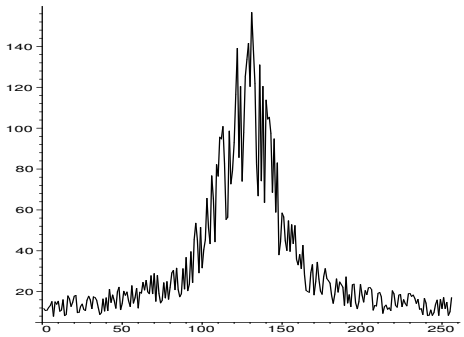
ファイルとの単純なデータのやりとりは writedata,readdata 関数が便利です．例えば，以下のようなデータを作ったとします．これをファイルへ書きだすには

```
> f1:=t->subs({a=10,b=40000,c=380,d=128},a+b/(c+(t-d)^2) ):
> T:=[seq(f1(i)*(0.6+0.8*evalf(rand()/10^12)),i=1..256)]:
> writedata(file1,T);
```

として T に格納されているデータを file1 へ出力します．同じようにして読み込んで表示させてみます．

```
> T:=readdata(file1,1):
> with(plots):
> listplot(T);
```

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



1.7.3　少し高度なデータのやりとり

writeto 関数で出力を外部ファイルへ切り替えることも可能．

```
> interface(quiet=true);
> writeto(file2);
> for i from 1 to 10 do
> s1:=data||i;
> printf("%10.5f %s\n",evalf(f1(i)),s1);
> end do:
> writeto(terminal):
> interface(quiet=false);
```

$$false$$

$$true$$

C 言語の標準的なデータ読み込みに似せた動きも出来ます．fopen，readline，sscanf，fclose を使ったデータの入力です．

```
> fd:=fopen(file2,READ);
> for i from 1 to 2 do
> l1:=readline(fd);
> d:=sscanf(l1,"%f %s");
> end do;
> fclose(fd):
```

$$fd := 1$$

$$l1 := \text{“ 12.42292 data1”}$$
$$d := [12.42292, \text{“data1”}]$$
$$l1 := \text{“ 12.46063 data2”}$$
$$d := [12.46063, \text{“data2”}]$$

fd にファイル識別子 (file descripter) を持っていき，readline で 1 行ずつ読ませます．これを sscanf で format にしたがって l1 に格納していきます．l1 には自動的に適切な形式で変数を入れてくれています．

```
> d[1];
> whattype(d[1]);
> d[2];
> whattype(d[2]);
```

$$12.46063$$
$$\textit{float}$$
$$\text{“data2”}$$
$$\textit{string}$$

前述の animation などの gif 形式の plot を外部ファイルへ出力して表示させるには，以下の一連のコマンドのようにします．

```
> plotsetup(gif,plotoutput=file2):
> display(tmp,insequence=true);
> plotsetup(default):
```

こいつを quicktime などに食わせれば Maple 以外のソフトで動画表示が可能となります．3 次元図形の標準規格である vrml も同じようにして作成することが可能です (?vrml; 参照)．

1.7.4 linux でのフィルターとしての利用法

linux 版では文字ベースの maple を使って，filter として高度な作業をさせることが出来ます．スクリプトの中に外部ファイルとの入出力を組み込めば，いままで紹介してきた複雑な動作をブラックボックスの内部処理としてそのまま使えます．例えば，

```
[bob@asura0 ~/test]$ cat test.txt
T:=readdata("./data101");
interface(quiet=true);
writeto("./result");
print(T[1]);
writeto(terminal);
interface(quiet=false);
```

とすれば，data101 から読み込んだデータに何らかの処理を施した結果を result に打ち出すことが可能．interface(quiet=true) で余計な出力を抑止しています．これを maple に食わせると

```
[bob@asura0 ~/test]$ /usr/local/maple9.5/bin/maple < test.txt
    |\^/|     Maple 9.5 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) Maplesoft, a division of Waterloo Maple Inc. 2004
 \  MAPLE  /  All rights reserved. Maple is a trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> T:=readdata("./data101");
                        T := [1.23, 2.35]
> interface(quiet=true);
                           false
                            true
> quit
bytes used=211000, alloc=262096, time=0.00
```

めでたく出力されているはず．

```
[bob@asura0 ~/test]$ cat result
                            1.23
```