

数式処理ソフト Maple で理解する数値計算の 基礎

西谷滋人

関西学院大学工学部情報科学科

Copyright 2005 Shigeto R. Nishitani

Department of Informatics,

Kwansei Gakuin University, Sanda, Japan

e-mail : nishitani@ksc.kwansei.ac.jp

平成 17 年 4 月 13 日

本書は PowerBookG4 上で Maple 9 と $L^A T_E X$ 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 はマサチューセッツ工科大学の登録商標です。

Microsoft Windows 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。

数式処理ソフト Maple で理解する数値計算の基礎

Copyright ©2005, 西谷滋人. この出版物はオープン・パブリケーション・ライセンス v1.0 またはそれ以降の版により定められた使用条件に基づいてのみ頒布できます (最新の版は <http://www.opencontent.org/openpub/> で入手できます).

この製品またはそれから派生した作品を，著作権所有者の前もっての許可なしに，あらゆる標準的な (紙媒体) 書籍として頒布することは禁じられています。

Maple : Essentials and Applications on Numerical Recipe

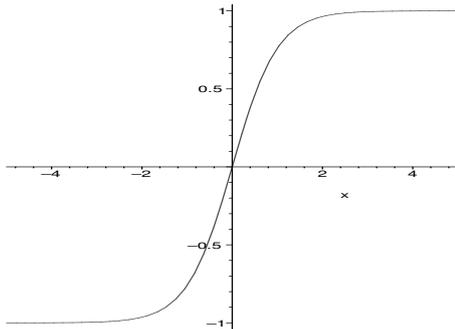
Copyright ©2005 by Shigeto R. Nishitani. This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.

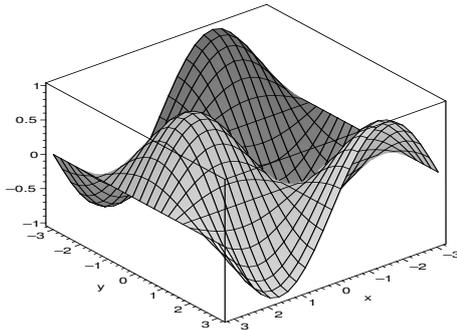
目次

第1章 基本操作	1
1.1 最初の一步	1
1.1.1 Mapleの起動法	1
1.1.2 コマンド入力	1
1.1.3 間違い修正	3
1.1.4 ヘルプファイル	3
1.2 簡単なコマンド	5
1.2.1 変数の代入とキャンセル	5
1.2.2 関数	6
1.2.3 ユーザー関数の定義	7
1.2.4 プロット	7
1.2.5 方程式の解	9
1.2.6 微分	10
1.2.7 積分	10
1.3 Mapleでのプログラミング	12
1.3.1 値の代入, 出力	12
1.3.2 算術演算	12
1.3.3 プログラムの流れの制御	14
1.3.4 配列	16
1.3.5 proc化	19
1.4 Mapleの線形代数	24
1.4.1 ベクトル, 行列の生成	24
1.4.2 ベクトル, 行列の演算	25
1.4.3 逆行列, 行列式, 転置	26
1.4.4 固有値	27
1.5 Mapleの数式処理	30
1.5.1 コマンド解説	30
1.5.2 鉄則とその具体例	34
1.5.3 実戦例	37

1.6	Mapleによるグラフ作成	45
1.6.1	簡単なグラフ化	45
1.6.2	Mapleの描画関数の覚書	49
1.6.3	アニメーション	49
1.7	データの入出力	51
1.7.1	ファイル名の取得	51
1.7.2	簡単なデータのやりとり	51
1.7.3	少し高度なデータのやりとり	52
1.7.4	linuxでのフィルターとしての利用法	53
第2章	情報科学の諸問題への応用	55
2.1	Mandelbrot 集合の描画	55
2.2	FFT(Fast Fourier Transfer)	57
2.3	非線形最小二乗法	60
2.4	巡回セールスマン問題	65
2.5	ソート	69
2.6	RSA 暗号	74
2.7	ハノイの塔	83



```
> 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 を入れた順番であり，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

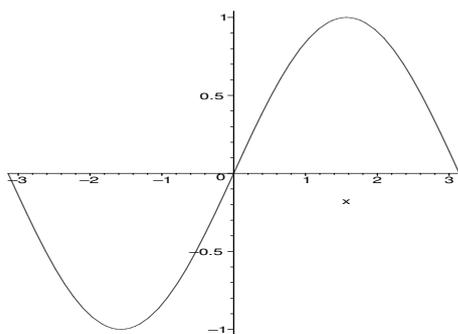
打ち間違いなどの訂正は普通のエディターソフトと同様に, アローキーあるいはマウスのクリックによってできます. 訂正して enter を入れれば入力されます. ある領域を選択して, 削除・カットおよびペーストなどの作業もマウスを使ってできます. 例えばサイン関数を $-\pi$ から $+\pi$ までプロットしようとした時

```
> 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, $\text{pi} \rightarrow \text{Pi}$ 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:=%;
                                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ちら辺で `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出すには `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)));
```

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

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);
```

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

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);
```

$$f3 := T \rightarrow eq2$$

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

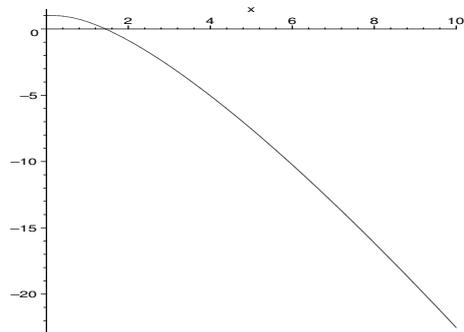
となり変数を変数と認識していないことがわかります。

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

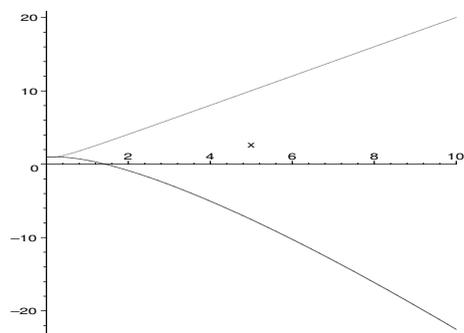
しょう。

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



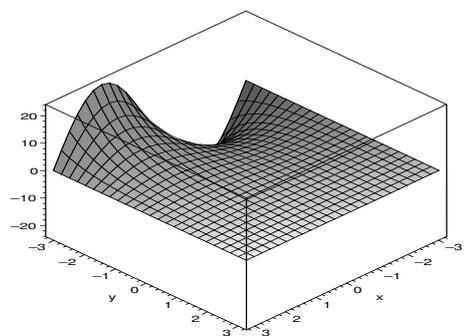
となります。2つの関数を一つの領域にプロットするには

```
> 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を用意されています. 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});
```

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

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

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});
```

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

```
> 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);
```

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

$$1.442695041$$

1.2.6 微分

微分は diff によって行います。

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

$$2x$$

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);
```

$$2y^2$$

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

$$c := (x, t) \rightarrow X(x) T(t)$$

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

$$\left(\frac{d}{dx} X(x)\right) T(t)$$

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

$$\left(\frac{d}{dx} X(x)\right) \left(\frac{d}{dt} T(t)\right)$$

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);
```

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

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

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

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

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 ($C(T) = dE/dT$) 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;
                                     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
```

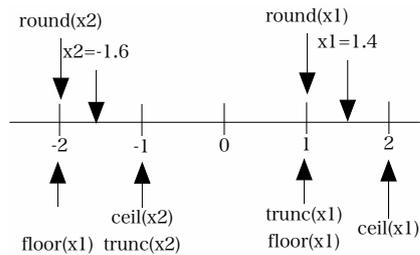


図 1.1: 数直線上での trunc, round, floor, ceil の関係.

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出すことができます。これを利用して、リストの頭やおしり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 := 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 の要素数を取りだすには、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う便利です。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, それぞれの桁の数值を `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.

それぞれの桁の数值を取りだす部分は, 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、

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad (1.2)$$

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 $x_0, x_1, f(x)$ 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)。

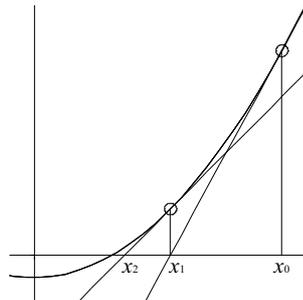


図 1.2: Newton-Raphson 法の模式図.

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

まず

```
> 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かの方法が用意されています。

まずは標準的な `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;

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

> 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);

$$\begin{bmatrix} x^2 & xy & xz \\ xy & y^2 & yz \\ xz & yz & z^2 \end{bmatrix}$$

1.4.3 逆行列, 行列式, 転置

逆行列は MatrixInverse で一発で求められます。

> 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);
```

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

浮動小数点に直します。1(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);
      [ 11.0584219844797715
        13.5876329772398865
        16.1168439700000015 ]
> A1.Column(V,2);
      [ 11.0584219847999989
        13.5876329772000002
        16.1168439695999979 ]
```

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: 式の操作に関するコマンド

式の変形	式の分割抽出	代入, 置換, 仮定
<code>simplify</code> : 簡単化	<code>lhs, rhs</code> : 左辺, 右辺	<code>subs</code> : 一時的代入
<code>expand</code> : 展開	<code>numer, denom</code> : 分子, 分母	<code>assume</code> : 仮定
<code>factor</code> : 因数分解	<code>coeff</code> : 係数	<code>assuming</code> : 一時的仮定
<code>normal</code> : 約分・通分	<code>nops, op</code>	<code>assign</code> : 値の確定
<code>combine</code> : 公式でまとめる		<code>about</code> : 仮定の中身
<code>collect</code> : 次数でまとめる		<code>anames('user')</code> : 使用変数名
<code>sort</code> : 昇べき, 降べき		<code>restart, a:= 'a'</code> : 初期化
<code>convert</code> : 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);
      -1
      2 I (e^(xI) - 1/e^(xI))
> convert(sinh(x),exp);
      1
      2 e^x - 1/2 e^-x
> convert(exp(I*x),trig);
      cos(x) + sin(x) I
> convert(1/(x-1)/(x+3),parfrac);
      1
      4(x+3) + 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;
      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);
      3 2 2
      x  - 3 x  + 4 x + 1
> sort(exp1,[x],ascending);
      3 2 3
      1 + 4 x - 3 x  + x
> exp2:=x^3-3*x*y+4*x^2+y^2:
> sort(exp2);
      3 2 2 2
      x  + 4 x  - 3 x y + y
> sort(exp2,[x]);
      3 2 2 2
      x  + 4 x  - 3 y x + y
> sort(exp2,[y],descending);
      2 3 2 2
      y  - 3 x y + x  + 4 x
```

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);
      2 2
      exp1 := x  - 4 x + 4
              (a + 2)  - 4 a - 4
```

assume: 仮定 `assume(関係式)`

assuming: 一時的仮定 `exp1 assuming 関係式`

```
> sqrt(exp1);
      2
      sqrt(-2 + x)
```

²ascending, 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 + 1/2 x^2 + 1/6 x^3 + 1/24 x^4 + 1/120 x^5 + O(x^6)
> series(sin(x),x=Pi/3,2);
      sqrt(3)/2 + 1/2 (x - pi/3) + O((x - pi/3)^2)
> convert(%,polynom);
      sqrt(3)/2 + x/2 - 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。

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

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);
```

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

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

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

と求まるようになっていきます。ここでは βc が正の場合 ($\text{csgn}(\beta c)=1$) 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);
```

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

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);
```

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

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;
```

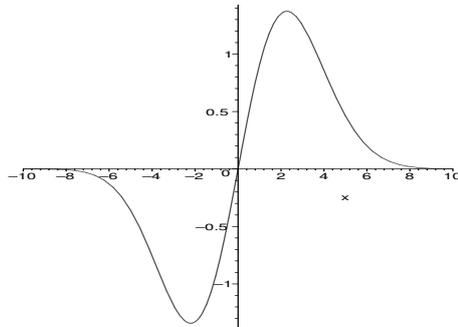
```
c := 1
```

```
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);

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

次に $x=-\alpha \dots \alpha$ の定積分を実行してみます.

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

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

さらに $\alpha \rightarrow \infty$ としてみます

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

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

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わかります. 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 - 12x^3 + 54x^2 - 108x + 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

$$U(x) = cx^2 - gx^3 \quad (1.6)$$

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められます。平均の位置 x は、熱力学的な確率で重みづけられ、

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

で計算できます。ここで $\beta \equiv 1/(k_B T)$ です。この積分を実行すると

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

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));
```

$$U := cx^2 - gx^3$$

$$eU := \frac{e^{\beta gx^3}}{e^{\beta cx^2}}$$

```
> ex:=convert(series( numer(eU), x, 4), polynom);
ex := 1 + beta g x^3
```

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

$$f1 := \frac{1 + \beta g x^3}{e^{\beta cx^2}}$$

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);
```

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

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

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

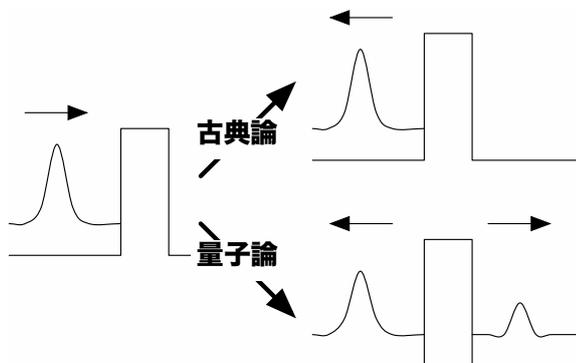
beta c > 0 の場合とそれ以外の結果を別々に返しています。題意から明らかのように beta c > 0 ですので、それを仮定 (assume) して、分子÷分母を実行します。

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

> num/den;

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

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

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

ですから、波動関数は

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

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 $\varepsilon \leq V_0$ によって事情が変わってきます。 $\varepsilon \geq V_0$ では $\kappa = \sqrt{2m(\varepsilon - V_0)/\hbar^2}$ と定義すると、波動関数は

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

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 - 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ぬという条件を $x = 0$ と $x = a$ に適用すると、条件はそれぞれ

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

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

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

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

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

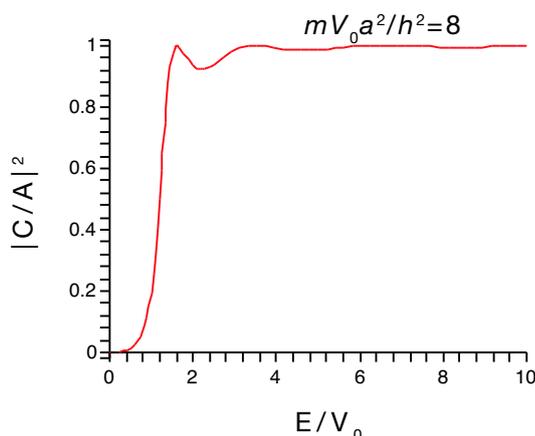
です。同様な計算によって

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

となります。 $mV_0 a^2/\hbar^2 = 8$ 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。まずは波動関数の定義です。

```
> restart;
> psi1:=x->A*exp(I*k*x)+B*exp(-I*k*x);
      psi1 := x -> A e^(k x I) + B e^(-I k x)
> psi2:=x->E*exp(I*kappa*x)+F*exp(-I*kappa*x);
      psi2 := x -> E e^(kappa x I) + F e^(-I kappa x)
```



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

$$\psi_3 := x \rightarrow C e^{(kxI)}$$

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});
```

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

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

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

```
> 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 κ , 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 , κ , 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;
```

$$NN := 8$$

$$a2 := a2$$

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

$$\kappa2 := \frac{2 m (\epsilon - V0)}{h2}$$

$$k2 := \frac{2 m \epsilon}{h2}$$

```
> 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);

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

> 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,

$$\cosh^2(\alpha * a) = 1 + \sinh^2(\alpha * a) \quad (1.16)$$

であることに注意。

1.6 Mapleによるグラフ作成

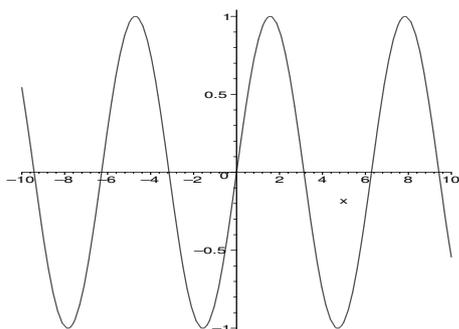
関数の視覚化などの便利な道具が多数用意されています。基本となるグラフ作成に絞っていくつかの手法を紹介します。

1.6.1 簡単なグラフ化

plot

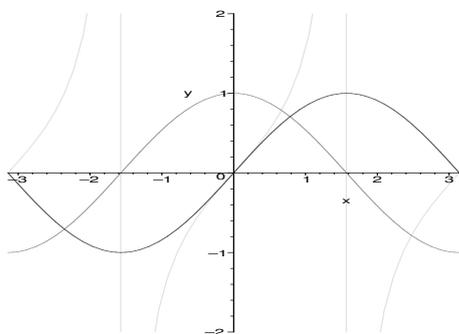
もっとも簡単に関数を表示するには、表示したい関数とその変数を指定するだけで、

```
> 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。

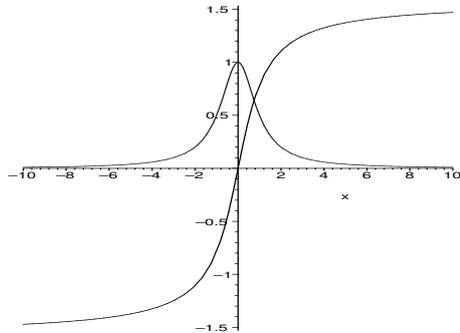
```
> plot({sin(x),cos(x),tan(x)},x=-Pi..Pi,y=-2..2);
```



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);
```

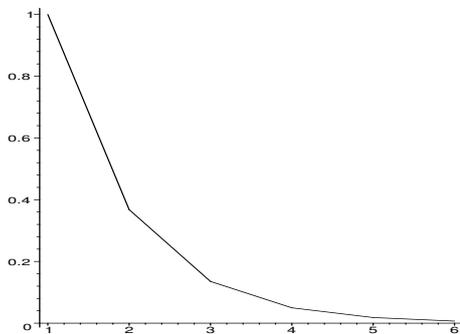


ここでは、`p1` と `p2` にそれぞれの関数のプロットを代入して、`display` 関数で一度に表示しています。

listplot

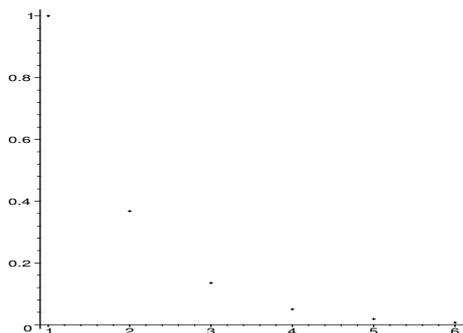
離散的なデータは `listplot` で表示してくれます。

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

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


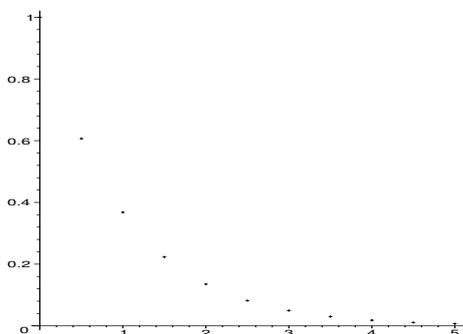
`listplot` は受け取った `list` の要素を y_i に、1 から始まる添字を x_i にして、デフォルトでは線でグラフを書きます。

```
> 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 $1, 2, 3, \dots$ 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、

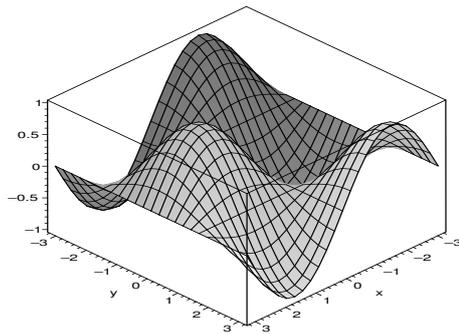
```
> pointplot(T,connect=true);
```

とします。

plot3d

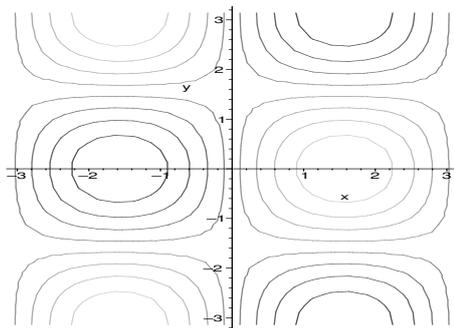
2次元の定義域を持つスカラー関数、つまり3次元の関数は `plot3d` 関数で表示されます。

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



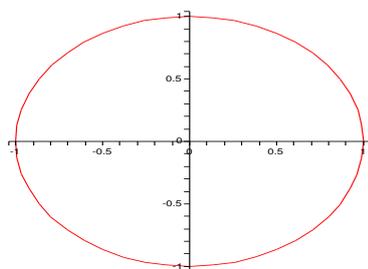
等高線図なども書けます.

```
> 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パッケージを呼び出す必要があります. `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 `parametric 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。

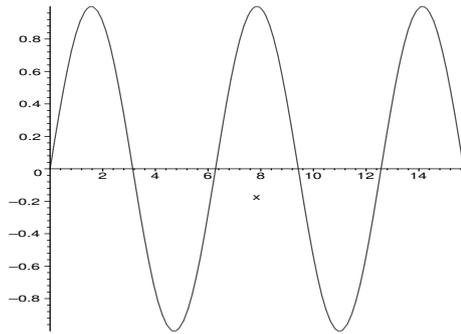
```
> with(plottools):with(plots):
> animate(plot, [sin(x-t),x=0..5*Pi], t=0..10):
```

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
```



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 := "/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 := "/Users/bob/Desktop/data2.txt"
```

1.7.2 簡単なデータのやりとり

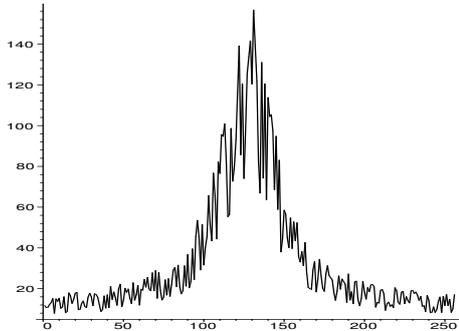
ファイルとの単純なデータのやりとりは writedata, readdata 関数が便利です．例えば，以下のようなデータを作ったとします．これをファイルへ書き出すには

```
> 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 := " 12.42292 data1"
d := [12.42292, "data1"]
l1 := " 12.46063 data2"
d := [12.46063, "data2"]
```

fd にファイル識別子 (file descriptor) を持っていき、`readline` で 1 行ずつ読み取ります。これを `sscanf` で `format` にしたがって `l1` に格納していき、`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
float
"data2"
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できます。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.

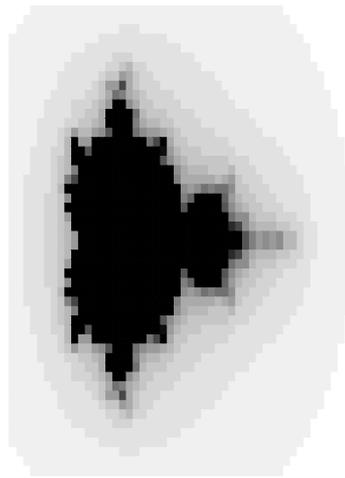
```
[bob@asura0 ~/test]$ cat result
                1.23
```

第2章 情報科学の諸問題への応用

2.1 Mandelbrot 集合の描画

課題

以下に示す Mandelbrot 集合 を描け.



課題の背景

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.

Maple の関連コマンド

これを Maple で実現しようと思うと, `Mandelbrot(x,y)` という関数 (procedure) を作って, `count` の初期値を例えば 20 として以下のようにして `densityplot` 関数に食わせます.

```
> with(plots):  
> densityplot('Mandelbrot(x,y)',x=-1..2.5,y=-1.5..1.5,  
> axes=none,style=patchnogrid,grid=[50,50]);
```

ここで, `densityplot` の第一引数の関数 `Mandelbrot` がシングルクォートで囲まれていないと描画の時にエラーがでます¹.

解法のヒントと発展問題

1. Mandelbrot 関数を作れ. `count` の初期値を 20 とした場合は, $[x,y]$ が $[0,0]$, $[-0.5,0]$, $[0,1.5]$ での値は, それぞれ 0, 14, 17 となることを確認せよ.
2. `plot('Mandelbrot(x,0)',x=-1..2.5);` を試せ.
3. `densityplot` 関数で表示せよ. さらにきれいになるように `densityplot` の option を, `grid=[200,200]`, `colorstyle=HUE` としてみよ.

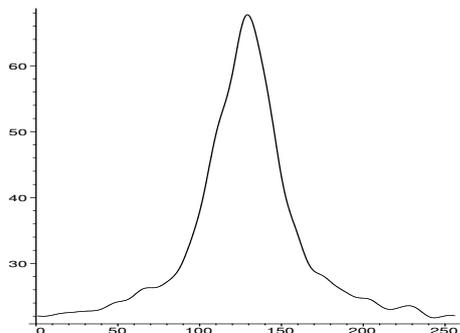
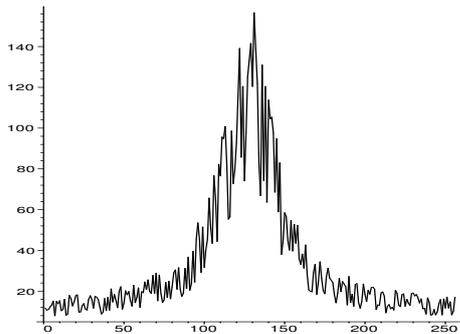
¹これは値の評価によるエラーです. 詳しくはラーニングガイドの「評価と簡単化」を参照ください.

2.2 FFT(Fast Fourier Transfer)

課題

下図の上段パネルのようなデータをフーリエ変換し，三角フィルタによってノイズを除去せよ．データは次のようにして作成・表示しました．

```
> 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)]:
> with(plots):listplot(T);
```



課題の背景

フーリエ変換を用いたデータのフィルタリングについてです²．時系列データ $x(t)$ の解析にはそのフーリエ変換

$$X(f) = \int_{-\infty}^{\infty} x(t) \exp(2\pi i f t) dt \quad (2.1)$$

が多用されます．これは，高速フーリエ変換 (Fast Fourier Transformation:略して FFT) アルゴリズムの発明によって計算機での処理が容易となったためです．時間

²参考 河合潤：『化学計測学』 合志陽一編 昭晃堂 1997.

経過と共に変化する時系列測定データ $x(t)$ に含まれるさまざまな周波数成分について、何 Hz の周波数成分がどの程度の割合でその時系列データの中に含まれているのかを明らかにできるからです。時系列データの中のノイズは δ 関数に似ており、そのフーリエ変換は全周波数成分を一様に含んでいます。一方、測定したいデータは一般に、時間に対してゆっくり変化する量です。測定によって得られたデータの時間軸は連続ではなくとびとびの値をとるので、離散フーリエ変換を取ることになります。この例では時間データは 256 チャンネルのデータです。高周波成分 (=ノイズ成分) を取り除くために、データに対して窓関数 (低域通過フィルタ) をかけてフーリエ逆変換

$$x(t) = \int_{-\infty}^{\infty} X(f) \exp(2\pi i f t) dt \quad (2.2)$$

することによってフィルタを掛けた後のスムーズなカーブが得られます。低域通過フィルタには、方形窓、修正方形窓、フォンハン窓 (2 乗余弦窓)、ハミング窓などさまざまなものがあります。

Maple の関連コマンド

実際の FFT を少ないデータで見てみます。まずデータを用意します。

```
> Rdata:=array([1,2,3,4,5,4,3,2]);
> Idata:=array(1..8,sparse);
      Rdata := [1, 2, 3, 4, 5, 4, 3, 2]
      Idata := array(sparse, 1..8, [])
```

これに FFT をかけます。

```
> FFT(3,Rdata,Idata);
```

8

これで終わりです。ここで FFT の第一引数の 3 はデータの個数が 2^3 であることを示しています。フーリエ変換された後のデータ構造を見ておきましょう。

```
> print(Rdata):
> print(Idata):
[24, -6.828427123, 0., -1.171572877, 0, -1.171572877, 0., -6.828427123]
      [0, 0.1 10-8, 0., 0.1 10-8, 0, -0.1 10-8, 0., -0.1 10-8]
```

逆変換とその結果です。

```
> iFFT(3,Rdata,Idata);
```

8

```
> print(Rdata);
> print(Idata);
```

```
[1.000000000, 2.000000001, 3.000000000, 3.999999999, 5.000000000, 3.999999999,
3.000000000, 2.000000001]
```

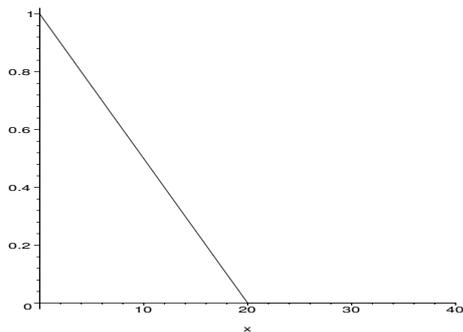
```
[0., -0.2500000000 10-9, 0., -0.2500000000 10-9, 0., 0.2500000000 10-9, 0.,
0.2500000000 10-9]
```

三角フィルター関数は以下のようにして作られます。

```
> filter:=x->piecewise(x>=0 and x<=20, (1-x/20));
```

$$filter := x \rightarrow \text{piecewise}(0 \leq x \text{ and } x \leq 20, 1 - \frac{1}{20}x)$$

```
> plot(filter(x),x=0..40);
```



解法のヒント

1. 256 個のデータを Rdata に読み込め.
2. FFT をかけよ.
3. FFT をかけた後のデータの強度 ($Rdata[i]^2 + Idata[i]^2$) を logplot せよ.
4. FFT をかけた後のデータに filter 関数を掛けよ.
5. フーリエ逆変換してその結果を表示せよ.

2.3 非線形最小二乗法

課題

以下のスクリプトで与えられたデータ

```
> 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)]:
> with(plots):listplot(T);
```

を,

$$f(x) = a + \frac{b}{c + (x - d)^2} \quad (2.3)$$

なるローレンツ型関数にカーブフィッティングするプログラムを作成し、そのパラメータを決定せよ。ここで a はバックグラウンドの強度、 b はローレンツ関数の強度、 c は線幅、 d はピーク位置を表す。

課題の背景

(2.3) 式の特徴は、パラメータが線形関係にないということです。非線形の最小二乗法を用いてデータフィットをしてくれる関数は default では Maple には用意されていないようです³。そこで、実際に非線形最小二乗法のプログラムを作成し、実際のデータへのフィッティングを試みてみましょう。非線形最小二乗法の注意事項は補足に記しました。

今、パラメータの初期値を $(a_0 + \Delta a_1, b_0 + \Delta b_1, c_0 + \Delta c_1, d_0 + \Delta d_1)$ としましょう。このとき関数 f を真値 (a_0, b_0, c_0, d_0) のまわりでテイラー展開し高次項を無視すると

$$\begin{aligned} \Delta f &= f(a_0 + \Delta a_1, b_0 + \Delta b_1, c_0 + \Delta c_1, d_0 + \Delta d_1) & (2.4) \\ &\quad - f(a_0, b_0, c_0, d_0) \\ &= \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial a} \Delta a_1 + \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial b} \Delta b_1 \\ &\quad + \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial c} \Delta c_1 + \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial d} \Delta d_1 \end{aligned}$$

となります。(2.3) 式の偏微分は計算可能です。'DATA101' のデータは $t = 1$ から 256 までの時刻に対応したデータ点 $f_1 \sim f_{256}$ とします。各測定値とモデル関数か

³公式のサポートではありませんが、世界中の科学者、技術者が Maple で開発した library を公開している The Maple Application Center (<http://www.mapleapps.com/>) の Data Analysis のカテゴリーに、後述する Levenberg-Marguqrdt 法を用いた Data fitting の汎用プログラム Generalized Weighted Non-Linear Regression Using the Levenberg-Marquardt Method by David Holmgren (http://www.mapleapps.com/categories/data_analysis_stats/data/html/genfit_6.html) があります。

ら予想される値との差 $\Delta f_1 \sim \Delta f_{256}$ は,

$$\begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} = J \begin{pmatrix} \Delta a_1 \\ \Delta b_1 \\ \Delta c_1 \\ \Delta d_1 \end{pmatrix} \quad (2.5)$$

となります。ここで 4 行 256 列の行列

$$J = \begin{pmatrix} \left(\frac{\partial f}{\partial a}\right)_1 & \left(\frac{\partial f}{\partial b}\right)_1 & \left(\frac{\partial f}{\partial c}\right)_1 & \left(\frac{\partial f}{\partial d}\right)_1 \\ \vdots & \vdots & \vdots & \vdots \\ \left(\frac{\partial f}{\partial a}\right)_{256} & \left(\frac{\partial f}{\partial b}\right)_{256} & \left(\frac{\partial f}{\partial c}\right)_{256} & \left(\frac{\partial f}{\partial d}\right)_{256} \end{pmatrix} \quad (2.6)$$

はヤコビ行列と呼ばれる行列です。逆行列 J^{-1} は転置行列 J^t を用いて

$$J^{-1} = (J^t J)^{-1} J^t \quad (2.7)$$

と表されます。従って真値からのずれは

$$\begin{pmatrix} \Delta a_2 \\ \Delta b_2 \\ \Delta c_2 \\ \Delta d_2 \end{pmatrix} = (J^t J)^{-1} J^t \begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} \quad (2.8)$$

として計算できます。理想的には $(\Delta a_2, \Delta b_2, \Delta c_2, \Delta d_2)$ は $(\Delta a, \Delta b, \Delta c, \Delta d)$ に一致するはずですが、測定誤差と高次項のために一致しません。初期値に比べ、より真値に近づくだけです。そこで、新たに得られたパラメータの組を新たな初期値に用いて、より良いパラメータに近付けていくという操作を繰り返します。新たに得られたパラメータと前のパラメータとの差がある誤差以下になったところで計算を打ち切り、フィッティングの終了となります。

Maple の関連コマンド

実際にパラメータフィットを行っていきましょう。線形代数計算のためにサブパッケージとして LinearAlgebra を呼びだしておきます。

```
> with(LinearAlgebra):
データを読み込みます。
> datapoint:= [seq([i, T[i]], i=1..256)]:
```

(2.3) 式のローレンツ型の関数を仮定しておきましょう.

```
> f:=a+b/(c+(x-d)^2);
```

$$f := a + \frac{b}{c + (x - d)^2}$$

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

$$f1 := x \rightarrow a + \frac{b}{c + (x - d)^2}$$

(2.3) 式の関数の微分を求め、新たな関数として定義します.

```
> dfda:=unapply(diff(f,a),x);
```

```
> dfdb:=unapply(diff(f,b),x);
```

```
> dfdc:=unapply(diff(f,c),x);
```

```
> dfdd:=unapply(diff(f,d),x);
```

$$dfda := x \rightarrow 1$$

$$dfdb := x \rightarrow \frac{1}{c + (x - d)^2}$$

$$dfdc := x \rightarrow -\frac{b}{(c + (x - d)^2)^2}$$

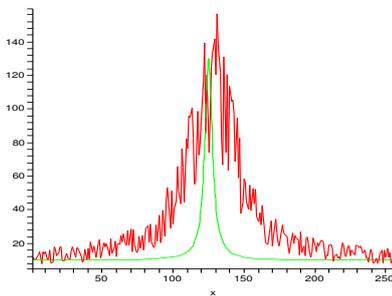
$$dfdd := x \rightarrow -\frac{b(-2x + 2d)}{(c + (x - d)^2)^2}$$

初期値を仮定して、表示します.

```
> guess1:={a=10,b=1200,c=10,d=125};
```

```
guess1 := {a = 10, b = 1200, d = 125, c = 10}
```

```
> plot([datapoint,subs(guess1,f1(x))],x=1..256);
```



解法のヒント

1. (2.5) 式の左辺の Δf_i を求めよ. $T[i]-f1(i)$ を 1..imax まで求め, Vector に入れる.
2. (2.6) 式に従ってヤコビ行列を求めよ.
3. (2.7) 式の公式によるヤコビ行列の逆行列を求めよ. また, (2.8) 式により, 新たなパラメータの組を求めよ.
4. 求めたパラメータを用いたモデル関数と, データをプロットしてみよ. 前回より近づいているのがわかるでしょう.
5. 上の操作を適当に繰り返し, パラメータを収束させよ. その値とプロットを示せ.

補足:最小二乗法に関するメモ

前述のフィッティング法の説明では, テイラー展開を用いた説明であり, まるで最小二乗法を用いてないような印象を与えたかもしれません. しかしこれは, 最小二乗法の χ^2 関数に Newton-Raphson 法を適用し, 二次微分を無視した方法と考えることも可能です. この様子を以下に記しておきます.

当てはめたいモデルは x がデータの横軸, \mathbf{a} がパラメータの組とすると,

$$y = f(x; \mathbf{a}) \quad (2.9)$$

です. 最小二乗法の χ^2 関数は

$$\chi^2(\mathbf{a}) = S(\mathbf{a}) = \sum_{i=1}^N [y_i - f(x_i; \mathbf{a})]^2 \quad (2.10)$$

です. 後の式を単純にするために $y_i - f(x_i; \mathbf{a}) \Rightarrow f(x_i; \mathbf{a})$ と置き換え, あらかじめ実験値を引いておきます. S が \mathbf{a} で極小値を持つ (安定である) ための条件は

$$\frac{\partial S(\mathbf{a})}{\partial a_k} = -2 \sum_{i=1}^N f(x_i; \mathbf{a}) \frac{\partial f(x_i; \mathbf{a})}{\partial a_k} = 0. \quad (2.11)$$

ヤコビ行列を $J_i(\mathbf{a}) = \left[\frac{\partial f_i}{\partial a_j} \right]$ と表すと

$$\nabla S(\mathbf{a}) = -2J^t(\mathbf{a})f(y; \mathbf{a}) = 0 \quad (2.12)$$

と書き換えることができます。これに Newton-Raphson 法を適用すると

$$\nabla^2 S(\mathbf{a}) \Delta \mathbf{a}^{(k)} = -\nabla S(\mathbf{a}^{(k)}) \quad (2.13)$$

となります。 $S(\mathbf{a})$ のあらわな 2 次微分の表現は

$$\frac{\partial^2 S(\mathbf{a})}{\partial a_k \partial a_{k'}} = -2 \left\{ \sum_{i=1}^N \frac{\partial f(x_i; \mathbf{a})}{\partial a_k} \frac{\partial f(x_i; \mathbf{a})}{\partial a_{k'}} + \sum_{i=1}^N \frac{\partial^2 f(x_i; \mathbf{a})}{\partial a_k \partial a_{k'}} f(x_i; \mathbf{a}) \right\} \quad (2.14)$$

です。ここで、 k 回目の推定パラメータ $\mathbf{a}^{(k)}$ に対する修正値を $\Delta \mathbf{a}^{(k)}$ とすると、

$$\left\{ J^t(\mathbf{a}^{(k)}) J(\mathbf{a}^{(k)}) + \sum_{i=1}^N f(x_i; \mathbf{a}^{(k)}) \nabla^2 f(x_i; \mathbf{a}^{(k)}) \right\} \Delta \mathbf{a}^{(k)} = J^t(\mathbf{a}^{(k)}) f(x_i; \mathbf{a}^{(k)}) \quad (2.15)$$

が得られます。鉤括弧内の第 2 項を無視すると、先程の (2.5) 式の線形的な関係が現れます。この無視する項は、線形の場合には確かに現れないし、1 次の項に比べて小さいと考えられます。さらに $f(x_i; \mathbf{a})$ を掛けて和を取っているため、それらがお互いにキャンセルしてくれる可能性があります。

この Gauss-Newton 法と呼ばれる非線形最小二乗法は線形問題から拡張した方法として論理的に簡明であり、広く使われています。しかし、収束性は高くななく、むしろ発散しやすいので注意が必要です。先程の 2 次の項を無視するのでなく、うまく見積もる方法を用いたのが Levenberg-Marquardt 法です。明快な解説が Numerical Recipes in C (C 言語による数値計算のレシピ) William H. Press 他著、技術評論社 1993 にあります。

2.4 巡回セールスマン問題

課題

ランダムに生成した 16 都市を順にめぐって元に戻ってくる最短経路をアニーリング法 (simulated annealing) を用いて求めよ。 $N = 16$ の場合の (a) 初期配置, (b) 単純に総距離が下がった場合だけを採用した結果, および (c) simulated annealing の計算結果を図 2.1 に示す。

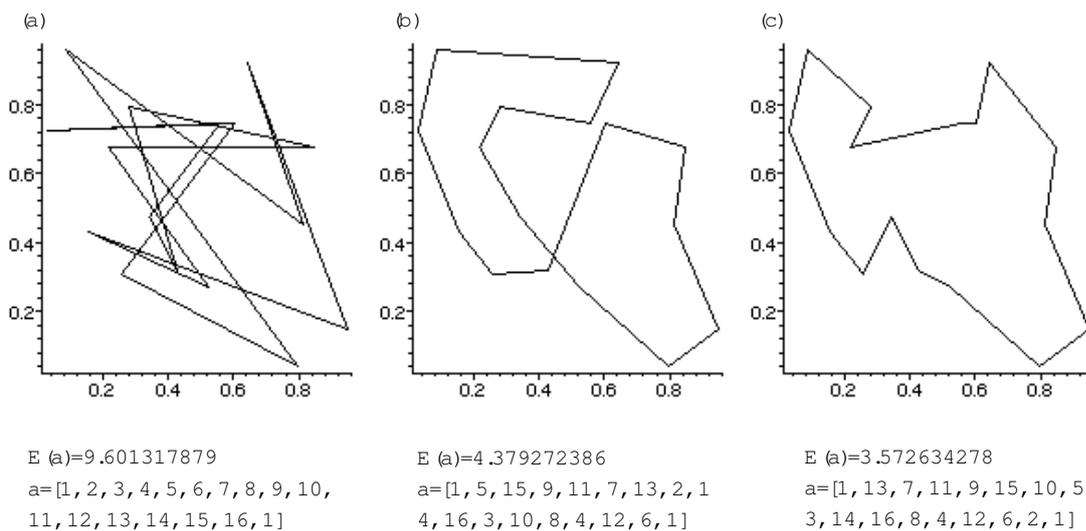


図 2.1: ランダムに生成した 16 個の街に適用した巡回セールスマン問題の (a) 初期配置, (b) 単純に総距離が下がった場合だけを採用した結果, および (c) simulated annealing の計算結果。ただし, (c) は最短経路ではない。

課題の背景

巡回セールスマン問題とは, ある街から出発していくつかの街を次々とめぐって元の街に戻ってくる最短の経路を求める問題です。訪れる街の数が少ないときにはすべての経路を数え上げればいいのですが, 数が増えるとその計算時間は指数関数的に増えてしまうと予想される問題です。このような問題はセールスマンだけでなく, コンピュータの CPU の配置や, 都市ガスの配管設計などでも出会う問題です。

対象となる経路の長さの合計を

$$E(\mathbf{a}) = \sum_{i=1..N} \|\mathbf{r}[\mathbf{a}_i] - \mathbf{r}[\mathbf{a}_{i+1}]\| \quad (2.16)$$

としておきます。ここで \mathbf{a} はそれぞれの街の順番あるいは配置を示しています。 \mathbf{r} はそれぞれ街の座標で $\|\mathbf{r}\|$ で距離を求めます。するとこの関数は模式的に図 2.2 のようになると考えることができます。すべての配置について $E(\mathbf{a})$ を求めれば、最小値が求まります。あるいは初期の配置を

$$\mathbf{a} = [1, 2, 3, \dots, N, 1] \quad (2.17)$$

として、一定の手順で変更 $\delta\mathbf{a}$ を加え、 $E(\mathbf{a})$ が下がった場合にその配置を採用するという方法をとることも出来ます。しかし、この方法は極小値に落ちてしまうことが分かるでしょう。最小値を探すには時として坂を駆け登る必要があるのです。このような問題の一つの解法としてアニーリング法 (simulated annealing) があります。これは格子欠陥を多く含んだ金属を高温へ上げて欠陥を掃き出し柔らかくする熱処理 (焼きなまし, annealing) からの類推で名付けられた手法です。

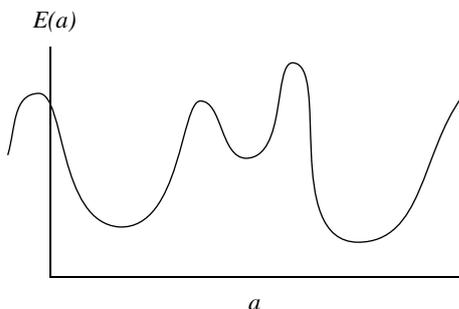


図 2.2: 配置 \mathbf{a} とその経路の総和 $E(\mathbf{a})$ を示す模式図

アルゴリズムは以下のとおりです。

1. 配置 \mathbf{a} を仮定し $E(\mathbf{a})$ を求める。
2. \mathbf{a} からすこし違った配置 $\mathbf{a} + \delta\mathbf{a}$ を作る。
3. $\Delta E = E(\mathbf{a} + \delta\mathbf{a}) - E(\mathbf{a})$ を求める。
4. $\Delta E < 0$ なら新たな配置を採用する。
5. $\Delta E > 0$ なら新たな配置を $\exp(-\Delta E/T)$ の確率で受け入れる。

6. 手順2以下を適当な回数繰り返す.

ここで T は温度から類推される制御パラメータで, 十分大きい場合はすべての状態が採用されます. 一方, T を下げるにつれて採用される試行が少なくなり, 徐々に状態が凍結されていきます. $\mathbf{a} + \delta\mathbf{a}$ の生成方法と T の下げ方をうまく取れば最小値に近い状態が確率的に高く出現し, 最小値かそれに近い状態へ落ち着くというアルゴリズムです.

Maple の関連コマンド

以下は街の位置をランダムに生成し, 経路を表示する Maple script.

```
restart;
with(plots):
N_city:=16;
Path:=[seq(i,i=1..N_city),1];
Position:=seq([evalf(rand()/10^12),evalf(rand()/10^12)],i=1..N_city):
Real_Path:=[seq(Position[Path[i]],i=1..N_city),Position[1]]:
PLOT(CURVES(Real_Path));
```

また,

```
sel_city:=rand(2..N_city):
```

によって2から N_city までの整数をランダムに生成する関数出来る.

解法のヒントと発展問題

1. 街の間の距離を計算した2次元のリストを作れ.
2. 配置を受け取って, 前問で作成したリストを参照して経路の総距離を返す関数を作れ.
3. 配置 \mathbf{a} ($=\text{Path}$) から任意に2都市を取りだして入れ替える試行を $\delta\mathbf{a}$ として上述のアルゴリズムにしたがって最短経路を探すプログラムを作成せよ.
4. 総距離を記録して, その結果を表示せよ.
5. 【発展問題】さらに総距離とそのルートを記録する関数を作って, その結果を表示せよ.

6. 【発展問題】 ルートの入れ替えを工夫して64都市でも解けるようにせよ。例えば、
- 切り出したルートを逆順にして埋めもどしたり、
 - ルートの一部を切り出して他へ移す
- 等の操作を組み込む。

2.5 ソート

課題

選択ソートとクイックソートアルゴリズムを用いて、配列データ $A := \text{array}([6, 4, 3, 1, 2, 5])$ を小さい値順に並べるソートスクリプトを作れ。

課題の背景

簡単な挿入 (insert) ソートを例に、配列に入った値がどのようにソートされていくかを Maple で表示してみます。挿入ソートのスクリプトは、

Maple Script : 挿入ソート

```
insertsort:=proc(A::array)
local n,x,i,j;
n:=op(2,op(2,eval(A)));
for i from 1 to n do
  print("Before",i,A[i],A);
  x:=A[i];
  for j from i-1 by -1 to 1 do
    if (A[j]<x) then break; end if;
    A[j+1]:=A[j];
  end do;
  A[j+1]:=x;
  print("After ",i,j+1,A);
end do:
eval(A);
end proc:
```

です。

ソートでは配列の並べ替えをおこなうためにどうしても複雑な index の動きをしますが、ゆっくり見ていけば理解できるはずです。上の関数を使ったたとえば

```
> A:=array([6,4,3,1,2,5]);
> insertsort(A);
```

とした場合の結果は、

```
      A := [6, 4, 3, 1, 2, 5]
"Before", 1, 6, [6, 4, 3, 1, 2, 5]
"After ", 1, 1, [6, 4, 3, 1, 2, 5]
```

```

"Before", 2, 4, [6, 4, 3, 1, 2, 5]
"After ", 2, 1, [4, 6, 3, 1, 2, 5]
"Before", 3, 3, [4, 6, 3, 1, 2, 5]
"After ", 3, 1, [3, 4, 6, 1, 2, 5]
"Before", 4, 1, [3, 4, 6, 1, 2, 5]
"After ", 4, 1, [1, 3, 4, 6, 2, 5]
"Before", 5, 2, [1, 3, 4, 6, 2, 5]
"After ", 5, 2, [1, 2, 3, 4, 6, 5]
"Before", 6, 5, [1, 2, 3, 4, 6, 5]
"After ", 6, 5, [1, 2, 3, 4, 5, 6]
[1, 2, 3, 4, 5, 6]

```

となります。挿入する前 (Before) と後 (After) の配列の違いを観察してください。たとえば、4 番目まで終わった、

```
"Before", 5, 2, [1, 3, 4, 6, 2, 5]
```

で内側の for-loop で何をしているかを考えます。5 番目の要素の"2"が選ばれます ($x:=A[i]$)。これを右側のもの ($A[j]$) と順番に比べていきます。大きければ空いた席 ($A[j+1]$) と入れ替えます。

Compare 出力

```

"Compare", 5, 5, [1, 3, 4, 6, _, 5]
"Compare", 5, 4, [1, 3, 4, _, 6, 5]
"Compare", 5, 3, [1, 3, _, 4, 6, 5]
"Compare", 5, 2, [1, _, 3, 4, 6, 5]

```

小さくなる ($A[j] < x$) かあるいは $j=1$ までいけば比較は終了です。5 番目の要素は 1 番目までいって、 $A[1] < x$ が成立して、 $A[2]$ に収まります。

Maple の関連コマンド

ソートにともなう値の並べ換えを視覚化することを試みます。以下のようにすると

```

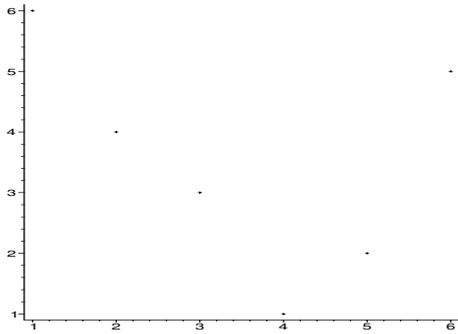
> A:=array([6,4,3,1,2,5]);
> tmp:=[seq([i,A[i]],i=1..6)];
> with(plots):
> pointplot(tmp);

```

```

A := [6, 4, 3, 1, 2, 5]
tmp := [[1, 6], [2, 4], [3, 3], [4, 1], [5, 2], [6, 5]]

```

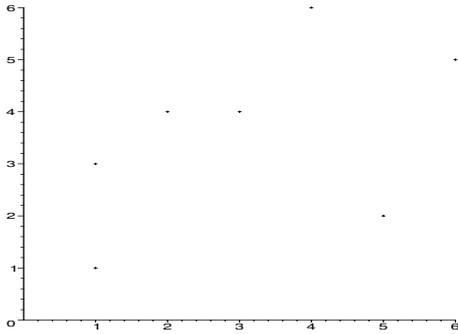


と表示されます。よこがindex, たてが入っている要素です。これを使って先程の `insertsort` の途中経過を表示させます。

`insertsort2` (ソートの視覚化を組み込んだ `insertsort`)

```
insertsort2:=proc(A::array)
local n,x,i,j;
global tmp;
n:=op(op(2,eval(A)))[2];
for i from 2 to n do
  x:=A[i];
  for j from i-1 to 1 by -1 do
    tmp:=[op(tmp),pointplot([j,x],seq([ii,A[ii]],ii=1..n))];
    if (A[j]<x) then break; end if;
    A[j+1]:=A[j];
  od;
  A[j+1]:=x;
od;
end proc;
```

```
> tmp:=[]:
> A:=array([6,4,3,1,2,5]):
> insertsort2(A):
> tmp:=[op(tmp),pointplot([seq([i,A[i]],i=1..6))]:
> display(tmp,insequence=true,view=[0..6,0..6]);
```



tmpの中に画像が蓄えられています。ここでは6番目の画像だけを表示しています。Mapleを実際に手元で動かしているときは、displayで表示させた絵をつまんで、コマ送りで一つずつどのように位置が置き変わっていくかを追ってみてください。

解法のヒントと発展問題

1. 上記の”Compare”以下が出力されるようにinsertsortを改良せよ。ただし下線”_”部には違う数字が入っている。その数字はなにか？
2. 次に示した選択ソートの考え方にもとづいてMapleスクリプト(selectsort)を書け。
 - (a) まず、全体の中で最小のものを見つけ、それを先頭のA[1]と交換する。
 - (b) 次にA[2..n]の中で最小のものを見つけ、A[2]と交換する。
 - (c) 以下同様に続ける。

さらに、視覚化を試みよ。
3. 上のinsertsortを参考にして、選んだ数より大きな数を右に、小さな数を左に置きかえる関数partition(A,first,last)を考えよ。選ぶ数としてはA[last]が簡単。これを使えば、quicksortは以下のようなスクリプトで書ける。

```
quicksort:=proc(A::array,first::integer,last::integer)
local p;
if first<last then
  p:=partition(A,first,last);
  quicksort(A,first,p-1);
end if;
end proc;
```

```
    quicksort(A,p+1,last);  
end if:  
eval(A);  
end proc:
```

4. 【発展問題】以下のランダムな整数列の作成の proc を用いて要素数を大きくしたときに, insertsort, selectsort, quicksort それぞれがどの程度の時間がかかるかを計測せよ.
-
-

```
BigRandom:=proc(max_n)  
local sel,A,i,j,k,tmp;  
sel:=rand(1..max_n):  
A:=array([seq(i,i=1..max_n)]):  
for k from 1 to 2*max_n do  
    i:=sel();j:=sel();  
    tmp:=A[i];A[i]:=A[j];A[j]:=tmp;  
end do:  
return A;  
end proc:
```

時間計測は以下のようにして出来る. (printf) 文をのぞいておくことを忘れないように.

```
> st:= time():  
> n:=160:  
> A:=BigRandom(n):  
> quicksort(A,1,n);  
> time() - st;
```

2.6 RSA 暗号

課題

以下の RSA モジュール N , 復号化指数 DD の組み合わせで送られてきた RSA 暗号文 `ciphertext` を復号化せよ. ただし, 数字列から平文への逆変換関数は後述の関数 `numtostring` を使え.

```
DD :=  
97160237787927197268399382847527128445\  
49853837391340357711330646764959436711\  
59170340750075707950776984678307988166\  
43653201556982906389265242669384110487\  
8601025;
```

```
N :=  
53889560798132910692104691551103464936\  
80295962602541240888011819541597381548\  
47328720465604349380066375753497760645\  
71193290783857018152083178164889182698\  
06475731;
```

```
ciphertext :=  
35617334193683758121458876563581777948\  
88686788640826301652725191921600228140\  
68263177796022170903531011472610378053\  
22691405568944970187110528328096932989\  
73447073
```

課題の背景

90年代の終わり頃のゴルゴ13に, ゴルゴ13が新しい暗号の開発を支援するという逸話(373話, 最終暗号)がありました. 暗号が通信の要であり, 非常に高度なしかし簡素な数学を使っているという, さいとうたかおのいつもながらの先見性に感心しました. その作り話の元にもなっていた RSA 暗号についてどのような原理なのかを Maple で見ていきます.

公開鍵暗号システムは Whitfield Diffie と Martin Hellmann によって 1976 年にそのアイデアが公表されました。そこでは、暗号化の鍵と復号化の鍵を分けることで鍵配送問題を解決しています。受信者は、前もって「暗号化の鍵」を送信者に知らせておきます。この「暗号化の鍵」は盗聴者に知られてもかまいません。送信者は、その「暗号化の鍵」を使って暗号化して受信者に送ります。復号化できるのは「復号化の鍵」を持っている人(受信者)だけです。こうすれば「復号化の鍵」を受信者に配送する必要がなくなります。対称暗号の鍵配送問題は、公開鍵暗号を使うことで解決するのです。

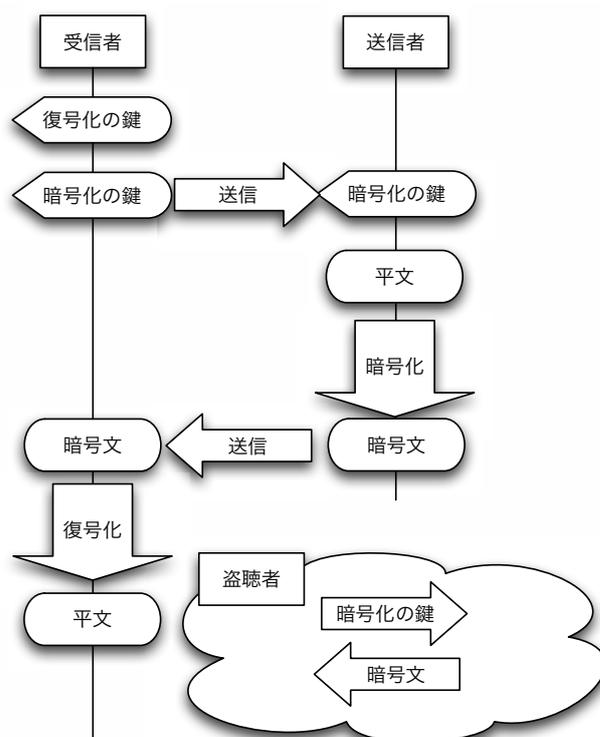


図 2.3: Diffie and Hellmann による公開鍵暗号のアイデア.

この公開鍵暗号のアイデア実現に必要な一方向関数は、1978 年に Ron Rivest, Adi Shamir, Leonard Adleman によって発表され、その RSA 暗号では素因数分解に非常に時間がかかることを利用しています⁴。RSA 暗号は

公開鍵：数 E と数 N

秘密鍵：数 D と数 N

⁴素因数分解などの RSA 暗号の基礎的な数学などに興味のある人は付記 1 を参照ください。

暗号化：暗号文=平文^E mod N (平文を E 乗して N で割った余り)

復号化：平文=暗号文^D mod N (暗号文を D 乗して N で割った余り)

です。そこで必要となる E,D,N などの鍵ペアは

N：二つの素数 p,q から， $N=p*q$ で求める。

L：p-1 と q-1 の最小公倍数 (lcm) を求める。

E：L と互いに素な E を求める。

D： $E*D \bmod L = 1$ となる D を求める。

で生成します。

Maple の関連コマンド

公開鍵，暗号鍵を生成する Maple コマンドを見てみましょう。

N: RSA module, RSA モジュール $N:=P*Q;$

L: $L:=\text{lcm}(P-1,Q-1);$

厳密には $L:=\text{lcm}(P-1,Q-1);$ ですが， $P*Q$ を法とすると，全ての数が自分自身に戻るべき乗数は，n を任意の整数として $n*(P-1 \text{ と } Q-1 \text{ の最小公倍数})+1$ と表すことができる。 $(P-1)*(Q-1)$ は，必ず $(P-1 \text{ と } Q-1 \text{ の最小公倍数})$ の倍数になるのでこちらを用いてもよい。

E:Encryption exponent, 暗号化指数

$\text{gcd}(E,L)=1$ となる E を求めるのですが，これはたくさんあります。以下のスクリプトで全てを表示することが可能です。

```
for i from 1 to L do
  if gcd(i,L)=1 then
    printf("%4d,",i);
  end if;
od;
```

実際は乱数発生器を用いたり，他の適当な素数を使ったりしているようです。

D:Decryption exponent, 復号化指数

$E \cdot D \bmod L = 1$ となる D を求める. これを Maple では L を法とする E の-1乗として計算することが出来ます. Maple では D は微分を表す予約語なので DD とかにしておきます.

```
DD:=eval(1/E mod L);
```

現実的なより長い文字でできた平文 (plain text) を暗号文 (cipher text) にする場合を考えます. 大きな素数を乱数発生器を使って作る例です.

```
> M1 := rand(10^80)();
> M2 := rand(10^80)();
> P := nextprime(M1);
> Q := nextprime(M2);
> E:= 2^16+1; isprime(E); gcd(E,L);
```

`nextprime` は input の数字より大きな次の素数を返してくれる関数です. E をここでは2進数で0が並んだ数を使っています.

次に平文を大きな数の列に換える関数です. 中身は分からなくてもいいですが, 文字の扱いに興味のある人はばらしてみてください. (Mike May, S. J. の”Using RSA”より)

リスト : 文字の2桁の16進数表示への変換

```
twodigitex := a -> substring(convert(a+256,hex),2..3):
```

リスト : 平文の数字列への変換関数

```
shortconverter := proc(messagestring)
  local stringofhex, lengthofmess, hexstring, i:
  #First we convert the ASCII string to a list of decimal equivalents
  #Then we convert the decimal numbers to hex equivalents
  stringofhex := map(twodigitex, convert(messagestring,bytes)):
  print(stringofhex);
  lengthofmess := nops(stringofhex):
  print(lengthofmess);
  #Now we concatenate the hex numbers
  hexstring := cat(seq(stringofhex[i],i=1..lengthofmess)):
  #Finally we convert the big number back to decimal
  convert(hexstring,decimal,hex):
end:
```

 リスト：数字列から平文への逆変換関数

```

numtostring := proc(bignum)
local hexstr1, listlength, declist, i:
#convert to a hex string
hexstr1 := convert(bignum,hex):
#make sure the hex string has even length
if (length(hexstr1) mod 2) = 1 then hexstr1 := cat('0',hexstr1) fi;
#compute the number of characters
listlength := length(hexstr1)/2:
#convert to a vector of decimal numbers
declist := linalg[vector](listlength);
for i from 1 to listlength do
declist[i] := convert(substring(hexstr1,2*i-1..2*i),decimal,hex);
od;
#convert the vector to a list, then to an ASCII string
convert(convert(convert(declist,list),bytes),name);
end:

```

実際の操作を試しておきます。

```

> plaintext := "Good evening Kids, this afternoon we explore around RSA.";
> numberedText := shortconverter(plaintext);
> numtostring(numberedText);

```

解法のヒント

1. $P:=3;Q:=11;E:=3$; として暗号化関数, 復号化関数を定義せよ. 平文が3の場合に, 暗号化した数と, 復号化の結果を調べよ.
2. 前問の鍵を使って, $N(=33)$ 以下の全ての自然数について暗号化した数を求めよ. また, 復号化可能 (間違いが起こらない) であることを確かめよ.
3. twodigithex, shortconverter, numtostring の変数, 関数名などを調整して, 短い単語の数字列との変換・逆変換を確かめよ.
4. 冒頭の RSA 暗号の復号化を試みよ.

付記 1 : RSA 暗号の基礎理論と関連する Maple コマンド

本文では省略した, RSA の原理となる素数の性質についてエッセンスだけを記しておきます.

先ず基本となる中学で習った (はずの), 素数の復習です. 自然数 $p > 1$ は, 二つの整数 $1, p$ のみを約数とする場合, 素数 (prime number) と言われる. 素数でない自然数 $a > 1$ は合成数 (composite number) と言われる. なお, 1 は素数でも合成数でもない. また, 素数 p が整数 a を割り切れれば, p は a の素因数 (prime divisor) と言われる. 整数 a, b について $\gcd(a, b) = 1$ が成り立つとき, a と b は互いに素 (coprime) であると言われる. 全ての正整数は素数の積で表現可能であり, これを素因数分解 (factorization) と言う.

暗号文 C が平文 M に戻る条件を導くと, 暗号化と復号化の計算式は,

$$\begin{aligned} C &= M^e \pmod{n} \\ M &= C^d \pmod{n} \end{aligned}$$

であるから, ここで mod の「大きくなる前に適宜割っていても, 最終的な余りは変わらない」という性質

$$\{a * b\} \pmod{n} = \{(a \pmod{n}) * (b \pmod{n})\} \pmod{n}$$

を使って, 前者を後者に代入すると,

$$\begin{aligned} M &= (M^e)^d \pmod{n} \\ &= M^{ed} \pmod{n} \\ &= M^{ed} \pmod{pq} \end{aligned}$$

となる. この式を変形すると

$$\begin{aligned} M^{ed} - M &= 0 \pmod{n} \\ M * (M^{ed-1} - 1) &= 0 \pmod{pq} \end{aligned}$$

となる. この式がなりたつ条件は Fermat の小定理を使って求まり, RSA 暗号が成立する.

このあたりの詳しくて分かりやすい解説が

- <http://pgp.iijlab.net/crypt/rsa.html>:はやわかり RSA⁵
- <http://www.maitou.gr.jp/rsa/rsa10.php>:さるにもわかる RSA 暗号 (10. RSA 暗号の世界)⁶

にある.

素数絡みの Maple のコマンドのいくつかをまとめておきます.

mod, modp n を m で割ったあまりを求める関数.

```
n mod m;
modp(n,m);
```

ifactor ある数の素因数分解を求める関数

```
ifactor(91);
```

isprime ある数が素数なら true, 違えば false を返す関数

```
isprime(101);
```

Power mod の「大きくなる前に適宜割っていても、最終的な余りは変わらない」という性質

$\{a * b\} \pmod{n} = \{(a \pmod{n}) * (b \pmod{n})\} \pmod{n}$
 を使ってべき計算してくれる関数. mod と組み合わせて使う.
 例えば,

```
time(numberedText^E mod N);
```

と

```
time(Power(numberedText,E) mod N);
```

とをくらべよ. 通常のべき表示 "^" のかわりに "&^" を使っても Power と同じ効果が得られる.

nextprime ある数より大きな次の素数を返す関数.

```
nextprime(100);
```

lcm(Least Common Multiple) 最小公倍数を返す関数.

```
lcm(35,100);
```

gcd(Greatest Common Divisor) 最大公約数を返す関数.

```
gcd(35,100);
```

gcd の結果が 1, つまり 1 以外に共通に割り切れる数がない場合
 二つの数は互いに素であるという.

⁵<http://pgp.iijlab.net/crypt/rsa.html>

⁶<http://www.maitou.gr.jp/rsa/rsa10.php>

ithprime i 番目の素数を返す関数.

Fermat の小定理 p を素数, a を整数とすると,

$$\text{modp}(a^p - a, p) = 0$$

が成り立つ. さらに a と p とが互いに素であれば,

$$\text{modp}(a^{p-1}, p) = 1$$

が成り立つ.

定理 自然数 n より小さい自然数 a が n と互いに素であるとき,

$\text{mod}(ab, n) = 1$ となる b ($0 < b < n$) がただ一つ存在する.

Euler の定理 (の特別な場合) 自然数 $n (= p \cdot q)$ より小さい自然数 M が n と互いに素であるとき,

$$\text{mod}(M^{((p-1) \cdot (q-1))}, n) = 1 \quad (0 < x < n)$$

Euclid 互除法

拡張 Euclid 互除法

付記 2: グーグル人材募集広告

直接 RSA 暗号に関係するわけではありませんが, 素数つながりで載せておきます.

グーグル、謎の人材募集広告-シリコンバレーのビルボードに

Stefanie Olsen (CNET News.com)

2004/07/12 08:40

先週、シリコンバレーの中心を走るハイウェイ 101 沿いのビルボードに、複雑な数学の問題を載せた広告が現れた。(中略)

この広告には、「 $\{e$ の値中の、最初の連続する 10 桁の素数 $\}$.com」 (“{first 10-digit prime found in consecutive digits e }.com.”) と書かれている。この答えの「7427466391.com」にアクセスすると、そのウェブページにはさらに別の問題(下記参照)が用意されているが、ここにも Google が関与していることを示すものは全くない。

この問題を解くと、Google の研究開発部門「Google Labs」へのページに辿りつく。このページには、「Google の構築を通して我々が学んだことの 1 つに、自分が何かを探しているとき、向こうも自分を探している場合のほうが見つかりやすいということがある。我々が探しているのは、世界最高のエンジニアであり、あなたこそその人なのだ」と書かれている。

リスト : 7427466391.com

Congratulations. You've made it to level 2. Go to
www.Linux.org and enter Bobsyouruncle as the login
and the answer to this equation as the password.

f(1)=7182818284

f(2)=8182845904

f(3)=8747135266

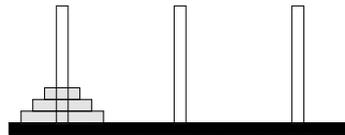
f(4)=7427466391

f(5)=_____

2.7 ハノイの塔

課題

3本の杭 (peg) があり, 最初の杭には, おおきな円盤 (ring) から小さな円盤が下から順に積まれて塔のようになっている. 適宜3本の杭に円盤を差しかえて, 円盤を最初の杭から最後の杭へすべて移し換えなさい. ただし, 一度に一つの円盤しか動かさない. また, 小さい円盤の上に大きな円盤を置いてはいけない.



課題の背景

ハノイ (バラモン) の塔伝説

ベナレスにあるバラモン教の大寺院は世界の中心と記されており, そこに, 杭が3本立った真鍮製の基盤がある. 神はその一本に64枚の純金の円盤をはめた. 僧侶たちは昼夜の別なくそれを別の杭に移し替える作業に専念している. そして移し替えが完了したとき, 寺院もバラモン僧たちも崩壊し, この世が塵に帰るという.

『ハノイの塔』は1883年にフランスの数学者E. リュカ (Edouard Lucas) が考えたゲームです. リュカがこの物語を創作したのか, はたまたこの物語から発案したのかは不明⁷.

Maple の関連コマンド

ハノイの塔の円盤の動きを視覚化してみます. 以下の一連のコマンドを打ち込んでください.

リスト: ハノイの塔の基本構成物 (基盤, 杭, 円盤) の描画.

```
with(plots):with(plottools):  
base:=rectangle([-4,0],[25,-1],color=black):  
peg1:=rectangle([0,0],[1,10],color=white):
```

⁷R. Douglas Hofstadter. Metamagical themas. Scientific American, 248(2):16-22, March 1983.

```

peg2:=rectangle([10,0],[11,10],color=white):
peg3:=rectangle([20,0],[21,10],color=white):
ring1:=rectangle([-3,0],[4,1],color=yellow):
ring2:=rectangle([-2,1],[3,2],color=yellow):
ring3:=rectangle([-1,2],[2,3],color=yellow):
display([ring1,ring2,ring3,peg1,peg2,peg3,base],axes=none,scaling=constrained);

```

リスト：初期位置と円盤の大きさ.

```

position=[[1,1],[1,2],[1,3]];
rw=[3,2,1]:

```

リスト：円盤の移動中の適切な描画.

```

move:=proc(input_list)
local rp1,rp2,rr1,tmp,width,i;
global rw;
rr1=[];
for i from 1 to 3 do
  tmp:=input_list[i];
  width:=rw[i];
  rp1=[-width+tmp[1]-1,tmp[2]-1];
  rp2=[width+tmp[1]-1+1,tmp[2]];
  rr1=[op(rr1),rectangle(rp1,rp2,color=yellow)]:
od:
[op(rr1),peg1,peg2,peg3,base];
end proc:

```

リスト：円盤の移動位置のコントロール.

```

move_control:=proc(ring_n,ini,fin)
local tmp,ini_p,fin_p,step,i;
global position;
tmp=[];
ini_p:=(ini[1]-1)*10+1;
fin_p:=(fin[1]-1)*10+1;
if (fin_p<ini_p) then step:=-1; else step:=1; end if;
for i from ini[2] to 12 do #UP
  position[ring_n]:=[ini_p,i];

```

```

    tmp:=[op(tmp),display(move(position))]:
od:
for i from ini_p to fin_p by step do #Lateral
    position[ring_n]:=[i,12];
    tmp:=[op(tmp),display(move(position))]:
od:
for i from 12 to fin[2] by -1 do #DOWN
    position[ring_n]:=[fin_p,i];
    tmp:=[op(tmp),display(move(position))]:
od:
end:

```

リスト : 円盤の移動の途中経過.

```

t1:=move_control(3,[1,3],[3,1]):
t1:=[op(t1),op(move_control(2,[1,2],[2,1]))]:
display(t1,insequence=true,axes=None);

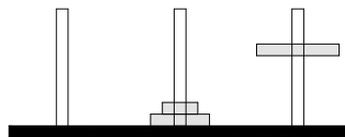
```

リスト : 三つの円盤の最終段階までの表示. 動画.

```

t1:=[op(t1),op(move_control(3,[3,1],[2,2]))]:
t1:=[op(t1),op(move_control(1,[1,1],[3,1]))]:
t1:=[op(t1),op(move_control(3,[2,2],[1,1]))]:
t1:=[op(t1),op(move_control(2,[2,1],[3,2]))]:
t1:=[op(t1),op(move_control(3,[1,1],[3,3]))]:
display(t1,insequence=true,axes=None);

```

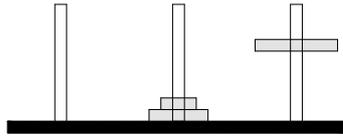


リスト : 外部ファイルへの動画の書きだし. QuickTime とかに食わせる.

```

plotsetup(gif,plotoutput='/Users/bob/Desktop/motion.gif');
display(t1,insequence=true,scaling=constrained,axes=None);
plotsetup(default);

```



解法のヒント

1. まず杭と円盤を抽象化して、配列に置き換えます。次のような peg 配列と top array を作れ。ここで配列の行が杭の番号を、配列の値がそこに刺さっている円盤の大きさを表す。また、top array は円盤が刺さっているもう一つ上の位置を示す。

$$\begin{bmatrix} 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, [4, 1, 1]$$

2. 円盤を動かす前後の杭の番号を受け取って、peg 配列、top array を操作する関数を作れ。この関数を MakeMove とすると以下のようなスクリプトによって移動が完了する。なお、ZeroArray は前問の初期配置を生成する関数である。

```
> max_disk:=3;
> ZeroArray(max_disk);
> MoveList:=[[1,3],[1,2],[3,2],[1,3],[2,1],[2,3],[1,3]]:
> n:=nops(MoveList):
> for i from 1 to n do
>   MakeMove(op(MoveList[i]));
> end do:
> print(peg):
```

3. 一番小さな円盤の動きをよく観察し、どうすれば自動化できるかを考えよ。
4. 一番小さな円盤の動きと、他の円盤の動きをコントロールする関数を考えよ。それぞれの関数を SmallestToWhere, DiskToWhere とすると以下のスクリプトによって円盤移動の自動化が達成される。なお、PositionSmallest は一番小さな円盤がどこにいるかを憶えておく変数。また MakeMove2 は先程の MakeMove に PositionSmallest の操作を加えた関数。

```
> max_disk:=3;
> count:=0;
> ZeroArray(max_disk);
> PositionSmallest:=1;
> MakeMove2(op(SmallestToWhere()));
> while peg[3,max_disk]=0 do
> MakeMove2(op(DiskToWhere()));
> MakeMove2(op(SmallestToWhere()));
> end do;
> print("Congratulations!");
```

5. 【発展問題】一枚の金貨を別の杭に移すのに1秒かかるとすると、バラモン僧たちが64枚の金貨を移すのに何年かかるか。