Maple で理解する数値計算の基礎

西谷@関西学院大・理工

2014 年 1 月 30 日

目次

第 1 章

代数方程式 (fsolve)

1.1 概要

代数方程式の解 f(x)=0 を数値的に求めることを考える．標準的な

二分法 (bisection method) とニュートン法 (Newton's method)

の考え方と例を説明し，

収束性 (convergency) と安定性 (stability)

について議論する．さらに収束判定条件について言及する．

二分法のアイデアは単純．中間値の定理より連続な関数では，関数の符号が変わる二つの変数の間には根が必ず存在する．したがって，この方法は収束性は決して高くはないが，確実．一方，Newton 法は関数の微分を用いて収束性を速めた方法である．しかし，不幸にして収束しない場合や微分に時間がかかる場合があり，初期値や使用対象には注意を要する．

1.2 Maple での解

Maple では代数方程式の解は，fsolve で求まる．

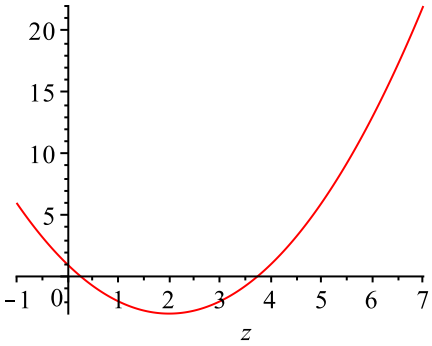
$$x^2 - 4x + 1 = 0$$

の解を考える．未知の問題では時として異常な振る舞いをする関数を相手にすることがあるので，先ずは関数の概形を見ることを常に心がけるべき．

```
> restart;
> func:=x->x^2-4*x+1;
```

$$func := x \mapsto x^2 - 4\,x + 1$$

```
> plot(func(z),z=-1..7);
```



もし，解析解が容易に求まるなら，その結果を使うほうがよい．Maple script の解析解を求める solve では，

```
> solve(func(x)=0,x);
```


と即座に求めてくれる．数値解は以下の通り求められる．

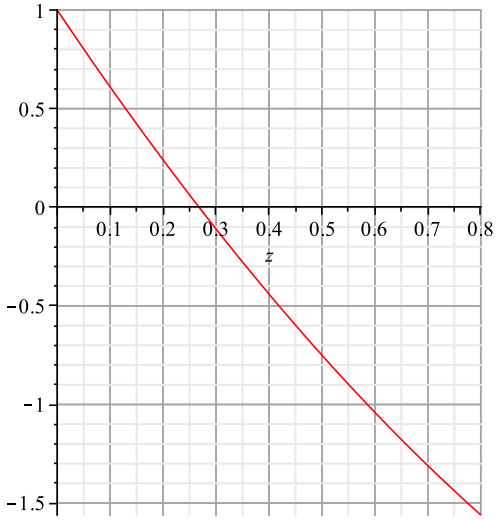
```
> fsolve(func(x)=0,x);
```


1.3　二分法と Newton 法の原理

1.3.1　二分法 (bisection)

二分法は領域の端 x_1, x_2 で関数値 $f(x_1), f(x_2)$ を求め，中間の値を次々に計算して，解を囲い込んでいく方法である．

```
> plot(func(z),z=0..0.8,gridlines=true);
```



x_1	x_2	$f(x_1)$	$f(x_2)$
0.0	0.8		

1.3.2 Newton 法 (あるいは Newton-Raphson 法)

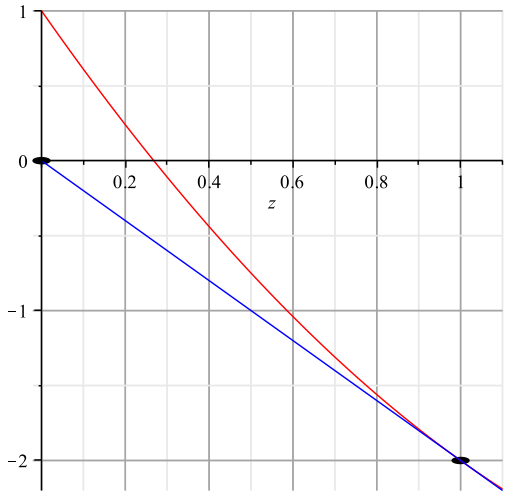
Newton 法は最初の点 x_1 から接線をひき，それが x 軸 (y=0) と交わった点を新たな点 x_2 とする．さらにそこでの接線を求めて...

という操作を繰り返しながら解を求める方法である．関数の微分を df(x) とすると，これらの間には

という関係が成り立つ．

```
> df:=unapply(diff(func(x),x),x);
```

```
> with(plots):with(plottools):
> x1:=1.0:x0:=0.0:
> p:=plot(func(z),z=0..1.1):
> p1:=plot(df(x1)*(z-x1)+func(x1),z=0..1.1,color=blue):
> p2:=[disk([x1,func(x1)],0.02), disk([x0,0],0.02)]:
> display(p,p1,p2,gridlines=true);
```



x_1	$f(x_1)$	$df(x_1)$
1.0		

1.4　二分法と Newton 法のコード

1.4.1　二分法 (bisection)

```
> x1:=0: x2:=0.8:
  f1:=func(x1): f2:=func(x2):
  for i from 1 to 5 do
    x:=(x1+x2)/2;
    f:=func(x);
    if f*f1>=0.0 then
      x1:=x; f1:=f;
    else
      x2:=x; f2:=f;
    end if;
    printf("%20.15f, %20.15f\n",x,f);
  end do:

0.400000000000000, -0.440000000000000
0.200000000000000,  0.240000000000000
0.300000000000000, -0.110000000000000
0.250000000000000,  0.062500000000000
0.275000000000000, -0.024375000000000
```

1.4.2　Newton 法 (あるいは Newton-Raphson 法)

```
> dfunc:=unapply(diff(func(z),z),z);
```

$$dfunc := z \mapsto 2z - 4$$

```
> x:=1: f:=func(x):
  printf("%15.10f, %+24.25f\n",x,f);
  for i from 1 to 5 do
    x:=x-f/dfunc(x);
    f:=func(x);
    printf("%15.10f, %+24.25f\n",x,f);
  end do:

1.0000000000, -2.0000000000000000000000000
0.0000000000, +1.0000000000000000000000000
```

```
0.2500000000, +0.062500000000000000000000
0.2678571429, +0.000318877551000000000000000
0.2679491900, +0.000000008472673797000000000
0.2679491924, +0.000000000000000000059821834
```

以下のように Digits を変更すれば，Maple では浮動小数点演算の有効数字を変えることができる．

```
> Digits:=40;
```

$$40$$

1.5 収束性と安定性

実際のコードの出力からも分かる通り，解の収束の速さは 2 つの手法で極端に違う．2 分法では一回の操作で解の区間が半分になる．このように繰り返しごとに誤差幅が前回の誤差幅の定数 (< 1) 倍になる方法は 1 次収束 (linear convergence) するという．Newton 法では関数・初期値が素直な場合 ($f'(x) <> 0$) に，収束が誤差の 2 乗に比例する 2 次収束を示す．以下はその導出を示した．

```
> restart; ff:=subs(xi-x[f]=ei,series(f(xi),xi=x[f],4));
```

$$ff := f(x_f) + D(f)(x_f)\,ei + \frac{1}{2}D^{(2)}(f)(x_f)\,ei^2 + \frac{1}{6}D^{(3)}(f)(x_f)\,ei^3 + O\left(ei^4\right)$$

```
> dff:=subs({0=x[f],x=ei},series(diff(f(x),x),x,3));
```

$$dff := D(f)(x_f) + D^{(2)}(f)(x_f)\,ei + \frac{1}{2}D^{(3)}(f)(x_f)\,ei^2 + O\left(ei^3\right)$$

```
> ei1:=ei-ff/dff;
```

$$ei1 := ei - \frac{f(x_f) + D(f)(x_f)\,ei + \frac{1}{2}D^{(2)}(f)(x_f)\,ei^2 + \frac{1}{6}D^{(3)}(f)(x_f)\,ei^3 + O\left(ei^4\right)}{D(f)(x_f) + D^{(2)}(f)(x_f)\,ei + \frac{1}{2}D^{(3)}(f)(x_f)\,ei^2 + O\left(ei^3\right)}$$

```
> ei2:=simplify(convert(ei1,polynom));
```

$$ei2 := \frac{1}{3}\,\frac{3D^{(2)}(f)(x_f)\,ei^2 + 2D^{(3)}(f)(x_f)\,ei^3 - 6f(x_f)}{2D(f)(x_f) + 2D^{(2)}(f)(x_f)\,ei + D^{(3)}(f)(x_f)\,ei^2}$$

```
> ei3:=series(ei2,ei,3);
```

$$ei3 := -\frac{f(x_f)}{D(f)(x_f)} + \frac{f(x_f)\left(D^{(2)}\right)(f)(x_f)\,ei}{\left(D(f)(x_f)\right)^2} + \frac{1}{6}\,\frac{3\left(D^{(2)}\right)(f)(x_f) + 3\,\frac{f(x_f)\left(D^{(3)}\right)(f)(x_f)}{D(f)(x_f)} - 6\,\frac{f(x_f)\left(\left(D^{(2)}\right)(f)(x_f)\right)^2}{\left(D(f)(x_f)\right)^2}}{\left(D(f)(x_f)\right)}\,ei^2 + O\left(ei^3\right) \tag{1.1}$$

```
> subs(f(x[f])=0,ei3);
```

$$\frac{1}{2}\frac{D^{(2)}(f)(x_f)\,ei^2}{D(f)(x_f)}+O\left(ei^3\right)$$

注意すべきは，この収束性には一回の計算時間の差は入っていないことである．Newton 法で解析的に微分が求まらない場合，数値的に求めるという手法がとられるが，これにかかる計算時間はばかにできない．二分法を改良した割線法 (secant method) がより速い場合がある (NumRecipe9 章参照)．

二分法では，収束は遅いが，正負の関数値の間に連続関数では必ず解が存在するという意味で解が保証されている．しかし，Newton 法では，収束は速いが，必ずしも素直に解に収束するとは限らない．解を確実に囲い込む，あるいは解に近い値を初期値に選ぶ手法が種々考案されている．解が安定であるかどうかは，問題，解法，初期値に大きく依存する．収束性と安定性のコントロールが数値計算のツボとなる．

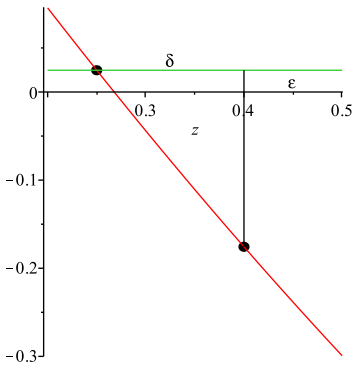
1.6　収束判定条件

どこまで値が解に近づけば計算を打ち切るかを決める条件を収束判定条件と呼ぶ．以下のような条件がある．

ε(イプシロン，epsilon) 法	
δ(デルタ，delta) 法	
占部法	数値計算の際の丸め誤差までも含めて判定する条件で， $\lvert f(x_{i+1})\rvert > \lvert f(x_i)\rvert$ とする．

■ϵ, δ を説明するための図

```
> with(plots):with(plottools):
> f2:=x->0.4*(x^2-4*x+1):x1:=0.25:x0:=0.4:
  p1:=plot([f2(z),f2(x1)],z=0.2..0.5):
  p2:=[disk([x1,f2(x1)],0.005),disk([x0,f2(x0)],0.005)]:
  l1:=line([x0,f2(x0)],[x0,f2(x1)]):
  t1 := textplot([0.45,0.0,‘epsilon‘],align=above):
  t2 := textplot([0.325,0.05,‘delta‘],align=below):
> display(p,p1,p2,l1,t1,t2);
```



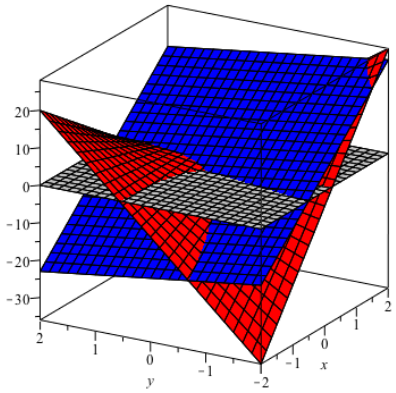
1.7　2 変数関数の場合

2 変数の関数では，解を求める一般的な手法は無い．この様子は実際に 2 変数の関数で構成される面の様子をみれば納得されよう．

```
> restart;
> f:=(x,y)->4*x+2*y-6*x*y; g:=(x,y)->10*x-2*y+1;
```

$$f := (x, y) \mapsto 4\,x + 2\,y - 6\,xy$$
$$g := (x, y) \mapsto 10\,x - 2\,y + 1$$

```
> p1:=plot3d({f(x,y)},x=-2..2,y=-2..2,color=red):
  p2:=plot3d({g(x,y)},x=-2..2,y=-2..2,color=blue):
  p3:=plot3d({0},x=-2..2,y=-2..2,color=gray):
  with(plots):
  display([p1,p2,p3],axes=boxed,orientation=[-150,70]);
```



解のある程度近くからは，Newton 法で効率良く求められる．

```
> fsolve({f(x,y)=0,g(x,y)=0},{x,y});
```

$$\{x = -0.07540291160, y = 0.1229854420\}$$

1.8　例題:二分法と Newton 法の収束性

代数方程式に関する次の課題に答えよ．(2004 年度期末試験)

1. $\exp(-x) = x^2$ を二分法およびニュートン法で解け．
2. n 回目の値 x_n と小数点以下 10 桁まで求めた値 $x_f = 0.7034674225$ との差 Δx_n の絶対値 (abs) の log を n の関数としてプロットし，その収束性を比較せよ．また，その傾きの違いを両解法の原理から説明せよ．

解答例

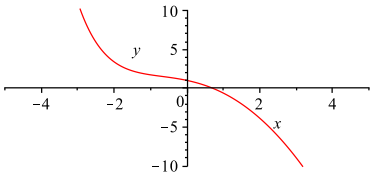
計算精度を 40 桁にしておく．func で関数を定義．

```
> restart; Digits:=40: func:=unapply(exp(-x)-x^2,x);
```

$$func := x \mapsto e^{-x} - x^2$$

先ずは，関数を plot して概形を確認.

```
> plot(func(x),x=-5..5,y=-10..10);
```



Maple の組み込みコマンドで正解を確認しておく.

```
> x0:=fsolve(func(x)=0,x);
```

0.70346742249839165204981860185990213034 29

テキストからプログラムをコピーして走らせてみる．環境によっては，printf 分の中の”\”が文字化けしているので修正.

```
> x1:=0: x2:=0.8: res1:=[]:
  f1:=func(x1): f2:=func(x2):
  for i from 1 to 20 do
    x:=(x1+x2)/2;
    f:=func(x);
    if f*f1>=0.0 then
      x1:=x; f1:=f;
    else
      x2:=x; f2:=f;
    end if;
    printf("%20.15f, %20.15f\n",x,f);
    res1:=[op(res1),[i,abs(x-x0)]]:
  end do:
```

```
0.400000000000000, 0.510320046035639
0.600000000000000, 0.188811636094026
0.700000000000000, 0.006585303791410
0.750000000000000, -0.090133447258985
0.725000000000000, -0.041300431044638
```

中略

```
0.703468322753906, -0.000001712107681
0.703467559814453, -0.000000261147873
```

プロットのためにリストを作成する．Maple でこれを実行するイディオムは以下の通り.

```
> res1:=[];
  for i from 1 to 3 do
```

```
    res1:=[op(res1),[i]];
  end do;
```

これを先のコードに組み込んでおいて得られた結果を logplot(片対数プロット) する.

```
> with(plots):
> logplot(res1); #res:結果は後に示す
```

同様に Newton 法での結果を res2 に入れる.

```
> dfunc:=unapply(diff(func(z),z),z);
```

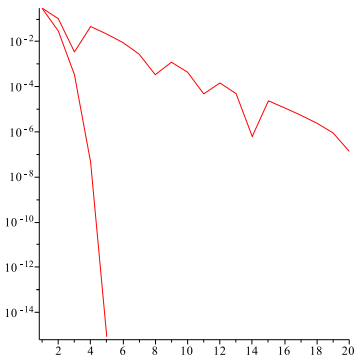
$$dfunc := z \mapsto -e^{-z} - 2\,z$$

```
> x:=1.0: f:=func(x):
  printf("%15.10f, %+24.25f\n",x,f);
  res2:=[[1,abs(x-x0)]]:
  for i from 2 to 5 do
    x:=x-f/dfunc(x);
    f:=func(x);
    printf("%15.10f, %+24.25f\n",x,f);
    res2:=[op(res2),[i,abs(x-x0)]]:
  end do:
```

```
1.0000000000, -0.6321205588285576784044762
0.7330436052, -0.0569084480040254074684576
0.7038077863, -0.0006473915387465014761973
0.7034674683, -0.0000000871660305624231097
0.7034674225, -0.0000000000000015809178420
```

res1, res2 を片対数プロットして同時に表示.

```
> l1:=logplot(res1);
> l2:=logplot(res2);
> display(l1,l2);
```

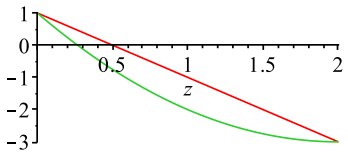


2 分法で求めた解は，Newton 法で求めた解よりもゆっくりと精密解へ収束している．これは，二分法が原理的に計算回数について一次収束なのに対して，Newton 法は 2 次収束であるためである．解の差 (δ) だけでなく，関数値 $f(x), \epsilon$ をとっても同様の振る舞いを示す.

1.9　課題

1. Newton 法の $f(x), df(x)$ の関係を示す式を導け．
2. 次の関数 $f(x) = \exp(-x) - 2\exp(-2x)$ の解を二分法，Newton 法で求めよ．
3. 代数方程式に関する次の課題に答えよ．(2004 年度期末試験)
   (a) $\exp(-x) = x^2$ を二分法およびニュートン法で解け．
   (b) n 回目の値 x_n と小数点以下 10 桁まで求めた値 $x_f = 0.7034674225$ との差 Δx_n の絶対値 (abs) の log を n の関数としてプロットし，その収束性を比較せよ．また，その傾きの違いを両解法の原理から説明せよ．
4. 次の方程式 $f(x) = x^4 - x - 0.12$ の正数解を二分法で求めよ．(2008 年度期末試験)
5. 収束条件がうまく機能しない例を示せ．
6. 割線法は，微分がうまく求まらないような場合に効率がよい，二分法を改良した方法である．二分法では新たな点を元の 2 点の中点に取っていた．そのかわりに下図に示すごとく，新たな点を元の 2 点を直線で内挿した点に取る．二分法のコードを少し換えて，割線法のコードを書け．また，収束の様子を二分法，Newton 法と比べよ．

```
> func:=x->x^2-4*x+1: x1:=0: x2:=2: f1:=func(x1): f2:=func(x2):
> plot({(z-x1)*(f1-f2)/(x1-x2)+f1,func(z)},z=0..2);
```



7. 次の方程式 $f(x) = \cos(x) - x^2$ の正数解を二分法で求めよ．割線法でも求め，収束性を比べよ．(2009 年度期末試験)
8. 次の方程式 $f(x) = x^3 - 3x + 3$ の解をニュートン法で求めよ．初期値をそれぞれ $x = -3, x = 2$ とした時を比べ，その差について論ぜよ．(2010 年度期末試験)

第 2 章

誤差 (Error)

2.1 打ち切り誤差と丸め誤差 (Truncation and round off errors)

数値計算のねらいは，できるだけ精確・高速に解を得ることである．誤差 (精度) と収束性 (安定性, 速度) が数値計算のキモとなる．前回に説明した収束判定条件による誤差は打ち切り誤差 (truncation error) と呼ばれる．ここでは，誤差のもう一つの代表例である，計算機に特有の丸め誤差 (roundoff error) について見ておこう．

2.1.1 整数型と実数型の内部表現

計算機は一般に無限精度の計算をおこなっているわけではない．CPU で足し算をおこなう以上，一般的な計算においては CPU が扱う一番効率のいい数の大きさが存在する．これが,32bit の CPU では 1 ワード,4byte(4x8bits) である．1 ワードで表現できる最大整数は, 符号に 1bit 必要なので,2^(31)-1 となる．実数は以下のような仮数部と指数を取る浮動小数点数で表わされる．

表 2.1　浮動小数点数の内部表現 (IEEE754).

	$s \times f \times B^{e-E}$
s	sign bit(符号ビット:正負の区別を表す)
e	biased exponent(指数部)
f	fraction portion of the number(仮数部)
B	base(基底) で通常は 2
E	bias(下駄) と呼ばれる
real(単精度)	s=1, e=8, f=23 E=127
double precision(倍精度)	s=1, e=11, f=52 E=1023

E は指数が負となる小数点以下の数を表現するためのもの．演算結果は実際の値から浮動小数点数に変換するための操作「丸め (round-off)」が常に行われる．それに伴って現れる誤差を丸め誤差と呼ぶ．

2.2 有効桁数 (Significant digits)

1 ワードの整数の最大値とその 2 進数表示.

```
> restart;
> 2^(4*8-1)-1;#res: 2147483647
```

この整数を 2 進数で表示するように変換するには，convert(n,binary) を用いて，

```
> convert(2^(4*8-1)-1,binary);  #res: 11111111111111111111111111111111
```

となり，31 個の 1 が並んでいることが分かる．1 ワードの整数の最大桁は，n の桁数を戻すコマンド length(n) を使って，

```
> length(2^(4*8-1)-1);  #res: 10
```

となり，たかだか 10 桁程度であることが分かる．一方，64bit の場合の整数の最大桁．

```
> length(2^(8*8-1)-1);  #res: 19
```

である．

Maple では多倍長計算するので，通常のプログラミング言語で起こる int の最大数あたりでの奇妙な振る舞いは示さない．

```
> 2147483647+100;  #res: 2147483747
```

単精度の浮動小数点数は, 仮数部 2 進数 23bit,2 倍長実数で 52bit である．この有効桁数は以下の通り．

```
> length(2^(23)); #res: 7
> length(2^(52)); #res: 16
```

2.3　浮動小数点演算による過ち (FloatingPointArithmetic)

「丸め」にともなって誤差が生じる．C や Fortran 等の通常のプログラミング言語では「丸める」仕様なのでプログラマーが気をつけなければならない．

プログラムリスト ： 実数のケタ落ち

```
#include <stdio.h>

int main(void){
  float a,b,c;
  double x,y,z;

  a=1.23456789;
  printf(" a= %17.10f\n",a);

  b=100.0;
  c=a+b;
  printf("%20.10f %20.10f %20.10f\n",a,b,c);

  x=(float)1.23456789;
  y=(double)100;
  z=x+y;
  printf("%20.12e %20.12e %20.12e\n",x,y,z);

  x=(double)1.23456789;
  y=(double)100;
  z=x+y;
  printf("%20.12e %20.12e %20.12e\n",x,y,z);
```

```
  return 0;
}
```

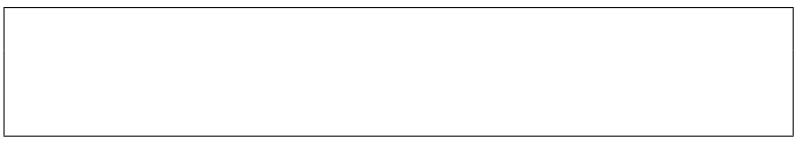
分かっているつもりでも，よくやる間違い．

プログラムリスト ： 丸め誤差

```
#include <stdio.h>

int main(void){
  float x=77777,y=7,y1,z,z1;
  y1=1/y;
  z=x/y;
  z1=x*y1;
  printf("%10.2f %10.2f\n",z,z1);
  if (z!=z1){
    printf("z is not equal to z1.\n");
  }
  printf("Surprising?? \n\n\n\n\n%10.5f %10.5f\n",z,z1);
  return 0;
}
```

これを避けるには，EPSILON という小さな数字を定義しておいて，値の差の絶対値を求める fabs を使って



とすべき．このときは数学関数である fabs を使っているので，

```
> gcc -lm test.c
```

と math library を明示的に呼ぶのを忘れないように．

2.4 機械精度 (Machine epsilon)

上の例では，浮動小数点数で計算した場合に小さい数の差を区別することができなくなるということを示している．これは，CPU に固有の精度で，機械精度 (Machine epsilon) と呼ばれる．つまり，小さい数を足したときにその計算機がその差を認識できなくなる限界ということで，以下のようにして求めることができる．

```
> Digits:=7;
> e:=evalf(1.0);
> w:=evalf(1.0+e);
> while (w>1.0) do
    printf("%-15.10e %-15.10e %-15.10e\n",e,w,evalf(w-1.0));
    e:=evalf(e/2.0);
    w:=evalf(1.0+e);
  end do:
```

7
1.0
2.0

```
1.0000000000e+00 2.0000000000e+00 1.0000000000e+00
5.0000000000e-01 1.5000000000e+00 5.0000000000e-01
2.5000000000e-01 1.2500000000e+00 2.5000000000e-01
1.2500000000e-01 1.1250000000e+00 1.2500000000e-01
6.2500000000e-02 1.0625000000e+00 6.2500000000e-02
3.1250000000e-02 1.0312500000e+00 3.1250000000e-02
1.5625000000e-02 1.0156250000e+00 1.5625000000e-02
7.8125000000e-03 1.0078120000e+00 7.8120000000e-03
3.9062500000e-03 1.0039060000e+00 3.9060000000e-03
1.9531250000e-03 1.0019530000e+00 1.9530000000e-03
9.7656250000e-04 1.0009770000e+00 9.7700000000e-04
4.8828120000e-04 1.0004880000e+00 4.8800000000e-04
2.4414060000e-04 1.0002440000e+00 2.4400000000e-04
1.2207030000e-04 1.0001220000e+00 1.2200000000e-04
6.1035150000e-05 1.0000610000e+00 6.1000000000e-05
3.0517580000e-05 1.0000310000e+00 3.1000000000e-05
1.5258790000e-05 1.0000150000e+00 1.5000000000e-05
7.6293950000e-06 1.0000080000e+00 8.0000000000e-06
3.8146980000e-06 1.0000040000e+00 4.0000000000e-06
1.9073490000e-06 1.0000020000e+00 2.0000000000e-06
9.5367450000e-07 1.0000010000e+00 1.0000000000e-06
```

2.5　桁落ち，情報落ち，積み残し (Cancellation)

■桁落ち (Cancellation)

$$\begin{array}{rr} & 0.723657 \\ - & 0.723649 \\ \hline \end{array}$$

■情報落ち (Loss of Information)

$$\begin{array}{rr} & 72365.7 \\ + & 1.23659 \\ \hline \end{array}$$

■積み残し

$$\begin{array}{rr} & 72365.7 \\ + & 0.001 \\ \hline \end{array}$$

2.6　課題

1. 次の項目について答えよ．(2004, 05, 06 年度期末試験)
   (a) 数値計算の精度を制約するデータ形式とその特徴は何か．
   (b) 丸め誤差とは何か．

(c) 打ち切り誤差とは何か.

(d) 安定性とは何か.

2. 10 進数 4 桁の有効桁数をもった計算機になったつもりで，以下の計算をおこなえ.
   (a) 2718-0.5818 (b) 2718+0.5818 (c) 2718/0.5818 (d) 2718*0.5818
3. 自分の計算機で機械精度がどの位かを確かめよ．Maple スクリプトを参照して, C あるいは Fortran で作成し, 適当に調べよ.
4. (2147483647 + 100) を C あるいは Fortran で試せ.
5. 係数を a = 1, b = 10000000, c = 1 としたときに, 通常の解の公式を使った解と, 解と係数の関係 (下記の記述を参照）を使った解とを出力するプログラムを作成し, 解を比べよ.

   2 次方程式 $ax^2 + bx + c = 0$ の係数 a, b, c が特殊な値をもつ場合，通常の解の公式

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

   にしたがって計算するとケタ落ちによる間違った答えを出す．その特殊な値とは

$$\sqrt{b^2 - 4ac} \approx |b|$$

   となる場合である.

   ケタ落ちを防ぐには, $b > 0$ の場合は,

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

   として，ケタ落ちを起こさずに求め, この解を使って, 解と係数の関係より

$$x_2 = \frac{c}{a\, x_1}$$

   で求める．$b < 0$ の場合は，解の公式の足し算の方を使って同様に求める.

第 3 章

線形代数–写像 (LAFundamentals)

3.1 行列と連立方程式

大学の理系で必修なのは微積分と線形代数です．線形代数というと逆行列と固有値の計算がすぐに思い浮かぶでしょう．計算がややこしくてそれだけでいやになります．でも，行列の計算法は一連の手順で記述できるので，Maple では微積分とおなじように一個のコマンドで片が付きます．それが 3x3 以上でも同じです．問題はその意味です．ここでは，線形代数の計算が Maple を使えばどれほど簡単にできるかを示すと共に，線形代数の基本となる概念についてスクリプトと描画を使って，直観的に理解することを目的とします．

先ずは連立方程式から入っていきます．中学の時に

$$4x = 2$$

というのを解きますよね．一般的には

$$\begin{aligned} ax &= b \\ x &= b/a \end{aligned}$$

と書けるというのは皆さんご存知のはず．これと同じようにして連立方程式を書こうというのが逆行列の基本．つまり

$$\begin{aligned} 2x + 5y &= 7 \\ 4x + y &= 5 \end{aligned}$$

という連立方程式は，係数から作られる 2x2 行列を係数行列 A，左辺の値で作るベクトルを b として，

$$\begin{aligned} Ax &= b \\ x &= b/A = A^{-1}b \end{aligned}$$

としたいわけです．

実際に Maple でやってみましょう．行列は英語で Matrix です．

```
> restart: A:=Matrix([[2,5],[4,1]]);
```

$$A := \begin{bmatrix} 2 & 5 \\ 4 & 1 \end{bmatrix}$$

こうして行列を作ります．

```
> b:=Vector([7,5]); #(2) ベクトルは英語で Vector です．これで縦ベクトルができます．
```

$$b := \begin{bmatrix} 7 \\ 5 \end{bmatrix}$$

線形代数は linear algebra と言います．with で LinearAlgebra というライブラリーパッケージを読み込んでおきます．

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

逆行列は matrix inverse と言います．

```
> x0:=MatrixInverse(A).b;
```

行列 A の MatrixInverse を求めて，ベクトル b に掛けています．

$$x0 := \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

と簡単に求めることができます．

3.2　掃き出し

係数行列 A とベクトル b を足して作られる行列は拡大係数行列と呼ばれます．Maple では，これは

```
> <A|b>;
```

$$\begin{bmatrix} 2 & 5 & 7 \\ 4 & 1 & 5 \end{bmatrix}$$

として作られます．ここから行列の掃き出し操作をおこなうには，LUDecomposition というコマンドを使います．

```
> P,L,U:=LUDecomposition(<A|b>);
```

$$P, L, U := \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 5 & 7 \\ 0 & -9 & -9 \end{bmatrix}$$

これは，下三角行列 (Lower Triangle Matrix) と上三角行列 (Upper Triangle Matrix) に分解 (decompose) するコマンドです．P 行列は置換 (permutation) 行列を意味します．LUDecomposition だけでは，前進消去が終わっただけの状態です．そこで，後退代入までおこなうには，option に output='R' をつけます．そうすると出力は，

```
> LUDecomposition(<A|b>,output='R');
```

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

で，b ベクトルの部分が解になっています．

3.3　写像

次に，これを 2 次元上のグラフで見てみましょう．先ず描画に必要なライブラリーパッケージ (plots および plottools) を with で読み込んでおきます．

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

ベクトルは，位置座標を意味するように list へ変換 (convert) しておきます．

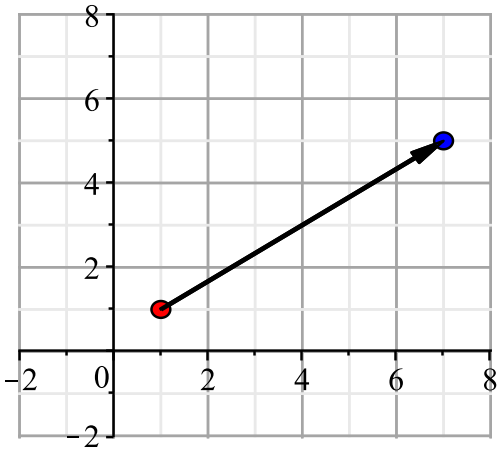
```
> p0:=convert(x0,list); p1:=convert(b,list);
```

位置 p0 に円 (disk) を半径 0.2, 赤色で描きます．同じように位置 p1 に半径 0.2，青色で disk を描きます．もう一つ，p0 から p1 に向かう矢印 (arrow) を適当な大きさで描きます．後ろの数字をいじると線の幅や矢印の大きさが変わります．

```
> point1:=[disk(p0,0.2,color=red), disk(p1,0.2,color=blue)]:
> line1:=arrow(p0,p1,.05,.3,.1 ):
```

これらをまとめて表示 (display) します．このとき，表示範囲を-8..8,-8..8 とします．

```
> display(point1,line1,view=[-2..8,-2..8],gridlines=true);
```



逆行列は

```
> MatrixInverse(A);
```

$$\begin{bmatrix} -1/18 & 5/18 \\ 2/9 & -1/9 \end{bmatrix}$$

で求まります．先ほどの矢印を逆に青から赤へたどる変換になっています．これが，連立方程式を解く様子をグラフで示しています．つまり，行列 A で示される変換によって求まる青点で示したベクトル b(7,5) を指す元の赤点を捜すというものです．答えは (1,1) となります．

では，元の赤点をもう少しいろいろ取って，行列 A でどのような点へ写されるかを見てみましょう．

```
> N:=30:point2:=[]:line2:=[]:
  for k from 0 to N-1 do
    x0:=Vector([sin(2*Pi*k/N),cos(2*Pi*k/N)]);
    x1:=A.x0;
    p0:=convert(x0,list);
    p1:=convert(x1,list);
    point2:=[op(point2),disk(p0,0.05,color=red)];
    point2:=[op(point2),disk(p1,0.05,color=blue)];
    line2:=[op(line2),line(p0,p1)];
  end do:
```

N:=30 で分割した円周上の点を x0 で求めて，point2 にその円とそれの A.x0 を，line2 にはその 2 点を結ぶ line(線) を足しています．使っているコマンドは，先ほどの描画とほぼ同じです．ただし，Maple スクリプトに特有の idiom(熟語) を使っています．この基本形をとり出すと，

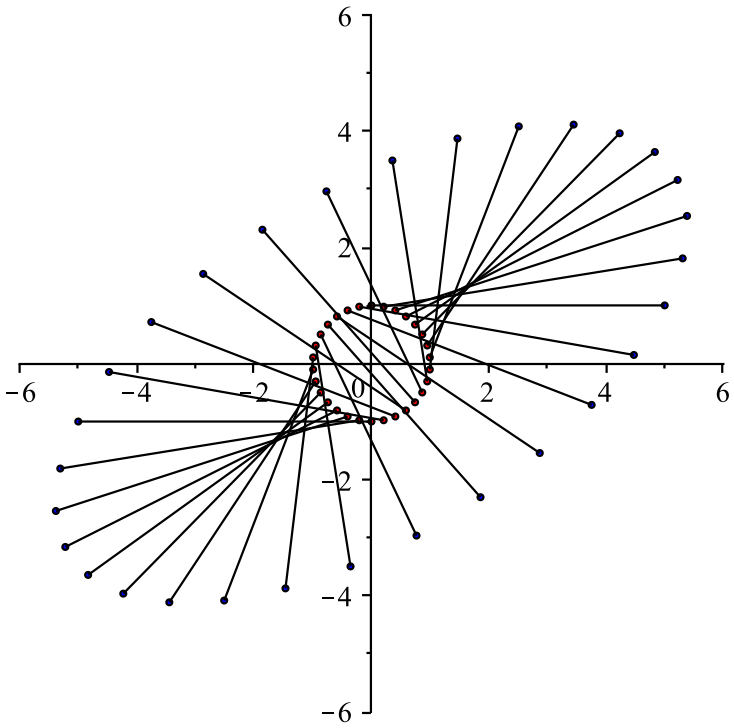
```
> list1:=[];
  for k from 0 to 2 do
    list1:=[op(list1),k];
  end do;
  list1;
```

$$[]$$
$$[0]$$
$$[0, 1]$$
$$[0, 1, 2]$$
$$[0, 1, 2]$$

となります．for-loop で k を 0 から 4 まで回し，list1 に次々と値を追加していくというテクです．

できあがりの次の図を見てください．

```
> d:=6: display(point2,line2,view=[-d..d,-d..d]);
```



何やっているか分かります? 中心の赤点で示される円が，青点で示される楕円へ写されていることが分かるでしょうか．

線形代数の講義で，写像を示すときによく使われるポンチ絵を現実の空間で示すとこのようになります．ポンチ絵では，赤で示した V 空間が青で示した W 空間へ行列 A によって写像され，それぞれの要素 v が w へ移されると意図しています．

3.4　固有ベクトルの幾何学的意味

では，ここでクイズです．固有ベクトルは上のグラフの何処に対応するか?　ヒントは，

行列 A の固有値，固有ベクトルを λ, x_0 とすると，

$$A\,x_0 = \lambda\,x_0$$

が成立する

です．固有値と固有ベクトルは Maple では以下のコマンドで求まります．

```
> lambda,P:=Eigenvectors(A);
```

$$\lambda,\ P := \begin{bmatrix} -3 \\ 6 \end{bmatrix}, \begin{bmatrix} -1 & 5/4 \\ 1 & 1 \end{bmatrix}$$

ここでは Maple コマンドの Eigenvectors で戻り値を λ(lambda と書きます)，P に代入しています．この後ろ側にある行列 P の 1 列目で構成されるベクトルが固有値-3 に対応する固有ベクトル，2 列目のベクトルが固有値 6 に対応する固有ベクトルです．

3.4.1 解答

固有値 λ，固有ベクトル x_0 の関係式

$$A\,x_0 = \lambda\,x_0$$

を言葉で言い直すと，

固有ベクトル x_0 は変換行列 A によって，自分の固有値倍のベクトル λx_0 に写されるベクトル

となります．つまり変換の図で言うと，

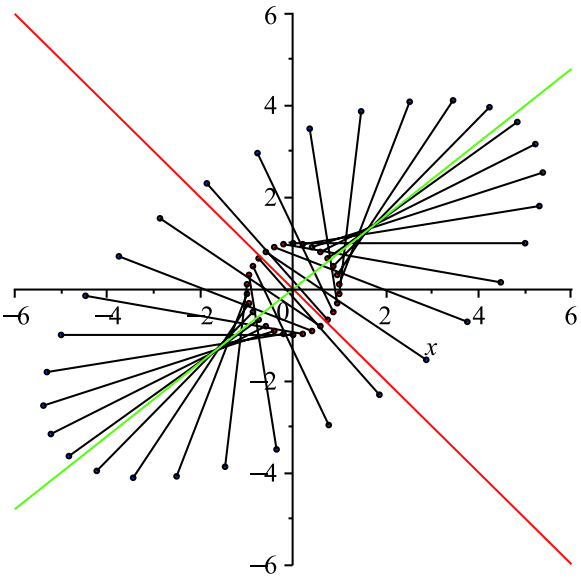
変換しても方向が変わらない赤点（の方向）

となります．これは図で書くと

```
> vv1:=Column(P,1): vv2:=Column(P,2):
  a1:=vv1[2]/vv1[1]: a2:=vv2[2]/vv2[1]:
  pp1:=plot({a2*x,a1*x},x=-d..d):
```

Column によって行列の第 i 列目をとりだし，その比によって直線の傾きを求めています．そうして引いた 2 本の直線を pp1 としてため込んで，先ほど描いた変換の図に加えて表示 (display) させます．

```
> display(point2,line2,pp1,view=[-d..d,-d..d]);
```



pp1 を入れて描いた直線が引かれた方向ではたしかに変換によっても方向が変わらなさそうに見えるでしょう．

おまけですが，行列の対角化は次のようにしてできます．

```
> MatrixInverse(P).A.P;
```

$$\begin{bmatrix} -3 & 0 \\ 0 & 6 \end{bmatrix}$$

3.5　行列式の幾何学的意味

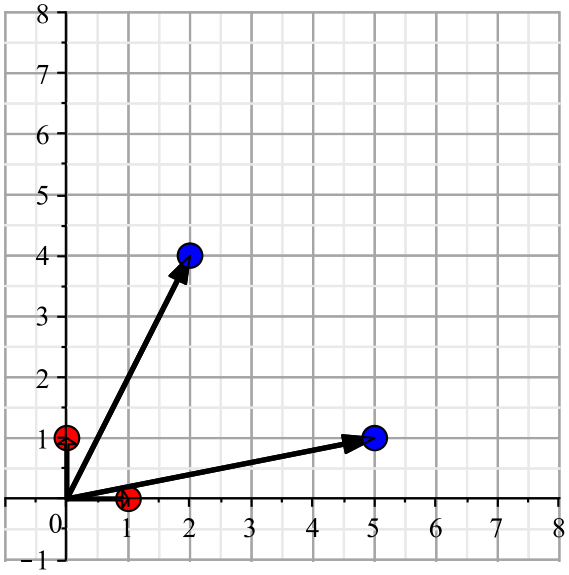
行列 A の行列式 ($|A|$ あるいは $\det A$ と表記) は Determinant で求まります．

```
> Determinant(A);
```

$$-18$$

では次のクイズ．先ほど求めた，行列 A の行列式は，どこに対応するでしょう？ 以下の (1,0),(0,1) の点を変換した点に原点からベクトルを結んでその意味を説明してください．さらに，そのマイナスの意味は？．

```
> point3:=[]:line3:=[]: XX:=Matrix([[1,0],[0,1]]):
  for i from 1 to 2 do
    x0:=Column(XX,i); x1:=A.x0;
    p0:=convert(x0,list):
    p1:=convert(x1,list):
    point3:=[op(point3),disk(p0,0.2,color=red),disk(p1,0.2,color=blue)]:
    line3:=[op(line3),arrow([0,0],p0,.05,.3,.1 ),arrow([0,0],p1,.05,.3,.1 )]:
  end do:
  display(point3,line3,view=[-1..8,-1..8],gridlines=true);
```



3.6　行列式が 0 の写像

では，行列式が 0 になるというのはどういう状態でしょう？ 次のような行列を考えてみましょう．

```
> A:=Matrix([[2,1],[4,2]]);
```

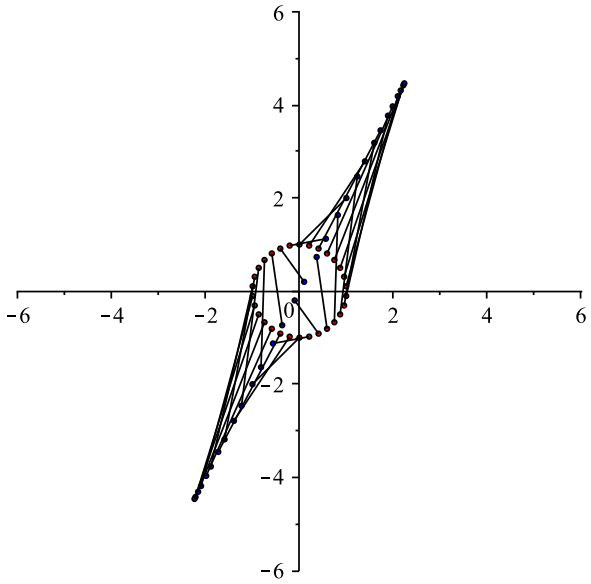
この行列式は

```
> Determinant(A);
```

$$0$$

です．この変換行列で，上と同じように写像の様子を表示させてみましょう．

```
> N:=30:point2:=[]:line2:=[]:
  for k from 0 to N-1 do
    x0:=Vector([sin(2*Pi*k/N),cos(2*Pi*k/N)]); x1:=A.x0; p0:=convert(x0,list);
    p1:=convert(x1,list);
    point2:=[op(point2),disk(p0,0.05,color=red),disk(p1,0.05,color=blue)];
    line2:=[op(line2),line(p0,p1)];
  end:
> d:=6: display(point2,line2,view=[-d..d,-d..d]);
```



わかります？

今回の移動先の青点は直線となっています．つまり，determinant が 0 ということは，変換すると面積がつぶれるという事を意味しています．平面がひとつ次元を落として線になるということです．

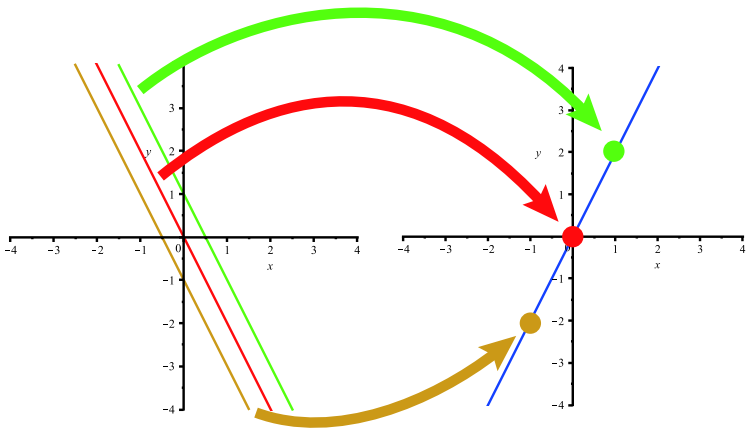
次に，この行列の表わす写像によって原点 (0,0) に写される元の座標を求めてみます．連立方程式に戻してみると

```
> A.Vector([x,y])=Vector([0,0]);
```

$$\begin{bmatrix} 2x+y \\ 4x+2y \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

となります．とよく見ると，1 行目も 2 行目もおなじ式になっています．2 次元正方行列で，行列式が 0 の時には必ずこういう形になり，直線の式となります．これを表示すると

```
> plot([-2*x,-2*x+1,-2*x-1],x=-4..4,y=-4..4);
> plot([2*x],x=-4..4,y=-4..4,color=blue);
```



左図の赤線となります．この直線上の全ての点が [0,0] へ写されることを確認してください．また，緑の線上の点は全て [1,2] へ写されることが確認できます．

```
> A.Vector([-1,2]);
```

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

こうしてすべて調べていけば，左の平面上のすべて点は右の青の直線上へ写されることが分かります．今まで見てきた円と楕円とはまったく違った写像が，行列式が 0 の行列では起こっていることが分かると思います．右の青線を行列 A による像 (Image, ImA と表記)，左の赤線，つまり写像によって [0,0] へ写される集合を核 (Kernel, Ker$A$ と表記) と呼びます．

これをポンチ絵で描くと，次の通りです．

像 (Image)	核 (Kernel)

3.7　全単射

行列 A による写像を f として，赤点に限らず元の点の集合を V, 移った先の点の集合を W とすると，

$$f : V \to W$$

と表記されます．v, w を V, W の要素としたとき，異なる v が異なる w に写されることを単射，全ての w に対応する v がある写像を全射と言います．全単射，つまり全射でかつ単射，だと要素は一対一に対応します．先ほどの A は全射でもなく，単射でもない例です．

行列式が 0 の場合の写像は単射ではありません．このとき，逆写像が作れそうにありません．これを連立方程式に戻して考えましょう．もともと，

$$v = A^{-1}w$$

の解 v は点 w が写像 A によってどこから写されてきたかという意味を持ちます．逆写像が作れない場合は，連立方程式の解はパラメータをひとつ持った複数の解 (直線) となります．これが係数行列の行列式が 0 の場合に，連立方程式の解が不定となる，あるいは像がつぶれるという関係です．

行列の次元が高い場合には，いろいろなつぶれかたをします．行列の階数と次元は

```
> Rank(A);
  Dimension(A);
```

$$1$$
$$2, 2$$

で求まります．

A を m 行 n 列の行列とするとき，

$$\text{Rank}(A) = \text{Dimension (Im } A)$$
$$\text{Dimension (Ker } A) = n \text{ - Rank}(A)$$

が成立し，これを次元定理といいます．全射と単射の関係は，下の表のような一変数の方程式での解の性質の拡張と捉えることができます．

表 3.1　代数方程式 $ax = b$ の解の存在性.

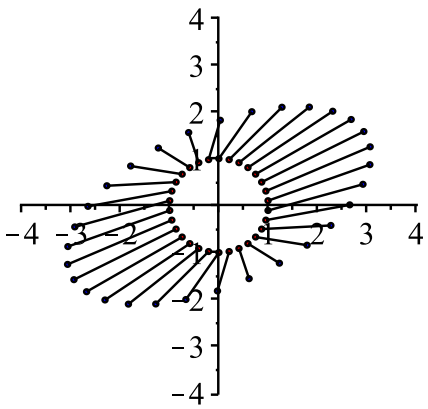
一意	$a <> 0$	$x = b/a$
不定	$a = 0, b = 0$	解は無数
不能	$a = 0, b <> 0$	解は存在しない

表 3.2　連立方程式 $Ax = b$ の解の存在性.

$m \times n$ 行列 A	全射でない (Im $A < m$), 値域上にあるときのみ解が存在	全射 (Im$A = m$), 解は必ず存在
単射でない (Ker $A <> 0$), 解は複数		
単射 (Ker $A = 0$), 解はひとつ		

3.8 課題

1. 下の図は $A := \begin{bmatrix} 3 & 2/3 \\ 2/3 & 2 \end{bmatrix}$ を用いて変換される像を表わしている．この行列の固有値，行列式が何処に対応するか説明せよ．また，固有ベクトルの方向を記せ．(2007 年度期末試験)



第 4 章

線形代数–逆行列 (LAMatrixInverse)

4.1 行列計算の概要

数値計算の中心課題の一つである，行列に関する演算について見ていく．多次元，大規模な行列に対する効率のよい計算法が多数開発されており，多くの既存のライブラリが用意されている．本章ではそれらの中心をなす，逆行列 (matrix inverse) と固有値 (Eigen values) に関して具体的な計算方法を示す．現実的な問題には既存のライブラリを使うのが上策であるが，それでも基礎となる原理の理解や，ちょっとした計算，ライブラリの結果の検証に使えるルーチンを示す．

逆行列は連立一次方程式を解くことと等価である．ルーチン的なやり方にガウスの消去法がある．これは上三角行列になれば代入を適宜おこなうことで解が容易に求まることを利用する．さらに，初期値から始めて次々に解に近づけていく反復法がある．この代表例である Jacobi(ヤコビ) 法と，収束性を高めた Gauss-Seidel(ガウス- ザイデル) 法を紹介する．

上記の手法をより高速にした修正コレスキー分解と共役傾斜 (共役勾配) 法があるが，少し複雑になるので割愛する．必要ならば NumRecipe を読め．

4.2 ガウス消去法による連立一次方程式の解

逆行列は連立一次方程式を解くことと等価である．すなわち，A を行列，x を未知数ベクトル，b を数値ベクトルとすると，

$$\begin{aligned} Ax &= b \\ A^{-1}Ax &= A^{-1}b \\ x &= A^{-1}b \end{aligned}$$

である．未知数の少ない連立一次方程式では，適当に組み合わせて未知数を消していけばいいが，未知数が多くなってしまうと破綻する．未知数の多い多元連立一次方程式で，ルーチン的に解を求めていく方法がガウス消去法で，前進消去と後退代入という 2 つの操作からなる．

後退代入 (Backward substitution) による解の求め方を先ず見よう．たとえば，

$$\begin{aligned} x + y - 2z &= -4 \\ -3y + 3z &= 9 \\ -z &= -2 \end{aligned}$$

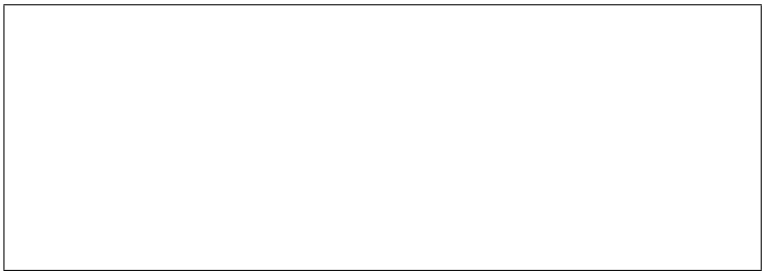
では，下から順番に $z \to y \to x$ と適宜代入することによって，簡単に解を求めることが出来る．係数で作る行列でこのような形をした上三角行列にする操作を前進消去あるいはガウスの消去法 (Gaussian elimination) という．下三角行列 L(lower triangular matrix) と上三角行列 U(upper triangular matrix) の積に分解する操作

$$A = L.U$$

を LU 分解 (LU decomposition) という．例えば先に示した上三角行列を係数とする連立方程式は，

$$\begin{aligned} x + y - 2z &= -4 \\ x - 2y + z &= 5 \\ 2x - 2y - z &= 2 \end{aligned}$$

を変形することで得られる．この変形を示せ．



4.3　Maple による LU 分解

係数行列 (coefficient matrix) と定数項 (b) との関係は以下の通りである．

```
> restart;
  A:=Matrix([[1,1,-2],[1,-2,1],[2,-2,-1]]):
  X:=Vector([x,y,z]):
  #X:=Vector([1,-1,2]):
  b:=Vector([-4,5,2]):
  A.X=b;
```

$$\begin{bmatrix} x+y-2\,z \\ x-2\,y+z \\ 2\,x-2\,y-z \end{bmatrix} = \begin{bmatrix} -4 \\ 5 \\ 2 \end{bmatrix}$$

単に逆行列を求める際は

```
> with(LinearAlgebra):
  MatrixInverse(A);
```

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

である．Maple では行列を三角行列に分解するために，LUDecomposition コマンドが用意されている．

```
> P,L,U:=LUDecomposition(<A|b>);
```

$$P,\,L,\,U := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 4/3 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 1 & -2 & -4 \\ 0 & -3 & 3 & 9 \\ 0 & 0 & -1 & -2 \end{bmatrix}$$

係数と定数項から作られる行列を拡大係数行列 (augmented matrix) といい，Maple では，$< A|b >$ で作られる．後退代入までおこなって連立方程式の解を求めるには，以下の通り option に output='R' をつける．

```
> LUDecomposition(<A|b>,output='R');
```

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

4.4　LU 分解のコード

LU 分解すれば線形方程式の解が容易に求まることは理解できると思う．具体的に A を LU 分解する行列 (消去行列と称す)T1,T2 の係数は次のようにして求められる．

```
> A0:=Matrix([[1,1,-2],[1,-2,1],[2,-2,-1]]):
  b0:=Vector([-4,5,2]):
  A:=Matrix(A0): B:=Vector(b0): n:=3:
  L:=Matrix(array(1..n,1..n,identity)):
  for i from 1 to n do #i 行目
    T[i]:=Matrix(array(1..n,1..n,identity)):
                              #i 番目の消去行列を作る
    for j from i+1 to n do
      am:=A[j,i]/A[i,i];      #i 行の要素を使って，i+1 行目の先頭を消す係数を求める
      T[i][j,i]:=-am;         #i 番目の消去行列に要素を入れる
      L[j,i]:=am;             #LTM の要素
      for k from 1 to n do
        A[j,k]:=A[j,k]-am*A[i,k]; #もとの行列を UTM にしていく
      end do;
      B[j]:=B[j]-B[i]*am;     #数値ベクトルも操作
    end do;
  end do:
```

上のコードによって得られた消去行列.

```
> T[1]; T[2];
```

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

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4/3 & 1 \end{bmatrix}$$

これを実際に元の行列 $A0$ に作用させると，UTM が求められる.

```
> U:=T[2].T[1].A0;
```

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

求められた LTM, UTM を掛けると

```
> L.U;
```

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

元の行列を得られる．L,A に求めたい行列が入っていることを確認.

```
> L;A;
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & 4/3 & 1 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 1 & -2 \\ 0 & -3 & 3 \\ 0 & 0 & -1 \end{bmatrix}$$

数値ベクトルも期待通り変換されている．

```
> B;
```

$$\begin{bmatrix} -4 \\ 9 \\ -2 \end{bmatrix}$$

4.5　ピボット操作

ガウス消去法で困るのは，割ろうとした対角要素が 0 の場合である．しかし，この場合にも，方程式の順序を，行列の行と右辺の値をペアにして入れ替えれば解決する．この割るほうの要素をピボット要素あるいはピボット (pivot, バスケの軸足を動かさずにくるくる回すやつ) と呼ぶ．この操作は，変数の並びを変えたわけではなく，単に方程式の順番を変更する操作に相当する．

さらに対角要素の数値が厳密に 0 でなくとも，極端に 0 に近づいた場合にも，その数で割った数値が大きくなり他の数との差を取ると以前に示した情報落ちの可能性が出てくる．この現象を防ぐためには，絶対値が最大のピボットを選んで行の入れ替えを毎回おこなうといい結果が得られることが知られている．

Maple の LUDecomposition コマンドをこのような行列に適用すると，置換行列 (permutation matrix)P が単位行列ではなく，ピボット操作に対応した行列となる．P.A=L.U となることに注意．

4.6　反復法による連立方程式の解

以下のような連立方程式を

$$\begin{bmatrix} 5\,x+y+z+u \\ x+3\,y+z+u \\ x-2\,y-9\,z+u \\ x+3\,y-2\,z+5\,u \end{bmatrix} = \begin{bmatrix} -6 \\ 2 \\ -7 \\ 3 \end{bmatrix}$$

形式的に解くと

$$x = \frac{-6-(y+z+u)}{5}$$

となる．他の未知数も，

y=
z=
u=

となる．適当に初期値 (x_0, y_0, z_0, u_0) をとり，下側の方程式に代入すると，得られた出力 (x_1, y_1, z_1, u_1) はより正解に近い値となる．これを繰り返すことによって正解が得られる．これをヤコビ (Jacobi) 法と呼び，係数行列の対角要素が非対角要素にくらべて大きいときに適用できる．多くの現実の問題ではこの状況が成り立っている．

Gauss-Seidel 法は Jacobi 法の高速版である．n 番目の解の組が得られた後に一度に次の解の組に入れ替えるのではなく，得られた解を順次改良した解として使っていく．これにより，収束が早まる．以下にはヤコビ法のコードを示した．x1[i] の配列を変数に換えるだけで，Gauss-Seidel 法となる．

```
> AA:=Matrix([[5,1,1,1],[1,3,1,1],[1,-2,-9,1],[1,3,-2,5]]):
  b:=Vector([-6,2,-7,3]): n:=4;
  x0:=[0,0,0,0]: x1:=[0,0,0,0]:
  for iter from 1 to 20 do
    for i from 1 to n do
      x1[i]:=b[i];
      for j from 1 to n do
        x1[i]:=x1[i]-AA[i,j]*x0[j];
      end do:
      x1[i]:=x1[i]+AA[i,i]*x0[i];
      x1[i]:=x1[i]/AA[i,i];
    end do:
    x0:=evalf(x1);
    print(iter,x0);
  end do:
```

```
                                4
      1, [-1.200000000, 0.6666666667, 0.7777777778, 0.6000000000]
      2, [-1.608888889, 0.6074074073, 0.5629629630, 0.7511111112]
      3, [-1.584296296, 0.7649382717, 0.5474897119, 0.7825185186]
      4, [-1.618989300, 0.7514293553, 0.5187050756, 0.6768921810]
      5, [-1.589405322, 0.8077973477, 0.5061160189, 0.6804222770]
      6, [-1.598867129, 0.8009556753, 0.4972691400, 0.6356490634]
      7, [-1.586774776, 0.8219829753, 0.4927633981, 0.6381076766]
      8, [-1.590570810, 0.8186345670, 0.4897074389, 0.6212705292]
      9, [-1.585922507, 0.8265309473, 0.4881589539, 0.6228163974]
     10, [-1.587501260, 0.8249823853, 0.4870924439, 0.6165295146]
     11, [-1.585720869, 0.8279597673, 0.4865626093, 0.6173477984]
     12, [-1.586374035, 0.8272701537, 0.4861897104, 0.6149933572]
     13, [-1.585690644, 0.8283969890, 0.4860087794, 0.6153885990]
     14, [-1.585958873, 0.8280977553, 0.4858782197, 0.6145034472]
     15, [-1.585695884, 0.8285257353, 0.4858165626, 0.6146844092]
     16, [-1.585805341, 0.8283983040, 0.4857707838, 0.6143503606]
     17, [-1.585703890, 0.8285613990, 0.4857498236, 0.6144303994]
     18, [-1.585748324, 0.8285078890, 0.4857337457, 0.6143038680]
     19, [-1.585709101, 0.8285702367, 0.4857266407, 0.6143384296]
     20, [-1.585727061, 0.8285480103, 0.4857209840, 0.6142903344]
```

4.7　課題

1. 後退代入法で解を求めよ．(2005 年度期末類題)

$$\begin{aligned} x + 4y - 3z &= 1 \\ -6y + 4z &= 1 \\ -\tfrac{5}{3}z &= \tfrac{1}{3} \end{aligned}$$

2. 次の行列 A を LU 分解せよ．

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

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

3. 次の連立方程式の係数行列を LU 分解し，上・下三角行列を求めよ．さらに連立方程式の解を求めよ.(2005 年度期末試験)

$$\begin{bmatrix} x_1 + 3\,x_2 + 4\,x_3 + 3\,x_4 \\ -2\,x_1 + 5\,x_2 + 3\,x_3 - 3\,x_4 \\ x_1 + 3\,x_2 - 2\,x_3 + 3\,x_4 \\ 3\,x_1 - 2\,x_2 + x_3 + 4\,x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 4 \\ -2 \\ 3 \end{bmatrix}$$

4. Jacobi 法のプログラムを参照して Gauss-Seidel 法のプログラムを作れ．Jacobi 法と収束性を比べよ．
5. 次の連立方程式の解を求めよ．ただし，pivot 操作が必要となる．

```
> with(LinearAlgebra):
  A:=Matrix([[3,2,2,1],[3,2,3,1],[1,-2,-3,1],[5,3,-2,5]]):
  X:=Vector([w,x,y,z]):
  b:=Vector([-6,2,-9,2]):
  A.X=b;
```

$$\begin{bmatrix} 3\,w + 2\,x + 2\,y + z \\ 3\,w + 2\,x + 3\,y + z \\ w - 2\,x - 3\,y + z \\ 5\,w + 3\,x - 2\,y + 5\,z \end{bmatrix} = \begin{bmatrix} -6 \\ 2 \\ -9 \\ 2 \end{bmatrix}$$

6. (おまけ) pivot 操作を含めた LU 分解のプログラムを作成せよ．上の問題を解き，その L, U 行列および $L^{-1}.b$ ベクトルを求めよ．

4.8　解答例

4. Jacobi 法のプログラムを参照して Gauss-Seidel 法のプログラムを作れ．Jacobi 法と収束性を比べよ．

```
#Gauss-Seidel
AA:=Matrix([[5,1,1,1],[1,3,1,1],[1,-2,-9,1],[1,3,-2,5]]):
b:=Vector([-6,2,-7,3]):
n:=4;
x0:=[0,0,0,0]:
x1:=[0,0,0,0]:
for iter from 1 to 20 do
for i from 1 to n do
  x1[i]:=b[i];
```

```
  for j from 1 to n do
    x1[i]:=x1[i]-AA[i,j]*x0[j];
  end do:
  x1[i]:=x1[i]+AA[i,i]*x0[i];
  x1[i]:=x1[i]/AA[i,i];
  x0:=evalf(x1);  #change here from ...
end do:
print(iter,x0);
end do:
```

```
                              4
   1, [-1.200000000, 1.066666667, 0.4074074073, 0.3629629628]
  2, [-1.567407407, 0.9323456790, 0.4367626887, 0.5287791494]
  3, [-1.579577503, 0.8713452217, 0.4673901337, 0.5800644210]
  4, [-1.583759955, 0.8454351333, 0.4783815777, 0.6008435420]
  5, [-1.584932051, 0.8352356437, 0.4828266893, 0.6089756998]
  6, [-1.585407607, 0.8312017393, 0.4845738460, 0.6121900162]
  7, [-1.585593120, 0.8296097527, 0.4852641546, 0.6134584342]
  8, [-1.585666468, 0.8289812930, 0.4855365978, 0.6139591570]
  9, [-1.585695410, 0.8287332183, 0.4856441456, 0.6141568092]
  10, [-1.585706835, 0.8286352933, 0.4856865986, 0.6142348304]
  11, [-1.585711344, 0.8285966383, 0.4857033566, 0.6142656284]
  12, [-1.585713125, 0.8285813800, 0.4857099714, 0.6142777856]
  13, [-1.585713827, 0.8285753567, 0.4857125829, 0.6142825846]
  14, [-1.585714105, 0.8285729793, 0.4857136134, 0.6142844788]
  15, [-1.585714214, 0.8285720407, 0.4857140204, 0.6142852266]
  16, [-1.585714258, 0.8285716703, 0.4857141809, 0.6142855218]
  17, [-1.585714275, 0.8285715240, 0.4857142443, 0.6142856384]
  18, [-1.585714281, 0.8285714660, 0.4857142694, 0.6142856844]
  19, [-1.585714284, 0.8285714433, 0.4857142792, 0.6142857024]
  20, [-1.585714285, 0.8285714343, 0.4857142831, 0.6142857096]
```

第 5 章

線形代数–固有値 (LAEigen)

5.1 固有値

A を対称正方行列，x をベクトルとしたときに，

$$Ax = \lambda x \tag{5.1}$$

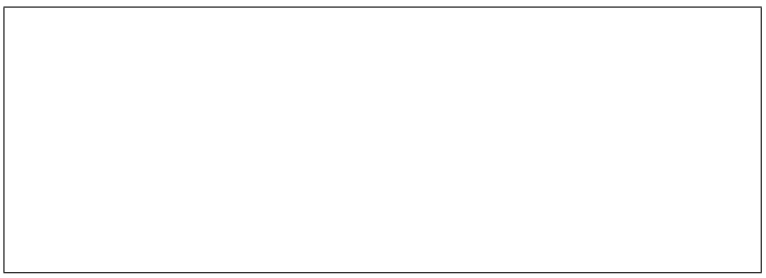
の解，λ を固有値，x を固有ベクトルという．x がゼロベクトルではない意味のある解は特性方程式 $\det(A - \lambda E) = 0$ が成り立つときにのみ得られる．

まず Maple で特性方程式を解いてみる．

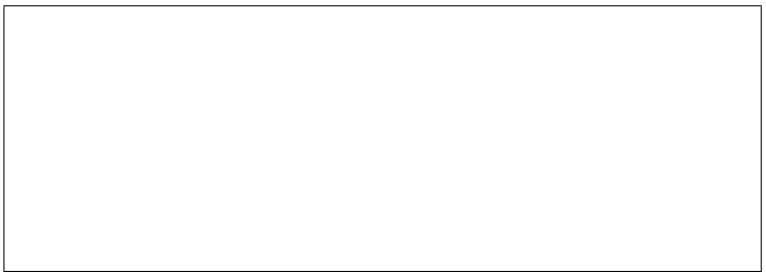
```
> restart;
> with(LinearAlgebra):with(plots):with(plottools):
> A:=Matrix(1..2,1..2,[[3,2/3],[2/3,2]]);
```

$$A := \begin{bmatrix} 3 & 2/3 \\ 2/3 & 2 \end{bmatrix}$$

```
> EE:=Matrix([[1,0],[0,1]]):
  A-lambda.EE;
```



```
> eq2:=Determinant(A-lambda.EE);
```



$$eq2 := \frac{50}{9} - 5\,\lambda + \lambda^2$$

```
> solve(eq2=0,lambda);
```

$$10/3,\ 5/3$$

固有値を求めるコマンド Eigenvectors を適用すると，固有値と固有ベクトルが求まる．ここで，固有ベクトルは行列の列 (Column) ベクトルに入っている．

```
> lambda,V:=Eigenvectors(A);
```

$$\lambda,\ V := \begin{bmatrix} 10/3 \\ 5/3 \end{bmatrix}, \begin{bmatrix} 2 & -1/2 \\ 1 & 1 \end{bmatrix}$$

得られた固有ベクトルは規格化されているわけではない．

行列の行を取り出すコマンド Column を用いて，方程式 (??) が成り立っていることを確認する．

```
> lambda[1].Column(V,1)=A.Column(V,1);
```

$$\begin{bmatrix} 20/3 \\ 10/3 \end{bmatrix} = \begin{bmatrix} 20/3 \\ 10/3 \end{bmatrix}$$

一般的な規格化は，コマンド Normalize(vector,Euclidean) によっておこなう．

```
> Normalize(Column(v,1),Euclidean);
```

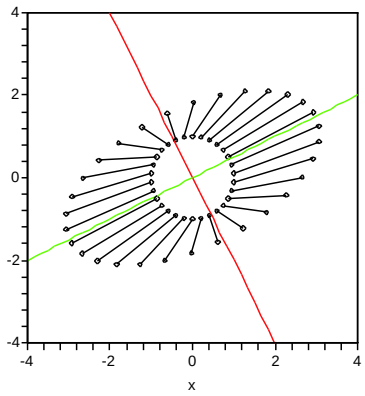
5.2　固有値の幾何学的意味

次に，固有値の幾何学的な意味を 2 次元行列で確認しておこう．ある点 x_0 に対称正方行列 A を作用すると，x_1 に移動する．これを原点を中心とする円上の点に次々に作用させ，移動前後の点を結ぶ．

```
> restart;
  with(LinearAlgebra):with(plots):with(plottools):
  A:=Matrix(1..2,1..2,[[3,2/3],[2/3,2]]):

> N:=30:p1:=[]:l1:=[]:
for k from 0 to N-1 do
  x0:=Vector([sin(2*Pi*k/N),cos(2*Pi*k/N)]);
  x1:=MatrixVectorMultiply(A,x0);
  p1:=[op(p1),pointplot({x0,x1})];
  l1:=[op(l1),line( evalf(convert(x0,list)),evalf(convert(x1,list)) )];
end do:

> n:=4;
  display(p1,l1,view=[-n..n,-n..n]);
```

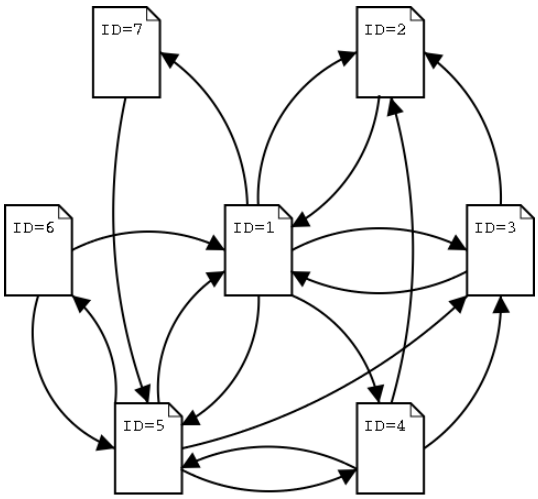


真ん中の円状の領域が，外側の楕円状の領域に写像されている様子が示されている．この図の中に固有ベクトル，固有値が隠れている．どこかわかる？

5.3 Google のページランク

多くの良質なページからリンクされているページはやはり良質なページである

Google の page rank は上のような非常に単純な仮定から成り立っている．ページランクを実際に求めよう．つぎのようなリンクが張られたページを考える．



計算手順は以下の通り[*1]．

1. リンクを再現する隣接行列を作る．ページに番号をつけて，その間が結ばれている i-j 要素を 1，そうでない要素を 0 とする．
2. 隣接行列を転置する
3. 列ベクトルの総和が 1 となるように規格化する．
4. こうして得られた推移確率行列の最大固有値に属する固有ベクトルを求め，適当に規格化する．

課題

1. 上記手順を参考にして，Maple でページランクを求めよ．
2. このような問題ではすべての固有値・固有ベクトルを求める必要はなく，最大の固有値を示す固有ベクトルを

[*1] 詳しくは http://www.kusastro.kyoto-u.ac.jp/~baba/wais/pagerank.html を参照せよ．

求めるだけでよい．初期ベクトルを適当に決めて，何度も推移確率行列を掛ける反復法でページランクを求めよ．

隣接行列

$$A1 := \left[\begin{array}{c|c|c|c|c|c|c|c} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & & & & & & & \\ 4 & & & & & & & \\ 5 & & & & & & & \\ 6 & & & & & & & \\ 7 & & & & & & & \end{array}\right]$$

転置行列

$$Transpose(A1) := \left[\begin{array}{c|c|c|c|c|c|c} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{array}\right]$$

規格化

$$\left[\begin{array}{c|c|c|c|c|c|c} & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \\ & & & & & & \end{array}\right]$$

遷移

$$\begin{pmatrix} 0 & 1 & 1/2 & 0 & 1/4 & 1/2 & 0 \\ 1/5 & 0 & 1/2 & 1/3 & 0 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/4 & 0 & 0 \\ 1/5 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1/7 \\ 1/7 \\ 1/7 \\ 1/7 \\ 1/7 \\ 1/7 \\ 1/7 \end{pmatrix} = \begin{pmatrix} \\ \\ \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} 0.32 \\ 0.15 \\ 0.11 \\ 0.06 \\ 0.29 \\ 0.04 \\ 0.03 \end{pmatrix}$$

5.4 累乗 (べき乗) 法により最大固有値が求まる原理

累乗 (べき乗) 法は，最大固有値とその固有ベクトルを効率的に見つける算法である．すこし，固有値について復習しておく．正方行列 A に対して，

$$Ax = \lambda x \tag{5.2}$$

の解 λ を固有値，x を固有ベクトルという．λ は，

$$\det(A - \lambda E) = 0 \tag{5.3}$$

として求まる永年方程式の解である．

では，なぜ適当な初期ベクトル x_0 から始めて，反復

$$x_{k+1} = Ax_k \tag{5.4}$$

を繰り返すと，A の絶対値最大の固有値に属する固有ベクトルに近づいていくのかを見ておこう．

すべての固有値がお互いに異なる場合を考える．今，行列の固有値を絶対値の大きなもの順に並べて，$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ とし，対応する長さを 1 に規格化した固有ベクトルを $x_1, x_2, \ldots, x_n$ とする．初期ベクトルは固有ベクトルの線形結合で表わせて，

$$X_0 = c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{5.5}$$

となるとする．これに行列 A を N 回掛けると，

$$A^N X_0 = c_1\lambda_1^N x_1 + c_2\lambda_2^N x_2 + \cdots + c_n\lambda_n^N x_n \tag{5.6}$$

となる．これを変形すると，

$$A^N X_0 = X_N = c_1\lambda_1^N \left\{ x_1 + \frac{c_2}{c_1}\left(\frac{\lambda_2}{\lambda_1}\right)^N x_2 + \cdots + \frac{c_n}{c_1}\left(\frac{\lambda_n}{\lambda_1}\right)^N x_n \right\} \tag{5.7}$$

となる．$|\lambda_1| > |\lambda_i| (i \geq 2)$ だから括弧の中は x_1 だけが生き残る．

こうして最大固有値に属する固有ベクトルが，反復計算を繰り返すだけで求められる．

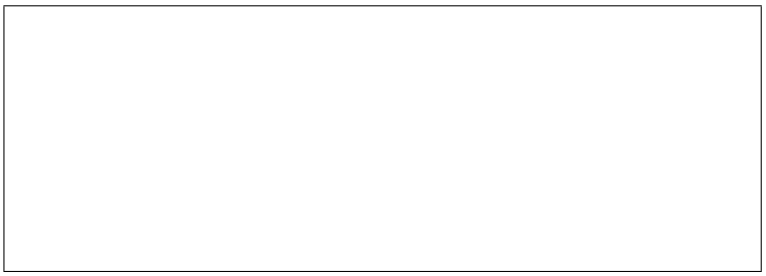
5.5 Jacobi 回転による固有値の求め方

固有値を求める手法として，永年方程式を解くというやり方は回りくどすぎる．少し古めかしいが非対角要素を 0 にする回転行列を反復的に作用させる Jacobi(ヤコビ) 法を紹介する．現在認められている最適の方策は，ハウスホルダー (Householder) 変換で行列を単純な三重対角化行列に変形してから，反復法で解を追い込んでいくやり方である．Jacobi 法は，Householder 法ほど万能ではないが，10 次程度までの行列には今でも役に立つ．

5.5.1 Maple でみる回転行列

行列の軸回転の復習をする．対称行列 B に回転行列 U を作用すると

$$B.U = \begin{pmatrix} a_{1\,1} & a_{1\,2} \\ a_{2\,1}(= a_{1\,2}) & a_{2\,2} \end{pmatrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \tag{5.8}$$



となる．回転行列を 4x4 の行列に

$$U^t B U \tag{5.9}$$

と作用させたときの各要素の様子を以下に示した．

```
> restart:
> n:=4:
> with(LinearAlgebra):
> B:=Matrix(n,n,shape=symmetric,symbol=a);
```

$$B := \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{1,2} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{1,3} & a_{2,3} & a_{3,3} & a_{3,4} \\ a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} \end{bmatrix}$$

```
> U:=Matrix(n,n,[[c,-s,0,0],[s,c,0,0],[0,0,1,0],[0,0,0,1]]);
#U:=Matrix(n,n,[[c,-s],[s,c]]);
```

$$U := \begin{bmatrix} c & -s & 0 & 0 \\ s & c & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
>TT:=Transpose(U).B.U;
```

$$TT := \tag{5.10}$$
$$\begin{bmatrix} (c\,a_{1,1} + s\,a_{1,2})\,c + (c\,a_{1,2} + s\,a_{2,2})\,s\,, & -(c\,a_{1,1} + s\,a_{1,2})\,s + (c\,a_{1,2} + s\,a_{2,2})\,c\,, \\ c\,a_{1,3} + s\,a_{2,3}\,, & c\,a_{1,4} + s\,a_{2,4} \end{bmatrix}$$
$$\begin{bmatrix} (-s\,a_{1,1} + c\,a_{1,2})\,c + (-s\,a_{1,2} + c\,a_{2,2})\,s\,, & -(-s\,a_{1,1} + c\,a_{1,2})\,s + (-s\,a_{1,2} + c\,a_{2,2})\,c\,, \\ -s\,a_{1,3} + c\,a_{2,3}\,, & -s\,a_{1,4} + c\,a_{2,4} \end{bmatrix}$$
$$[c\,a_{1,3} + s\,a_{2,3}\,,\ -s\,a_{1,3} + c\,a_{2,3}\,,\ a_{3,3}\,,\ a_{3,4}]$$
$$[c\,a_{1,4} + s\,a_{2,4}\,,\ -s\,a_{1,4} + c\,a_{2,4}\,,\ a_{3,4}\,,\ a_{4,4}]$$

```
>expand(TT[1,1]);
expand(TT[2,2]);
expand(TT[1,2]);
expand(TT[2,1]);
```

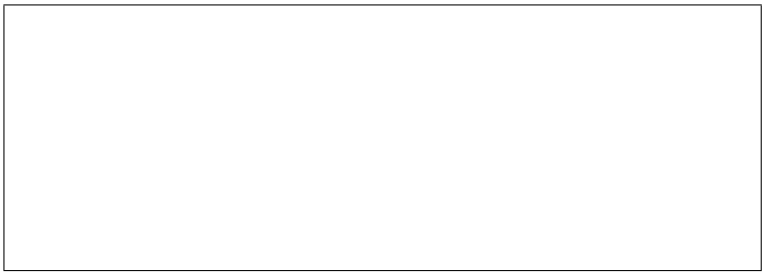
$$c^2\,a_{1,1} + 2\,c\,s\,a_{1,2} + s^2\,a_{2,2}$$

$$s^2\,a_{1,1} - 2\,c\,s\,a_{1,2} + c^2\,a_{2,2}$$

$$-s\,c\,a_{1,1} - s^2\,a_{1,2} + c^2\,a_{1,2} + c\,s\,a_{2,2}$$

$$-s\,c\,a_{1,1} - s^2\,a_{1,2} + c^2\,a_{1,2} + c\,s\,a_{2,2}$$

この非対角要素を 0 にする θ は以下のように求まる.



このとき注目している $i, j = 1, 2$ 以外の要素も変化する.

```
>expand(TT[3,1]);
expand(TT[3,2]);
```

$$c\,a_{1,3} + s\,a_{2,3}$$

$$-s\,a_{1,3} + c\,a_{2,3}$$

これによって一旦 0 になった要素も値を持つが，なんども繰り返すことによって，徐々に 0 へ近づいていく.

5.6 Jacobi 法による固有値を求める C コード

以下にはヤコビ法を用いた固有値と固有ベクトルを求めるコードを示した．結果は，固有値とそれに対応する規格化された固有ベクトルが縦 (column) ベクトルで表示される.

リスト: ヤコビ法.

```
#include <stdio.h>
#include <math.h>

#define M 10
void PrintMatrix(double a[M][M], int n);

int main(void){
  double a[M][M],v[M][M];
  double eps=0.0001,div,r,t,s,c,apj,aqj,aip,aiq,vip,viq;
  int i,j,n,iter,count,iterMax=1000000,p,q;

  scanf("%d",&n);
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++) scanf("%lf",&a[i][j]);
  }
  PrintMatrix(a,n);
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++) v[i][j]=0.;
    v[i][i]=1.;
  }

  for(iter=1;iter<=iterMax;iter++){
```

```
    count=0;
    for(p=1;p<=n-1;p++){
      for(q=p+1;q<=n;q++){
if(fabs(a[p][q])<eps) continue;
count++;
div=a[p][p]-a[q][q];
if (div != 0.0){
  r=2.0*a[p][q]/div;
  t=0.5*atan(r);
} else {
  t=0.78539818;
}
s=sin(t);
c=cos(t);
for(j=1;j<=n;j++){
  apj=a[p][j];
  aqj=a[q][j];
  a[p][j]=apj*c+aqj*s;
  a[q][j]=-apj*s+aqj*c;
}
for(i=1;i<=n;i++){
  aip=a[i][p];
  aiq=a[i][q];
  a[i][p]=aip*c+aiq*s;
  a[i][q]=-aip*s+aiq*c;
  vip=v[i][p];
  viq=v[i][q];
  v[i][p]=vip*c+viq*s;
  v[i][q]=-vip*s+viq*c;
}
printf("p,q=%3d,%3d\n",p,q);
PrintMatrix(a,n);
      }
    }
    if (count==0) break;
  }
  printf("Eigen values:\n");
  for(i=1;i<=n;i++) printf("%6.2f",a[i][i]);
  printf("\nEigen vectors:\n");
  PrintMatrix(v,n);

  return 0;
}

void PrintMatrix(double a[M][M], int n){
  int i,j;
```

```
  for(i=1;i<=n;i++){
    for(j=1;j<=n;j++) printf("%6.2f",a[i][j]);
    printf("\n");
  }
  printf("\n");
}
```

リスト: ヤコビ法の計算結果.

```
[BobsNewPBG4:~/NumRecipe/chap8] bob% cat input.txt
4
5 4 1 1
4 5 1 1
1 1 4 2
1 1 2 4

BobsNewPBG4:~/NumRecipe/chap8] bob% Jacobi2<input.txt
  5.00  4.00  1.00  1.00
  4.00  5.00  1.00  1.00
  1.00  1.00  4.00  2.00
  1.00  1.00  2.00  4.00

p,q=  1,  2
  9.00 -0.00  1.41  1.41
 -0.00  1.00 -0.00 -0.00
  1.41 -0.00  4.00  2.00
  1.41 -0.00  2.00  4.00

p,q=  1,  3
  9.37 -0.00 -0.00  1.88
 -0.00  1.00  0.00 -0.00
 -0.00  0.00  3.63  1.57
  1.88 -0.00  1.57  4.00

p,q=  1,  4
  9.96 -0.00  0.47 -0.00
 -0.00  1.00  0.00  0.00
  0.47  0.00  3.63  1.50
  0.00  0.00  1.50  3.41

...<中略>...

Eigen values:
 10.00  1.00  5.00  2.00
Eigen vectors:
  0.63 -0.71 -0.32  0.00
  0.63  0.71 -0.32  0.00
```

```
  0.32   0.00   0.63 -0.71
  0.32   0.00   0.63  0.71
```

5.7　数値計算ライブラリーについて

一般の数値計算ライブラリーについては，時間の関係で講義ではその能力を紹介するにとどめる．昔の演習で詳しく取り上げていたので，研究や今後のために必要と思うときは，テキストを取りにおいで．

行列の計算は，数値計算の中でも特に利用する機会が多く，また，律速ルーチンとなる可能性が高い．そこで，古くから行列計算の高速ルーチンが開発されてきた．なかでも BLAS と LAPACK はフリーながら非常に高速である．

前回に示した，逆行列を求める単純な LU 分解法を C 言語でコーディングしたものと，LAPACK のルーチンを比べた場合，1000 次元の行列で計測すると

```
>  1000 [dim]      2.5200 [sec] #BOB
>  1000 [dim]      0.4700 [sec] #LAPACK
```

となった．用いた PC は MacBook(2GHz Interl Core Duo) であるが，この計算での 0.47 秒は 1.4GFLOP に相当する．07 年の MacBook(2GHz Interl Core 2 Duo) ではさらに早くなって

```
bob% gcc -O3 bob.c -o bob
bob% ./bob
1000
 1000 [dim]      1.7543 [sec] #BOB
bob% gcc -O3 lapack.c -llapack -lblas -o lapack
bob% ./lapack
1000
 1000 [dim]      0.1893 [sec] #LAPACK
```

で，3.5GFLOPS が出ている．

ライブラリーは世界中の計算機屋さんがよってたかって検証しているので，バグがほとんど無く，また，高速である．初学者はライブラリーを使うべきである．ただし，下のサンプルプログラムの行列生成の違いのように，ブラックボックス化すると思わぬ間違い（ここでは Fortran と C での行列の並び順の違いが原因) をしでかすことがあるので，プログラムに組み込む前に必ず小さい次元 (サンプルコード) で検証しておくこと．

添付のコードはちょっと長いが時間があればフォローせよ．コンパイルは，OSX では

```
> gcc -O3 -UPRINT lapack.c -llapack -lblas
```

とすればできる．linux では LAPACK, BLAS がインストールされていれば，

```
> #include <vecLib/vecLib.h>
```

をコメントアウトして，

```
> gcc -O3 -DPRINT lapack.c -L/usr/local/lib64 -llapack -lblas -lg2c
```

などとすればコンパイルできるはず．

■リスト: **西谷製** lazy **逆行列計算プログラム**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include <time.h>

//#undef PRINT
//#define PRINT

void printMatrix(double *a, double *b, long n);
int MatrixInverse(double *a, double *b, long n);

int main(void){
  clock_t start, end;
  int i,j;
  long n;
  double *a,*b;

  scanf("%ld",&n);

  a=(double *)malloc(n*n*sizeof(double));
  b=(double *)malloc(n*sizeof(double));

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      a[i*n+j]= 2*(double) random() / RAND_MAX - 1.0;
    }
  }
  for (i=0;i<n;i++){
    b[i]= 2*(double) random() / RAND_MAX - 1.0;
  }
  printMatrix(a,b,n);

  start = clock();
  MatrixInverse(a,b,n);
  end = clock();
  printf("%5d [dim] %10.4f [sec] #BOB\n",
 n,(double)(end-start)/CLOCKS_PER_SEC);
  printMatrix(a,b,n);

  free(a);
  free(b);
  return 0;
}

int MatrixInverse(double *a, double *b, long n){
  double *x;
  double pvt=0.00005,am;
  int i,j,k;
```

```
  x=(double *)malloc(n*sizeof(double));

  for(i=0;i<n-1;i++){
    if(fabs(a[i*n+i])<pvt){
      printf("Pivot %3d=%10.5f is too small.\n",i,a[i*n+i]);
      return 1;
    }
    for(j=i+1;j<n;j++){
      am=a[j*n+i]/a[i*n+i];
      for(k=0;k<n;k++) a[j*n+k]-=am*a[i*n+k];
      b[j]-=am*b[i];
    }
  }
  //Backward substitution
  for(j=n-1;j>=0;j--){
    x[j]=b[j];
    for(k=j+1;k<n;k++){
      x[j]-=a[j*n+k]*x[k];
    }
    b[j]=x[j]/=a[j*n+j];
  }
  free(x);
  return 0;
}

void printMatrix(double *a, double *b, long n){
  int i,j;
#ifdef PRINT
  printf("\n");
  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      printf("%10.5f",a[i*n+j]);
    }
    printf(":%10.5f",b[i]);
    printf("\n");
  }
  printf("\n");
#endif
  return;
}
```

■リスト：LAPACK **謹製** smart **逆行列計算プログラム**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
```

```
#include <time.h>

//#define PRINT
//#undef PRINT

void printMatrix(double *a, double *b, long n);

int main(void){
  clock_t start, end;
  int i,j;
  double *a,*b;
  long n,nrhs=1, lda,ldb, info, *ipiv;

  scanf("%ld",&n);

  a=(double *)malloc(n*n*sizeof(double));
  b=(double *)malloc(n*sizeof(double));
  lda=ldb=n;
  ipiv=(long *)malloc(n*sizeof(long));

  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      a[j*n+i]= 2*(double) random() / RAND_MAX - 1.0;
    }
  }

  for (i=0;i<n;i++){
    b[i]= 2*(double) random() / RAND_MAX - 1.0;
  }
  printMatrix(a,b,n);

  start = clock();
  dgesv_(&n, &nrhs, a, &lda, ipiv, b, &ldb, &info);
  end = clock();
  printf("%5d [dim] %10.4f [sec] #LAPACK\n",
 n, (double)(end-start)/CLOCKS_PER_SEC);
  printMatrix(a,b,n);

  free(a);
  free(b);
  free(ipiv);

  return 0;
}

void printMatrix(double *a, double *b, long n){
```

```
  int i,j;
#ifdef PRINT
  printf("\n");
  for (i=0;i<n;i++){
    for (j=0;j<n;j++){
      printf("%10.5f",a[i*n+j]);
    }
    printf(":%10.5f",b[i]);
    printf("\n");
  }
  printf("\n");
#endif
  return;
}
```

5.8 課題

1. 4x4 の行列を適当に作り，Maple で固有値を求めよ．求め方はマニュアルを参照せよ．
2. Jacobi 法によって固有値を求めよ．
3. LAPACK に含まれている dsyev 関数を用いて実対称行列の固有値を求めよ．(演習で詳しく取り上げている．研究や今後のために必要と思うときは，テキストを取りにおいで)

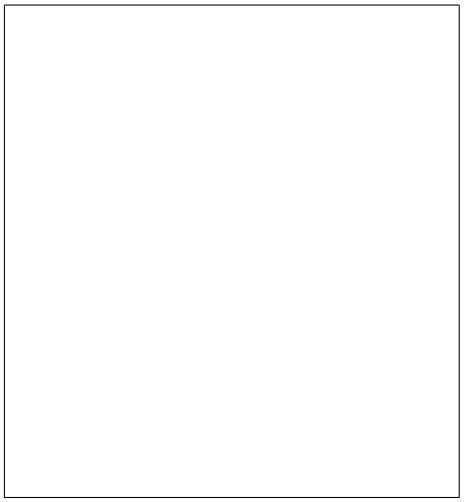
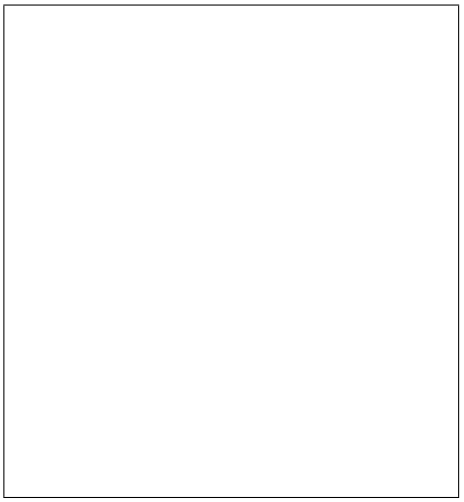
第 6 章

補間 (interpolation) と数値積分 (Integral)

6.1 概要:補間と近似

単純な 2 次元データについて補間と近似を考える．補間はたんに点をつなぐことを，近似はある関数にできるだけ近くなるようにフィットすることを言う．補間は Illustrator などのドロー系ツールで曲線を引くときの，ベジエやスプライン補間の基本となる．本章では補間とそれに密接に関連した積分について述べる．

表 6.1　補間と近似の模式図.

補間	近似
	

6.2 多項式補間 (polynomial interpolation)

データを単純に多項式で補間する方法を先ず示そう．$N+1$ 点を N 次の多項式でつなぐ．この場合の補間関数は,

$$F(x) = \sum_{i=0}^{N} a_i x^i = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N$$

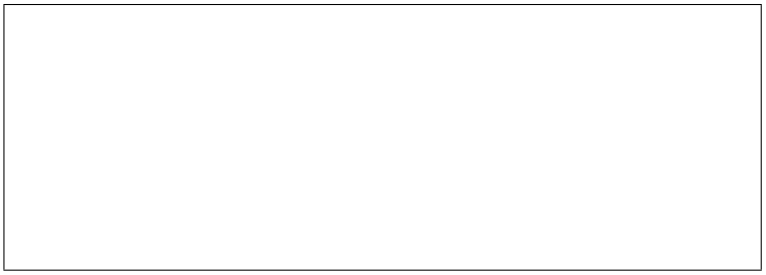
である．データの点を $(x_i, y_i), i = 0..N$ とすると

$$\begin{aligned} a_0 + a_1 x_0 + a_2 x_0^2 + \cdots + a_N x_0^N &= y_0 \\ a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_N x_1^N &= y_1 \\ &\vdots \\ a_0 + a_1 x_N + a_2 x_N^2 + \cdots + a_N x_N^N &= y_N \end{aligned}$$

が，係数 a_i を未知数と見なした線形の連立方程式となっている．係数行列は

$$A = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^N \\ 1 & x_1 & x_1^2 & \cdots & x_1^N \\ \vdots & & & & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^N \end{bmatrix}$$

となる．a_i と y_i をそれぞれベクトルとみなすと



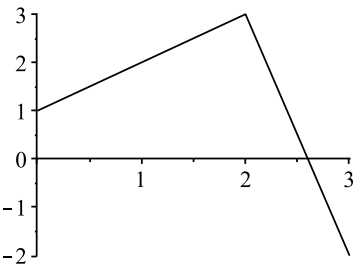
により未知数ベクトル a_i が求まる．これは単純に，前に紹介した Gauss の消去法や LU 分解で解ける．

6.2.1　Maple による多項式補間の実例

```
> restart; X:=[0,1,2,3]: Y:=[1,2,3,-2]:
> with(LinearAlgebra):
> list1:=[X,Y];
```

$$list1 := [[0,1,2,3],[1,2,3,-2]]$$

```
> with(plots):
  l1p:=listplot(Transpose(Matrix(list1))):
  display(l1p);
```




```
> A:=Matrix(4,4):
  for i from 1 to 4 do
    for j from 1 to 4 do
      A[i,j]:=X[i]^(j-1);
    end do;
  end do:
  A;
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix}$$

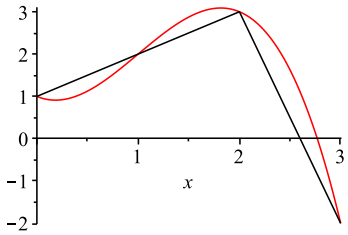
```
> a1:=MatrixInverse(A).Vector(Y);
```

$$a1 := \begin{bmatrix} 1 \\ -1 \\ 3 \\ -1 \end{bmatrix}$$

```
> f1:=unapply(add(a1[ii]*x^(ii-1),ii=1..4),x);
```

$$f1 := x \mapsto 1 - x + 3x^2 - x^3$$

```
> f1p:=plot(f1(x),x=0..3):
  l1p:=listplot(Transpose(Matrix(list1))):
  display(f1p,l1p);
```



6.3 Lagrange(ラグランジュ) の内挿公式

多項式補間は手続きが簡単であるため，計算間違いが少なく，プログラムとして組むのに適している．しかし，あまり”みとうし”のよい方法とはいえない．その点，Lagrange(ラグランジュ) の内挿公式は見通しがよい．これは

$$F(x) = \sum_{k=0}^{N} \frac{\prod_{j\neq k}(x - x_j)}{\prod_{j\neq k}(x_k - x_j)} y_k = \sum_{k=0}^{N} \frac{\frac{(x-x_0)(x-x_1)\cdots(x-x_N)}{(x-x_k)}}{\frac{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_N)}{(x_k-x_k)}} y_k$$

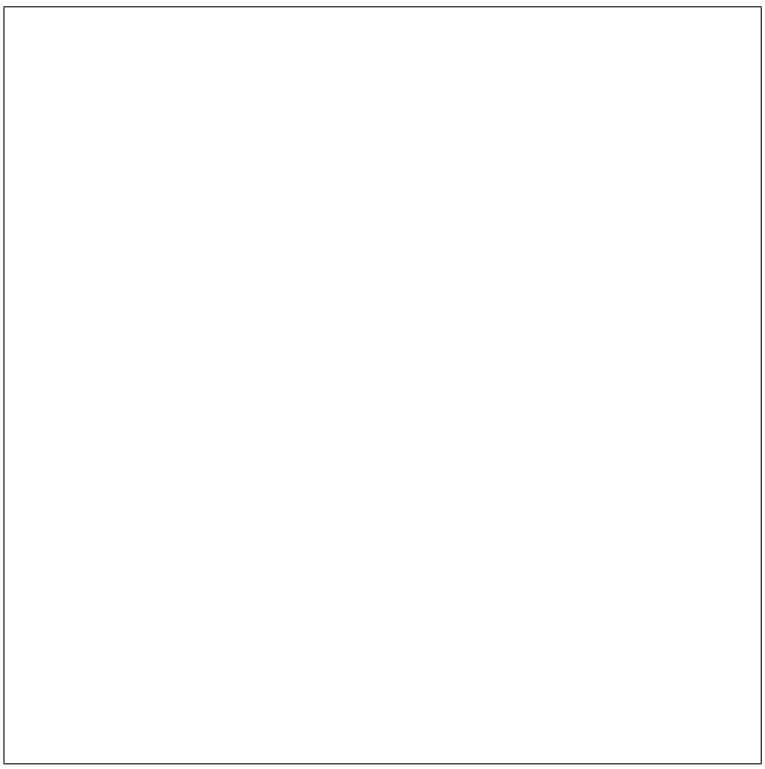
と表わされる．数学的に 2 つ目の表記は間違っているが，先に割り算を実行すると読み取って欲しい．これは一見複雑に見えるが，単純な発想から出発している．求めたい関数 $F(x)$ を

$$F(x) = y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x)$$

とすると

$$\begin{array}{lll} L_0(x_0) = 1 & L_0(x_1) = 0 & L_0(x_2) = 0 \\ L_1(x_0) = 0 & L_1(x_1) = 1 & L_1(x_2) = 0 \\ L_2(x_0) = 0 & L_2(x_1) = 0 & L_2(x_2) = 1 \end{array}$$

となるように関数 $L_i(x)$ を決めればよい．これを以下のようにとれば Lagrange の内挿公式となる．



6.4　Newton(ニュートン) の差分商公式

もう一つ有名な Newton(ニュートン) の内挿公式は,

$$\begin{aligned} F(x) \quad &= F(x_0) + (x-x_0) f_1\lfloor x_0, x_1 \rfloor + (x-x_0)(x-x_1) f_2\lfloor x_0, x_1, x_2 \rfloor + \\ &\quad \cdots + \prod_{i=0}^{n-1} (x-x_i)\, f_n\lfloor x_0, x_1, \cdots, x_n \rfloor \end{aligned}$$

となる．ここで $f_i\lfloor\ \rfloor$ は次のような関数を意味していて,

$$\begin{aligned} f_1\lfloor x_0, x_1 \rfloor &= \frac{y_1 - y_0}{x_1 - x_0} \\ f_2\lfloor x_0, x_1, x_2 \rfloor &= \frac{f_1\lfloor x_1, x_2 \rfloor - f_1\lfloor x_0, x_1 \rfloor}{x_2 - x_0} \\ &\vdots \\ f_n\lfloor x_0, x_1, \cdots, x_n \rfloor &= \frac{f_{n-1}\lfloor x_1, x_2 \cdots, x_n \rfloor - f_{n-1}\lfloor x_0, x_1, \cdots, x_{n-1} \rfloor}{x_n - x_0} \end{aligned}$$

差分商と呼ばれる．得られた多項式は，Lagrange の内挿公式で得られたものと当然一致する．Newton の内挿公式の利点は，新たなデータ点が増えたときに，新たな項を加えるだけで，内挿式が得られる点である．

6.4.1　Newton 補間と多項式補間の一致の検証

関数 $F(x)$ を x の多項式として展開．その時の，係数の取るべき値と，差分商で得られる値が一致.

```
> restart: F:=x->f0+(x-x0)*f1p+(x-x0)*(x-x1)*f2p;
```

$$F := x \mapsto f0 + (x - x0) f1p + (x - x0)(x - x1) f2p$$

```
> F(x1);
  sf1p:=solve(F(x1)=f1,f1p);
```

$$f0 + (x1 - x0)f1p$$
$$sf1p := \frac{f0 - f1}{-x1 + x0}$$

f20 の取るべき値の導出

```
> sf2p:=solve(F(x2)=f2,f2p);
  fac_f2p:=factor(subs(f1p=sf1p,sf2p));
```

$$sf2p := -\frac{f0 + f1p\,x2 - f1p\,x0 - f2}{(-x2 + x0)(-x2 + x1)}$$
$$fac\_f2p := \frac{f0\,x1 - x2\,f0 + x2\,f1 - x0\,f1 - f2\,x1 + f2\,x0}{(-x1 + x0)(-x2 + x0)(-x2 + x1)}$$

ニュートンの差分商公式を変形

```
> ff11:=(f0-f1)/(x0-x1);
  ff12:=(f1-f2)/(x1-x2);
  ff2:=(ff11-ff12)/(x0-x2);
  fac_newton:=factor(ff2);
```

$$ff11 := \frac{f0 - f1}{-x1 + x0}$$
$$ff12 := \frac{f1 - f2}{-x2 + x1}$$
$$ff2 := \frac{\frac{f0 - f1}{-x1 + x0} - \frac{f1 - f2}{-x2 + x1}}{-x2 + x0}$$
$$fac\_newton := \frac{f0\,x1 - x2\,f0 + x2\,f1 - x0\,f1 - f2\,x1 + f2\,x0}{(-x1 + x0)(-x2 + x0)(-x2 + x1)}$$

二式が等しいかどうかを evalb で判定

```
> evalb(fac_f2p=fac_newton);
```

$$true$$

6.5 数値積分 (Numerical integration)

積分,

$$I = \int_a^b f(x)dx$$

を求めよう．1 次元の数値積分法では連続した領域を細かい短冊に分けて，それぞれの面積を寄せ集めることに相当する．分点の数を N とすると,

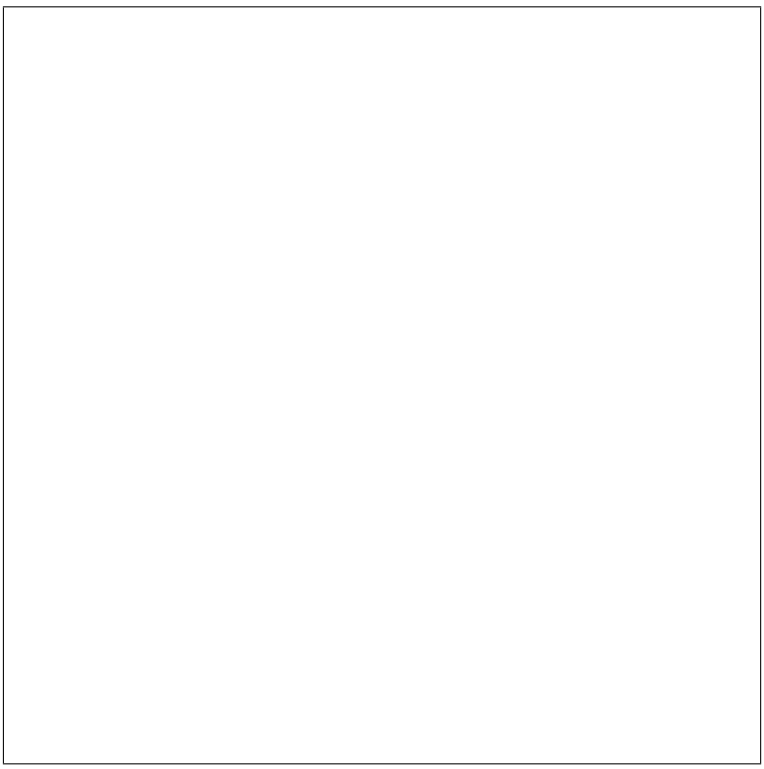
$$x_i = a + \frac{b - a}{N} i = a + h \times i$$
$$h = \frac{b - a}{N}$$

ととれる．そうすると，もっとも単純には,

$$I_N = \left\{ \sum_{i=0}^{N-1} f(x_i) \right\} h = \left\{ \sum_{i=0}^{N-1} f(a + i \times h) \right\} h$$

となる．

表 6.2　数値積分の模式図.



6.5.1　中点則 (midpoint rule)

中点法 (midpoint rule) は，短冊を左端から書くのではなく，真ん中から書くことに対応し，

$$I_N = \left\{ \sum_{i=0}^{N-1} f\left(a + \left(i + \frac{1}{2}\right) \times h\right) \right\} h$$

となる.

6.5.2　台形則 (trapezoidal rule)

さらに短冊の上側を斜めにして，短冊を台形にすれば精度が上がりそうに思う．その場合は，短冊一枚の面積 S_i は,

$$S_i = \frac{f(x_i) + f(x_{i+1}}{2} h$$

で求まる．これを端から端まで加えあわせると,

$$i_N = \sum_{i=0}^{N-1} S_i = h \left\{ \frac{1}{2} f(x_0) + \sum_{i=1}^{N-1} f(x_i) + \frac{1}{2} f\left(x_N\right) \right\}$$

が得られる.

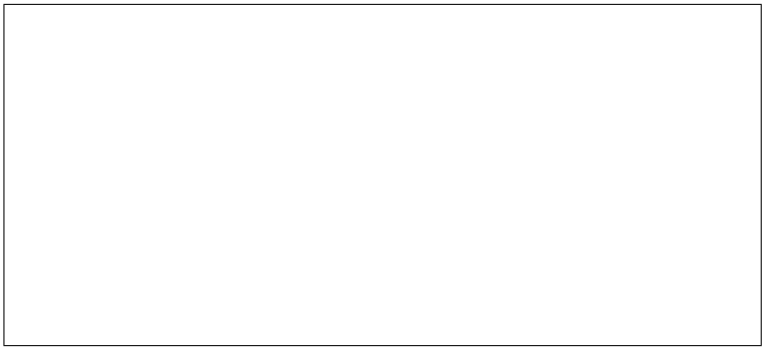
6.5.3　Simpson(シンプソン) 則

Simpson(シンプソン) 則では，短冊を 2 次関数,

$$f(x) = ax^2 + bx + c$$

で近似することに対応する．こうすると，

$$S_i = \int_{x_i}^{x_{i+1}} f(x)\,dx = \int_{x_i}^{x_{i+1}} (ax^2 + bx + c)\,dx$$



$$\frac{h}{6}\left\{f(x_i) + 4f\left(x_i + \frac{h}{2}\right) + f(x_i + h)\right\}$$

となる．これより，

$$I_N = \frac{h}{6}\left\{f(x_0) + 4\sum_{i=0}^{N-1} f\left(x_i + \frac{h}{2}\right) + 2\sum_{i=1}^{N-1} f(x_i) + f(x_N)\right\}$$

として計算できる．ただし，関数値を計算する点の数は台形則などの倍となっている．

教科書によっては，分割数 N を偶数にして，点を偶数番目 (even) と奇数番目 (odd) に分けて，

$$I_N = \frac{h}{3}\left\{f(x_0) + 4\sum_{i=even}^{N-2} f\left(x_i + \frac{h}{2}\right) + 2\sum_{i=odd}^{N-1} f(x_i) + f(x_N)\right\}$$

としている記述があるが，同じ計算になるので誤解せぬよう．

6.6　数値積分のコード

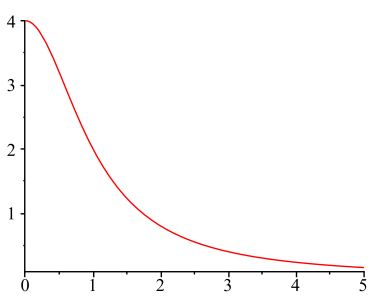
次の積分を例に，Maple のコードを示す．

$$\int_0^1 \frac{4}{1+x^2}\,dx$$

先ずは問題が与えられたらできるだけ Maple で解いてしまう．答えをあらかじめ知っておくと間違いを見つけるのが容易．プロットしてみる．

```
> restart;
  f1:=x->4/(1+x^2);
  plot(f1(x),x=0..5);
```

$$f1 := x \mapsto \frac{4}{1+x^2}$$



Maple で解いてみる.

```
>int(f1(x),x=0..1);
```

$$\pi$$

えっと思うかも知れないが,

```
>int(1/(1+x^2),x);
```

$$arctan(x)$$

となるので，納得できるでしょう.
具体的に Maple でコードを示す．先ずは初期設定.

```
>N:=8: x0:=0: xn:=1: Digits:=20:
```

■Midpoint rule(**中点法**)

```
> h:=(xn-x0)/N: S:=0:
  for i from 0 to N-1 do
    xi:=x0+(i+1/2)*h;
    dS:=h*f1(xi);
    S:=S+dS;
  end do:
  evalf(S);
```

3.1428947295916887799

■Trapezoidal rule(**台形公式**)

```
> h:=(xn-x0)/N: S:=f1(x0)/2:
  for i from 1 to N-1 do
    xi:=x0+i*h;
    dS:=f1(xi);
    S:=S+dS;
  end do:
  S:=S+f1(xn)/2:
  evalf(h*S);
```

3.1389884944910890093

■Simpson's rule(**シンプソンの公式**)

```
> M:=N/2: h:=(xn-x0)/(2*M): Seven:=0: Sodd:=0:
  for i from 1 to 2*M-1 by 2 do
    xi:=x0+i*h;
    Sodd:=Sodd+f1(xi);
  end do:
  for i from 2 to 2*M-1 by 2 do
```

```
  xi:=x0+i*h;
  Seven:=Seven+f1(xi);
end do:
evalf(h*(f1(x0)+4*Sodd+2*Seven+f1(xn))/3);
```

3.1415925024587069144

6.7 課題

1. 補間と近似の違いについて，適切な図を描いて説明せよ．
2. 次の 4 点

   x y
   0 1
   1 2
   2 3
   3 -2

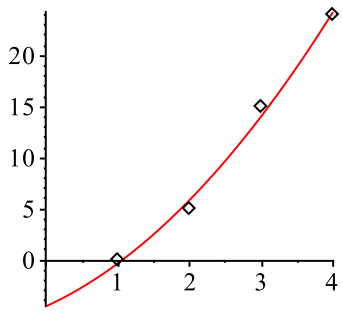
   を通る多項式を以下のそれぞれの手法で求めよ．(a) 逆行列, (b) ラグランジュ補間, (c) ニュートンの差分商公式
3. tan(5°)=0.08748866355, tan(10°)=.1763269807, tan(15°)=.2679491924 の値を用いて，ラグランジュ補間法により，tan(17°) の値を推定せよ．(2008 年度期末試験)
4. exp(0)=1.0, exp(0.1)=1.1052, exp(0.3)=1.3499 の値を用いて，ラグランジュ補間法により，exp(0.2) の値を推定せよ．(2009 年度期末試験)
5. 次の関数
$$f(x) = \frac{4}{1+x^2}$$
   を $x = 0..1$ で数値積分する．
   (a) N を 2,4,8,…256 ととり，N 個の等間隔な区間にわけて中点法で求めよ．(15)
   (b) 小数点以下 10 桁まで求めた値 3.141592654 との差を dX とする．dX と分割数 N とを両対数プロット (loglogplot) して比較せよ (10)
   (2008 年度期末試験)
6. 次の関数
$$y = \frac{1}{1+x^2}$$
   を $x = 0..1$ で等間隔に $N+1$ 点とり，N 個の区間にわけて数値積分で求める．N を 2, 4, 8, 16, 32, 64, 128, 256 と取ったときの (a) 中点法, (b) 台形公式, (c) シンプソン公式それぞれの収束性を比較せよ．
   ヒント：Maple script にあるそれぞれの数値積分法を関数 (procedure) に直して，for-loop で回せば楽．出来なければ，一つ一つ手で変えても OK. 両対数プロット (loglogplot) すると見やすい．

第 7 章

線形最小 2 乗法 (LeastSquareFit)

7.1 Maple による最小 2 乗法

前章では，データに多項式を完全にフィットする補間についてみた．今回は，近似的にフィットする最小二乗法について詳しくみていく．図のようなデータに直線をフィットする場合を考えよう．



コマンド leastsquare による fitting(2 変数の例)

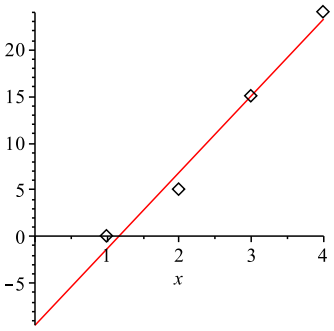
```
> restart: X:=[1,2,3,4]: Y:=[0,5,15,24]:
> with(plots):with(linalg):with(stats):
> l1:=pointplot(transpose([X,Y]),symbolsize=30):
> eq_fit:=fit[leastsquare[[x, y], y = a0+a1*x, {a0,a1}]]([X, Y]);
```

$$eq\_fit := y = -\frac{19}{2} + \frac{41}{5}x$$

```
> f1:=unapply(rhs(eq_fit),x);
```

$$f1 := x \mapsto -\frac{19}{2} + \frac{41}{5}x$$

```
> p1:=plot(f1(x),x=0..4):
> display(p1,l1);
```



7.2　最小 2 乗法の原理

もっとも簡単な例で原理を解説する．近似関数として，

$$F(x) = a_0 + a_1\, x$$

という直線近似を考える．もっともらしい関数は N 点の測定データとの差 $d_i = F(x_i) - y_i$ を最小にすればよさそうであるが，これはプラスマイナスですぐに消えて不定になる．そこで，

$$\chi^2 = \sum_i^N d_i^2 = \sum_i^N (a_0 + a_1\, x_i - y_i)^2$$

という関数を考える．この χ^2(カイ二乗) 関数が，a_0, a_1 をパラメータとして変えた時に最小となる a_0, a_1 を求める．これは，それらの微分がそれぞれ 0 となる場合である．これは χ^2 の和 $\sum$(sum) の中身を展開し，

$\chi^2 =$

a_0, a_1 でそれぞれ微分すれば

という a_0, a_1 を未知変数とする 2 元の連立方程式が得られる．これは前に説明した通り逆行列で解くことができる．

7.3　χ^2 の極小値から (2 変数の例)

```
> restart; X:=[1,2,3,4]: Y:=[0,5,15,24]: f1:=x->a0+a1*x:
  S:=0:
  for i from 1 to 4 do
```

$$\frac{\partial}{\partial a_0}\chi^2 =$$

$$\frac{\partial}{\partial a_1}\chi^2 =$$

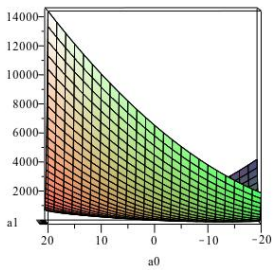
```
    S:=S+(f1(X[i])-Y[i])^2;
  end do:
> fS:=unapply(S,(a0,a1));
```

$$fS := (a0, a1) \mapsto (a0 + a1)^2 + (a0 + 2\,a1 - 5)^2 + (a0 + 3\,a1 - 15)^2 + (a0 + 4\,a1 - 24)^2$$

```
> expand(fS(a0,a1));
```

$$4\,a0^2 + 20\,a0\,a1 + 30\,a1^2 - 88\,a0 - 302\,a1 + 826$$

```
> plot3d(fS(a0,a1),a0=-20..20,a1=0..20);
```




```
> eqs:={diff(expand(S),a0)=0, diff(expand(S),a1)=0};
```

$$eqs := \{8\,a0 + 20\,a1 - 88 = 0, 20\,a0 + 60\,a1 - 302 = 0\}$$

```
> solve(eqs,{a0,a1});
```

$$\left\{a0 = -\frac{19}{2}, a1 = \frac{41}{5}\right\}$$

7.4　正規方程式 (Normal Equations) による解

より一般的な場合の最小二乗法の解法を説明する．先程の例では 1 次の多項式を近似関数とした．これをより一般的な関数，例えば，$\sin, \cos, \tan, \exp, \sinh$ などとする．これを線形につないだ関数を

$$F(x) = a_0 \sin(x) + a_1 \cos(x) + a_2 \exp(-x) + a_3 \sinh(x) + \cdots = \sum_{k=1}^{M} a_k X_k(x)$$

ととる．実際には，$X_k(x)$ はモデルや，多項式の高次項など論拠のある関数列をとる．これらを基底関数 (base functions) と呼ぶ．ここで線形といっているのは，パラメータ a_k について線形という意味である．このような，より一般的な基底関数を使っても，χ^2 関数は

$$\chi^2 = \sum_{i=1}^{N} (F(x_i) - y_i)^2 = \sum_{i=1}^{N} \left(\sum_{k=1}^{M} a_k X_k(x_i) - y_i \right)^2$$

と求めることができる．この関数を，a_k を変数とする関数とみなす．この関数が最小値を取るのは，χ^2 を M 個の a_k で偏微分した式がすべて 0 となる場合である．これを実際に求めてみると，

$$\sum_{i=1}^{N} \left(\sum_{j=1}^{M} a_j X_j(x_i) - y_i \right) X_k(x_i) = 0$$

となる．ここで，$k = 1..M$ の M 個の連立方程式である．この連立方程式を最小二乗法の正規方程式 (normal equations) と呼ぶ．

上記の記法のままでは，ややこしいので，行列形式で書き直す．$N \times M$ で，各要素を

$$A_{ij} = X_j(x_i)$$

とする行列 A を導入する．この行列は，

$$A = \begin{bmatrix} X_1(x_1) & X_2(x_1) & \cdots & X_M(x_1) \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ \vdots & \vdots & \cdots & \vdots \\ X_1(x_N) & X_2(x_N) & \cdots & X_M(x_N) \end{bmatrix}$$

となる．これをデザイン行列と呼ぶ．すると先程の正規方程式は，

$$A^t.A.a = A^t.y$$

で与えられる．A^t は行列 A の転置 (transpose)

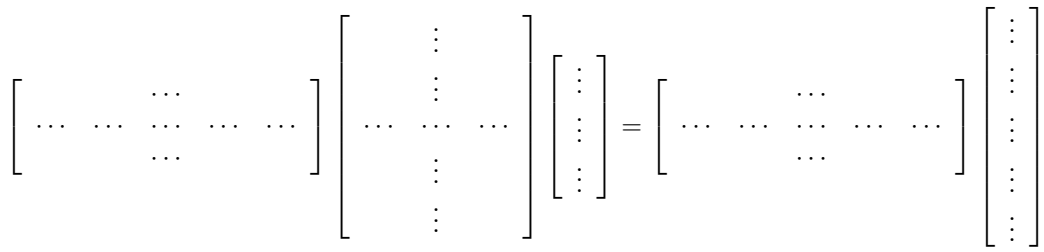
$$A^t = A^t_{ij} = A_{ji}$$

を意味し，得られた行列は，$M \times N$ である．a, y はそれぞれ，

$$a = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_M \end{bmatrix}, \; y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

である．

$M = 3, N = 25$ として行列の次元だけで表現すると，



となる．これは少しの計算で 3×3 の逆行列を解く問題に変形できる．

7.4.1 Maple による具体例

```
> restart; X:=[1,2,3,4]: Y:=[0,5,15,24]:
  f1:=x->a[1]+a[2]*x+a[3]*x^2:
  with(LinearAlgebra): Av:=Matrix(1..4,1..3):
  ff:=(x,i)->x^(i-1):
  for i from 1 to 3 do
    for j from 1 to 4 do
      Av[j,i]:=ff(X[j],i);
    end do;
  end do;
  Av;
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix}$$

```
> Ai:=MatrixInverse(Transpose(Av).Av);
```

$$Ai := \begin{bmatrix} \frac{31}{4} & -\frac{27}{4} & \frac{5}{4} \\ -\frac{27}{4} & \frac{129}{20} & -\frac{5}{4} \\ \frac{5}{4} & -\frac{5}{4} & \frac{1}{4} \end{bmatrix}$$

```
> b:=Transpose(Av).Vector(Y);
```

$$b := \begin{bmatrix} 44 \\ 151 \\ 539 \end{bmatrix}$$

```
> Ai.b;
```

$$\begin{bmatrix} -\frac{9}{2} \\ \frac{16}{5} \\ 1 \end{bmatrix}$$

7.5　特異値分解 (Singular Value Decomposition) による解

正規方程式を解くときには，少し注意が必要である．正規方程式での共分散行列, 特異値分解の導出や標準偏差との関係は NumRecipe を参照せよ．

```
> restart; X:=[1,2,3,4]: Y:=[0,5,15,24]: f1:=x->a[1]+a[2]*x+a[3]*x^2:
> with(LinearAlgebra): Av:=Matrix(1..4,1..3):
> ff:=(x,i)->x^(i-1):
  for i from 1 to 3 do
    for j from 1 to 4 do
      Av[j,i]:=ff(X[j],i);
    end do;
  end do;
  Av;
```

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix}$$

```
> U,S,Vt:=evalf(SingularValues(Av,output=['U','S','Vt'])):
> DiagonalMatrix(S[1..3],4,3); U.DiagonalMatrix(S[1..3],4,3).Vt:
```

$$\begin{bmatrix} 19.6213640200000015 & 0 & 0 \\ 0 & 1.71206987399999999 & 0 \\ 0 & 0 & 0.266252879300000022 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> iS:=Vector(3):
  for i from 1 to 3 do
    iS[i]:=1/S[i];
  end do:
> DiS:=DiagonalMatrix(iS[1..3],3,4);
```

$$DiS := \begin{bmatrix} 0.05096485642 & 0 & 0 & 0 \\ 0 & 0.5840883104 & 0 & 0 \\ 0 & 0 & 3.755827928 & 0 \end{bmatrix}$$

```
> Transpose(Vt).DiS.(Transpose(U).Vector(Y));
```

$$\begin{bmatrix} -4.50000000198176498 \\ 3.20000000035008324 \\ 1.00000000040565196 \end{bmatrix}$$

7.6　2 次元曲面へのフィット

先程の一般化をより発展させると，3 次元 (x_i, y_i, z_i) で提供されるデータへの，2 次元平面でのフィットも可能となる．2 次元の単純な曲面は，方程式を使って，

$$F(x, y) = a_1 + a_2\, x + a_3\, y + a_4\, xy + a_5\, x^2 + a_6\, y^2$$

となる．デザイン行列の i 行目の要素は，

$$[1, x_i, y_i, x_i\, y_i, x_i^2, y_i^2]$$

として，それぞれ求める．このデータの変換の様子を Maple スクリプトで詳しく示した．後は，通常の正規方程式を解くようにすれば，このデータを近似する曲面を定めるパラメータ $a_1, a_2, \cdots, a_6$ が求まる．最小二乗法はパラメータ a_k について線形であればよい．

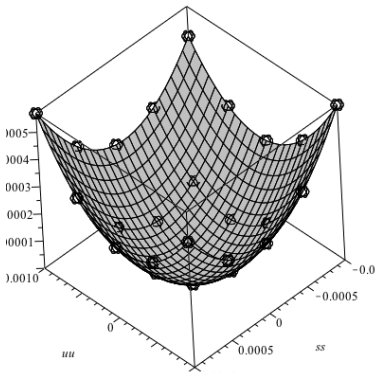
7.6.1　Maple による具体例

実際のデータ解析での例．データの座標を x,y,z で用意して，Maple の埋め込み関数の leastsquare で fit している．

```
> with(plots):with(plottools):
  z:=[0.000046079702088, 0.000029479057275,
  0.000025769637830, 0.000034951410953, 0.000057024385455, 0.000029485453808,
  0.000011519913869, 0.000006442404299, 0.000014252898382, 0.000034951410953,
  0.000025769637773, 0.000006442404242, 0.000000000000057, 0.000006442404242,
  0.000025769637773, 0.000034932221524, 0.000014246501905, 0.000006442404299,
  0.000011519913926, 0.000029479057332, 0.000056973214100, 0.000034932221467,
  0.000025769637773, 0.000029485453808, 0.000046079702031]:
>  x:=[]:
  y:=[]:
  p1:=2:
  for i from -p1 to p1 do
    for j from -p1 to p1 do
      x:=[op(x),i*0.0005];
      y:=[op(y),j*0.0005];
    end do;
  end do;
> with(LinearAlgebra): p2:=convert(Transpose(Matrix([x,y,z])),listlist):
  pp2:=pointplot3d(p2,symbol=circle,symbolsize=30,color=black):
  with(stats): data:=[x,y,z]:
  fit1:=fit[leastsquare[[t,s,u],
    u=a1+a2*t+a3*s+a4*t*s+a5*t^2+a6*s^2,
    {a1,a2,a3,a4,a5,a6}]](data);
```

$$fit1 := u = -8.657142857 \times 10^{-13} - 0.000006396456800\, t + 0.000006396438400\, s$$
$$-5.459553587\, ts + 25.76962838\, t^2 + 25.76962835\, s^2$$

```
> f1:=unapply(rhs(fit1),(s,t)):
> pf1:=plot3d(f1(ss,uu),ss=-0.001..0.001,uu=-0.001..0.001,color=gray):
> display(pf1,pp2,axes=boxed);
```



7.6.2　正規方程式による解法

デザイン行列へのデータ変換

```
> bb:=Vector(25): A:=Matrix(25,6):
  p1:=2:
  for i from 1 to 25 do
    A[i,1]:=1;
    A[i,2]:=x_i;
    A[i,3]:=y_i;
    A[i,4]:=x_i*y_i;
    A[i,5]:=x_i^2;
    A[i,6]:=y_i^2;
    bb_i:=z_i;
  end do:
```

正規方程式の解

```
> MatrixInverse(Transpose(A).A).(Transpose(A).bb);
```

$$\begin{bmatrix} -9.185257196 \times 10^{-13} \\ -0.00000639644675999994798 \\ 0.00000639644220000032532 \\ -5.45955358336000173 \\ 25.7696284050857187 \\ 25.7696284050857543 \end{bmatrix}$$

7.7　課題

1. 1 次元の線形最小二乗法
   次の 4 点のデータを $y = a_1 + a_2 x + a_3 x^2$ で近似せよ (2006 年度期末試験).

   ```
   X:=[0,1,2,3];
   Y:=[1,3,4,10];
   ```

2. 2 次元の最小二乗フィット
   以下のデータを

   $$f(x, y) = a_1 + a_2 x + a_3 y + a_4 xy$$

   で近似せよ

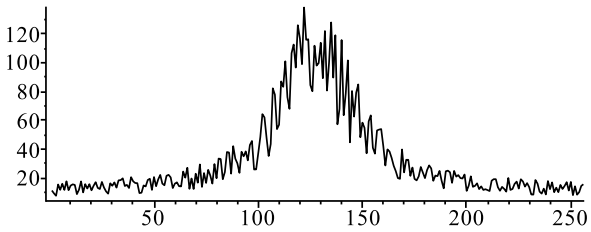
```
 x,  y,  z
-1, -1,  2.00000
-1,  0,  0.50000
-1,  1, -1.00000
 0, -1,  0.50000
 0,  0,  1.00000
 0,  1,  1.50000
 1, -1, -1.00000
 1,  0,  1.50000
 1,  1,  4.00000
```

第 8 章

非線形最小 2 乗法 (NonLinearFit)

8.1 非線形最小 2 乗法の原理

前章では，データに近似的にフィットする最小二乗法を紹介した．ここでは，フィット式が多項式のような線形関係にない関数の最小二乗法を紹介する．図のようなデータにフィットする場合を考えよう．



このデータにあてはめるのはローレンツ関数,

$$F\left(x;\boldsymbol{a}\right)=a_1+\frac{a_2}{a_3+\left(x-a_4\right)^2}$$

である．この関数の特徴は，今まで見てきた関数と違いパラメータが線形関係になっていない．誤差関数は，いままでと同様に

$$\chi^2\left(\boldsymbol{a}\right)=\sum_i^N d_i^2=\sum_i^N\left(F\left(x_i;a\right)-y_i\right)^2$$

で，$a=a_0,a_1,..$ をパラメータとして変えた時に最小となる値を求める点もかわらない．しかし，線形の最小二乗法のように微分しても一元の方程式にならず，連立方程式を単に解くだけでは求まらない．

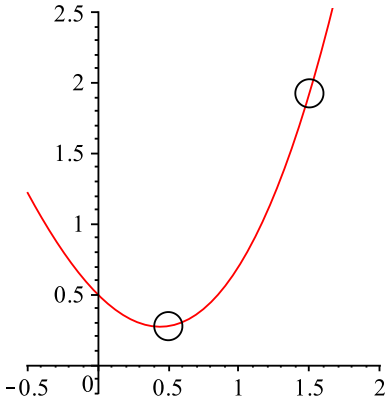
そこで図のような 2 次関数の最小値を求める場合を考える．最小値の点 a_0 のまわりで，Taylor 展開すると，$\boldsymbol{d},\boldsymbol{D}$ をそれぞれの係数とすると，

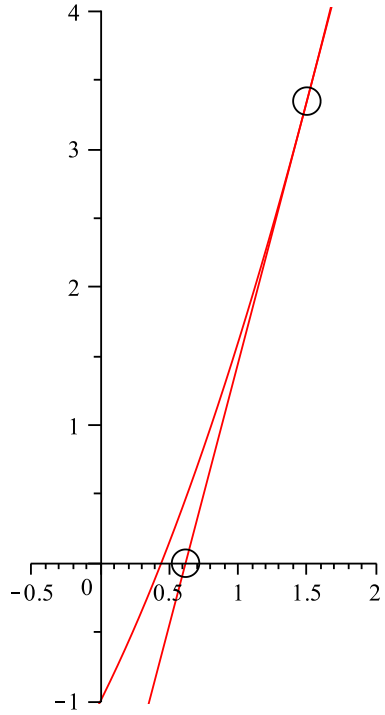
$$\chi\left(\boldsymbol{a}\right)^2=\chi\left(\boldsymbol{a_0}\right)^2-\boldsymbol{d}\left(\boldsymbol{a}-\boldsymbol{a_0}\right)+\frac{1}{2}\boldsymbol{D}\left(\boldsymbol{a}-\boldsymbol{a_0}\right)^2$$

である．最小の点 a_0 は，微分が 0 になるので,

$$\boldsymbol{a_0}=\boldsymbol{a}+\boldsymbol{D}^{-1}\times\left(-\boldsymbol{d}\right)$$

と予測される．図を参照して上の式を導け．またその意味を考察せよ．





現実には高次項の影響で計算通りにはいかず，単に最小値の近似値を求めるだけである．これは，$\chi(\boldsymbol{a})^2$ の微分関数の解を Newton 法で求める操作に対応する．つまり，この操作を何度も繰り返せばいずれ解がある精度で求まるはず．

8.2　具体的な手順

パラメータの初期値を

$$a_0 + \Delta a,\ b_0 + \Delta b,\ c_0 + \Delta c,\ d_0 + \Delta d$$

とする．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

$$\Delta f = f(a_0 + \Delta a_1, b_0 + \Delta b_1, c_0 + \Delta c_1, d_0 + \Delta d_1) - f(a_0, b_0, c_0, d_0)$$

$$= \left(\frac{\partial}{\partial a} f\right)_0 \Delta a_1 + \left(\frac{\partial}{\partial b} f\right)_0 \Delta b_1 + \left(\frac{\partial}{\partial c} f\right)_0 \Delta c_1 + \left(\frac{\partial}{\partial d} f\right)_0 \Delta d_1$$

となる．

課題でつくったデータは $\mathrm{t} = 1$ から $\mathrm{t} = 256$ までの時刻に対応したデータ点 $f_1, f_2, \cdots f_{256}$ とする．各測定値とモ

デル関数から予想される値との差 $\Delta f_1, \Delta f_2, \cdots, \Delta f_{256}$ は，

$$\begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} = J \begin{pmatrix} \Delta a_1 \\ \Delta b_1 \\ \Delta c_1 \\ \Delta d_1 \end{pmatrix} \tag{8.1}$$

となる．ここで J はヤコビ行列と呼ばれる行列で，4 行 256 列

$$J = \begin{pmatrix} \left(\frac{\partial}{\partial a} f\right)_1 & \left(\frac{\partial}{\partial b} f\right)_1 & \left(\frac{\partial}{\partial c} f\right)_1 & \left(\frac{\partial}{\partial d} f\right)_1 \\ \vdots & \vdots & \vdots & \vdots \\ \left(\frac{\partial}{\partial a} f\right)_{256} & \left(\frac{\partial}{\partial b} f\right)_{256} & \left(\frac{\partial}{\partial c} f\right)_{256} & \left(\frac{\partial}{\partial d} f\right)_{256} \end{pmatrix} \tag{8.2}$$

である．このような矩形行列の逆行列は転置行列 J^T を用いて，‘

$$J^{-1} = \left(J^T J\right)^{-1} J^T \tag{8.3}$$

と表わされる．したがって，真値からのずれは

$$\begin{pmatrix} \Delta a_2 \\ \Delta b_2 \\ \Delta c_2 \\ \Delta d_2 \end{pmatrix} = \left(J^T J\right)^{-1} J^T \begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} \tag{8.4}$$

で求められる．理想的には $(\Delta a_2, \Delta b_2, \Delta c_2, \Delta d_2)$ は $(\Delta a, \Delta b, \Delta c, \Delta d)$ に一致するはずだが，測定誤差と高次項のために一致しない．初期値に比べ，より真値に近づくだけ．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る．

8.3 Maple による解法の指針

線形代数計算のためにサブパッケージとして LinearAlgebra を呼びだしておく．

```
> restart;
  with(plots):
  with(LinearAlgebra):
```

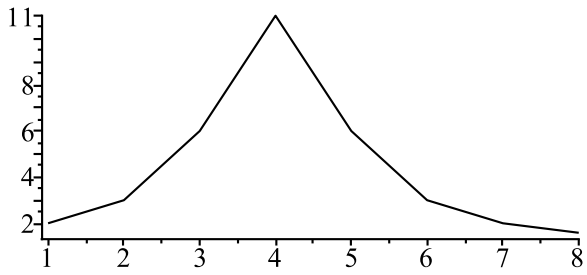
データを読み込む．

```
> ndata:=8:
  f1:=t->subs({a1=1,a2=10,a3=1,a4=4},a1+a2/(a3+(t-a4)^2) );
```

$$f1 := t \mapsto 1 + 10\left(1 + (t-4)^2\right)^{-1}$$

データの表示

```
> T:=[seq(f1(i),i=1..ndata)]:
  listplot(T);
  l1:=listplot(T):
```



ローレンツ型の関数を仮定し，関数として定義.

```
> f:=t->a1+a2/(a3+(t-a4)^2); nparam:=4:
```

$$f := t \mapsto a1 + \frac{a2}{a3 + (t - a4)^2}$$

ヤコビアンの中の微分を新たな関数として定義.

```
> for i from 1 to nparam do
    dfda||i:=unapply(diff(f(x),a||i),x);
  end do;
```

$$dfda1 := x \mapsto 1$$

$$dfda2 := x \mapsto \left(a3 + (x - a4)^2\right)^{-1}$$

$$dfda3 := x \mapsto -\frac{a2}{\left(a3 + (x - a4)^2\right)^2}$$

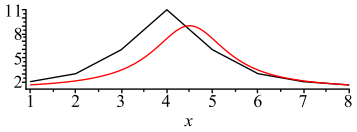
$$dfda4 := x \mapsto -\frac{a2\,(-2\,x + 2\,a4)}{\left(a3 + (x - a4)^2\right)^2}$$

ここで，”||”は連結作用素とよばれる Maple のコマンドで，$dfda||1 \mapsto dfda1$ と連結する．初期値を仮定して，データとともに関数を表示.

```
> g1:=Vector([1,8,1,4.5]):
  guess1:={}:
  for i from 1 to nparam do
    guess1:={op(guess1),a||i=g1[i]};
  end do:
  guess1;
```

$$\{a1 = 1, a2 = 8, a3 = 1, a4 = 4.5\}$$

```
> p1:=plot(subs(guess1,f(x)),x=1..ndata):
  display(l1,p1);
```



見やすいように，小数点以下を 3 桁表示に制限する.

```
> interface(displayprecision=3):
> df:=Vector([seq(subs(guess1,T[i]-f(i)),i=1..ndata)]);
```

$$df := \begin{bmatrix} 0.396 \\ 0.897 \\ 2.538 \\ 3.600 \\ -1.400 \\ -0.462 \\ -0.103 \\ -0.016 \end{bmatrix}$$

```
> Jac:=Matrix(ndata,nparam):
  for i from 1 to ndata do
    for j from 1 to nparam do
      Jac[i,j]:=evalf(subs(guess1,dfda||j(i)));
    end do:
  end do:
  Jac;
```

$$\begin{bmatrix} 1.0 & 0.075 & -0.046 & -0.319 \\ 1.0 & 0.138 & -0.152 & -0.761 \\ 1.0 & 0.308 & -0.757 & -2.272 \\ 1.0 & 0.800 & -5.120 & -5.120 \\ 1.0 & 0.800 & -5.120 & 5.120 \\ 1.0 & 0.308 & -0.757 & 2.272 \\ 1.0 & 0.138 & -0.152 & 0.761 \\ 1.0 & 0.075 & -0.046 & 0.319 \end{bmatrix}$$

```
> tJac:=(MatrixInverse(Transpose(Jac).Jac)).Transpose(Jac);
```

$$tJac := \begin{bmatrix} 0.565 & 0.249 & -0.354 & 0.040 & 0.040 & -0.354 & 0.249 & 0.565 \\ -2.954 & -0.506 & 4.012 & -0.552 & -0.552 & 4.012 & -0.506 & -2.954 \\ -0.352 & -0.029 & 0.557 & -0.176 & -0.176 & 0.557 & -0.029 & -0.352 \\ -0.005 & -0.012 & -0.035 & -0.080 & 0.080 & 0.035 & 0.012 & 0.005 \end{bmatrix}$$

```
> g2:=tJac.df;
  g1:=g1+g2;
```

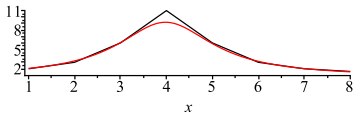
$$g2 := \begin{bmatrix} -0.235 \\ 5.592 \\ 0.613 \\ -0.520 \end{bmatrix}$$

$$g1 := \begin{bmatrix} 0.765 \\ 13.592 \\ 1.613 \\ 3.980 \end{bmatrix}$$

これをまたもとの近似値 (guess) に入れ直して表示させると以下のようになる．カーブがデータに近づいているのが確認できよう．この操作をずれが十分小さくなるまで繰り返す．

```
> guess1:={seq(a||i=g1[i],i=1..nparam)};
  p1:=plot(subs(guess1,f(x)),x=1..ndata):
  display(l1,p1);
```

$$guess1 := \{a1 = 0.765, a2 = 13.592, a3 = 1.613, a4 = 3.980\}$$



4 回ほど繰り返すと以下の通り，いい値に収束している．

$$guess1 := \{a1 = 1.006, a2 = 9.926, a3 = .989, a4 = 4.000\}$$

8.4　Gauss-Newton 法に関するメモ

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，広く使われている．しかし，収束性は高くなく，むしろ発散しやすいので注意が必要．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 法である．明快な解説が Numerical Recipes in C(C 言語による数値計算のレシピ）WilliamH.Press 他著，技術評論社 1993 にある．

8.5　課題

1. 一山ピークへのフィット
   以下の 256 個のデータ

```
> ndata:=256; f1:=t->subs({a1=10,a2=40000,a3=380,a4=128},a1+a2/(a3+(t-a4)^2) );
> T:=[seq(f1(i)*(0.6+0.8*evalf(rand()/10^12)),i=1..ndata)]:
> f:=t->a1+a2/(a3+(t-a4)^2);
```

   で近似したときのパラメータ a1,a2,a3,a4 を求めよ．ただし，パラメータの初期値は，ある程度近い値にしないと収束しない．
2. 二山ピークのフィット以下のように作成したデータ

```
> ndata:=256; f1:=t->subs({a=10,b=40000,c=380,d=128},a+b/(c+(t-d)^2) );
> f2:=t->subs({a=10,b=40000,c=380,e=90},a+b/(c+(t-e)^2) );
> T:=[seq((f1(i)+f2(i))*(0.6+0.2*evalf(rand()/10^12)),i=1..ndata)]:
```

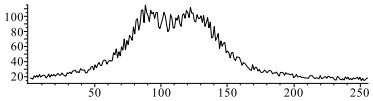
   を

```
> f:=t->a1+a2/(a3+(t-a4)^2)+a2/(a3+(t-a5)^2);
```

$$f := t \mapsto a1 + \frac{a2}{a3 + (t - a4)^2} + \frac{a2}{a3 + (t - a5)^2}$$

   で近似したときのパラメータを求めよ．

```
> l1:=listplot(T): display(l1);
```



8.6 解答例

2. ふた山ピークへのフィット.

```
> restart; with(plots): with(LinearAlgebra):
> f1:=t->subs({a=10,b=40000,c=380,d=128},a+b/(c+(t-d)^2) );
> f2:=t->subs({a=10,b=40000,c=380,e=90},a+b/(c+(t-e)^2) );
> T:=[seq((f1(i)+f2(i))*(0.6+0.2*evalf(rand()/10^12)),i=1..256)]:
```

$$f1 := t \mapsto 10 + 40000 \left(380 + (t-128)^2\right)^{-1}$$
$$f2 := t \mapsto 10 + 40000 \left(380 + (t-90)^2\right)^{-1}$$

```
> l1:=listplot(T):
> f:=t->a1+a2/(a3+(t-a4)^2)+a2/(a3+(t-a5)^2);
  nparam:=5:
```

$$f := t \mapsto a1 + \frac{a2}{a3 + (t-a4)^2} + \frac{a2}{a3 + (t-a5)^2}$$

```
> for i from 1 to nparam do
    dfda||i:=unapply(diff(f(x),a||i),x);
  end do;
```

$$dfda1 := x \mapsto 1$$
$$dfda2 := x \mapsto \left(a3 + (x-a4)^2\right)^{-1} + \left(a3 + (x-a5)^2\right)^{-1}$$
$$dfda3 := x \mapsto -\frac{a2}{\left(a3 + (x-a4)^2\right)^2} - \frac{a2}{\left(a3 + (x-a5)^2\right)^2}$$
$$dfda4 := x \mapsto -\frac{a2\ (-2\,x + 2\,a4)}{\left(a3 + (x-a4)^2\right)^2}$$
$$dfda5 := x \mapsto -\frac{a2\ (-2\,x + 2\,a5)}{\left(a3 + (x-a5)^2\right)^2}$$

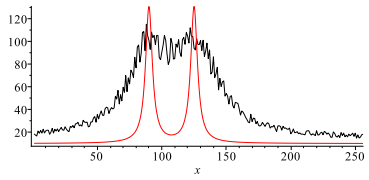
```
> g1:=Vector([10,1200,10,125,90]);
```

$$g1 := \begin{bmatrix} 10 \\ 1200 \\ 10 \\ 125 \\ 90 \end{bmatrix}$$

```
> guess1:={seq(a||i=g1[i],i=1..nparam)};
```

$$guess1 := \{a1 = 10, a2 = 1200, a3 = 10, a4 = 125, a5 = 90\}$$

```
> p1:=plot(subs(guess1,f(x)),x=1..256):
  display(l1);
```



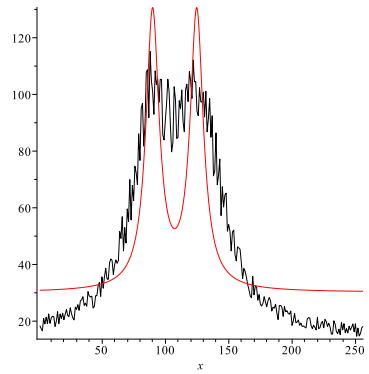

```
> df:=Vector([seq(subs(guess1,T[i]-f(i)),i=1..256)]):
  Jac:=Matrix(1..256,1..nparam,sparse):
  for i from 1 to 256 do
    for j from 1 to nparam do
      Jac[i,j]:=evalf(subs(guess1,dfda||j(i)));
    end do:
  end do:
  tJac:=(MatrixInverse(Transpose(Jac).Jac)).Transpose(Jac):
  g2:=tJac.df; g1:=g1+g2;
```

$$g2 := \begin{bmatrix} -0.390553882992161205 \\ 1584.55290636967129 \\ 24.9577909601538366 \\ -0.0472041829705451138 \\ -0.00719532042503852940 \end{bmatrix}$$

$$g1 := \begin{bmatrix} 13.6348019182603064 \\ 29567.3667677707381 \\ 410.545681677467769 \\ 128.512734548828887 \\ 90.9223109918718678 \end{bmatrix}$$

```
> guess1:={seq(a||i=g1[i],i=1..nparam)};
  p1:=plot(subs(guess1,f(x)),x=1..256):
  display(l1,p1);
```

$$guess1 := \{a1 = 30.251, a2 = 3854.136, a3 = 39.571, a4 = 124.800, a5 = 89.960\}$$



何回か繰り返せば，データ点に近づいてくるはず．

第 9 章

FFT(Fast Fourier Transformation)

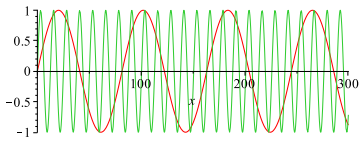
9.1 FFT の応用

Fast Fourier Transformation(FFT) 高速フーリエ変換 (あるいはデジタル (離散) フーリエ変換 (DFT)) は，周波数分解やフィルターを初め，画像処理などの多くの分野で使われている．基本となる考え方は，直交基底による関数の内挿法である．最初にその応用例を見た後，どのような理屈で FFT が動いているかを解説する．

9.1.1 周波数分解

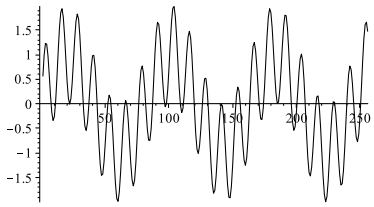
はじめの例は，周波数分解．先ずは，非整合な波を二つ用意しておく．

```
> restart:
  funcs:=[sin(i/13),sin(i/2)];
  #funcs:=[sin(i*2),2*sin(i/2)];
  plot(funcs,i=0..300);
```



これを重ねあわせた波を作る．

```
> data1:=[]:
  for i from 1 to 256 do
    data1:=[op(data1),evalf(funcs[1]+funcs[2]))];
  end do:
  with(plots):
  listplot(data1);
```



ゆっくり変化する波に，激しく変化する波が重なっていることが読み取れる．これに FFT を掛ける

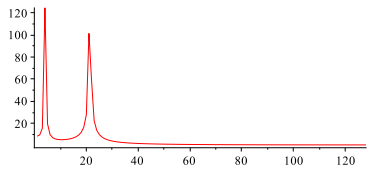
```
> X:=array(data1):
```

```
Y:=array(1..256,sparse):
FFT(8,X,Y);
```

256

その強さを求めて，周波数で表示すると，

```
> Data2:=[seq([i,sqrt(X[i]^2+Y[i]^2)],i=1..128)]:
  plot(Data2);
```



もとの 2 つの周波数に対応するところにピークができているのが確認できる．広がりは，誤差のせい．logplot でも良い．

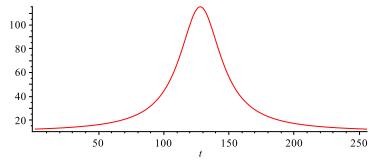
9.1.2 高周波フィルター

次の例は，高周波フィルター．たとえば次のようなローレンツ関数を考える．

```
> restart;
  f1:=t->subs(a=10,b=40000,c=380,d=128,a+b/(c+(t-d)^2));
```

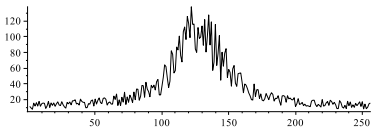
$$f1 := t \mapsto 10 + \frac{40000}{380 + (t - 128)^2}$$

```
> plot(f1(t),t=1..256);
```



これにノイズがのると，次のようになる．

```
> T:=[seq(f1(i)*(0.6+0.8*evalf(rand()/10^12)),i=1..256)]:
  #T:=[seq(evalf(rand()/10^12),i=1..256)]:  #これはホワイトノイズ
  #T:=[seq(f1(i),i=1..256)]: #これは元の関数そのまま
  with(plots):
  listplot(T);
```



これに高周波フィルターを掛けるとノイズが消えるが，その様子を示そう．先ずは，FFT を掛ける．

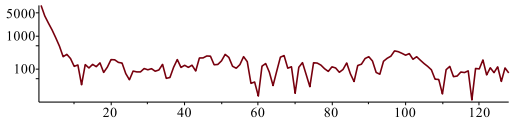
```
> Idata:=array([seq(0,i=1..256)]):
```

```
Rdata:=convert(T,array):
FFT(8,Rdata,Idata);
```

256

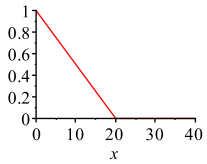
これは次のような強度分布をもっている．

```
> Adata:=[seq([i,sqrt(Idata[i]^2+Rdata[i]^2)],i=1..128)]:
> logplot(Adata);
```



低周波の部分に，ゆっくりとした変化を表す成分が固まっている．次のような三角フィルターを用意する．これは，低周波ほど影響を大きくするフィルター．

```
> filter:=x->piecewise(x>=0 and x<=20,(1-x/20)): #三角フィルター
  #filter:=x->piecewise(x>=0 and x<=20,1); #方形フィルター
  plot(filter(x),x=0..40);
```

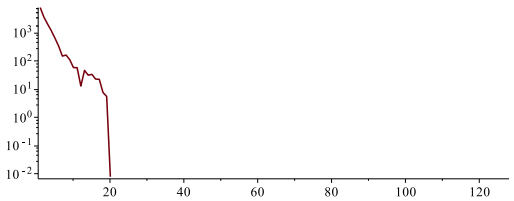


これとデータを各点で掛けあわせる事によって，フィルターを通したことになる．

```
> FRdata:=array([seq(Rdata[i]*filter(i),i=1..256)]):
> FIdata:=array([seq(Idata[i]*filter(i),i=1..256)]):
```

先ほどと同様に表示すると

```
> Bdata:=[seq([i,sqrt(FIdata[i]^2+FRdata[i]^2)],i=1..128)]:
> logplot(Adata);
```



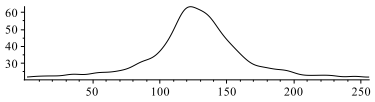
$i = 20$ 以上の領域がフィルターによってちょん切られていることが確認できる．これを逆フーリエ変換する．

```
> iFFT(8,FRdata,FIdata);
```

256

これを表示すると，

```
> listplot(FRdata);
```



となる．ノイズが取り除かれているのが確認できる．元の関数に加えたホワイトノイズに FFT を掛ければ分かるが，全周波数域にわたって均質に広がった関数となる．これを三角フィルターなどで高周波成分をカットすることで，ノイズが取り除かれていくのが理解されよう．

9.2 FFT の動作原理

このように便利な FFT であるが，どのような理屈で導かれるのか？ Fourier 変換法は，この課題だけでも何回ものコマ数が必要なほどの内容を含んでいる．ここでは，その基本となる考え方（のひとつ) だけを提示する．

1. 関数の内挿で導入した基底関数を直交関数系でとる．ところが，展開係数を逆行列で求める手法では計算が破綻．
2. 直交関係からの積分による係数決定．
3. 選点直交性による計算の簡素化．
4. 高速フーリエ変換アルゴリズムによる高速化．

9.3 関数内挿としての Fourier 関数系

一連の関数系による関数の内挿は，基底関数を $\varphi_n(x)$ として

$$F(x) = \sum_{n=1}^{N} a_n \varphi_n(x)$$

で得られることを見た．Fourier 変換では基底関数として $\varphi_n\left(x\right) = \sin\left(2\,\pi nx\right), \cos\left(2\,\pi nx\right)$ をとる．関数の内挿法で示したように，この x_i での値 $f_i, i = 1 \cdots M$ と，近似の次数 (N) とでつくる係数行列，

$$A = \begin{bmatrix} \varphi_0(x_0) & \varphi_1(x_0) & \cdots & \varphi_N(x_0) \\ \vdots & \vdots & \vdots & \vdots \\ \varphi_0(x_M) & \varphi_1(x_M) & \cdots & \varphi_N(x_M) \end{bmatrix}$$

を求めて，係数 a_i とデータ点 f_i をそれぞれベクトルと考えると，

$$\boldsymbol{A}.\boldsymbol{a} = \boldsymbol{f}$$

から，通常の逆行列を求める手法で係数を決定することもできる．しかし，この強引な方法はデータ数，関数の次数が多い，フーリエ変換が対象としようとする問題では破綻する．もっといい方法が必要で，それが直交関数系では存在する．

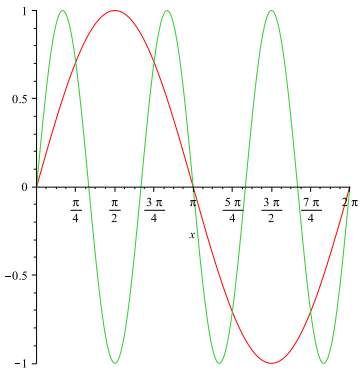
9.4 直交関係からの積分による係数決定

関数の直交関係は，

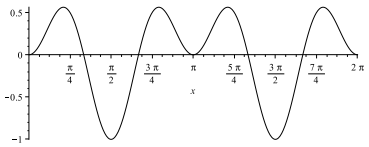
$$\int_a^b \varphi_n\left(x\right)\varphi_m\left(x\right)dx = \delta_{mn}C_n = \begin{cases} C_m & at\, n = m \\ 0 & at\, n \neq m \end{cases} \tag{9.1}$$

である．定数 C_m は，$\sin, \cos$ の三角関数系では次の通り．

```
> plot([sin(x),sin(3*x)],x=0..2*Pi);
```



```
> plot([sin(x)*sin(3*x)],x=0..2*Pi, color=black);
```



```
> int(sin(x)*sin(3*x),x=0..2*Pi);
```

$$0$$

```
> for i from 1 to 3 do for j from 1 to 3 do S:=int(sin(i*x)*sin(j*x),x=0..2*Pi);
> print(i,j,S); end do; end do:
```

$$1, 1, \pi$$
$$1, 2, 0$$
$$1, 3, 0$$
$$2, 1, 0$$
$$2, 2, \pi$$
$$2, 3, 0$$
$$3, 1, 0$$
$$3, 2, 0$$
$$3, 3, \pi$$

$$\int_a^b F(x)\,\varphi_m(x)\,dx$$

を考える．先程の**??**式をいれると

$$\int_a^b F(x)\,\varphi_m(x)\,dx = \int_a^b \sum_{n=1}^{N} a_n \varphi_n(x)\,\varphi_m(x)\,dx = \begin{cases} a_m C_m & at\ n = m \\ 0 & at\ n \neq m \end{cases} \tag{9.2}$$

となる．こうして，係数 a_n が

$$a_n = \frac{1}{C_n}\int_a^b F(x)\,\varphi_n(x)\,dx$$

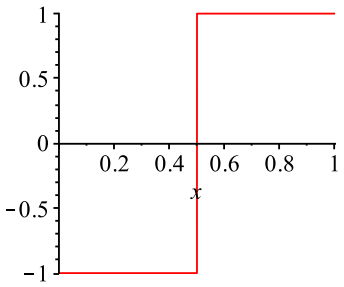
で決定できる．

9.5 直接積分によるフーリエ係数

対象とする関数をまず作る.

```
> restart;
> #F:=x->piecewise(x=0,1/2,x>0,x);
  #F:=x->piecewise(x<1/2,x,x>=1/2,1-x);
  #F:=x->piecewise(x<1/2,-1,x>1/2,1);
  F:=x->piecewise(x<1/2,-1,x>=1/2,1);
  #F:=x->piecewise(x>0 and x<1/2,-1,x>1/2,1);
  #F:=x->x-1/2;
  plot(F(x),x=0..1);
```

$$F := x \mapsto piecewise(x < \frac{1}{2}, -1, \frac{1}{2} \le x, 1)$$



piecewise 関数は階段関数で，振る舞いはコメント (#) を適当に外して確認せよ．初期設定.

```
> KK:=3; N:=2^KK;L:=1-0;
> 2*Pi*1/L*x;
```

$$2\pi x$$

```
> int(F(x)*cos(2*Pi*1/L*x),x=0..L);
```

$$0$$

```
> for n from 0 to N do
    a[n]:=2/L*int(F(x)*cos(2*Pi*n/L*x),x=0..L);
  end do;
```

$$a_0 := 0$$
$$a_1 := 0$$
$$a_2 := 0$$
$$a_3 := 0$$
$$a_4 := 0$$
$$a_5 := 0$$
$$a_6 := 0$$
$$a_7 := 0$$
$$a_8 := 0$$

```
> for n from 0 to N do
    b[n]:=2/L*int(F(x)*sin(2*Pi*n/L*x),x=0..L);
  end do;
```

$$b_0 := 0$$
$$b_1 := \frac{4}{\pi}$$
$$b_2 := 0$$
$$b_3 := \frac{4}{3\pi}$$
$$b_4 := 0$$
$$b_5 := \frac{4}{5\pi}$$
$$b_6 := 0$$
$$b_7 := \frac{4}{7\pi}$$
$$b_8 := 0$$

ここで，オイラーの関係

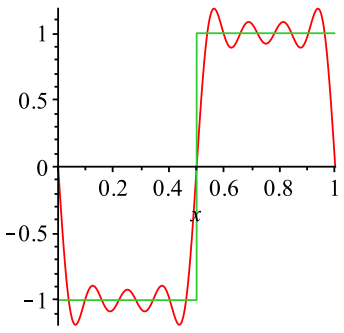
$$a[n] = c[n] + c[-n],\ b[n] = I(c[n] - c[-n])$$
$$c[-n] = \frac{1}{2}(a[n] + b[n]),\ c[n] = \frac{1}{2}(a[n] - Ib[n]) \tag{9.3}$$

を使って，三角関数系から exp へ変換する．

```
> for n from 0 to N do c[n]:=1/L*int(F(x)*exp(-I*2*Pi*n/L*x),x=0..L); end do;
> for n from 1 to N do c[-n]:=1/L*int(F(x)*exp(I*2*Pi*n/L*x),x=0..L); end do;
```

$$
\begin{aligned}
c_0 &:= 0 \\
c_1 &:= \frac{2I}{\pi} \\
c_2 &:= 0 \\
c_3 &:= \frac{2I}{3\pi} \\
c_4 &:= 0 \\
c_5 &:= \frac{2I}{5\pi} \\
c_6 &:= 0 \\
c_7 &:= \frac{2I}{7\pi} \\
c_8 &:= 0 \\
c_{-1} &:= -\frac{2I}{\pi} \\
c_{-2} &:= 0 \\
c_{-3} &:= -\frac{2I}{3\pi} \\
c_{-4} &:= 0 \\
c_{-5} &:= -\frac{2I}{5\pi} \\
c_{-6} &:= 0 \\
c_{-7} &:= -\frac{2I}{7\pi} \\
c_{-8} &:= 0
\end{aligned}
$$

```
> F1:=unapply(sum(evalf(c[i]*exp(I*2*Pi*i/L*x)),i=-(N-1)..(N-1)),x):
> plot({Re(F1(x)),F(x)},x=0..1);
```




```
> evalf(2/3/Pi);
```

0.2122065907

9.6　選点直交性による計算の簡素化

ところが，実際に積分していては，時間がかかりすぎる．直交関数系の選点直交性を使うとより簡単になる．

■直交関数系の選点直交性

直交多項式は，

$$\varphi_n(x) = 0 \, at \, x_1, \, x_2, \, \cdots x_n$$

である．$n-1$ 以下の次数 m, l では，

$$\sum_{i=1}^{n} \phi_l(x_i)\,\varphi_m(x_i) = \delta_{ml} C_l$$

が成り立つ．これは，直交関係と違い積分でないことに注意．証明は略．

これを使えば，この先程の直交関数展開

$$F(x) = \sum_{l=1}^{N} a_l \varphi_l(x)$$

の両辺に $\varphi_m(x_i)$ を掛けて i について和をとれば，

$$\begin{aligned}\sum_{i=1}^{n} F(x_i)\,\phi_m(x_i) &= \sum_{i=1}^{n}\sum_{l=1}^{N} a_l \varphi_l(x_i)\,\varphi_m(x_i) \\ &= \sum_{l=1}^{N} a_l \sum_{i=1}^{n} \varphi_l(x_i)\,\varphi_m(x_i) \\ &= \sum_{l=1}^{N} a_l \delta_{ml} C_m = a_m C_m\end{aligned}$$

となる．つまり，

$$a_m = \frac{1}{C_m}\sum_{i=1}^{n} F(x_i)\,\varphi_m(x_i)$$

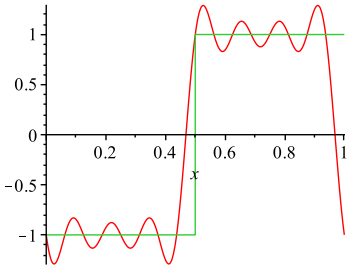
となり，単純な関数の代入とかけ算で係数が決定される．

9.6.1 選点直交性を用いた結果

```
> KK:=4; N:=2^KK;L:=1-0;
> for k from 0 to N-1 do
    c[k]:=evalf(sum(F(i*L/N)*exp(-I*2*Pi*k*i/N),i=0..N-1));
  end do;

c_0:=0.
c_1:=-2.000000000 + 10.05467898 I
c_2:=0.
c_3:=-2.000000000 + 2.993211524 I
c_4:=0.
c_5:=-2.000000001 + 1.336357276 I
c_6:=0.
c_7:=-2.000000001 + 0.3978247331 I
c_8:=0.
c_9:=-2.000000001 - 0.3978247331 I
c_10:=0.
c_11:=-2.000000001 - 1.336357276 I
c_12:=0.
c_13:=-2.000000000 - 2.993211524 I
c_14:=0.
c_15:=-2.000000000 - 10.05467898 I
```

```
> F1:=unapply(sum(evalf(c[i]*exp(I*2*Pi*i/L*x)/N),i=0..(N/2-1))+
> sum(evalf(c[N-i]*exp(-I*2*Pi*i/L*x)/N),i=1..(N/2-1)),x):
> plot({Re(F1(x)),F(x)},x=0..1);
```



9.7 高速フーリエ変換アルゴリズムによる高速化

sin, cos と exp 関数を結びつけるオイラーの関係を使うと，

$$\exp\left(\frac{2\pi}{N}I\right) = \cos\left(\frac{2\,\pi}{N}\right) + I\,\sin\left(\frac{2\,\pi}{N}\right)$$

と変換できる．これを使うと，

$$c_k = \frac{1}{C_m}\sum_{i=0}^{N-1} F\left(x_i\right)\exp\left(\frac{-2\,\pi I}{N}\right)$$

となる．$N = 8$ の場合を実際に計算すると，$z = \exp(-\frac{2\pi}{8}I)$ として，$z^8 = 1, z^9 = z, \cdots$ を使うと，

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & z & z^2 & z^3 & z^4 & z^5 & z^6 & z^7 \\ 1 & z^2 & z^4 & z^6 & 1 & z^2 & z^4 & z^6 \\ 1 & z^3 & z^6 & z & z^4 & z^7 & z^2 & z^5 \\ 1 & z^4 & 1 & z^4 & 1 & z^4 & 1 & z^4 \\ 1 & z^5 & z^2 & z^7 & z^4 & z^1 & z^6 & z^3 \\ \vdots & & & & & & & \\ \vdots & & & & & & & \end{bmatrix} \begin{bmatrix} F_0 \\ F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \end{bmatrix}$$

となる．この行列計算を素直に実行すると，8 × 8=64 回の演算が必要となる．これを減らせないかと考えたのが，高速フーリエ変換の始まりである．この行列をよく見ると同じ計算を重複しておこなっていることが分かる．そこで，行列の左側と右側で同じ計算をしている部分をまとめると，

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z & z^2 & z^3 \\ 1 & z^2 & z^4 & z^6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z^3 & z^6 & z \\ 1 & z^4 & 1 & z^4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z^5 & z^2 & z^7 \\ 1 & z^6 & z^4 & z^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & z^7 & z^6 & z^5 \end{bmatrix} \begin{bmatrix} F_0 + F_4 \\ F_1 + F_5 \\ F_2 + F_6 \\ F_3 + F_7 \\ F_0 - F_4 \\ F_1 - F_5 \\ F_{[\,]} - F_6 \\ F_{[\,]} - F_7 \end{bmatrix}$$

とすることができる．ここで，$z^4 = -1$ などを使っている．右側のベクトルの計算でロスするが，行列の中の計算の回数を半分に減らすことができる．再度できあがった行列を見れば，同じ計算をさらにまとめることができそうである．こうして，次々と計算回数を減らしていくことが可能で，最終的に行列部分の計算が一切なくなる．残るのは，右側のベクトルの足し算引き算だけになる．

このベクトルの組み合わせは，一見相当複雑そうで，その条件分岐で時間がかかりそうに思われる．しかし，よく調べてみれば，単純なビット演算で処理することが可能であることが判明した．こうして，2 の整数乗のデータの組

に対しては，極めて高速にフーリエ変換を実行することが可能となった．FFT での演算回数は，データ数を N とすると

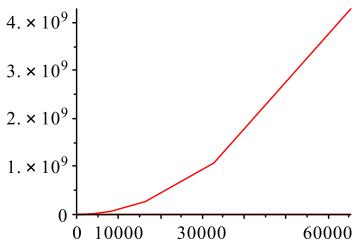
$$N \log_2 N$$

となる．単純な場合の N^2 と比較すると，以下のようになり，どれだけ高速化されているかが理解されよう．

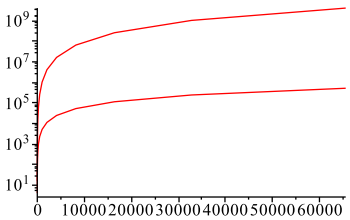
```
> dN2:=[]; dFft:=[]; for i from 2 to 16 do N:=2^i; n2:=N*N; Fft:=N/2*log[2](N);
> Fft/n2; printf("%10d %12d %12d %10.5f\n",N,n2,Fft,evalf(Fft/n2));
> dN2:=[op(dN2),[N,n2]]; dFft:=[op(dFft),[N,Fft]]; end do:
```

```
         4           16            4    0.25000
         8           64           12    0.18750
        16          256           32    0.12500
        32         1024           80    0.07812
        64         4096          192    0.04688
       128        16384          448    0.02734
       256        65536         1024    0.01562
       512       262144         2304    0.00879
      1024      1048576         5120    0.00488
      2048      4194304        11264    0.00269
      4096     16777216        24576    0.00146
      8192     67108864        53248    0.00079
     16384    268435456       114688    0.00043
     32768   1073741824       245760    0.00023
     65536   4294967296       524288    0.00012
```

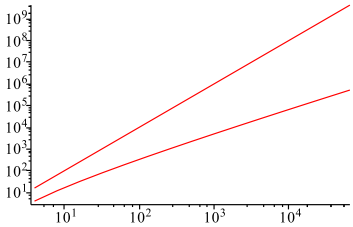
```
> with(plots):
> l1:=plot(dN2): l2:=plot(dFft):
> display(l1,l2);
```




```
> l1:=logplot(dN2): l2:=logplot(dFft):
> display(l1,l2);
```



```
> l1:=loglogplot(dN2): l2:=loglogplot(dFft):
> display(l1,l2);
```



9.8 FFT 関数を用いた結果

```
> KK:=4;
  N:=2^KK;i:='i';L:=1-0;
  x1:=array([evalf(seq(F(i/N),i=0..N-1))]);
  y1:=array([evalf(seq(0,i=0..N-1))]);
```

$$x1 := \left[-1.0 \quad -1.0 \quad -1.0 \quad -1.0 \quad -1.0 \quad -1.0 \quad -1.0 \quad -1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \quad 1.0 \right]$$

$$y1 := \left[0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \quad 0.0 \right]$$

```
> FFT(KK,x1,y1);
```

16

```
> interface(displayprecision=2):
  print(x1);print(y1);
```

$$\left[0.0 \quad -2.0 \quad 0.0 \quad -2.0 \quad 0.0 \quad -2.0 \quad 0.0 \quad -2.0 \quad 0.0 \quad -2.0 \quad 0.0 \quad -2.0 \quad 0.0 \quad -2.0 \quad 0.0 \quad -2.0 \right]$$

$$\left[0.0 \quad 10.05 \quad 0.0 \quad 2.99 \quad 0.0 \quad 1.34 \quad 0.0 \quad 0.40 \quad 0.0 \quad -0.40 \quad 0.0 \quad -1.34 \quad 0.0 \quad -2.99 \quad 0.0 \quad -10.05 \right]$$

```
> F2:=unapply(sum(evalf((x1[i]+I*y1[i])*exp(I*2*Pi*(i-1)/L*x)/N),i=1..N/2)+
        sum(evalf((x1[N-i+2]+I*y1[N-i+2])*exp(-I*2*Pi*(i-1)/L*x)/N),i=2..N/2),x):
> plot({Re(F2(x)),F(x)},x=0..1);
```