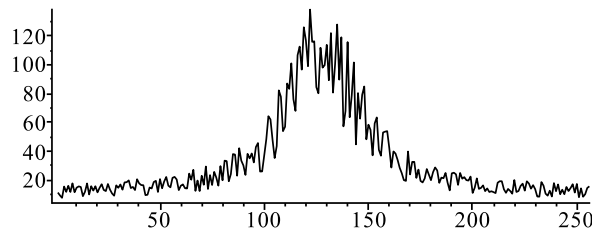


# 第1章 非線形最小2乗法 (NonLinearFit)

## 1.1 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,

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

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

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

で、 $\mathbf{a} = a_0, a_1, \dots$  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。

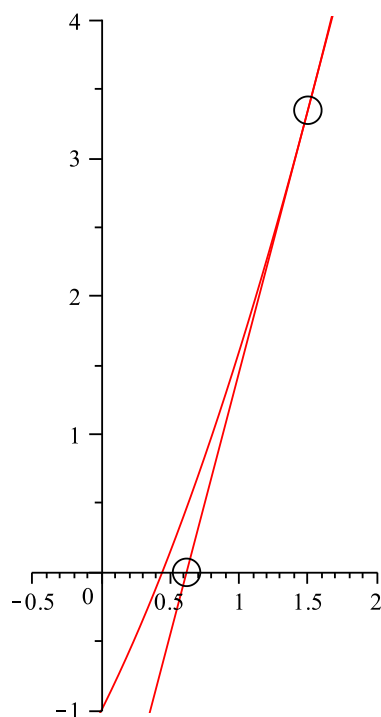
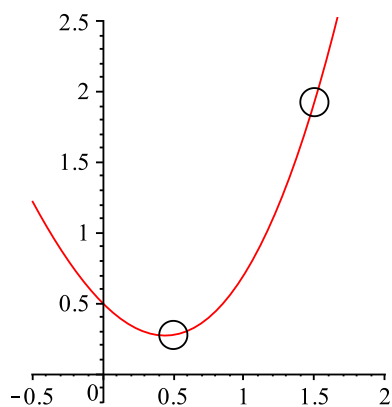
そこで図のような2次関数の最小値を求める場合を考える。最小値の点  $a_0$  のまわりで、Taylor 展開すると、 $\mathbf{d}, \mathbf{D}$  をそれぞれの係数とすると、

$$\chi(\mathbf{a})^2 = \chi(\mathbf{a}_0)^2 - \mathbf{d}(\mathbf{a} - \mathbf{a}_0) + \frac{1}{2} \mathbf{D}(\mathbf{a} - \mathbf{a}_0)^2$$

である。最小の点  $a_0$  は、微分が0になるので、

$$\mathbf{a}_0 = \mathbf{a} + \mathbf{D}^{-1} \times (-\mathbf{d})$$

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、 $\chi(\mathbf{a})^2$  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。

## 1.2 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

$$\begin{aligned} \Delta f &= f(a_0 + \Delta a, b_0 + \Delta b, c_0 + \Delta c, d_0 + \Delta d) - f(a_0, b_0, c_0, d_0) \\ &= \left( \frac{\partial}{\partial a} f \right)_0 \Delta a_1 + \left( \frac{\partial}{\partial b} f \right)_0 \Delta b_1 + \left( \frac{\partial}{\partial c} f \right)_0 \Delta c_1 + \left( \frac{\partial}{\partial d} f \right)_0 \Delta d_1 \end{aligned} \quad (1.1)$$

となる。

課題でつくったデータは  $t = 1$  から  $t = 256$  までの時刻に対応したデータ点  $f_1, f_2, \dots, f_{256}$  とする. 各測定値とモデル関数から予想される値との差  $\Delta f_1, \Delta f_2, \dots, \Delta f_{256}$  は,

$$\begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} = J \begin{pmatrix} \Delta a_1 \\ \Delta b_1 \\ \Delta c_1 \\ \Delta d_1 \end{pmatrix} \quad (1.2)$$

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

$$J = \begin{pmatrix} \left(\frac{\partial}{\partial a} f\right)_1 & \left(\frac{\partial}{\partial b} f\right)_1 & \left(\frac{\partial}{\partial c} f\right)_1 & \left(\frac{\partial}{\partial d} f\right)_1 \\ \vdots & \vdots & \vdots & \vdots \\ \left(\frac{\partial}{\partial a} f\right)_{256} & \left(\frac{\partial}{\partial b} f\right)_{256} & \left(\frac{\partial}{\partial c} f\right)_{256} & \left(\frac{\partial}{\partial d} f\right)_{256} \end{pmatrix} \quad (1.3)$$

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,

$$J^{-1} = (J^T J)^{-1} J^T \quad (1.4)$$

と表わされる. したがって, 真値からのずれは

$$\begin{pmatrix} \Delta a_2 \\ \Delta b_2 \\ \Delta c_2 \\ \Delta d_2 \end{pmatrix} = (J^T J)^{-1} J^T \begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} \quad (1.5)$$

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.

### 1.3 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):
```

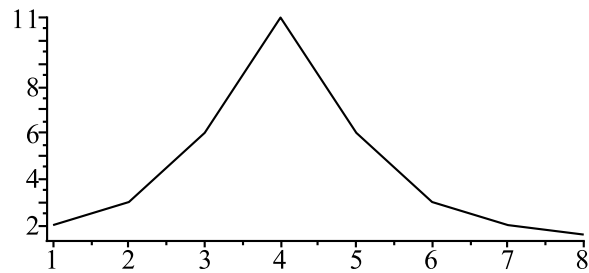
データを読み込む.

```
> 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;
```

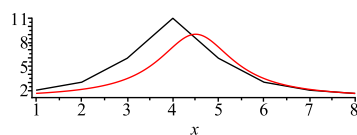
$$\begin{aligned} dfda1 &:= x \mapsto 1 \\ dfda2 &:= x \mapsto \left( a3 + (x - a4)^2 \right)^{-1} \\ dfda3 &:= x \mapsto -\frac{a2}{\left( a3 + (x - a4)^2 \right)^2} \\ dfda4 &:= x \mapsto -\frac{a2 (-2x + 2a4)}{\left( a3 + (x - a4)^2 \right)^2} \end{aligned}$$

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;
```

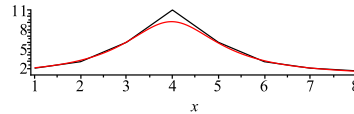
$$g2 := \begin{bmatrix} -0.235 \\ 5.592 \\ 0.613 \\ -0.520 \end{bmatrix}$$

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

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．

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

## 1.4 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) William H. Press 他著，技術評論社 1993 にある．

## 1.5 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)];
```

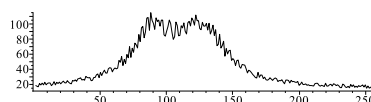
を

```
> 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);
```



## 1.6 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;
```

$$dfda1 := x \mapsto 1$$

$$dfda2 := x \mapsto \left( a3 + (x - a4)^2 \right)^{-1} + \left( a3 + (x - a5)^2 \right)^{-1}$$

$$dfda3 := x \mapsto -\frac{a2}{\left( a3 + (x - a4)^2 \right)^2} - \frac{a2}{\left( a3 + (x - a5)^2 \right)^2}$$

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

$$dfda5 := x \mapsto -\frac{a2 (-2x + 2a5)}{\left( a3 + (x - a5)^2 \right)^2}$$

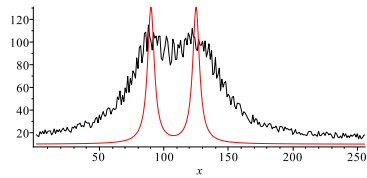
```
> g1:=Vector([10,1200,10,125,90]);
```

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

```
> 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);
```



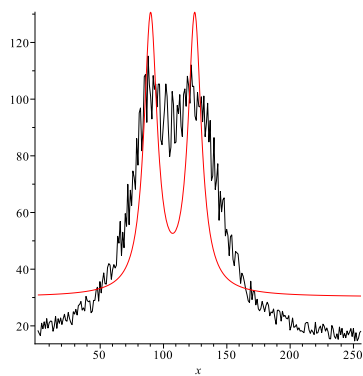
```
> df:=Vector([seq(subs(guess1,T[i]-f(i)),i=1..256)]):
Jac:=Matrix(1..256,1..nparam,sparse):
for i from 1 to 256 do
  for j from 1 to nparam do
    Jac[i,j]:=evalf(subs(guess1,dfda[j](i)));
  end do:
end do:
tJac:=(MatrixInverse(Transpose(Jac).Jac)).Transpose(Jac):
g2:=tJac.df; g1:=g1+g2;
```

$$g^2 := \begin{bmatrix} -0.390553882992161205 \\ 1584.55290636967129 \\ 24.9577909601538366 \\ -0.0472041829705451138 \\ -0.00719532042503852940 \end{bmatrix}$$

$$g^1 := \begin{bmatrix} 13.6348019182603064 \\ 29567.3667677707381 \\ 410.545681677467769 \\ 128.512734548828887 \\ 90.9223109918718678 \end{bmatrix}$$

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

$$guess1 := \{a_1 = 30.251, a_2 = 3854.136, a_3 = 39.571, a_4 = 124.800, a_5 = 89.960\}$$



何回か繰り返せば，データ点に近づいてくるはず。