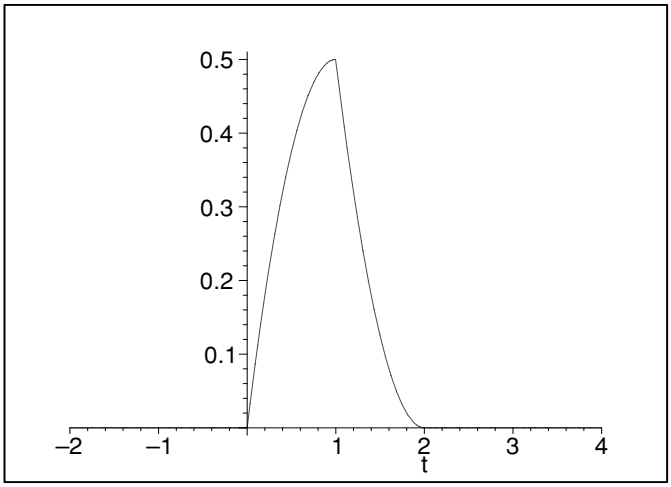


と求まります.

1.6.4 非線形最小二乗法

非線形の最小二乗法を用いてデータフィットをしてくれる関数は default では Maple には用意されていないようです[3]. そこで, 実際に非線形最小二乗法のプログラムを作成し, 実際のデータへのフィッティングを試みてみましょう.

問題

与えられたデータ (ファイル名'DATA101') を,

$$f(x) = a + \frac{b}{c + (x - d)^2} \tag{1.8}$$

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

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

解説

(1.8) 式の特徴は，パラメータが線形関係にないということですが，以下では線形近似に基づいた反復解法を用います．非線形最小二乗法の注意事項は補足に記しました．

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

$$\begin{aligned}\Delta f &= f(a_0+\Delta a_1, b_0+\Delta b_1, c_0+\Delta c_1, d_0+\Delta d_1) \\ &\quad -f(a_0, b_0, c_0, d_0) \\ &= \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial a}\Delta a_1 + \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial b}\Delta b_1 \\ &+ \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial c}\Delta c_1 + \frac{\partial f(a_0, b_0, c_0, d_0)}{\partial d}\Delta d_1\end{aligned} \tag{1.9}$$

となります．(1.8) 式の偏微分は計算可能です．'DATA101' のデータは $t=1$ から 256 までの時刻に対応したデータ点 $f_1 \sim f_{256}$ とします．各測定値とモデル関数から予想される値との差 $\Delta f_1 \sim \Delta f_{256}$ は，

$$\begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} = J \begin{pmatrix} \Delta a_1 \\ \Delta b_1 \\ \Delta c_1 \\ \Delta d_1 \end{pmatrix} \tag{1.10}$$

となります．ここで 4 行 256 列の行列

$$J = \begin{pmatrix} \left(\frac{\partial f}{\partial a}\right)_1 & \left(\frac{\partial f}{\partial b}\right)_1 & \left(\frac{\partial f}{\partial c}\right)_1 & \left(\frac{\partial f}{\partial d}\right)_1 \\ \vdots & \vdots & \vdots & \vdots \\ \left(\frac{\partial f}{\partial a}\right)_{256} & \left(\frac{\partial f}{\partial b}\right)_{256} & \left(\frac{\partial f}{\partial c}\right)_{256} & \left(\frac{\partial f}{\partial d}\right)_{256} \end{pmatrix} \tag{1.11}$$

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

$$J^{-1} = (J^t J)^{-1} J^t \tag{1.12}$$

と表されます．従って真値からのずれは

$$\begin{pmatrix} \Delta a_2 \\ \Delta b_2 \\ \Delta c_2 \\ \Delta d_2 \end{pmatrix} = (J^t J)^{-1} J^t \begin{pmatrix} \Delta f_1 \\ \Delta f_2 \\ \vdots \\ \Delta f_{256} \end{pmatrix} \tag{1.13}$$

として計算できます．理想的には $(\Delta a_2, \Delta b_2, \Delta c_2, \Delta d_2)$ は $(\Delta a, \Delta b, \Delta c, \Delta d)$ に一致するはずですが，測定誤差と高次項のために一致しません．初期値に

比べ，より真値に近づくだけです．そこで，新たに得られたパラメータの組を新たな初期値に用いて，より良いパラメータに近付けていくという操作を繰り返します．新たに得られたパラメータと前のパラメータとの差がある誤差以下になったところで計算を打ち切り，フィッティングの終了となります．

実践

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

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

データを読み込みます．

```
> T:=readdata("DATA101",1):
> datapoint:=[seq([i,T[i]],i=1..256)]:
```

(1.8) 式のローレンツ型の関数を仮定しておきましょう．

```
> f:=a+b/(c+(x-d)^2);
> f1:=unapply(f,x);
```

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

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

(1.8) 式の関数の微分を求め，新たな関数として定義します．

```
> dfda:=unapply(diff(f,a),x);
> dfdb:=unapply(diff(f,b),x);
> dfdc:=unapply(diff(f,c),x);
> dfdd:=unapply(diff(f,d),x);
```

$$dfda := 1$$

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

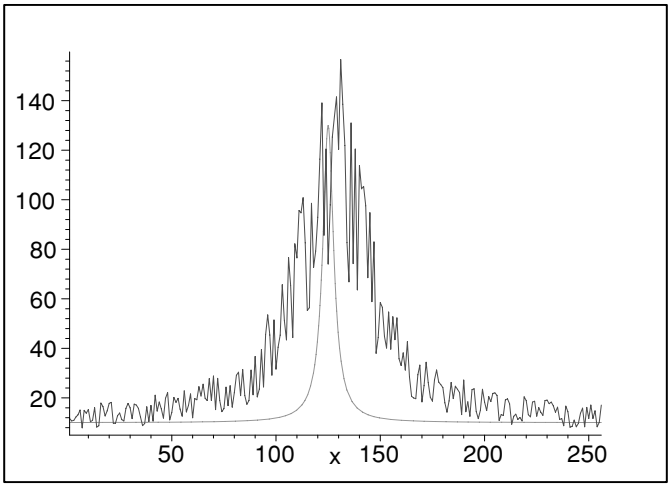
$$dfdc := x \to -\frac{b}{(c + (x-d)^2)^2}$$

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

初期値を仮定します．

```
> guess1:={a=10,b=1200,c=10,d=125};
> plot([datapoint,subs(guess1,f1(x))],x=1..256);
```

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



(1.10) 式の左辺の Δf_i を求めます．データの出力を抑制するために od の後はセミコロンではなくコロンにしてあります．デバッグや慣れないときには出力を多めに，データ数を少なめにして，関数の内側から順番に内容を確認しながら打ち込んで下さい．

```
> imax:=nops(T);
> df:=Vector([seq(evalf(subs(guess1,T[i]-f1(i))),i=1..imax)]);
```

$$imax := 256$$

$df :=$ [256 Element Column Vector Data Type : anything Storage : rectangular Order : Fortran_order]

次に (1.11) 式に従ってヤコビ行列を求めます．

```
> Jac:=Matrix(sparse,1..imax,1..4);
> for i from 1 to imax do
> Jac[i,1]:=evalf(subs(guess1,dfda(i)));
> Jac[i,2]:=evalf(subs(guess1,dfdb(i)));
> Jac[i,3]:=evalf(subs(guess1,dfdc(i)));
> Jac[i,4]:=evalf(subs(guess1,dfdd(i)));
> od:
```

$Jac :=$ [256 x 4 Matrix Data Type : anything Storage : rectangular Order : Fortran_order]

(1.12) 式の公式によるヤコビ行列の逆行列の計算です．

```
> IJac:=MatrixInverse(Multiply(Transpose(Jac),Jac)):
> tJac:=Multiply(IJac,Transpose(Jac)):
```

最後に (1.13) 式により，新たなパラメータの組を求めます．

```
> guess2:=Multiply(tJac,df);
```

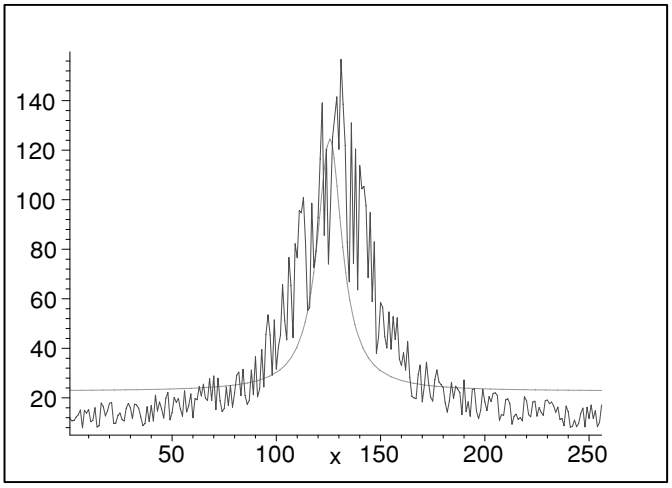
$$guess2 := \begin{bmatrix} 12.6937323162751525 \\ 4132.31700686929162 \\ 42.3461242313161322 \\ .616791465529281768 \end{bmatrix}$$

```
> guess3:={a=subs(guess1,a)+guess2[1],
> b=subs(guess1,b)+guess2[2],
> c=subs(guess1,c)+guess2[3],
> d=subs(guess1,d)+guess2[4]};
```

$guess3 := \{a = 22.69373232,\ d = 125.6167915,\ b = 5332.317007,\ c = 52.34612423\}$

求めたパラメータを用いたモデル関数と，データをプロットしてみます．
前回より近づいているのがわかるでしょう．

```
> plot({datapoint,subs(guess3,f1(x))},x=1..256);
```




```
> guess1:=guess3:
```

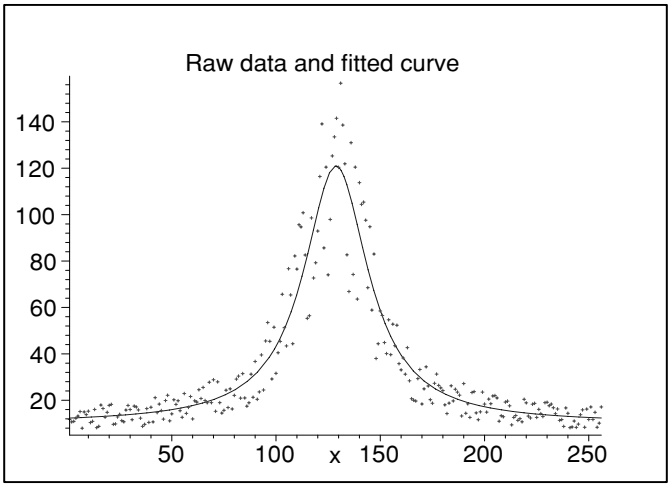
として新たな初期値とし，Δf_i 以下の操作を数回繰り返し，誤差をあらわす guess2 の値がすべて 10^{-5} 以下になった後の結果です．見やすくするために出力を少し工夫してあります．

```
> print(guess1);
```

$\{c = 353.4305681,\ a = 9.976958126,\ b = 39219.46517,\ d = 128.7272859\}$

```
> with(plots):
> F2:=plot({datapoint},x=1..256,color=red,style=point):
> F1:=plot(subs(guess1,f1(x)),x=1..256,style=line,color=blue,thickness=1
> ):
> display({F1,F2},title="Raw data and fitted curve");
```

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



補足:最小二乗法に関するメモ

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

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

$$y = f(x;\mathbf{a}) \tag{1.14}$$

です．最小二乗法の χ^2 関数は

$$\chi^2(\mathbf{a}) = S(\mathbf{a}) = \sum_{i=1}^{N} [y_i - f(x_i;\mathbf{a})]^2 \tag{1.15}$$

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

$$\frac{\partial S(\mathbf{a})}{\partial a_k} = -2\sum_{i=1}^{N} f(x_i;\mathbf{a})\frac{\partial f(x_i;\mathbf{a})}{\partial a_k} = 0. \tag{1.16}$$

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

$$\nabla S(\mathbf{a}) = -2J^t(\mathbf{a})f(y;\mathbf{a}) = 0 \tag{1.17}$$

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

$$\nabla^2 S(\mathbf{a})\Delta\mathbf{a}^{(k)} = -\nabla S(\mathbf{a}^{(k)}) \tag{1.18}$$

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

$$\frac{\partial^2 S(\mathbf{a})}{\partial a_k \partial a_{k'}} = -2\left\{\sum_{i=1}^{N} \frac{\partial f(x_i;\mathbf{a})}{\partial a_k}\frac{\partial f(x_i;\mathbf{a})}{\partial a_{k'}} + \sum_{i=1}^{N} \frac{\partial^2 f(x_i;\mathbf{a})}{\partial a_k \partial a_{k'}} f(x_i;\mathbf{a})\right\} \quad (1.19)$$

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

$$\left\{J^t\left(\mathbf{a}^{(k)}\right) J\left(\mathbf{a}^{(k)}\right) + \sum_{i=1}^{N} f\left(x_i;\mathbf{a}^{(k)}\right) \nabla^2 f\left(x_i;\mathbf{a}^{(k)}\right)\right\} \Delta\mathbf{a}^{(k)} = J^t\left(\mathbf{a}^{(k)}\right) f\left(x_i;\mathbf{a}^{(k)}\right) \quad (1.20)$$

が得られます．鉤括弧内の第 2 項を無視すると，先程の（1.10）式の線形的な関係が現れます．この無視する項は，線形の場合には確かに現れないし，1 次の項に比べて小さいと考えられます．さらに $f(x_i;\mathbf{a})$ を掛けて和を取っているため，それらがお互いにキャンセルしてくれる可能性があります．

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