14　非線形関数の最小２乗法

この章では、データへの非線形関数の当てはめ (フィッティング (fitting)) について述べる。数値計算法概論としては「応用」の部類に入るので全員が理解する必要は無いが、学生実験のデータ整理等で必要のあるものや興味のあるものは、理解して積極的に活用して欲しい。

14.1　線形化して解く方法

前章で述べたように、線形な場合は既に手法が確立している。したがって、問題を 1 次微係数を用いて線形化してしまえば既存の方法がそのまま使える。

具体的には、m 個の変数 $a_1, \ldots, a_m$ を用いて表される式 $f(x;a)$(a は $a_1, \ldots, a_m$ を表す) にデータを当てはめる場合を考える。ここで、真の解 $a_1, \ldots, a_m$ に近い近似解 $a_1^{(0)}, \ldots, a_m^{(0)}$ が分かっているとする。

$$\begin{aligned} a_1 &= a_1^{(0)} + \delta a_1 \\ &\vdots \\ a_k &= a_m^{(0)} + \delta a_m \end{aligned} \tag{14.1}$$

関数 $f(x;a)$ を $a_1^{(0)}, \ldots, a_m^{(0)}$ の近傍で Taylor 展開し、1 次の項までとり線形化する。ここで、

$$f(x;a) \approx \underbrace{f(x;a^{(0)})}_{f_0(x)} + \sum_{j=1}^{m} \underbrace{\left.\frac{\partial f(x;a)}{\partial a_j}\right|_{a=a^{(0)}}}_{g_j(x)} \delta a_j \tag{14.2}$$

と、あらためて $f_0(x)$ と $g_j(x)$ を定義する。

式 (14.2) を見ると $f(x;a)$ は、m **個の変数** $\delta a_j (j = 1, \ldots, m)$ **の線形 1 次式**になっており、線形関数の最小 2 乗法を用いることができる。具体的には、最小 2 乗法の条件は

$$\frac{\partial \chi^2}{\partial a_j} = \sum_{i=1}^{n} \frac{2(y_i - f(x_i;a))}{\sigma_i^2} \frac{\partial f(x_i;a)}{\partial a_j} = 0 \quad (j = 1, \ldots, m) \tag{14.3}$$

であるが、

$$f(x_i;a) = f_0(x_i) + \sum_{k=1}^{m} g_k(x_i)\delta a_k, \quad \frac{\partial f(x_i;a)}{\partial a_j} = g_j(x_i) \tag{14.4}$$

であるので、

$$\frac{\partial \chi^2}{\partial a_j} = 2\underbrace{\sum_{i=1}^{n} \frac{(y_i - f_0(x_i))}{\sigma_i^2} g_j(x_i)}_{\beta_j} - 2\sum_{k=1}^{m} \underbrace{\left[\sum_{i=1}^{n} \frac{g_j(x_i)g_k(x_i)}{\sigma_i^2}\right]}_{\alpha_{jk}} \delta a_k = 0 \quad (j = 1, \ldots, k) \tag{14.5}$$

と表すことができる。したがって、m 個の変数 $\delta a_j (j = 1, \ldots, m)$ を求める式は以下の行列形式

$$\beta_j = \sum_{k=1}^{m} \alpha_{jk}\delta a_k \quad (j = 1, \ldots, m) \tag{14.6}$$

となる。したがって、対称行列 α_{jk} の逆行列を $(\alpha^{-1})_{jk}$ で表すと、変数 δa_j は

$$\delta a_j = \sum_{k=1}^{m} (\alpha^{-1})_{jk} \beta_k \tag{14.7}$$

により求めることができる。

式 (14.1) から得られる新たな解 $a_1, \ldots, a_m$ を、改めて初期値 $a_1^{(0)}, \ldots, a_m^{(0)}$ として上記の手順を繰り返せば、**逐次漸近的**に真の解に近付いていく。これが、「線形化して解く」手法の基本的な考え方である。

この方法は理解が容易 (従ってプログラムも容易) ではあるが、初期値が真の値に近い場合を除き一般に収束が遅く実用的ではない。そこで実際には後述のような別の方法が、非線形関数の最小 2 乗法では用いられる。

14.2 Newton 法

関数 $f(x)$ の根、すなわち

$$f(a) = 0 \tag{14.8}$$

となる解 a を数値計算により求める方法として広く知られているものに**Newton 法**がある。初期値 $a^{(0)}$ が与えられたとき、曲線 $y = f(x)$ に $x = a^{(0)}$ で接する直線

$$y = \left.\frac{\partial f(x)}{\partial x}\right|_{x=a^{(0)}} (x - a^{(0)}) + f(a^{(0)}) \tag{14.9}$$

と $y = 0$(x 軸) の交点を改めて初期値 $a^{(0)}$ として、逐次近似的に a を求めていく方法である。

前節の、「線形化して解く」解法においても、

$$\frac{\partial \chi^2}{\partial a_j} = 0 \quad (j = 1, \ldots, m) \tag{14.10}$$

の根 $a_1, \ldots, a_m$ を求める点に関しては同じ方法を用いていることに注意せよ。このことから、「線形化して解く」解法のことを Newton 法と呼ぶことが多い[1]。

14.3 Marquardt 法

前述のように Newton 法には収束が遅く実用的ではないという欠点がある。現在、非線形関数の最小 2 乗法として実用的に使われているものに、

Simplex 法 関数の微分を使わない

Davidon-Fletcher-Powell(DFP) 法 関数の微分を使う

Marquardt 法 関数の微分を使う

などがある。ここで、Simplex 法と DFP 法は、各々線形計画法及び非線形計画法の代表的な解法である。どの方法が最適であるかは一概には言えない (万能な解法は無い) が、

[1] 但し、Newton 自身がこの方法を提案した訳ではない。

1. データを当てはめる関数の微分が簡単に計算できるか
   (数値計算ではなく、解析的に計算できる方が良い)

2. データを当てはめる関数が滑らかか
   (細かな凹凸や発散がないか)

といったことが判断材料となる。

ここでは多くのケースで優れた収束性を示す Marquardt 法[2]を紹介する。他の解法については、数値計算の教科書などを参照のこと。

式 (14.2) のように線形化できる場合に戻って χ^2 の微分を考える。式 (14.5) と (14.6) から、

$$\begin{aligned} -\frac{1}{2}\frac{\partial\chi^2}{\partial a_j} &= \underbrace{\sum_{i=1}^{n}\frac{(y_i - f(x_i;a))}{\sigma_i^2}g_j(x_i)}_{\beta_j} \\ &= \sum_{k=1}^{m}\alpha_{jk}\delta a_k \quad (j=1,\ldots,m) \end{aligned} \tag{14.11}$$

である。α_{jk} の対角成分の働きを見るために、非対角成分が小さく無視できるとする。すると、

$$-\frac{1}{2}\frac{\partial\chi^2}{\partial a_j} \simeq \alpha_{jj}\delta a_j \tag{14.12}$$

となる。ここで改めて $b_j \equiv a_j\sqrt{a_{jj}}$ を定義すると

$$\frac{\partial\chi^2}{\partial a_j} = \frac{\partial\chi^2}{\partial b_j}\frac{\partial b_j}{\partial a_j} = \frac{\partial\chi^2}{\partial b_j}\sqrt{a_{jj}} = -2a_{jj}\delta a_j \tag{14.13}$$

であるので

$$\delta b_j = \delta a_j\sqrt{a_{jj}} = -\frac{1}{2}\frac{\partial\chi^2}{\partial b_j} \tag{14.14}$$

を得る。ベクトル表記で書けば、

$$\delta\boldsymbol{b} = -\frac{1}{2}\boldsymbol{\nabla}\chi^2 \tag{14.15}$$

となる。

以上から、b_j は χ^2 最も急勾配で減少する方向に変化 (最速降下) していくことが分かる。そこで、a_jk の対角成分を強調するパラメータ λ を導入し、

- フィッティングの最初の段階 ($a_j^{(0)}$ が真の解に近くない時) には λ を大きく採り、対角成分を強調して χ^2 の勾配を最速降下させる。その代わり、a_{jj} が大きいので変化量 δa_j は小さい。

- 真の解に近付くにつれ $\lambda \to 0$ とし、Newton 法に近付けて一気に収束させる。

というのが Marquardt 法のアイディアである。

式 (14.2) に従って f_0 と g_j を定義すると、アルゴリズムは以下のようになる。

1. 初期値 $a_j^{(0)}$ を使って χ^2 を計算する。この値を χ_0^2 とおく

[2]フランス人の名前なので、語尾の t は発音しない。したがってカタカナで書くと「マルカール」となる。

2. $\alpha_{jk} = \sum_{i=1}^{n} g_j g_k/\sigma_i^2$ と $\beta_j = \sum_{i=1}^{n}(y_i - f_0)g_j/\sigma_i^2$ を計算
3. $\alpha'_{jk} = (1+\lambda\delta_{jk})\alpha_{jk}$($\delta$ はクロネッカーのデルタ) の逆行列を使って、$\delta a_k = \sum_j (\alpha'^{-1})_{kj}\beta_j$ を求める
4. $a_j = a_j^{(0)} + \delta a_j$ を使って χ^2 を計算
5. もし $\chi^2 > \chi_0^2$ ならば、λ を 10 倍し、3. に戻ってやり直す
6. もし $\chi^2 - \chi_0^2 < \epsilon\chi^2$ ならば計算を終了する。ここで ϵ は収束条件である
7. $a_j = a_j^{(0)}$、$\chi_0^2 = \chi^2$ とし、λ を 1/10 倍して 2. に戻る

パラメータ a_j の誤差 Δa_j は、求めた a_j が真の値に十分近ければ線形化は妥当であると仮定すると、誤差行列の対角項により与えられる。

Marquardt 法には、

- λ の与え方が直観的である
  (10 倍や 1/10 倍が妥当なのか?初期値はどうとるか?)
- 収束の判定が甘い

等の批判もあり、これらの点を改良した修正 Marquardt 法も提案されている。詳しくは数値計算の本を参考にして欲しい。

練習

図 14.1 のデータを使って、データの非線型関数への当てはめを行ってみよう。データは、

```
/home/teacher/z6wt01in/SAMPLE/gauss_bg.dat
```

として置いてある。当てはめる関数は、このデータが Gauss 関数と 1 次式の和

$$f(x) = a_1 \exp\left[-\frac{1}{2}\left(\frac{y-a_2}{a_3}\right)\right] + a_4 x + a_5 \tag{14.16}$$

で表されると仮定し、5 つの変数 $a_1, \ldots, a_5$ を最適化 (χ^2 を最小に) する。

以下のサンプルプログラムでは、逆行列の計算には既出の `matinv_jordan` を用い、実数型変数は計算精度を考慮して `real*8` を用いている。プログラムは、

```
/home/teacher/z6wt01in/SAMPLE/
```

に

```
fit_arb_func.f marquardt.f fit_func.f
```

として置いてある。変数の初期値とデータ (ファイル名) はコマンドラインから

```
% fit_arb_func 150 30 5 1 5 < gauss_bg.dat
```

というように与えるようになっている。

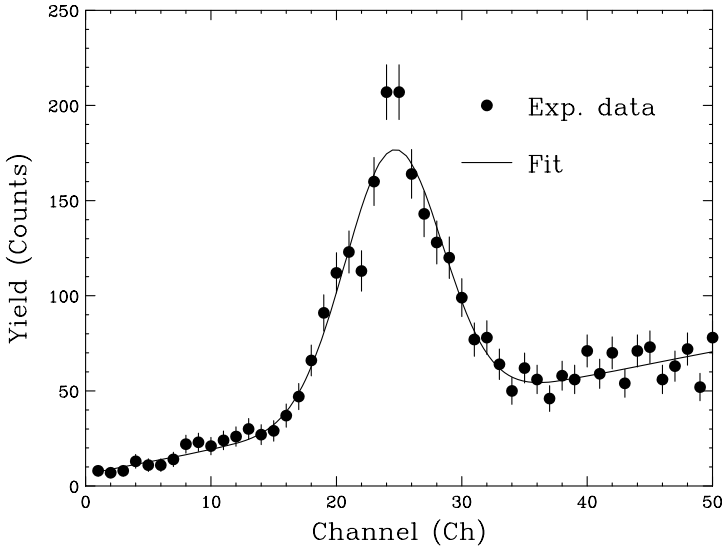

図 14.1: 実験データに対する Gauss 関数+1 次式で表される関数の当てはめの結果。

- プログラム中の iargc がコマンドラインの引数の数を返し、getarg が具体的に引数の内容を得る為の関数である。データは標準入力から得るように書かれているので、ファイルから読ませるには<を使ってリダイレクトすればよい。
- サブルーチン marquardt の出力 (write 文) で論理機番 (Logical Unit Number) に 0 を用いている場所がある。論理機番 0 は標準エラー出力 (Standard Error Ourput(略称 stderr)) を意味し、通常は画面が割り当てられている。write 文の場合、論理機番*は標準出力の画面が割り当てられている。従って、

  ```
  % fit_arb_func 150 30 5 1 5 < gauss_bg.dat > gauss_bg.out
  ```

  と標準出力をファイル gauss_bg.dat にリダイレクトすると、標準出力 (write(*,*)) はファイルに、標準エラー出力 (write(0,*)) は画面にと出力を振り分けることができる。

サンプルプログラム (fit_arb_func.f)

```
      program fit_arb_func
      implicit none
c const:
      integer NMAX,MMAX                 ! データと
      parameter(NMAX=500,MMAX=10)       !    パラメータの最大数
      logical FOREVER
      parameter(FOREVER=.true.)
```

```
c local:
      real*8  x(NMAX),y(NMAX),dy(NMAX)  ! データ用配列
      real*8  f(NMAX),f0(NMAX)          !   同上
      real*8  a(MMAX),a0(MMAX)          ! パラメータ用配列
      real*8  g(MMAX*NMAX),g0(MMAX*NMAX)!   同上
      real*8  alpha(MMAX,MMAX),beta(MMAX)
      real*8  delta(MMAX)
      integer n/0/,m                    ! データとパラメータ数
      integer i                         ! ループ用
      character key*20                  ! コマンドライン引数
c function
      integer iargc                     ! コマンドライン引数の数
c begin:
      m = iargc()                       ! 引数の数=パラメータ数
      do i=1,m                          ! パラメータ初期値読み込み
         call getarg(i,key)
         read(key,*) a0(i)
      end do
      do while (FOREVER)
         read(*,*,end=10000)            ! データの読み込み
     &        x(n+1),y(n+1),dy(n+1)
         n = n+1
      end do
10000 call marquardt(n,x,y,dy,f,f0,m,a,a0,g,g0,
     &     alpha,beta,delta)
      stop
      end
```

サンプルプログラム (marquardt.f)

```
      subroutine marquardt(n,x,y,dy,f,f0,m,a,a0,g,g0,
     &     alpha,beta,delta)
      implicit none
c const:
      integer NLOOP                     ! ループの最大数
      parameter(NLOOP=50)
      real*8  EPSILON                   ! 収束条件
      parameter(EPSILON=1.0D-6)
c input/output:
      integer n,m                       ! データとパラメータ数
      real*8  x(n),y(n),dy(n)           ! データ用配列
      real*8  f(n),f0(n)                !   同上
```

```
      real*8  a(m),a0(m)              ! パラメータ用配列
      real*8  g(m,n),g0(m,n)          !    同上
c local
      real*8  chi2,chi20/0.0/         ! \chi^2 と初期値
      real*8  alpha(m,m),beta(m)
      real*8  delta(m)
      real*8  lambda/10.0/            ! 加速パラメータ
      integer i,j,k,loop              ! ループ用
c begin:
      call fit_func(n,m,x,a0,f0,g0)   ! 関数の初期値
      do i=1,n                        ! \chi^2 の初期値
         chi20=chi20+(y(i)-f0(i))**2/dy(i)**2
      end do
      do loop=1,NLOOP
         do j=1,m
            beta(j)=0.0               ! \beta_j の計算
            do i=1,n
               beta(j)=beta(j)+g0(j,i)*(y(i)-f0(i))/dy(i)**2
            end do
            do k=j,m
               alpha(j,k)=0.0
               do i=1,n
                  alpha(j,k)=alpha(j,k)
     &                 +g0(j,i)*g0(k,i)/dy(i)**2
               end do
               alpha(k,j)=alpha(j,k)
            end do                    ! \alpha_jj の強調
            alpha(j,j)=(1.0+lambda)*alpha(j,j)
         end do
         call matinv_jordan(m,alpha)
         do k=1,m
            delta(k)=0.0
            do j=1,m
               delta(k)=delta(k)+alpha(k,j)*beta(j)
            end do
         end do
         do j=1,m
            a(j)=a0(j)+delta(j)       ! 新しいパラメータ
         end do
         call fit_func(n,m,x,a,f,g)   ! 新しい関数値
         chi2=0.0
         do i=1,n                     ! 新しい\chi^2 の計算
```

```
            chi2=chi2+(y(i)-f(i))**2/dy(i)**2
         end do
         if (chi2.gt.chi20) then       ! もし\chi^2 が増加していたら
            lambda=lambda*10.0         ! \lambda を 10 倍してやり直し
         else
            if ((chi20-chi2).lt.chi2*EPSILON)
     &           goto 10000            ! 収束条件を満たしたら終了
            lambda=lambda/10.0         ! 満たしていなかったら
            do j=1,m                   ! 1. lambda を 1/10 にする
               a0(j)=a(j)              ! 2. 求めた解を次の初期値にする
            end do
            do i=1,n
               f0(i)=f(i)
               do j=1,m
                  g0(j,i)=g(j,i)
               end do
            end do
            chi20=chi2
         endif
      end do
10000 if (loop.le.NLOOP) then
         write(0,'(A,I3,A)') 'After',loop-1,
     &        ' iterations the fit converged.'
      else
         write(0,'(A,I3,A)') 'The fit NOT converged within',loop-1,
     &        ' iterations.'
      endif
      write(0,'(A,I1,A,F8.3,A,F8.3)')
     &     ('a[',j,'] =',a(j),' +/-',sqrt(alpha(j,j)),j=1,m)
      write(*,'(F8.1,F10.4,F8.2,F10.4)')
     &     (x(i),y(i),dy(i),f(i),i=1,n)
      write(0,'(A,F7.3)') 'Normalized chi^2 =',chi2/(n-m)
      return
      end
```

サンプルプログラム (fit_func.f)

```
      subroutine fit_func(n,m,x,a,f,g)
      implicit none
c
      integer n,m,i
      real*8  x(n),a(m),f(n),g(m,n),u,v
```

```
do i=1,n
   u=(x(i)-a(2))/a(3)
   v=exp(-u*u/2.0)
   f(i)=a(1)*v+a(4)*x(i)+a(5)
   g(1,i)=v
   g(2,i)=a(1)/a(3)*u*v
   g(3,i)=g(2,i)*u
   g(4,i)=x(i)
   g(5,i)=1.0
end do
return
end
```

計算・メモ用余白