

Table of Contents

- 1 pythonによる最小2乗法
- 1.1 python code
- 2 最小2乗法の原理
- 3 χ^2 の極小値から(2変数の例)
- 4 正規方程式(Normal Equations)による解
- 4.1 python codeによる具体例
- 5 特異値分解(Singular Value Decomposition)による解
- 6 scipy.linalg.lstsq
- 6.1 正規方程式によるもの . . .
- 7 2次元曲面へのフィット
- 7.1 具体例
- 8 課題
- 8.1 1次元の線形最小二乗法
- 8.2 2次元の最小二乗フィット

線形最小2乗法 (LeastSquareFit)

file:/Users/bob/Github/TeamNishitani/jupyter_num_calc/leastsquarefit
https://github.com/daddygongon/jupyter_num_calc/tree/master/notebooks_p
cc by Shigeto R. Nishitani 2017-19

pythonによる最小2乗法

fitting

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。

python code

$x = [1,2,3,4]$, $y=[0,5,15,24]$ に $y = a_0 + a_1 x$ をフィットする例を考える。pythonのcodeは以下の通り。

In [1]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

```
def f(x, a0, a1): #, a2):
    return a0 + a1*x #+ a2*x**2
```

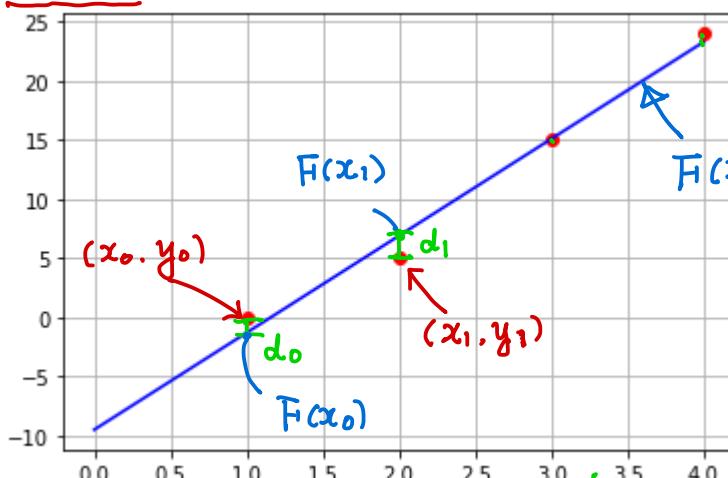
```
xdata = np.array([1,2,3,4])
ydata = np.array([0,5,15,24])
plt.plot(xdata,ydata, 'o', color='r')
```

```
params, cov = curve_fit(f, xdata, ydata)
print(params)
```

```
x = np.linspace(0,4,20)
y = f(x,params[0],params[1]) #,params[2])
plt.plot(x,y, color='b')
```

```
plt.grid()
plt.show()
```

[-9.5 8.2]



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

$$\sum_i d_i^2 = F(x_i) - y_i \\ = a_0 + a_1 x_i - y_i \\ = \sum_i (a_j x_j (x_i) - y_i)$$

$$\sum (\underline{a_1} \underline{x_i}^2 + \underline{a_0} \underline{x_i} - \underline{x_i} \underline{y_i}) = 0$$

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、

$$\textcircled{O} \quad \chi^2 = \sum_i d_i^2 = \sum_i (a_0 + a_1 x_i - y_i)^2$$

9565文

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、

$$\chi^2 = \sum_i \left(\underline{a_0}^2 + \underline{a_1}^2 \underline{x_i}^2 + \underline{y_i}^2 + 2 \underline{a_0} \underline{a_1} \underline{x_i} - 2 \underline{a_1} \underline{x_i} \underline{y_i} - 2 \underline{a_0} \underline{y_i} \right)$$

$$\sum (\underline{a_1} \underline{x_i}^2 + \underline{a_0} \underline{x_i} - \underline{x_i} \underline{y_i})$$

$$\underline{a_1} \underline{\sum x_i^2} + \underline{a_0} \underline{\sum x_i} - \underline{\sum x_i y_i} = 0$$

9584
半田 (Astro)
9076
濱垣
(ヤハギ)

$$\frac{\partial}{\partial a_0} \chi^2 = \frac{\sum (\underline{a_0} + \underline{a_1} \underline{x_i} - \underline{y_i})}{N \underline{a_0} + \underline{a_1} \sum \underline{x_i} - \sum \underline{y_i}} = 0$$

$$\frac{\partial}{\partial a_1} \chi^2 = \frac{\sum (\underline{a_1} \underline{x_i}^2 + \underline{a_0} \underline{x_i} - \underline{x_i} \underline{y_i})}{\underline{a_1} \sum \underline{x_i}^2 + \underline{a_0} \sum \underline{x_i} - \sum \underline{x_i} \underline{y_i}} = 0$$

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。

χ^2 の極小値から(2変数の例)

先ほどの例をもとに何をしているか別の角度からみる。データを関数に入れて sum をとると次のような関数が得られる。

```
In [2]: %matplotlib notebook
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np
from sympy import *

a0, a1 = symbols('a0, a1')
def func(x):
    return a0+a1*x

def z_surf(xx,yy):
    sum = 0
    for i in range(0,4):
        tmp = xx[i] - yy[i]
        sum = sum + tmp**tmp
    return sum

x1 = np.array([1,2,3,4])
y1 = np.array([0,5,15,24])
eq = z_surf(func(x1),y1)
print(eq)
print(expand(eq))
```

$$(a_0 + a_1)^{**2} + (a_0 + 2*a_1 - 5)^{**2} + (a_0 + 3*a_1 - 15)^{**2} + (a_0 + 4*a_1 - 24)^{**2} \\ 4*a0^{**2} + 20*a0*a1 - 88*a0 + 30*a1^{**2} - 302*a1 + 826$$

これは a_0, a_1 を変数とする関数となっている。データ点 (x_i, y_i) はすでに数値を持っており、未知なのは a_0, a_1 である。そうすると $\chi^2(a_0, a_1)$ つまり a_0, a_1 をパラメータとして、 χ^2 の値を z 軸とする 3 次元関数とみなすことができて、それを plot すると次の通り。

```
In [3]: %matplotlib notebook
def f(a0, a1):
    return 4*a0**2 + 20*a0*a1 - 88.0*a0 + 30*a1**2 - 302.0*a1 + 826.0

a0 = np.arange(-20, 20, 5)
a1 = np.arange(-20, 20, 5)
A0, A1 = np.meshgrid(a0, a1)
Z1 = f(A0, A1)

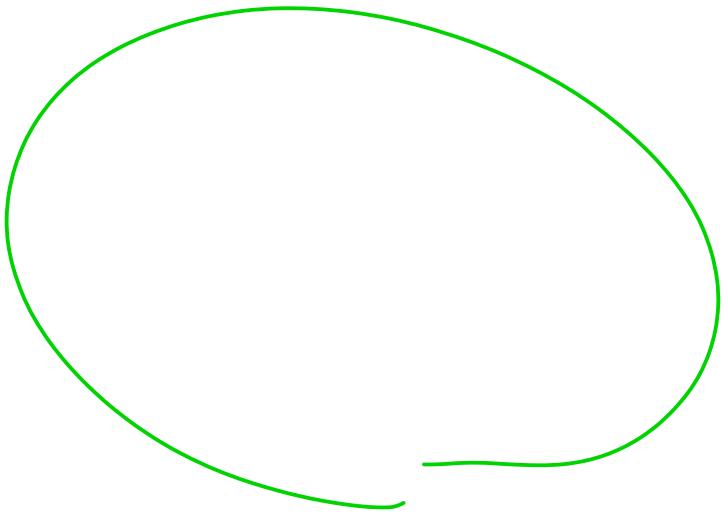
fig = plt.figure()
```

```

plot3d = Axes3D(fig)
plot3d.plot_surface(A0, A1, Z1)

plt.show()

```



$a_0=-9.5, a_1=8.2$ あたりに最小値があるはずですが... 見にくいよね。こういうのをsteepな関数って言いますが、それが後で述べる特異値分解を使わなければいけない理由です。値が微妙でなければ、微分して0において、連立方程式とみなして解くことができます。それが、上の「最小2乗法の原理」で述べた解法になります。

$$F(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \dots$$

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(linear)につないだ関数を

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

となる。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

$$\chi^2 = \sum_{i=1}^N (F(x_i) - y_i)^2 = \sum_{i=1}^N \left(\sum_{k=1}^M a_k X_k(x_i) - y_i \right)^2 \quad \text{ただし } \sum_j (a_j X_j(x_i) - y_i)$$

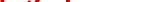
と求めることができる。この関数を、 a_k を変数とする関数とみなす。この関数が最小値を取るのは、 χ^2 を M 個の a_k で偏微分した式がすべて0となる場合である。これを実際に求めてみると、

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

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

$$\underline{A}_{ij} = \underline{X}_j(\underline{x}_i)$$

とする行列 A を導入する。この行列は、
 行列 

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

↑
N行
↓

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、

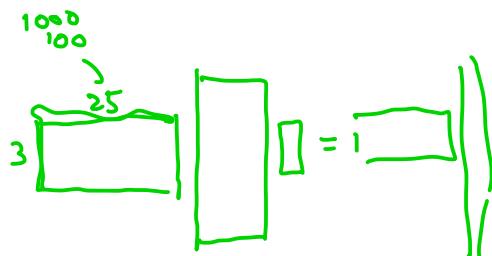
$$\cancel{*} \quad A^t \cdot A \cdot \underline{a} = A^t \cdot y$$

で与えられる。 A^t は行列 A の転置(transpose)

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

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

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



$$\boxed{A^T \cdot A} \quad \alpha = \boxed{A^T y}$$

$$A \vec{x} = \vec{b}$$

また、
 9619 $\times 2$ $M = 3, N = 25$ 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。

python codeによる具体例

N M
4点のデータに対して、2次関数つまり3個のパラメータでfitする。その場合、デザイン行列は4行3列になる。

```
In [4]: import numpy as np  
from pprint import pprint  
import scipy.linalg as linalg  
  
xdata=np.array([1,2,3,4])  
ydata=np.array([0.5,15,24])
```

```

def ff(x,i):
    return x**i

Av = np.zeros([4,3])
for i in range(0,3):
    for j in range(0,4):
        Av[j][i]=ff(xdata[j],i)

pprint(Av)

Ai = linalg.inv(np.dot(np.transpose(Av),Av))
b = np.dot(np.transpose(Av),ydata)
np.dot(Ai,b)

```

A 2
array([[1., 1., 1.],
 [1., 2., 4.], 4
 [1., 3., 9.],
 [1., 4., 16.]])
Out[4]: array([-4.5, 3.2, 1.]) ①

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

正規方程式を解くときには、少し注意が必要である。単純な逆行列による解法では、間違った答えに行き着く可能性が高い。より信頼性の高い方法では、特異値分解を用いる。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NumRecipieを参照せよ。

```

In [5]: import numpy as np
from pprint import pprint
import scipy.linalg as linalg

xdata=np.array([1,2,3,4])
ydata=np.array([0,5,15,24])

#def f(x,a1,a2,a3):
#    return a1+a2*x+a3*x**2
def ff(x,i):
    return x**i

Av = np.zeros([4,3])
for i in range(0,3):
    for j in range(0,4):
        Av[j][i]=ff(xdata[j],i)
m,n = Av.shape
pprint(Av)

U, s, Vs = linalg.svd(Av)
pprint(s)
S = linalg.diagsvd(s,m,n)
pprint(S)
iS = np.zeros([3,4])
for i in range(0,3):
    iS[i][i] = 1.0/s[i]
print(iS)
left = np.dot(np.transpose(Vs),iS)
right= np.dot(np.transpose(U),ydata)
np.dot(left,right)
print(right)

```

956
足立

```

array([[ 1.,  1.,  1.],
       [ 1.,  2.,  4.],
       [ 1.,  3.,  9.],
       [ 1.,  4., 16.]])
→ array([19.62136402, 1.71206987, 0.26625288])
array([[19.62136402, 0.        , 0.        ],
       [ 0.        , 1.71206987, 0.        ],
       [ 0.        , 0.        , 0.26625288],
       [ 0.        , 0.        , 0.        ]])
[[0.05096486 0.        0.        ]
 [0.        0.58408831 0.        0.        ]
 [0.        0.        3.75582793 0.        ]]
→ [-28.61521162  1.83564305 -1.41424125  1.34164079]

```

scipy.linalg.lstsq

scipy.linalg.lstsqによるcurve fitについて紹介しておく。あらかじめ、デザイン行列 A を作つておいて、これを

$$Ax = b$$

とみなした場合の x について解く。

```
In [6]: import numpy as np
from pprint import pprint
import scipy.linalg as linalg

xdata=np.array([1,2,3,4])
ydata=np.array([0,5,15,24])

def ff(x,i):
    return x**i

Av = np.zeros([4,3])
for i in range(0,3):
    for j in range(0,4):
        Av[j][i]=ff(xdata[j],i)
print(Av)

c, resid, rank, sigma = linalg.lstsq(Av, ydata)
print(c,resid,rank,sigma)
```

```

[[ 1.  1.  1.]
 [ 1.  2.  4.]
 [ 1.  3.  9.]
 [ 1.  4. 16.]]
[-4.5  3.2  1. ] 1.7999999999999976 3 [19.62136402 1.71206987 0.26625288]
```

正規方程式によるのも。 . .

lstsqは正規方程式によるのと同じかな。SVDかも。

```
In [7]: Ai = linalg.inv(np.dot(np.transpose(Av),Av))
b = np.dot(np.transpose(Av), ydata)
np.dot(Ai,b)
```

Out[7]: array([-4.5, 3.2, 1.])

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、

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

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python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, \dots, 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。

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、scipy.linalgのlinalg.lstsqでfitしている。正規方程式による解法、つまり逆行列で求めた値と一致していることを確認してください。

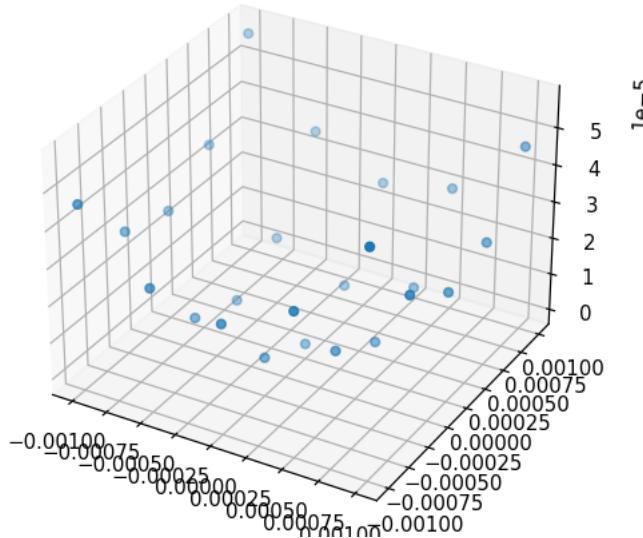
```
In [8]: import numpy as np
z = np.array([0.000046079702088, 0.000029479057275,
    0.000025769637830, 0.000034951410953, 0.000057024385455, 0.000029485453808
    0.000011519913869, 0.000006442404299, 0.000014252898382, 0.000034951410953
    0.000025769637773, 0.000006442404242, 0.000000000000057, 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 = []
for i in range(-2,3):
    for j in range(-2,3):
        x.append(i*0.0005)
        y.append(j*0.0005)]
print(x)
print(y)
```

```
[-0.001, -0.001, -0.001, -0.001, -0.001, -0.0005, -0.0005, -0.0005, -0.0005, 0.0, 0.
0, 0.0, 0.0, 0.0005, 0.0005, 0.0005, 0.0005, 0.001, 0.001, 0.001, 0.001, 0.001]
[-0.001, -0.0005, 0.0, 0.0005, 0.001, -0.001, -0.0005, 0.0, 0.0005, 0.001, -0.001, -0.0005,
0.0, 0.0005, 0.001, -0.001, -0.0005, 0.0, 0.0005, 0.001, -0.001, -0.0005, 0.0, 0.0005, 0.00
1]
```

```
In [9]: %matplotlib notebook
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure()
plot3d = Axes3D(fig)
plot3d.scatter3D(np.array(x),np.array(y),z)

plt.show()
```



```
In [10]: from pprint import pprint
import scipy.linalg as linalg
```

```
n = z.size
n_j = 6
bb=np.zeros([n])
A=np.zeros([n,n_j])
for i in range(0,n):
    A[i,0]=1
    A[i,1]=x[i]
    A[i,2]=y[i]
    A[i,3]=x[i]*y[i]
    A[i,4]=x[i]**2
    A[i,5]=y[i]**2
    bb[i]=z[i]
```

} デザイニ行列 A^t
左辺 b

$$A_i \cdot b = (A^t \cdot A)^{-1} \cdot (A^t \cdot b)$$

```
c, resid, rank, sigma = linalg.lstsq(A, bb)
pprint(c)

Ai = linalg.inv(np.dot(np.transpose(A),A))
b = np.dot(np.transpose(A),bb)
np.dot(Ai,b)
```

```
array([-9.18521222e-13, -6.39644676e-06, 6.39644220e-06, -5.45955358e+00,
       2.57696284e+01, 2.57696284e+01])
```

```
Out[10]: array([-9.18525713e-13, -6.39644676e-06, 6.39644220e-06, -5.45955358e+00,
       2.57696284e+01, 2.57696284e+01])
```

$$a_1 + a_2 x + a_3 y + a_4 xy + a_5 x^2 + a_6 y^2$$

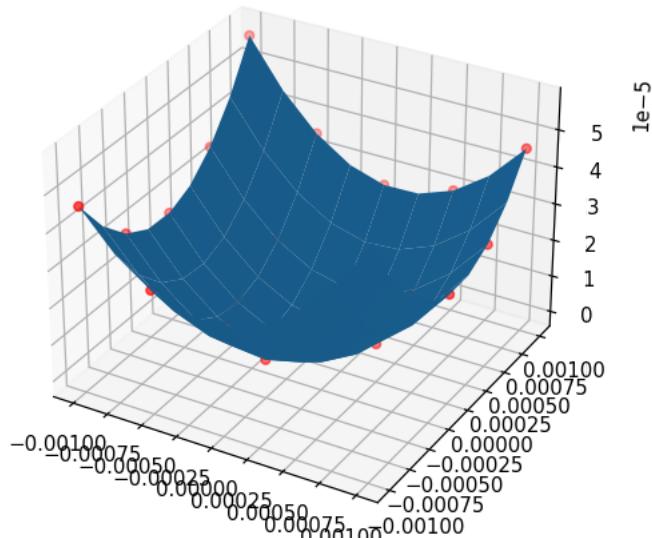
```
In [11]: def z_surf(xx,yy):
```

```
    val = c[0] + c[1]*xx + c[2]*yy
    val += c[3]*xx*yy + c[4]*xx**2
    val += c[5]*yy**2
    return val
```

```
x1 = np.arange(-0.001, 0.00125, 0.00025)
y1 = np.arange(-0.001, 0.00125, 0.00025)
X, Y = np.meshgrid(x1, y1)
Z1 = z_surf(X,Y)
```

```
fig = plt.figure()
plot3d = Axes3D(fig)
plot3d.scatter(np.array(x), np.array(y), z, color='r')
plot3d.plot_surface(X, Y, Z1)
```

```
plt.show()
```



課題

1次元の線形最小二乗法

次の4点のデータを $y = a_0 + a_1x + a_2x^2$ で近似せよ(2006年度期末試験).

```
xdata = np.array([1, 2, 3, 4])
ydata = np.array([1, 3, 4, 10])
```

2次元の最小二乗フィット

以下のデータを

$$f(x, y) = a_0 + a_1x + a_2y + a_3xy$$

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
```

結果は以下の通り。鞍点になってます。



In []: