最小二乗法

1　実験データを取り扱おう

みなさんの多くは三年生になると化学実験で実験データをプロットすることになる。例えば時間と放射性量の関係であったり、pH と反応速度の関係であったり、温度とゴムの長さの関係であったり、時間と温度の関係であったり、様々な関係をプロットすることになる。

実験値には往々にして誤差が含まれるものなので、今回は非常に簡単なノイズ付き線形データを扱う。このデータは

$$y = -2x + 50 + \epsilon$$

という式から生成したデータであり、ここでϵはデータに含まれるノイズを示しており、平均 0、分散 10 の正規分布から生成したノイズである。

このデータは図 1 のような構造をしており、「data.csv」に csv 形式で格納されている。ファイルを開いてもらえばわかるが、このファイルは 50 行のファイルで、一行が一点に対応しており、「,」の左側が x 軸の値を、「,」の右側が y 軸の値を示している。

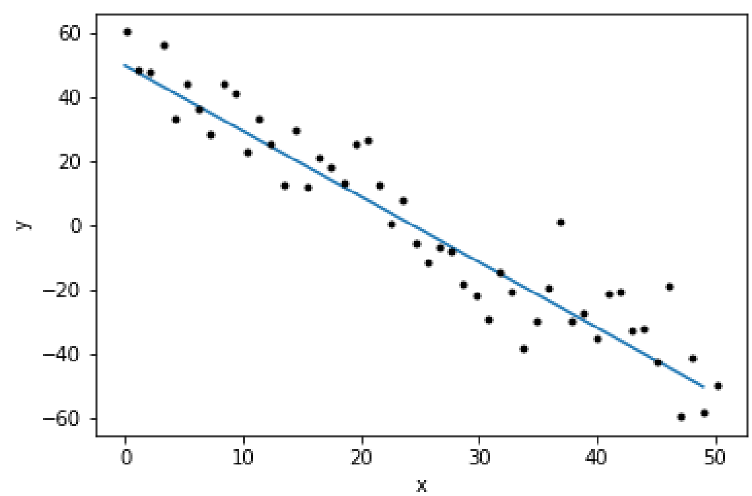

図 1.$y = -2x + 50$の線（青線）と、この式から生成されたノイズつきデータ点（黒点）

さて、まずは Jupyter を用いてこのファイルを解釈してみることにしよう。最終的に、x 軸のみを集めたデータと y 軸のみを集めたデータが欲しい。まずはデータを小数として解釈したものを print してみよう。

```
with open("data.csv") as f:
    for line in f:
        print([float(w) for w in line.strip().split(',')])
```

各行について、0 番目の要素が x 軸を表し、1 番目の要素が y 軸を表している。なので、これを分割代入することが出来る。

```
with open("data.csv") as f:
    for line in f:
        x, y = [float(w) for w in line.strip().split(',')]
        print(x, y)
```

では、これらを格納するための配列をそれぞれ用意しよう。この配列を後で numpy の配列に変換することを考慮に入れ、x_axis_lst、y_axis_lst と名付けよう

```
x_data_lst = []
y_data_lst = []
```

では、for 文を用いてここにデータを格納していこう。

```
with open("data.csv") as f:
    for line in f:
        x, y = [float(w) for w in line.strip().split(',')]
        x_data_lst.append(x)
        y_data_lst.append(y)
```

最後に `x_axis_lst` と `y_axis_lst` を numpy の配列に直してしまおう。

```
import numpy as np
x_data = np.array(x_data_lst)
y_data = np.array(y_data_lst)
```

さて、ここまでの動きを一つの関数にまとめてしまおう。

以下のようになる。

```
def read_data(filename):
    x_data_lst = []
    y_data_lst = []
    with open(filename) as f:
        for line in f:
            x, y = [float(w) for w in line.strip().split(',')]
            x_data_lst.append(x)
```

```
            y_data_lst.append(y)
    return np.array(x_data_lst), np.array(y_data_lst)
```

この read_data は以下のように利用可能である。

```
x_data, y_data = read_data("data.csv")
```

それぞれ出力して確認して欲しい。

ここまで来たら、関数にまとめた部分以外の記述は削除してカーネルを再起動しても構わない。

2 Plot

さて、前章で得られたデータをプロットしよう。前回学んだ pyplot を使う。

```
plt.plot(x_data, y_data, 'k.')
```

ここで、「k.」は「k」+「.」であり、「k」は「blac**k**」、「.」は「**.**型のデータ点」の意味である。「k」の他にもいろいろなアルファベットが対応しているし、「.」の他にもいろいろな記号を使うことが出来る。

今回は$y = -2x + 50$に誤差を加えて生成したデータだということが分かっているので、一緒にプロットしてみよう。

```
plt.plot(x_data, y_data, 'k.')
plt.plot(x_data, -2 * x_data + 50)
```

ついでに軸ラベルもつけよう。

```
plt.plot(x_data, y_data, 'k.')
plt.plot(x_data, -2 * x_data + 50)
plt.xlabel('x')
plt.ylabel('y')
```

3 最小二乗法によるフィッティング

今回は式の形が$y = -2x + 50$であると分かっているが、実際の実験では先にデータ点が得られて、その後$y = ax + b$のaやbを推定したいと思うことがほとんどである。このような場合はたいてい最小二乗法を用いる。最小二乗法は各データ点とフィッティング線の差の二乗和を最小にする方法である。

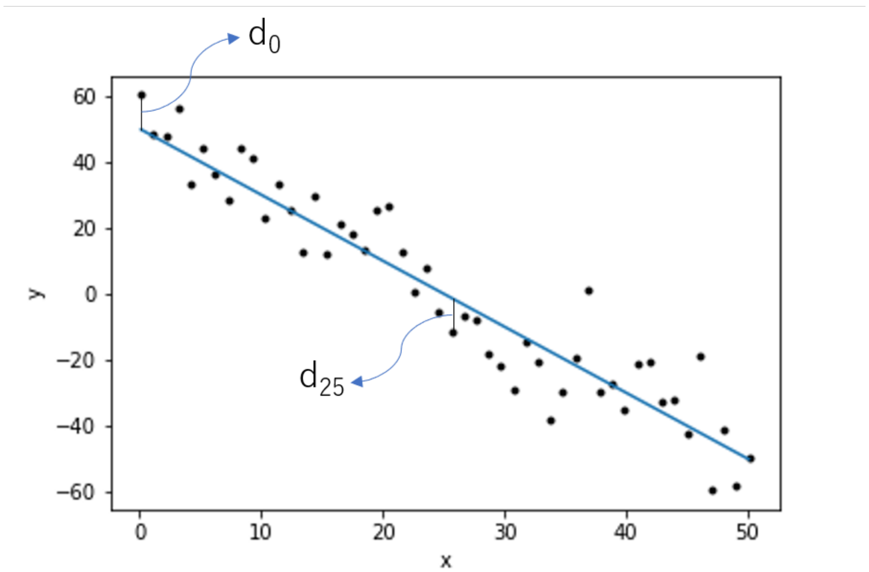

図 2.最小二乗法に用いる d_i

図 2 のように i 番目のデータ点と$y = ax + b$の差をd_iと定義し、その二乗和

$$L = \sum_i d_i^2$$

が最小になるようにaとbを決定する。では、最小二乗法の式を導こう。

i 番目のデータ点に関して、x 座標をx_i、y 座標をy_iとする。この時、

$$d_i = y_i - (ax_i + b)$$

とおく。このようなd_iの二乗和を L とおく。

$$L = \sum_i d_i^2 = \sum_i (y_i - ax_i - b)^2 = \sum_i (ax_i + b - y_i)^2$$

そして、L を最小化するように a と b を決定する。a と b が最小になっている時に満たす式は、

$$\begin{cases} \dfrac{\partial L}{\partial b} = 0 \\ \dfrac{\partial L}{\partial a} = 0 \end{cases}$$

である。これを変形すると、

$$\begin{cases} 0 = \dfrac{\partial L}{\partial b} = \displaystyle\sum_{i=0}^{n-1} 2(ax_i + b - y_i) \\ 0 = \dfrac{\partial L}{\partial a} = \displaystyle\sum_{i=0}^{n-1} 2(ax_i + b - y_i)x_i \end{cases}$$

となり、さらに展開して、

$$\begin{cases} nb + a\displaystyle\sum_{i=0}^{n-1} x_i = \sum_{i=0}^{n-1} y_i \\ b\displaystyle\sum_{i=0}^{n-1} x_i + a\sum_{i=0}^{n-1} x_i^2 = \sum_{i=0}^{n-1} y_i x_i \end{cases}$$

となる。これを行列を用いて表現すると、

$$\begin{pmatrix} n & \displaystyle\sum_{i=0}^{n-1} x_i \\ \displaystyle\sum_{i=0}^{n-1} x_i & \displaystyle\sum_{i=0}^{n-1} x_i^2 \end{pmatrix} \cdot \begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} \displaystyle\sum_{i=0}^{n-1} y_i \\ \displaystyle\sum_{i=0}^{n-1} y_i x_i \end{pmatrix}$$

となり、

$$\begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} n & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n-1} x_i & \sum_{i=0}^{n-1} x_i^2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \sum_{i=0}^{n-1} y_i \\ \sum_{i=0}^{n-1} y_i x_i \end{pmatrix} \tag{1}$$

となる。ここで、線形代数でならった逆行列の式$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad-bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$を使うと、

$$\begin{pmatrix} n & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n-1} x_i & \sum_{i=0}^{n-1} x_i^2 \end{pmatrix}^{-1} = \frac{1}{n\sum_{i=0}^{n-1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2} \begin{pmatrix} \sum_{i=0}^{n-1} x_i^2 & -\sum_{i=0}^{n-1} x_i \\ -\sum_{i=0}^{n-1} x_i & n \end{pmatrix}$$

となり、

$$\begin{pmatrix} b \\ a \end{pmatrix} = \frac{1}{n\sum_{i=0}^{n-1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2} \begin{pmatrix} \sum_{i=0}^{n-1} x_i^2 & -\sum_{i=0}^{n-1} x_i \\ -\sum_{i=0}^{n-1} x_i & n \end{pmatrix} \cdot \begin{pmatrix} \sum_{i=0}^{n-1} y_i \\ \sum_{i=0}^{n-1} x_i y_i \end{pmatrix}$$

$$= \begin{pmatrix} \dfrac{\sum_{i=0}^{n-1} x_i^2 \sum_{i=0}^{n-1} y_i - \sum_{i=0}^{n-1} x_i \sum_{i=0}^{n-1} x_i y_i}{n\sum_{i=0}^{n-1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2} \\ \dfrac{n\sum_{i=0}^{n-1} x_i y_i - \sum_{i=0}^{n-1} x_i \sum_{i=0}^{n-1} y_i}{n\sum_{i=0}^{n-1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2} \end{pmatrix}$$

が得られる。従って、

$$\begin{cases} b = \dfrac{\sum_{i=0}^{n-1} x_i^2 \sum_{i=0}^{n-1} y_i - \sum_{i=0}^{n-1} x_i y_i \sum_{i=0}^{n-1} x_i}{n\sum_{i=0}^{n-1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2} \\ a = \dfrac{n\sum_{i=0}^{n-1} x_i y_i - \sum_{i=0}^{n-1} x_i \sum_{i=0}^{n-1} y_i}{n\sum_{i=0}^{n-1} x_i^2 - \left(\sum_{i=0}^{n-1} x_i\right)^2} \end{cases}$$

となる。

　では、この関数を実装してみよう。この関数は np. sum を使うことで非常に簡単に実装することが出来る。b に関してはお手本を乗せるので、a に関しては自分で実装してみて欲しい。

```
def least_square(x, y):
    n = len(x)
    b = (np.sum(x ** 2) * np.sum(y) - np.sum(x * y) * np.sum(x)) / (n * np.sum(x ** 2) - np.sum(x) ** 2)
    a = 「自分で考えて書いてみよう」
    return b, a
```

実際に使ってみよう

```
least_square(x_data, y_data)
```

どうだろう？かなり 50 と-2 に近い値が得られたのではないだろうか？

では、この結果を使って、改めてフィッティング直線を描いてみよう。

```
b, a = least_square(x_data, y_data)
plt.plot(x_data, y_data, 'k.')
plt.plot(x_data, a * x_data + b)
plt.xlabel('x')
plt.ylabel('y')
```

せっかくなので、記念に保存しておこう。

```
b, a = least_square(x_data, y_data)
plt.plot(x_data, y_data, 'k.')
plt.plot(x_data, a * x_data + b)
plt.xlabel('x')
plt.ylabel('y')
plt.savefig("least_square.png")
```

出てきた図は提出して欲しい。

4 おまけ・最小二乗法の簡単な記述

最小二乗法の式はもうちょっと頑張って変形すると、二行で書ける。結論から申し上げると、このアルゴリズムは以下のようになる。

```
def least_square(x, y):
    X = np.array([np.ones_like(x), x])
    return np.linalg.inv(X @ X.T) @ X @ y
```

もしくは、下のように書き換えても良い。この二つは等価である。

```
def least_square(x, y):
    X = np.array([np.ones_like(x), x])
    return np.linalg.solve(X @ X.T, X) @ y
```

この式を導出しよう。先ほどの式(1)より、

$$\begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} n & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n-1} x_i & \sum_{i=0}^{n-1} x_i^2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \sum_{i=0}^{n-1} y_i \\ \sum_{i=0}^{n-1} x_i y_i \end{pmatrix}$$

であるが、実は

$$\mathrm{X}[0,i] = 1, \qquad \mathrm{X}[1,i] = x_i, \qquad \mathrm{y}[i] = y_i$$

となるような行列$\mathbf{X}$とベクトル$\boldsymbol{y}$を導入することでこの式は非常に簡単に記述出来る。この「1」のお気持ちは、$y = ax + b = b \cdot 1 + a \cdot x$という形に由来する。さて、$(\mathrm{AB})[i,j] = \sum_k A_{ik} B_{kj}$より、

$$\begin{pmatrix} n & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n-1} x_i & \sum_{i=0}^{n-1} x_i^2 \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n-1} 1 \cdot 1 & \sum_{i=0}^{n-1} 1 \cdot x_i \\ \sum_{i=0}^{n-1} 1 \cdot x_i & \sum_{i=0}^{n-1} x_i \cdot x_i \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n-1} X[0,i] \cdot X[0,i] & \sum_{i=0}^{n-1} X[0,i] \cdot X[1,i] \\ \sum_{i=0}^{n-1} X[1,i] \cdot X[0,i] & \sum_{i=0}^{n-1} X[1,i] \cdot X[1,i] \end{pmatrix}$$

$$= \begin{pmatrix} \sum_{i=0}^{n-1} X[0,i] \cdot X^T[i,0] & \sum_{i=0}^{n-1} X[0,i] \cdot X^T[i,1] \\ \sum_{i=0}^{n-1} X[1,i] \cdot X^T[i,0] & \sum_{i=0}^{n-1} X[1,i] \cdot X^T[i,1] \end{pmatrix} = \mathbf{X} \cdot \mathbf{X}^{\mathrm{T}}$$

である。また、$(Ax)_i = \sum_j A_{ij} x_j$より、

$$\begin{pmatrix} \sum_{i=0}^{n-1} y_i \\ \sum_{i=0}^{n-1} x_i y_i \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n-1} 1 \cdot y_i \\ \sum_{i=0}^{n-1} x_i \cdot y_i \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n-1} X[0,i] \cdot y[i] \\ \sum_{i=0}^{n-1} X[1,i] \cdot y[i] \end{pmatrix} = \mathbf{X} \cdot \mathbf{y}$$

となる。なので、

$$\begin{pmatrix} b \\ a \end{pmatrix} = \begin{pmatrix} n & \sum_{i=0}^{n-1} x_i \\ \sum_{i=0}^{n-1} x_i & \sum_{i=0}^{n-1} x_i^2 \end{pmatrix}^{-1} \cdot \begin{pmatrix} \sum_{i=0}^{n-1} y_i \\ \sum_{i=0}^{n-1} x_i y_i \end{pmatrix} = (\mathbf{X} \cdot \mathbf{X}^{\mathrm{T}})^{-1} \mathbf{X} \cdot \mathbf{y}$$

となる。なお、$(\mathbf{X} \cdot \mathbf{X}^{\mathrm{T}})^{-1} \mathbf{X}$にはムーアーペンローズの疑似逆行列という名前がついている。

これを実装すると、前述の実装となる。

```
def least_square(x, y):
    X = np.array([np.ones_like(x), x])
    return np.linalg.inv(X @ X.T) @ X @ y
```