2018年航空宇宙情報システム学第2
第2部「プログラミングと数値計算」

Numpyによる数値計算入門3
最小二乗法
(最終回)

2018年7月10日

最終課題について

最終課題

自由なテーマでPythonのプログラムを作成せよ。
また、そのテーマを選んだ動機、プログラムの仕組や実行結果について解説せよ。

- 必ず自分で考え自分で作成すること。コピー厳禁。
- ネット上の情報を参考にしたり、既存のプログラムを独自に発展させるのはOK。ただし、出典を明記し、オリジナルな部分を明らかにすること
- 動機やアイデアのユニークさで勝負するのも歓迎。
- 授業で扱わなかった関数やライブラリを使うのももちろんOK。
- レポート（後述）も必ず提出すること。

提出物

- 作成したPythonのプログラム
  - データファイルなどを利用する場合は一緒に提出
- 以下について記したレポート。テキストエディタで書いたプレインテキスト、Word で作成したファイル、あるいはPDF ファイルで提出
  1. 学科、学年、学生証番号、氏名、（連絡が取れる）メールアドレス
  2. 作成したプログラムのタイトル(例:「XX シミュレーションプログラム」など）
  3. プログラムの解説
  4. 評価してもらいたい点
  5. 参考にした情報（書籍、インターネットサイト、等）

提出と締切

- これまでの小課題と同様に、ITC-LMSから提出
- 複数ファイルの提出方法

  (1) 「成果物提出」のところで「ファイル追加」ボタンを押して複数のファイルを選択

  (2) あらかじめzipやlzhなどで1個のアーカイブファイルにまとめてから提出

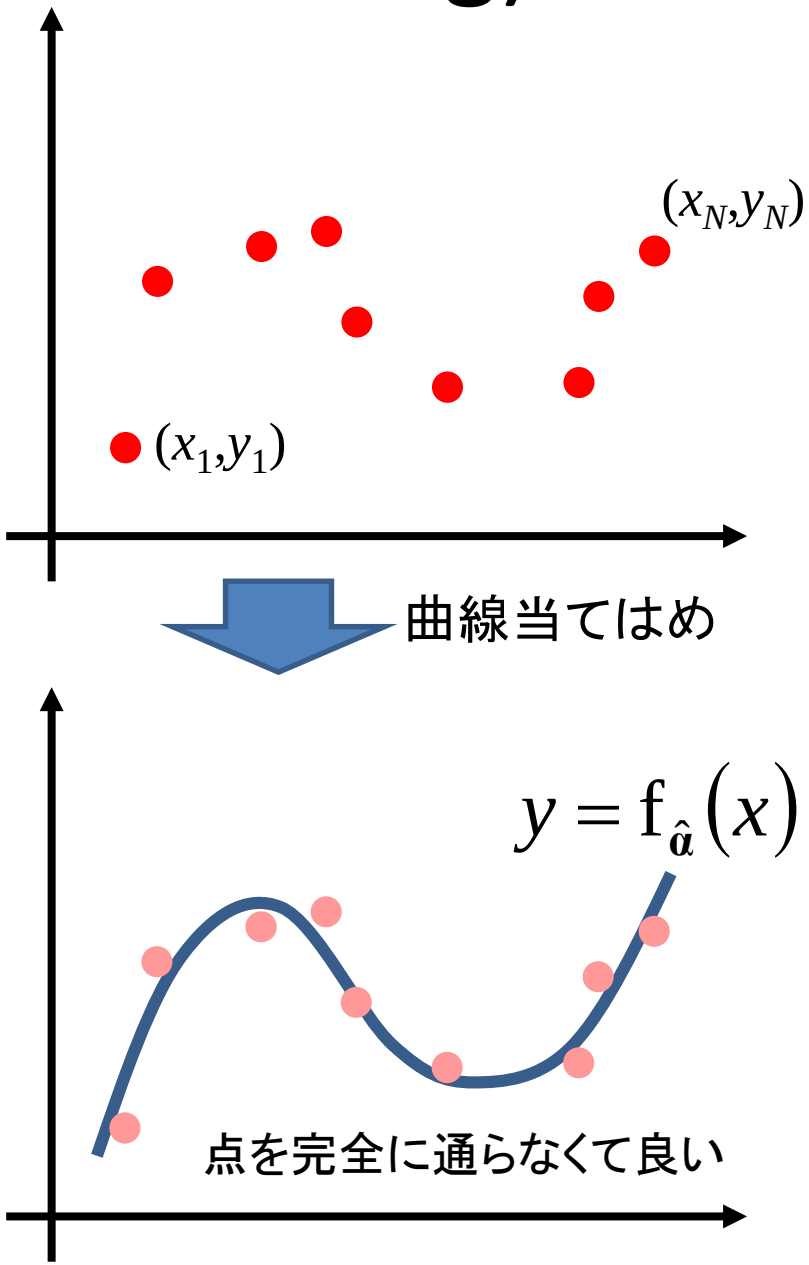
  ファイルが数個程度であれば(1)を推奨
- 締切：8月10日（金）午後11時59分 厳守
  - 成績報告期限まで余裕がないので注意
- 提出物に不備が無いように注意すること

最小二乗法

曲線当てはめ(Curve-fitting)

N 個のデータ点（サンプル）$(x_1, y_1), (x_2, y_2), \ldots, (x_N, yN)$ があるとき、このデータに良く当てはまるような、関数 $y = \mathrm{f}_\alpha(x)$ を求める

- 実際には、関数そのものではなくパラメータ $\boldsymbol{\alpha}$ の最適値（$\hat{\boldsymbol{\alpha}}$）を求める。
- xを入力変数, $\boldsymbol{\alpha}$ をパラメータとする関数 f を $\mathrm{f}_\alpha(x)$ や $\mathrm{f}(x;\alpha)$のように書く
- 補間（interpolation）とは違う点にも注意



最小二乗法

データに対する当てはまりの良し悪し（コスト関数、目的関数、などと呼ばれる）を、残差2乗和で表す

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} |y_i - f_{\boldsymbol{\alpha}}(\boldsymbol{x}_i)|^2 = \left(y_1 - f_{\boldsymbol{\alpha}}(\boldsymbol{x}_1)\right)^2 + \cdots + \left(y_N - f_{\boldsymbol{\alpha}}(\boldsymbol{x}_N)\right)^2$$

1つ目のサンプル　　N個目のサンプル

実際のｙの値　　関数によるｙの値

注：一般化して、x はベクトルとする

この残差2乗和が最小となるような α の値を求めるのが、最小二乗法（least squares method)

- 関数が、パラメータ α に対して線形な場合を線形最小二乗法、非線形な場合を非線形最小二乗法と呼ぶ

線形最小二乗法

関数 $\mathrm{f}_{\boldsymbol{\alpha}}(\boldsymbol{x})$ が、**パラメータ** $\boldsymbol{\alpha} = \{\alpha_0, \alpha_1, \cdots \alpha_K\}$ **に対して線形**な場合を、線形最小二乗法と呼ぶ. つまり、

$$\mathrm{f}_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \alpha_0 \cdot \mathrm{f}_0(\boldsymbol{x}) + \alpha_1 \cdot \mathrm{f}_1(\boldsymbol{x}) + \cdots + \alpha_K \cdot \mathrm{f}_K(\boldsymbol{x})$$

- ここで、$\mathrm{f}_0(\boldsymbol{x}), \mathrm{f}_1(\boldsymbol{x}), \cdots, \mathrm{f}_K(\boldsymbol{x})$は**既知**の関数
- 変数$\boldsymbol{x}$に対して非線形でも構わない

例1 : (1変数)多項式:

$$\mathrm{f}_{\boldsymbol{\alpha}}(x) = \alpha_0 \cdot 1 + \alpha_1 \cdot x + \alpha_2 \cdot x^2 + \cdots + \alpha_K \cdot x^K$$

例2 : フーリエ級数:

$$\mathrm{f}_{\boldsymbol{\alpha}}(x) = \frac{a_0}{2} + \sum_{j=1}^{\infty} (a_k \cos kx + b_k \sin kx)$$

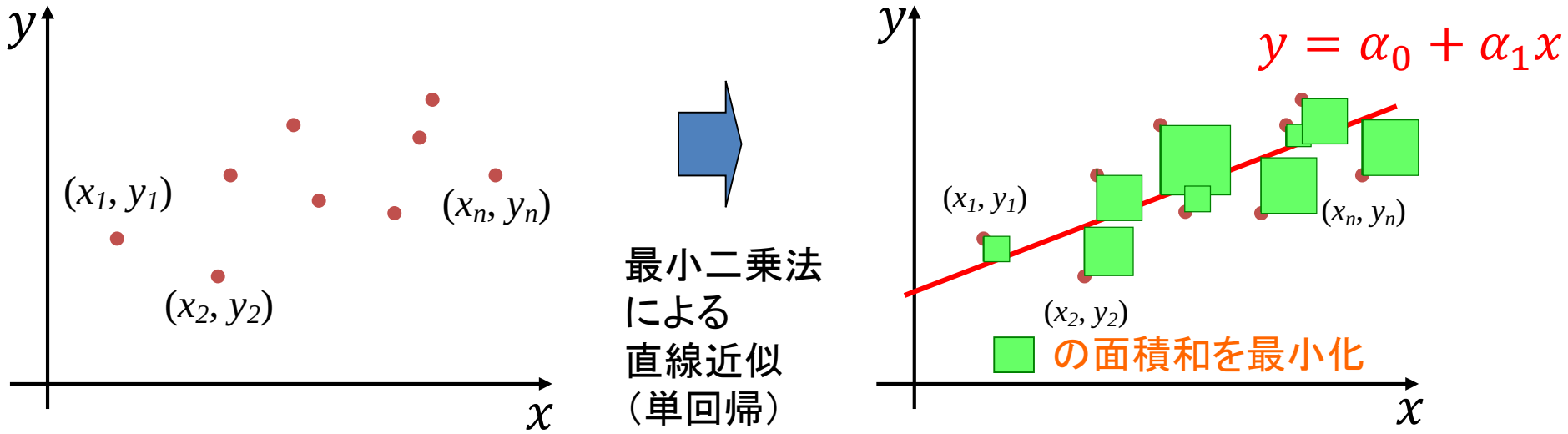
線形最小二乗法 1：単回帰

- 入力変数（説明変数）x は1次元（スカラー）
- 関数 $\mathrm{f}_{\boldsymbol{\alpha}}(x)$ は x の線形（1次）関数

$$y = \mathrm{f}_{\boldsymbol{\alpha}}(x) = \alpha_0 \cdot 1 + \alpha_1 \cdot x$$

- いわゆる「直線近似」

  $-\ \alpha_0$ は切片、α_1 は傾きを表す。



線形最小二乗法 1：単回帰（続き）

残差2乗和 $Q(\alpha_0, \alpha_1)$ は、

$$Q(\alpha_0, \alpha_1) = \sum_{i=1}^{n} (y_i - \alpha_0 - \alpha_1 x_i)^2$$

α_0, α_1について最小化→微分してゼロとおく

$$\frac{\partial Q}{\partial \alpha_0} = -2 \sum_{i=1}^{n} (y_i - \alpha_0 - \alpha_1 x_i) = 0$$

$$\frac{\partial Q}{\partial \alpha_1} = -2 \sum_{i=1}^{n} x_i (y_i - \alpha_0 - \alpha_1 x_i) = 0$$

整理して連立方程式の形にすると、

$$\begin{bmatrix} n & \sum_i x_i \\ \sum_i x_i & \sum_i {x_i}^2 \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{bmatrix}$$

この1次方程式を解けば、最適なα_0, α_1が求まる

線形最小二乗法 1：単回帰（続き）

- n個のデータサンプルのxの値、yの値を、それぞれ n 次元の列ベクトルで表すとする

$$\boldsymbol{x} = [x_1, x_2, \cdots, x_n]^T \, , \boldsymbol{y} = [y_1, y_2, \cdots, y_n]^T$$

- $\boldsymbol{x}$については、全て値が1である列を付け足す

$$\boldsymbol{X} = [\boldsymbol{1}_n, \boldsymbol{x}] = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

- パラメータも、$\boldsymbol{\alpha} = [\alpha_0, \alpha_1]^T$ のように表す
- すると、先ほどの連立方程式は、

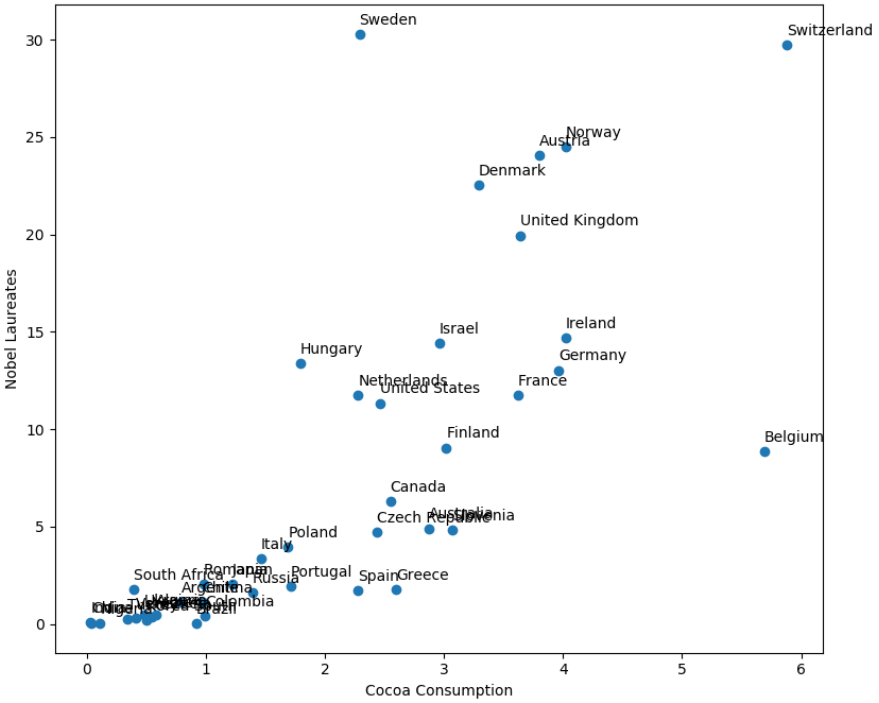
$$\boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\alpha} = \boldsymbol{X}^T \boldsymbol{y}$$

のように簡潔に書くことができる（正規方程式と呼ぶ）

[例題]ココア消費量とノーベル賞受賞者数

H Messerli, Franz. (2012). Chocolate Consumption, Cognitive Function, and Nobel Laureates. The New England journal of medicine. 367. 1562-4. 10.1056/NEJMon1211064.

国名	一人当たりの年間ココア消費量	1000万人あたりのNobel賞受賞者数
Switzerland	5.88	29.728
Belgium	5.69	8.85
Norway	4.02	24.503
Ireland	4.02	14.701
Germany	3.96	13.031
:	:	:
Japan	1.22	2.039
:	:	:
India	0.03	0.071



相関係数 0.75

データ出典:

http://onegoldenticket.blogspot.com/2013/01/chocolate-consumption-statistics-by.html

https://en.wikipedia.org/wiki/Nobel_laureates_per_capita

単回帰の例

（詳しくは、lsq_cocoa.py を参照）

x に各国の1人あたりココア消費量(kg/人/年) x = [5.88, 5.69, …, 0.03]

yに各国の1000万人あたりのノーベル賞受賞者数 y= [29.728, 8.85,…, 0.071]

#が読み込まれているとする

xベクトルの左に1ベクトルを加えて作った行列

X = np.column_stack((np.ones(numpts),x))

$$\boldsymbol{X} = [\mathbf{1}_n, \boldsymbol{x}] = \begin{bmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}$$

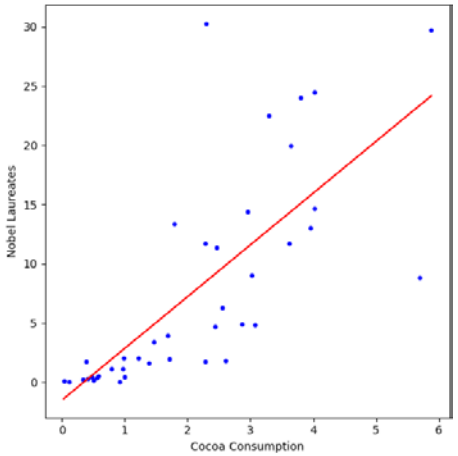
正規方程式を解く

alp = la.solve(np.dot(X.T,X),np.dot(X.T,y))

$$\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\alpha} = \boldsymbol{X}^T\boldsymbol{y}$$

>> print(alp)

[-1.51800489 4.36907945]



線形最小二乗法2：重回帰

- 入力変数（説明変数）が M 個ある場合に拡張
  - i 番目のデータサンプルを、$\boldsymbol{x}_i = [1, x_{i,1}, x_{i,2}, \cdots, x_{i,M}]$ の行ベクトルで表すとする
- 関数 $\mathrm{f}_{\boldsymbol{\alpha}}(x)$ は$x_1, x_2, \cdots, x_M$の線形関数

$$y = \mathrm{f}_{\boldsymbol{\alpha}}(\boldsymbol{x}) = \alpha_0 \cdot 1 + \alpha_1 \cdot x_1 + \cdots + \alpha_M \cdot x_M$$

- 入力データの行列を以下のように構成

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \vdots \\ \boldsymbol{x}_n \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,M} \\ 1 & x_{2,1} & \cdots & x_{2,M} \\ \vdots & \vdots & & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,M} \end{bmatrix}$$

注: $x_{i,j}$は、i番目のサンプルのj番目の入力変数の値

線形最小二乗法2：重回帰（続き）

出力データ: $\boldsymbol{y} = [y_1, y_2, \cdots, y_n]^T$（単回帰と同じ）

パラメータ :$\boldsymbol{\alpha} = [\alpha_0, \alpha_1, \cdots, \alpha_M]^T$

とすると、残差2乗和 $Q(\boldsymbol{\alpha})$ は、

$$
\begin{aligned}
Q(\boldsymbol{\alpha}) &= \sum_{i=1}^{n} |y_i - \boldsymbol{x}_i \boldsymbol{\alpha}|^2 \\
&= \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\alpha}\|^2 = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\alpha})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\alpha}) \\
&= \boldsymbol{y}^T \boldsymbol{y} - 2\boldsymbol{\alpha}^T \boldsymbol{X}^T \boldsymbol{y} + \boldsymbol{\alpha}^T \boldsymbol{X}^T \boldsymbol{X} \boldsymbol{\alpha}
\end{aligned}
$$

と書ける

線形最小二乗法2：重回帰（続き）

$Q(\boldsymbol{\alpha})$が$\boldsymbol{\alpha}$について最小値（極値）を取る条件

$$\frac{\partial Q}{\partial \boldsymbol{\alpha}} = -2\boldsymbol{X}^T\boldsymbol{y} + 2\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\alpha} = \boldsymbol{0}$$

公式

$$\frac{\partial \boldsymbol{\alpha}^T\boldsymbol{\beta}}{\partial \boldsymbol{\alpha}} = \boldsymbol{\beta}$$

$$\frac{\partial \boldsymbol{\alpha}^T\boldsymbol{B}\boldsymbol{\alpha}}{\partial \boldsymbol{\alpha}} = (\boldsymbol{B} + \boldsymbol{B}^T)\boldsymbol{\alpha}$$

- $\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{\alpha} = \boldsymbol{X}^T\boldsymbol{y}$　正規方程式
- $\hat{\boldsymbol{\alpha}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$　線形最小二乗法の解

データから、行列$\boldsymbol{X}$, ベクトル $\boldsymbol{y}$ を作成すれば、NumPyで簡単に求められる

- 注：逆行列 $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ を明示的に求めず、正規方程式を直接解いて$\boldsymbol{\alpha}$ を求める方が良い

（例題）収穫量の予測

東京の7,8,9月の平均気温、降雨量、日照時間から、その年の全国の水稲作況指数を予測する線形回帰モデルを作ってみよう。

入力（説明）変数

	y	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
	年	水稲作況指数	気温7月	降水量7月	日照時間7月	気温8月	降水量8月	日照時間8月	気温9月	降水量9月	日照時間9月
i=1	1981	96	26.3	167	181.3	26.2	133	190.7	21.8	138.5	140
i=2	1982	96	23.1	140	102.3	27.1	152.5	170.1	22.3	371.5	104.3
	1983	96	23.8	141.5	119.4	27.5	168.5	163.7	23.1	242.5	92.8
	1984	108	26.2	81.5	180.1	28.6	31.5	252.3	23.5	61	146.9

データは、1981年から2015年までの35年間分。

ファイル名：ricecrop-weather-1981-2015.csv

（例題）収穫量の予測（続き）

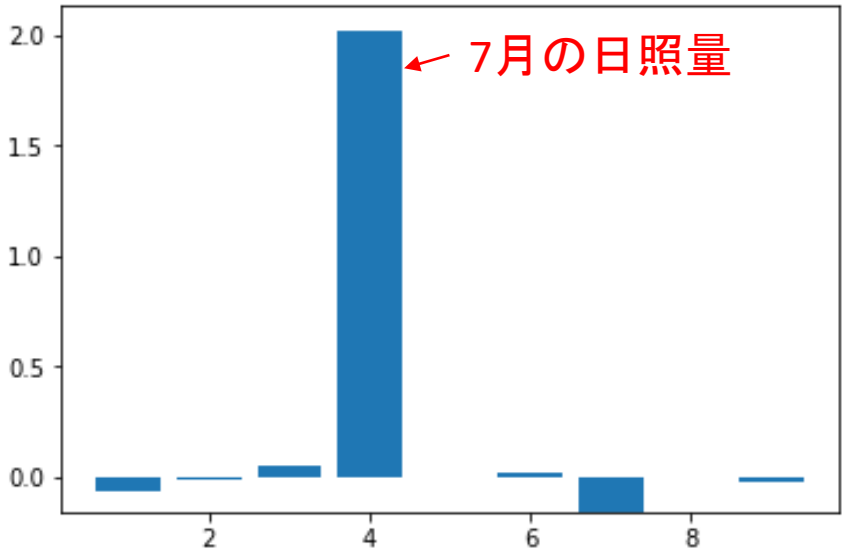
```
from numpy import *
import numpy.linalg as la

# データの読み込み（numpy の loadtxt 関数利用)
data = loadtxt('ricecrop-weather-1981-2015.csv',delimiter=",",skiprows=1)
years = data.shape[0]
# データから、ベクトルy , 行列 X を作成
y = data[:,1]
X = hstack((np.ones((35,1)),data[:,2:]))
# 正規方程式を解いて、係数ベクトルを求める
alp = la.solve(dot(X.T,X),dot(X.T,y))
```

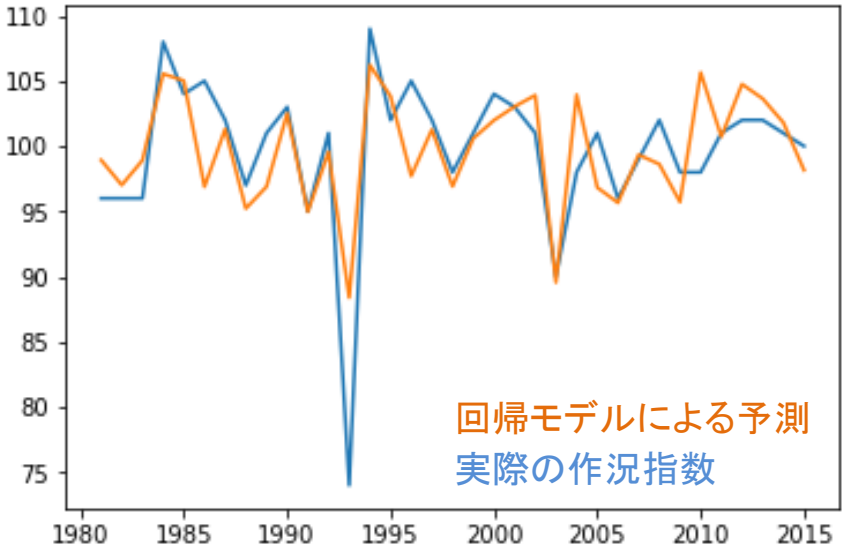
lsq_rice_crop.py 参照

（例題）収穫量の予測（結果）

各係数の値（定数項除く）



作況指数と回帰モデルによる予測値



注意事項：

- 気温、降雨量、日照量は次元（単位）が異なるので、係数の絶対値を比較することは本来意味が無い（無次元化が必要）
- 右のグラフはα を求めるのに使ったXに対して再度適用しているので、正しくは「予測」ではなく「再現」あるいは「復元」である。（2016年、2017、2018年に適用すれば、「予測」）

非線形最小二乗法（Gauss-Newton法）

$\mathrm{f}_{\boldsymbol{\alpha}}(x)$（ここでは$\mathrm{f}(x, \boldsymbol{\alpha})$と書く）がパラメータ$\boldsymbol{\alpha}$に対して非線形な場合は、一般には解析的な解が存在しないので、反復的な計算によって$\boldsymbol{\alpha}$の値を求めることを考える。k 回目まで反復計算したときの$\boldsymbol{\alpha}$の値を$\boldsymbol{\alpha}_k$,次の反復で求めるべき$\boldsymbol{\alpha}$の値を$\boldsymbol{\alpha}_{k+1}$とすると、

$$\mathrm{f}(\boldsymbol{x}_i, \boldsymbol{\alpha}_{k+1}) \approx \mathrm{f}(\boldsymbol{x}_i, \boldsymbol{\alpha}_k) + \frac{\partial \mathrm{f}}{\partial \boldsymbol{\alpha}}(\boldsymbol{\alpha}_{k+1} - \boldsymbol{\alpha}_k)$$

$$= \mathrm{f}(\boldsymbol{x}_i, \boldsymbol{\alpha}_k) + \boldsymbol{J}_i \Delta\boldsymbol{\alpha}_{k+1} \text{ なので、}$$

$$Q(\boldsymbol{\alpha}_{k+1}) \approx \sum_{i=1}^{n} |y_i - \mathrm{f}(\boldsymbol{x}_i, \boldsymbol{\alpha}_k) - \boldsymbol{J}_i \Delta\boldsymbol{\alpha}_{k+1}|^2 \text{ を最小化するのは、}$$

Gauss-Newton法（続き）

先ほどの線形最小二乗法の結果を用いると、

$$\Delta\boldsymbol{\alpha}_{k+1} = (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T\big(\boldsymbol{y} - \mathrm{f}(\boldsymbol{\alpha}_k)\big)$$

J はJacob行列を表し、その(i,j)成分は、$J_{i,j} = \dfrac{\partial \mathrm{f}(\boldsymbol{x}_i,\boldsymbol{\alpha}_k)}{\partial \alpha_j}$

また、$\mathrm{f}(\boldsymbol{\alpha}_k) = \begin{bmatrix} \mathrm{f}(\boldsymbol{x}_1,\boldsymbol{\alpha}_k) \\ \vdots \\ \mathrm{f}(\boldsymbol{x}_n,\boldsymbol{\alpha}_k) \end{bmatrix}$とする

よって、$k+1$回目の反復では、

$$\boldsymbol{\alpha}_{k+1} = \boldsymbol{\alpha}_k + (\boldsymbol{J}^T\boldsymbol{J})^{-1}\boldsymbol{J}^T\big(\boldsymbol{y} - \mathrm{f}(\boldsymbol{\alpha}_k)\big)$$

とすれば良い

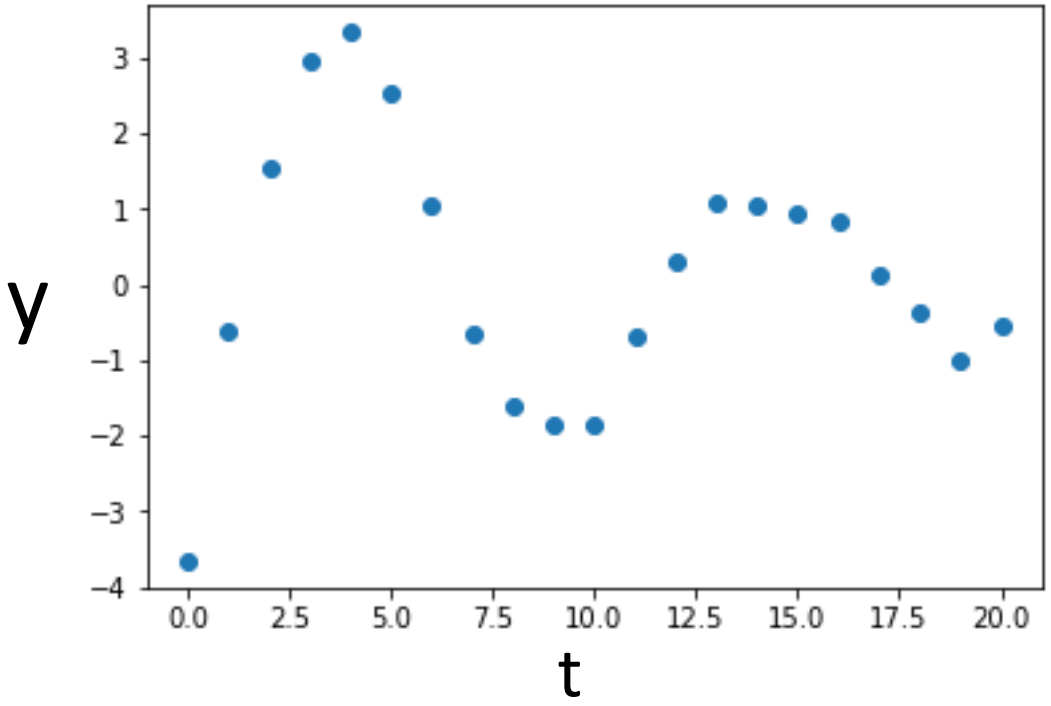
非線形最小二乗法の実際

- Gauss-Newton 法は、初期解が真の解に十分近くないとうまく動かないことが多い
- 代わりに、Levenberg-Marquardt 法がよく用いられる
- scipyでは、optimizationモジュールの中に、least_squares 関数が用意されている
- Jacob行列を求めるために、あらかじめ関数f(x,α)のαに関する偏微分（勾配）を手計算しておくのは面倒なので、近似的に求めることが多い

$$J_{i,j} = \frac{\partial \mathrm{f}\left(\mathbf{x}_i, \boldsymbol{\alpha}\right)}{\partial \alpha_j} \approx \frac{\mathrm{f}\left(\mathbf{x}_i, \boldsymbol{\alpha} + \left[0, \cdots, \delta, \cdots, 0\right]^T\right) - \mathrm{f}\left(\mathbf{x}_i, \boldsymbol{\alpha}\right)}{\delta}$$

例題：減衰振動のパラメータ推定

減衰振動の一般解 $y = C \cdot e^{-\gamma t} \cdot \sin(\omega t - \theta)$
のパラメータ $\boldsymbol{\alpha} = [C, \gamma, \omega, \theta]^T$を適当に設定して
データを生成した後、非線形最小二乗法でパラメータを推定してみよう。（damp_osci_sim.py を参照）

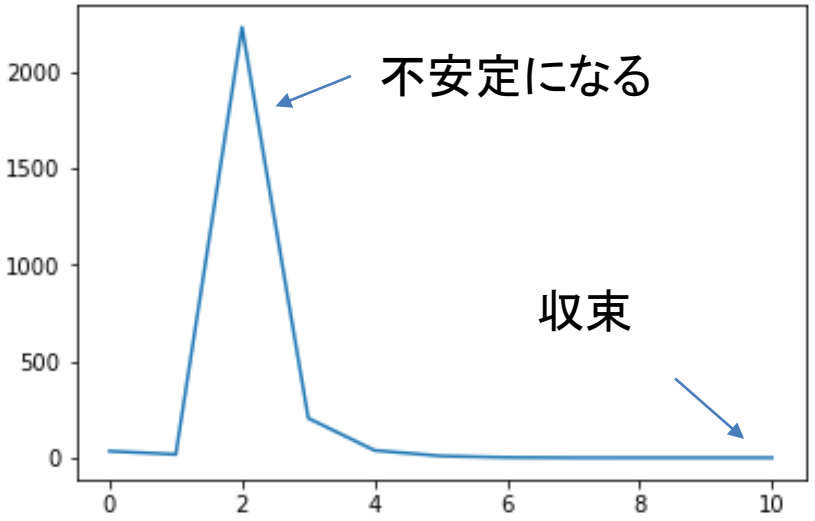


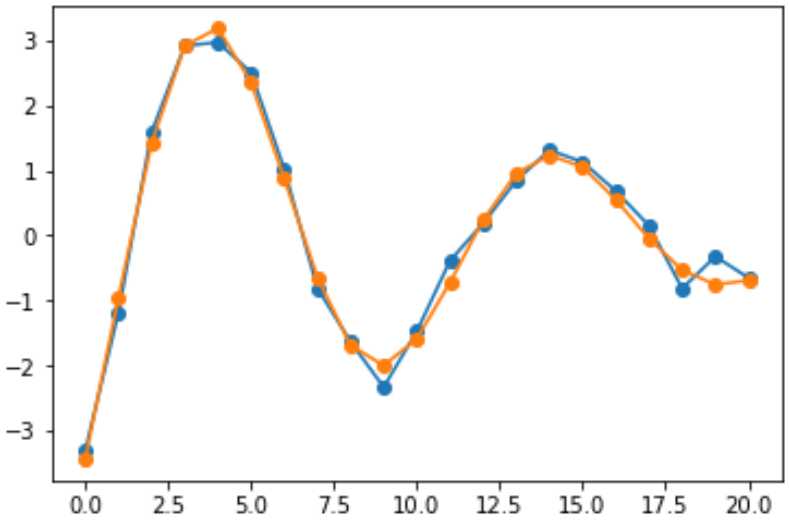
観測ノイズが
加わって
いる点に注意

例題: 結果の例

- 真のパラメータ:
  - a_true= [5.0, 0.1, 0.6, pi/4]
- 与えた初期解
  - a_ini = [1.0, 0.0, 0.5, 0.5]
- Gauss-Newton 法による解の例（毎回異なる）:
  - a_est = [4.63014962, 0.09296732, 0.60646356, 0.8351123]

残差2乗和の変遷





データと推定された
パラメータによる再現