材料計算科学・データ解析に関するチュートリアルコース
2023/1/31 10:00~11:00

最小二乗法、回帰、最適化の基礎

機械学習とは

Wikipedia:
https://ja.wikipedia.org/wiki/%E6%A9%9F%E6%A2%B0%E5%AD%A6%E7%BF%92

- 経験からの学習により自動で改善するコンピューターアルゴリズムもしくはその研究領域
- 人工知能の一種であるとみなされている

機械学習の種類

- 教師あり学習: 回帰、分類など
  正解がわかっている「学習データ」によりモデルを学習し、
  そのモデルを使って予測する

- 教師なし学習: クラスタリング、次元削減 (主成分分析) など
  正解がわかっている「学習データ」を使わない。
  背景には「データ間の距離（測度）」が定義されている

- 強化学習: ベイズ推定など
  学習データを追加するごとにモデルを更新する。
  学習データの追加に人間が関与することもあれば、
  コンピュータシステムだけで完結させることもできる
  囲碁対戦など ⇔ 昔のオセロやチェスのCPU対戦は
  全パターン検索なので、機械学習やAIと呼ばれない

機械学習の種類

- 次元削減: 多数の記述子から、モデルを記述するのに
  必要な少数の記述子を抽出する
  - LASSO回帰
  - 主成分分析
- 分類・認識:目的関数 (データの種類) が既知のデータ (記述子＋種類) を
  学習させ、種類が未知のデータの記述子から種類を推測する
  例: 犬、鳥、魚の写真を学習させ、別の写真がどれかを推測する
- クラスター解析: 種類が既知のデータを与えることなく、データ間の距離から
  グループ(クラスター)分けを行う
- 回帰: データ (記述子＋目的関数) の集合を学習 (フィッティング) させ、
  データの集合を記述する数値解析モデルを作る。
  モデルに記述子を入力することで目的関数を予測する。
  売上予測: 目的関数をコンビニの売上高にする
  制御: 以前の機械の動作パラメータと目的関数を学習データとして
  モデルを構築。
  所望の目的関数が得られる記述子(動作パラメータ)を
  装置に反映させる

回帰: 従来のデータ解析、数値解析

回帰: 要するに 「フィッティング」

従来のデータ解析、数値解析 : Arrheniusプロットなど

・フィッティングさせる数値モデルは関数モデル、物理モデルや、
比較的少ない変数 (記述子) の数学的モデルによって与えられている
パラメトリックモデル、パラメトリック回帰

良い点: 比較的少ない実験データ (目的関数) と変数 (記述子) でよい
実験量が少なくて済む、解析時間が短い

悪い点: モデルが悪いとフィッティングが合わない

良い点: フィッティングが合うことにより、モデルの正当性、
モデルを導出した理論の正当性を確認できる。
得られた変数に物理・化学・科学的意味・一般性 => 回帰の目的

悪い点: モデルを知らないと解析できない

悪い点: モデル毎にプログラムを作らないといけない

回帰 (regression): 機械学習

機械学習: 回帰により予測することを目的とすることが多い
=> 予測性能・信頼性が保証される必要がある

良い点: モデルを知らなくても解析できる
・モデルは柔軟 (どのような関数でも表現できる) である必要がある
非常に多数の自由度、多数の変数を含む
ノンパラメトリックモデルを含む

悪い点: 非常に多数の自由度、多数の変数を決めるため、
非常に多数の「学習データ」が必要
計算時間がかかる

悪い点: 回帰結果が学習データを良く再現していたとしても、
学習データ以外のデータを正しく予測できるとは限らない
・過学習 (over learning。過剰適合 over fitting) を起こしやすい
過学習を抑えるため、さらに学習データが必要

良い点: 予測性能が確認されれば、中身がわからなくても、
どのようないいかげんな手順で得たモデルでも利用できる　ブラックボックス
↑すいません。言い過ぎです

良い点: プログラムにモデルの汎用性がある。
悪い点: しかし、使えるモデルを見つけるには試行錯誤が必要 (知識があればなおよい)

今日のチュートリアル

・回帰は、データ解析、予測、制御の中枢である

・従来の回帰と、機械学習の目的は異なる

今日のチュートリアルの目的

・研究データの整理・解析に必要な手段 (アルゴリズム) を
選べるようになる。必ずしも機械学習が最適ではない

2. 従来の回帰 (線形回帰、非線形回帰、**Kernel回帰**) を学ぶ

1. 非線形回帰を学ぶために最適化法を学ぶ

3. 非線形最適化を学ぶために方程式の解法を学ぶ

第1回の強化学習: ベイズ推定＋ガウシアン**Kernel回帰**
=> 第3回の講義へ

最適化: 最大化問題、最小化問題

変数: 記述子の組　$\{x_i\}$
目的関数: $f(x_i)$

目的: $f(x_i)$ を最大化、あるいは最小化する

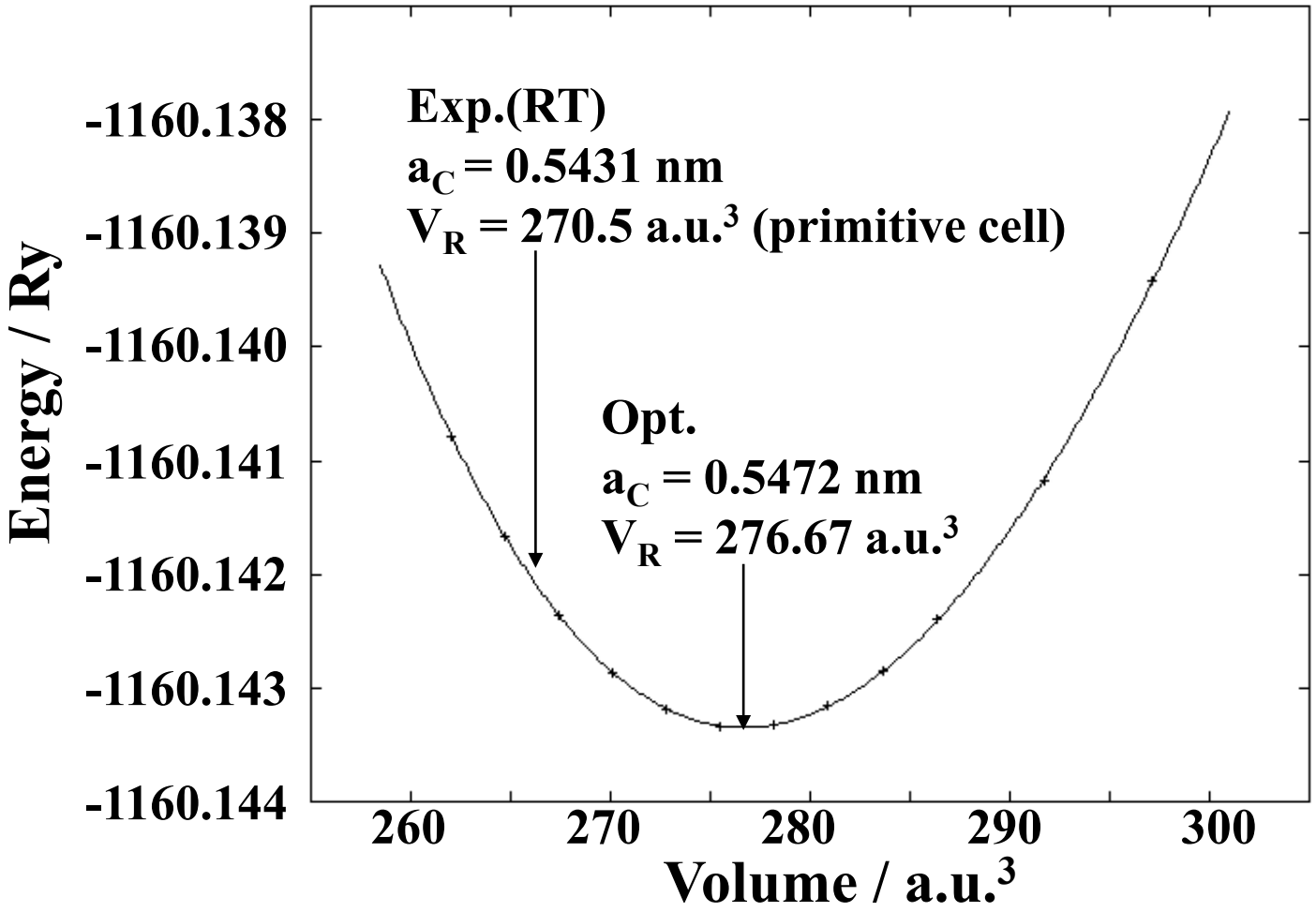
最大化問題は、$-f(x_i)$ を目的関数にとることにより、
最小化問題になる

今日は、基本的に最小化問題を想定する

最小化問題: 安定構造 (構造緩和計算)

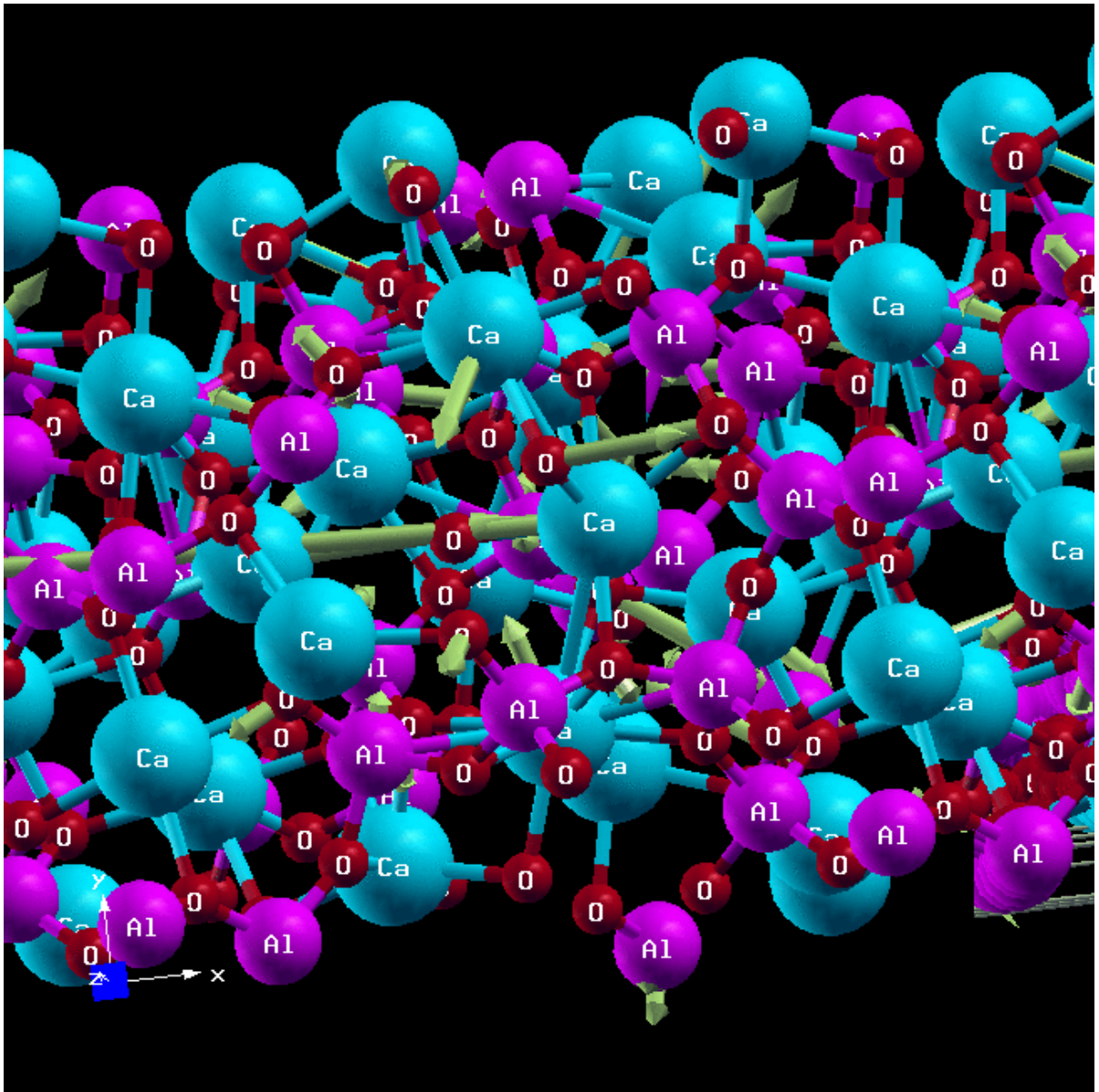
分子構造や結晶構造 (記述子) を与えて 全エネルギー (目的関数) を計算する: 量子計算
構造を変化させ、全エネルギーが最小になる構造が最安定構造である

結晶Siの単位格子体積を変えながら全エネルギーを第一原理量子計算で計算した例:



複雑な結晶の構造緩和：$12CaO \cdot 7Al_2O_3$

格子定数だけでなく、原子の座標も変えながら全エネルギーを最小化する



回帰は最小化問題: 最小二乗法

問題: あるデータの組 (x_1, y_1), ・・・ (x_n, y_n) が理論式 $f(x) = a + bx$ に従うことがわかっている。未知変数 a と b を求めよ。
ただし、データ (y_i) には誤差 ε_I が含まれている: $y_i = f(x_i) + \varepsilon_i$

直観的に、以下の最小化問題を解けばいいと見当がつく

- $\max_{x_i}|f(x_i) - y_i|$を最小化: ミニマックス近似
- L1ノルム　$S = \Sigma|f(x_i) - y_i|$ を最小化
  プログラムが比較的面倒

- L2ノルム (ユークリッド距離) の二乗　$S = \Sigma(f(x_i) - y_i)^2$ を最小化: 最小二乗法
  (線形最適化では) プログラムが非常に簡単

$$L_p\text{ノルム}: [\Sigma|f(x_i) - y_i|^p]^{1/p}$$

***f*(*x*): モデル**

ミニマックス近似:多項式

$\max_{x_i}|f(x_i)-y_i|$ を最小にする

=> $\max_{x_i}(f(x_i)-y_i)=\max_{x_i}(y_i-f(x_i))$ となるように変数を調整する

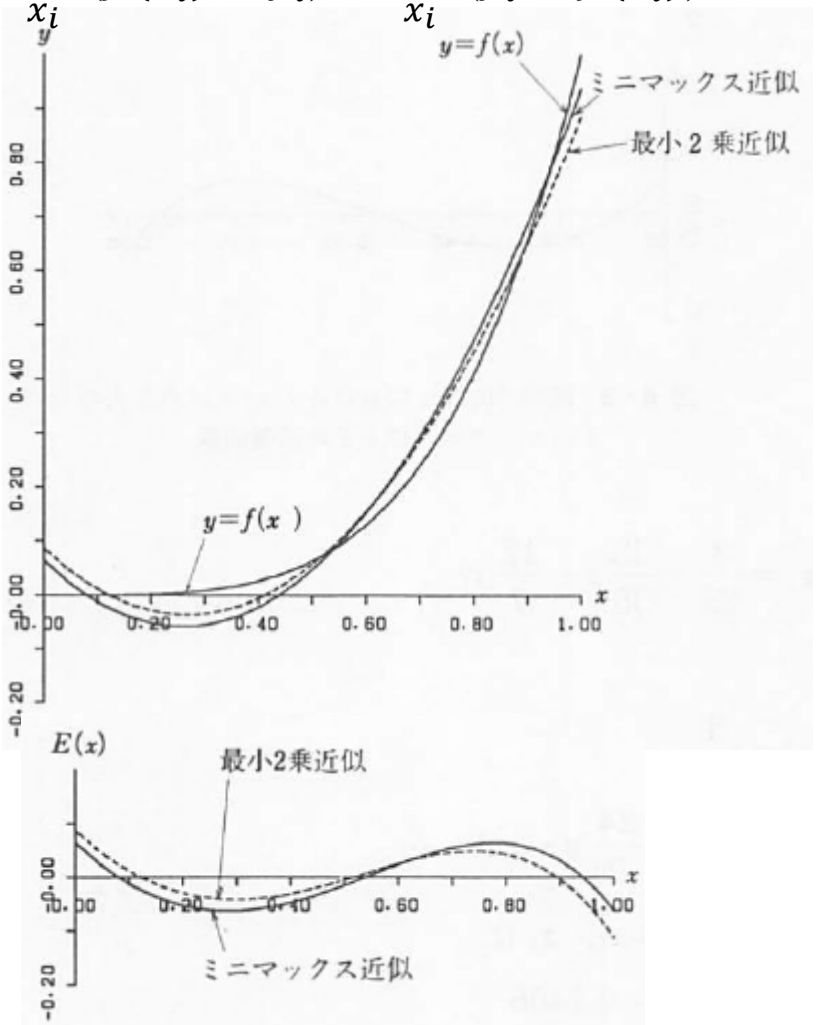


図 6・3　ミニマックス近似と最小 2 乗近似

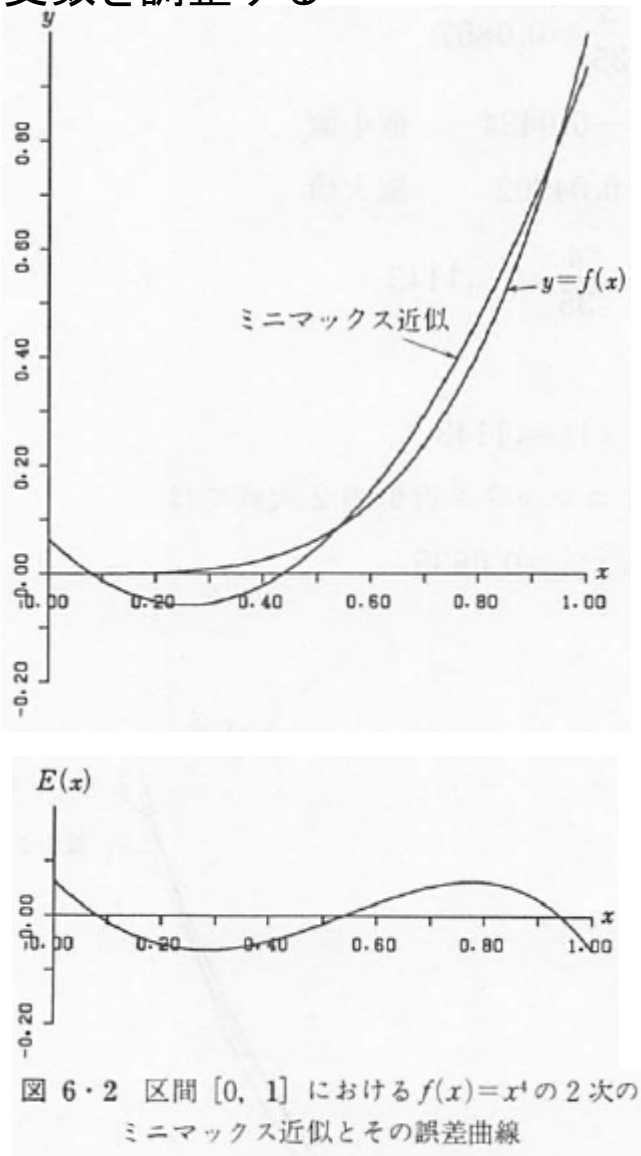


図 6・2　区間 [0, 1] における $f(x)=x^4$ の 2 次のミニマックス近似とその誤差曲線

線形最小二乗法 (線形回帰)

線形問題: $f(x_i) = a + bx_i$ **のように、**$f(x_i)$ **が未知変数** a, b **の線形関数になっている**

x_i の線形関数である必要はない

$f(x_i) = a + bx_i$

目的関数 $S = \Sigma(a + bx_i - y_i)^2$ **を最小化**

$$dS/da = 2\Sigma(a + bx_i - y_i) = 2an + 2b\Sigma x_i - 2\Sigma y_i = 0$$
$$dS/db = 2\Sigma x_i(a + bx_i - y_i) = 2a\Sigma x_i + 2b\Sigma x_i^2 - 2\Sigma x_i y_i = 0$$

※ $2n \cdot a + 2\Sigma x_i \cdot b = 2\Sigma y_i$

$2\Sigma x_i \cdot a + 2\Sigma x_i^2 \cdot b = 2\Sigma x_i y_i$

の連立方程式を解けばよい

行列で表したほうが見通しがいい

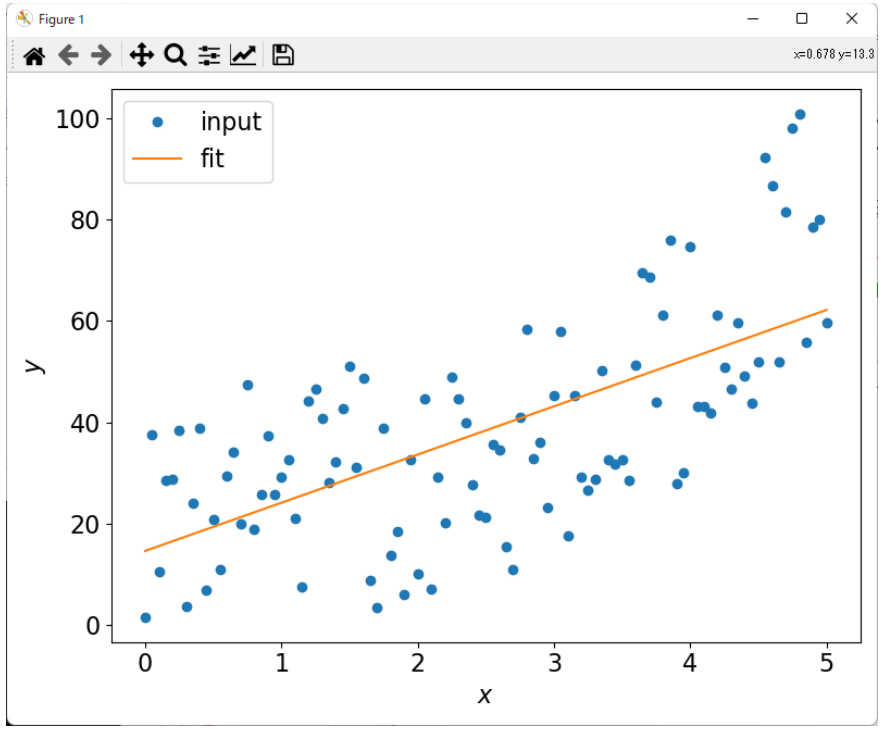
$$\begin{pmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix}$$

2x2の行列方程式を解けば、(a, b) **が求まる**

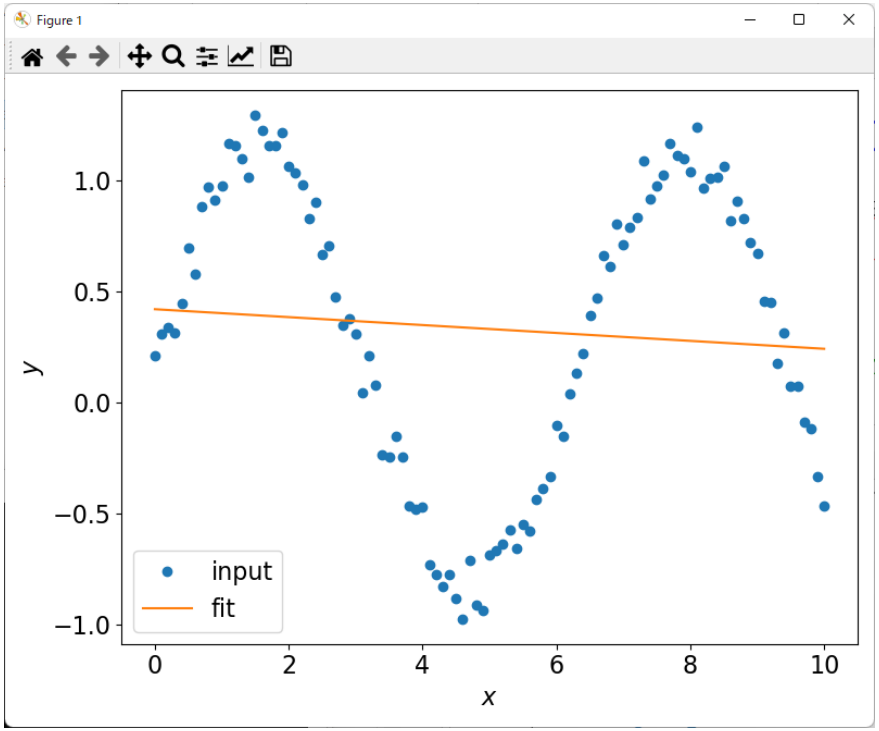
Program: lsq-line.py

場所: [tkProg]¥tkprog_tutorial¥regression
Usage: python lsq-line.py input_path

python lsq-line.py random-poly.xlsx



python lsq-line.py random-sin.xlsx



線形最小二乗法: 多項式

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_p x^p = \sum_{k=0}^{p} a_k x^k$$

これも、$f(x)$は a_i に関する線形関数

$f(x)$は x に関する非線形関数 (p次多項式)

目的関数 $S = \sum_{j=1}^{n} (\sum_{k=0}^{p} a_k x_j^{\,k} - y_j)^2$ を最小化

$$\frac{dS}{da_{k'}} = 2 \sum_{j=1}^{n} x_j^{\,k'} (\sum_{k=0}^{p} a_k x_j^{\,k} - y_j) = 0 \qquad k'=0, 1, \cdots, p$$

$$\sum_{j=1}^{n} (\sum_{k=0}^{p} a_k x_j^{\,k+k'} - y_j x_j^{\,k'}) = 0$$

※ $\sum_{k=0}^{p} a_k \sum_{j=1}^{n} x_j^{\,k+k'} = \sum_{j=1}^{n} y_j x_j^{\,k'}$

線形最小二乗法: 多項式

$$\sum_{k=0}^{p} a_k \sum_{j=0}^{n} x_j^{k+k'} = \sum_{j=0}^{n} y_j x_j^{k'}$$

$$\begin{pmatrix} n & \sum x_i & \sum x_i^2 & \cdots & \sum x_i^p \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & & \sum x_i^{p+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & & \sum x_i^{p+2} \\ \vdots & & & \ddots & \\ \sum x_i^p & \sum x_i^{p+1} & \sum x_i^{p+2} & & \sum x_i^{2p} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i x_i^2 \\ \vdots \\ \sum y_i x_i^p \end{pmatrix}$$

$$X_{k,k'} = \sum_{j=1}^{n} x_j^{k+k'} \qquad A_k = a_k \quad Y_k = \sum_{j=1}^{n} y_j x_j^{k'}$$

$$XA=Y$$

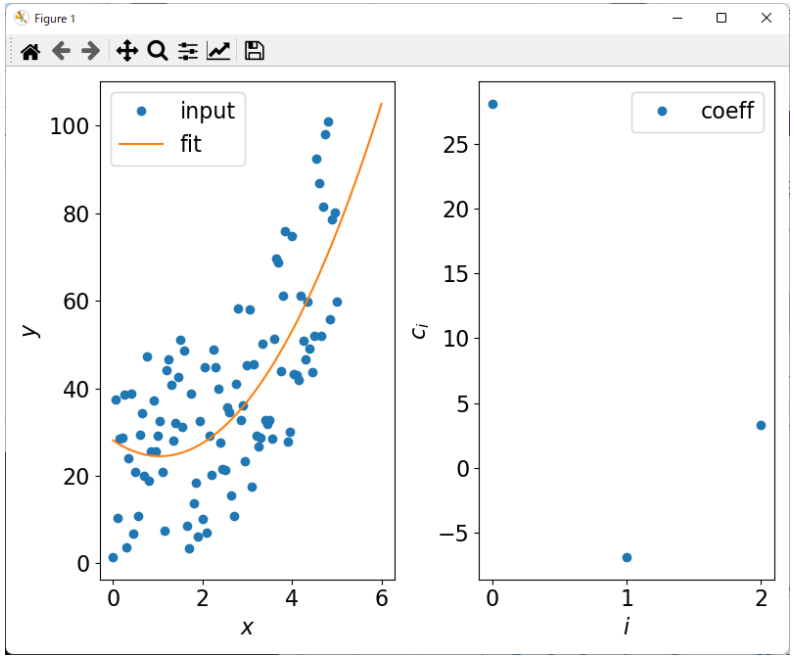
$$A=X^{-1}Y$$

Program: lsq-polynomial.py

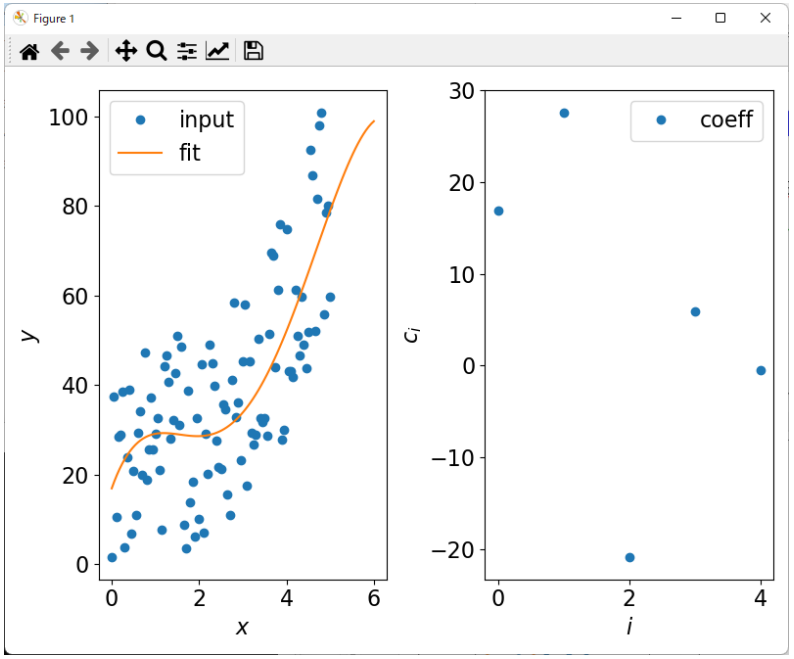
場所: [tkProg]¥tkprog_tutorial¥regression

Usage: python lsq-polynomial.py input_path norder

python lsq-polynomial.py random-poly.xlsx 2



python lsq-polynomial.py random-poly.xlsx 4

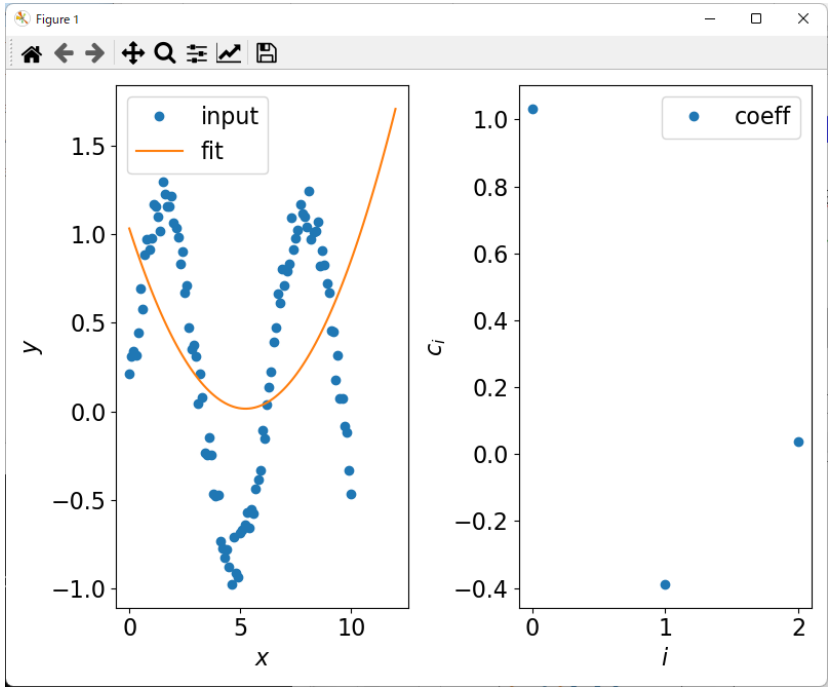


Program: lsq-polynomial.py

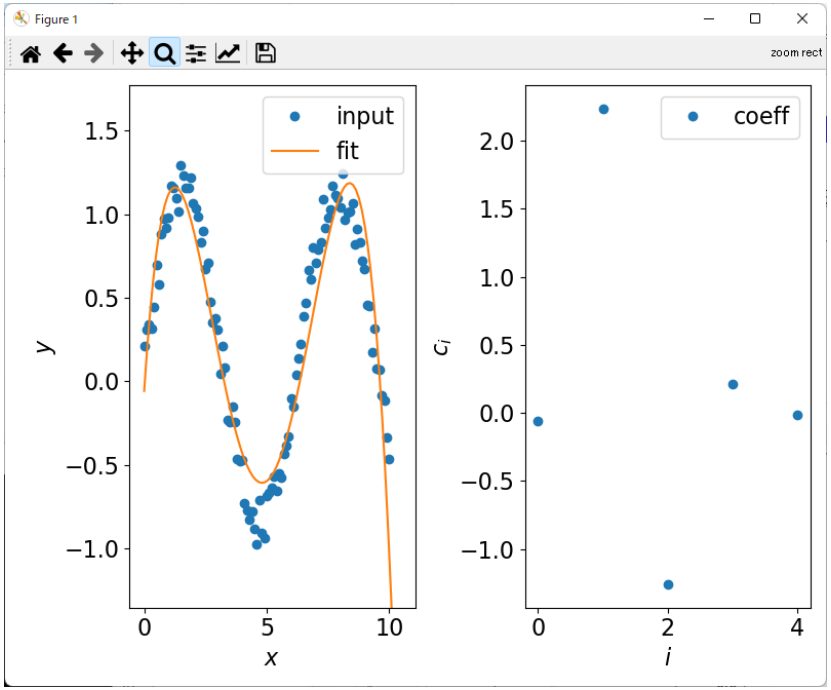
場所: [tkProg]¥tkprog_tutorial¥regression

Usage: python lsq-polynomial.py input_path norder

python lsq-polynomial.py random-sin.xlsx 2



python lsq-polynomial.py random-sin.xlsx 4



Program: numpy-polyfit.py

場所: [tkProg]¥tkprog_tutorial¥regression
Usage: python numpy-polyfit.py input_path norder

基本的に lsq-polynoimial.py と同じ。
numpy.polyfit() を使っているので、LSQ部分は以下のように簡単になる

```
# 入力データ x, y から norder次の多項式で回帰した係数を ci に受け取りる
    ci = np.polyfit(x, y, norder)

# numpy.poly1d に 係数リスト ci と、計算する x のリストを渡して、
# フィッティングした y の値のリストを受け取る
     ycal =  np.poly1d(ci)(x)
```

線形最小二乗法: 任意の関数

$$f(x) = a_1 f_1(x) + a_2 f_2(x) + \cdots + a_p f_p(x) = \sum_{k=0}^{p} a_k f_k(x)$$

$f_k(x)$がどんなに複雑な関数でも、あるいは数値テーブルだったとしても、

$f(x)$は a_i に関する線形関数

目的関数 $S = \sum_{j=1}^{n} (\sum_{k=1}^{p} a_k f_k(x_j) - y_j)^2$ **を最小化**

$$\frac{dS}{da_{k'}} = 2 \sum_{j=1}^{n} f_{k'}(x_j)(\sum_{k=1}^{p} a_k f_k(x_j) - y_j) = 0 \qquad k'=0, 1, \cdots, p$$

$$\sum_{j=1}^{n} (\sum_{k=1}^{p} a_k f_{k'}(x_j) f_k(x_j) - y_j f_{k'}(x_j)) = 0$$

※ $\sum_{k=1}^{p} a_k \sum_{j=1}^{n} f_{k'}(x_j) f_k(x_j) = \sum_{j=1}^{n} y_j f_{k'}(x_j)$

線形最小二乗法: 任意の関数

$$\sum_{k=1}^{p} a_k \sum_{j=0}^{n} f_{k'}(x_j) f_k(x_j) = \sum_{j=0}^{n} y_j f_k(x_j)$$

$$\begin{pmatrix} \sum f_1(x_i)f_1(x_i) & \sum f_1(x_i)f_2(x_i) & \sum f_1(x_i)f_3(x_i) & \cdots & \sum f_1(x_i)f_p(x_i) \\ \sum f_2(x_i)f_1(x_i) & \sum f_2(x_i)f_2(x_i) & \sum f_2(x_i)f_3(x_i) & & \sum f_2(x_i)f_p(x_i) \\ \sum f_3(x_i)f_1(x_i) & \sum f_3(x_i)f_2(x_i) & \sum f_3(x_i)f_3(x_i) & & \sum f_3(x_i)f_p(x_i) \\ \vdots & & & \ddots & \\ \sum f_p(x_i)f_1(x_i) & \sum f_p(x_i)f_2(x_i) & \sum f_N(x_i)f_3(x_i) & & \sum f_p(x_i)f_p(x_i) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum y_i f_1(x_i) \\ \sum y_i f_2(x_i) \\ \sum y_i f_3(x_i) \\ \vdots \\ \sum y_i f_p(x_i) \end{pmatrix}$$

$$X_{k,k'} = \sum_{j=1}^{n} f_{k'}(x_j) f_k(x_j) \qquad A_k = a_k$$

$$Y_k = \sum_{j=1}^{n} y_j f_k(x_j)$$

XA=Y　を解く

$A=X^{-1}Y$

$f_k(x)$は複数の変数$\{x^{(m)}\}$の関数 $f_k(\{x_j^{(m)}\})$ でもよい

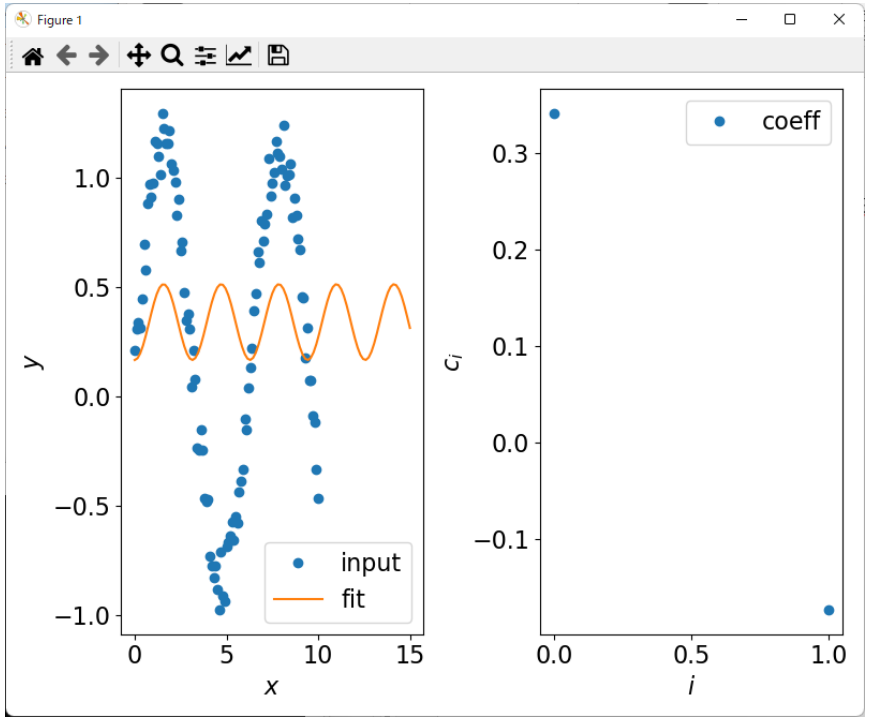
Program: lsq-general.py

場所: [tkProg]¥tkprog_tutorial¥regression

Usage: python lsq-general.py input_path nfunc

python lsq-general.py random-sin.xlsx 2

f(x) = 0.341 –0.173cos(2x)

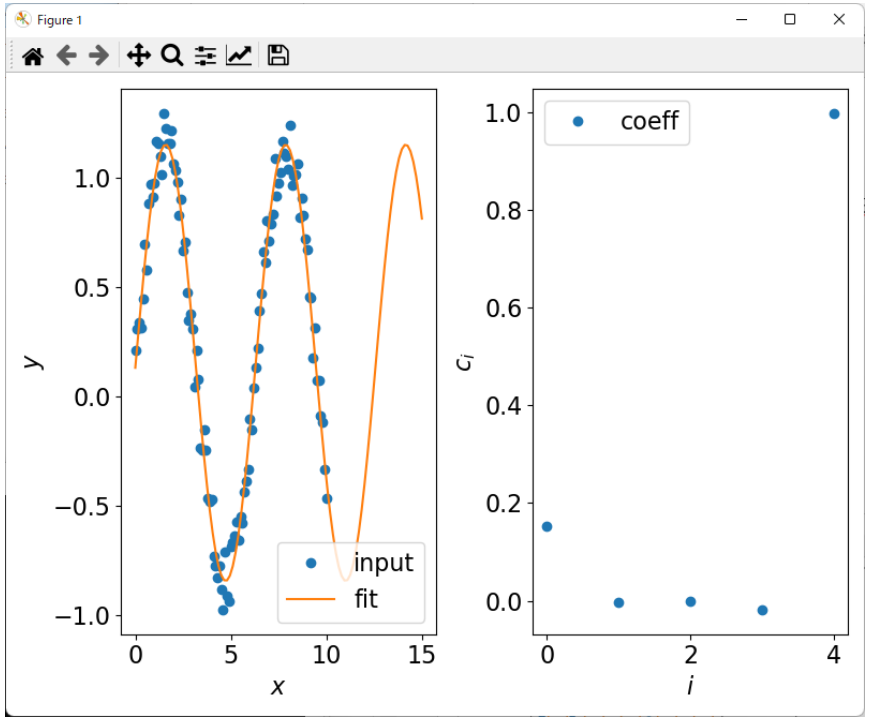


python lsq-general.py random-sin.xlsx 5

f(x) = 0.152 – 0.00252 cos(2x)

– 0.000347sin(2x) – 0.0177cos(x)

+ 0.997sin(x)



線形回帰 その他の話題

最尤推定法
正規化、正則化

最小二乗法の統計学的基盤: 最尤推定法

尤度関数とは:

事象 (x_k) が起こる確率を、既知のパラメータ (a_k)の確率密度関数

$$P(X = x_i|a_k) = \prod_i \left\{\frac{1}{\sqrt{2\pi\sigma_i}}\exp\left[-\frac{\varepsilon_i(x_i|a_k)^2}{2{\sigma_i}^2}\right]\right\} = \prod_i \left(\frac{1}{\sqrt{2\pi\sigma_i}}\right)\cdot\exp\left[-\sum_i\frac{\varepsilon_i(x_i|a_k)^2}{2{\sigma_i}^2}\right]$$

などとする。 $(\varepsilon_i(x_i|a_k)$ は 誤差。$(x_i|a_k)$ は、x_iが確率変数でa_kがパラメータであることを示す)

逆に $X = (x_i)$ がわかっているとし、パラメータ (a_k) がどれだけ尤もらしいか (尤度) を表す確率密度関数とみなし、上記の確率密度関数を変数 (a_k) の関数として

尤度関数 $P(a_i) = P(x_i|a_i)$ と呼ぶ (x_iがパラメータでa_kが確率変数)。

最尤推定法

誤差 $\varepsilon_i = f(x_i, a_i) - y_i$ が分散 σ_i の正規分布に従うとする。

データ (x_i, y_i) に対するパラメータ (a_i) の尤度関数は

$$\boldsymbol{P(a_i) = \prod_i \left(\frac{1}{\sqrt{2\pi\sigma_i}}\right)\cdot\exp\left[-\sum_i\frac{\varepsilon_i(x_i|a_k)^2}{2{\sigma_i}^2}\right]}$$

尤度を最大化するパラメータを求めるのが「最尤推定法」。

$$\boldsymbol{max\ P(a_i) = max\ \ln P(a_i) = min\sum_i \frac{{\varepsilon_i}^2}{{\sigma_i}^2}}$$: **最小二乗法に一致する**

正規化の必要性

多項式最小二乗法: $f(x) = \sum_{k=0}^{n} a_k x^k$

$$\begin{pmatrix} n & \sum x_i & \sum {x_i}^2 & \cdots & \sum {x_i}^p \\ \sum x_i & \sum {x_i}^2 & \sum {x_i}^3 & & \sum {x_i}^{p+1} \\ \sum {x_i}^2 & \sum {x_i}^3 & \sum {x_i}^4 & & \sum {x_i}^{p+2} \\ \vdots & & & \ddots & \\ \sum {x_i}^p & \sum {x_i}^{p+1} & \sum {x_i}^{p+2} & & \sum {x_i}^{2p} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum y_i x_i \\ \sum y_i {x_i}^2 \\ \vdots \\ \sum y_i {x_i}^p \end{pmatrix}$$

X行列の要素は 最小次数は 定数 n、最高次数は ${x_i}^{2p}$

・データが $|x_i| >> 1$ の場合、オーバーフロー、$|x_i| << 1$ ではアンダーフロー
丸め誤差など、計算誤差が大きくなる

=> 入力データを 1 のオーダーの数値に変換する必要がある

正規化の種類

よく使う最小二乗法: 正規化はしないことが多い

64bit CPUで誤差が問題になるケースは少ない

- 高次の関数が必要な物理モデルは少ない (せいぜい6次)
- それほど高次の多項式を使うと、過適合の問題がでる

機械学習: 正規化あるいは標準化はほぼ必須

比較しようのない記述子 (猫の体重、身長、年齢など) から、
単位依存性が無いように目的関数を作る必要がある

正規化 (normalization): $x_i' = \frac{x_i - x_{\min}}{x_{\max} - x_{\min}}$: データを 0 ~ 1の間に変換

ほかの範囲に変換してもいいが、計算誤差を減らすには
1のオーダーの範囲が望ましい

正規化: $x_i' = 2\frac{x_i - x_{\mathrm{mid}}}{x_{\max} - x_{\min}}$: データを –1 ~ 1の間に変換

標準化 (standardization): $x_i{}' = 2\frac{x_i - \langle x \rangle}{\sigma_x}$: 平均$\langle x \rangle$と標準偏差$\sigma_x$で変換

アルゴリズムによっては、
平均 0、標準偏差 1 を前提としている場合がある
標準化は必須