本講義 (第1回を含む) に関する質問
第4回以降のチュートリアルに関する希望
プログラムに関するバグ報告、要望

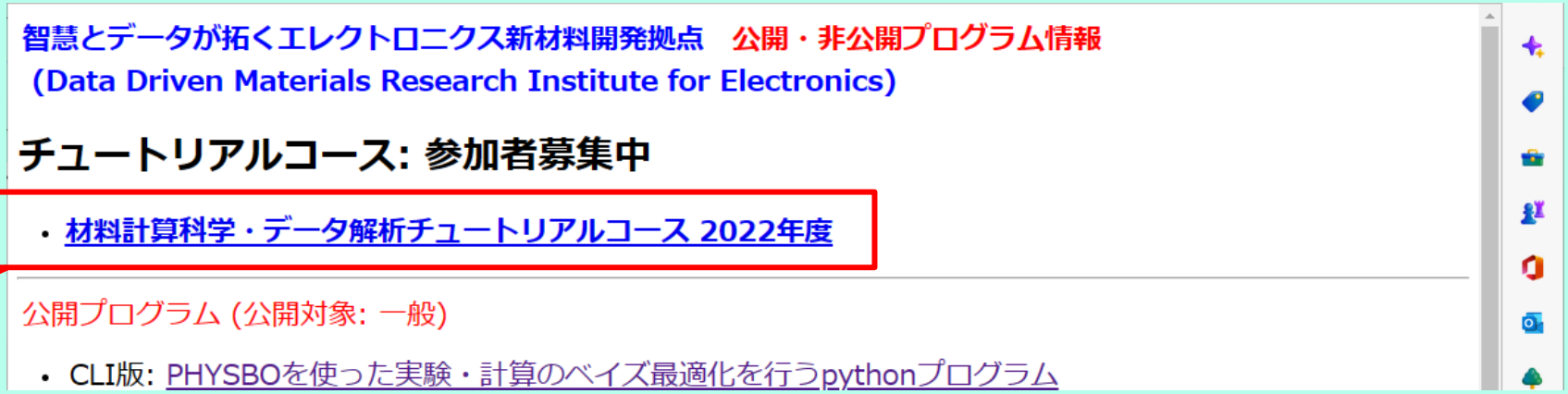
などは、本チュートリアル中にチャットでも受け付けます。
チャットを開き、「神谷利夫」あてに投稿してください。

チュートリアル以降は、メールで受け付けます
tkamiya@msl.titech.ac.jp

最新講義資料:
http://conf.msl.titech.ac.jp/D2MatE/tutorial2022.html

最新版ファイル

http://conf.msl.titech.ac.jp/D2MatE_programs.html



智慧とデータが拓くエレクトロニクス新材料開発拠点　公開・非公開プログラム情報
(Data Driven Materials Research Institute for Electronics)

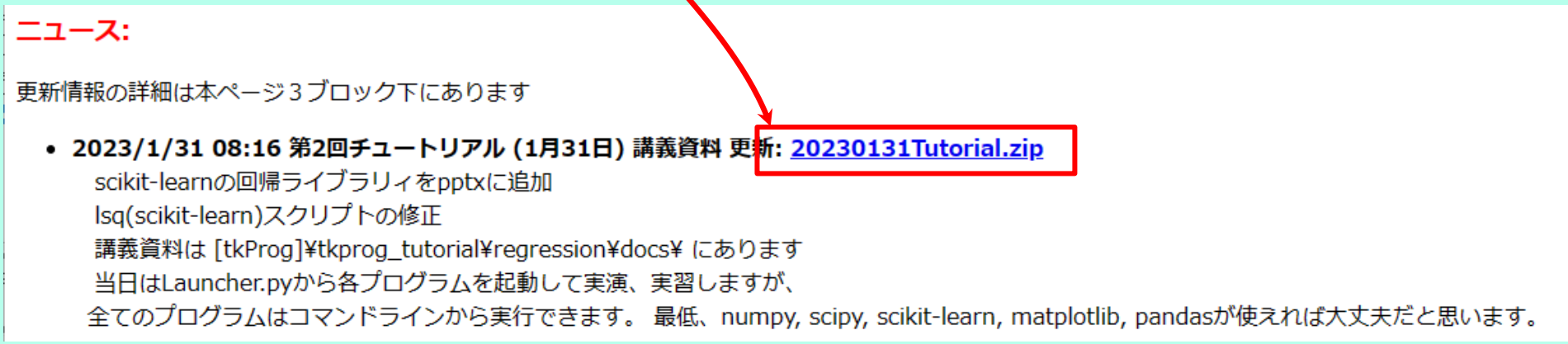
チュートリアルコース: 参加者募集中

- 材料計算科学・データ解析チュートリアルコース 2022年度

公開プログラム (公開対象: 一般)

- CLI版: PHYSBOを使った実験・計算のベイズ最適化を行うpythonプログラム

http://conf.msl.titech.ac.jp/D2MatE/tutorial2022.html
にて更新。(1/31 8:16に最終更新)



ニュース:

更新情報の詳細は本ページ３ブロック下にあります

- **2023/1/31 08:16 第2回チュートリアル (1月31日) 講義資料 更新: 20230131Tutorial.zip**
  scikit-learnの回帰ライブラリィをpptxに追加
  lsq(scikit-learn)スクリプトの修正
  講義資料は [tkProg]¥tkprog_tutorial¥regression¥docs¥ にあります
  当日はLauncher.pyから各プログラムを起動して実演、実習しますが、
  全てのプログラムはコマンドラインから実行できます。 最低、numpy, scipy, scikit-learn, matplotlib, pandasが使えれば大丈夫だと思います。

注意

本チュートリアルで配布している、神谷作成のプログラム群および
pythonライブラリ tklib:

- 再利用・再配布してかまいません。
- 動作の正確さ・安全性を保証するものではありません。使用者の判断と責任でお使いください。
- ネットワークに接続するなどの動作は含まれておりません。
- PC, 個人等の情報を収集するなど、セキュリティに問題を発生させる可能性がある動作は含まれておりません
- Pythonのみで書かれています。ソースコードを配布していますので、
必要があれば、使用者側で正確性・安全性の確認をしてください
- バグ修正、機能追加等の要望を受け付けます

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

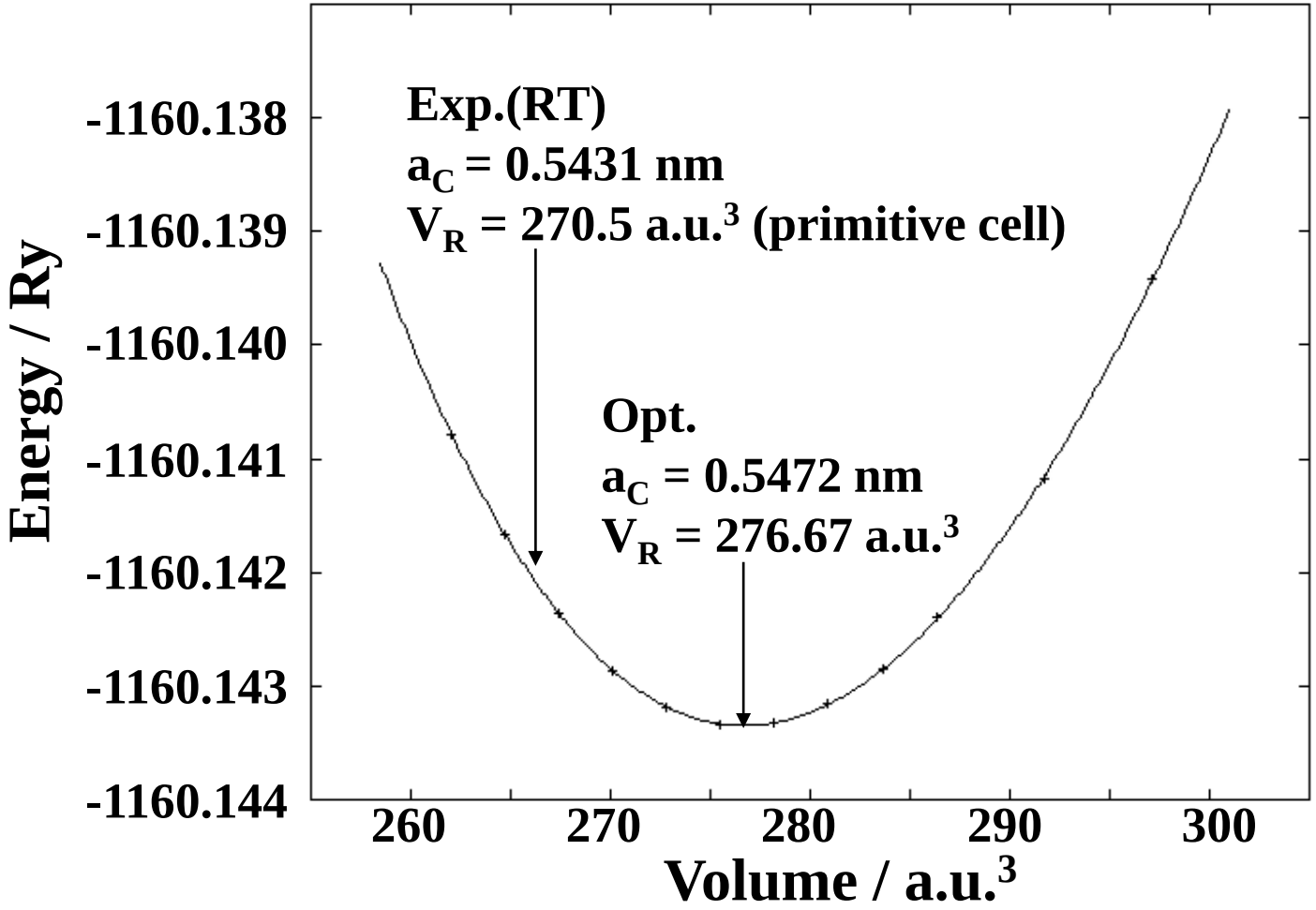
最大化問題は、$-f(x_i)$ を目的関数にとることにより、
最小化問題になる

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)

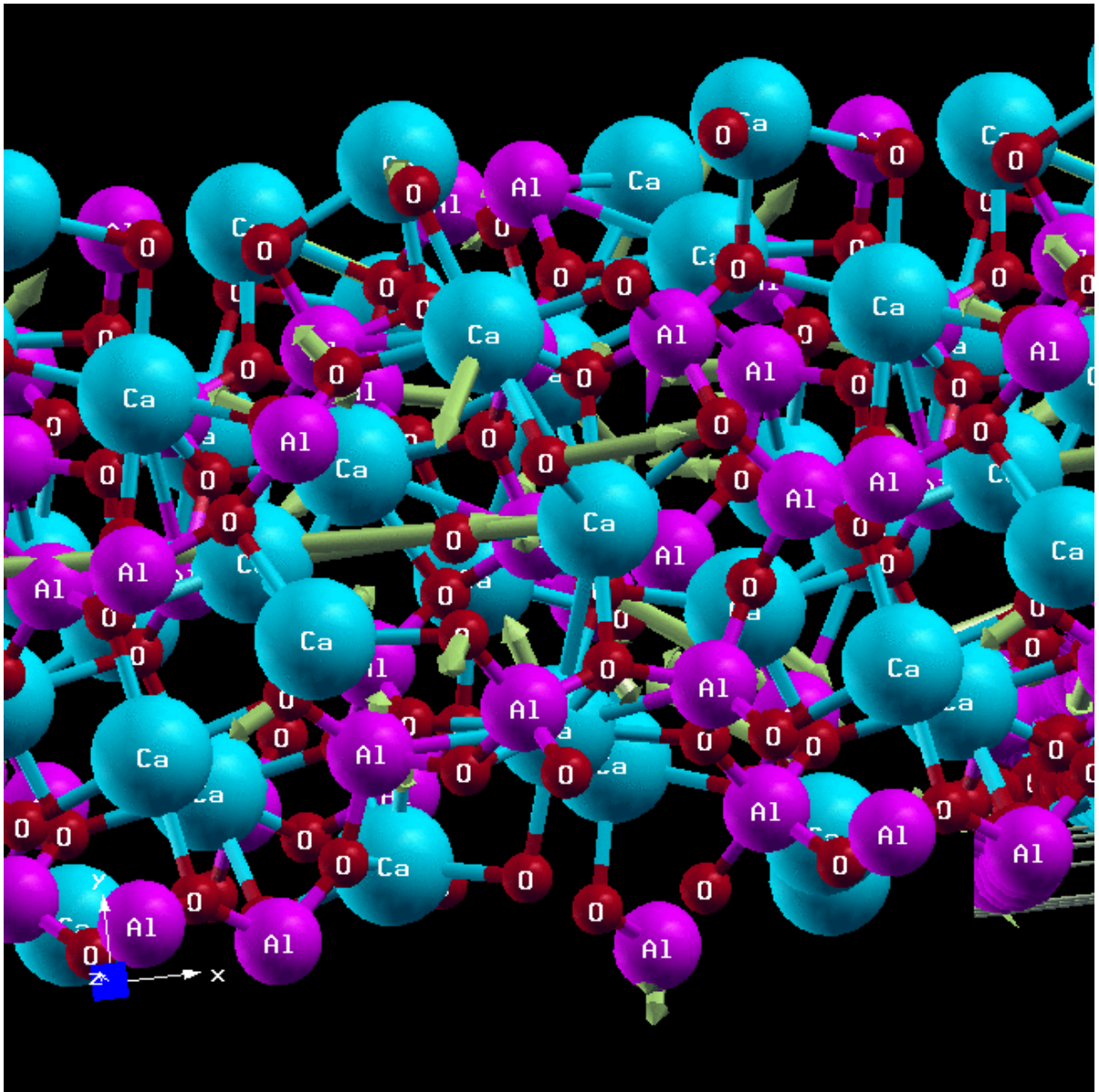
分子構造や結晶構造 (記述子) を与えて 全エネルギー (目的関数) を計算する: 量子計算
　構造を変化させ、全エネルギーが最小になる構造が最安定構造である

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

Lp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

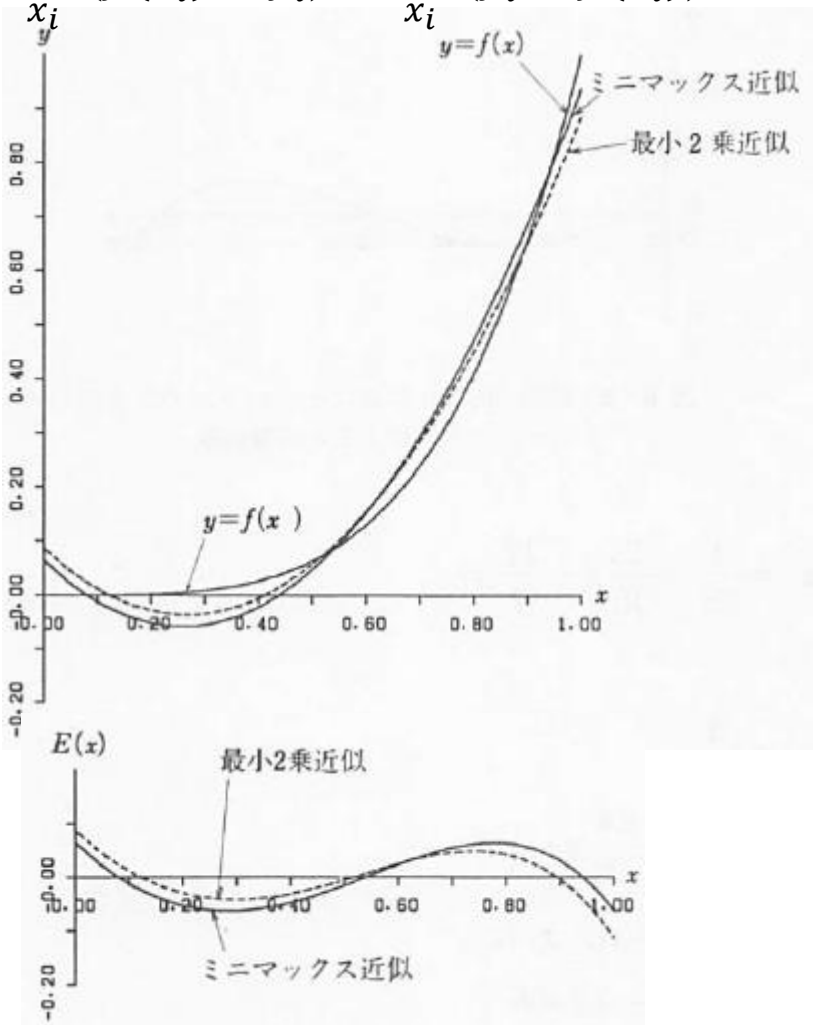
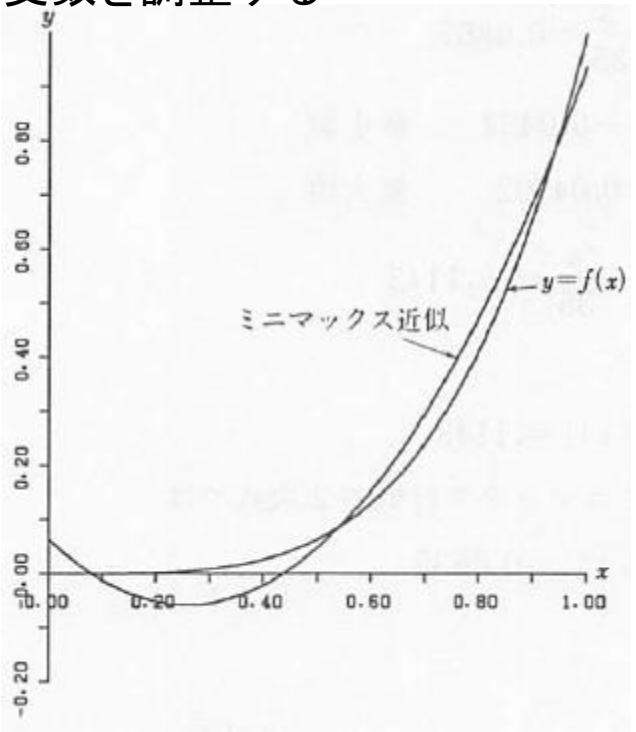


図 6・3　ミニマックス近似と最小 2 乗近似



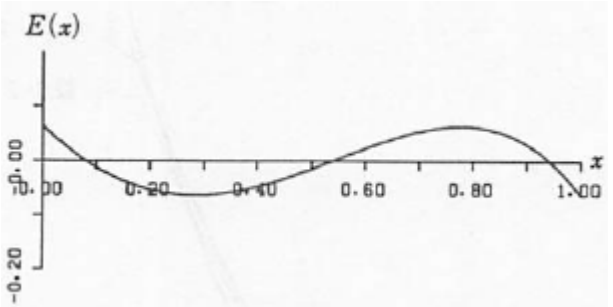


図 6・2　区間 [0, 1] における $f(x)=x^4$ の 2 次のミニマックス近似とその誤差曲線

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

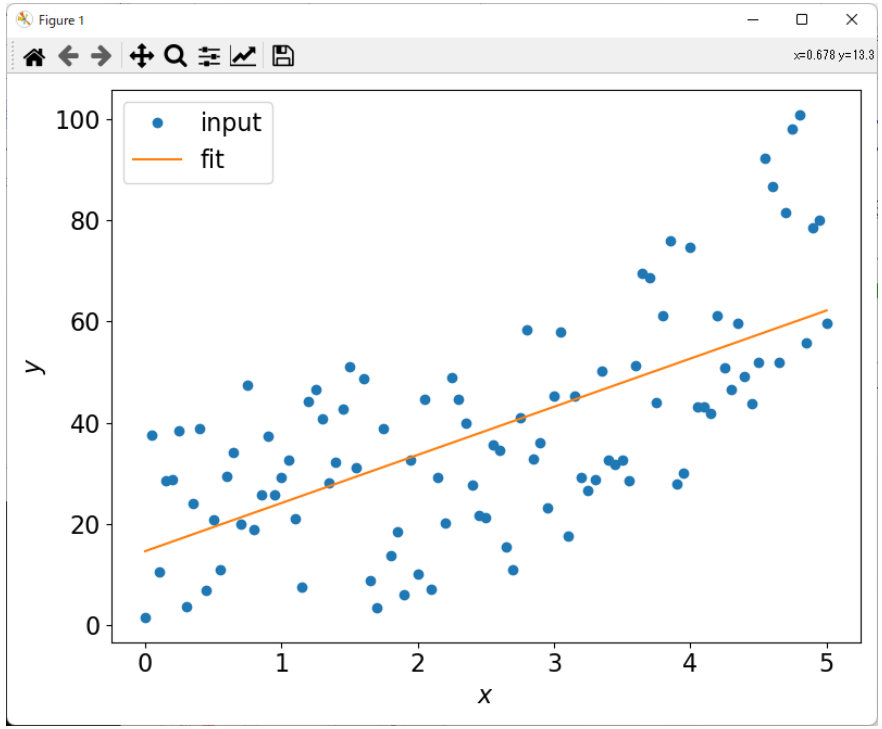
$$\begin{pmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} \sum y_i \\ \sum x_i y_i \end{pmatrix}$$

2x2の行列方程式を解けば、(a, b) が求まる

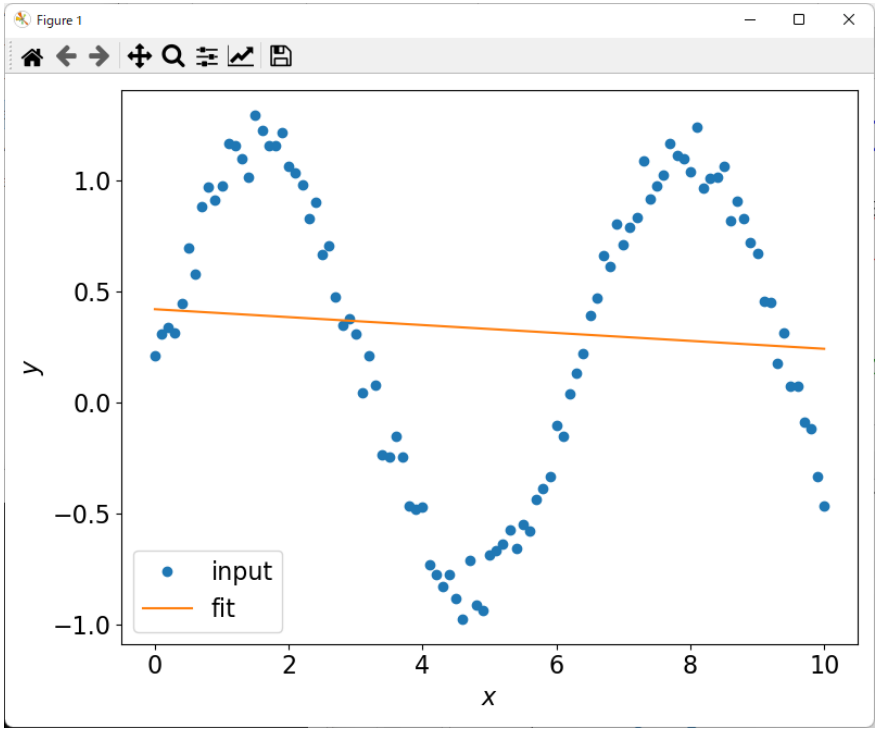
Program: lsq-line.py

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

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

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

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

$$XA=Y$$

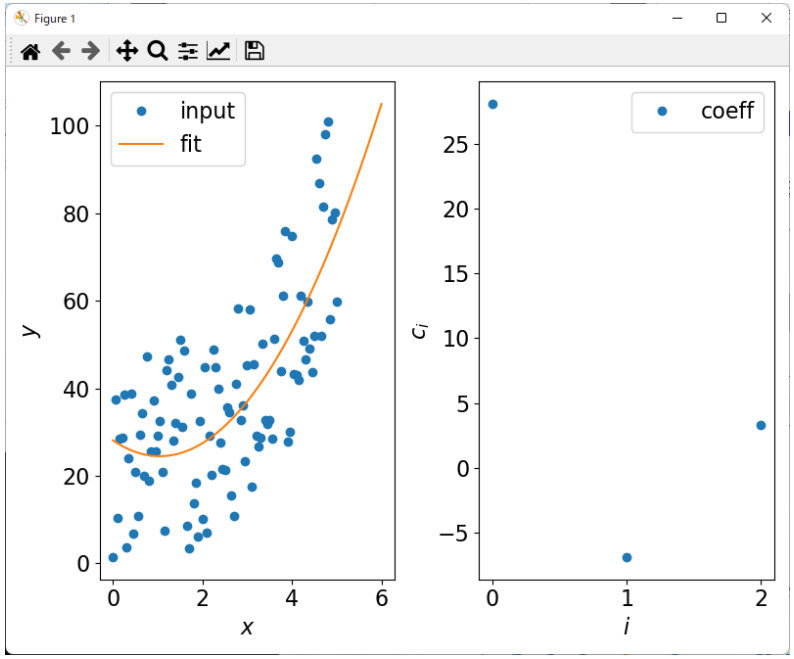
$$A=X^{-1}Y$$

Program: lsq-polynomial.py

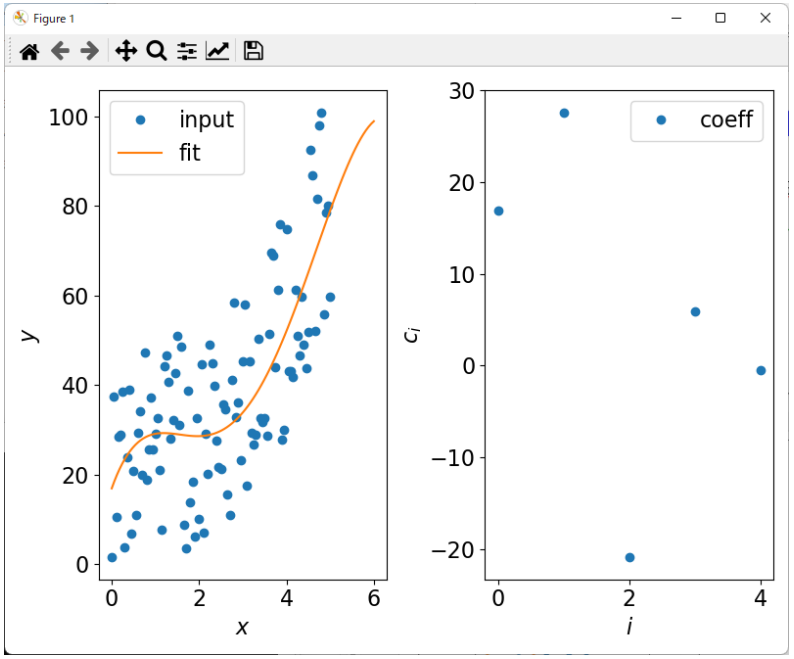
場所: [tkProg]¥tkprog_tutorial¥regression

Usage: python lsq-polynomial.py input_path norder

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



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

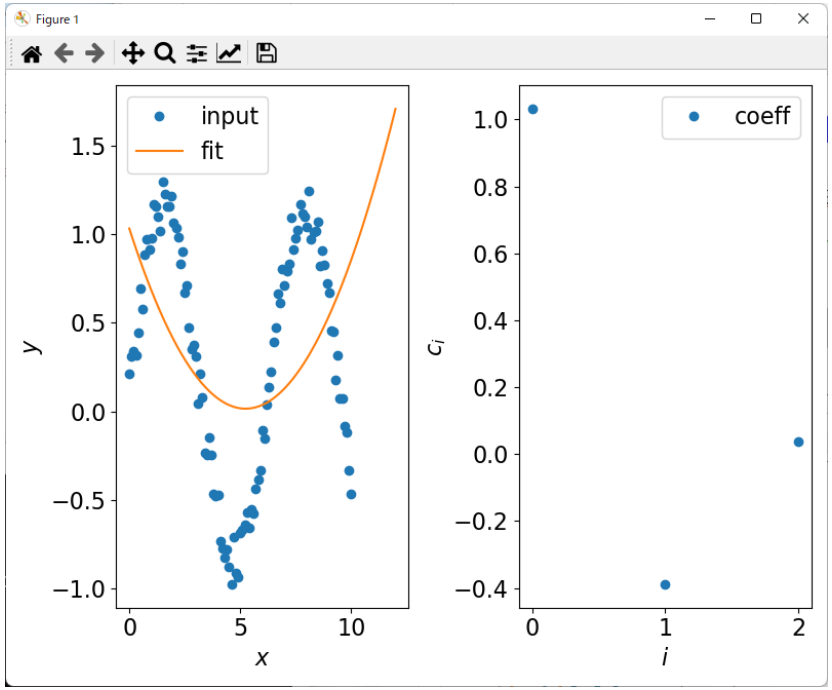


Program: lsq-polynomial.py

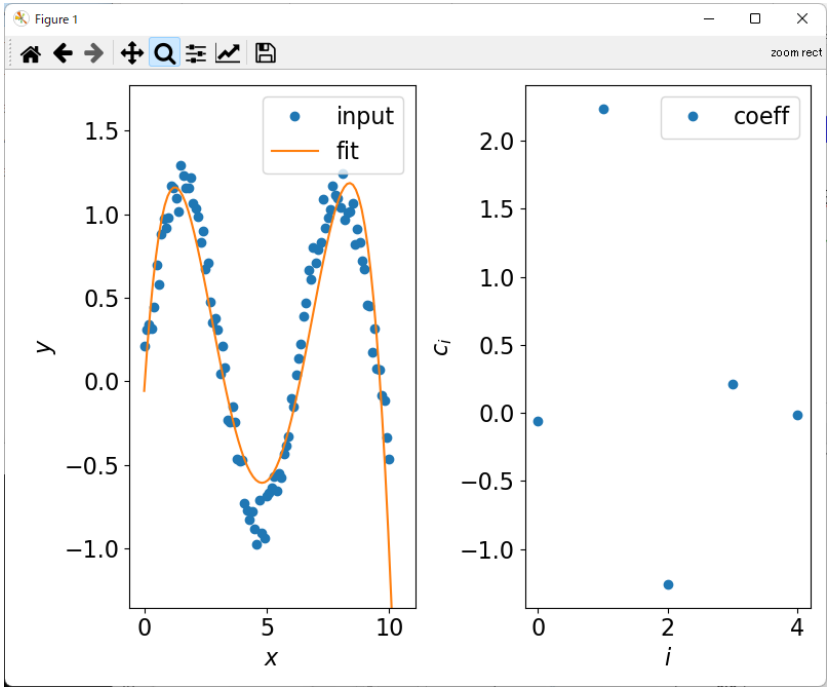
場所: [tkProg]¥tkprog_tutorial¥regression

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

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

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

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

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

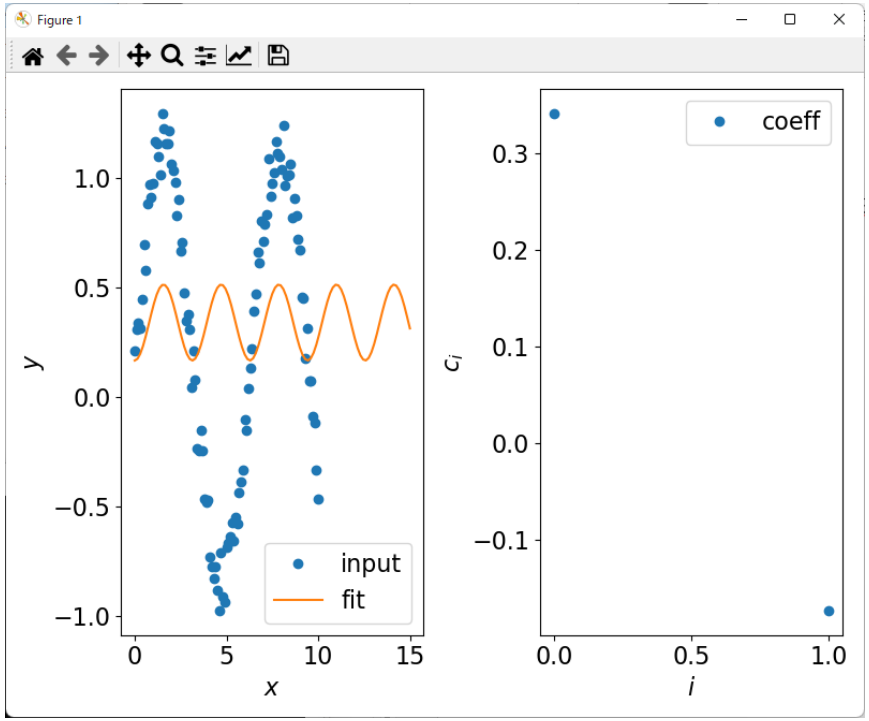
Program: lsq-general.py

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)

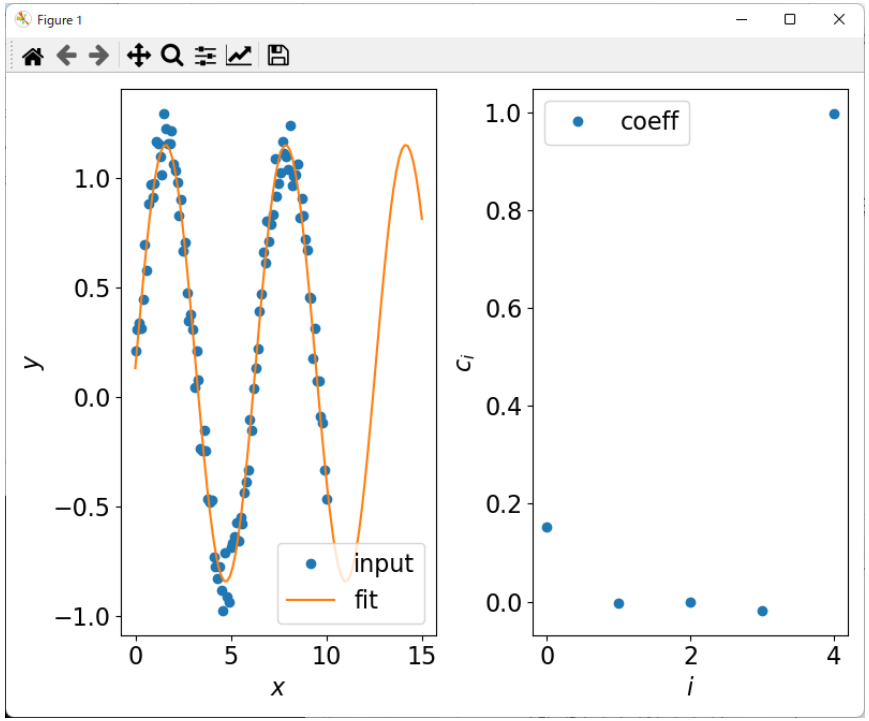


python lsq-general.py random-sin.xlsx 5

f(x) = 0.152 – 0.00252 cos(2x)

– 0.000347sin(2x) – 0.0177cos(x)

+ 0.997sin(x)



最小二乗法から
機械学習 (線形回帰) へ

モデル: $f(x) = a_1 x^{(1)} + a_2 x^{(2)} + \cdots + a_p x^{(p)} = \sum_{k=0}^{p} a_k x^{(k)}$

${x^{(k)}}_j$: j番目のデータのk番目の記述子

$$\boldsymbol{j = 1, 2, \cdots, n}$$

$$\boldsymbol{k = 1, 2, \cdots, p}$$

$$y_j = f(x_j) = a_1 {x_j}^{(1)} + a_2 {x_j}^{(2)} + \cdots + a_p {x_j}^{(p)} = \sum_{k=0}^{p} a_k {x_j}^{(p)}$$

重回帰 (多変量解析): 複数変数の線形回帰

多変数解析 (複数の変数(記述子))

記述子　$\{x^{(k)}\}=(x^{(1)}, x^{(2)}, \cdots, x^{(p)})$

データ　$\left(\left(\{x^{(k)}\}_1, y_1\right), \left(\{x^{(k)}\}_2, y_2\right), \cdots, \left(\{x^{(k)}\}_n, y_n\right)\right)$

$\{x^{(k)}\}_j$の成分を $x^{(k)}{}_j$ と書く

モデル: $f(x) = a_1 x^{(1)} + a_2 x^{(2)} + \cdots + a_p x^{(p)} = \sum_{k=0}^{p} a_k x^{(k)}$

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

$$\frac{dS}{\mathrm{d}a_{k\prime}} = \sum_{j=1}^{n} f_{k\prime}(x_j)(\sum_{k=1}^{p} a_k x^{(k)}{}_j - y_j) = 0 \qquad k'=0, 1, \cdots, p$$

$$\sum_{j=1}^{n} (\sum_{k=1}^{p} a_k x^{(k')}{}_j x^{(k)}{}_j - y_j x^{(k')}{}_j) = 0$$

※ $\sum_{k=1}^{p} a_k \sum_{j=1}^{n} x^{(k')}{}_j x^{(k)}{}_j = \sum_{j=1}^{n} y_j x^{(k)}{}_j$

重回帰 (多変量解析): 行列表示とベクトル表示

$$f(x) = a_1 x^{(1)} + a_2 x^{(2)} + \cdots + a_p x^{(p)} = \sum_{k=0}^{p} a_k x^{(k)}$$

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

解法: $\sum_{k=1}^{p} a_k \sum_{j=1}^{n} x^{(k')}{}_j x^{(k)}{}_j = \sum_{j=1}^{n} y_j x^{(k)}{}_j$

行列 (ベクトル) で表示: XA=Y $((\boldsymbol{x_{k'} \cdot x_k})(a_k) = (\boldsymbol{y \cdot x_k}))$ を解く

$$\boldsymbol{x_k} = \left(x^{(k)}{}_1, x^{(k)}{}_2, \cdots, x^{(k)}{}_n \right)$$

$$\boldsymbol{y} = (y_1, y_2, \cdots, y_n)$$

$$\mathrm{X} = (X_{k'k}) = \left(\sum_{j=1}^{n} x^{(k')}{}_j x^{(k)}{}_j \right) = \left(\boldsymbol{x^{(k')^T} x^{(k)}} \right)$$

$$\mathrm{A} = (A_k) = (a_k)$$

$$\mathrm{Y} = (Y_k) = \left(\sum_{j=1}^{n} y_j x^{(k)}{}_j \right) = \boldsymbol{y^T x^{(k)}}$$

機械学習の入力データ: 重回帰 (線形)

一般的な最小二乗法の入力データ random-poly.xlsx

x	y
0	4.810154
0.05	32.30261
0.1	18.78473
0.15	1.707199
0.2	15.75602

フィッティング関数 $f_i(x) = \{1, x, x^2, x^3, \cdots\}$ はプログラム中で計算する

機械学習の回帰の入力データ random-poly-ML.xlsx

x	y	x^1	x^2	x^3
0	4.810154	0	0	0
0.05	32.30261	0.05	0.0025	0.000125
0.1	18.78473	0.1	0.01	0.001
0.15	1.707199	0.15	0.0225	0.003375
0.2	15.75602	0.2	0.04	0.008

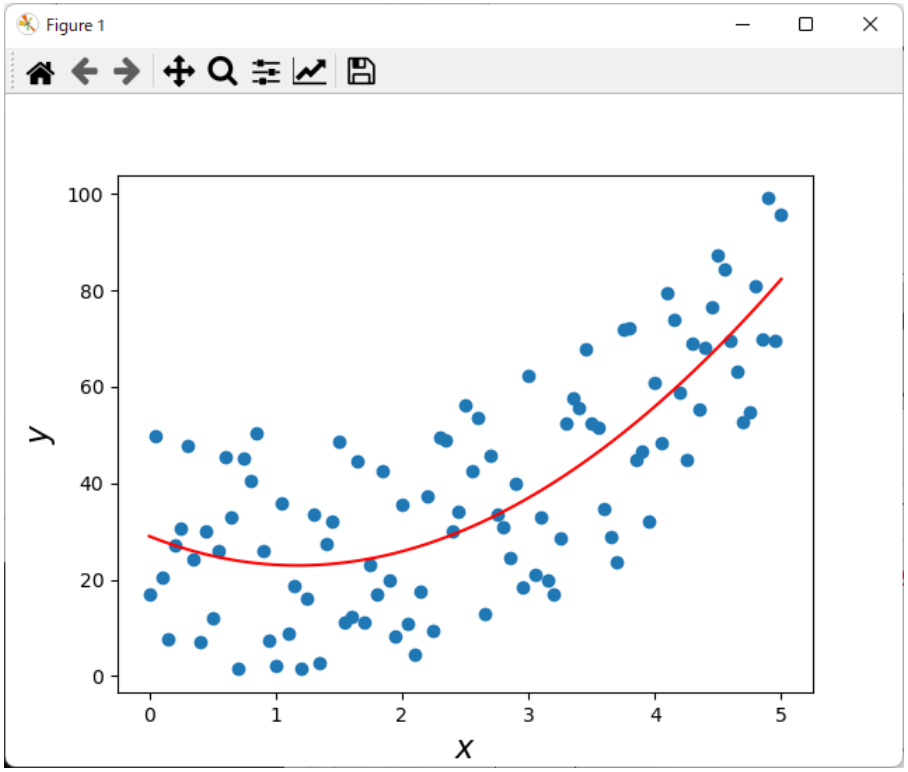
フィッティング関数 $f_i(x)$ の値は記述子として与える

LinearRegression() では定数部分は勝手に計算して intercept として返すので、
定数以外の項を記述子として与える
（x^1, x^2, x^3)

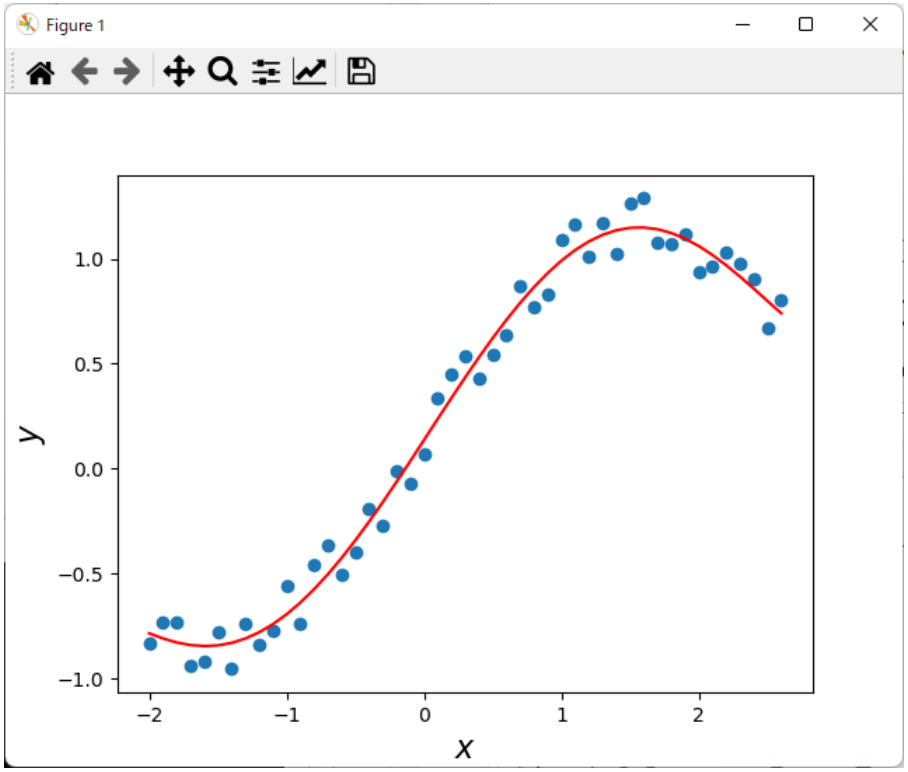
Program: lsq-polynomial-ML.py

Usage: python lsq-polynomial-ML.py input file

python lsq-polynomial-ML.py **random-poly-ML-xy.xlsx**
3次多項式で回帰



python lsq-polynomial-ML.py **random-sin-ML-xy.xlsx**
5次多項式で回帰



機械学習: カーネル法

モデル (**線形回帰**): $f(x) = a_1 x^{(1)} + a_2 x^{(2)} + \cdots + a_p x^{(p)} = \sum_{k=0}^{p} a_k x^{(k)}$

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

解法: XA=Y ($(\boldsymbol{x}_{k'} \cdot \boldsymbol{x}_k)(a_k) = (\boldsymbol{y} \cdot \boldsymbol{x}_k)$) を解く

機械学習 (回帰、分類、クラスター分析・・・、要するに「距離」を最小化するアルゴリズム)
には頻繁に内積 $\boldsymbol{x}_{k'} \cdot \boldsymbol{x}_k = \boldsymbol{X}_{k,k'} = \sum_{j=1}^{n} \boldsymbol{x}^{(k')}{}_j \boldsymbol{x}^{(k)}{}_j$ (**カーネル**) **が現れる**

=> $\boldsymbol{x}_{k'} \cdot \boldsymbol{x}_k$を適当な非線形な関数で置き換えれば、
線形回帰で非線形問題を解ける

カーネル法:

$\boldsymbol{x}_{k'} \cdot \boldsymbol{x}_k = \mathrm{X}_{k,k'}$ (記述子のベクトルの内積) を
他の関数で置き換える

$\boldsymbol{x}_{k'} \cdot \boldsymbol{x}_k$ => $k(\boldsymbol{x}_{k'}, \boldsymbol{x}_k)$: カーネル関数 ($p \times p$正方行列)

カーネル ＝ 波動関数の基底関数

カーネル法:

$x_{k'} \cdot x_k = \mathrm{X}_{k,k'}$ (記述子のベクトルの内積) を

“なんでもいいから適当な” 他の関数で置き換える

$x_{k'} \cdot x_k$ => $k(x_{k'}, x_k)$: カーネル関数 ($p \times p$正方行列)

思い出しましょう:

機械学習では、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

他の例:

量子化学では分子や結晶の波動関数を展開する「基底関数」は何でもいい

- 原子の波動関数
- 平面波
- 球面波
- **ガウス基底 (GTO)**
  - **=> データの分布に対してもガウス基底を使ってもいいでしょう？**
- スレーター基底 (STO)

線形最小二乗法 vs. カーネル法

任意関数の線形最小二乗法:

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

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

私たちが普段行う最小二乗法では、

$f_k(x)$には物理(科学)的意味がある

カーネル法:

$f(\boldsymbol{x}) = a_1 x^{(1)} + a_2 x^{(2)} + \cdots + a_p x^{(p)} = \sum_{k=0}^{p} a_k x^{(k)} = \boldsymbol{a} \cdot \boldsymbol{x}$

$\boldsymbol{x_{k'}} \cdot \boldsymbol{x_k} = (\mathrm{X}_{k',k})$を適当な関数 (カーネル関数) $k(\boldsymbol{x_{k'}}, \boldsymbol{x_k})$ で置き換える

$f(\boldsymbol{x}) = \sum_{k=0}^{p} a_k k(\boldsymbol{x}, \boldsymbol{x_k})$

カーネル関数に物理(科学)的な意味はなくてもいい。

必要な条件: 対称性 $k(\boldsymbol{x_{k'}}, \boldsymbol{x_k}) = k(\boldsymbol{x_k}, \boldsymbol{x_{k'}})$

正定値性 $\sum_{k,k'}^{p} a_{k'} a_k k(\boldsymbol{x_{k'}}, \boldsymbol{x_k}) \geq 0$

$(k(\boldsymbol{x_{k'}}, \boldsymbol{x_k}))(a_k) = (\boldsymbol{y} \cdot \boldsymbol{x_k})$ を解いて最小値が得られるための条件

カーネル関数の例

ガウシアンカーネル: $k(\boldsymbol{x_k}, \boldsymbol{x_{k'}}) = \exp\left(-\frac{|\boldsymbol{x_k}-\boldsymbol{x_{k'}}|^2}{2\sigma^2}\right)$

多項式カーネル　　: $k(\boldsymbol{x_k}, \boldsymbol{x_{k'}}) = (1 + \boldsymbol{x_k} \cdot \boldsymbol{x_{k'}})^p$

$\boldsymbol{x_{k'}}$を定数 (ハイパーパラメータ) $\boldsymbol{x_{c,i}}$と書き換えるとわかりやすい

例えば: $k(\boldsymbol{x_k}, \boldsymbol{x_{c,i}}) = \exp\left(-\frac{|\boldsymbol{x_k}-\boldsymbol{x_{c,i}}|^2}{2\sigma^2}\right)$

“記述子空間”のデータ点 $\boldsymbol{x_c}^{(k)} = (x_1{}^{(k)}, x_2{}^{(k)}, \cdots, x_p{}^{(k)})$

一つずつにガウス関数を置いた関数

カーネル回帰: 行列 $\mathrm{K} = (K_{k,k'}) = (k(\boldsymbol{x_k}, \boldsymbol{x_{k'}}))$ に対して

$\mathbf{y} = f(\boldsymbol{x_k}) = \sum_{k=1}^{p} a_k\, k(\boldsymbol{x_k}, \boldsymbol{x_{k'}}) = \mathrm{K}\boldsymbol{A}$

$S = (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y}) \cdot (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y}) = (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y})^T (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y})$を$A$の成分について最小化

$\Rightarrow \mathrm{K}^T(\mathrm{K}\boldsymbol{A} - \boldsymbol{Y}) = \boldsymbol{0}$ を解けばよい

Kは正定値行列なので $\boldsymbol{A} = K^{-1}\boldsymbol{Y}$ と、カーネルの逆行列だけで解ける

任意関数の線形最小二乗法 ～ カーネル回帰

任意関数 $f_k(x)$ の線形最小二乗法はカーネル法と(ほぼ)同じ

モデル: $y_j = \sum_{k=1}^{p} a_k f_k(x_j)$　　$\boldsymbol{Y} = \mathrm{F}\boldsymbol{A}$

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

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

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

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

$\mathrm{X}\boldsymbol{A}=\boldsymbol{Y}$ を解く　$\boldsymbol{A}=\mathrm{X}^{-1}\boldsymbol{Y}$

カーネル法: $f_k(x_j)$ をハイパーパラメータ X_k をいれて

カーネル関数 $k(x_j, X_k)$ で置き換える

$\boldsymbol{Y} = \mathrm{K}\boldsymbol{A}$　=> $\boldsymbol{A} = \mathrm{K}^{-1}\boldsymbol{Y}$ を解く

つまり、カーネル回帰　＝ $f_k(x)$ にある関係 (カーネルの対称性・正定値性) がある

任意関数の線形最小二乗法

カーネル法: 行列表示

$$\mathbf{y} = f(\boldsymbol{x_k}) = \sum_{k=1}^{p} a_k \, k(\boldsymbol{x_k}, \boldsymbol{x_{k'}}) = \mathrm{K}\boldsymbol{A}$$

$\boldsymbol{Y} = \mathrm{K}\boldsymbol{A}$を解く

$$(\boldsymbol{x_k}) = \left({x_1}^{(k)}, {x_2}^{(k)}, \cdots, {x_n}^{(k)}\right)$$

${x_j}^{(k)}$: j番目のデータのk番目の記述子

$$\begin{pmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & k(\boldsymbol{x_1}, \boldsymbol{x_2}) & k(\boldsymbol{x_1}, \boldsymbol{x_3}) & \cdots & k(\boldsymbol{x_1}, \boldsymbol{x_p}) \\ k(\boldsymbol{x_2}, \boldsymbol{x_1}) & k(\boldsymbol{x_2}, \boldsymbol{x_2}) & k(\boldsymbol{x_2}, \boldsymbol{x_3}) & & k(\boldsymbol{x_2}, \boldsymbol{x_p}) \\ k(\boldsymbol{x_3}, \boldsymbol{x_1}) & k(\boldsymbol{x_2}, \boldsymbol{x_2}) & k(\boldsymbol{x_3}, \boldsymbol{x_3}) & & k(\boldsymbol{x_3}, \boldsymbol{x_p}) \\ \vdots & & & \ddots & \\ k(\boldsymbol{x_p}, \boldsymbol{x_1}) & k(\boldsymbol{x_p}, \boldsymbol{x_2}) & k(\boldsymbol{x_p}, \boldsymbol{x_3}) & & k(\boldsymbol{x_p}, \boldsymbol{x_p}) \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_p \end{pmatrix}$$

注意: 回帰するデータ数 n は関数の数 p に等しくなければいけない

変数の数 ＝ 方程式の数

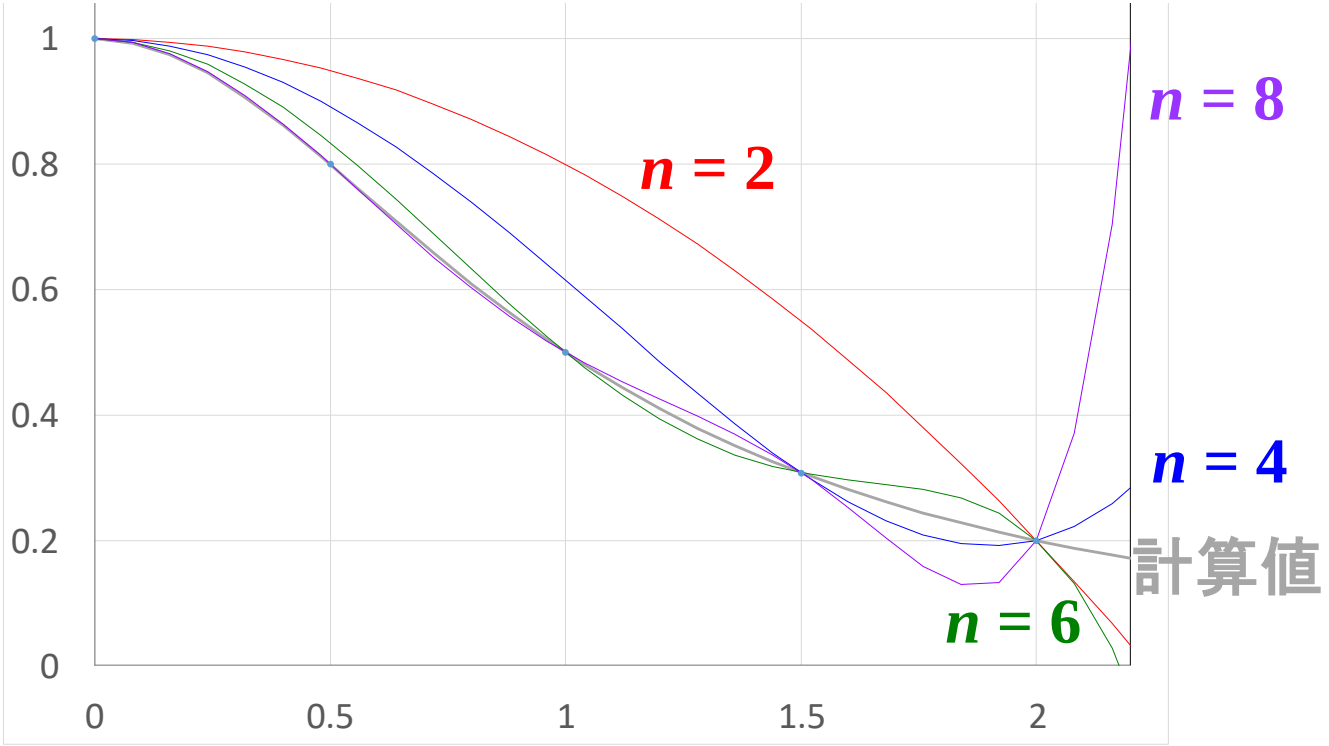
=> データ点の再現はできるが、それ以外の点の予測性が悪くなる

すべてのデータ点を通る多項式の問題

- **Rungeの現象**

  多項式の次数が大きくなると (特に三次以上)、
  標本点以外で大きく振動することがある

$f(x) = 1 / (1 + x^2)$, x = [-2, 2]中の $(n+1)$点を通るように補間



カーネル法: 1次元 $y = f(x)$ への回帰

データ点 x_j 一つづつにカーネル関数を置き、データ点自身を

を記述子とみなす

(“記述子”ではなく、”基底関数をx_j に置く” の方がわかりやすい)

$$y_j = f(x_j) = \sum_{k=1}^{p} a_k \, k(x_j, x_k) \qquad j = 1,2,\cdots,n = p$$

ガウシアンカーネルの例:

$$\left(\exp\left(-\frac{(x_j - x_{j'})^2}{2\sigma^2}\right)\right)\begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}$$

を解く。σ ($\sigma_{jj'}$) はハイパーパラメータ

過剰適合: Program: Ridge-Gauusian.py

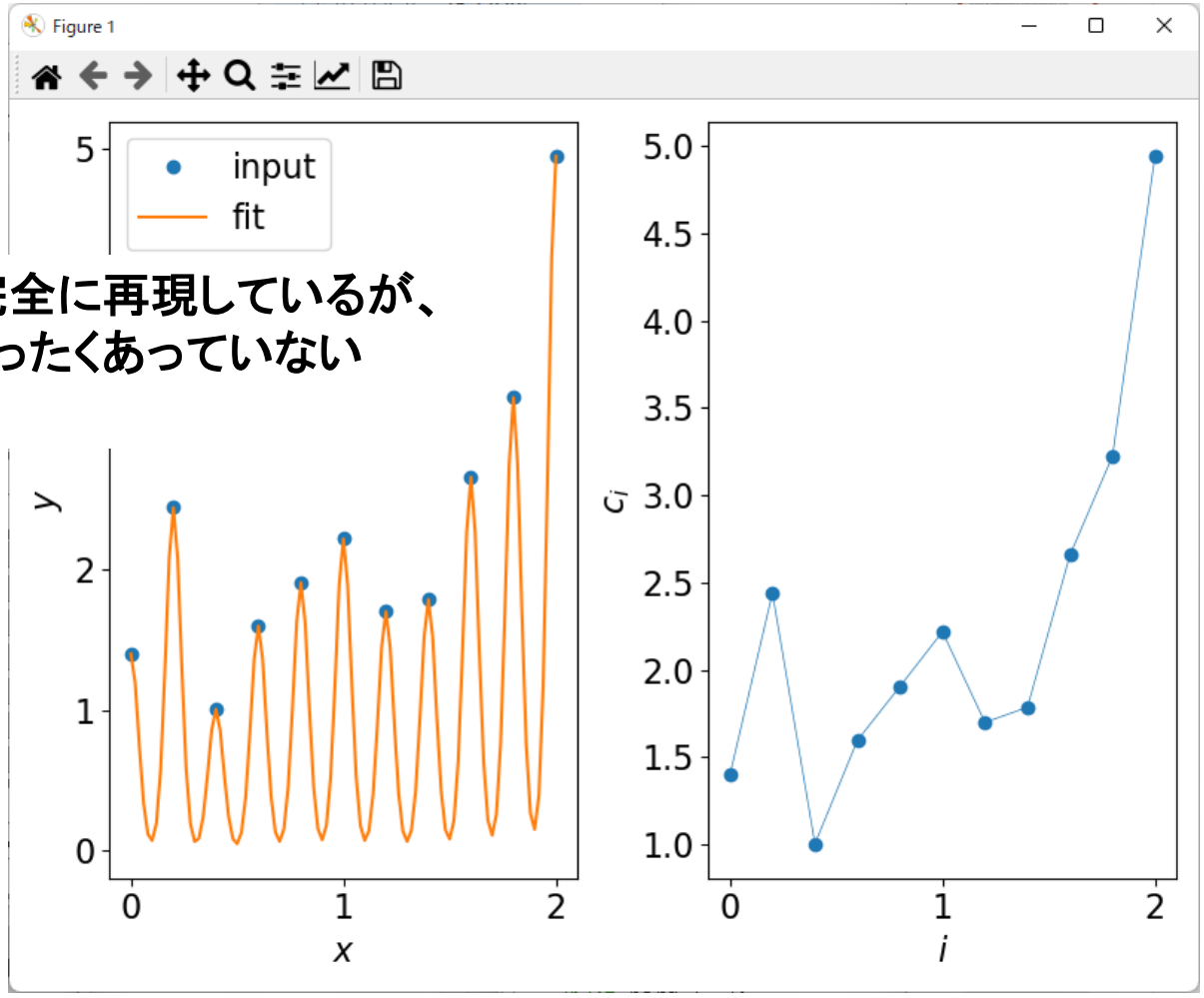
Usage: python KernelRidge-Gauusian.py infile nG wG lambda

ex: python KernelRidge-Gauusian.py random-poly-Gauss .xlsx 11 0.1 0.0

random-poly-Gauss .xlsx (データ点数11) を読み込み、11個の幅0.1のガウス関数でフィッティング。L2正規化項は入れない (Ridge解析をしない)

データ点は完全に再現しているが、他のxではまったくあっていない

過剰適合

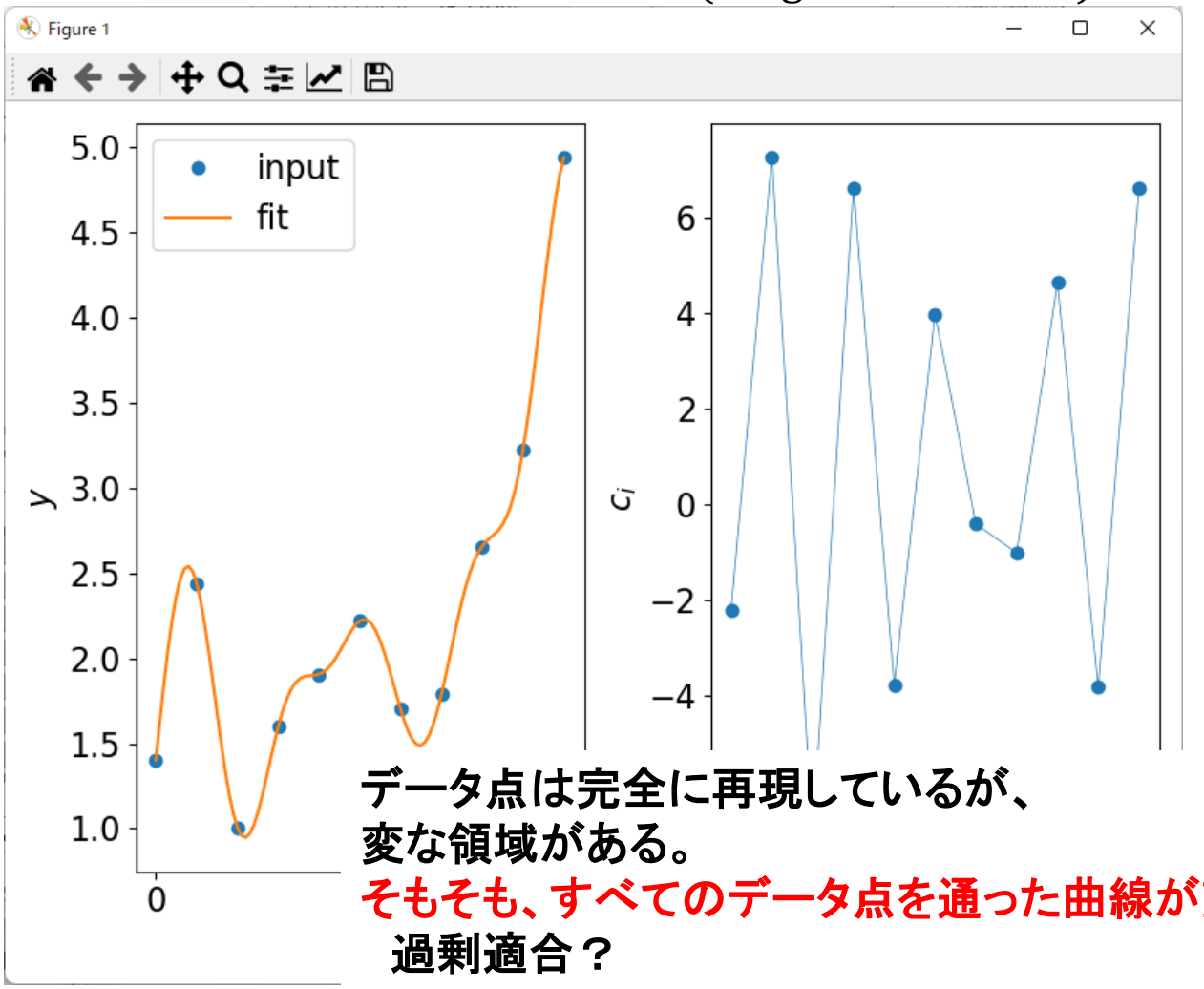


過剰適合: Program: Ridge-Gauusian.py

Usage: python KernelRidge-Gauusian.py infile nG wG lambda

ex: python KernelRidge-Gauusian.py random-poly-Gauss .xlsx 11 0.3 0.0

random-poly-Gauss .xlsx (データ点数11) を読み込み、11個の幅0.3のガウス関数でフィッティング。L2正規化項は入れない (Ridge解析をしない)



データ点は完全に再現しているが、
変な領域がある。
そもそも、すべてのデータ点を通った曲線が望ましい解か？
過剰適合？

Ridge回帰

係数 a_i に制約をつける。Ridge回帰では a_i の絶対値が大きくならないように
目的関数にL2ノルムの二乗のペナルティ (正則化項) を加える

$$f(x) = \sum_{k=0}^{p} a_k x^{(k)}$$

目的関数 $S = \sum_{j=1}^{n} (\sum_{k=1}^{p} a_k \, x^{(k)}{}_j - y_j)^2 + \boldsymbol{\lambda} \sum_{k=0}^{p} \boldsymbol{a_k}^2 \qquad \boldsymbol{\lambda \geq 0}$

$$\frac{dS}{da_{k'}} = 2 \sum_{j=1}^{n} x^{(k')}{}_j (\sum_{k=1}^{p} a_k \, x^{(k)}{}_j - y_j) + \mathbf{2}\boldsymbol{\lambda a_{k'}} = \mathbf{0}$$

$$\sum_{k=1}^{p} a_k \sum_{j=1}^{n} x^{(k')}{}_j \, x^{(k)}{}_j + \boldsymbol{\lambda a_{k'}} = \sum_{k=1}^{p} \sum_{j=1}^{n} x^{(k')}{}_j y_j$$

$$\begin{pmatrix} \sum x^{(1)}x^{(1)} + \boldsymbol{\lambda} & \sum x^{(1)}x^{(2)} & \sum x^{(1)}x^{(3)} & \cdots & \sum x^{(1)}x^{(p)} \\ \sum x^{(2)}x^{(1)} & \sum x^{(2)}x^{(2)} + \boldsymbol{\lambda} & \sum x^{(2)}x^{(3)} & & \sum x^{(2)}x^{(p)} \\ \sum x^{(3)}x^{(1)} & \sum x^{(3)}x^{(2)} & \sum x^{(3)}x^{(3)} + \boldsymbol{\lambda} & & \sum x^{(3)}x^{(p)} \\ \vdots & & & \ddots & \\ \sum x^{(p)}x^{(1)} & \sum x^{(p)}x^{(2)} & \sum x^{(p)}x^{(3)} & & \sum x^{(p)}x^{(p)} + \boldsymbol{\lambda} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} \sum x^{(1)}y_i \\ \sum x^{(2)}y_i \\ \sum x^{(3)}y_i \\ \vdots \\ \sum x^{(p)}y_i \end{pmatrix}$$

$$X_{k,k'} = \sum_{j=1}^{n} x^{(k)}{}_j x^{(k')}{}_j + \lambda \delta_{kk'} = \boldsymbol{x}^{(\boldsymbol{k})} \cdot \boldsymbol{x}^{(\boldsymbol{k'})} + \lambda I_p$$

XA=Y　を解く

リッジカーネル回帰

カーネル法の最小化問題: $\mathrm{K} = (k(\boldsymbol{x_k}, \boldsymbol{x_{k'}}))$ に対して

$S = (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y}) \cdot (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y}) = (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y})^T (\mathrm{K}\boldsymbol{A} - \boldsymbol{Y})$を最小化

$=> \boldsymbol{A} = K^{-1}\boldsymbol{Y}$を解けばよい

L^2正規化項を加えて過学習を抑える: Ridge Kernel Regression

$S = (\mathrm{K}\boldsymbol{A} - Y) \cdot (\mathrm{K}\boldsymbol{A} - Y) + p\lambda \boldsymbol{A} \cdot \mathrm{K}\boldsymbol{A}$ を最小化

(pは正規化係数が記述子の次元に依存しないように導入)

$\boldsymbol{A} = \left(K + p\lambda \boldsymbol{I_p}\right)^{-1} \boldsymbol{Y}$　　$\boldsymbol{I_p}$はp次元単位行列

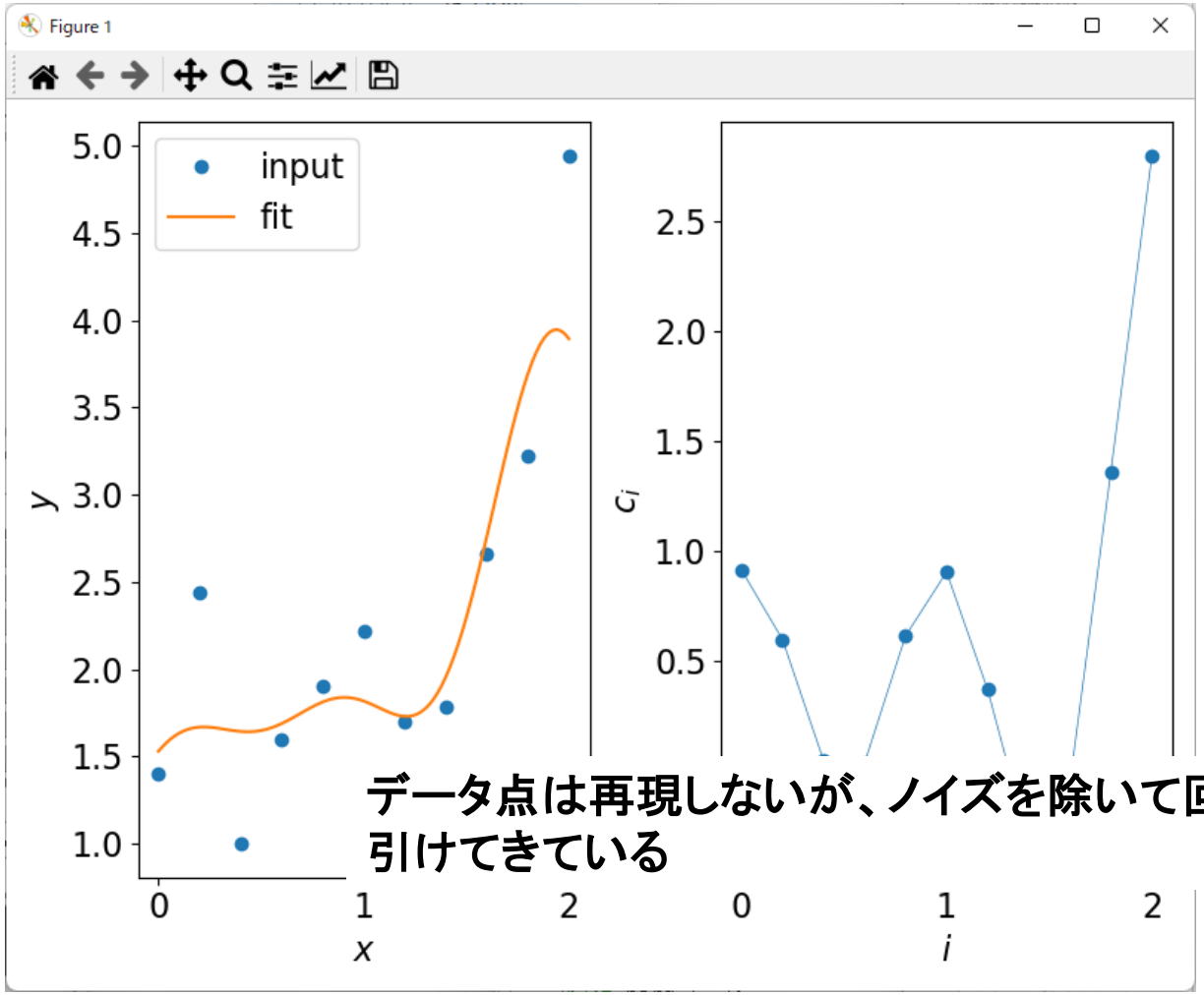
$f(\boldsymbol{x_k}) = \sum_{k'=1}^{p} a_{k'}\, k(\boldsymbol{x_k}, \boldsymbol{x_{k'}}) = \mathrm{K}\boldsymbol{A} = \mathrm{K}\left(K + p\lambda \boldsymbol{I_p}\right)^{-1} \boldsymbol{Y}$

Program: Ridge-Gauusian.py

Usage: python KernelRidge-Gauusian.py infile nG wG lambda

ex: python KernelRidge-Gauusian.py random-poly-Gauss .xlsx 11 0.5 0.2

random-poly-Gauss .xlsx (データ点数11) を読み込み、11個の幅0.5のガウス関数でフィッティング。L2正規化項の係数 0.2 を入れる (Ridge回帰)



データ点は再現しないが、ノイズを除いて回帰曲線を引けてきている

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

scikit-learn: 回帰の手順

lsq-polynomial-ML.py :

```
#Import
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression                    # 方法 (アルゴリズム) をimport
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score          # 評価関数をimport
#データ読み込み
  df = pd.read_excel(infile, engine = 'openpyxl‘)
  x = df[labels[2:]]
  y = df[labels[1]]
#記述子の標準化
   scaler = StandardScaler()
   scaler.fit(x)
   x_scaled = scaler.transform(x)
#フィッティング
   model = LinearRegression()                                        # 方法 (アルゴリズム) を選択
   model.fit(x_scaled, y)
#フィッティング結果から計算
   y_cal = model.predict(x_scaled)
#評価
   mae  = mean_absolute_error(y, y_cal)
   mse  = mean_squared_error(y, y_cal)
   rmse = sqrt(mse)
#パラメータ
   print(f“   intercept: {model.intercept_}”)                        #  定数項。決定木、NNなどには無い
   for iv in range(len(x_labels)):
      print(f“   {x_labels[iv]:>10}: {model.coef_[iv]:12.4g}”)   # 多項式回帰などの場合、記述子の係数。
                                                                    # 決定木、NNなどには無い
```

scikit-learn: 回帰

線形回帰系:

多重線形回帰

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Ridge回帰

```
from sklearn.linear_model import Ridge
model = Ridge(alpha = 0.05)
```

LASSO回帰

```
from sklearn.linear_model import Lasso
model = Lasso(alpha = 0.05)
```

Elastic Net回帰

```
from sklearn.linear_model import ElasticNet
model = ElasticNet(alpha = 0.05)
```

カーネル回帰系:

Kernal Ridge回帰

```
from sklearn.kernel_ridge import KernelRidge
model = KernelRidge(alpha = 1.0, kernel = 'rbf')
```

ニューラルネットワーク系:

多層パーセプトロン (多層ニューラルネットワーク)

```
from sklearn.neural_network import MLPClassifier
model = MLPClassifier(max_iter = 1000, hidden_layer_sizes = (10,), activation = 'logistic', solver = 'sgd',
                      learning_rate_init = 0.01)
```

scikit-learn: 回帰

分類器系:

Random Forest回帰

```
from sklearn.ensemble import RandomForestRegressor
model = RandomForestRegressor()
```

勾配ブースティング木 回帰

```
from sklearn.ensemble import GradientBoostingRegressor
model = GradientBoostingRegressor()
```

サポートベクターマシーン 回帰

```
from sklearn.svm import SVR
model = SVR(kernel='linear', C=1, epsilon=0.1, gamma='auto’)
```

回帰と機械学習

機械学習: 記述子、データが非常に多い

- 過剰適合の危険性が高い
- ブラックボックスでも、予測性能が高ければよい

過剰適合していないか、予測性能はあるかを検証 (Validation)

1. 既知データを 学習データ (training) と 検証データ (test) に分ける
2. 学習データでモデルを作る
3. 学習データと検証データの予測値 (prediction) と入力値から、評価を行う
4. 評価は MAE (Mean Absolute Error), MSE (Mean Squared Error), R^2 (決定係数) などで行う。

$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - f(\{x_i{}^{(k)}\})\right|$ 小さいほどいい

$MAE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - f(\{x_i{}^{(k)}\})\right)^2$ 小さいほどいい

$RMAE = \mathrm{sqrt}(MAE)$ 小さいほどいい

$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y_i - f(\{x_i{}^{(k)}\})\right)^2}{\sum_{i=1}^{n}(y_i - \langle y_i \rangle))^2}$ 1に近いほどいい